ECS 50 Winter 2019
Homework Assignments 6a and 6b
Due no later than 10:00pm Friday, March 15

The Game of Life is a well-known mathematical game that gives rise to amazingly complex behavior, although it can be specified by a few simple rules. (It is not actually a game in the traditional sense, with players competing for a win.) Here are the rules. The game is played on a rectangular grid. Each square on the grid can be either empty (represented by a 0 in our version) or occupied by a "living" cell (represented by a 1). At the beginning, you can specify empty and occupied cells in some way; then the game runs automatically. In each *generation*, the next generation is computed. A new cell is born on an empty square if it is surrounded by exactly three occupied neighbor cells. A cell dies of overcrowding if it is surrounded by four or more living neighbors, and it dies of loneliness if it is surrounded by zero or one living neighbor. A neighbor is an occupant of an adjacent square to the left, right, top, or bottom or in a diagonal direction. Figure 16 below shows a cell and its neighbor cells.

Many configurations show interesting behavior when subjected to these rules. Figure 17 shows a *glider*, observed over five generations. Note how it moves. After four generations, it is transformed into the identical shape, but located one square to the right and below.

One of the more amazing configurations is the glider gun: a complex collection of cells that, after 30 moves, turns back into itself and a glider (see the large figure at the end).

Your task is to use CUSP to program the game to eliminate the drudgery of computing successive generations by hand. Your homework assignment comes in two parts, also known as Homework Assignment 6a and Homework Assignment 6b. The Wikipedia page on **The Game of Life** has a lot of useful information, as well as some real-time examples of program behavior. You should take a look at it.

Implementing this game requires a plan for what to do about the cells at the edges of the grid, since those cells don't have a full complement of eight neighbors. The Wikipedia entry does bring up this issue, and options range from simple to complex. For our purposes, it is sufficient to assume that every square outside of the grid is empty (i.e., 0) and will remain empty for the duration of the game.

(Much of the description above, and all the figures below, is borrowed from a textbook by Cay Horstmann.)

Homework Assignment 6a

The core problem to be solved in the implementation of the Game of Life is how to generate the next grid from the current grid. The grid will be a two-dimensional row-major array in memory, in which each word represents a cell in the grid. The value in each word is either 0 (empty or dead cell) or 1 (occupied or alive cell). Your task is to write a CUSP subroutine called nextGen that creates the next grid from the current grid by applying the rules described above. Your subroutine should expect the following parameters to be pushed on the stack by the calling procedure in this order:

> The starting address of the array representing the current grid will be pushed first on the stack
>
> The number of rows in that array will be pushed next
>
> The number of columns in the array will be pushed next
>
> The starting address of the new array representing the next grid will be pushed last

Your subroutine should create a new array that is the next generation of the original array and place the new array in memory starting at the address given in the last parameter pushed on the stack. Assume that both arrays are implemented in row-major order. Note that the original array should not be altered in any way by your subroutine.

As an example, assume that your subroutine is passed an address to an array in memory corresponding to this grid:

```
0000000
0010000
0001000
0111000
0000000
0000000
```

your subroutine should create a new array (starting at the address that is the last parameter pushed on the stack) corresponding to this grid:

```
0000000
0000000
0101000
0011000
0010000
0000000
```

These grids correspond to generations 0 and 1 in Figure 17 below.

Have your subroutine load into memory at address $400. Your subroutine should not use any memory at addresses less than $400. That space is reserved for your solution to Homework Assignment 6b, the "main program".

Do not turn in this assignment by itself. Submit it with the work you do for Homework Assignment 6b, which you will find following the figures.

You may work on this assignment with another student enrolled in this class, or you may work on your own. You may not work with more than one other person.

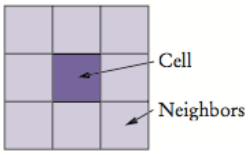


Figure 16
Neighborhood of a Cell

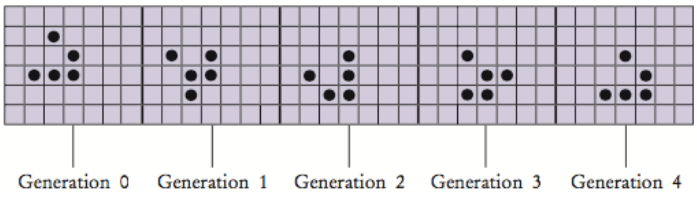
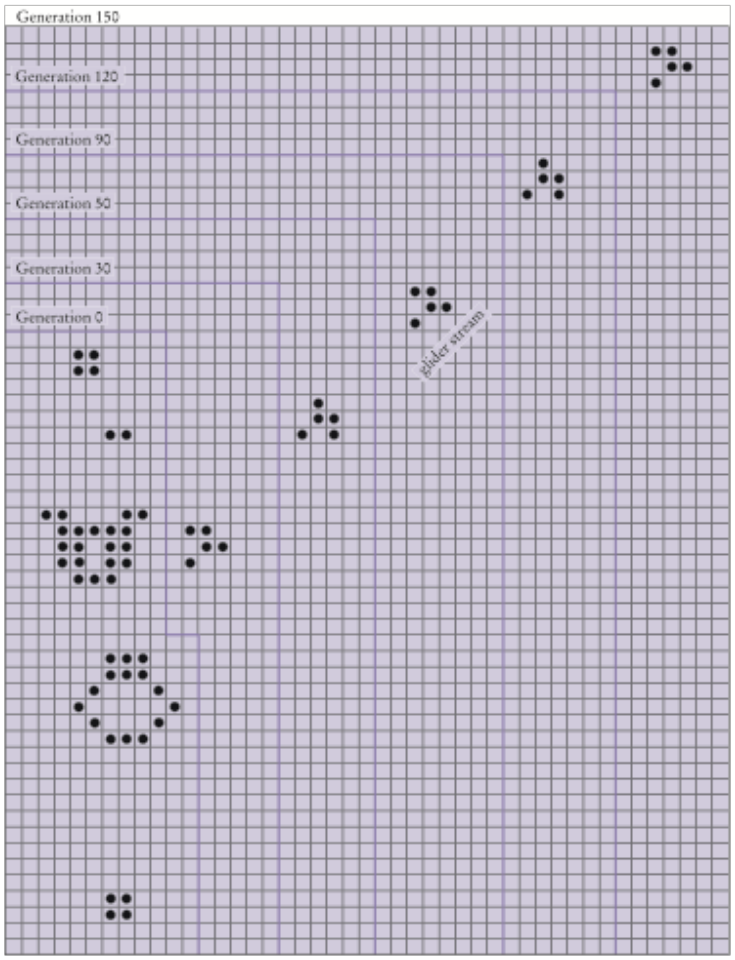


Figure 17
Glider


Generation 150
Generation 120
Generation 90
Generation 60
Generation 30
Generation 0
glider stream

Homework Assignment 6b

Your final programming assignment is to complete the Game of Life program that you began in Homework Assignment 6a. You are to write the main program that serves as an interface between the user and the nextGen subroutine you created for Homework Assignment 6. The interaction between the user and the main program should look like this:

```
Here is the starting grid:

Generation:  0
. . . . . . .
. . * . . . .
. . . * . . .
. * * * . . .
. . . . . . .
. . . . . . .

How many new generations would you like to print (enter 0 to
end)? 4

Generation:  1
. . . . . . .
. . . . . . .
. * . * . . .
. . * * . . .
. . * . . . .
. . . . . . .

Generation:  2
. . . . . . .
. . . . . . .
. . . * . . .
. * . * . . .
. . * * . . .
. . . . . . .

Generation:  3
. . . . . . .
. . . . . . .
. . * . . . .
. . . * * . .
. . * * . . .
. . . . . . .

Generation:  4
. . . . . . .
. . . . . . .
. . . * . . .
. . . . * . .
. . * * * . .
. . . . . . .
```

```
How many new generations would you like to print (enter 0 to
end)? 5

Generation:  5
. . . . . . .
. . . . . . .
. . . . . . .
. . * . * . .
. . . * * . .
. . . * . . .

Generation:  6
. . . . . . .
. . . . . . .
. . . . . . .
. . . . * . .
. . * . * . .
. . . * * . .

Generation:  7
. . . . . . .
. . . . . . .
. . . . . . .
. . . * . . .
. . . . * * .
. . . * * . .

Generation:  8
. . . . . . .
. . . . . . .
. . . . . . .
. . . . * . .
. . . . . * .
. . . * * * .

Generation:  9
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . * . * .
. . . . * * .

How many new generations would you like to print (enter 0 to
end)? 0

End of program.
```

It's not sufficient for your main program to print some number of generations in succession on the screen. Your program must use CUSP's timer to pause between the printing of one grid and the next. The pause should be about ten seconds. While the

printing is paused, your program should be sensitive to keyboard interrupts. During this time, when any key is pressed, your program should immediately print the next grid and reset the timer.

Your main program should load at address $000. Your main program and any test data should not extend past address $3FF.

It is important to remember that the dimensions of the initial grid may be different than what is shown in the example above.

Your main program must be written so that it uses your nextGen subroutine from Homework Assignment 6a to compute the new grids. Your nextGen subroutine must behave as specified in Homework Assignment 6a, and your main program must communicate with the nextGen subroutine in exactly the way specified in Homework Assignment 6b. We may replace your nextGen subroutine with our own when grading your program.

As we said above, you may work on Assignments 6a and 6b with another student enrolled in this class, or you may work on your own. You may not work with more than one other person.

When you are ready to submit solutions to Homework Assignments 6a and 6b, put your main program and your nextGen subroutine (along with all supporting subroutines that you have created) in a text file named hw6.csp. Put the object code in a file named hw6.obj. Make sure that your name and student id number are included as comments in your .csp file at the very beginning of the file. If you have worked with another student on these assignments, include the other student's name and student id number in those comments too.

Include both files (the .csp file and the .obj file) in your submission through Canvas. If you are working with another student, only one student may submit the files through Canvas. We do not want both students to submit. If you are worried that neither of you will submit your solutions, the solution is simple: communicate with each other. Decide who is responsible for submitting, then stay in touch until you are both sure that your work has been submitted successfully.