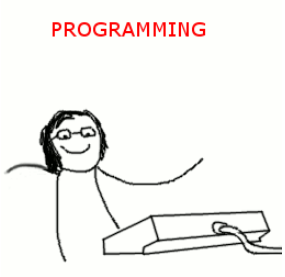


COMP2160

Programming Practices

ROASS Lectures Assignments Labs Programming Environment
Programming Standards Best Practices

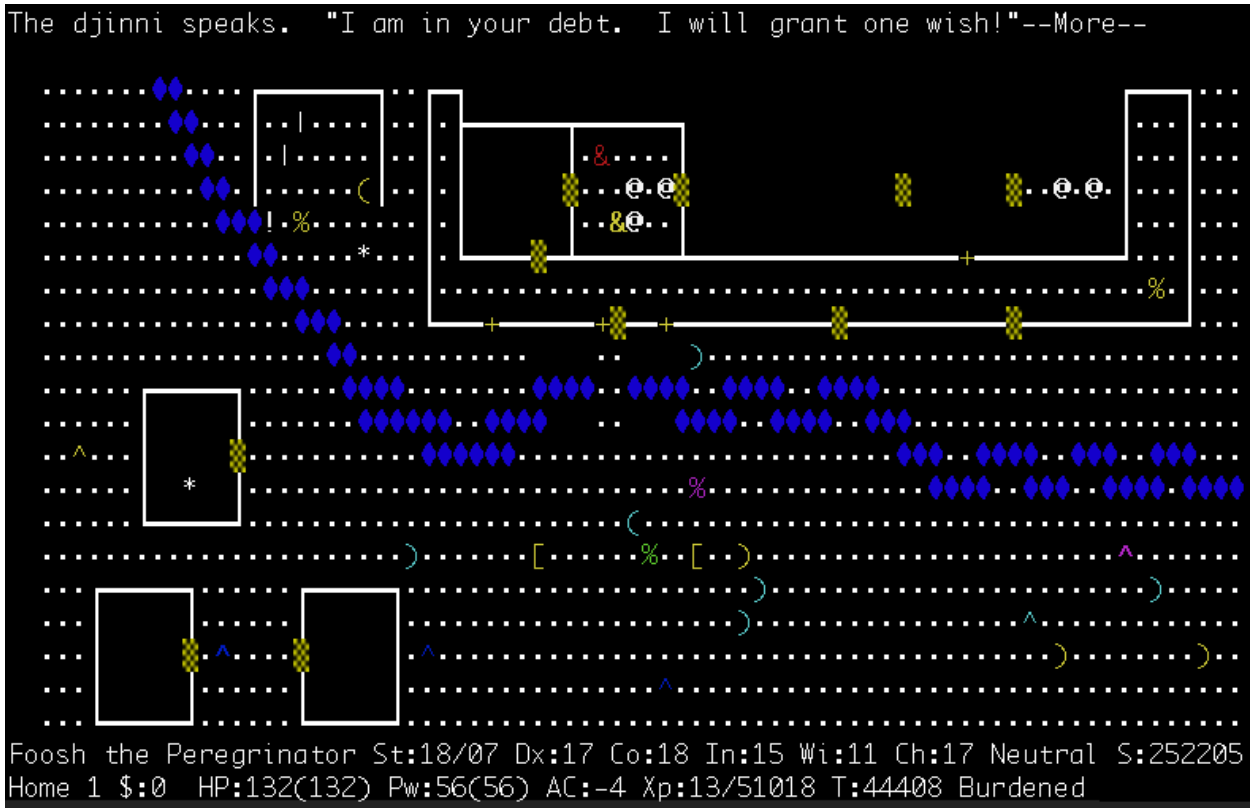
Instructor University of Manitoba Computer Science

Assignment 2: How Illuminating

Due Date: Monday, March 6th @ 11:59pm

Summary

Imagine, for just a second, that you've been hired at a company that's producing a Roguelike game. An excellent example of a Roguelike is Nethack:



Your task in this job is to implement a lighting feature for the game, where a light source (the character) will iteratively move around in a dungeon.

This assignment was inspired by an article by Jakub Wasilewsli on lighting effects on the PICO-8.

Table of Contents

Objectives

- Reading complex data
- Modelling complex data with `struct`, `enum`, and matrices
- Design by contract
- Using `git`

`git` usage

For this assignment you are **required** to use `git`. No marks will be awarded for *how* you use `git`, but marks *will* be deducted if you *do not* use `git`. Take the time to think carefully about when you should make commits to your repository so that you can revert if you get too far off the beaten path!

The Rules of Illumination

The dungeon consists of a two-dimensional matrix of tiles, where each tile is in one of four states:

1. The tile cannot be illuminated (it's a wall, or outside the dungeon).
2. The tile may be illuminated, and *is* currently illuminated.
3. The tile may be illuminated, but *is not* currently illuminated, and has *never* been illuminated.
4. The tile may be illuminated, and *is not* currently illuminated, but *was* illuminated at some point.

You must do a few different calculations to determine if a tile should be illuminated, and what level of illumination the tile should have:

1. Can this tile be illuminated at all?
2. How far away from the light source is the tile?
    - Lighting follows a gradient pattern (it gets darker as it gets farther away from the source). For your solution, the maximum lighting distance should be 3 units away from the light source.
    - Pythagoras and Euclid are going to help you out here!
3. If the tile is too far away from the light source, has the tile been illuminated in the past?
    - Discovered tiles should *always* remain discovered
4. Is there an *obstruction* between the light source and the tile?
    - Light can't go through walls!
    - You need to know about all the tiles *between* this tile and the light source. A linear equation will help you with this.

We haven't explicitly talked about two dimensional arrays in class, but you can find everything you'll need to know for this assignment here.

Implementation Requirements

All of the following conditions must be met by your solution.

1. Your program must read the name of the file used for input as the first argument to the program. The grader should be able to run your program as follows (you may name

your program *whatever* you want):

```
./illuminating illuminating.txt
```

- Hint: Use `fgetc` to read a single `char`, and `fscanf` to read numeric data from the file.
- Your program must read the initial state from the file passed as the first argument to the command. Here is a sample input file that you can use for testing. You will be expected to adequately test your program – the input file that the marker uses **will be different**.
- The input file will consist of several games. For each game, your program will:
  - Print out the game title as read in from the file.
  - Print the starting dungeon as read in.
  - Then, depending on whether or not your program was compiled with the `-DNDEBUG` flag:
    - `-DNDEBUG` is **not** set: Print *every* state of the dungeon.
    - `-DNDEBUG` is set: Print only the last 10 states of the dungeon.
  - All printing goes to standard output, and each dungeon must be preceded by a label indicating the move number (where the initial dungeon state is move 0). Here is a sample of the output expected.

Structured Input

The input will consist of starting dungeons for multiple games. Play through each game completely, in the order they appear in the file. The structure of the file is:

- The first line of the game starts with an asterisk and contains the game title.
- The second line of the game contains three numbers, separated by a space, which indicate the number of rows and columns for the 2D matrix, and the number of moves in the game.
- Then there is one line for each row in the matrix:
  - A blank character represents a tile that may be, but is not and has never been illuminated.
  - A `~` character represents a tile that cannot be illuminated (a wall).
  - A `.` character represents a tile that may beilluminated, is not currently illuminated, but has been illuminated in the past.
  - A `@` character represents the light source.
- Then there is a line with a series of moves that the light source makes, separated by the space character. A move is one of:
  - `^`, the light source moves up one space
  - `<`, the light source moves left one space
  - `>`, the light source moves right one space
  - `v`, the light source moves down one space
- The next game starts on the next line, until `EOF`

Structured Output

For each game, you must produce:

- The name of the dungeon, as read in from the file.
- A label indicating how many moves have been made.
- A complete output of the dungeon for each move (depending, of course, on the presence of `NDEBUG`).

When you're printing the dungeon:

- The dungeon should have a border (use `+`, `-`, and `|`).
- The light source should be represented by a `%` character.
- The lighting gradient around the light source should be one of three characters of lighting intensity: `'#'`, `'='`, `'-'`.
- The walls should be represented by a `!` character
- Tiles that are not currently illuminated but *have* been illuminated should be represented by the `,` (comma) character
- Tiles that have never been illuminated should be represented by the `' '` character (single whitespace).

Notes

- You're going to need to `#include <math.h>` for some of the calculations you're doing. On a Linux system you'll need to pass the `-lm` flag to `clang` to tell it to include the math library. You don't need to change how `clang` is run on macOS.
- Think carefully about how you should internally represent your state. You should **not** store everything as characters.
- While dynamic allocation is not strictly forbidden on this assignment, the implementation is simpler *without* it. If you choose to use dynamic allocation, you **must** ensure that you clean up all memory you allocate.

Evaluation

This assignment is worth 30 points.

Your solution must demonstrate good Design by Contract – you must make sure that you have defined pre and postconditions for each function that you implement, and include an invariant for the state of your application. (10 marks)

Your solution must demonstrate an appropriate level of functional decomposition. Focus on building *one* part of the assignment at a time, in stages. For example, build *just* the file input routine first. Take the time to **plan** what pieces you need to implement, and then implement each of those pieces one at a time. (5 marks)

Your solution must produce the correct output. You have been provided with *one* sample file that contains only *some* of the cases that you need to cover. You should create your own test files. Make sure that you cover **all** of the cases when you're building up your test file. The testing data will consist of 5 cases, each output is worth 3 marks. (15 marks)

Bonus

Printing out a series of matrices is rather boring; *animate* the output. You can either use the appropriate ASCII characters to move your cursor around (think `\r`, for example) *or* you may use the `ncurses` library.

You need to think about several things:

1. How long should each frame appear on the screen?
2. How do I actually implement the animation?
3. How do I *compile* this code?

The bonus is worth 5 marks. If you choose to implement the bonus, please indicate that you have done so in your `README`.

Warning about the bonus

The bonus part of this assignment is worth a trivial amount of marks compared to the actual assignment. **Do not** attempt the bonus if you're struggling with the implementation of the main part of the assignment.