Third programming exercise: Life menu

The assignment

The idea of this assignment is to make a C++-program that lets the user play the game *Life* through a system of different menus. This means that the user can choose a set of different options to show the *Life* algorithm, these are the options of the program. There is one submenu which by itself also has options. The idea is that the whole menu consists of just one rule underneath the *Life* world. The user should be able to choose the options by entering the first letter of the word of the name of that option (followed by an enter). E.g. s or S to stop the program. Naturally, if the input is not clear or indecisive, the user is told so and the program keeps on running normally. Do not use *any* recursion! It's almost never the case in C++ that recursion leads to good programming solutions.
All user input should be checked for consistency. This means that, within reasonable margin, erroneous input should be accepted. Inputting x or & should for example be ignored. Numbers should also be read character by character (using cin.get(), if you use cin.get(), do not use cin >> anywhere!!! this will result in problems, cin.get everywhere). You should also make sure that the numbers that are given as input are not too large. Thus write a proper function 'readNumber' that transforms the read characters into a number (hint: ignore all enters before this number; process everything up till the next enter; and make a number out of this as closely fitting as possible to the input; for example abc123defg999h can be turned into 1239 if you require numbers smaller than 10000). Also make a function 'readOption' which reads one character and also processes enters properly. We can give 'reasonable limitations' to the user, for example that the number that are entered are 4 digits or smaller. The program should be able to thwart attempts to enter numbers with more than 4 digits properly. Giving letters instead of numbers also should not have any influence on the function. Keep it simple!

Life is a cellular automaton, invented in 1970 by John Horton Conway. Read Wikipedia or the ConwayLife website and Johan Bontes' implementation, with examples. In a 2-dimensional grid, about 1000x1000 large we start with a set number of living cells and dead cells. Each cell in the grid that has less than 2 or more than 3 neighbours dies (loneliness or overpopulation). With exactly two or three neighbours the cell survives. In a dead cell with exactly three living neighbours a new cell is born. This leads the system to the next generation of cells. Make sure this happens for every grid-cell at the same time (invariant of how you go through your array)!!!
The original game takes place on an infinite grid, but we will implement the finite variant only. To prevent any complex difficulties we agree that the edge of the world always consists of dead cells.

The user only sees a small portion of the world, this is named the 'view'. Every step the coordinates of the upper-left corner appear in the screen clearly. The height and width of the view are constants, for example 25 and 80. For the zealous students among you, you can create your own parameter-selection-options-sub-menu for this.

In the main menu we have the following options:

1. **Stop.**
2. **Clean. Clear the screen, all cells die.**
3. **Move move to left, right, up or down.**
4. **Parameters. This goes into the sub-menu to change the parameters.**
5. **File. Here a life configuration file is read and added to the world.**
6. **Random. Fill the world with random living and dead cells.**
7. **One. Calculate one evolution step.**
8. **Go. A whole series of evolutions is ran, and output without any enters (for those so inclined, you can make a nice animation of this (mind the outputting of the menu?).**

Everytime (perhaps not for the Go option) the view is shown, starting originally somewhere in the middle of the world. For the option Random you have to make your own random-generator. Look up on the internet how you make your own simple pseudorandom generator, and make sure it also accepts seeds. You can make this part as complicated or easy as you want!

There are several parameters that can be changed in the Parameter submenu:

1. **The translation (Move) step-size of the view. This parameter is used everytime the view is moved in one of the four directions. Make sure you also output the edge of the world when it is in view, and also make sure that it isn't crossed by the view!**
2. **The average percentage of cells that should be living after performing the Random selection**
3. **The two characters that are shown on screen for living and dead cells (make something nice as a default, preferably empty for dead cells, and something filling for living cells (ascii 219?).**

Choose reasonable starting values for the parameters. Also make sure you can return to the main screen from this menu.

For the File option the user can read a file (check if it exists!). This file is then put into the view (VIEW, not world), starting from the point in the top left (or one square right below that one if you think it's prettier). In the file spaces are dead cells. Line endings '\n' mark the transition from one row of the view to the next. '\r' is ignored. Every other character represents a living cell. Deze file represents a glider gun for example.

Other cells remain unchanged. The cells that are read can possibly extend to beyond the view itself (to the bottom right). Naturally, they can not be output beyond the world itself (check for this!). You can decide for yourself what should happen when a user provides a file that goes out of bounds.

For the option Go, after the set amount of iterations you have to show on screen two numbers indicating the two largest amounts of cells that were alive anywhere within the evolution sequence.

You should make a small class named World, with in it functions that represent each of the menu options. The parameters are typically member variables. You are not required to use header files yet, everything can be in one file.

Extra remarks±

Use proper member functions. For this exercise, NO functions should be longer than 20 lines (not even your main)! Every function should have proper comments indicating its purpose and use. Make sure you pass all the parameters to functions properly with exception of the member variables. Variables should be declared at the start of the main loop, or in the class, or at the start of a function. The only header files you should use are iostream, fstream, cstdlib and string. A rough indication of the length of the C++ program: 400 lines. Do remember the information block!