# CS100J A6 Eating Brussel Sprouts Fall 2006 Due: 23:59, Tue, 21 Nov


## Preamble

We explain a game that you will write. Then, we explain what you have to do for this assignment, which gives you practice using two-dimensional arrays, loops, and if-statements and working with `static` information. You will also see GUIS (graphical user interfaces) at work, you will see how a program that uses a GUI can be structured, 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.

## Overview of the game

You will write a two-player game. It is not the game called Sprouts (click on such links, please) that was created by John Conway. Jack and Sue are caught in a maze, and they frantically run around and eat Brussels sprouts.

### The maze

To the right is a sample maze. Jack (`J`) is in the upper-left corner, and Sue (`S`) is near the upper-right. `*` is an inner wall, `.` is a hallway, - is a horizontal outer wall, | is a vertical outer wall, 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 is rectangular. Jack and Sue must appear inside the maze. The outside edges of the maze are walls, as indicated above: a hallway, a Brussels sprout, Jack, or Sue may not be on the outside edge.

### Controls

The maze is displayed in a `JFrame`, and the players move using the keyboard. The keys to move Jack and Sue are shown to the right (like the four arrow keys on your keyboard); both have the "inverted T" layout, which is used in many games. The keys look like this:

```
 s       i
zxc     jkl
```

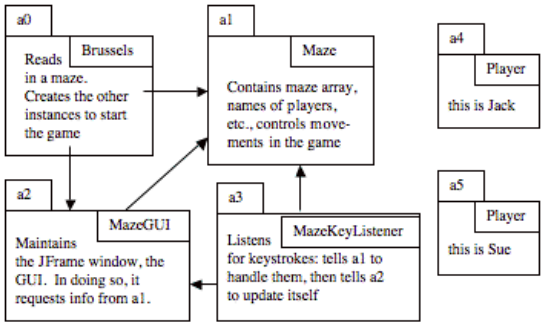
```
Jack          Sue
--------      --------
s: up         i: up
x: down       k: down
z: left       j: left
c: right      l: right
```

### Game play

The players, Jack and Sue, move around the maze. They cannot move into walls. If they try to move into walls, nothing happens. Jack and Sue can occupy the same space, although if this happens only Sue will show on the map (Jack is a gentleman). 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 a message announces how many Brussels sprouts each player ate and who won.

You can download a finished version of this game from the website or just click here: brussels.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 brussels.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 `Brussels` is used to read in the initial maze and create the necessary instances of the rest of the classes.

Class `Maze` is the major calculator. It maintains the maze, it keeps track of the two players 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 Brussel Sprouts are in the maze).

Class `MazeKeyListener` "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, performs its own task. 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.

### Files to download and keletons for four classes

Download skeleton.zip and unzip it to get (1) a jar file of the game that you can play, (2) two mazes, and (3) skeletons for four classes: Brussels.java, Maze.java, MazeGUI.java, and MazeKeyListener.java. Put them all in the same directory.

### Your assignment

Most of what we have told you thus far is just background, giving you a taste for what is to come and giving you a little idea about how GUI programs are put together. We now list the tasks to do for this assignment, in order.

### Task 1

Write class `Player`, an instance of which represents a player. The other classes won't compile until this class is written, so don't open the other classes in DrJava until this is written. You figure out what fields it needs, based on the methods that it requires. Note that an instance has no knowledge of the maze itself. It just keeps track of where the player is and how many Brussel Sprouts the player has eaten.

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

### Task 2

Write (and test thoroughly) function `getMap(String)` of class `Brussels`. This method reads in a maze from a file and returns a two-dimensional char array that contains it. Here are some points to consider.

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.

Notes: A suitable method to obtain a buffered reader is already in class `Brussels`, for you to use. To help you check `getMap` out, we have included a method `Brussels.toString`.

### Task 3

Your next task is to write and test all the methods of class `Maze`. As mentioned above, class `Maze` contains the methods that manipulate the internal representation of the maze, including the positions of the two players. Besides the constructor, there are 6 other methods bodies to write. Most of these are simple.

Important point: Note that parameter `m` of the constructor is a rectangular array that contains `'S'` and `'J'` to mark the position of the two players. However, when this maze is stored in field `b`, the `'S'` and `'J'` should NOT be put in `b` --they should be replaced by `HALL`. Instead, the positions of the players are given by the two Player objects `JACK` and `SUE`.

Important point. Do NOT use the character constants `"."`, `"@"`, etc. in writing the method bodies. Instead, rely on the static constants `HALL`, `SPROUT`, etc. The same goes for directions. Don't use integers like -1 and +1 to indicate directions. Instead, use the static constants `LEFT`, `RIGHT`, `NOCHANGE`, `UP`, and `DOWN`. Points will be deducted if you don't follow these rules.

### Task 4

You last task is to write the body of method `keyTyped` of class `MazeKeyListener`. This method is called when the `JFrame` window has the focus and a key is typed. This method should determine what key was typed and perform appropriately. If it is one of the keys `szxcijkl` that are part of the game, the method should call `maze.move` appropriately. Note that this method has access to the two players (using `maze.JACK` and `maze.SUE`) and also to the static constants of `Maze`, like `Maze.RIGHT`. It should make use of the constants and not use integer constants directly, like -1 or 0 or 1.

Once this method is working properly, you will be able to play the game with your friends. Have a good time with it.

### What to submit

On the CMS, submit your files Player.java and Brussels.java by midnight, 21 November.