Recursive BackTracking

One of the problems from the current project asks you to complete method `findExit()` in class `ObstacleCourse`. A `findExit()` message should find a path to an exit and set the two instance variables `foundRow` and `foundCol` indicating the location of the found exit. In this case, `findExit()` should return true. The exit need not be found in the shortest path. Any exit will do.

If `findExit()` can not locate an exit to the fact that there is none reachable, `findExit()` should return false. In this case `getExitRow()` and `getExitColumn()` still return -1.

When `findExit()` is called after constructing an `ObstacleCourse` object with a start row of 5 and a start column of 7:

```
++ +++ ++++++++ +
+  +  +  ++  ++
+ + ++++++++  +
+ + + ++++++++ +
++++++   +  ++  +
+++++++ +++  ++  +
+++++++ +  ++  +
+++++++ ++++++++ +
```

After calling `findExit()`:
```
++ +++O ++++++++ +
+   +  +  ++  ++
+ O  O O O+++++++ +
+ +O+++....+++ +
+ 0   0 0 O+++....+++ +
++++++O +O+++O+++O+++ +
+++++++O+++O+++O+++ +
+++++++ +O+++O+++O+++ +
```

When `findExit()` is called after constructing an `ObstacleCourse` object with a start row of 3 and a start column of 17:

```
++ +++ ++++++++ +
+  +  +  ++  ++
+ + ++++++++  +
+ + ++++++++ +
++++++   +  ++  +
+++++++ +++  ++  +
+++++++ +  ++  +
+++++++ ++++++++ +
```

After calling `findExit()`:
```
++ +++ ++++++++ +
+  +  +  ++  ++
+ + ++++++++  +
+ + ++++++++ +
++++++   +  ++  +
+++++++ +++  ++  +
+++++++ +  ++  +
+++++++ ++++++++ +
```

One algorithm to escape to the first exit found in the obstacle course is shown in the context of part of the class (next page). Use this algorithm. If you try another, don't ask for help.

```java
public class ObstacleCourse {
    private char[][] course;
    private int sRow;
    private int sCol;
    private int foundRow;
    private int foundCol;

    /**
     * Initializes the 2d char array course.
     */
```
public ObstacleCourse(int startRow, int startCol, char[][] grid) {
    course = grid;
    sRow = startRow;
    sCol = startCol;
}

// .... code removed to save space

/**
* This public method call the private helper provided as a stub below
*/
public boolean findTheExit() {
    return findTheExit(getStartRow(), getStartColumn());
}

/**
* This recursive backtracking method exhaustively searches the maze for an
* exit. It returns true if at the start row and start column set in the
* constructor an exit was found.
*
* This method must also set these two instance variables when the exit is
* found foundRow and foundCol = -1;
*
* Return false if an exit cannot be found from the start position.
*/
public boolean findTheExit(int row, int col) {
    // TODO: Complete this method
    boolean escaped = false;
    if (course[row][col] is a blank space {
        set current location to ';
        if on the border, (a base case) {
            set escaped = true
            set foundRow to the current row
            set foundCol to the current col
        } else {
            let escaped = (RP1) success of escaping below
            if still not escaped
                let escaped = (RP2) success of escaping right
                if still not escaped
                    let escaped = (RP3) success of escaping above
                    if still not escaped
                        let escaped = (RP4) success of escaping left
        }
        if escaped
            let current location = 'O'
    }
    return escaped;
}

// end findTheExit