Project 1. Search

Rush Hour: Solve the Traffic Jam!

1 Introduction

Rush Hour is a sliding block puzzle game where the goal is to move the specified car to the exit, navigating through a gridlocked traffic jam. All vehicles can only move forward or backward, and must remain on their respective horizontal or vertical tracks [1].

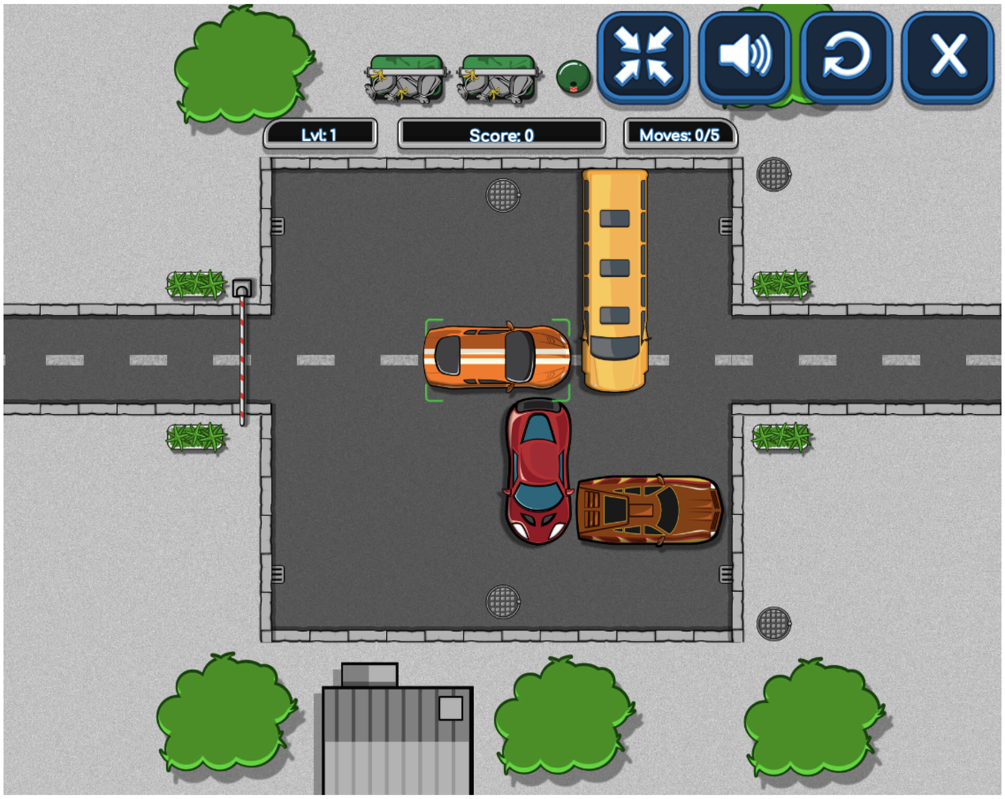

Figure 1: Example Rush Hour board. You can try it via this link.

In this first project, students will develop an intelligent solver for the Rush Hour puzzle using various **search algorithms**, including (1) **Breadth-First Search (BFS)**, (2) **Depth-First Search (DFS)**, (3) **Uniform-Cost Search (UCS)**, and (4) **A* Search**. The performance of each algorithm will be evaluated on metrics such as (1) **search time**, (2) **memory usage**, and (3) **number of expanded nodes**.

[1] "Rush Hour (puzzle)." Wikipedia. `https://en.wikipedia.org/wiki/Rush_Hour_(puzzle)`.

2 Description

Rush hour puzzle is played on a 6 × 6 grid that contains **cars** (length 2) and **trucks** (length 3). Each vehicle is either **horizontally or vertically oriented** and can only move along its axis, which is **forwards** or **backwards**, but **cannot turn or move sideways**. A move consists of **shifting a vehicle by one empty cell** in its allowed direction (see Figure 2).

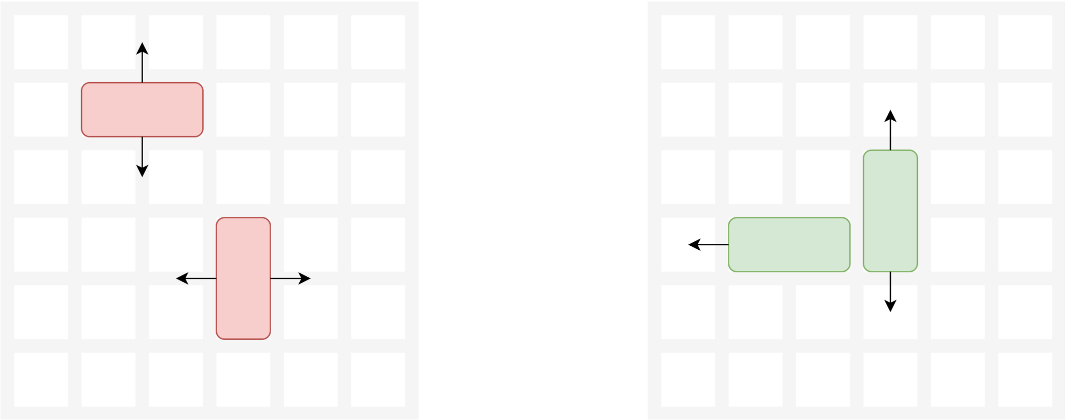

Figure 2: **Invalid moves** (left) and **valid moves** (right).

Vehicles **cannot overlap** and **cannot jump over one another**. A move is only valid if all the cells that the vehicle will occupy are currently empty (see Figure 3).

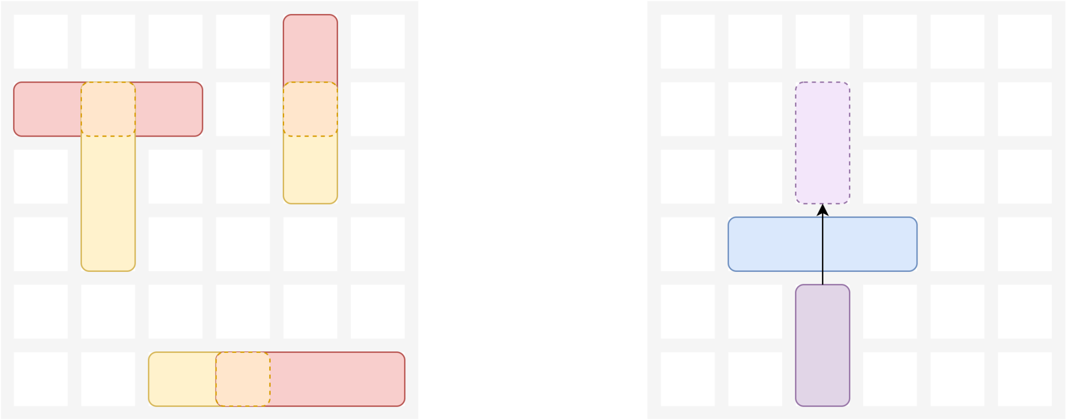

Figure 3: **Vehicles cannot overlap or jump.** On the left: overlapping vehicles are not allowed. On the right: jumping over another vehicle is not permitted.

The puzzle is solved when the **specified car** successfully exits via the gate (at the column 6).

3 Requirements

Before we get started, here are some important things to consider:

- The source code must be written in **Python**.
- You can use supporting libraries, but you must implement the search algorithms **yourself**.
- There is **no restrictions** on how to organize your code. However, it should be structured professionally and clearly.

3.1 BFS Solver (10 pts)

- Implement a Rush Hour solver using **Breadth-First Search**.

3.2 DFS Solver (10 pts)

- Implement a Rush Hour solver using **Depth-First Search**.
- The Iterative Deepening Search (IDS) algorithm can be used as an alternative if you want.

3.3 UCS Solver (10 pts)

The **cost of a move** is defined as the **length of the vehicle being moved**. For example:

- Moving a car of length 2 (e.g., a red car) by one unit has a cost of 2.
- Moving a truck of length 3 by one unit has a cost of 3.

Implement a Rush Hour solver using **Uniform-Cost Search**.

3.4 A* Solver (10 pts)

- Implement a Rush Hour solver using **A* Search** with an **appropriate heuristic** (e.g., number of blocking cars).
- Justify your choice of heuristic and analyze its effectiveness in the report.

3.5 Interactive GUI (10 pts)

- Animate the solution step-by-step, with controls to play/pause, reset, and select algorithms.
- Show real-time statistics such as step count and total cost.
- Display a message if no solution is found.

3.6 Design maps (10 pts)

- Design at least 10 maps with varying levels of difficulty and structural diversity.
- Evaluate the implemented search algorithms and record the **search time**, **memory usage**, and **number of expanded nodes** to fulfill the **Experiments** section in the **Report** below.

3.7 Report (30 pts)

Submit a well-formatted PDF report with:

- **Project Planning and Task Distribution**: Document the team member responsibilities, which includes information on each task assigned to team members and the completion rate. E.g., Student A has percentage of completion 90% and the group work has total score of 9.0, then A receives a score of 9.0 * 90% = 8.1.

  Therefore, it is important to evaluate the contribution of group members fairly.
- **Algorithm Description (10 pts)**: Provide a detailed explanation of each search algorithm.
- **Description of the maps (5 pts)**: Provide the detailed information for each map.
- **Experiments (15 pts)**: Assess the search performance via Search Time, Memory Usage, and Expanded Nodes; with measurements, visualizations (chart) and insights.
- **References** and **Appendix**.

3.8 Video (10 pts)

Upload a demo to YouTube and provide the **public link** in the report. The video content includes:

- **Graphical Interface (5 pts)**: Demonstrate the GUI, highlighting key visual elements.
- **Feature Presentation (5 pts)**: Demonstrate your implemented functionalities concisely. The video should have subtitles or narration to make it easy to follow.

4 Submission

Please follow to the following submission guidelines:

- Your source code and report must be contributed in the form of a compressed file (.zip, .rar) and named according to the format `StudentID1_StudentID2_...`
- If the compressed file is larger than 25MB, upload it to Google Drive and share it via a link. **Absolutely no modifications are allowed after the deadline**.

Example details of the directory organization:

```
StudentID1_StudentID2_...
 |__Source
 |   |__main.py
 |   |__Map (folder for map data)
 |   |__README.txt (how to run source code)
 |   |__requirements.txt (libraries to be installed)
 |   |__...
 |__Report.pdf (included demo video URLs)
```

5 Notices

Please pay attention to the following notices:

- This is a **GROUP** assignment. Each group has 3 - 4 members.

  Groups with fewer or more members than the specified limit require lab instructor approval.
- Duration: about 3 weeks.
- AI tools are **not restricted**; however, students should use them wisely. Lab instructors have the right to conduct additional oral interviews with random groups to assess their knowledge of the project.
- Any form of plagiarism, dishonesty, or misconduct will result in a grade of zero for the course.

The end.