**Conway’s Game of Life:**

**Documentation:**

Conway’s Game of Life was made by John Horton Conway, it is an evolutionary game to represent a simplistic model of how life functions. Game of Life is a cellular automaton, which means it is a grid of cells that change states due to its surroundings. In Game of Life, the board is divided into a grid, and every cell in the grid is considered either alive or dead. When we loop through every cell it is considered a generation, whether a cell lives are dies into the next generation, it is dependent on it’s neighbors. A cell can die through either isolation or over population. An isolation death is considered when the cell has 1 or fewer living neighbors. A cell dies through over population when it has 4 or more living neighbors. A cell could become alive if it has exactly three living neighbors. These simple rules that make up this game makes this one of the most popular and famous cellular automata.

**How I made it:**

Looking at my code, I can understand why someone might find this overwhelming, but if we go from top to bottom it’s easy to see that that the only concept we haven’t covered are canvases and arrays—I had 4 years of Java knowledge behind me (sadly we didn’t even have to make custom made objects). I defined global variables right above methods where they are mention. I first define the DEAD and ALIVE constants and then some variables about like grid size and canvas size so the canvas can be more flexible (so I could easily change the gird size and canvas size). I then create the references that will soon be the canvases context graphics and the grid. IntGame() stands for Initialize game, this is where I initiate my graphics grid (and add a mouse event listener). The following few functions sets up the board, whether it’s clearing a board or setting presets. Following those functions, we have interval management, this is where we make timers and stop them, it’s important to stop any interval that is running before making a new one. Note every time we do any major changes to the board, we pause the game. After the timer functions, we then have one function designated to drawing the board, this is simply done with a nested for loop (to loop through the 2d array we have) and a simple if statement asking whether a cell is dead or alive and change the color relative to the answer. And lastly we have or runGame() method, which is the heart of everything. When creating a new generation, we have to act upon a different grid, a temporary grid if you will. If we acted on the same grid we would find that the output won’t be as expected. This is because if we switch a cells state from dead to alive, the cell immediately next to it will act as if it was always that state (kind of awkward to explain). After creating a temporary grid, we nested for loop through the original grid looking at reach cell. When looking at a cell we want to look at all 8 adjacent cells and count the ones that are alive. I did this with another constant variable that stores off-set points. Using those offset points and helper function that makes sure the point won’t give an index out of bound error, we can count every cell’s neighbors. Once we get a number we can use Conway’s game of life rules to determine if a cell in the temporary grid should be alive or dead. Once we fully filled out the temporary, we can now assign it to our original grid, and that’s it. Lastly, we have a little function that allows the user to turn on certain cells with a mouse click. This is easily done by getting the clicks X and Y coordinate and dividing it by the cell’s sizes. The HTML code is pretty self-explanatory.

**Manuel:**

My rendition of Conway’s Game of Life allows users to step through generations, set a slow interval of steps and a some what fast—10 steps per second—interval. You are also able to load pre sets that I deemed cool enough that the user had to see.

**Paused/ Running Tag:** The tag, located right above all the buttons, tells us whether the game is running the step procedure on an interval.

**Step Button:** The step button loads one full generation.

**Slow Button:** The slow button sets an interval of step procedure, about once every second.

**Run Button:** Similar to the slow button, the run button creates a faster interval of step procedure, about 10 times per second.

**Pause Button:** The pause button will stop any running intervals

**Clear Board:** The clear button sets every cell on the board to state dead.

**Option Tab:** The option tabs allow the user to load certain presets: Random; Gospel Gun; and a Straight Line. These are all presets that I thought were worthy, for their coolness, to have their own preset so the user doesn’t have to click in all the cells.

**Load Board Button:** The load board button loads whatever option you have selected from the option tabs

**Generation Tag:** This tells the user what generation they are. This could be helpful to find out how long it takes for everything to settle or how many generations it takes for the Gospel Gun to loop back into the starting position.

**Canvas:** The canvas shows us the board that the game is operating on.

**Clicking the Canvas:** When the user clicks the canvas, the cell that the course was hover over would switch its state.

**Website:** https://www.cs.drexel.edu/~mp3492/Project2.html