Computer Science 327
Project 2, Part b
C++ Programming Project
Cellular Automata and not so much Sand Painting

Notes

Some parts of this project contain mathematical formulas, but nothing more than you have seen in Math 165. There may be parts of this project that may be ambiguous (not on purpose), but it is your job to also clarify program specifications given in a natural language. Please read the assignments early and asks questions in class. Questions that lead to further specification or changes in the project specification will be posted in Piazza or in the announcements section of the Canvas course.

Introduction

Cellular automata have long been used to model a variety of real and imaginary systems, and a Google search on "cellular automata" will yield a plethora web sites dedicated to different forms of cellular automata. In this project, we will build a C++ program to create a rudimentary sand painting game.

The project is divided into two parts. For submitting part b, we will ask that you create a git tag for your submission. More details are given below.

Project 2 part b

For part b of the project, we will create a rudimentary control and gui for your CA simulation in C++. Below are the minimum requirements; however, you may add additional features if you like.

1. The simulation must run in the 800 by 600 pixel windows produced by the GraphicsServer.

2. The area where the CA is simulated is 600 by 600, with 200 pixels on the right side of the window used for user input.

3. The user interface must use a mouse to select options. You may include keyboard entries; however, they will not be used for grading

4. The CA rule is the game of life CA.

5. The user interface must allow the user to select the following functionality via the mouse: STEP, RUN, PAUSE, QUIT, RESET, LOAD, and RANDOMIZE. This functionality is described in detail below.

    a. STEP. Exceutes one step of the CA and displays the result.
    b. RUN. Continulously runs (steps) the CA at a rate of approximately 1 step every 100 ms. (You may add a speed control for variable speed control if you wish.)
    c. PAUSE. If the CA is in the run mode, the CA will stop running.
    d. QUIT. Terminites the and exits the program.
    e. RESET. Sets the state of the CA back to the initial state when it was loaded
    f. LOAD. Uses a "file browser" to select a file to load. (see additional messages in revised message format document)
    g. CLEAR. Sets all the cells to state 0.
    h. RANDOMIZE. Sets the cells in the CA to random initial states.

6. Create and use buttons and menus to select various options in the simulation. While you may use any look and feel you wish, simple graphics are shown below for reference You are not required to use this type of user interface, and it is just an example of what you could do. For example, the RUN/PAUSE button could be the same button and have the text in the button change depending on what state the simulation is in.

7. When a file is loaded, the cell size is set automatically based on the size of the CA in the same way as part 2A.

8. The user may select with buttons one of 3 predfined sizes of 600 by 600, 150 by 150, or 40 by 40. When a user selects these, the existing cells are cleared.

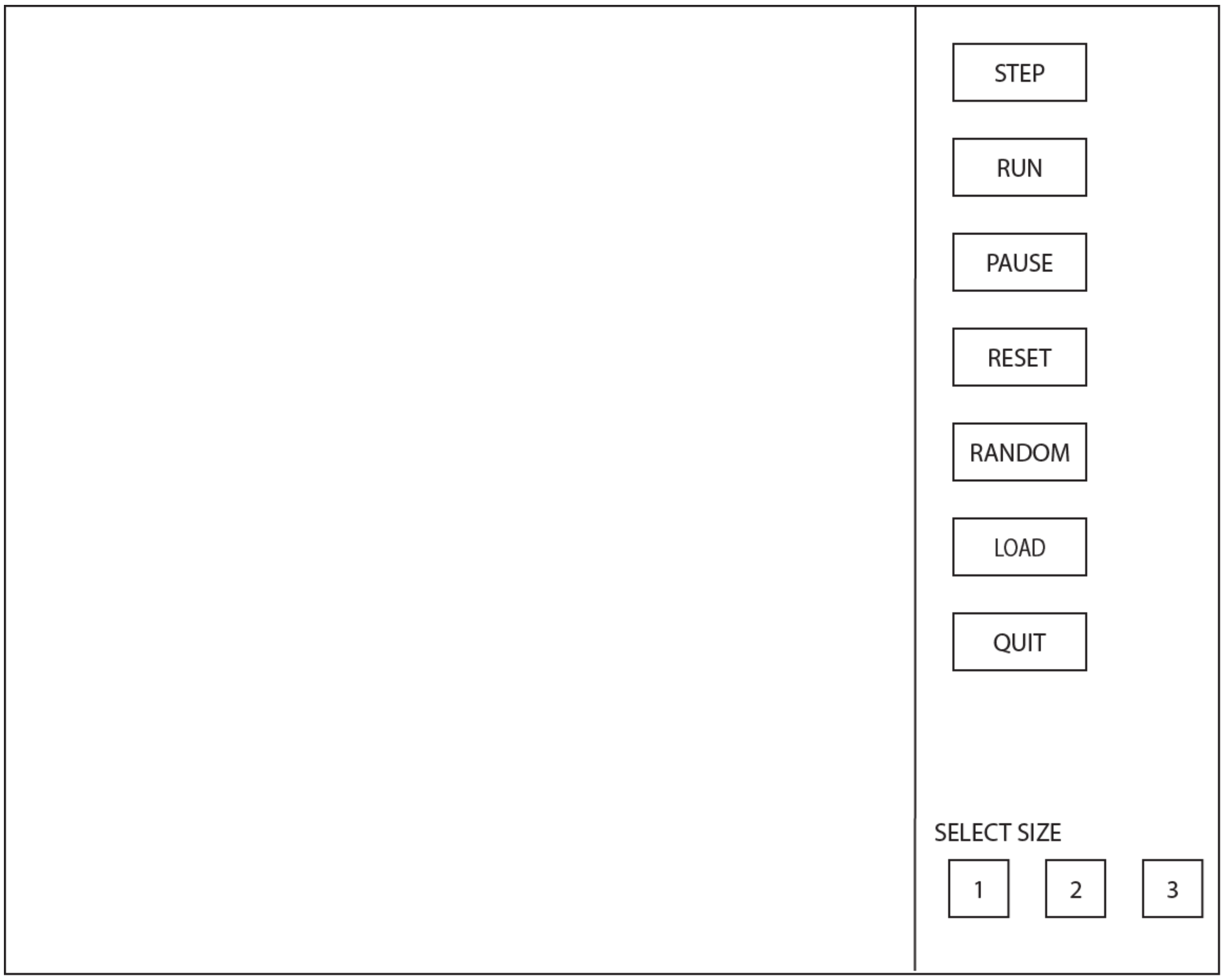
9. The user may toggle a cell value by clicking on it when the CA is paused OR running.

10. Use good OO practices for completing this project. You must use the CellularAutomaton and GraphicsClient object defined in part a. You may create additional classes and objects as needed. The main method that starts your program should reside in the file casimulator.cpp.

11. Write an appropriate make file.

12. Write appropropriate documentation.

13. Tag the repository for submission with the tag "proj2b"



Hints

The message format document has been updated to include mouse message and file selector messages.

You will need to pause your program for 100 ms. The easiest way to do this is with the following nanosleep c function. https://man7.org/linux/man-pages/man2/nanosleep.2.html

You can use the "read" function to read bytes sent to you from the socket. https://man7.org/linux/man-pages/man2/read.2.html

The problem with this is that read will clock until it has read the number of specified bytes. Since we want the CA to continue to simulate, we must ehck if there are enough bytes to read before doing the read. This can be accomplished with an ioctl function. For example,

```
int count;
ioctl(sockfd, FIONREAD, &count);
```

returns the number of bytes ready to read in the variable count.