**Lab 8: The A-Mazing DS4 Race - Part 2**

# **Objectives:**

* Practice top-down program design, problem-solving in C
* Work with 2-D arrays
* Develop skills in writing loops and solving problems in loops

# **Starting Point:**

* The code that you wrote in part 1

# **Turn-in:**

# Single lab report in PDF format containing answers to questions in both part 1 and 2. Include your source code.

* Demonstrate your working code to an undergraduate TA

# **Process:**

## **Problem:**

This week, we will add to the program from part 1. Currently, your avatar character should fall down the screen on its own and you should be able to control your avatar’s left and right movements. Now, we will be adding obstacles by randomly generating a “maze” which the avatar has to navigate through to get to the bottom of the screen.

## **Criteria:**

* Maze is randomly generated with difficulty set with a command line argument
* Avatar doesn’t move through walls, and doesn’t go out of bounds of maze
* Prints a winning message once avatar successfully reaches the bottom
* Prints losing message when avatar has absolutely nowhere to move
* **Optional Bonus:** print losing message when avatar can move left or right, but not down

## **Design:**

### Part C: Develop the Maze

You will generate a random maze of characters in a two-dimensional array. This will be the same maze that you output to the screen. Using good modular programming techniques, generate the maze and display it on the screen.

The **rand()** function in **<stdlib.h>** will be very useful, as it will return a new random int every time it is called that is from 0 to a very large number. HINT: Taking **rand % 100** may be useful. Don’t forget to call **srand()** as well.

To allow for differing levels of difficulty, your code will be given a difficulty value of 0-100 on the command line. This difficulty is the probability in percent that a given space of the maze will be an obstacle. Hence, difficulty zero indicates a blank maze, while 50 indicates that roughly half of the characters will be blank. A difficulty of 25 should have a maze that is approximately 25% full.

### Part D: Avatar - Maze Interaction

With the maze now in place, implement checking avatar movement to make sure it is not possible to move through walls and that the avatar cannot move outside the bounds of the maze. You may want to check the position of the avatar against where it is attempting to move and set the next position accordingly.

When the avatar reaches the bottom of the maze successfully, the program should print out a message indicating the player has won after exiting ncurses.

We also want to print out a losing message for when the player is stuck and can no longer move down, left, or right.

For example, the avatar can no longer move if in position something like this:

\*A\*

\*

### 

### 

### Bonus (+10pts)

In addition to printing out a losing message for when the avatar has no possible movements, also check for when the avatar can move left and right, but can has no possible way to move down.

For example, if the avatar is in a position like this:

\* A \*

\*\*\*\*\*\*\*

Left and right movement is possible, but downward movement is not.

Note: The large bucket problem does NOT expect you to solve cases described in the image below. That would be too complicated to write.

**\* A**

**\*\*\*\* \*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\***

**When you have your program working, have your undergraduate TA check off your source code and include a copy of it in your lab report.**

## **Questions**

1. Describe how you checked if the avatar could safely move down, and go left/right.
2. Describe what was necessary to check for the player losing the game..