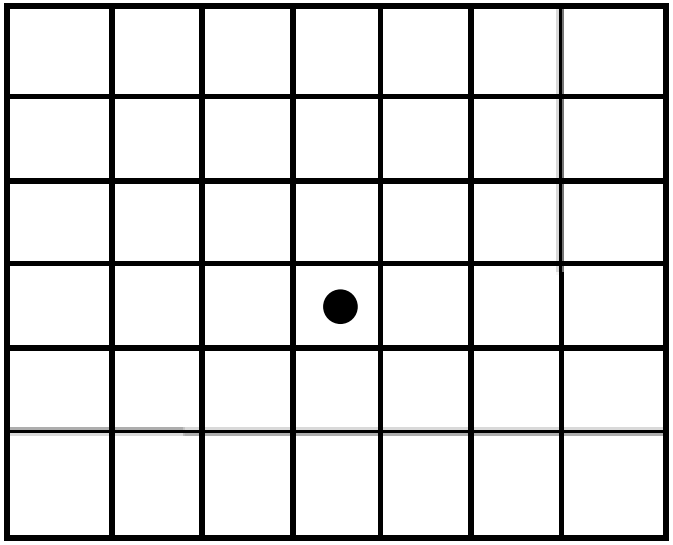
# Design and Analysis of Algorithms

*Lecture 3: Game of Life*

# Game of Life

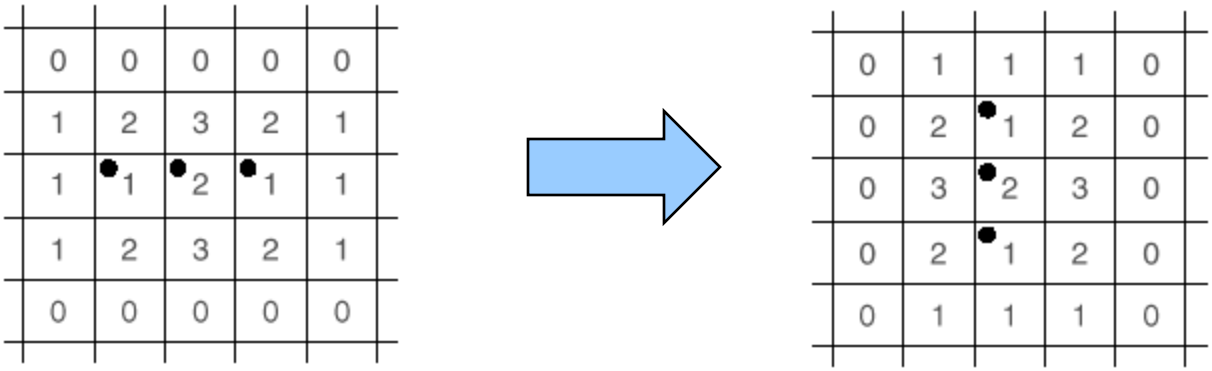


- Every cell is alive or dead
- Neighbor is every cell adjacent to it

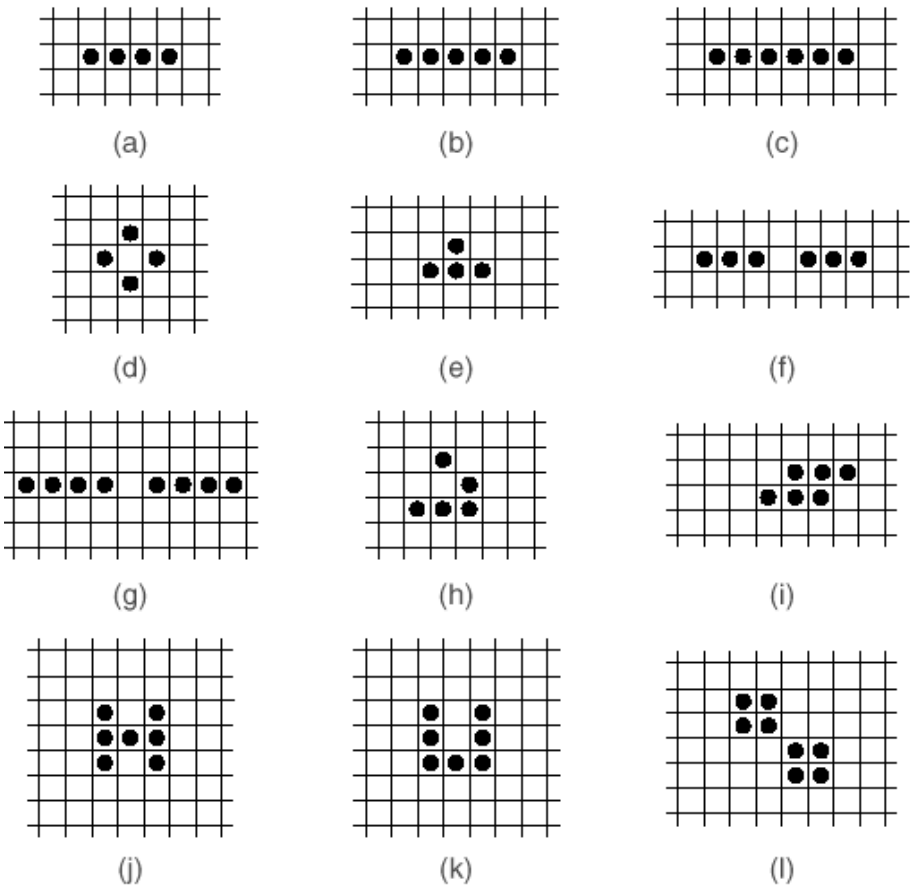
# Rules of Game of Life

- Alive cell stays alive in the next generation if it has either 2 or 3 living neighbors
- It dies if it has 0, 1, 4, or more living neighbors.
- A dead cell becomes alive in the next generation if it has exactly three neighboring cells, no more or fewer, that are already alive.
- All other dead cells remain dead in the next generation.

# Example


0 0 0 0 0
1 2 3 2 1
1 1 2 1 1
1 2 3 2 1
0 0 0 0 0
0 1 1 1 0
0 2 1 2 0
0 3 2 3 0
0 2 1 2 0
0 1 1 1 0

# More Examples



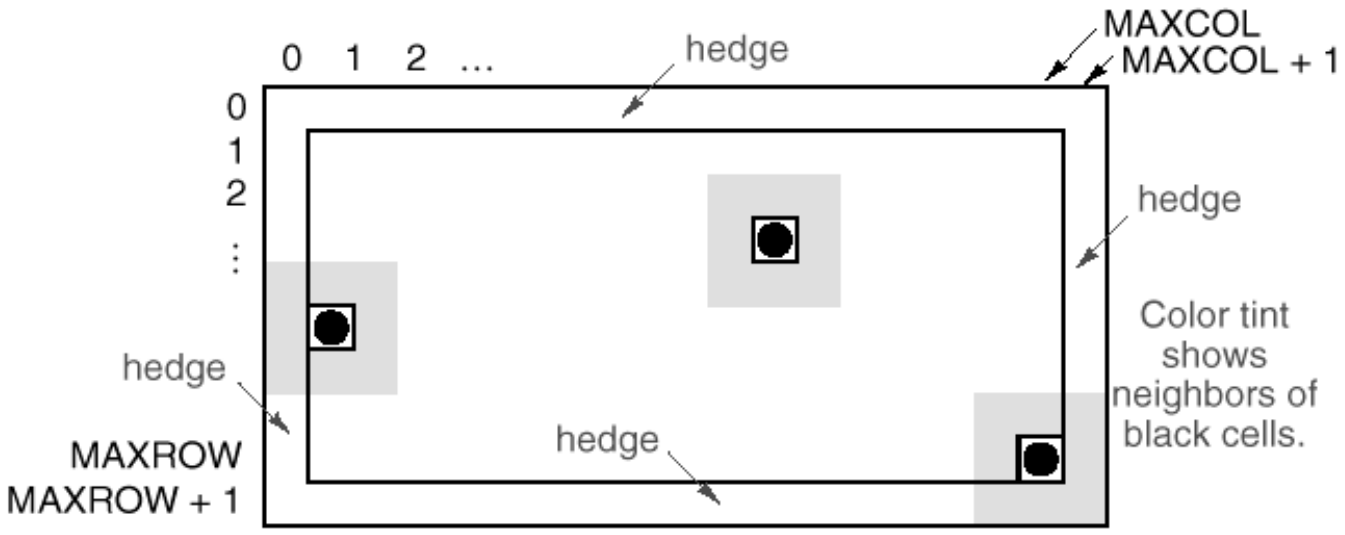
# Question

- How do we implement the GoL?

# Algorithm for Game of Life

- Initialize an array called map to contain the initial configuration of living cells
- Repeat as many times as desired
  - For each cell in the array
    - Count the number of living neighbors of the cell
    - If the count is 0,or greater than 4 then mark the cell in the newmap as **dead**
    - If the count is 3 the cell in the new map than the new cell is **alive**
    - If the count is 2 the content of the cell are copied to the corresponding cell in the newmap
- Copy the array newmap into array map
- Print the array map to the user

# Simplifying Neighbors Count



# Demo

- http://vlab.infotech.monash.edu.au/simulations/cellular-automata/game-of-life/demo/

# Analys

```
#include    "common.h"    /* common include files and definitions */
#include    "life.h"      /* Life's defines, typedefs, */
void main(void)
{
    int  row, col;
    Grid map;                          /* current generation   */
    Grid newmap;                       /* next generation      */

    Initialize(map);
    WriteMap(map);
    printf("This is the initial configuration you have chosen.\n"
           "Press <Enter> to continue.\n");
    while(getchar() != '\n')
        ;
    do {
        for (row = 1; row <= MAXROW;    row++)
            for (col = 1; col <= MAXCOL; col++)
                switch(NeighborCount(map, row, col)) {
                case 0:
                case 1:
                    newmap[row][col]= DEAD;
                    break;
                case 2:
                    newmap[row][col]= map[row][col];
                    break;
                case 3:
                    newmap[row][col]= ALIVE;
                    break;
                case 4:
                case 5:
                case 6:
                case 7:
                case 8:
                    newmap[row][col]= DEAD;
                    break;
                }
        CopyMap(map, newmap);
        WriteMap(map);
        printf("Do you wish to continue ");
    } while (UserSaysYes());
}
```

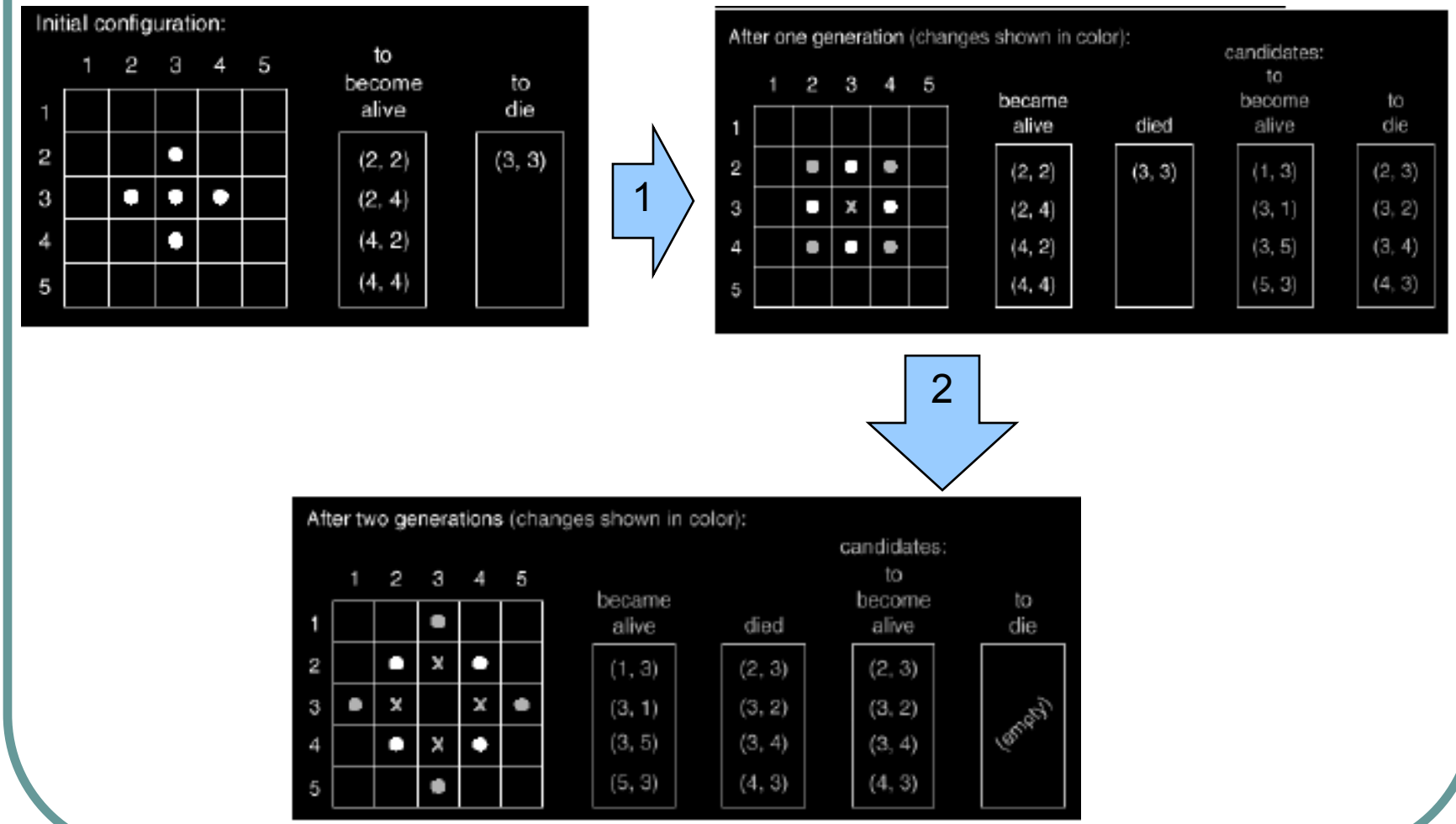
# Neighbor Count

```
int NeighborCount(Grid map, int row, int col)
{
    int i;          /*row of a neighbor */
    int j;          /*column ofaneighbor */

    int count = 0;

    for (i = row - 1; i <= row + 1; i++)
        for (j=col - 1; j<= col + 1; j++)
            if (map[i][j] == ALIVE)
                count++;
    if (map[row][col] == ALIVE)
        count--;
    return count;
}
```

# How to improve?

- Programming Concept
  - Most programs spend 90% of their time doing 10% of their instructions
  - Find those 10%, improve them

# Using Lists


Initial configuration:
1 2 3 4 5
to become alive
(2, 2)
(2, 4)
(4, 2)
(4, 4)
to die
(3, 3)
1
After one generation (changes shown in color):
became alive
(2, 2)
(2, 4)
(4, 2)
(4, 4)
died
(3, 3)
candidates: to become alive
(1, 3)
(3, 1)
(3, 5)
(5, 3)
to die
(2, 3)
(3, 2)
(3, 4)
(4, 3)
2
After two generations (changes shown in color):
became alive
(1, 3)
(3, 1)
(3, 5)
(5, 3)
died
(2, 3)
(3, 2)
(3, 4)
(4, 3)
candidates: to become alive
(2, 3)
(3, 2)
(3, 4)
(4, 3)
to die
(empty)

# Game of Life 2

Get the initial configuration of living cells and use it to calculate an array holding the neighbor counts of all cells. Determine the cells that will become alive and that will become dead in the first generation;

Repeat the following steps as long as desired:

Vivify each cell that is ready to become alive;

Kill each cell that is ready to die;

Write out the map for the user;

Increase the neighbor counts for each neighbor of each cell that has become alive; If a neighbor count reaches the appropriate value, then keep track of the cell as a candidate to be made alive or dead in the next generation;

Decrease the neighbor counts for each neighbor of each cell that has become dead; If a neighbor count reaches the appropriate value, then keep track of the cell as a candidate to be made alive or dead in the next generation.

**void** Vivify(ListEntry cell);

*Pre*: The cell is a candidate to become alive.

*Post*: Checks that cell meets all the requirements to become alive. If not, no change is made. If so, then cell is added to the list newlive, and array map is updated.

*Uses*: Function AddList, array numbernbrs, changes array map and list newlive as global variables (side effects).

**void** AddNeighbors(ListEntry cell);

*Pre*: cell has just become alive.

*Post*: Array numbernbrs has increased counts for all the cells neighboring cell. If the cell thereby becomes a candidate to be vivified [*resp.* killed] then the cell has been added to list maylive [*resp.* maydie].

*Uses*: Function AddList; changes array numbernbrs and lists maylive and maydie as global variables (side effects).

# Main2.c

```
#include " common.h"
#include "life2.h"

Grid map ;           /* global: square array holding cells
*/
Gridcount  numbernbrs ;/* global: square array holding neighbor counts
*/
List  newlive,  /* global: the cells that have just been vivified
*/
      newdie,        /* global: the cells that have just died
*/
      maylive,  /* global: candidates to vivify in the next generation
*/
      maydie;        /* global: candidates to kill in the next generatio
*/
int  maxrow,  maxcol; /* global: user defined grid size
*/
int main(void)
{
      Initialize(map, numbernbrs, & newlive, & newdie, & maylive, & maydie);
      WriteMap(map);
      printf("Proceed with the demonstration");
      while (UserSaysYes()) {
[redacted]
and newdie */
            WriteMap(map);
            ClearList(&maylive);
            ClearList(&maydie);
[redacted]
            ClearList(&newlive);
            ClearList(&newdie);
            printf("Do you want to continue viewing new generations");
      }
      return 0;
}
```

# Proof of correctness

- “Initialize” function should be written properly
- “Maylive”, “Maydie “are mutually exclusive sets so same sell cannot be revived and then killed
- Only sells where the count changes can change their status and those cases are taken care of with “Add”, “Subtract” neighbor.

# Vivify

```
void Vivify(ListEntry cell)
{
    if (map[cell.row][cell.col] == DEAD &&
        numbernbrs[cell.row][cell.col] == 3)
        if (cell.row >= 1 && cell.row <= maxrow
&&      /* not on hedge */
            cell.col >= 1 && cell.col <= maxcol)
{
            map[cell.row][cell.col] = ALIVE;
            AddList(cell, &newlive);
        }
}
```

# AddNeighbors

```
void AddNeighbors(ListEntry cell)
{
    int nbrrow,     /* loop index for row of neighbor loops */
        nbrcol;     /* column loop index    */
    Cell neighbor;  /* structure form of a neighbor */

    for (nbrrow = cell.row-1; nbrrow <= cell.row+1; nbrrow++)
        for (nbrcol = cell.col-1; nbrcol <= cell.col+1; nbrcol++)
            if (nbrrow != cell.row || nbrcol != cell.col) { /* Skip cell itself. */
                numbernbrs[nbrrow][nbrcol]++;
                switch (numbernbrs[nbrrow][nbrcol]) {

                    case 0:
                         Error("Impossible case in AddNeighbors.");
                         break;


                    case 3:
                         if (map[nbrrow][nbrcol] == DEAD) {
                              neighbor.row = nbrrow; /* Set up a coordinate record. */
                              neighbor.col = nbrcol;
                              AddList(neighbor, &maylive);
                         }
                         break;


                    case 4:
                         if (map[nbrrow][nbrcol] == ALIVE) {
                              neighbor.row = nbrrow; /* Set up a coordinate record. */
                              neighbor.col = nbrcol;
                              AddList(neighbor, &maydie);
                         }
                         break;
                }   /* switch statement */
            }
}
```

# Game of Life V2 analysis

- All most all work is done in the 4 traversing statements – Vivify, Kill, AddNeighbors, SubtractNeighbors
- The amount of work is no longer dependant on the size of the grid but on the number of state changes.
- If there are 50 cells occupied for a typical configuration, then on the average there will be 25 state changes.

# Conclusion

- Running Time Complexity ?
  - GoL v1
  - GoL v2
- Space Complexity
  - GoL v1
  - GoL v2
- Effort