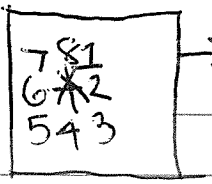
Conway's Game of life: Project 1

- To start the design of the game we are going to have to outline the array matrix that will hold the values.
  - array is 40 x 20
  - 2 for-loops to hold values but won't display anything.
- There will need to be a second array which will hold the locations of the elements "in the game"

- The biggest function we will have to implement is the one that decides if a cell lives, dies, or reproduces. This will have to be done by counting around each element and specifying rules for a new element to be born.
  - This could be done by counting around each element (8 possible coordinates to count)
    - ex:



→ need an array to check the 8 locations and count the neighbors/ empty spots.

  - Based on the array element we also need a variable that will count the number of neighbors, or lack their of.
    - that number will be added in the for loop, and then a variety of if else statements will determine the specific elements fate.

- Since we have now determined what elements will live, die, or reproduce we can implement the rules in it's entirety. We can also do this because we have gone through the array and determined the outcomes.
  - 4+ neighbors = death
  - 0 or 1 neighbors = death
  - 3 live neighbors = new life (must be 3 exactly)

- Now that the array's and conditions are set, we can ask the user to enter their start position and what shape (glider, gun or cannon).

- Observations from tests

One thing I noticed right away is that I needed to develop a way to track my array and it's location in the different functions. Below is a diagram I used when going through each function.

1. start
2. Get Shape
3. Creat the shape in array
4. Compare to other elements
5. Copy the new array to the dd
6. Display the array

this loops until the user specifies to end the program

- One of the tests I forgot to address was the "edge test". Prior to testing I expected the shape to start to deform when it hit the edges, and turn to mush. When testing it would sometimes bounce back. This is different then what I had originally thought.

- The edge case I thought I could just make another array that was about twice as big as the first so when the "live" cells went pass the edge they would just continue into the 2-D array and not be visible. This ended up being the way to go but definitely has its limits. I'm interest in seeing other students solutions to this problem.

- Another issue that took a lot of time working on was making sure the game could be played as long as the user wanted to. The biggest problem with this was the input validation I had to go through in order to ensure the user was entering the correct letters, strings, variables, etc. And if they did not that they were prompted to enter the correct information until they did.

- The function that compared the two arrays worked about the same that I thought it would BUT getting the EXACT array values to count their neighbors was trickier than I had thought originally. Often times it was something simple like using == when it needed to be =.
    - Sometimes in the original document (pre-test) I doubled the arrays and left a value out. It took some trial and error but eventually was able to get working.