C++ Programming Methods

Assignment 3, Game of Life

Bas Terwijn <b.terwijn@uva.nl>

In this assignment we will create a Game of Life simulation as conceived by John Conway. Here we practice using arrays and classes.

Conway's Game of Life

Create a 2-dimensional 200x200 grid which represents the world where each cell is either dead or alive. At each time step apply the following rules that determine the status of each cell in the next time step. A cell's status in the next time step depends on the number of life cells in the current time step in its 8 surrounding neigbor cells.

- a life cell with less than 2 life neighbors dies
- a life cell with 2 or 3 life neighbors lives on
- a life cell with more than 3 life neighbors dies
- a dead cell with 3 life neighbors becomes alive

Viewport

Visualize the world in text by representing the status of each cell with a character. Only show around 80 by 40 character at a time, we will call this the “viewport”. As the viewport does not show the entire grid at once the user should be able to move the viewport up, down, left and right over the grid. Show the current viewport position (in coordinates) above the viewport. Make sure the viewport can not go completely off the grid and use a special character to represent the cells in the viewport that are not on the grid. In the next timestep simply visualize the world by printing it again (below the world of the previous time step).

Menu

Below the viewport show a one line textual menu with the following options. Make the menu intuitive to use and add basic checks on user input.

- Stop, exit from program
- Clean, set all cells to be dead
- Randomize, set status of each cell to random dead or alive

- One, move the grid one timestap forward
- Go, move the grid 100 timesteps forward showing the state at each step
- Move, opens another menu where the user can move the viewport
- Parameter, opens another menu where the user can set parameters:
    - horizontal step size of the viewport
    - vertical step size of the viewport
    - probability of a cell being alive when randomizing the grid
    - character representation of a life cell
    - character representation of a dead cell
    - optionally others
- File, allows user to load a predefined pattern of cells from a file

Pattern File

The pattern file “gliderGun.txt” is provided to test your implementation of the rules, it should over time produce a stream of moving objects. Search the web for this or other (simpler) Game of Life patterns in case you need them for testing. A pattern file too consists of a 2-dimensonal grid where dead cells are represented by a dot or whitespace ('.' or ' ') character and living cells are represented by any other character except a new-line character which represents the start of a new line of the pattern. When a pattern file is loaded its pattern overwrites cells in the grid. The user can move the viewport to where in the grid the user wants the pattern to be inserted. A pattern should be inserted at the top left corner of the viewport when loaded.

Random Number

For randomzing the world do not use the standard rand() function, instead create your own random number implementation based on an appropriate algorithm you should search on the web. Ofcourse reference the source. Also make sure it does not produce the same set of random numbers each time you run the program by allowing to set a random seed similar to what the srand() function allows you to do.

Design

Use a class each to implement the world, the menu, and your random number algorithm. All functions except the main() function should be methods of a class.

Submission

Submit your solution before the deadline to blackboard. Add to each solution:

- your name, student number and the name of the assignment

- references to the source of any algorithm or code that you did not create yourself

- operating system and compiler that was used to test the code