DESKTOP APPLICATION FOR THE PUZZLE BOARD GAME “RUSH HOUR”

Huanqing Nong

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd

Part of the Artificial Intelligence and Robotics Commons, Software Engineering Commons, and the Theory and Algorithms Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd/1278

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
DESKTOP APPLICATION FOR THE PUZZLE BOARD GAME “RUSH HOUR”

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Huanqing Nong
August 2021
DESKTOP APPLICATION FOR THE PUZZLE BOARD GAME “RUSH HOUR”

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Huanqing Nong
August 2021
Approved by:

Dr. Kerstin Voigt, Committee Chair
Dr. Tong Lai Yu, Committee Member
Dr. Owen J. Murphy, Committee Member
ABSTRACT

Rush Hour is a sliding block puzzle board game. This game comes with a board of 6 x 6 grid simulating a parking lot with an exit at the right end of the third row and some vehicle models of size 1 x 2 or 1 x 3 which can slide along the grooves of the grid forward or backward. The goal of the game is to clear the path by moving the vehicles on the board in a certain way for the target car, which lies on the third row of the grid, to merge out the “parking lot” through the exit.

It is somehow tedious that the board need to be set up manually every time for starting a new challenge or restart a challenge. In this project, a desktop version of Rush Hour puzzle game was developed so that players do not have to carry physical items to play and are able to start a challenge easily by just some clicks. Besides visualizing and simulating the game, this application comes with some additional functionality that the physical version does not have, such as scoring, hint provision, etc.
ACKNOWLEDGEMENTS

Throughout this project, I received a great deal of assistance and support.

I would like to express my deepest gratitude to Dr. Voigt, my project advisor. She inspired me with the idea of this project, guided me in the process of the project and provided assistance to me through the project generously.

I would like to show my gratitude to Dr. Tong Yu and Dr. Owen Murphy. I took some courses with them and learned much. They are kind and always willing to help.

I would like to thank Dr. George Georgiou for his presence to the presentation for this project on behalf of Dr. Murphy when Dr. Murphy was not available at that moment.

I am grateful for the suggestions that every professor made in the presentation.

Additionally, I would like to thank every professor, classmate and friend who ever helped me during my study at CSUSB.
# TABLE OF CONTENTS

ABSTRACT ......................................................................................................................... iii

ACKNOWLEDGEMENTS ................................................................................................. iv

LIST OF TABLES ............................................................................................................... viii

LIST OF FIGURES .............................................................................................................. ix

CHAPTER ONE: INTRODUCTION ...................................................................................... 1

  Background ....................................................................................................................... 1
  Overview ............................................................................................................................ 1

CHAPTER TWO: SOFTWARE REQUIREMENTS ANALYSIS .............................................. 3

  Software Requirement Specification .............................................................................. 3
    Basic Features ................................................................................................................ 3
    Additional Features ....................................................................................................... 3

CHAPTER THREE: SYSTEM DESIGN AND METHODOLOGY ........................................... 5

  Software Structure ......................................................................................................... 5
    Views ............................................................................................................................... 5
    Class Diagram ............................................................................................................... 6
    Use Case Diagram ......................................................................................................... 7
    Data Flow Diagram ...................................................................................................... 8
    Sequence Diagram ....................................................................................................... 9
    Methodology .................................................................................................................. 10

CHAPTER FOUR: IMPLEMENTATION .............................................................................. 12

  Data Storage ................................................................................................................... 12
  Challenges Data .............................................................................................................. 12
APPENDIX B: ATTRIBUTION ........................................................................... 123
REFERENCES ........................................................................................................ 125
LIST OF TABLES

Table 1. Comparison of Performance between BFS and A* on Challenges ....... 23
LIST OF FIGURES

Figure 1. Class Diagram ........................................................................................................... 7
Figure 2. Use Case Diagram ..................................................................................................... 8
Figure 3. Data Flow Diagram .................................................................................................. 9
Figure 4. Sequence Diagram .................................................................................................. 10
Figure 5. Main View .............................................................................................................. 13
Figure 6. Challenge View – Initial State of Challenge 333 .................................................... 14
Figure 7. Moving the Pink Car .............................................................................................. 15
Figure 8. Upon Mouse Left Button Up, the Pink Car Slide to the Nearest Spot
       Automatically .................................................................................................................. 16
Figure 9. Challenge View upon Waiting for Solution Search .............................................. 17
Figure 10. Solution Demonstration will Leave the Last Move to Player ......................... 18
Figure 11. View of Challenge Complete ................................................................................ 19
Figure 12. Score will be 82 If 40 Moves are Taken ............................................................. 20
Figure 13. Too Many Moves will Result in a Score of 0 ...................................................... 21
CHAPTER ONE

INTRODUCTION

Background

Rush Hour is a sliding block puzzle invented by Nob Yoshigahara in the 1970s and is currently manufactured by ThinkFun [1],[2]. The game comes with a board of 6 x 6 grid simulating a parking lot with an exit at the right end of the third row and some preset challenges. At the beginning of each challenge, vehicle models with size of 1 x 2 or 1 x 3 are placed on the board according to the corresponding challenge card. Vehicles can slide along the grooves of the grid forward or backward only, which means rotation is prohibited. The goal of the game is to help clear the path for the target vehicle to merge out the board through the exit, by moving the vehicles on the board.

Overview

The purpose of this project is to develop a desktop version of Rush Hour so that players do not have to carry physical items and set up the board manually to play. This application simulates the game on desktop with basic functionalities that the physical version has, but in a more convenient way. Players can easily initiate and reset challenges easily by just clicking and dragging the mouse. Additionally, some functionalities that the physical version does not have are also implemented, such as scoring and real-time hint provision.
This application is implemented in programming language Python with the help of the package *pygame*. More details about the application will be discussed in the following chapters.
CHAPTER TWO
SOFTWARE REQUIREMENTS ANALYSIS

Software Requirement Specification

Basic Features

**Graphical Representation.** This application needs to create a window and draw on the screen the components of the game including the board, the grid and vehicles.

**User-friendly UI.** All the UI components should be understandable and easy for users to interact with. Moving vehicles should be easily done by simple click-and-drag operations. Invalid movements such as moving a vehicle to an occupied spot should be avoided.

**Preset Challenges of Different Difficulty Levels.** This application should provide challenges of difficulty levels varying from beginner to grand master. Challenges are preset according to the physical challenge cards.

Additional Features

**Score system.** Upon completing a challenge, a player should receive a score based on their performance, evaluated by the number of moves taken, the fewer the better.

**Record system.** The minimal moves taken and the best score for each challenge that a user has ever achieved should be recorded and refreshed over time. The scoring and recording system aim to encourage players to pursue a sense of accomplishment so as to increase user stickiness.
**Hint Provision.** It is very common that a user has tried a great number of moves and finally gets stuck, especially in challenges of high difficulty levels. If the user wants to get help at such a moment, the hint system should find out a solution based on the current state of the board and demonstrate to the user.
CHAPTER THREE
SYSTEM DESIGN AND METHODOLOGY

This chapter describes how the software is designed and implemented.

Software Structure
This section describes how the application is structured and the relationship between different parts.

Views
This part will simply depict of the idea of views for different game states and the functionality that each view should have.

Main View. Main view is the view that shows up when the game application starts running. It should at least display the game’s name, and present the user with a choice among challenges with which to begin.

Challenge View. Once the user has confirmed a challenge to start, the challenge view will be displayed with all the vehicles of that challenge has been set up. This view should have the following features:

1. It should display the board, the grid, the vehicles in place, challenge alias, current number of moves taken, best ever performance of the user.

2. The user should be able to click and drag the cars to move them.

3. The recorded number of moves taken should keep track of the actual number of moves that the user has made.

4. The user should be able to restart the challenge.
5. The user should be able to return to the challenge select view (or the main view).

6. The user should be able to get help from “Hint”.

**Challenge Complete View.** Once the user completes the challenge, the “challenge complete view” should display. This view should have the following features.

1. The final score should be computed and displayed.

2. If the record of the challenge is broken, new record should be stored and the record should be refreshed.

3. Player should be able to return to the challenge select view (or the main view).

4. Player should be able to restart the challenge.

5. Hint provision should be disabled.

**Global Widgets.** Player should be able to disable and enable the background music at any time except while waiting for the hint provision system to search for a solution.

**Class Diagram**

The class diagram below shows the overall structure of this application.
Use Case Diagram

This diagram describes how a user will interact with the software.
Figure 2. Use Case Diagram

Data Flow Diagram

This diagram simply depicts how the data are flowing.
Figure 3. Data Flow Diagram

Sequence Diagram

The sequence diagram shows the sequence of the basic actions for a player to play a challenge.
**Methodology**

**Programming Language.** This application is implemented in the Python programming language. The version of Python used is 3.9.5.

**Software.** PyCharm is used as IDE for this project.

**Dependencies.** Besides the built-in modules, the following packages are required.
1. pip
2. setuptools
3. pygame (version 2.0.1)
4. pygame-functions (version 0.0.1)
CHAPTER FOUR
IMPLEMENTATION

This chapter explains the details of the implemented application.

Data Storage

Data stored include the data of preset and pre-calculated challenges and the user data.

Challenges Data

Challenge Configuration. Challenges are preset according to the physical challenge cards that are included in packaged board game by ThinkFun. The data are stored in a text file in a particular format that the main module can read and load the challenge settings into memory when the application starts. The file stores information of each challenge including challenge alias, number of vehicles, alias of each car, initial position and orientation of each car.

Minimal Moves. The Minimal moves for each challenge to be solved are pre-calculated by breadth-first search (BFS) and stored in a text file in order for final score computation upon challenge completion.

User Data

User data, including the best performance for each challenge of each player, are stored in a json file.
Graphic User Interfaces

The graphic user interfaces (GUI) are demonstrated in this section with brief description.

Main View

![Figure 5. Main View](image)

When the user runs the application, a window will be created with the “main view” displayed on the window. On this view, the user will be asked to choose a challenge to begin with. All the preset challenges will be listed, each followed by its difficulty level which is presented in the form of star icons. At the center of the bottom, “Selected challenge” will be updated every time the user
clicks on a challenge, telling the current selected challenge. Once a challenge is selected, the “GO!” button will be displayed and enabled.

**Challenge View**

![Challenge View – Initial State of Challenge 333](image)

After picking a challenge and clicking “Go!”, the initial state of that challenge will be loaded on the screen. Player now can click on a vehicle and drag it to move. The implementation ensures that no vehicle can be dragged across another vehicle in its path. Upon release of the mouse left button, the dragged car will be released as well, and it will slide to the nearest spot.
automatically, as shown in Figure 7 and Figure 8. If the position of this vehicle is changed after this action, the number of moves taken will be incremented by 1.

Figure 7. Moving the Pink Car
Figure 8. Upon Mouse Left Button Up, the Pink Car Slide to the Nearest Spot Automatically

During challenge, clicking on “Return” button will make the game return to the challenge select view. Clicking on “Restart” button will reset the current challenge, including the vehicles and the number of moves taken. Clicking on “Quit” will terminate the whole game process. Clicking on “Hint” button will pause the game for a certain amount of time while the program is searching for the best solution based on the current state of the board.
While waiting for the program to find the best solution for the current state, interactions are prohibited. After a solution is found, the notice of red font shown in Figure 9 will disappear, and solution demonstration based on the current state will start right away. Solution demonstration will automatically move the vehicles according to the solution found. During the auto-moving, clicking on “Hint” button will terminate the auto-moving process at the end of the current single move demonstration. If not terminated half-way, solution demonstration will demonstrate all the moves in the solution except the last move, which is dragging the target vehicle out of the exit. This last move is left for the player to execute.
Figure 10. Solution Demonstration will Leave the Last Move to Player
Challenge Complete

Upon the target car of current challenge being dragged out of the board, the challenge is said to be completed. When the player completes a challenge, the score will be calculated and the record for this challenge will be updated and stored. The range of the score is $[0, 100]$. The fewer moves made to complete the challenge, the higher score will the player get. It is related to the ratio of the excess pass to the minimum number of moves. The formula for the score is as below.

$$\text{Score} = \max (\lfloor 100 \times (1 - (n - m) / m) \rfloor, 0)$$
Where \( n \) = number of moves that the player has made to complete the challenge, and \( m \) = the minimal moves to complete the challenge.

In Figure 11, for example, the player applied 34 moves to complete challenge 333, and the minimal number of moves to complete challenge 333 is 34. In this case, \( n = 34 \), \( m = 34 \). Then

\[
\text{Score} = \max \left( \lfloor 100 \times (1 - (34 - 34) / 34) \rfloor, 0 \right) = 100.
\]

If the number of moves taken is 40 instead, the score will be

\[
\text{Score} = \max \left( \lfloor 100 \times (1 - (40 - 34) / 34) \rfloor, 0 \right) = 82,
\]

as shown in Figure 12.

Figure 12. Score will be 82 If 40 Moves are Taken
If the number of moves taken is greater than or equal to the double of minimal number of moves for this challenge, the score will be 0.

Figure 13. Too Many Moves will Result in a Score of 0

Global widgets

There is a music button at the bottom left corner of the window. It is always displayed despite the state of the game. User can switch on or off the background music of the game by clicking on this button.
Search Algorithm

There are two scenarios in this program where search algorithms are involved, pre-computing the minimal moves for each challenge and real-time searching for solution. BFS was chosen for minimal moves computation, which is simply because BFS guarantees the first path found is the shortest. In terms of solution search, breadth-first and A*[^4] were both examined.

The heuristic function constructed counts the number of vertical vehicles on the right-hand side of the target vehicle blocking the way out[^5]. The advantage of this heuristic is that it is admissible. Here this heuristic does not overestimate the actual number of moves needed, since each of the blocking vehicles has to be moved away, sooner or later, or the target vehicle will not be able to merge out. Therefore, this heuristic guarantees the first path found is the shortest as well. Nevertheless, in the meanwhile, this heuristic may not help in most states. In most cases, to move away those blocking vehicles is not straightforward but requires certain movement operations on not only the blocking vehicles. It is possible that a move that increases the number of blocking vehicles is a necessary move to solve the puzzle. In this case, this heuristic does not reduce the number of nodes to be visited. However, it does help reduce the nodes visited on the last levels of the search tree, since the last move before moving the target vehicle out of the board has to be moving a vertical vehicle out of the way out. The more continuous movements of blocking vehicle to be out of
the third row at the end of the solution, the greater number of nodes visited are reduced.

In the table below, the performance between these two algorithms are compared.

Table 1. Comparison of Performance between BFS and A* on Challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th># Vehicles</th>
<th>Min # Moves</th>
<th># Nodes Visited</th>
<th>Execution Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Cars</td>
<td>Trucks</td>
<td>BFS</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>11</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
<td>9</td>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>302</td>
<td>13</td>
<td>10</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>303</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>304</td>
<td>13</td>
<td>8</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>321</td>
<td>14</td>
<td>10</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>325</td>
<td>13</td>
<td>10</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>332</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>34</td>
</tr>
<tr>
<td>333</td>
<td>13</td>
<td>9</td>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>340</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>43</td>
</tr>
</tbody>
</table>

As we can see in the table, A* search does expand less nodes than BFS does in most cases. Even so, it cannot be said that A* perform better than BFS in terms of execution time. A* consumed more execution time than BFS did in 11 out of 13 cases, ranging from 0.04 seconds to 4.18 seconds, while in the rest 2 cases, A* consumed less execution time than BFS did, ranging from 8.70 seconds to 9.75 seconds. It is explicable since A* needs to deal with the open queue (implemented as a heap queue in this program), such as to traversing for finding duplicates and heapifying the queue. In addition, it needs to calculate the
f score (f = g + h) for each node, which also increase the execution time marginally. Execution time is a significant consideration in this game since it determines the duration of idle time of the GUI when searching for solution in the background and thus negatively influences the player’s experience. Based on the data shown in the table, BFS is chosen to be used for real-time solution search since it takes less time in more cases than A*. Anyway, it is possible to find a better heuristic to improve the efficiency of A* search or to apply different search algorithms for different challenges in the future enhancement of this application.

In terms of a better heuristic, based on theoretical considerations, it means a more informative and admissible evaluation function. For example, on top of the number of blocking vehicles, the number of vehicles that are blocking the blockings car from moving out of the third row of the board can be taken into consideration. However, a more informative heuristic function also means lower efficiency in the computation of the evaluation of each node. It is possible to find a better balance between the information that the evaluation function carries and its computational cost.
CHAPTER FIVE
SOFTWARE TESTING

Testing is important to verify that the software works properly as expected. Unit test was done in every phase of development of this application. Every time a new feature was implemented or an existing part was modified, there followed a unit test and a full smoke test. Furthermore, an overall test was done covering all the functionality after the application was finished. Defects exposed in each test were fixed in time.

Unit tests were done by code review and running the program with print statements for the key variables added in the functions. Smoke testing was done to assure that the application still worked properly and would not crash after the modification was made.

The overall test covers all of UI components and the interaction with them on all the views of the game, as well as the interactions and data transfers between different views to assure that the game does not behave in unexpected way. All the requirements are satisfied, and the software is deliverable.
CHAPTER SIX
FUTURE ENHANCEMENT

There are several aspects of this application can be enhanced in the future.

GUI

In this project, the GUI of the application leaves room for improvement with regard to artistic rendering and response times when hints are requested.

Search Algorithm

As can be seen in this project, real-time computation takes time, which can negatively affect the user experience. There are at least two ways to solve this issue. One is to do research for a better algorithm or heuristic function to boost the speed of real-time solution searching. The other one is to store all the possible state of each challenge and the solution based on that state, like a huge directed graph. Upon hint provision, the only thing the program needs to do is to find the state in the graph and read its solution. However, the efficiency in searching for the matching state is significant. If the number of states to store is large \(^6\), then it needs to be made sure that the cost of finding matching states does not become too high. It makes no sense having stored states, when the number of states to search becomes too costly. Linear search (e.g., storing states in a vector or list) may not give good performance due to O(N) linear
search. However, if the search for matches could be reduced to \(O(\log N)\) (e.g., by storing states in a suitably set up binary search tree), or even close to \(O(1)\) with some clever hash table scheme, the idea of using pre-stored states can become a worthwhile option. The empirical exploration of both options in the future, computing with more informative heuristic and finding matching stored states in suitably efficient data structures, would help to identify the enhancements with the least amount of idle time for the player of the Rush Hour puzzle game.

**Customized Challenges**

A system with a GUI where users can easily create and test a new challenges may further enrich the game.

**Cloud Data Storage**

Cloud data storage will make it easier for extensions. User data are stored more safely. Additionally, players can compete over internet, which makes the game of more fun.
A desktop application simulating the sliding block puzzle board game Rush Hour was implemented in this project. Basic features, such as graphical representation, user-friendly UI, preset challenges, as well as additional features, such as score system, record system and hint provision, were implemented. The fans of this game can easily play the game on desktop without carrying it and set up challenges manually every time. The score and record system encourage players to pursue a sense of accomplishment, which brings to the game more of user stickiness. Breadth-first search was implemented and chosen as the algorithm for real-time solution searching. A* algorithm with an admissible heuristic function, taking into account the number of vehicles blocking the way out of the board, was tried. Even though the total number of nodes visited was reduced, but the performance, in terms of time consumed for searching, was increased in most cases. Taking more and deeper conditions into account may result in a better heuristic function and may take less time for real-time search than breadth-first search algorithm. Besides, pre-stored all the possible states of each challenge and the solution to that state is worth exploring as well.
APPENDIX A

APPLICATION CODE
This application contains images and sounds which are required to run the program. The entire project has been uploaded to a GitLab inventory: https://gitlab.com/005814662/rush-hour with a readme file. The functional source code of this project is included in this appendix, while the rest of the source code and resources can be found in the GitLab inventory.

README.md

# Rush Hour

A simulator of the board game Rush Hour

## To install the dependencies with pip

Run the following command from the root directory of this project

`pip install -r requirements.txt`

## To run the game, make sure to use Python3 interpreter

Run command

`python rushHour.py`
To add more challenges

1. Add the configuration of the new challenge to file `challenge/challenges_raw.txt`
2. Run the following command from the root directory

`python challenge/calc_min_steps.py`

__init__.py

(empty file)

common.py

```python
import pygame
import os
directory = os.path.split(os.path.realpath(__file__))[0]
import json

class Common:

    @staticmethod
def translate_index_to_coordinate(j, i):
        return int(80 * (j - 1)), int(80 * (i - 1))

    @staticmethod
```
def translate_x_to_j(x):  # i ranges from 1 to 6
    return int(x / 80 + 1)

@staticmethod
def translate_y_to_i(y):  # j ranges from 1 to 6
    return int(y / 80 + 1)

@staticmethod
def translate_coordinate_to_index(x, y):
    return int(80 * (x - 1)), int(80 * (y - 1))

@staticmethod
def translate_j_to_x(j):
    return int(80 * (j - 1))

@staticmethod
def translate_i_to_y(i):
    return int(80 * (i - 1))

@staticmethod
def load_challenges():
    challenges = dict()
lines = ""

min_steps = dict()

with open(f"{directory}/challenge/challenges_raw.txt", "rt") as f:
    lines = f.readlines()

with open(f"{directory}/challenge/min_steps.json", "rt") as f1:
    min_steps = json.loads(f1.read())

for i in range(len(lines)):
    if lines[i][0] == "#":
        temp = lines[i].split()
        challenge_alias = temp[1]
        challenges[challenge_alias] = {}
        difficulty = lines[i + 1].strip()
        temp = lines[i + 2].split()
        n_cars = int(temp[0])
        # n_moves_sol = int(temp[1])
        cars = {}
        for j in range(1, n_cars + 1):
            attr = lines[i + 2 + j].split()
            car_alias = attr[0]
            cars[car_alias] = {}
            cars[car_alias]["length"] = int(attr[1])
            cars[car_alias]["]"] = int(attr[2])
cars[car_alias][“i”] = int(attr[3])

cars[car_alias][“direction”] = attr[4]

challenges[challenge_alias][“difficulty”] = difficulty

challenges[challenge_alias][“config”] = cars

challenges[challenge_alias][“n_moves_sol”] =

min_steps[challenge_alias]

    # # sort the challenges by int(alias)

    challenges = {k: v for k, v in sorted(challenges.items(), key=lambda item: int(item[0])))

    return challenges

class RHButton(object):

    name = "unnamed"

    mouse_down = False

    is_active = False

    def __init__(self, x, y, w, h, ic, ac, action=None, param=None):

        self.x = x

        self.y = y

        self.w = w

        self.h = h
self.rect = pygame.Rect((self.x, self.y, self.w, self.h))

self.ic = ic
self.ac = ac
self.action = action
self.param = param

def trigger(self, param=None):
    if self.action is not None:
        if param is not None:
            self.action(param)
        elif self.param is not None:
            self.action(self.param)
        else:
            self.action()
    else:
        raise ValueError("undefined function being called")

def get_rect(self):
    return self.rect

def on_mouse_down(self):
    self.mouse_down = True
def on_mouse_hover(self):
    self.is_active = True

def on_mouse_hover_out(self):
    self.is_active = False

def on_mouse_up(self, param=None):
    if self.mouse_down:
        self.trigger(param)
        # print("RHButton.on_mouse_up")
    self.mouse_down = False

def blit(self, surface):
    mouse_x, mouse_y = pygame.mouse.get_pos()
    if self.get_rect().collidepoint((mouse_x, mouse_y)):
        self.on_mouse_hover()
    else:
        self.on_mouse_hover_out()
    if self.is_active:
        back = pygame.draw.rect(surface, self.ac, self.get_rect())
    else:
back = pygame.draw.rect(surface, self.ic, self.get_rect())

def set_name(self, name):
    self.name = name

@classmethod
def text_objects(cls, text, font):
    text_surface = font.render(text, True, (0, 0, 0))
    return text_surface, text_surface.get_rect()

class TextButton(RHButton):
    def __init__(self, x, y, w, h, ic, ac, msg='', action=None, param=None):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.rect = pygame.Rect((self.x, self.y, self.w, self.h))
        self.ic = ic
        self.ac = ac
        self.msg = msg
        self.action = action
self.param = param

def blit(self, surface):
    mouse_x, mouse_y = pygame.mouse.get_pos()
    if self.get_rect().collidepoint((mouse_x, mouse_y)):
        self.on_mouse_hover()
    else:
        self.on_mouse_hover_out()
    if self.is_active:
        back = pygame.draw.rect(surface, self.ac, self.get_rect())
    else:
        back = pygame.draw.rect(surface, self.ic, self.get_rect())
    small_text = pygame.font.SysFont("comicsansms", 20)
    text_surf, text_rect = self.text_objects(self.msg, small_text)
    text_rect.center = ((self.x + (self.w / 2)), (self.y + (self.h / 2)))
    surface.blit(text_surf, text_rect)

class IconButton(RHButton):
    def __init__(self, x, y, image1=None, image2=None, action1to2=None,
    action2to1=None, param=None):
        self.x = x
self.y = y
self.img1 = image1
self.img2 = image2
self.img_index = 1
self.action1to2 = action1to2
self.action2to1 = action2to1
self.param = param

def get_curr_img(self):
    img_key = f"img{self.img_index}"
    return self.__dict__[img_key]

def blit(self, surface):
    # mouse_x, mouse_y = pygame.mouse.get_pos()
    # if self.get_rect().collidepoint((mouse_x, mouse_y)):
    #     self.on_mouse_hover()
    # else:
    #     self.on_mouse_hover_out()
    surface.blit(self.get_curr_img(), (self.x, self.y))

def get_rect(self):
    curr_img = self.get_curr_img()
return curr_img.get_rect(left = self.x, top = self.y)

def trigger(self, param=None):
    if self.img_index == 1:
        # print("triggering button from 1 to 2")
        self.img_index = 2
    if self.action1to2 is not None:
        if param is not None:
            self.action1to2(param)
        elif self.param is not None:
            self.action1to2(self.param)
        else:
            self.action1to2()
    else:
        raise ValueError("undefined function being called")

else:
    self.img_index = 1
    if self.action2to1 is not None:
        if param is not None:
            self.action2to1(param)
        elif self.param is not None:
            self.action2to1(self.param)
self.action2to1(self.param)

else:
    self.action2to1()
else:
    raise ValueError("undefined function being called")

import pygame
import os

class Car:
    x = 2
    y = 4
    direction = "H"  # horizontal
    image = pygame.image.load((os.path.split(os.path.realpath(__file__))[0] + "/img/car1.png"))
    collider = image.get_rect()

    def __init__(self, alias, length, x, y, direction, image, **kwargs):
        for k, v in kwargs.items():
```python
# print(type(k))

    self.__dict__[k] = v

self.alias = alias
self.length = length
self.x = x
self.y = y
self.j = int(x / 80 + 1) - 1
self.i = int(y / 80 + 1) - 1
self.direction = direction
self.image = image

if self.length == 2:
    self.image = pygame.transform.scale(self.image, (80, 160))

elif self.length == 3:
    self.image = pygame.transform.scale(self.image, (80, 240))

if direction == "H":
    self.image = pygame.transform.rotate(self.image, -90)

self.collider = self.image.get_rect()
self.collider.topleft = (x, y)

def print_dict(self):
    for a, _ in self.__dict__.items():
```
print("self.__dict__ + a")
for a, _ in Car.__dict__.items():
    print("Car.__dict__ + a")
print(Car.x)
print(self.x)
print(self.y)

gameState.py

from enum import Enum, unique

@unique
class GameState(Enum):
    MAIN = 1
    CHOOSE_CHALLENGE = 2
    CHALLENGING = 3
    CHALLENGE_COMPLETE = 4

class GameSwitches:
    auto_moving = False
    auto_move_stopping = False
```python
def reset(self):
    self.auto_moving = False
    self.auto_move_stopping = False
```

```python
import operator
import random
import pygame
import pprint
import copy
import time
import sys
import heapq

sys.path.append(".")
# from rushHour import *
from car import Car
from common import Common

class Node:
```
```python
def __init__(self, parent, move, state):
    self.parent = parent
    self.move = move
    self.state = state

class ValuedNode(Node):
    path = list()

    def __init__(self, parent, move, state, target_car):
        super().__init__(parent, move, state)
        self.target_car = target_car
        # self.get_path()
        self.depth = self.parent.depth + 1 if self.parent is not None else 0
        g_score = self.g_func()
        h_score = self.heuristic()
        # print(f"g score:{g_score}, h score:{h_score}")
        # pprint.pprint(self.path)
        self.eval = g_score + h_score

    def get_path(self):
        if self.parent is not None:
            pass
```

The code snippet represents a Python class `ValuedNode` that inherits from `Node`. It defines an `__init__` method to initialize the node with a parent, move, state, and target car. It also includes methods for setting the depth based on the parent, calculating the g_score and h_score, and retrieving the path. The `get_path` method, if present, is left to be completed.
# self.path = copy.deepcopy(self.parent.path)
#
# self.path.append(self.state)
self.path = self.parent.path + [self.state]

else:
    self.path = [self.state]

def g_func(self):
    # return len(self.path)
    return self.depth

def heuristic(self):
    eval = 0
    target_found = False
    for i in range(0, 6):
        curr_spot = self.state[2][i]
        # print(curr_spot + " " + self.target_car)
        if curr_spot == self.target_car:
            target_found = True
        if not target_found:
            continue
        else:
            if curr_spot != self.target_car and curr_spot != ".":
                # continue
eval += 1

    return eval

def __cmp__(self, other):
    if self.eval < other.eval:
        return -1
    elif self.eval == other.eval:
        return 0
    else:
        return 1

def __eq__(self, other):
    return self.state == other.state

def __lt__(self, other):
    return self.eval < other.eval

def __le__(self, other):
    return self.eval <= other.eval

def __gt__(self, other):
    return self.eval > other.eval
def __ge__(self, other):
    return self.eval >= other.eval

def load_challenges():
    global challenges
    challenges = Common.load_challenges()

def init_cars(config):
    global cars, target_car
    cars = {}
    for k, v in config.items():
print(k)

carImg = pygame.image.load("img/car_{}.png".format(k))  # returns a surface

car = Car(k, int(v["length"]), Common.translate_j_to_x(v["j"]),
Common.translate_i_to_y(v["i"]), v["direction"],
    carImg,
    board_offset=outer_edge_width)

cars[k] = car

target_car = cars[next(iter(cars))]

def init_challenge(i):
    challenge = challenges[str(i)]
    init_cars(challenge["config"])  

def init_lot_state() -> list:
    lot_state = [["." for i in range(6)] for j in range(6)]
    for _ in range(max_car_length):
        lot_state[2].append(".")  # Give additional space for the row where the exit is located
    return lot_state
def get_current_lot_state() -> list:
    current_lot_state = init_lot_state()
    for car in cars.values():
        # j = Common.translate_x_to_j(car.collider.x) - 1
        # i = Common.translate_y_to_i(car.collider.y) - 1
        j = car.j
        i = car.i
        if car.direction == "H":
            for k in range(car.length):
                current_lot_state[i][j + k] = car.alias
        elif car.direction == "V":
            for k in range(car.length):
                current_lot_state[i + k][j] = car.alias
    return current_lot_state

def locate_a_car(state, car):
    i = -1
    j = -1
    # print(f"finding {car}"")
for ii in range(6):
    if ii != 2:
        for jj in range(6):
            # print(curr_state[ii][jj])
            if state[ii][jj] == car:
                i = ii
                j = jj
                break
            else:
                continue
        break
    else:
        continue
        break
else:
    for jj in range(9):
        # print(curr_state[ii][jj])
        if state[ii][jj] == car:
            i = ii
            j = jj
            break
        else:
            continue
    break
if i == -1 or j == -1:
    raise ValueError(f"car {car} not found on the board")

return i, j

def getPossibleOneSpotMovesForOneCar(curr_state, cars, car):
    possibleMoves = list()

    i, j = locate_a_car(curr_state, car)

    # print(f"{i}{j}")

    if cars[car].direction == "H":
        # move to its right by 1 spot
        if (i != 2 and j + (cars[car].length - 1) + 1 <= 5) or (i == 2 and j + (cars[car].length - 1) + 1 <= 6):
            # in the possible movement search, right bound j index need not to be set to 8. Moving beyong index 6 is considered to be an end.

            if curr_state[i][j + (cars[car].length - 1) + 1] == ".":
                possibleMoves.append((car, "H", 1))

        # move to its left by 1 spot

        if j - 1 >= 0:
            if curr_state[i][j - 1] == ":":
                possibleMoves.append((car, "H", -1))

    else:
        # down
if i + (cars[car].length - 1) + 1 <= 5:
    if curr_state[i + (cars[car].length - 1) + 1][j] == ".":
        possibleMoves.append((car, "V", 1))
    # up
    if i - 1 >= 0:
        if curr_state[i - 1][j] == ".":
            possibleMoves.append((car, "V", -1))
return possibleMoves

def get_possible_moves_for_one_car(curr_state, cars, car):
    possibleMoves = list()
    i, j = locate_a_car(curr_state, car)
    # print(f"{i}{j}")
    if cars[car].direction == "H":
        leftSpace = j
        rightSpace = 6 - j - cars[car].length if i != 2 else 7 - j - cars[car].length
        # move left
        if leftSpace > 0:
            for s in range(1, leftSpace + 1):
                if curr_state[i][j - s] == ".":
                    possibleMoves.append((car, "H", -s))
else:
    break  # break when it hits another car

# move right
if rightSpace > 0:
    for s in range(1, rightSpace + 1):
        if curr_state[i][j + (cars[car].length - 1) + s] == ".":
            possibleMoves.append((car, "H", s))
        else:
            break  # break when it hits another car
else:
    break  # break when it hits another car

else:
    upSpace = i
    downSpace = 6 - i - cars[car].length

# up
if upSpace > 0:
    for s in range(1, upSpace + 1):
        if curr_state[i - s][j] == ".":
            possibleMoves.append((car, "V", -s))
        else:
            break  # break when it hits another car

# down
if downSpace > 0:
    for s in range(1, downSpace + 1):
if curr_state[i + (cars[car].length - 1) + s][j] == ".":
    possibleMoves.append((car, "V", s))
else:
    break  # break when it hits another car
return possibleMoves

def get_possible_one_spot_moves(curr_state, cars):
    possibleMoves = list()
    for car in cars.keys():
        pmfoc = get_possible_one_spot_moves_for_one_car(curr_state, cars, car)
        for pm in pmfoc:
            possibleMoves.append(pm)
    return possibleMoves

def get_possible_moves(curr_state, cars):
    possibleMoves = list()
    for car in cars.keys():
        pmfoc = get_possible_moves_for_one_car(curr_state, cars, car)
        for pm in pmfoc:
            possibleMoves.append(pm)
return possibleMoves

def slide(cars, state, move):
    new_state = copy.deepcopy(state)
    car = move[0]
    i, j = locate_a_car(new_state, car)
    if move[1] == "H":
        # if move[2] == 1:
        #     new_state[i][j] = "."
        #     new_state[i][j + (cars[car].length - 1) + 1] = car
        # elif move[2] == -1:
        #     new_state[i][j + (cars[car].length - 1)] = "."
        #     new_state[i][j-1] = car
        # else:
        #     raise ValueError("invalid movement distance for one heuristic step")
        for k in range(cars[car].length):
            new_state[i][j + k] = "."
    for k in range(cars[car].length):
    elif move[1] == "V":
        # if move[2] == 1:
        #     new_state[i][j] = "."
        #     new_state[i][j + (cars[car].length - 1) + 1] = car
        # elif move[2] == -1:
        #     new_state[i][j + (cars[car].length - 1)] = "."
        #     new_state[i][j-1] = car
        # else:
        #     raise ValueError("invalid movement distance for one heuristic step")
        for k in range(cars[car].length):
            new_state[i][j] = "."
    for k in range(cars[car].length):
        new_state[i][j] = car
# new_state[i][i] = "."  
# new_state[i + (cars[car].length - 1) + 1][i] = car  
# elif move[2] == -1:  
#     new_state[i + (cars[car].length - 1)][i] = "."  
#     new_state[i-1][j] = car  
# else:  
#     raise ValueError("invalid movement distance for one heuristic step")  
for k in range(cars[car].length):  
    new_state[i + k][j] = "."  
    for k in range(cars[car].length):  
        new_state[i + move[2] + k][j] = car  
else:  
    raise ValueError("invalid movement direction for a car")  
return new_state

def is_end(state, car):
    if state[2][6] == car:
        return True
    else:
        return False
def trace_path(node):
    path = list()
    n_moves = 0
    path.append(node.state)
    while isinstance(node, Node) and node.parent is not None:
        parent = node.parent
        move = node.move
        n_moves += 1
        path.append(move)
        path.append(parent.state)
        node = node.parent
    # else:
    #     path.append(node.state)
    path.reverse()
    path[len(path) - 2] = (path[len(path) - 2][0], path[len(path) - 2][1], path[len(path) - 2][2] + 2)
    return path, n_moves

def bfs(startState, cars, target_car):
    print("Starting bfs")
```python
solution = list()
n_moves = -1
n_node_expanded = 0
queue = list()

startNode = Node(None, None, startState)

queue.append(startNode)

seen = list()

seen.append(startState)

while len(queue) > 0:
    node = queue.pop(0)  # pop the next node to visit
    n_node_expanded += 1

    pprint.pprint(node.state)

    if is_end(node.state, target_car.alias):
        solution, n_moves = trace_path(node)
        break

possibleMoves = getPossibleMoves(node.state, cars)

# possibleMoves = getPossibleOneSpotMoves(node.state, cars)

print(possibleMoves)

nodes = list()

for move in possibleMoves:
    nodes.append(Node(node, move, slide(cars, node.state, move)))

for node in nodes:
```
if node.state not in seen:  # determine if the state has been visited (seen)
    queue.append(node)
    seen.append(node.state)

    # pprint.pprint(state)
    print()

    print(f"total number of node visited = {n_node_expanded}")

return solution, n_moves, n_node_expanded

def a_star_search(start_state, cars, target_car):
    solution = list()
    n_moves = 0
    open = list()
    closed = list()
    n_node_expanded = 0

    # open.append(ValuedNode(None, None, start_state, target_car.alias))
    heapq.heappush(open, ValuedNode(None, None, start_state, target_car.alias))

    while len(open) > 0:
        # nxt = open.pop(0)
        nxt = heapq.heappop(open)
        n_node_expanded += 1
if is_end(nxt.state, target_car.alias):
    solution, n_moves = trace_path(nxt)
    break

if nxt.state in closed:
    continue

closed.append(nxt.state)

possibleMoves = getPossibleMoves(nxt.state, cars)
random.shuffle(possibleMoves)
print(possibleMoves)

nodes = list()

for move in possibleMoves:
    newnode = ValuedNode(nxt, move, slide(cars, nxt.state, move),
                          target_car.alias)
    keeper = True
    for c in closed:
        if newnode.state == c: # and newnode.depth >= c.depth:
            keeper = False
            break
    if not keeper:
continue

idx_replace = -1

# for op in open:
for i in range(len(open)):
    if newnode.state == open[i].state:
        # if len(newnode.path) >= len(open[i].path):
        if newnode.depth >= open[i].depth:
            keeper = False
        else:
            idx_replace = i
        break

if keeper:
    if idx_replace > -1:
        open[idx_replace] = newnode
        heapq.heapify(open)
    else:
        # open.append(newnode)
        heapq.heappush(open, newnode)

pprint.pprint(newnode.state)
print(f"eval = {newnode.eval}"
# open.sort(key=operator.attrgetter('eval'))  # works for Python3

# open.sort()  # works for Python3

# pprint.pprint(state)

print()

print(f"total number of node visited = {n_node_expanded}")

return solution, n_moves, n_node_expanded

if __name__ == "__main__":

    load_challenges()

    init_challenge("7")

    start_time = time.time()

    solution, n_moves_a_star, n_nodes_visited_a_star = a_star_search(get_current_lot_state(), cars, target_car)

    end_time = time.time()

    a_star_time = end_time - start_time

    print("Solution found. Time elapsed for a star search: {}".format(a_star_time))

    print(f"solution:")

    pprint.pprint(solution)

    print(f"num of node visited in A* is {n_nodes_visited_a_star}")

    print(f"The number of steps taken is {n_moves_a_star}")
start_time = time.time()

solution, n_moves_bfs, n_nodes_visited_bfs = bfs(get_current_lot_state(), cars, target_car)

end_time = time.time()

bfs_time = end_time - start_time

print("Solution found. Time elapsed for bfs: {}".format(bfs_time))

print("solution:")

pprint.pprint(solution)

print("num of node visited in bfs is {}".format(n_nodes_visited_bfs))

print("The number of steps taken is {}".format(n_moves_bfs))

print("Summary:"))

print("Time elapsed for a star search: {}".format(a_star_time))

print("num of node visited in bfs is {}".format(n_nodes_visited_a_star))

print("The number of steps taken is {}".format(n_moves_a_star))

print("Time elapsed for bfs: {}".format(bfs_time))

print("num of node visited in bfs is {}".format(n_nodes_visited_bfs))

print("The number of steps taken is {}".format(n_moves_bfs))
//Format

// difficulty

//(number of cars in this challenge) (number of moves in the pre-set solution)

//(CarAlias) (length) (j) (i) (direction)

# 7

beginner

9

X 2 2 3 H
A 2 2 1 V
B 2 3 1 H
C 2 5 1 V
D 2 6 1 V
E 2 4 2 V
F 2 6 3 V
H 2 4 5 V
I 2 3 4 H

# 8
<table>
<thead>
<tr>
<th>Letter</th>
<th>Row</th>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>2</td>
<td>V</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
<td>H</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>H</td>
</tr>
<tr>
<td>O</td>
<td>3</td>
<td>6</td>
<td>H</td>
</tr>
<tr>
<td>P</td>
<td>3</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>Q</td>
<td>3</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
<td>3</td>
<td>V</td>
</tr>
</tbody>
</table>

# 0

**Beginner**

1

**Intermediate**

13

<table>
<thead>
<tr>
<th>Letter</th>
<th>Row</th>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>2</td>
<td>1</td>
<td>H</td>
</tr>
</tbody>
</table>

# 302

**Beginner**

1

**Intermediate**

13

<table>
<thead>
<tr>
<th>Letter</th>
<th>Row</th>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>3</td>
<td>2</td>
<td>H</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>2</td>
<td>V</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>5</td>
<td>H</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
<td>H</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>V</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>H</td>
</tr>
</tbody>
</table>
grand_master
10
X 2 4 3 H
A 2 4 1 V
B 2 1 2 V
C 2 2 2 H
D 2 1 4 H
O 3 1 1 H
P 3 6 1 V
Q 3 3 3 V
R 3 4 5 H
Y 3 3 6 H

# 3
beginner
6
X 2 2 3 H
A 2 2 4 H
B 2 2 5 V
C 2 3 6 H
O 3 4 3 V
P 3 6 4 V
# 303
intermediate
10
X 2 1 3 H
A 2 4 2 H
B 2 4 3 V
C 2 5 4 H
D 2 5 5 V
O 3 3 1 H
P 3 6 1 V
Q 3 3 2 V
R 3 2 5 H
Y 3 2 6 H

# 304
intermediate
13
X 2 1 3 H
A 2 6 1 V
B 2 3 2 H
C 2 3 3 V
# 29
advanced

12
X 2 1 3 H
A 2 3 2 V
B 2 6 3 V
C 2 1 4 V
D 2 2 4 H
E 2 4 4 H
F 2 2 5 H
G 2 4 5 V
H 2 6 5 V
O 3 1 1 H
P 3 5 1 V
R 3 1 6 H

# 19
intermediate

8
X 2 3 3 H
A 2 3 1 V
B 2 4 1 H
import json

import pygame
import sys
import math
import time
import pygame_functions
import pprint
import math

sys.path.append(".")
import os

directory = os.path.split(os.path.realpath(__file__))[0]
from rushhour.car import Car
from rushhour.gameState import GameState
from rushhour.gameState import GameSwitches
from rushhour.common import *
import rushhour.search

# from pygame_textinput import TextInput

print(pygame.ver)

# initialize the pygame
pygame.init()

globalWidth = 1000
globalHeight = 700

# colors
white = (255, 255, 255)
black = (0, 0, 0)
red = (200, 0, 0)
green = (0, 200, 0)
yellow = (200, 200, 0)
bright_red = (255, 0, 0)
bright_green = (0, 255, 0)
bright_yellow = (255, 255, 0)
gameState = GameState.MAIN

game_switches = GameSwitches()

# create the screen
screen = pygame.display.set_mode((1000, 700))  # return a Surface
pygame_functions.screen = screen
pygame_functions.background = pygame_functions.Background()
pygame_functions.background.surface = screen.copy()

# title and icon
pygame.display.set_caption("Rush Hour Emulator")
icon = pygame.image.load(f"{directory}/img/rush-hour.jpg")
pygame.display.set_icon(icon)

# Background
background = pygame.image.load(f"{directory}/img/background.png").convert()
screen.blit(background, (0, 0))

# music
pygame.mixer.init()
pygame.mixer.music.load(f"{directory}/audio/bgm.mp3")
pygame.mixer.music.play(-1, 0)
pygame.mixer.music.set_volume(0.05)

# sounds
sound_hooray = pygame.mixer.Sound(f"{directory}/audio/children-hooray-joy-shout.wav")
sound_hooray.set_volume(0.2)

# clock
fpsClock = pygame.time.Clock()

# global widgets
buttons_global = list()

# challenging buttons
buttons_challenging = list()

# challenge_complete buttons
buttons_challenge_complete = list()

# default username
username = "player A"

# data
user_data = {}

# icons in game
icon_difficulty_beginner = pygame.image.load(f"{directory}/img/difficulty_beginner.png")
icon_difficulty_intermediate = pygame.image.load(f"{directory}/img/difficulty_intermediate.png")
icon_difficulty_advanced = pygame.image.load(f"{directory}/img/difficulty_advanced.png")
icon_difficulty_expert = pygame.image.load(f"{directory}/img/difficulty_expert.png")
icon_difficulty_grand_master = pygame.image.load(f"{directory}/img/difficulty_grand_master.png")

def blit_background():
    screen.blit(background, (0, 0))
    # show_global_buttons()
def init_user_data(username):
    global user_data
    user_data[username] = dict()
    # user_data[username]["nick"] = username
    user_data[username]["challenges_record"] = {k: dict() for k in challenges.keys()}
    for k, v in user_data[username]["challenges_record"].items():
        v["challenge_alias"] = k
        v["best_score"] = 0
        v["min_moves_taken"] = "N/A"

def load_user_data():
    global user_data
    if os.path.exists(directory + "/data/userdata.json"):
        with open((directory + "/data/userdata.json"), "rt") as f:
            datastr = f.read()
        if datastr !=""
            user_data = json.loads(datastr)
        if len(user_data) == 0:
            init_user_data(username)
else:
    with open((directory + "/data/userdata.json"), "w+t") as f:  # if data/userdata.json does not exist, create one
        init_user_data(username)
        f.write(json.dumps(user_data, sort_keys=False, indent=4, separators=(',', ':')))

def write_user_data():
    with open((directory + "/data/userdata.json"), "wt") as f:
        f.write(json.dumps(user_data, sort_keys=False, indent=4, separators=(',', ':')))

_circle_cache = {}

def _circlepoints(r):
    r = int(round(r))
    if r in _circle_cache:
        return _circle_cache[r]
    x, y, e = r, 0, 1 - r
    _circle_cache[r] = points = []
    while x >= y:
points.append((x, y))
y += 1
if e < 0:
    e += 2 * y - 1
else:
    x -= 1
    e += 2 * (y - x) - 1
points += [(y, x) for x, y in points if x > y]
points += [(-x, y) for x, y in points if x]
points += [(x, -y) for x, y in points if y]
points.sort()
return points

def render(text, font, gfcolor=pygame.Color('dodgerblue'), ocolor=(255, 255, 255), opx=2):
    textsurface = font.render(text, True, gfcolor).convert_alpha()
w = textsurface.get_width() + 2 * opx
h = font.get_height()

osurf = pygame.Surface((w, h + 2 * opx)).convert_alpha()
osurf.fill((0, 0, 0, 0))
surf = osurf.copy()

osurf.blit(font.render(text, True, ocolor).convert_alpha(), (0, 0))

for dx, dy in _circlepoints(opx):
    surf.blit(osurf, (dx + opx, dy + opx))

surf.blit(textsurface, (opx, opx))
return surf

def text_objects(text, font):
    text_surface = font.render(text, True, black)
    return text_surface, text_surface.get_rect()

def button(msg, x, y, w, h, ic, ac, action=None, param=None):
    """
    ic: inactive color
    ac: active color
    """
mouse = pygame.mouse.get_pos()

click = pygame.mouse.get_pressed()

# print(click)

if x + w > mouse[0] > x and y + h > mouse[1] > y:
    pygame.draw.rect(screen, ac, (x, y, w, h))

if click[0] == 1 and action is not None:
    if param is not None:
        action(param)
    else:
        action()
    return

else:
    pygame.draw.rect(screen, ic, (x, y, w, h))

small_text = pygame.font.SysFont("comicsansms", 20)

text_surf, text_rect = text_objects(msg, small_text)

text_rect.center = ((x + (w / 2)), (y + (h / 2)))

screen.blit(text_surf, text_rect)

# def register_challenging_button(x, y, w, h):

#     global buttons_challenging
buttons_challenging.append(pygame.Rect(x, y, w, h))

def clear_register_challenging_button(x, y, w, h):
    global buttons_challenging
    buttons_challenging.clear()

names = locals()

def challenge_button(msg, x, y, w, h, ic, ac, name='', action=None, param=None):
    ""
    ic: inactive color
    ac: active color
    ""
    global names
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()
    # print(click)
    names[name] = pygame.Rect(x, y, w, h)
    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(screen, ac, names[name])
if click[0] == 1 and action is not None:
    if param is not None:
        action(name, param)
    else:
        action(name)
else:
    pygame.draw.rect(screen, ic, (x, y, w, h))

small_text = pygame.font.SysFont("comicsansms", 20)
text_surf, text_rect = text_objects(msg, small_text)
text_rect.center = ((x + (w / 2)), (y + (h / 2)))
screen.blit(text_surf, text_rect)

def register_global_buttons():
    global buttons_global

    # music button
    button_img_music = pygame.image.load("{directory}/img/icon_music.png")
    button_img_music_disabled = pygame.image.load("{directory}/img/icon_music_disabled.png")
    button_img_music = pygame.transform.scale(button_img_music, (50, 50))
    button_img_music_disabled =
pygame.transform.scale(button_img_music_disabled, (50, 50))

    button_music = IconButton(20, globalHeight - 50 - 20, button_img_music,
button_img_music_disabled, on_disable_music, on_enable_music)

    buttons_global.append(button_music)

def show_global_buttons():
    for button in buttons_global:
        button.blit(screen)

def check_global_buttons(event):
    # print(f"called check_global_buttons with event type{event.type}")
    if hasattr(event, "pos"):
        mouse_x, mouse_y = event.pos
    else:
        mouse_x, mouse_y = pygame.mouse.get_pos()

    button_to_operate = None
    for button in buttons_global:
        # print(f"button postion {button.get_rect()}, mouse position {mouse_x},
{mouse_y}"")
        if button.get_rect().collidepoint(mouse_x, mouse_y):
button_to_operate = button

    break

else:
    return

if event.type == pygame.MOUSEBUTTONDOWN:
    button_to_operate.on_mouse_down()

elif event.type == pygame.MOUSEBUTTONUP:
    button_to_operate.on_mouse_up()

def on_disable_music():
    print("on_disable_music")
    pygame.mixer.music.stop()

def on_enable_music():
    print("on_enable_music")
    pygame.mixer.music.play(-1, 0)

def on_start_challenge():
    global gameState, n_moves_taken, moves_history, score,
selected_challenge_alias, dragging_car

init_challenge(selected_challenge_alias)

gameState = GameState.CHALLENGING

register_challenging_buttons()

# screen.fill(black)

blit_background()

game_switches.reset()

reset_challenge()

def register_challenging_buttons():

    global buttons_challenging

    # hint button

    button_hint = TextButton(550, 240, 100, 50, green, bright_green, "Hint", on_hint)

    button_hint.set_name("button_hint")

    buttons_challenging.append(button_hint)

    # return button

    button_return = TextButton(550, 310, 100, 50, green, bright_green, "Return", on_return_to_view_choose_challenge)

    buttons_challenging.append(button_return)

    # restart button
button_restart = TextButton(550, 380, 100, 50, yellow, bright_yellow, "Restart", on_restart_challenge)

buttons_challenging.append(button_restart)

# quit button

button_quit = TextButton(550, 450, 100, 50, red, bright_red, "Quit", on_quit)

buttons_challenging.append(button_quit)

def show_challenging_buttons():
    for button in buttons_challenging:
        if button.name == "button_hint":
            if game_switches.auto_moving:
                button.param = True
            else:
                button.param = False

        button.blit(screen)

def check_challenging_buttons(event):
    # print(f"called check_challenging_buttons with event type{event.type}"")
    if hasattr(event, "pos"):
        mouse_x, mouse_y = event.pos
else:
    mouse_x, mouse_y = pygame.mouse.get_pos()

button_to_operate = None

for button in buttons_challenging:
    # print(f"button position {button.get_rect()}, mouse position {mouse_x}, {mouse_y}"
    if button.get_rect().collidepoint(mouse_x, mouse_y):
        button_to_operate = button
        break
else:
    return

if event.type == pygame.MOUSEBUTTONDOWN:
    button_to_operate.on_mouse_down()
elif event.type == pygame.MOUSEBUTTONUP:
    button_to_operate.on_mouse_up()

def register_challenge_complete_buttons():
    global buttons_challenge_complete

    # return button
    button_return = TextButton(550, 310, 100, 50, green, bright_green, "Return", on_return_to_view_choose_challenge)
buttons_challenge_complete.append(button_return)

# restart button

button_restart = TextButton(550, 380, 100, 50, yellow, bright_yellow, "Restart", on_restart_challenge)

buttons_challenge_complete.append(button_restart)

# quit button

button_quit = TextButton(550, 450, 100, 50, red, bright_red, "Quit", on_quit)

buttons_challenge_complete.append(button_quit)

def show_challenge_complete_buttons():
    for button in buttons_challenge_complete:
        button.blit(screen)

def check_challenge_complete_buttons(event):
    # print(f"called check_challenge_complete_buttons with event
type{event.type}")

    if hasattr(event, "pos"):
        mouse_x, mouse_y = event.pos
    else:
        mouse_x, mouse_y = pygame.mouse.get_pos()
button_to_operate = None

for button in buttons_challenge_complete:
    # print(f"button position {button.get_rect()}, mouse position {mouse_x},
    (mouse_y)"
    if button.get_rect().collidepoint(mouse_x, mouse_y):
        button_to_operate = button
        break

else:
    return

if event.type == pygame.MOUSEBUTTONDOWN:
    button_to_operate.on_mouse_down()

elif event.type == pygame.MOUSEBUTTONUP:
    button_to_operate.on_mouse_up()

# initial
register_global_buttons()

# board outline 外框
challenge_area_width = 540
challenge_area_height = 540
rect_boardOutline = pygame.Rect(0, 0, challenge_area_width,
challenge_area_height)
outer_edge_width = 30

def blit_challenge_area():
    pygame.draw.rect(screen, white, rect_boardOutline)

# interactive board area (Surface) 停车场部分（Surface）
board_area = pygame.Surface((480, 480))

def blit_board_area():
    board_area.fill(white)
    blit_lot()
    blit_spots()
    blit_cars()
    screen.blit(board_area, (outer_edge_width, outer_edge_width))

# draws the parking lot on the board area (Rect) 在停车场上画停车位（Rect）
lot = pygame.Rect(0, 0, 480, 480)
def blit_lot():
    pygame.draw.rect(board_area, black, lot, 1)
    exit_cover = pygame.draw.line(board_area, white, (480, 160), (480, 240), 8)

def blit_spots():
    for i in range(6):
        for j in range(6):
            spot = pygame.Rect(20 + 80 * i, 20 + 80 * j, 40, 40)
            pygame.draw.rect(board_area, black, spot, 3)

# car for test

car1Img = pygame.image.load(f"{directory}/img/car1.png") # returns a surface

# car1 = Car("X", 2, 0, 160, "h", car1Img)
#
# car2Img = pygame.image.load("img/car1.png") # returns a surface
# car2 = Car("A", 2, 400, 160, "v", car2Img)
# print(car2.collider.x)
# print(car2.collider.y)
# print(car2.collider)

challenges = {}
cars = {}

def load_challenges():
    global challenges
    challenges = Common.load_challenges()

load_challenges()
print(challenges)
load_user_data()
pprint.pprint(user_data)

selected_challenge_alias = ""

def on_challenge_selected(selected_challenge):
    global selected_challenge_alias
    selected_challenge_alias = selected_challenge
def view_choose_challenge():

global challenges, challenge_alias, names

blit_background()

font = pygame.font.Font("freesansbold.ttf", 48)
# font = pygame.font.SysFont("Arial", 96, True)
title = font.render("Choose a Challenge to Start", True, white)
title_x = (globalWidth - title.get_width()) / 2

title_y = 50

screen.blit(title, (int(title_x), int(title_y)))

font = pygame.font.SysFont("comicsansms", 20, True)
desc_be = font.render("Beginner", True, white)
desc_y = 50+48+10
desc_be_x = 20

screen.blit(desc_be, (int(desc_be_x), int(desc_y)))
screen.blit(icon_difficulty_beginner, (desc_be_x + desc_be.get_width() + 10, desc_y))

desc_in = font.render("Intermediate", True, white)
desc_in_x = 20 + desc_be.get_width() + 10 + 35 + 20

screen.blit(desc_in, (int(desc_in_x), int(desc_y)))
screen.blit(icon_difficulty_intermediate, (desc_in_x + desc_in.get_width() + 10, desc_y))

desc_ad = font.render("Advanced", True, white)
desc_ad_x = 20 + desc_in_x + desc_in.get_width() + 10 + 35 + 20
screen.blit(desc_ad, (int(desc_ad_x), int(desc_y)))
screen.blit(icon_difficulty_advanced, (desc_ad_x + desc_ad.get_width() + 10, desc_y))

desc_ex = font.render("Expert", True, white)
desc_ex_x = 20 + desc_ad_x + desc_ad.get_width() + 10 + 35 + 20
screen.blit(desc_ex, (int(desc_ex_x), int(desc_y)))
screen.blit(icon_difficulty_expert, (desc_ex_x + desc_ex.get_width() + 10, desc_y))

desc_gm = font.render("Grand Master", True, white)
desc_gm_x = 20 + desc_ex_x + desc_ex.get_width() + 10 + 70 + 20
screen.blit(desc_gm, (int(desc_gm_x), int(desc_y)))
screen.blit(icon_difficulty_grand_master, (desc_gm_x + desc_gm.get_width() + 10, desc_y))
# font = pygame.font.Font("freesansbold.ttf", 32)
font = pygame.font.SysFont("Arial", 32, True)
selected = font.render("Selected challenge: \{\}".format(selected_challenge_alias), True, white)
    selected_x = (globalWidth - selected.get_width()) / 2
    selected_y = 550+30
# screen.blit(selected, (int(selected_x), int(selected_y)))
    screen.blit(render("Selected challenge: \{\}".format(selected_challenge_alias), pygame.font.SysFont("Arial", 32, True), white, black, 2),
        (int(selected_x), int(selected_y)))

# blits challenges available
    i = 0
    j = 0
    for k in challenges.keys():
        if i > 9:
            i = 1
            j += 1
        else:
            i += 1
        challenge_button(k, 30+230*j, 130+40*i, 60, 30, white, red, k,
on_challenge_selected)

difficulty_icon = names["icon_difficulty_{challenges[k]['difficulty']}"]

screen.blit(difficulty_icon, (30+230*j+60+10, 130+40*i))

def show_challenge_alias():

    # font = pygame.font.Font("freesansbold.ttf", 24)
    # font = pygame.font.SysFont("Arial", 30, True)
    # font = pygame.font.SysFont("comicsansms", 30, True)

    # title = font.render("Challenge {}".format(selected_challenge_alias), True, white)

    title_x = rect_boardOutline.width + 20

    title_y = 10

    # screen.blit(title, (int(title_x), int(title_y)))

    screen.blit(render("CHALLENGE {}".format(selected_challenge_alias),
                      pygame.font.SysFont("comicsansms", 36, True),
                      pygame.Color("dodgerblue"),
                      black, 3), (int(title_x), int(title_y)))

def show_best_history():

    # font = pygame.font.Font("freesansbold.ttf", 24)

    # font = pygame.font.SysFont("Arial", 24, True)
# font = pygame.font.SysFont("comicsansms", 30, True)

# title = font.render("Your Best Score: ".format(selected_challenge_alias), True, white)

title_x = rect_boardOutline.width + 20

title_y = 70

# screen.blit(title, (int(title_x), int(title_y)))

screen.blit(render("Your Best Score: {} with {} moves".

format(user_data[username]["challenges_record"]

[selected_challenge_alias]["best_score"],

user_data[username]["challenges_record"]

[selected_challenge_alias]["min_moves_taken"]

, pygame.font.SysFont("freesansbold", 30, True),

pygame.Color(("springgreen3")), black, 2),

(int(title_x), int(title_y)))

def show_n_moves_taken():

    # font = pygame.font.Font("freesansbold.ttf", 24)

    # font = pygame.font.SysFont("Arial", 24, True)

    # font = pygame.font.SysFont("comicsansms", 30, True)

    # title = font.render("# moves taken.", True, white)
title_x = rect_boardOutline.width + 20

title_y = 100

# screen.blit(title, (int(title_x), int(title_y)))

screen.blit(render("# moves taken:", pygame.font.SysFont("Arial", 24, True),
white, black, 2),
    (int(title_x), int(title_y)))

button(str(n_moves_taken), rect_boardOutline.width + 20, 130, 60, 30, white, white)

def show_score():

    # font = pygame.font.Font("freesansbold.ttf", 24)
    # font = pygame.font.SysFont("Arial", 24, True)
    # font = pygame.font.SysFont("comicsansms", 30, True)
    # title = font.render("Score:", True, white)

    title_x = rect_boardOutline.width + 20
    title_y = 170

    # screen.blit(title, (int(title_x), int(title_y)))

    screen.blit(render("Score:", pygame.font.SysFont("Arial", 24, True), white, black, 2),
        (int(title_x), int(title_y)))

    # button("{0:.2f}".format(score), rect_boardOutline.width + 20, 200, 60, 30,
white, white)

    button("{0:d}".format(score), rect_boardOutline.width + 20, 200, 60, 30, white, white)

def on_return_to_view_choose_challenge():
    print("Clicked return")
    global gameState
    game_switches.reset()
    gameState = GameState.MAIN
    blit_background()

def init_cars(config):
    global cars, target_car
    cars = {}
    for k, v in config.items():
        print(k)
        carImg = pygame.image.load("img/car_{}.png").format(k)  # returns a surface
        car = Car(k, int(v["length"]), Common.translate_j_to_x(v["j"]), Common.translate_i_to_y(v["i"]), v["direction"], carImg,
board_offset=outer_edge_width)

cars[k] = car
target_car = cars[next(iter(cars))]

def init_challenge(i):
    challenge = challenges[str(i)]
    init_cars(challenge["config"])

# 命令行选择关卡
# print("Challenges list:")
# challenge_list_string = ""
# for k in challenges.keys():
#     challenge_list_string += k + " "
# print(challenge_list_string)
# selected_challenge_alias = input("Please choose a challenge to begin:")
# selected_challenge_alias = "7"

# init_challenge(selected_challenge_alias)
# target_car = cars[next(iter(cars))]  # Moved into function init_challenge()
max_car_length = 3  # the possible max length of all cars. It can be dynamically
calculated from the data of the each challenge but at present it is hard coded

```
n_moves_taken = 0  # the number of moves that the player has made
moves_history = []  # the history
score = 0
```

def blit_cars():
    for car in cars.values():
        board_area.blit(car.image, (car.collider.x, car.collider.y))
        # pygame.draw.rect(board_area, red, car.collider)  # Cover the cars images with red rectangles.

```
def init_lot_state() -> list:
    lot_state = ["." for i in range(6)] for j in range(6]
    for _ in range(max_car_length):
        lot_state[2].append(".")  # Give additional space for the row where the exit is located
    return lot_state
```

def get_current_lot_state() -> list:
current_lot_state = init_lot_state()

for car in cars.values():
    j = Common.translate_x_to_j(car.collider.x) - 1
    i = Common.translate_y_to_i(car.collider.y) - 1

    if car.direction == "H":
        for k in range(car.length):
            current_lot_state[i][j + k] = car.alias

    elif car.direction == "V":
        for k in range(car.length):
            current_lot_state[i + k][j] = car.alias

return current_lot_state

def print_lot_state(lot_state: list) -> None:
    for i in range(len(lot_state)):
        print(lot_state[i])

def show_waiting():
    print("waiting")

    font = pygame.font.SysFont("comicsansms", 30, True)
    title = font.render("Waiting for Computation...", True, white)
title_x = (board_area.get_width() - title.get_width()) / 2

title_y = board_area.get_height() / 2 - 30 / 2

# screen.blit(title, (int(title_x), int(title_y)))

board_area.blit(render("Waiting for Computation...",
pysame.font.SysFont("comicsansms", 30, True), bright_red, black, 2),
(int(title_x), int(title_y)))

screen.blit(board_area, (outer_edge_width, outer_edge_width))

pygame.display.update()

def get_hint(curr_state, cars, target_car):
    solution, min_n_step, _ = rushhour.search.bfs(curr_state, cars, target_car)
    moves = list()
    for i in range(1, len(solution), 2):
        moves.append(solution[i])
    return moves

def on_hint(stop = False):
    if stop:
print("on hint stopping")

else:
    print("on_hint")

global game_switches, moves, auto_move_generator

if not stop:
    show_waiting()
    curr_state = get_current_lot_state()
    moves = get_hint(curr_state, cars, target_car)
    moves.pop()
    game_switches.auto_moving = True
    auto_move_generator = auto_move(moves)
else:
    game_switches.auto_move_stopping = True

def auto_move(moves):
    global n_moves_taken, auto_move_generator
    for move in moves:
        car = cars[move[0]]
        origin_x = car.collider.x
        origin_y = car.collider.y
        car_i = Common.translate_y_to_i(origin_y)
car_j = Common.translate_x_to_j(origin_x)

if move[1] == "H":
    car_j += move[2]
    dest_x = Common.translate_j_to_x(car_j)
    dx = abs(dest_x - origin_x)
    inter_x = interpolation_x(origin_x, dest_x)
    for x in inter_x:
        car.collider.x = x
        yield
else:
    car_i += move[2]
    dest_y = Common.translate_i_to_y(car_i)
    dy = abs(dest_y - origin_y)
    inter_y = interpolation_y(origin_y, dest_y)
    for y in inter_y:
        car.collider.y = y
        yield

n_moves_taken += 1

if game_switches.auto_move_stopping:
    break

game_switches.auto_moving = False

game_switches.auto_move_stopping = False
moves.clear()

auto_move_generator = None

# Considering moving these four lines above to the main while loop when StopIteration is catched.

def interpolation_x(sink_x, dest_x):
    x = sink_x

    if sink_x > dest_x:
        # move left
        for i in range(1, sink_x - dest_x + 1, 1):
            x += -1
            yield x
    else:
        # move right
        for i in range(1, dest_x - sink_x + 1, 1):
            # print(i)
            x += 1
            yield x

def interpolation_y(sink_y, dest_y):

y = sink_y

if sink_y > dest_y:
    
    # move up
    for i in range(1, sink_y - dest_y + 1, 1):
        y += -1
    yield y

else:
    
    # move right
    for i in range(1, dest_y - sink_y + 1, 1):
        y += 1
    yield y

def reset_challenge():
    
    global dragging_car, rectangle_dragging, n_moves_taken, moves_history, score

    dragging_car = ""
    rectangle_dragging = False
    n_moves_taken = 0
    moves_history = []
    moves_history.append(get_current_lot_state())
    score = 0
def on_restart_challenge():
    # game_switches.reset()
    # init_challenge(selected_challenge_alias)
    on_start_challenge()

def on_complete_challenge():
    global gameState, score
    gameState = GameState.CHALLENGE_COMPLETE
    register_challenge_complete_buttons()

    blit_background()
    blit_challenge_area()
    blit_board_area()

    print("Challenge {} complete".format(selected_challenge_alias))
    # score = (1 - 1 / n_moves_sol * (n_moves_taken - n_moves_sol)) * 100
    score = 100 * (1 - (n_moves_taken -

    int(challenges[selected_challenge_alias]["n_moves_sol"]) / )

    int(challenges[selected_challenge_alias]["n_moves_sol"])

    if score < 0:
score = 0
score = math.floor(score)
print(score)

user_data[username]["challenges_record"][selected_challenge_alias]["best_score"] = max(score,
user_data[username]["challenges_record"][selected_challenge_alias]["best_score"])

user_history_min_moves_taken =
user_data[username]["challenges_record"][selected_challenge_alias]["min_moves_taken"]

if user_history_min_moves_taken == "N/A":

user_data[username]["challenges_record"][selected_challenge_alias]["min_moves_taken"] = n_moves_taken

else:

user_data[username]["challenges_record"][selected_challenge_alias]["min_moves_taken"] = min(n_moves_taken, user_history_min_moves_taken)

write_user_data()
sound_hooray.play()
def on_quit():
    # pass
    global running
    running = False
    quit()

rectangle_dragging = False
dragging_car = ""
offset_x = 0
offset_y = 0
running = True
while running:
    # show_global_buttons()
    if gameState == GameState.MAIN:
        view_choose_challenge()
        show_global_buttons()
        # word_box = pygame_functions.makeTextBox(300, 450, 150, 0, "", 3, 22)
        # pygame_functions.showTextBox(word_box)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
running = False
quit()
elif event.type in [pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP]:
    check_global_buttons(event)
    if selected_challenge_alias != "":
        button("GO!", 150, 580, 100, 50, green, bright_green, on_start_challenge)
        if gameState == GameState.MAIN:
            button("Quit", 750, 580, 100, 50, red, bright_red, on_quit)
elif gameState == GameState.CHALLENGGING:
    show_challenging_buttons()
    show_global_buttons()
    # button("Return", 550, 310, 100, 50, green, bright_green, on_return_to_view_choose_challenge)
    if gameState == GameState.CHALLING:
        # button("Hint", 550, 240, 100, 50, green, bright_green, on_hint)
        # print(game_switches.auto_moving)
        if game_switches.auto_moving:
            if auto_move_generator is not None:
                try:
                    auto_move_generator.__next__()
                except StopIteration as e:
print("StopIteration happened")

print(e)

# button("Restart", 550, 380, 100, 50, yellow, bright_yellow, on_restart_challenge)

# button("Quit", 550, 450, 100, 50, red, bright_red, on_quit)

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
        quit()
    elif event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1:
            check_global_buttons(event)
            check_challenging_buttons(event)
            mouse_x, mouse_y = event.pos
            mouse_x -= outer_edge_width  # Translate to its relative coordinate against board_area
            mouse_y -= outer_edge_width

for car in cars.values():
    if car.collider.collidepoint((mouse_x, mouse_y)):
        # print(event.pos)
        # print(car.collider)
        dragging_car = car.alias
print("Clicked on car " + car.alias)

rectangle_dragging = True

# mouse_x, mouse_y = event.pos
offset_x = car.collider.x - mouse_x
offset_y = car.collider.y - mouse_y

break

elif event.type == pygame.MOUSEBUTTONDOWN:  # Ends dragging
    if event.button == 1:
        rectangle_dragging = False
        check_global_buttons(event)
        check_challenging_buttons(event)

if dragging_car != "":
    target = cars[dragging_car]

    # Upon dragging ending, move the dragged car to the nearest
    # spot depending on its current position
    if target.direction == "H":
        for i in range(5):
            if -40 + i * 80 < target.collider.x < 40 + i * 80:
                target.collider.x = i * 80
                break

    elif target.direction == "V":
        for i in range(5):
if -40 + i * 80 < target.collider.y < 40 + i * 80:

target.collider.y = i * 80

break

cls = get_current_lot_state()

print_lot_state(cls)

if cls != moves_history[len(moves_history) - 1]:
    moves_history.append(cls)
    n_moves_taken += 1

elif event.type == pygame.MOUSEMOTION:  # Move the car being dragged

    if rectangle_dragging:
        mouse_x, mouse_y = event.pos
        mouse_x -= outer_edge_width
        mouse_y -= outer_edge_width

        # print(mouse_x)

        target = cars[dragging_car]
        colliders = [v.collider for k, v in cars.items() if k != dragging_car]

        if target.direction == "H":
            prevx = target.collider.x

            target.collider.x = mouse_x + offset_x

            if target.collider.collidelist(colliders) > -1 or (not
                lot.contains(target.collider) and target.collider.y != Common.translate_i_to_y(3) or
if target.collider.y == Common.translate_i_to_y(3) and target.collider.x < 0):
    target.collider.x = prevx
    offset_x = target.collider.x - mouse_x  # For avoiding the
dragged car from flashing over another car when the cursor moves over another
car. 为了防止闪现跨过车辆。
    
    **elif** target.direction == "V":
        prevy = target.collider.y
        target.collider.y = mouse_y + offset_y
        if target.collider.collidelist(colliders) > -1 or not
        lot.contains(target.collider):
            target.collider.y = prevy
            offset_y = target.collider.y - mouse_y  # For avoid the
dragged car from flashing over another car when the cursor moves over another
car. 为了防止闪现跨过车辆。

    # print("{} {}").format(target_car.collider.x, challenge_area_width -
2 * outer_edge_width))
    if target_car.collider.x > challenge_area_width - 2 *
outer_edge_width:
        cls = get_current_lot_state()
        print_lot_state(cls)
        if cls != moves_history[len(moves_history) - 1]:
moves_history.append(cls)

n_moves_taken += 1

on_complete_challenge()

# else:
#     check_challenging_buttons(event)

elif event.type == pygame.KEYDOWN:  # for testing
    if event.key == pygame.K_p:
        cls = get_current_lot_state()
        print_lot_state(cls)
        # blit_board_outline()
        blit_challenge_area()
        blit_board_area()
        show_challenge_alias()
        show_best_history()
        show_n_moves_taken()

        # update cars position
        # blit_cars()

elif gameState == GameState.CHALLENGE_COMPLETE:
    show_challenge_complete_buttons()
    show_global_buttons()
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
        quit()
    elif event.type in [pygame.MOUSEBUTTONDOWN,
                        pygame.MOUSEBUTTONUP]:
        check_global_buttons(event)
        check_challenge_complete_buttons(event)

        # screen.fill(black)

        # button("Return", 550, 310, 100, 50, green, bright_green,
        on_return_to_view_choose_challenge)

        if gameState == GameState.CHALLENGE_COMPLETE:
            # button("Restart", 550, 380, 100, 50, yellow, bright_yellow,
            on_restart_challenge)

            # button("Quit", 550, 450, 100, 50, red, bright_red, on_quit)

            show_challenge_alias()
            show_best_history()
            show_n_moves_taken()
            show_score()

        # update
        pygame.display.update()
# clock ticks

def fpsClock.tick(60):

input("Please enter any key to exit")

if __name__ == "__main__":
    # first_screen()
    pass
APPENDIX B

ATTRIBUTION
The image of the white truck used in the application is designed by Freepik.
REFERENCES


