Computer Science 327
Project 2, Part a
C++ Programming Project
Cellular Automata and 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 a, we will ask that you create a git tag for your submission. More details are given below.

Project 2 part a

1. For part a of the project, we will create a class that interfaces with the graphics server code via a network port. For message formats and how to talk to the graphics server, please see the associated documents and example code already posted. Note that there is no C++ socket interface, so you will have to use the C code to do this. For this first part, you will create the header file for the class.

   The filename of the header class is named "**GraphicsClient.h**" and the class name is "**GraphicsClient**". This class encapsulates drawing methods that can be made to a graphics server program via network messages. (described in another document) The idea is that when you create GraphicsClient object, it links the object to a specific window that is opened by the graphics server. If you create two of these obejcts, then two windows are created by the graphics server program. When an object is destroyed, then the network connection should be terminated and the appropriate actions will be taken on the graphics server side. Note that the server program may be given connections

to the same address and port, each connection gets a different socket to write data. If ;you need more information on how this works, please ask in class and look at the example code that is posted. The class must include the following methods.

a. A constructor with two parameters: GraphicsClient(std::string, int) where the first parameter is the URL to connect, e.g., “localhost” or “lamb.cs.iastate.edu” and the second parameter is the port number to connect.

b. A copy constructor that creates a new connection to the same address and port number.

c. A destructor that performs the appropriate operations to release all allocated resources.

d. An operator = method that closes the existing connection and creates a new connection with the parameters from the right-hand side.

e. setBackgroundColor(int , int, int) sets the background color of the associated display. The parameters are the red, green and then blue values respectively.

f. setDrawingColor(int, int, int) set the color that objects will be drawn at until another setDrawingColor call is made. The parameters are again, red, green and blue.

g. clear() clears the display to the background color.

h. setPixel(int, int, int, int, int) sets the pixel at the location given by the first two parameters to the color given by the last three parameters. The first two parameter are the location given by the x and then y coordinate. The last three parameters are the color given by red, green, and blue in that order.

i. drawRectangle(int, int, int, int) draws a rectangle at the specified coordinates given by the first two parameters of the specified size given by the last two parameters. The first two parameters are again the x and y coordinate in that order, and the last two parameters are the width and height, also given in that order.

j. fillRectangle(int, int, int, int) draws a filled rectangle at the position and size given by the parameters. The parameters are the same as the drawRectangle parameters.

k. clearRectangle(int, int, int, int) clears (sets the pixels to the background color) at the location and size specified by the parameters. These parameters are the same as the drawRectangle parameters.

l. drawOval(int, int, int, int) draws an oval at the specified location inscribed in a rectangle of the specified size. The parameters are the same as given in drawRectangle.

m. fillOval(int, int, int, int) is the same as the drawOval method except the oval is filled.

n. drawLine(int, int, int, int) draws a line starting a point 1 and ending at point 2. Point 1 is given by the first two parameters, x and y, in that order, and point 2 is given by the last two parameters, x followed by y.

o. drawstring(int, int, string) draws a string of characters on the display given by the last parameter at the position given by the first two parameters, x, y, in that order.

p. repaint() send the redraw (repaint) signal to the attached graphics server.

2. Implement the class defined in 1) in the file "**GraphicsClient.cpp**"**.**
3. Write a class definition (header file CellularAutomaton.h) that represents a 2D cellular automaton named "**CellularAutomaton**" The ideas behind this class is that it represents a 2DCA and therefore should have methods associated with loading and simulating a 2DCA. The class contains the following definitions.
    a. A constructor that takes two arguments. First, a C++ std::string argument that is a name of a file. Second, it gives an int parameter that is the quiescent state of the CA. It then reads a 2DCA in the same format as Project 1C.
    b. Proper copy constructor, assignment = , and destructors. The first two must perform deep copies.
    c. A **step** function that takes a single argument that is the rule function and performs one step of the 2DCA.
    d. A method that takes a single reference parameter to a GraphicsClient object and display the 2DCA on the attached graphics window associated with the GraphicsClient object. The size of the cell displayed on the window is dependent on the size of the 2DCA width and height according to the following table. Let m be the maximum value of the width and height of the 2DCA.

e.

m	Cell size in pixels	Cell gap in pixels
200 < m <= 600	1	0
100 < m <= 200	2	1
50 < m <= 100	4	1
1 < m <= 50	10	2

Note that the cell gap is the number of pixels between two adjacent cells.

4. Implement the CellularAutomaton class in the file named CellularAutomaton.cpp.

5. Write a main method that reads a whose name is given in the first parameter of the command line that contains a 2DCA definition. You must use the object defined above to do this. The program then displays the 2DCA in a display using the graphics server, and then waits for user input. Each time the user presses the return key, another step of the CA is displayed. If the user types any character followed by a return key, the simulation stops, the display is disconnected, and the program terminates.

6. Write an appropriate makefile.

7. Write appropriate documentation.

8. Tag the repository with the name "**proj2a**"