CS 510: Introduction to Artificial Intelligence

Assignment 1

Rush Hour, Part I: State Representation and Move Generation

Rush Hour [https://www.thinkfun.com/products/rush-hour/] is a puzzle game that involves sliding "vehicles" around a grid to alleviate gridlock and free your "car" from the traffic jam. The real game looks like this:

In the game, each vehicle slides only in its normal direction of travel—that is, forward and backward, not side-to-side. The red car in the middle is your car, and your goal in the game is to free up space such that the red car can escape out of the opening on the right.

In this part, we will write the code needed to represent a single board state, and to compute possible next boards after moving one piece. In the next part, we will extend this code to search through a space of boards to find solutions to a given problem. The code for this assignment should be written in Python to run on tux.cs.drexel.edu.

Implementation Setup

The various parts of this assignment will require a shell script that can pass an argument to your code—thus allowing you to use **python** or **python3** while allowing us to be able to run your code with a consistent interface. However, **you may only use built-in standard libraries (e.g., math, random, etc.); you may NOT use any external libraries or packages** (if you have any questions at all about what is allowable, please email the instructor).

To this end, please create a shell script **run.sh** that calls your code and includes 2 command-line arguments that are passed to your code. For example, a shell script for **python3** might look as follows:

```
#!/bin/sh
if [ "$#" -gt 1 ]; then
    python3 rushhour.py "$1" "$2"
else
    python3 rushhour.py "$1"
fi
```

Of course, if you're using **python** instead of **python3**, you'll need to change the command used above.

Your code (in our example above, the python code in **rushhour.py**) will need to accept these two arguments and use them properly for each particular command. As you'll see in the sections below, running the code will have the general format:

```
sh run.sh <command> [<optional-argument>]
```

Again, this scheme will allow you to test your code thoroughly, and also allow us to test your code using a variety of arguments.

State Representation

For this assignment, you first need to create a representation for the state of the board. Let's assume that we can represent the board as a two-dimensional array, which we can draw in ASCII characters like this:

```
 ------
|  o aa|
|  o   |
|xxo
|ppp  q|
|     q|
|     q|
 ------
```

Each sequence of letters represents a vehicle, and the "xx" represents your car, which you are trying to get through the opening on the right.

Write a class **Board** that takes a string representation for a board with **arbitrary size** and stores the two-dimensional array for that board. We will assume that the input string will have the following format: (make sure to export errors in case the input string is wrong)

```
"  o aa|  o   |xxo   |ppp  q|     q|     q"
```

Each cell of the board contains a single letter or a blank character (' '), and the rows are separated by a vertical-bar character ('|'). You should also implement a **print()** function that can print the board in ASCII characters. It should be runnable from the command line with the "**print**" command as the first argument and the board as the second argument. If the board argument is not provided, please use the board above as the default. Here are two examples of running from the command line:

```
> sh run.sh print
 ------
|  o aa|
|  o   |
|xxo
|ppp  q|
|     q|
|     q|
 ------
> sh run.sh print "  ooo |ppp q |xx  qa|rrr qa|b c dd|b c ee"
 ------
|  ooo |
|ppp q |
|xx  qa
|rrr qa|
|b c dd|
|b c ee|
 ------
```

Please note that in the next sections, you will need a function that prints a list of boards horizontally—so you may want to implement a more general function with this in mind.

Identifying Solutions

Write a method that determines whether the given board is at the solution state. This should be very easy: if your car "**xx**" is touching the open space on the right, then you have reached the solution state. Then, augment the possible commands to accept a "**done**" command that prints "**True**" or "**False**" depending on whether the given board is at the solution state or not. For example:

```
> sh run.sh done
False
> sh run.sh done "  oaa |  o   |  o xx|  pppq|     q|     q"
True
```

Computing Next Boards

Your next task is compute "next" boards from a given board—that is, the set of possible board states that can be reached from the given board with a single movement of a single vehicle. For our purposes here, we will assume that a single movement can be a movement of a single vehicle across multiple spaces: for example, if a car can be moved 1, 2, or 3 spaces in a particular direction, each of these movements should be considered a possible next board.

In implementing this functionality, within your Board class, you are required to include a function

```
next_for_car(self, car)
```

that returns the next boards when trying to move the given car, where the **car** argument is the character representing that car (e.g., 'x', 'a', etc). Then, you can implement a function **next()** that uses your **next_for_car()** function to find all possible next boards for all cars on the board. You will also need a **clone()** method that clones the current board so that you can move a piece on a cloned board without affecting the original board.

Below is the desired output for our default board:

```
> sh run.sh next
 ------   ------   ------   ------   ------
|  oaa | |  o aa| |  o aa| |  o aa| |  o aa|
|  o   | |  o   | |  o   | |  o   | |  o  q|
|xxo     |xxo     |xxo     |xxo  q  |xxo  q
|ppp  q| | ppp q| |  pppq| |ppp  q| |ppp  q|
|     q| |     q| |     q| |     q| |      |
|     q| |     q| |     q| |      | |      |
 ------   ------   ------   ------   ------
```

Looking at this example, the first board represents the board state after moving the "**aa**" car to the left; the second board after moving "**ppp**" one space to the right; the third board after moving "**ppp**" two spaces to the right; and so on.

Here is another example:

```
> sh run.sh next "  ooo |ppp q |xx  qa|rrr qa|b c dd|b c ee"
 ------   ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
|   ooo| | ooo  | |ooo   | |  ooo | |  ooo | |  ooo | |  ooo | |  ooo | |  oooa| |  ooo | |  ooo |
|ppp q | |ppp q | |ppp q | | pppq | |ppp q | |ppp q | |ppp q | |ppp qa| |ppp qa| |ppp q | |ppp q |
|xx  qa  |xx  qa  |xx  qa  |xx  qa  | xx qa  |  xxqa  |xx  qa  |xx  qa  |xx  q   |xx  qa  |xx  qa
|rrr qa| |rrr qa| |rrr qa| |rrr qa| |rrr qa| |rrr qa| | rrrqa| |rrr q | |rrr q | |rrr qa| |rrr qa|
|b c dd| |b c dd| |b c dd| |b c dd| |b c dd| |b c dd| |b c dd| |b c dd| |b c dd| |b cdd | |b c dd|
|b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b c ee| |b cee |
 ------   ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
```

Academic Honesty

Please remember that you must write all the code for this (and all) assignments by yourself, on your own, without help from anyone except the course TA or instructor.

Submission

Remember that your code must run on **tux.cs.drexel.edu**—that's where we will run the code for testing and grading purposes. Code that doesn't compile or run there will receive a grade of zero.

For this assignment, you must submit:

- Your Python code for this assignment.
- Your **run.sh** shell script that can be run as noted in the examples.

Please use a compression utility to compress your files into a single ZIP file (NOT RAR, nor any other compression format). The final ZIP file must be submitted electronically using Blackboard—do not email your assignment to a TA or instructor! If you are having difficulty with your Blackboard account, you are responsible for resolving these problems with a TA or someone from IRT before the assignment it due. If you have any doubts, complete your work early so that someone can help you.