Lab 1: Code

Consult the CS 105 LEARN Site for lab due dates and standard lab instructions.

Rush Hour

Rush Hour is a puzzle game where players move a red car to an exit on the right side of a grid. The challenge is that there are other cars and trucks in the way. Players must move the other cars vertically or horizontally along spaces of the grid to make a path for the red car. You can only move vehicles forward or backward (not sideways) and they must always be on the board.

You can watch a short video demonstrating how to play and you can play a free knockoff, online version (sorry it's Flash with lots of ads).

In order to make a path for the red car to the exit, players must follow a series of steps to move the other vehicles. You should recognize this as an algorithm. You're going to write computer code to express algorithms that move the vehicles and solve a specific starting state of the game board.

A Simple Rush Hour Programming Language

In our first made-up programming language for playing Rush Hour, there are four different functions to move a vehicle by one square in the direction stated.
moveUp(✪);
moveDown(✪);
moveLeft(✪);
moveRight(✪);

When you write your code, you replace the symbol ✪ with the name of colour of the vehicle you are trying to move. Here are all the vehicle colour names that can be used:

RED
GREEN
ORANGE
TEAL
YELLOW
BLUE
PURPLE

You use the move functions and vehicle colour names to change the state of the game board. Let’s say your start with the example1 game board below. Let’s call its initial state **State A** (shown below) and let’s say you want to move the green car up one space to change the board to **State B**. You would write the following code in a small computer program to express this simple algorithm.

moveUp(GREEN);

Remember that computer code has a very particular syntax. The name of the function and the colour of the car must be an exact match, even the pattern of uppercase and lowercase letters.

Even though JavaScript is loose about semicolons, in CS 105 we require you to always add a semicolon at the end of each statement (which is usually the end of a line). This is considered good style by most coders, and not doing so can get you in trouble.
Illegal moves will cause errors. Just like in real life, vehicles can only move forward or backward based on the orientation of the vehicle in the grid. A vehicle that tries to move into a square occupied by another vehicle will crash. Also, vehicles must always stay within the board. This means that some function “calls” would result in an error depending on the game state. For example:

- Calling `moveUp(ORANGE)` when the game is in **State A** would cause a “vehicle can’t move sideways” error.
- Calling `moveRight(ORANGE)` when the game is in **State A** would cause a “vehicles must not collide” error.
- Calling `moveLeft(ORANGE)` when the game is in **State A** would cause a “vehicle must stay inside board” error.

**Call a function multiple times to move a vehicle multiple squares.** For example, to move the yellow truck down two squares, then we would write the following:

```java
moveDown(YELLOW);
moveDown(YELLOW);
```

This would change the game board from **State B** to **State C** (shown below).

![State B](image1.png) ![State C](image2.png)

Now that the green car and yellow truck are out of the way, and the game is in **State C**, you can move the red car through the exit to solve the puzzle and win the game.

```java
moveRight(RED);
moveRight(RED);
moveRight(RED);
```

Since the way was clear in **State C**, calling these functions will change the game to **State D** and solve the puzzle. **Note only the red car can exit, and only the front of the red car has to cross the exit.**
You combine the code from the three steps into a single algorithm to solve the puzzle beginning in State A. You can express your algorithm in a single program like this:

```javascript
moveUp(GREEN);
moveDown(YELLOW);
moveDown(YELLOW);
moveRight(RED);
moveRight(RED);
moveRight(RED);
```

Run a Rush Hour Program in Processing p5.js

We've created a p5.js sketch that lets you write and run Rush Hour programs like the one above: https://editor.p5js.org/cs105/sketches/ZTBDL_dYI

If you run that sketch, you'll see it solve the example1 board using the program we just explained above. You should see “SOLVED!” in the “Console” panel at the bottom. If your program has an error, the “Console” panel is also where you'll see the error message.
You can enter your own Rush Hour program by changing the code below the “solve by writing algorithm below” comment. Try changing this code to match the examples in the next section.

Remember, there can be only one Rush Hour program in the sketch at one time.

**More Examples**

*Alternate Solutions for the Example 1 Board*

There is more than one algorithm to solve the same `example1` game board above. For example, this is an alternative sequence of code that will also solve the puzzle and still finish in the same **State D** as before:

```
moveDown(YELLOW);
moveDown(YELLOW);
moveUp(GREEN);
moveRight(RED);
moveRight(RED);
moveRight(RED);
moveRight(RED);
```

Note that not all programs using these four functions are solutions, for example this will result in a crash when the red car attempts to move to the exit.
moveDown(YELLOW);
moveDown(YELLOW);
moveRight(RED);
moveRight(RED);
moveRight(RED);
moveRight(RED);
moveUp(GREEN);

You can write a program that solves the game starting in **State A** but finishes in a different final state:

moveDown(GREEN);
moveDown(GREEN);
moveDown(YELLOW);
moveDown(YELLOW);
moveRight(RED);
moveRight(RED);
moveRight(RED);
moveRight(RED);
Solutions for a different game board

Consider this new example2 game board:

You can tell the Processing Rush Hour program to use this game board by changing this line:

```java
    game("example1");
```

to this:

```java
    game("example2");
```

Below is one program that solves it:

```java
    moveLeft(RED);
    moveUp(GREEN);
    moveUp(GREEN);
    moveUp(GREEN);
    moveUp(GREEN);
    moveLeft(YELLOW);
    moveLeft(YELLOW);
    moveLeft(YELLOW);
    moveDown(ORANGE);
    moveRight(RED);
    moveRight(RED);
    moveRight(RED);
    moveRight(RED);
    moveRight(RED);
```

Learn to Design Programs on Paper and Trace Their Execution “in Your Head”

It's tempting to program by trial and error. You write a program by almost guessing at some functions and some statement order, then you run the program to see what happens, then you pseudo-randomly change something in your program without thinking too hard about what it might do, then you run it again to see what happens, and so on. The problem with this kind of approach is
that it's slow, it will result in inefficient messy solutions, and you'll never really understand how programs work.

It's important that you design your program on paper and then try to trace through the code “in your head”¹ before seeing what the computer will do. If you can read code and understand how it will run, the easier you will find programming and better your solutions will be.

To help you practice this skill, the Processing Rush Hour will not show how a program will run for certain game boards. This means you can’t rely on watching what the computer does to see if it finds a solution or if it ends in an error. Instead, it only displays a static game board (for your reference) and checks the syntax of your program. When you “run” the Processing sketch, nothing appears to happen.

Try this now by telling the Processing Rush Hour program to use game board example3 with:

    newBoard("example3");

The program below is almost a solution, but it has two errors. Trace through it “in your head” to find the two errors, then correct them so your program would find a solution. Remember, the Processing Rush Hour program won’t tell you if there are errors or if the solution worked for example3.

    moveRight(RED);
    moveLeft(PURPLE);
    moveDown(PURPLE);
    moveRight(RED);
    moveUp(ORANGE);
    moveRight(RED);
    moveUp(ORANGE);
    moveRight(RED);
    moveRight(RED);

Now complete the exercises starting on the next page.

¹ or on paper, on a whiteboard, etc.
Exercises

Now that you understand the game and the commands available, complete the following questions by writing code in the Processing Rush Hour sketch.

IMPORTANT: Start each part of each question by “Duplicating” this starting Rush Hour sketch:
https://editor.p5js.org/cs105/sketches/ZTBDL_dYl

ALSO IMPORTANT: After testing your solution in the online editor, you must copy and paste your solution for each part of each question to the Processing IDE and use it to save and prepare for submission, not in P5 Online Editor. If unsure about how to save using Processing IDE, take a look at How to Save and Submit document.

Remember, there will often be multiple algorithms that solve the puzzle. As an extra challenge, try to find the most efficient one with the fewest steps.

1 - Write a program to solve each game board below.

   a) Use game board: exercise_a

      NAME your sketch “lab-01-1a”

   b) Use game board: exercise_b

      NAME your sketch “lab-01-1b”
c) Use game board: exercise_c
   NOTE: this is a static game board, you need to “run the program in your head” to make sure it works.

   NAME your sketch “lab-01-1c”

   ![Game Board Image]

---

d) Use game board: exercise_d

   NAME your sketch “lab-01-1d”

   ![Game Board Image]
An Enhanced Rush Hour Language

In the Rush Hour language used in the previous question, moving a vehicle multiple spaces required calling the same function multiple times. It would be nicer if we could just tell the computer to move a vehicle a certain number of spaces.

In this question, we'll use a new version of our made-up Rushhour language that supports moving vehicles more than one space.

For example, instead of moving the red car to the right four spaces like this:

```
moveRight(RED);
moveRight(RED);
moveRight(RED);
moveRight(RED);
```

You can move it to the right four spaces with this single function call:

```
moveRight(RED, 4);
```

This is easier to read and easier to write.

Here's an example using this new language for the example1 game from the previous section:

```
moveUp(GREEN, 1);
moveDown(YELLOW, 2);
moveRight(RED, 3);
```
Here's another example program in this new language to solve the example2 game:

```plaintext
moveLeft(RED, 1);
moveUp(GREEN, 4);
moveLeft(YELLOW, 3);
moveDown(ORANGE, 1);
moveRight(RED, 5);
```

Exercises

2. Rewrite your code from parts a, b, c, and d in question 1 using this new language.

NAME your sketches: “lab-01-2a”, “lab-01-2b”, “lab-01-2c”, “lab-01-2d”

Submitting

Submit a single ZIP file called “lab-01.zip” containing all your sketches, see “How to Save and Submit” (on CS 105 LEARN Site).

It is your responsibility to submit to the correct files to the correct dropbox before the deadline. Otherwise you will receive a mark of 0.