Assignment 1: a maze game

Introduction

This assignment requires you to develop a program where a user navigates through a maze collecting potions and tries to exit the maze.

You are provided with start code (`maze_gen.c`) that generates the maze. You have to implement code to

- call a function from `maze_gen.c` that generates the maze
- read the user's keyboard movement commands
- keep track of the user's position and movement through the maze.

The maze

The `maze_gen.c` start code is provided. This code generates random mazes.

A maze is a rectangular grid of cells separated by walls

- the walls are represented with the 'w' character
- cells and the passageways between them are represented with the space character
- cells are square and their size (in number of characters) is user-specified; having larger cells makes it easier to see the way through the maze
- the maze is surrounded by walls on all sides of the rectangular grid
- there are two openings in the outside walls, one of the left side the other on the right side of the rectangle; these should be used as entrance and exit.

The exit of the maze is blocked by a wizard that will ask if the user has collected all the potions

- potions are scattered around the maze and marked with the '#' character
- only when the user has all the potions (3) they can exit the maze.

Requirements

Implement a program that allows a user to navigate through the maze. Start from the provided code and implement the functionality listed below.

In `maze_gen.c` the function that generates the maze is

```
struct maze generate_maze(unsigned int width, unsigned int height, unsigned int cell_size, int rand_seed)
```

The `generate_maze` function takes as input the following parameters:

- width of the maze in number of cells
- height of the maze in number of cells
- the size of the cells, which should be an odd number such as 1,3, or 5

- a random seed that controls the configuration of the maze; using the same random seed generates the same maze.

The function returns a variable of type `struct maze` that has the following fields:

- **a**, a 2D char array that contains the representation of the maze as described above
- **w**, the width of the array (number of columns)
- **h**, the height of the array (number of rows).

Functionality

- start by reading in the parameters for generating the maze
  - width, height of the maze (these are in maze cells, not array elements)
  - cell size: should be an odd number, determines the number of spaces used in the array to mark a cell; for example, if the cell size is 3 then cells will be 3x3 spaces; walls are only 1 character wide (or tall); this is only important when outputing as smaller cell sizes might make it more difficult to see the paths inside the maze
  - random seed allows to create many different maze configurations and to recreate the same one (if the same seed is used)
  - all these should be provided as parameters to the maze generation function
- generate the maze with the provided function
- find a point of entry into the maze (it can be on the left or right side of the array)
- use any characters (e.g. wasd) to allow the user to navigate up, down, left, right
- restrict the user's movement according to the maze configuration, that is, do not allow the user to go through walls
- keep track of the user's location inside the maze
- allow the user to pick up the potions and keep track of how many they have
- block the user from exiting the maze unless they have all potions
- congratulate the user when they exit the maze
- write a method to output (print to console) the maze and the position of the user in the maze
- additional functionality: implement a "fog" feature where the user can only see several cells around it
  - let the user specify the radius of the fog (can be zero, in which case the entire maze is output).

Hints

- use **scanf** to read data from the user and **getchar** to read the movement commands
- use a **switch** statement to process the movement commands
- the array storing the maze
  - the 2D array storing the maze is a matrix with *height* rows and *width* columns
  - a position in the maze can be represented with an x and a y, where the x represents the column number and y represents the row number.

Drawing the maze

To output the maze you can use printf. However this means you will have a new maze drawn below the previous one with each user move. What you want is to draw the maze over the previous one (like clearing the screen and drawing again).

For those using Linux or Mac you can use the **ncurses** library. Some tutorials are available:

- http://www.cs.ukzn.ac.za/~hughm/os/notes/ncurses.html
- https://www.linuxjournal.com/content/getting-started-ncurses

For Windows the **Console API** might work, although I don't have experience there:

- https://docs.microsoft.com/en-us/windows/console/

This is just for fun and will not be graded.

Marking scheme

- declaring the necessary variables: 5
- main function: 5
- calling the maze generation function and obtaining the maze: 5
- read the maze width, height, cell size, etc.: 5
- find the entry into the maze: 10
- read the user's movement command: 5
- process the movement command: 5
  - with "if" statements you get 1
  - with a "switch" statement you get 5
- check the validity of a move: 10
  - prevent moving through walls
  - prevent going outside the maze
- pick up potions: 5
  - remove from maze once picked up
  - keep track of the number of potions the user has
- detect when the user tries to exit the maze: 5
  - block the user if they don't have enough potions
  - congratulate the user on successful escape
- output the maze, including the position of the user and the potions: 20
  - inline code
  - separate function
- implement the fog feature when outputting the maze: 20

Learning outcomes

- reading data from the user
  - formatted input
  - single character input
- parsing user input
- indexing 2D arrays
- basic struct.

Honor code

- it is ok to exchange ideas with colleagues
- it is not ok to share a solution or parts of a solution with colleagues
- it is not ok to copy solutions from other sources (e.g. Internet)
- we use automated plagiarism detection software to find indications of plagiarism
- plagiarism consequences range from zero marks on sections or the entire assignment to higher level investigation
- when several parties share a solution (or parts), all will be penalised, regardless of who was the source and who copied.