CS100J Assignment A5 Rat Race Spring 2005 Due: 23:59, Friday, 8 April

Preamble

In this handout, we explain a game that you will write, in two assignments. Then, we will explain what you have to do for this assignment. This assignment (together with the next) will give you practice using two dimensional arrays, writing loops and `if` statements, and working with `static` information. You will also see GUIS (graphical user interfaces) at work, and you will learn how a Java program "listens" for keystrokes and responds to them.

spend time reading this handout so that you thoroughly understand what we are asking for. Do this *before* you start programming! Take notes as you read.

If part of this handout is unclear, then please post a message to CS100J news group.

Overview of the game

This assignment has you write a two-player rat race game. Jen and Steve are rats caught in a maze, and they frantically run around and eat Brussels sprouts (please click on this link, and other such links that you see).

The maze

To the right is a sample maze. Jen (`J`) is in the upper-left corner, and Steve (`S`) is near the upper-right. `#` is a wall, `.` is a hallway, and `@` is a Brussels sprout .

```
#######
#J..S.#
#.###.#
#..@#.#
#@#.@.#
#######
```

The maze is given in a file, with one line of the file for each line of the maze. The maze must be rectangular. Jen and Steve must appear inside the maze. The outside edges of the maze must all be walls: you must not have a hallway, a Brussels sprout, or Jen or Steve on the outside edge.

Controls

The maze will be displayed in a `JFrame`, and the players will move the rats around using the keyboard. The controls to move Jen and Steve are shown to the right; both have the classic "inverted T" layout used in many games (like the four arrow keys on your keyboard). Here's what the keys look like:

```
 w       i
asd     jkl
```

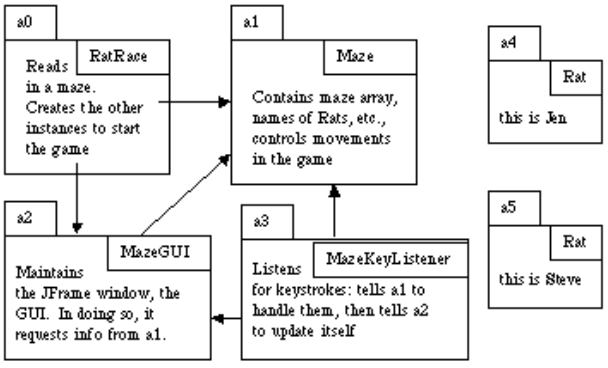
```
Jen          Steve
--------     --------
w: up        i: up
s: down      k: down
a: left      j: left
d: right     l: right
```

Game play

The players move Jen and Steve around the maze. They cannot move into walls. If the players try to move them into walls, nothing happens. Jen and Steve can occupy the same space, although if this happens only one of them will show on the map. (It doesn't matter which one.) When they move over a Brussels sprout they eat it and the `@` disappears. The game keeps track of how many Brussels sprouts each has eaten. When there are no Brussels sprouts left, the game ends with a message announcing how many Brussels sprouts each rat ate.

You can download a finished version of this game from the website or just click here: ratrace.jar (At some point, we will show you how you can make your own Java programs into jar files.) Here are two mazes that you can play with: bigmaze.txt littlemaze.txt Put these two mazes in a directory, along with ratrace.jar. When the game starts, it asks for a maze to play with, using a dialog window; use that dialog box to navigate to the appropriate directory and select one of the mazes.

The classes



The program uses five classes, as indicated in the diagram on the right. Class RatRace is used to read in the initial maze and create the necessary instances of the rest of them.

Class Maze is the major calculator. It maintains the maze, it keeps track of the two Rats and the number of sprouts still left, and it is the only one to actually change the maze because of keystrokes. It is the engine. It is important that Maze knows *nothing* about the GUI or the key listener (see below). All it does is keep track of the maze.

Class MazeGUI is an extension of JFrame. An instance of this class maintains the GUI. It has a procedure update(), which is called when the GUI has to be updated; in turn, this procedure calls Maze instance a1 for information it needs (e.g. how many Brusssel Sprouts are in the maze).

Class MazeKeyListener is a new kind of animal for you. It "listens" for keystrokes on the keyboard. When there is a keystroke, its method keyTyped(k) is called by the system. This method figures out what key was typed and calls a method of Maze instance a1 to handle the keystroke; after that, it calls a2.update() in order to change the GUI.

As you can see, each class, or instance of the class, has its own task to do. In this manner, each class can be written fairly easily.

When writing programs that use GUIS, one generally tries to separate the GUI maintenance from the calculation in the program, as we have done.

Skeletons for four of the files.

The skeletons have stubs for all the methods that you will have to write in this assignment or the next: RatRace.java Maze.java MazeGUI.java MazeKeyListener.java

Your assignment A5

Most of what we have told you thus far is just background, giving you a taste for what is to come, giving you a little idea about how GUI programs are put together. For the rest of this assignment A4, put most of it out of your minds, for we have *two* relatively simple tasks for you to complete, which have nothing to do with the GUI. First, you will write class Rat. Second, you will write the method that reads in a maze from a file and creates a two-dimensional array. Reading from a file is covered on Lab on 29-30 April.

Task 1

Write class Rat. You figure out what fields it needs, based on the methods that it requires:

Rat()	Constructor: a rat at position (0,0), who has eaten no brussel sprouts
getRow()	= the row number in which this rat currently is
getCol()	= the column number in which this rat current is
getNumSprouts()	= number of brussel sprouts this rat has eaten
move(int r, int c)	Move this rat to row r column c of the maze (this a procedure)
eatSprout()	Register that this rat has eaten another sprout (this a procedure)

Task 2

Write (and test thoroughly) function getMap(String) of class RatRace. This method is supposed to read in a maze from a file and return a two-dimensional char array that contains it.

The array cannot be created until the number of rows is known. We suggest that you use a temporary Vector, as follows. Read the lines of the file one by one, adding them to the Vector. Once *one* line has been read, the number of columns is known. Once *all* the lines have been read, the number of rows is known. So, now, create the two-dimensional array. Then, process the elements of the Vector, one at a time, placing the characters in it into the appropriate positions of the two-dimensional array.

Note: A suitable method to obtain a buffered reader is already in class RatRace, for you to use.

What to submit

On the CMS, submit your files Rat.java and RatRace.java by midnight, Friday, 8 April.