CS 430
Individual Homework 1
1000 points

Introduction

This is an individual homework that will require programming using Windows forms under the .NET framework. No code should be copied from any source and ideas from other sources must be cited appropriately.

Cellular Automata are a highly parallel model of computation and have been study since the early 1940s. A simple search of the Internet reveals numerous sites devoted to this model. At one time Cellular Automata were thought to be the way to model and simulate large dynamical systems such as weather or molecules. However, the technical aspects of programming a massively parallel system like CA lagged behind the ability to build hardware, and consequently have not been used as extensively as researchers thought. Recently, nano-technology has revised interest in this model and others for computing. In this assignment you will write a program that will simulate a cellular automaton as defined by a configuration file using a GUI interface.

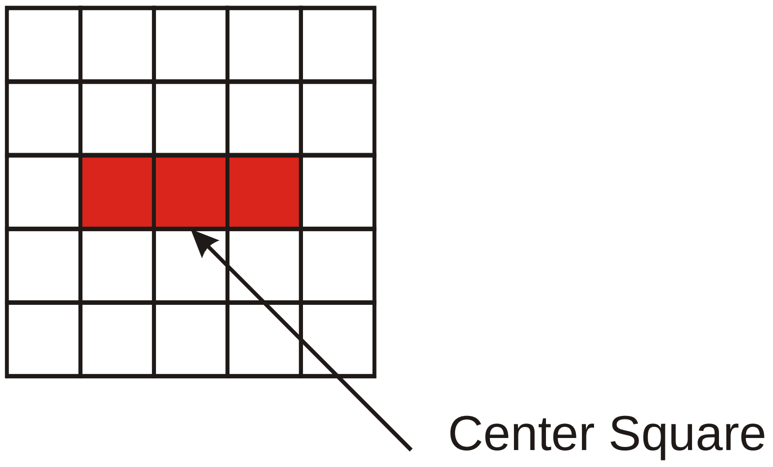
Cellular Automata

Cellular automata are defined on an infinite lattice such as the integers or the discrete plane. In general they can be of any dimension, but your program is only required to simulate and display cellular automata on the plane. This is also called a two-dimensional cellular automaton, or a 2CA.

To define a cellular automaton, we specify a finite neighborhood set, a finite set of states, and a function that takes the states of each neighbor to a single state. Given that that we are only concerned with 2CAs on the plane, the neighborhood set is relative to a center square and relative coordinates to the center square are given to define the neighborhood set. A couple of examples should make this clear.

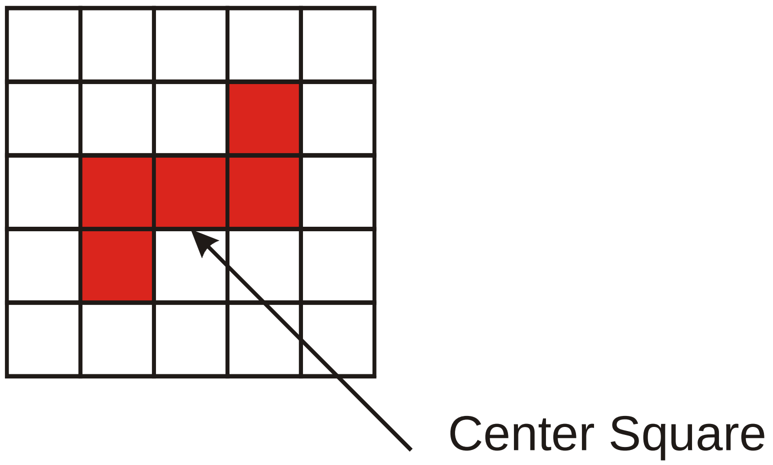
Example 1:

N = { (0, 0), (-1, 0), (1, 0)} corresponds to the following squares in the lattice, relative to the center square. (Note that since 0,0 is included in the neighborhood that the center square itself is included.)



Example 2:

N = { (-1, -1), (-1, 0), (0, 0), (1, 0), (1, 1) }



The semantics of a 2CA are quire simple. All squares on the lattice are initialized to one of the states in the 2CA definition. A finite set of squares (also called cells) can be initialized to any states, which give the initial configuration. For each cell on the lattice, consider the current state in each of the cells defined by the neighborhood. The next state of the center cell is then computed by the function for the 2CA. Note that each cell is updated synchronously, according to a clock.

Example:

N = { (-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1) }

(This is just the 3 by 3 square around a center square.)

S (The set of states) = { 0, 1 }

The next state function (delta) is given by

(0, 0, 0, 0, 0, 0, 0, 0, 0) -> 0

(0, 0, 0, 0, 0, 0, 0, 0, 1) -> 0

(0, 0, 0, 0, 0, 0, 0, 1, 0) -> 0

…

(1, 1, 1, 1, 1, 1, 1, 1, 1) -> 0

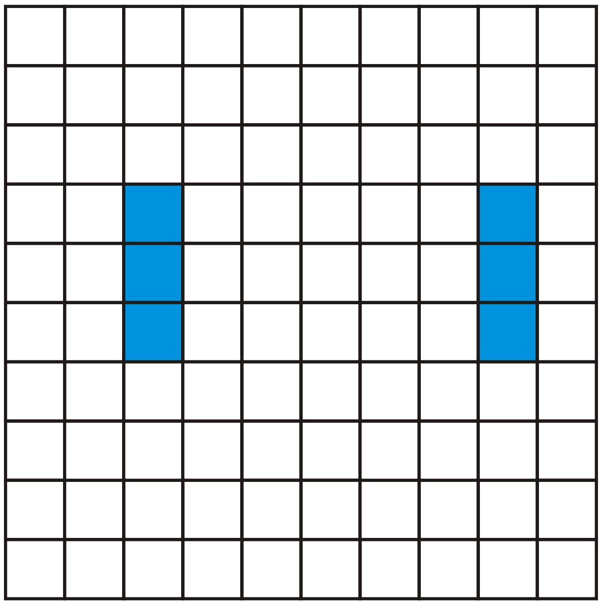
Of course it is tedious to define all 512 possibilities here, so for this example we will define the function as follows.

If the center cell is in state D and it is surrounded by exactly (3) three cells in state A, then the center cell becomes state A.

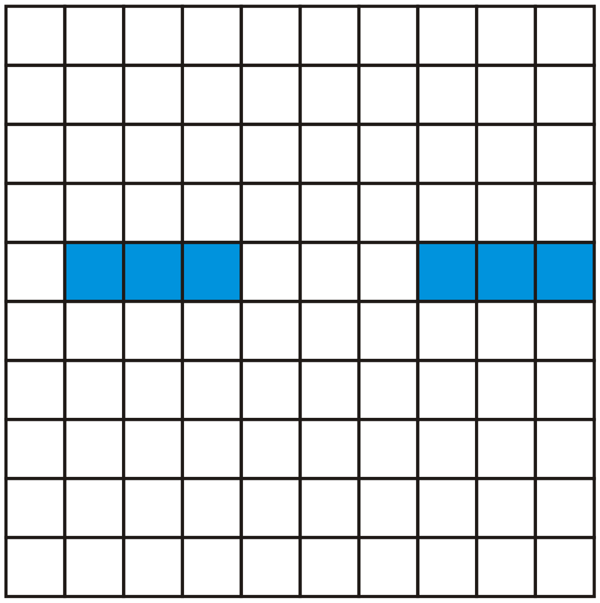
If the center cell is in state A and it is surrounded by either (2) two or (3) three cells in state A, then the center cell remains in state A.

Otherwise the center cell becomes state D.

For example, if the lattice contain cells in the following states at time 0. (Blue squares represent state 1 and white squares represent



At time 1 (one step later) the 2CA has the following configuration.



Homework Requirements

For this assignment, you are to design and implement a Windows application program using C# and .NET libraries that simulates a 2CA with the following requirements.

1. The simulation need only simulate a 2CA on a finite grid of 500 cells by 500 cells. Cells on the left and right edges are wrapped as well as cells on the top and bottom edges.

2. The CA definition is composed of C# code that defines the delta function, the number of states, and the default initial state of all cells. This information may be entered by the user directly into the client application, or by a file which the client reads. Proper error detection and reporting should be used.

3. The program must also allow the user to specify the initial state for specific cells. This may be either directly entered into the client, or read from a file by the client. The format for this file and data is your choice and must be properly documented.

4. The program shall allow the user to save the state of a cell lattice to a "cell initialization" type file. This file can then be reloaded later.

5. The program shall provide a client with graphical interface that allows the user to run, stop, and single step the simulation. In addition, a clear function initializes the CA uniformly to a user selected state.

6. The program shall provide a display area for the simulation that can display all 500 by 500 cells. In addition, the program shall provide at least 4 levels of zoom, with controls for zooming in and out. When a zoomed in so that the entire display cannot be shown at a single time, sliders shall be used to allow navigation.

7. The client configuration (also saved in a file) may specify an RGB color (using integer numbers) that is used to show on the display what state each cell is in. If a color is not specified in the CA definition file, then colors are chosen by the program.

8. The simulation of the 2CA is executed on a server machine using a separate C# program. C# remoting and messaging shall be used for communication between the client and server. The simulation must run as fast as possible and distributed processing used if possible.

9. When the "run" function" is initiated, the CA simulation runs as fast as possible, displaying each new configuration until either the "stop" or "step" function is initiated. Note that the logical method for implementing these functions is with a button, but this is not a requirement if you can think of a better GUI design. Activating the run function must not interfere with the responsiveness of activating or performing other functions such as stop.