CMPT225, Fall 2026
Final Project
Due date: December 7, 2026, 23:59. No late submissions.

Instructions:

- You may do the project alone or in pairs. If you do the project in pairs you should write in the documentation who did each part. (It's ok if some parts were done together.)
- You need to submit a zip file called *rushhour.zip* to Coursys. The zip file will contain your solution and documentation of your project.
- The solution should contain the class **rushhour.Solver**. The main() method of **Solver** takes command line arguments with the names of the input file and the output file that will contain a solution. That is, we invoke the solver from command line as follows:
  java rushhour.Solver inputPath outputPath
- Place all your files in the zip file under the folder src. That is, your zip file must contain at least one file: **src/rushhour/Solver.java**. Your solution will most probably have more files.

Project Description:

- In this project you will write a solver for the Rush Hour game. Your input will be a file containing a rush hour puzzle. Your goal is to get the XX car to the exit on the right, and write the solution to the output file.
- The input files are provided with the assignment. There will be no other puzzles.
- For the exact format see the provided input and output files.
- The class *rushhourtesting.TestRushHour.java* is included for your convenience.
- If your algorithm doesn't find a solution to a puzzle within 30 seconds, the TAs will skip it and move on to the next puzzle.

To solve the puzzles you may use the graph exploration algorithms we discussed in class: BFS/DFS/A*. For A* you will need to think of different heuristics how to evaluate the distance from the current state to a solution.

*Some puzzles are hard, and you can get 100% on the project even if you don't solve everything.

References:

- You may use textbooks, wiki, stack overflow, geeksforgeeks, etc.
- If you do, specify the references in comments.
- **Using AI tools for solving the assignment is prohibited.**
- **If I suspect that you used AI tools to generate the code, I can ask you questions to see if you understand the code.**

Documentation:

I don't expect any specific format for documentation, you may use word/latex/plain text.

- Use plain English or diagrams or whatever you feel is right.
- Explain your choices of classes, data structures, algorithms, heuristics, etc.
- Was one of the heuristics always better? Or different heuristics solved different puzzles?
- Describe which parts were easier, which parts required more time.
- Describe what you wrote but then didn't include in the final project (e.g., switched from one data structure to another)

Good luck!

Here are some simple ideas for heuristics. You don't have to use these ideas, these are only suggestions.

1. *Remove cars from the main path:* Let the cost be the number of vehicles blocking the red car's path to the exit. It clearly underestimates the length of the solution, and when cost==0 the red car can exit.

2. *Remove cars from the main path:* Let the cost be the number of vehicles blocking the red car's path to the exit + an extra cost if these cars are blocked.

3. *Clear room in front of the exit*: For each square give a cost (the closer it is to exit, the higher the cost), and the total cost will be the sum of all costs. That is, if there are no cars near the exit, then the estimated cost is low.

4. *Count freedom of movement:* Count the number of cars that have a move. Similarly, count for each car how many moves it has.

   *Strictly speaking, 3 and 4 do not estimate the length of the solution, but they may be used as an indicator when combined with other heuristics.

5. A hybrid of the above by taking a (weighted) average.

More suggestions:

1. You may want to use hashCode inside your *.equals()* before actually comparing the entire board to save time.

2. For the open set you will probably want to use *java.util.PriorityQueue*.

3. For the closed set you will probably want to use *java.util.HashMap*.

4. These are only suggestions, and you should decide whether to use them or not depending on your implementation.