Assignment 1 - Conway’s Game of Life

Design

- World should be an array[44][24] with a visible world of array[40][20]
- Alive cells will be recorded using a 1
- Dead cells will be recorded using a 0

Dead_Live Function:

- Needs to check neighbors of all cells beginning with array[2][2], because we will have a border consisting of 2 cells all around
- We will check neighbors of array[i][j] by evaluating these cells:
    - array[i-1][j-1]
    - array[i-1][j]
    - array[i-1][j+1]
    - array[i][j -1]
    - array[i][j+1]
    - array[i+1][j-1]
    - array[i+1][j]
    - array[i+1][j +1]
- If an alive cell as 2 or 3 neighbors it stays alive, if not it dies.
- If a dead cell has 3 neighbors it comes alive.
- If cell is alive (1), we run a loop that counts all the live cells. If the count is <2 or >3 the cell dies (0). If the the count ==2 || ==3 then the cell stays alive. We pass the result/outcome to a different array at the same location in the array.
- If the cell is dead (0), we run a different loop that counts its live (1) neighbors. If the count == 3, then the cell comes alive (1). If the count is != 3, then the cell stays dead. We pass the result/outcome to a different array at the same location in the array.

- We need to create 2 identical 2D arrays. As each generation passes, alternating arrays get printed to the screen
- Create a generation variable (int gen) that counts each passing generation. Variable is incremented at the end of each generation.

Printing Function

- Function prints alternating arrays by calculating if (gen % 2 == 0) we print x array, else print y array.
- function should take a pointer to array as parameter

Starting Point Function: Return Function that sets starting point

- takes pointer and two ints as parameters
- function should populate arrays based on user selected shapes
- display a menu of shapes; oscillator, a glider, or glider cannon
- based on selection, function returns an integer to main function as a representation of starting shape.

Order in which functions run:

Starting Point Function

Loop -
Printing Function
Dead_Live Function

PseudoCode:

```
Create gen variable, initialize to 0;
Create const vars for arrays;
Create two 2D arrays at [const var][const var]; [44][24]
initialize arrays;

Call Starting Point Function

int startingPoint(ptr, int row, int col)
        cout << which pattern do you want to start with? <<
        create selection variable;
        display patterns with cout
        user chooses pattern and choice is assigned to selection variable (1, 2 or 3)
        using ptr, populate the first array with the pattern
```

Loop From Here

```
Call Printing Function using array1 if gen %2 == 0, else use array2

printingFunction(*gameBoard, int row, int col, &int gen) // takes a pointer and two ints as a
parameter, takes gen int
for loop (int i = 2; i<row-2; i++)
        for loop(int j = 2; j<col-2; j++)
                cout << gameBoard[i][j] << " " <<;
                cout << endl;
increment gen;


Call deadLive function

dead_live(*gameBoard1, *gameBoard2, int row, int col, int gen) // takes two pointers and two
ints as parameters; also takes generation as pointer

create gen variable, initialize with incoming argument;
create counter variable
```

If gen%2!=0 run this loop:

```
for loop(int i = 2; i<row-2; i++)
        for loop (int j = 2; j<col-2; j++)
        {
                if gameBoard1[i][j] == 0
                {
                        if gameBoard1[i-1][ j-1] == 1
                                increment counter
                        if gameBoard1[i-1][ j ] == 1
                                increment counter
                        if gameBoard1[i-1][ j+1] == 1
                                increment counter
                        if gameBoard1[ i ][ j -1] == 1
                                increment counter
                        if gameBoard1[ i ][ j+1] == 1
                                increment counter
                        if gameBoard1[i+1][ j-1] == 1
                                increment counter
                        if gameBoard1[i+1][ j ] == 1
                                increment counter
                        if gameBoard1[i+1][ j +1] == 1
                                increment counter

                        if counter == 3
                                gameBoard2[i][j] == 1
                        else
                                gameBoard2[i][j] == 0

                }

                else

                {
                        if gameBoard1[i-1][ j-1] == 1
                                increment counter
                        if gameBoard1[i-1][ j ] == 1
                                increment counter
                        if gameBoard1[i-1][ j+1] == 1
                                increment counter
                        if gameBoard1[ i ][ j -1] == 1
                                increment counter
                        if gameBoard1[ i ][ j+1] == 1
                                increment counter
                        if gameBoard1[i+1][ j-1] == 1
                                increment counter
                        if gameBoard1[i+1][ j ] == 1
                                increment counter
                        if gameBoard1[i+1][ j +1] == 1
                increment counter

            if counter <2 ll >3
                gameBoard2[i][j] == 0
            else
                gameBoard2[i][j] == 1
        }

    }
```

If gen%2==0 run this loop:

```
for loop(increment row)
    for loop (increment column)
    {
        if gameBoard2[i][j] == 0 (cell is dead)
        {
            if gameBoard2[i-1][ j-1] == 1
                increment counter
            if gameBoard2[i-1][ j ] == 1
                increment counter
            if gameBoard2[i-1][ j+1] == 1
                increment counter
            if gameBoard2[ i ][ j -1] == 1
                increment counter
            if gameBoard2[ i ][ j+1] == 1
                increment counter
            if gameBoard2[i+1][ j-1] == 1
                increment counter
            if gameBoard2[i+1][ j ] == 1
                increment counter
            if gameBoard2[i+1][ j +1] == 1
                increment counter

            if counter == 3
                gameBoard1[i][j] == 1
            else
                gameBoard1[i][j] == 0

        }

        else (cell is alive)

        {
            if gameBoard2[i-1][ j-1] == 1
                increment counter
            if gameBoard2[i-1][ j ] == 1
                increment counter
```

```
                    if gameBoard2[i-1][ j+1] == 1
                            increment counter
                    if gameBoard2[ i ][ j -1] == 1
                            increment counter
                    if gameBoard2[ i ][ j+1] == 1
                            increment counter
                    if gameBoard2[i+1][ j-1] == 1
                            increment counter
                    if gameBoard2[i+1][ j ] == 1
                            increment counter
                    if gameBoard2[i+1][ j +1] == 1
                            increment counter

                    if counter ==2 ll ==3
                            gameBoard1[i][j] == 1
                    else
                            gameBoard1[i][j] == 0
            }

    }
```

End Loop Here

Reflection:

Conway's Game of Life was a very cool stretch in programming for me. The most difficult aspect of the program was processing 2D arrays from another function. To begin, as seen in my pseudocode and design statements, I attempted to adjust certain cells in the arrays by using the array[ ] [ ] call. However, I quickly found that a function, when taking a pointer as an argument, can not access memory this way. Therefore, I had to develop an algorithm that allowed the function to point to the correct place in memory.

The second algorithm I needed to develop was used for the startingPoint function, the function that allows the user to choose which shape to implement, and where to start the shape.
The algorithm needed to be able to receive user submitted coordinates and use them to find the place to input the first cell. Since the cells in a Glider Gun are not capable of being place using a loop (so I think :), I had to measure distances between cells in relation to the first cell place. Doing so allowed me to input variables that would give me the starting coordinates of the pattern.

Now that the program has been constructed, I can say that I would like to know how to use a graphical display rather than a text-based display. I'd also like to implement a more intuitive method for selecting the cell in which to start a pattern. Many examples I saw online used a mouse click to select which cell to begin the game.

In conclusion, this assignment was challenging, enlightening and exciting, and I am looking forward to continuing my understanding of OOP.

Testing

	Input Type	Expected Result	Actual Result
Test 1:	whole number	Program to run as exp.	Program runs as expected
Test 2:	floating pt num	Program w/ truncate and run as expected	Program fails when floating point number is entered
Test 3:	character	unknown result	Program Fails
Test 4:	negative whole num	Program to run as expo.	Program runs as expected

Based on my testing, I found that any whole number entered, negative or positive, caused the program to run properly. However, entering a floating point number or character into the user inputs caused the program to fail and continually run the loops in which the cin was entered.

To fix this issue I had to research how to keep the program from keeping the extra digits in the buffer stream. I found cin.clear() and in.ignore(10000, '\n') to do the trick. Now, after each cin, these two functions clear the input buffer to remove any data that could possible flow into the next cin statement.