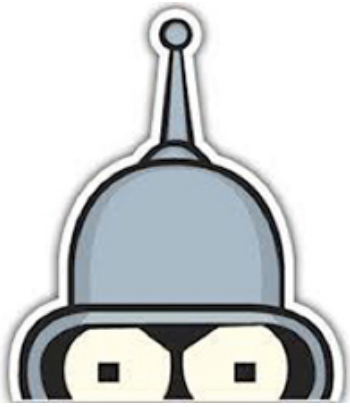
B2 - Elementary Programming in C

B-CPE-200

Bootstrap

Dante



Bootstrap

binary name: isPerfect
language: C
compilation: via Makefile, including re, clean and fclean rules

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

The goal of the Dante is to understand, generate and solve mazes. They come in all sizes and shapes, perfect and imperfect. During the project you will focus on rectangle mazes, but every maze you will encounter will come with its set of challenges.

The goal of this bootstrap is to make you understand and experiment with those nuances. Depending on your class' size and your pedagogical team's advice, you can choose either section of this bootstrap.

The first section is destined to take place with a group. You and your classmates will each research an algorithm on the topic of mazes and present it to your group in a short presentation.

The second section can be done alone. It will make you analyze mazes and determine their characteristics.

Section 1 - Research and Presentation

Step 0 - Organisation

Find a group of no more than 15 people that want to do the first part of the bootstrap with you, and a member of your pedagoglogical team to guide you.
You must work together with your group to compile a list of generation and resolution algorithms. Each of you will then be assigned one of them. Your guide will give you some time to prepare a presentation.
At the end of the alloted time, you will expose your research to the group in the form of a presentation.

Step 1 - Presentation

Here are examples of questions you should strive to answer with your presentation:
- How does the algorithm work?
- If Solver: When does it work best?
- If Generator: What will be the particularities of the generated maze?
- Are there cases where the algorithm is impractical or non-efficient?
- Do you have any ideas or suggestions pertaining to the implementation of the algorithm?
- Can you do a short demo of the algorithm working with a small maze? You will have to write this demo yourself.

Section 2 - Categorization

Step 0 - Reading and Writing

Create a new program that opens a file whose path is given as a parameter, stores the maze in a two-dimentionnal array, and prints it.

For a more step-by-step approach of this task, you can do the BSQ Bootstrap again

Step 1 - Perfection 1/2 : Can you visit every space?

Use the program from step0.
After printing the maze, replace the entrance with a new symbol that is not a star or an X. Then, iterate through your array multiple times. Everytime you find a star adjacent to your new symbol, replace it with a symbol.
Once you have gone through with enough iterations, check if you still find stars in your array. If you do, you can be sure of something: The maze was not perfect.
Print a version of the maze where the unreachable spaces are replaced with walls.

Is this the most efficient ways to do things? Definitly not. But it's fine for now, you will have a lot more time to optimize those opperations during your project

Step 2 - Perfection 2/2 : Are there any loops?

Use the technique from step1 but add a way to tell if you're looping around yourself. If you are, before printing the map, replace the issues of the loop (the single stars connected to the loop itself, from which you can enter or leave the loop) with a capital Y.
Again, if there is even a single loop, your maze is not perfect.

Step 3 - So you have a perfect maze. Now what?

Imagine a map that is all walls except for a single straight line from the entrance to the exit. Is it considered perfect? Sure. But is it really a maze? Not quite. Mazes should come with decisions to make, dead-ends and suspense.
Before printing your maze again, replace every dead-end with a capital D. Remember how many they were. It can be useful to evaluate yours and others mazes and their complexity.

Step 4 - One step left, one step right

Are dead-ends the best way to evaluate complexity? Imagine again the pseudo-maze from step3 that was just a straight line. What if you added dead ends alongside it, like small alcoves in the walls. Would it be a satisfying maze to solve? Definitly not.
Print your maze again, but this time, replace every remaining star with the number of options that your generator will have when arriving at this cell. A star that is part of a straight line gives no choice but to go forward. A star that has four other stars next to it will give you 3 different options.

Wait. The previous step didn't solve our straight-line-with-alcoves problem! Even if I choose the wrong option every time, the amount of backtracking I will have to do is very small. Could it be that the number of options is not the only thing that matters, but also the compounded number of options this choice will lead to? I need paper to draw a decision tree…