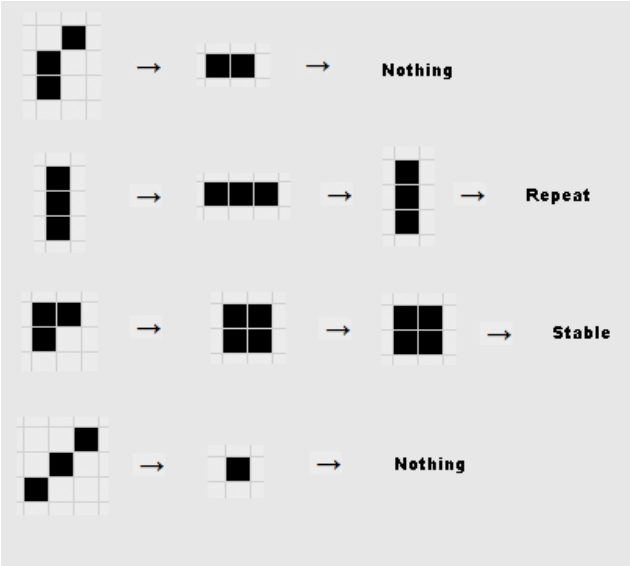
Tatyana VlaskinAssignment 1 Design–Program Design and Testing

We need to submit this part before the implementation and reflection
Understanding:
We need to design an array that will store live and dead cells
Specify initial configuration of the world
Ask the user if they want to move on to the next generation
If they say yes, we need to update the world using the following rules:

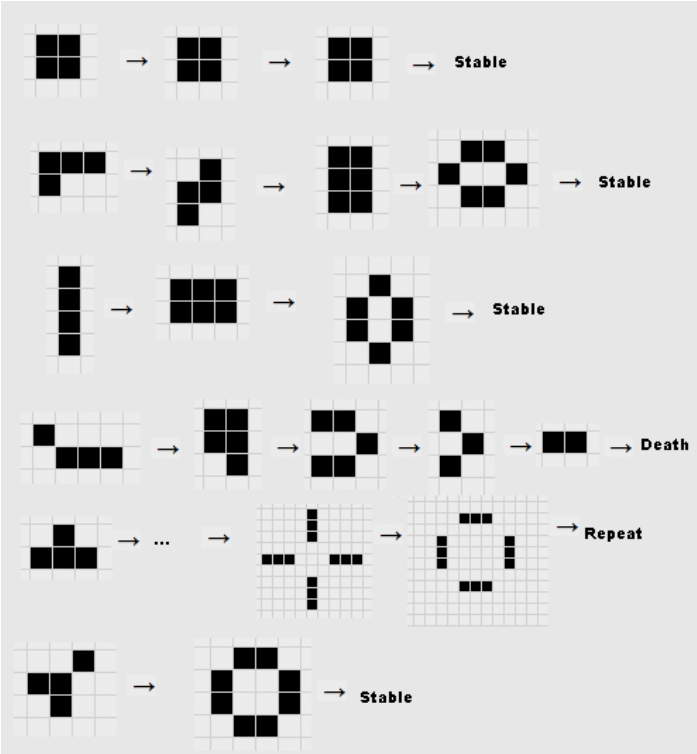
1. We need to look at the neighbors around each cell. Cell is surrounded by 8 neighbors.
2. We need to determine what to do with the cells at the edges because they do not have 8 neighbors.
3. We need to count live cells about each live cell and determine whether the cell will die, move on to the next generation or a neighbor is born.
4. Alive cell dies of overcrowding if it has >3 of living neighbors. We should verify that any living cell with > 3 neighbors dies.
5. Alive cell dies of loneliness if it has <2 of living neighbors. We should verify that any single living cell or any pair of living cells will die during the next iteration
6. An empty cell becomes alive if it has exactly 3 living neighbors.
7. All other worlds remain unchanged
1. A cell dying for whatever reason may help cause birth, but a newborn cell cannot resurrect a cell that is dying, nor will a cell's death prevent the death of another.

Following these rules, the world can fade away completely(from overcrowding or from becoming to spars or the word can settle into a stable configuration that remains unchanged thereafter or enters in an oscillating phase in which they repeat an endless cycle.
I found these patterns online: http://www.math.cornell.edu/~lipa/mec/lesson6.html

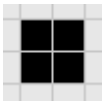
Some possible **triomino patterns** (and their evolution) and outcome:

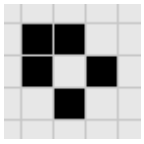


Some tetromino patterns

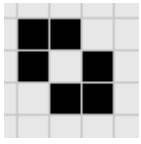


Some example still lifes:

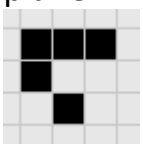
Square : 

Boat : 

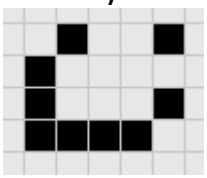
Loaf : 

Ship : 

The following pattern is called a "glider."The pattern repeats every 4 generations, but translated up and to the left one square. A glider will keep on moving forever across the plane.

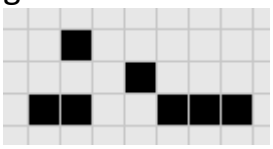


Another pattern similar to the glider is called the "lightweight space ship." It too slowly and steadily moves across the grid.



Early on (without the use of computers), Conway found that the F-pentomino (or R-pentomino) did not evolve into a stable pattern after a few iterations. In fact, it doesn't stabilize until generation 1103.

The F-pentomino stabilizes (meaning future iterations are easy to predict) after 1,103 iterations. The class of patterns which start off small but take a very long time to become periodic and predictable are called Methuselahs. The students should use the computer programs to view the evolution of this pattern and see how/where it becomes stable. The "acorn" is another example of a Methuselah that becomes predictable only after 5206 generations.



DESIGN

1. We need an array. The problem asks us the construct a 22x80 world. Ill make my array in memory 2 tiles larger in each dimension to make sure that I do not run out of bounds when I check cells on the edges. When I display the array to the user, rows 0 and 23 as well as columns 0 and 81 will not be visible (ill loop from 1-22 and 1-80 for rows and columns respectively).
2. Initially array will be filled with '-' to indicate that all cells are dead. The user will not see this array.
3. There will be 2 options to generate the initial configuration of alive cells in the world: the user will have an option to let the computer randomly generate 200 live cells or the user will have an option to indicate which cells they want to make alive. I'll make a menu for that.
4. So for the randomly generated world, I'll make a simple function:

```
void continueAutomate(char generation[][COLUMNUMNS]){
   int coutAlive;
   srand(time(0));
   for (int countAlive =0; countAlive < 200; countAlive++){
            generation[rand()%22+1][rand()%80+1] = 2;
   }
```

```
}
```
Please note that row 0 and 23 as well as columns 0-81 are excluded, so live cell will not be placed there. Those cells will be always dead.

5. For the manual generation of the world, the function will look like that:

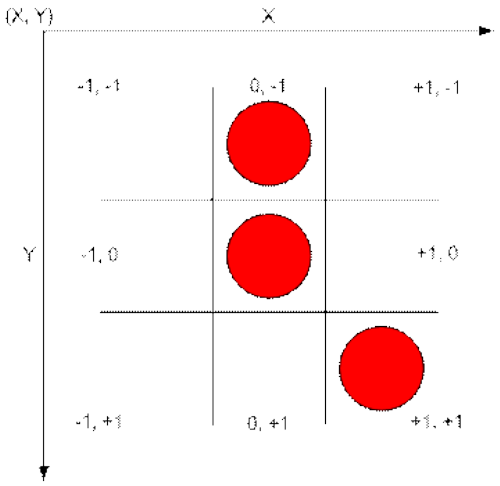
```
void continueManual(int row, int column){
   char nextMove;
   bool cont=true;
   while(cont==true){
      do{
         cout<<"Enter row index of an alive world(1-22): ";
      }while(row<1||row>22);
      do{
         cout<<"Enter colum index of an alive world(1-80): ";
      }while(column<1||column>80);

      world[row][column]=2; CELL WILL BE ASSIGNED TO LIVE
       cout<<"\nDo you want to mark another alive world?(y/n): ";
      do{
         cin>>nextMove;
      }while(user wants to assign another cell to be alive);
      if(nextMove=='n'||nextMove=='N')
         cont=false;
   }
}
```

6. Once the world is generated, we'll ask the user if they want to go to the next generation.
7. If the answer is yes, we need a function to count live neighbors around each cell.
   Rows go from 0 to 23.
   Columns go from 0-81. We do not check cells in row 0 and 23, and column 0 and 81.
   Cells in those 2 rows and 2 columns will be dead all the time. So when I will test cells at the edges, all 8 neighbors will be checked and 3 cells that are "out of bounds" will be counted as dead.

   I am planning to look at the neighbors by rows.: row above, same row and row below.

On the following website: http://ptgmedia.pearsoncmg.com/images/0672320665/downloads/The%20Game%20of%20Life.html, I found useful diagram:



Lets say we are looking at the cell [0][0] and we want to check its neighbors above, so we'll be looking at the row [0-1] (to generalize it is[idex-1] row). As for the column, we need to check column to the right and column to the left, so we are looking at (-1,-1) and (+1,-1) coordinates. (to generalize, the indexes of the columns are [column-1] and [column +1]. If there is a life in those cell, the cell will be counted as an alive neighbor.

```
//checking above row
if(((index_X-1)>=0)&&((index_Y-1)>=0)){ neighbors++;} TO THE LEFT
if(((index_Y-1)>=0)){neighbors++;}ABOVE
if(((index_Y-1)>=0)&&((index_X+1)<COLUMNUMNS){neighbors++;}TO THE RIGHT
```

Similar thing will be done for the row on which the cell is located and the row below.
I will assign live cell to char 2, which is displayed as smiley face.

8. After neighbors are counted, we need to determine which cells will die, which will stay alive and which will stay unchanged. There will be a function for this. The outcome will be stored in the temporary array to avoid changes in the original array before all counting is complete. So array is updated after we loop through the whole array.

```
if(neighbors>3 && generation[index_X][index_Y]==2){//overcrowding death
      Replace live cell with dead : '-';
   }
   else if(neighbors<=1 &&generation[index_X][index_Y]==2){//dies from loneliness
       Replace live cell with dead : '-';
    }
```

else if(neighbors==3 &&generation[index_X][index_Y]=='-'){//birth if 3 neighbors

Replace live cell with alive cell : 2;

}

else

cell stays the same

9. Steps #8 and #9 needs to be done for every single cell in the array.

for(i=1;i<ROWS-1;i++){//WE WILL IGNORE ROW 0 AND 23 AS WELL AS COLUMNUMN 0 AND 81

for(j=1;j<COLUMNUMNS-1;j++){

count cell

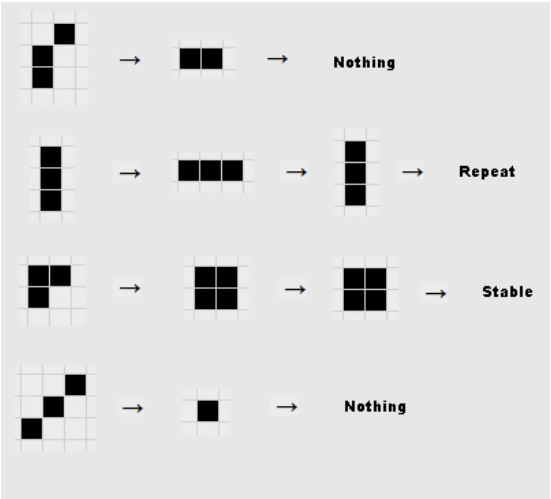
determine what will happen in the next generation

}}

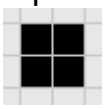
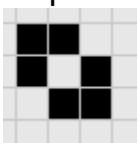
10. Once we are done looping through all cells, old world is replaced with the new world.
11. The user is asked if they want to move on to the next generation.
12. If the user decides to go to the next generation, above mentioned steps are repeated
13. If the answer is no, they program terminates with the thanks you message or something like that.

TESTING

First I'll try to set the following initial configuration and find out if they give expected outcome.



STABLE POSITIONS:

Square :	Boat :	Loaf :	Ship :
			

The following table is color coded. I'll assign the following position to alive to find out if the expected outcome is achieved in the next generation. The expected outcomes shown above.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,1	1,1	1,1	1,1	1,1	1,1	.....	1,7	1,7	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10	2,1	2,1	2,1	2,1	2,1	2,1	.....	2,7	2,7	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,1	3,1	3,1	3,1	3,1	3,1	.....	3,7	3,7	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,1	4,1	4,1	4,1	4,1	4,1	.....	4,7	4,7	4,7	4,8
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10	5,1	5,1	5,1	5,1	5,1	5,1	.....	5,7	5,7	5,7	5,8
6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9	6,10	6,1	6,1	6,1	6,1	6,1	6,1	.....	6,7	6,7	6,7	6,8
7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9	7,10	7,1	7,1	7,1	7,1	7,1	7,1	.....	7,7	7,7	7,7	7,8
8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9	8,10	8,1	8,1	8,1	8,1	8,1	8,1	.....	8,7	8,7	8,7	8,8
9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9	9,10	9,1	9,1	9,1	9,1	9,1	9,1	.....	9,7	9,7	9,7	9,8
10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9	10,10	10,11	10,12	10,13	10,14	10,15	10,16	.....	10,77	10,78	10,79	10,80
11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9	11,10	11,11	11,12	11,13	11,14	11,15	11,16	.....	11,77	11,78	11,79	11,80
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18,9	18,10	18,11	18,12	18,13	18,14	18,15	18,16	.....	18,18	18,19	18,20	18,21
19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19,9	19,10	19,11	19,12	19,13	19,14	19,15	19,16	.....	19,77	19,78	19,79	19,80
20,1	20,2	20,3	20,4	20,5	20,6	20,7	20,8	20,9	20,10	20,11	20,12	20,13	20,14	20,15	20,16	.....	20,77	20,78	20,79	20,80
21,1	21,2	21,3	21,4	21,5	21,6	21,7	21,8	21,9	21,10	21,11	21,12	21,13	21,14	21,15	21,16	.....	21,77	21,78	21,79	21,80
22,1	22,2	22,3	22,4	22,5	22,6	22,7	22,8	22,9	22,10	22,11	22,12	22,13	22,14	22,15	22,16	.....	22,77	22,80	22,79	22,80

My next step will be to test the edges.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1	1,10	1,11	1,12	1,1	1,1	1,1	1,1	.....	1,7	1,7	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2	2,10	2,11	2,12	2,1	2,1	2,1	2,1	.....	2,7	2,7	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3	3,10	3,11	3,12	3,1	3,1	3,1	3,1	.....	3,7	3,7	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4	4,10	4,11	4,12	4,1	4,1	4,1	4,1	.....	4,7	4,7	4,7	4,8
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5	5,10	5,11	5,12	5,1	5,1	5,1	5,1	.....	5,7	5,7	5,7	5,8
6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6	6,10	6,11	6,12	6,1	6,1	6,1	6,1	.....	6,7	6,7	6,7	6,8
7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7	7,10	7,11	7,12	7,1	7,1	7,1	7,1	.....	7,7	7,7	7,7	7,8
8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8	8,10	8,11	8,12	8,1	8,1	8,1	8,1	.....	8,7	8,7	8,7	8,8
9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9	9,10	9,11	9,12	9,1	9,1	9,1	9,1	.....	9,7	9,7	9,7	9,8
10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10	10,10	10,11	10,12	10,13	10,14	10,15	10,16	.....	10,77	10,78	10,79	10,80
11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11	11,10	11,11	11,12	11,13	11,14	11,15	11,16	.....	11,77	11,78	11,79	11,80
.....	.....	.....	.....	.....	.....	.....	.....	...	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
18,1	18,2	18,3	18,4	18,5	18,6	18,7	18,8	18	18,10	18,11	18,12	18,13	18,14	18,15	18,16	.....	18,18	18,19	18,20	18,21
19,1	19,2	19,3	19,4	19,5	19,6	19,7	19,8	19	19,10	19,11	19,12	19,13	19,14	19,15	19,16	.....	19,77	19,78	19,79	19,80
20,1	20,2	20,3	20,4	20,5	20,6	20,7	20,8	20	20,10	20,11	20,12	20,13	20,14	20,15	20,16	.....	20,77	20,78	20,79	20,80
21,1	21,2	21,3	21,4	21,5	21,6	21,7	21,8	21	21,10	21,11	21,12	21,13	21,14	21,15	21,16	.....	21,77	21,78	21,79	21,80
22,1	22,2	22,3	22,4	22,5	22,6	22,7	22,8	22	22,10	22,11	22,12	22,13	22,14	22,15	22,16	.....	22,77	22,80	22,79	22,80

Expected outcome:

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1	1,10	1,11	1,12	1,1	1,1	1,1	1,1	……	1,7	1,7	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2	2,10	2,11	2,12	2,1	2,1	2,1	2,1	……	2,7	2,7	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3	3,10	3,11	3,12	3,1	3,1	3,1	3,1	……	3,7	3,7	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4	4,10	4,11	4,12	4,1	4,1	4,1	4,1	……	4,7	4,7	4,7	4,8
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5	5,10	5,11	5,12	5,1	5,1	5,1	5,1	……	5,7	5,7	5,7	5,8
6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6	6,10	6,11	6,12	6,1	6,1	6,1	6,1	……	6,7	6,7	6,7	6,8
7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7	7,10	7,11	7,12	7,1	7,1	7,1	7,1	……	7,7	7,7	7,7	7,8
8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8	8,10	8,11	8,12	8,1	8,1	8,1	8,1	……	8,7	8,7	8,7	8,8
9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9	9,10	9,11	9,12	9,1	9,1	9,1	9,1	……	9,7	9,7	9,7	9,8
10, 1	10, 2	10, 3	10, 4	10, 5	10, 6	10, 7	10, 8	10	10,10	10,11	10,12	10, 13	10, 14	10, 15	10, 16	……	10, 77	10, 78	10, 79	10, 80
11, 1	11, 2	11, 3	11, 4	11, 5	11, 6	11, 7	11, 8	11	11,10	11,11	11,12	11, 13	11, 14	11, 15	11, 16	……	11, 77	11, 78	11, 79	11, 80
…..	…..	…..	…..	…..	…..	…..	…..	…	…..	…..	…..	…..	…..	…..	…..	……	…..	…..	…..	…..
18, 1	18, 2	18, 3	18, 4	18, 5	18, 6	18, 7	18, 8	18	18,10	18,11	18 ,12	18, 13	18, 14	18, 15	18, 16	……	18, 18	18, 19	18, 20	18, 21
19, 1	19, 2	19, 3	19, 4	19, 5	19, 6	19, 7	19, 8	19	19,10	19,11	19, 12	19, 13	19, 14	19, 15	19, 16	……	19, 77	19, 78	19, 79	19, 80
20, 1	20, 2	20, 3	20, 4	20, 5	20, 6	20, 7	20, 8	20	20,10	20,11	20, 12	20, 13	20, 14	20, 15	20, 16	……	20, 77	20, 78	20, 79	20, 80
21, 1	21, 2	21, 3	21, 4	21, 5	21, 6	21, 7	21, 8	21	21,10	21,11	21,12	21, 13	21, 14	21, 15	21, 16	……	21, 77	21, 78	21, 79	21, 80
22, 1	22, 2	22, 3	22, 4	22, 5	22, 6	22, 7	22, 8	22	22,10	22,11	22,12	22, 13	22, 14	22, 15	22, 16	……	22, 77	22, 80	22, 79	22, 80

Also, I need to check that if the cell has less than 2 neighbors it dies.