COMP 3170 Summer 2012: Assignment 1

due in class May 22

This assignment is worth 10% of the course grade; it will be marked out of 50. You must hand in a signed Honesty Declaration (the form is linked from the course Web site) with this assignment.

Life Without Death

You've probably seen the "Game of Life" cellular automaton: cells in a grid, switching between "alive" and "dead" states according to simple rules based on the number of nearby cells that are alive. In this assignment we will look at a related game called Life Without Death, which follows the same rules as Life except that once a cell is alive, it never dies.

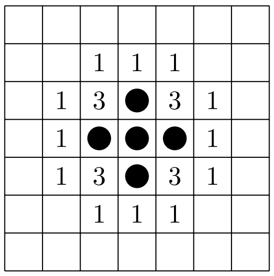
The game starts with a large two-dimensional grid of cells; most of the cells are considered to be "dead" at the start, but we will set a few of them to be "alive." Then we apply rules to the grid to determine the state of the cells in the next generation. All cells switch to their new values all at once. Then we calculate a new generation.

For every cell, we look at its eight neighbours, which surround it on all sides in the grid: left, right, up, down, and all four diagonals. The rule is that if a cell is already alive, it remains alive; if it is dead but exactly three of its eight neighbours are alive in the current generation, then it becomes alive in the next generation; and otherwise, it remains dead in the next generation.

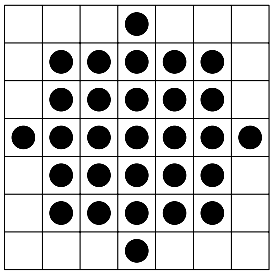
For example, suppose we have this grid with a line of three living cells denoted by the circles:

	1	2	3	2	1	
	1	●	●	●	1	
	1	2	3	2	1	

The numbers show the number of living neighbours for each nearby dead cell. The two dead cells that have three neighbours are the ones that will be "born" (become alive) in the next generation:



Then four new cells will be born in the third generation, and so on. This particular example will terminate in the sixth generation with the following pattern, from which no new cells will ever become alive:



1 Analyzing a simple algorithm [5]

Here's a simple algorithm for simulating the game of Life Without Death:

Start with a 2-D array A where each cell can be "alive" or "dead," containing the initial pattern. Also keep a similar array B for temporarily storing the next generation.

```
for each generation
   changed ← 0
   for i ← each x-coordinate in A
      for j ← each y-coordinate in A
         if A[i, j] is alive
            B[i, j] ← alive
         else if A[i, j] has three living neighbours
            B[i, j] ← alive
            changed ← changed + 1
         else
            B[i, j] ← dead
         end if
      end for
   end for
   if changed = 0 then terminate
   A ← B
end for
```

Answer the following questions related to the analysis of this algorithm. Bounds should be tight worst-case asymptotic bounds, with brief justifications; detailed proofs are not required. Assume the grid is n cells by n cells, and a word RAM model of computation.

- How much memory does the algorithm require?
- How long does this algorithm take to simulate one generation?
- Explain why the algorithm must terminate after some finite number of generations, and give an upper bound on the number of generations.
- How much time is required to run the algorithm until it terminates?

2 Analyzing a more complex algorithm[15]

It's easy to show that (after the initial setup) a cell in Life Without Death can only be born (change from dead to alive) in a given generation if one of its neighbours was born on the previous generation. That means instead of checking every cell as in the simple algorithm, we could record the cells born in each generation and check only their neighbours when computing the next generation. That seems like it should speed up the calculation. Here is an algorithm that works that way:

Start with a 2-D array A where each cell can be "alive" or "dead," containing the initial pattern. Also keep two queues P and Q of cell coordinates.

```
Initialization: start with $P$ empty
**for** $i \leftarrow$ each $x$-coordinate in $A$
   **for** $j \leftarrow$ each $y$-coordinate in $A$
      **if** $A[i,j]$ is alive
         add $(i,j)$ to $P$
      **end if**
   **end for**
**end for**

**for** each generation
   $Q \leftarrow$ empty
   **for** $(i,j) \leftarrow$ each item in $P$
      **for** $i' \leftarrow (i-1)\dots(i+1)$
         **for** $j' \leftarrow (j-1)\dots(j+1)$
            **if** $A[i',j']$ is dead **and** has three living neighbours
               add $(i',j')$ to $Q$
            **end if**
         **end for**
      **end for**
```

```
    **end for**
    **if** $Q$ is empty **then terminate**
    $P \leftarrow$ empty
    **for** $(i, j) \leftarrow$ each item in $Q$
        **if** $A[i, j]$ is dead
            $A[i, j] \leftarrow$ alive
            add $(i, j)$ to $P$
        **end for**
    **end for**
**end for**
```

As with the simpler algorithm, answer the following questions. Give worst-case asymptotic bounds, with brief justifications, assuming the grid is n cells by n cells, with a word RAM model of computation.

- How much memory does the algorithm require?
- How long does this algorithm take to simulate one generation?
- How much time is required to run the algorithm until it terminates?
- Why do we update A by scanning through Q at the end of the generation, instead of using a separate array B and copying it over as in the simpler algorithm?
- Why do we need the **if** statement inside the loop that updates A and P?

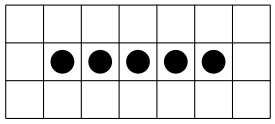
3 Implementation [30]

Implement the game of Life Without Death.

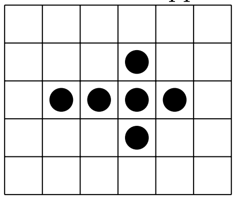
Choose one of the two algorithms described above. Your implementation should be able to accept as input a starting pattern of living cells and a number of generations; it must place the starting pattern in the middle of a grid of at least 200×200 cells, run the game for the chosen number of generations, and report the result. Problems you should solve include choosing a reasonable way of handling the boundaries at the edges of the grid, and finding a way to present the output as clearly and usefully as possible. You may use third-party software (whether libraries or standalone programs) to help format your output, but the Life Without Death calculation must be done by your own code.

Run the following simulations, and hand in the results. Also hand in your source code, and describe any additional steps you performed to present the results.

Simulate this pattern until it terminates, and report how many generations that takes:



Simulate the following pattern twice, once for 50 generations and once for 100 generations. Briefly describe what would happen if we simulated it indefinitely.



Simulate this pattern for 200 generations: