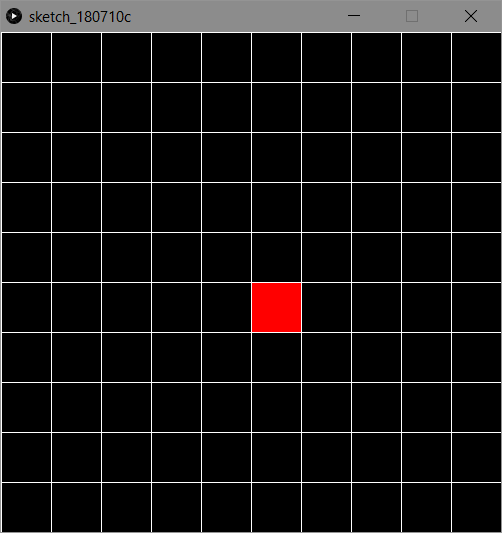
# Assignment 5: Bacteria Wars!

### Notes:

* Name your sketches using your name, the assignment number, and the question number, *exactly* as in this example: LastnameFirstnameA5Q1.
* Your programs must run upon download to receive any marks.
* Submit one PDE file for each question.
* Assignments must follow the programming standards document published on the course website on UMLearn.
* After the due date and time assignments may be submitted but will lose 2% of marks per hour late or portion thereof.
* You may submit a question multiple times, but only the most recent version will be marked.
* These assignments are your chance to learn the material for the exams. *Code your assignments independently*. We use software to compare all submitted assignments to each other, and pursue academic dishonestly vigorously.

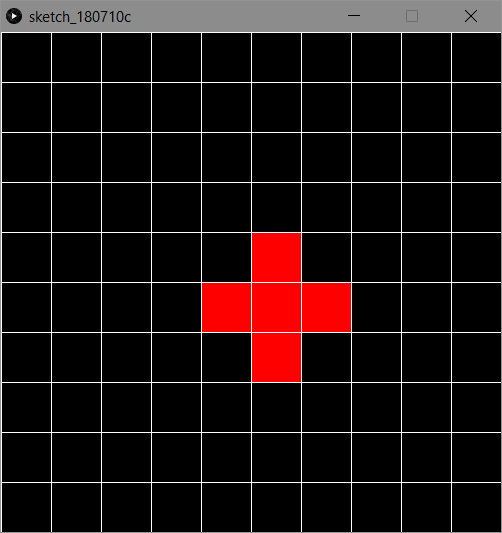
### Q1: Bacteria Colonies

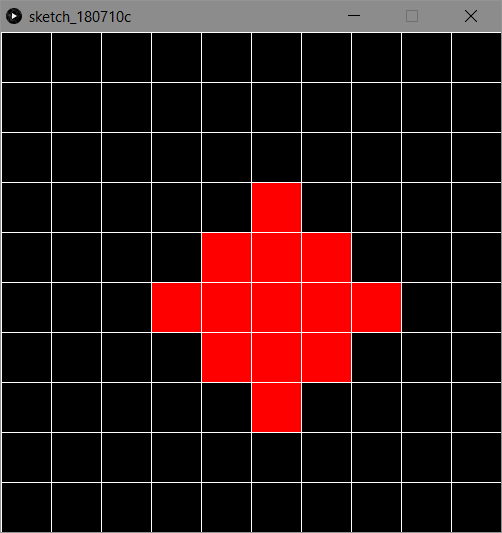
**Phase 1: Seeding a colony**

Start by finding out how large your squares array will need to be. If we decide we want to have SIDE\_LENGTH=10, so 10 squares per side. This means the canvas will be divided into 10x10 number of squares, so 100 for this example.

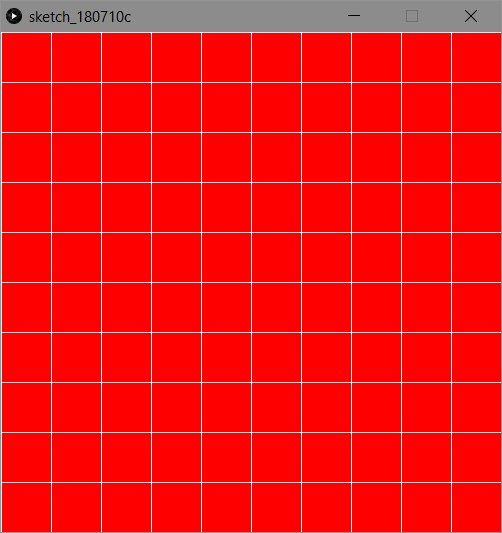
1. Make it so that your program automatically creates an array the correct size no matter the SIDE\_LENGTH.
2. All the values of the array will start at 0 so we will use that as the state of an uninfected cell.
3. Create a constant for the state of an infected cell and give it some value that is not 0.
4. Make an array of colours for the different states, use hex colours like #ff0000, #00ff00, #0000ff. Or use grayscale.
5. Create a function that will choose a random cell to become infected and give it a corresponding value, call it **infectRandom**().
6. Create a **drawBoard**() function to print out the current state of the board. It will be called at every draw()
7. Remember there are SIDE\_LENGTH number of cells per side, which means per row and per column so take the appropriate steps so that all the cells are properly printed. You will either need one loop and the divide and modulo operators or two loops.
8. **Set “framerate(2)” or another similarly small value in setup so that the program doesn’t go too fast!**

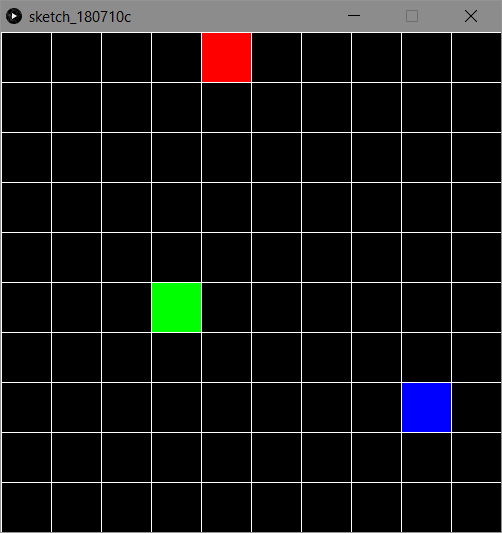
**Phase 2: Spread the infection!**

 Our bacteria won’t stay still for long, it wants to spread across the canvas! It will start to infect the cells that are around it.

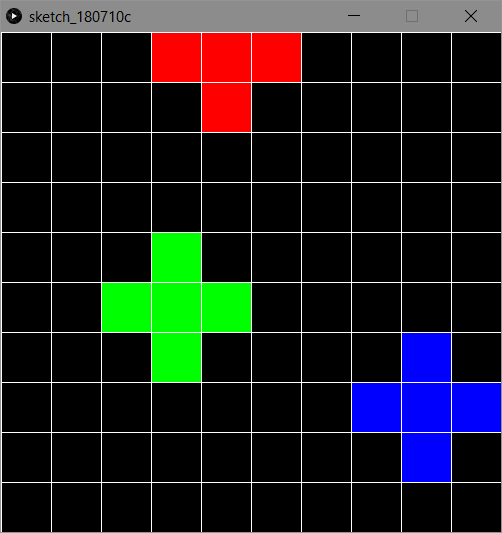
1. Create a **spreadInfection**() function which will contain a loop that will find all the infected cells.
2. Each infected cell found in **spreadInfection**() should infect it’s neighbours, create another function with the signature “**void infect(int source, target)**”, you can change this signature if you need to.
3. Be sure that **spreadInfection**() and infect() do not go outside the array and properly checks the cells above and below the infected cell you are looking at.
4. Be careful also that the infection does not jump from one side of the canvas to the other, check to see that the cell you are infecting is either in the same column or the same row as the cell that is infecting it.
5. Make sure that **infect**() only works on a target that is **not already infected.**

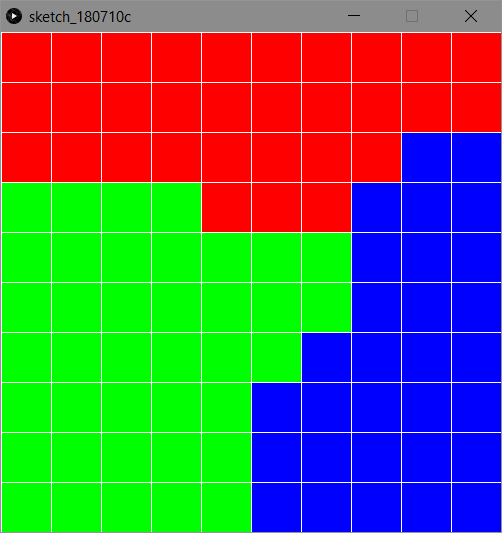
Eventually…

1. You will need to make a ***deep copy***of your arrays at every frame and store the changes you make in temporary arrays (like squaresTemp) that you’ll ***shallow copy*** to the originals after the **spreadInfection**() loop.
2. This is to avoid spreading the infection with a cell that just got infected this frame, a compound effect.
3. At this time your program should draw a board with an infection that slowly grows to encompass the whole canvas.
4. **Save your program at this point and hand it in as Q1.**

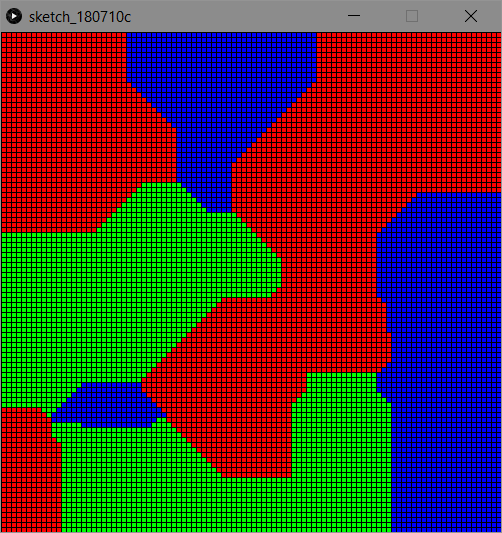
**Q2, Phase 3: Rival Infections!**

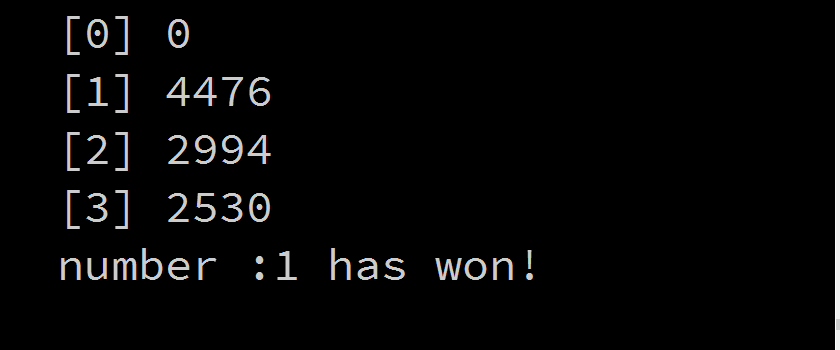
Looks like there are some new germs on the block, and they’re looking to crave themselves a piece of the board.

1. Make a copy of your Q1 and rename it for Q2.
2. Add another 2 types of cell, with different values and colours from our current two.
3. These cells will also infect their neighbours but cannot infect a cell that has already been infected.
4. Everything should be the same as it was in Q1, only with more than 1 type of cell.



Eventually…

**Q2, Phase 4: First the canvas, and now the world!**

1. Increase the number of cells on the canvas, at least 100 per side.
2. Remove the stroke lines using nostroke();
3. Add another 4 random start points for each infected cell type, for a total of 5 each
4. Use a Boolean variable to track if all the cells have been infected, you’ll know when that happens when spreadInfection() doesn’t lead to any new infections.
5. When all cells have been infected tally up the number of cells for each type of infected cell and print out the number for each and the overall winner.
6. Remember, there should not be any type 0s left!

### Assignment 5 Marking Guide

Q1: Bacteria Phase 1&2 (14 marks)

* Split the canvas into even squares with 1 array (1 mark)
* All cells start as healthy (1 mark)
* Proper use and naming of constants and variables (2 marks)
* Constant and colour variables for infected cells (1 mark)
* infectRandom() properly created. (1 marks)
* infect() infects a neighbor cell that is not already infected (1 mark)
* spreadInfection() correctly find all currently infected cells and calls infect on their neighbours (2 marks)
* Only the cells that were infected at the start of the frame spread the infection (2 mark)
* Infection spreads in expected manner (2 marks)
* drawBoard() correctly prints out current state of cells (1 marks)

Q2: Bacteria Phase 3&4 (16 marks)

* Additional types of infected cells added (1 mark)
* New types of infected cells do not infect other types of infected cells, only healthy cells (1 mark)
* New types of infected cells have a different colour (1 marks)
* Initial random placement of infected cells is done in a clean and efficient manner (1 marks)
* drawBoard() correctly prints out current state of cells with the new types and colours (1 marks)
* Infection spreads in expected manner (2 marks)
* Program correctly works with larger canvas sizes (1 marks)
* Program correctly works with multiple random starting locations of the same type (1 marks)
* Program correctly tracks when there are no more healthy cells left (2 points)
* Program tallies the cell totals of each type and announces the winner. {2 points)
* Code is clean, readable, and well organized (3 points)

Grade: /30