Assignment 4

Assignment 4: Organic Growth Simulation

Background

In this assignment you will write a two-dimensional simulation of the stochastic growth of a colony of single-celled organisms called *B-cells*. This is a purely hypothetical problem, but it is similar to how one would model many different types of biological processes. Like many organisms, a B-cell will reproduce only under certain conditions on its immediate surroundings. There are two factors: how crowded it is, and whether or not there is a pathogen nearby. If it is too crowded, it dies off, and if it is not surrounded by enough of its kind, it will also die off. If there is a pathogen in a neighboring cell, it dies off regardless of how many B-cells are adjacent.

Simulation

These organisms do not live in an orthogonal world, but we will simulate their colony with one. We will model their world as a two-dimensional grid of rows and columns. The intersections of these are called ***cells***. A cell can be in one of three states:

- empty,
- inhabited by a B-cell, or
- inhabited by a pathogen.

A cell containing a B-cell will be called an ***active cell*** and one containing a pathogen will be called a ***threat cell***. Every cell has eight adjacent (neighboring) cells.

neighbor 1	neighbor 2	neighbor 3
neighbor 4	cell	neighbor 5
neighbor 6	neighbor 7	neighbor 8

These organisms do not synchronize their moments of procreation or death, but to model their behavior, we must. We will assume that time is divided into equal length steps, denoted 0,1,2, and so on, and that in each time step, the population changes as a result of births and deaths. The rules by which organisms are born, survive, or die are as follows:

- If at time t, any of its adjacent cells is a threat cell, it ceases to exist at time $t+1$.
- If at time t, an active cell has fewer than 2 neighbors, it ceases to exist at time $t+1$.
- If at time t, an active cell has more than 3 neighbors, it ceases to exist at time $t+1$.
- If at time t, an active cell has exactly 2 or 3 neighbors and no neighbor is a threat cell, it continues to exist at time $t+1$.
- If at time t, an empty cell has exactly 3 neighbors and no neighbor is a threat cell, at time $t+1$ it is active.

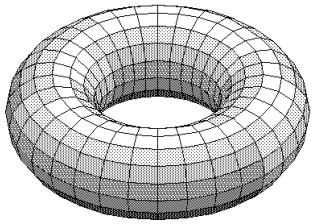

Figure 1: Surface on which B-cells live.

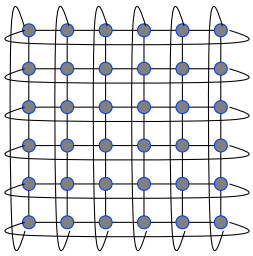

Figure 2: Rectangular grid representation of a torus.

- A threat cell created at time t lives for M time steps and dies at time $t + M$. M is a parameter of the simulation.

Pathogens come into existence randomly. At each time step t there is a probability $p_t$that T pathogens will enter the world, where T is a program parameter. The cells that they occupy are also random. The details will be explained below.

B-cells do not line on a planar surface; they live on a doughnut-shaped surface known mathematically as a torus. See Figure 1.

In the figure, the lines represent column and row edges. Notice that if we traveled up or down any column, we would return to the cell in which we started. Similarly, if we travel across any row, east or west, we would return to our starting cell. We can use a rectangular grid to represent a torus, as in Figure 2. It simply amounts to declaring that the top and bottom rows of the grid are adjacent and the left and right columns are adjacent.

For example, if we have a 100 by 100 array of cells, with columns $0, 1, 2, ., 99..$ and rows $0, 1, 2, ..., 99$ then we think of column 0 as following column 99, and row 0 as following row 99. Therefore, for example, `cell[0,0]` is adjacent to both `cell[0,99]` and `cell[99,0]`.

The initial state of the population will usually determine whether any organisms survive over time, or whether the population lives in perpetuity. Certain initial states may result in an infinite cyclic pattern, whereas others may result in the death of all organisms. In both cases it depends upon the probabilities associated with the formation of pathogens. If you play around with this you will find examples of both.

Program Requirements

You are to write a C or C++ simulation of the population model described above. The details follow.

The Terminal Window

The terminal window will be divided into rows and columns. The bottom row will be reserved for interaction with the user. All other rows and columns will represent cells. The program must detect the size of the

window when it starts and make sure that it creates a rectangular grid that maximizes its use of the window. The row just above the bottom row should be treated as adjacent to the top row, and the left and right sides of the window adjacent to each other.

If a cell is active, it will contain the symbol '`b`' to indicate a living B-cell. If a cell contains a pathogen, it should contain the symbol '`x`'. Otherwise there should be no symbol in it.

Initialization

When the program is started, it will read an initial state from a file named `.genesis` in the current working directory and use that as its initial state. The program will have a single, optional command line argument, which is the name of a file to use instead of the `.genesis` file. If the expected start-up file does not exist, the program will use `.genesis` instead. If neither exists, the program should exit with an error message.

A start up file must be a text file having a first line containing the value of M, the lifetime of a pathogen, T, the number of pathogens created at a given time step, and p_t, the probability that at a given time instant, a new pathogen will be created. The format of this first line in the file is

M`-value` T`-value` p_t`-value`

with any amount of white space between the values. M and T must be at least 1, and $0 \leq p_t \leq 1$. After this line, there will be one line for each active cell. Each line should have two integers separated by white space. The first is the row coordinate, the second, the column coordinate. The coordinates are 0-based. The upper-left hand corner of the window is (0,0). The program should ignore blank lines and lines that start with #. If the file has any other type of line, the program should exit with an error message. For example, the file contents:

```
4   2  0.25
# B-cell positions follow
24   16
22  16
24 57
23 15
```

indicate that the initial state of the grid has active cells at (24,16), (22,16), (24,57), and (23,15), and that $M = 4$, $T = 2$, and $p_t = 0.25$. If any cell's coordinates lie outside the size of the terminal window, that cell is discarded.

Once the initial state is loaded, the program should display it in the window. It should also display the prompt "`command:`" in leftmost position of the bottom row and "park the cursor" to the right of it. Your program will allow the user to enter the following commands in response to the prompt:

- Begin the simulation with '`b`' (only in the initial state.)
- Control the speed of the simulation by pressing '`+`' to increase speed and '`-`' to decrease speed (only if the simulation is not paused.)
- Terminate the simulation by pressing '`q`' (at any time.)
- Pause the simulation with '`p`' (only if the simulation is not already paused.)
- Resume the simulation with '`r`' (only if it is paused.)

The commands should be echoed at the cursor so that the user can see her typing. When the user types, the entered text should always be displayed immediately to the right of the command prompt. After the text is typed and the command is executed, the text should be erased.

More Details

Speed. The initial speed should be 2 steps every second. When the '+' is pressed, the speed will increase from n steps per 2 seconds to n+1 steps per 2 seconds, up to a maximum of 20 steps per 2 seconds. When the '-' key is pressed, the speed should decrease from n steps per second to n-1 steps per second, to a minimum of 1 step per 2 seconds. Attempts to exceed either bound should be silently ignored – no error messages or warnings should be displayed. Since you will not be able to obtain some of these speeds to 100% precision, use the highest possible precision possible. (You cannot express 1/3 of a second exactly.)

Pausing and resuming. If the user pauses the simulation, the simulation enters a state in which the pattern remains on the screen without changing, indefinitely. If the user then resumes the simulation, it will continue from where it left off. When the simulation is paused, attempts to change the speed (using '+' or '-') should be ignored, but the user should be able to quit. Attempts to pause the game should be ignored as well. Similarly, resuming a game that is not paused should have no effect.

Granularity. If a key is pressed during an update of the state of the screen, the program should finish the screen update before acting on the key-press. For example, if a speed change is requested, the next state should be computed completely, the speed changed, and the state displayed. If a pause is requested, the state should be updated before the pause takes place.

Pathogen Lifetime and Placement. At each time step, the program should flip a biased coin that has probability of $p_t$as success and $1 - p_t$ as failure. If the throw is successful, T pathogens shall be placed on the screen. The positions of the pathogen shall be determined by at least T successive random tosses. Think of the screen as being in row-major order – row 2 follows row 1, row 3 follows row 2 and so on. If the part of the screen containing cells is R rows by C columns, then there are RC cells. Each random toss shall generate a random number between 0 and $RC - 1$. The number n represents cell $(n/C, n\%C)$. For example, if there are 24 columns, then the number 54 is the cell (2,6). If that cell is occupied already, a number is generated again, and this is repeated until an empty cell is found. This takes place until T empty cells have been filled with pathogens.

The placement of pathogens occurs before the transition from time t to time $t + 1$. In other words, the pathogens are placed into the cells of the grid that exists at time t, and then the grid at time $t + 1$ is computed.

Extra Credit:

For an extra 10%, if the user presses 's', the program should save the state of the grid in a file named `.genesis_<pid>` in the user's working directory where `<pid>` is replaced by the processid of the process itself. If the file exists, it should be replaced. The file format should be exactly the same as that of an initialization file. The order of the lines does not matter of course.

Submitting the Assignment

You are to create a zip file containing all of your source code and put that zip file in the directory

```
/data/biocs/b/student.accounts/cs493.66/projects/project4
```

naming it `username_hwk4.zip`. Give it permissions 600 so that only you have access to it. Whether you have a single file or multiple files, you are to create a directory named `username_hwk4`, putting all files into it and using the command

```
zip -r username_hwk4.zip username_hwk4
```

to create the zip file. Make sure that you create a Makefile if you use multiple files, and include that Makefile in the directory. If you use a Makefile, make sure that it creates an executable named `encode` when it is built. If it is a single file program, name the source file `genesis.X`, where `X` is the GNU extension for the language (.c for C, .cpp or .C for C++, etc.)

The program must be well-documented and must conform to my programming guidelines for full credit. It must be placed in the directory no later than midnight of the due date.