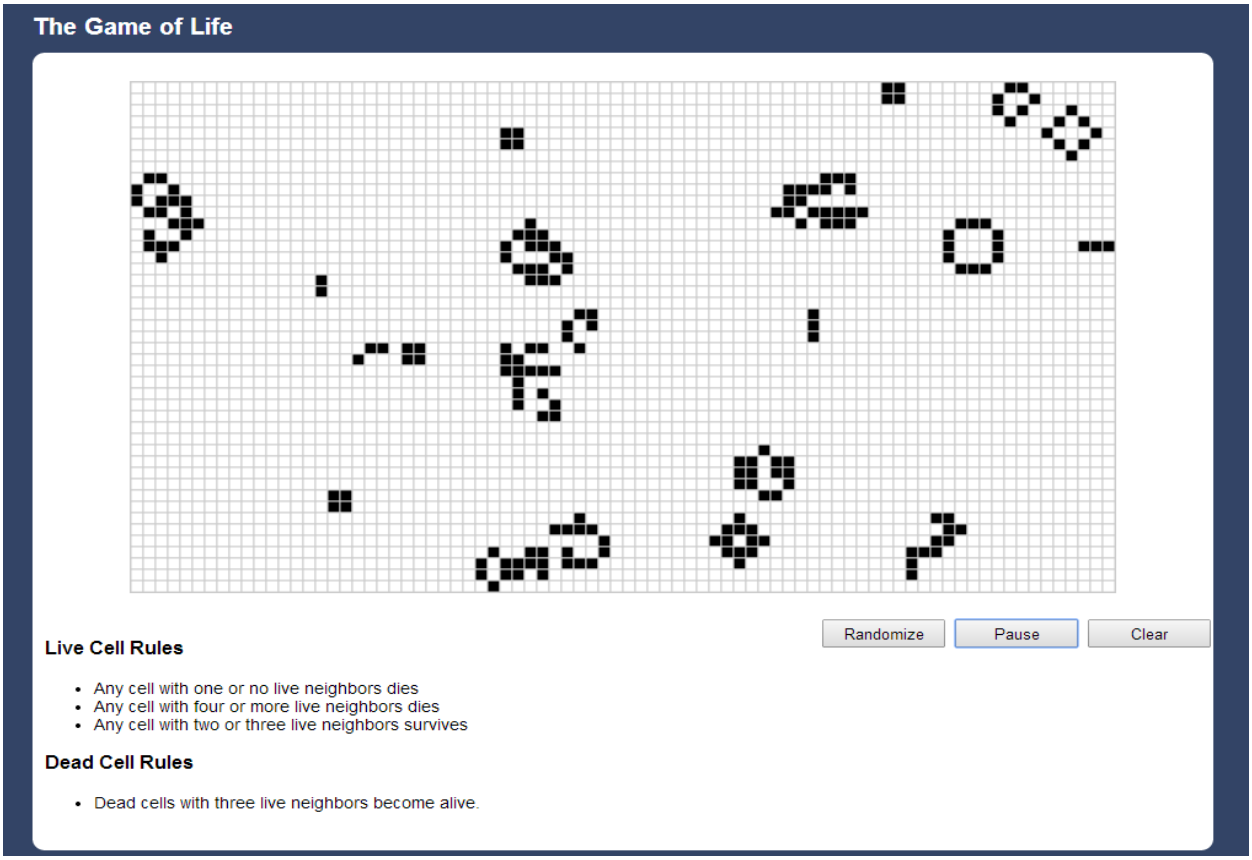
**Lab 9 – Conway's Game of Life**

Life is a simulation game where the player sets up the initial board conditions, and then observes how the system naturally evolves based on a simple set of rules.

We're going to use the canvas element and an object oriented design to recreate this game in HTML5.



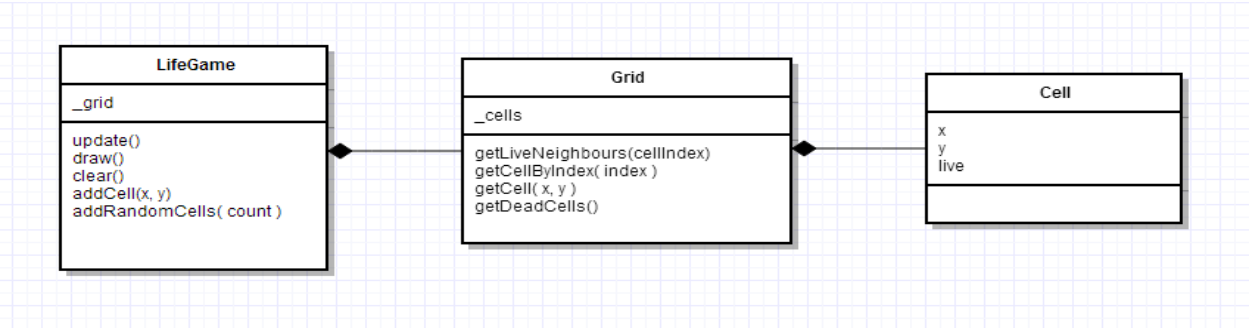

**Step 1 – Create our canvas element**

In our main content div – create a canvas element with a width of 800 and height of 450. Be sure to provide a fallback message for clients without an HTML5 browser.

```
<canvas id="gameCanvas" width="800" height="450">
       Your browser does not support HTML5
</canvas>
```

**Step 2 – Planning the Software**

Break the application/problem up into manageable pieces. The following is the design I came up with




The black diamond indicates a containment relationship – LifeGame contains a Grid, and a Grid is made up of (contains) Cells

**Step 3 – Create the Cell type**

Cells are our most basic data type – they have an x/y position and a flag to tell if they're alive

```
/**
    An element of a grid
*/
function Cell(x, y) {
    this.x = x;
    this.y = y;
    this.live = false;
}
```

**Step 4 – Create the Grid Type**

The Grid will have an array of cells – these will be created when the grid is created

```
/**
    A 2 dimensional array of cells
*/
function Grid( cellsPerRow, cellsPerColumn ){
    var _cells = [];
    var _totalCells = cellsPerRow * cellsPerColumn;

    //Create the cells
    for (var i = 0; i < _totalCells; ++i) {
        var x = i % cellsPerRow;
        var y = Math.floor( i / cellsPerRow );
        _cells.push( new Cell( x, y ) );
    }

    /**
        Get the cell at the provided index
    */
```

```
    this.getCellByIndex = function (index) {
        return _cells[index];
    }
}
```

**Step 5 – Create the Application class**

The LifeGame type will manage the active grid as well as taking the responsibility of rendering it. We'll provide it with the width, height, and cellSize we want to use for our gird, as well as the canvas that we're going to draw to.

```
function LifeGame(width, height, cellSize, canvas) {
    //Private
    var _cellsPerRow = width / cellSize;
    var _cellsPerColumn = height / cellSize;
    var _totalCells = _cellsPerRow * _cellsPerColumn;
    var _ctx = canvas.getContext('2d');
    var _grid = new Grid(_cellsPerRow, _cellsPerColumn);

    this.clear = function () {
        _grid = new Grid(_cellsPerRow, _cellsPerColumn);
    }
}
```

**Step 6 - Build our draw function within the application class**

Clear the screen and then draw a rectangle for each cell in the grid

```
    /**
        Draw the current grid
    */
    this.draw = function () {
        //Clear the background
        _ctx.clearRect(0, 0, _ctx.canvas.width, _ctx.canvas.height);
        _ctx.lineWidth = 1;
        _ctx.strokeStyle = "#ccc";

        //Draw the cells
        for (var i = 0; i < _totalCells; ++i) {
            var cell = _grid.getCellByIndex(i);

            if (cell.live)
                _ctx.fillStyle = "black"; //draw black
            else
                _ctx.fillStyle = "white";//draw white

            //Render the cell
            _ctx.beginPath();

            _ctx.rect(cell.x * cellSize, //X location of top left
                cell.y * cellSize,       //Y location of top left
                cellSize,                //width
                cellSize);               //height
            _ctx.fill();
            _ctx.stroke();
```

```
    }
}
```

**Step 7 – Get Live Neighbours**

In order to enforce our game rules, we'll need a way to discover the number of living neighbours to a given cell – let's fill in the getLiveNeighbours() function of our Grid class

```
/**
    Get all living cells next to the provided index
*/
this.getLiveNeighbours = function (cellIndex) {
    var liveNeighbours = 0;
    var cell = _cells[cellIndex];

    var startX = Math.max(cell.x - 1, 0);
    var startY = Math.max(cell.y - 1, 0);
    var endX = Math.min(cell.x + 1, cellsPerRow - 1);
    var endY = Math.min(cell.y + 1, cellsPerColumn - 1);

    for (var x = startX; x <= endX; ++x) {
        for (var y = startY; y <= endY; ++y) {

            //Ignore the cell being considered
            if (x == cell.x && y == cell.y)
                continue;

            var cellIndex = x + y * cellsPerRow;
            if (_cells[cellIndex].live)
                liveNeighbours++;
        }
    }

    return liveNeighbours;
}
```

**Step 8 – Updating and Enforcing the Game Rules.**

In our LifeGame class, create a temporary empty grid of the same dimensions of our active grid. Fill it with the results of applying the game rules to the current generation. Then replace the active grid with the new grid and redraw the game world.

```
/**
    Advance the game a generation
*/
this.update = function () {
    //create a new grid based on the old one
    var nextGeneration = new Grid(_cellsPerRow, _cellsPerColumn);

    //Apply the rules to all members of the current generation
    for (var i = 0; i < _totalCells; ++i) {
        var oldCell = _grid.getCellByIndex(i);
        var newCell = nextGeneration.getCellByIndex(i);
```

```
            var neighbours = _grid.getLiveNeighbours(i);

            //Apply living rules
            if (oldCell.live) {
                //Cells die unless they have exactly 2 or 3 neighbors
                newCell.live = neighbours > 1 && neighbours < 4;
            }
            else {
                //Check if this cell should come to life
                if (neightbours == 3)
                    newCell.live = true;
            }
        }

        //Replace the old generation with the new one
        _grid = nextGeneration;
        _this.draw();
    }
```

**Step 9 – Allow the user to click to add cells**

Add a Click event listener to your canvas element

```
canvas.addEventListener("click", onCanvasClick);
```

Get convert the click coordinates to your 2D grid cell coordinates, and spawn a new cell at that location

```
function onCanvasClick(evt) {
       var rect = canvas.getBoundingClientRect();

       //Get the x/y position of the cell clicked on
       var cellX = Math.floor((evt.clientX - rect.left) / CELL_DIMENSION );
       var cellY = Math.floor((evt.clientY - rect.top) / CELL_DIMENSION);

       gameWorld.addCell(cellX, cellY);
       gameWorld.draw();
}
```

**Step 10 – Add Random Cells**

When the user clicks the random button – add a bunch of random cells to the game world

First add a function to our Grid class that will get all available (dead) cells

```
    /**
        Collect all dead cells
    */
    this.getDeadCells = function () {
        var deadCells = [];

        for (var i = 0; i < _totalCells; ++i) {
            if (!_cells[i].live)
```

```
                deadCells.push(_cells[i]);
        }

        return deadCells;
    }
```

Randomly bring a subset of them to life – expose this function in our LifeGame Class

```
    /**
        Make the provided number of cells come to life
    */
    this.addRandomCells = function (count) {
        var deadCells = _grid.getDeadCells();

        for (var i = 0; i < count && deadCells.length > 0; ++i) {
            var randomIndex = Math.floor( Math.random() * deadCells.length );

            deadCells[randomIndex].live = true;
            deadCells.splice(randomIndex, 1);
        }
    }
```

**Step 11 – Start, Pause, and Clear the Game by adding click event handlers to the correct buttons**

Add the following event handlers to the appropriate buttons

```
var interval = -1;

/**
       reset the game board
*/
function clearGame() {
       pauseGame();
       gameWorld.clear();
       gameWorld.draw();
}


/**
       Start game click handler
*/
function startGame() {
       if (interval == -1) {
              interval = setInterval(gameWorld.update, 300);
              startButton.innerHTML = "Pause";
       }
       else
              pauseGame();

}


/**
       Pause the game
*/
```

```
function pauseGame() {
	if (interval != -1) {
		clearInterval(interval);
		interval = -1;
	}

	startButton.innerHTML = "Start";
}
```