Conway's Game of Life

Report written by :

Widad Mokhtar Saidia

Code Explanation

```
class Game {
            constructor(rows, cols, cellSize) {
                this.rows = rows;
                this.cols = cols;
                this.cellSize = cellSize;
                this.grid = this.createEmptyGrid();
                this.isRunning = false;
            }

            createEmptyGrid() {
                return Array(this.rows).fill().map(() => Array(this.cols).
fill(false));
            }
```

The constructor initializes the game's grid (defines rows, cols and cell's size), at first the grid is empty that's why it draws an empty grid (blank cells), and sets game status to false (it's not running)

The method **createEmptyGrid** sets all cells status to false meaning every single cell is now dead, each row has array of columns, and each column has array of cells, each one of these is set to false

```
randomize() {
                this.grid = this.grid.map(row =>
                    row.map(() => Math.random() > 0.85)
                );
            }
```

Randomly sets each cell value to true (alive) or false (dead) by verifying the condition “if the generated random number is bigger than 0.85 then make this cell status is set to alive otherwise dead”

```
drawGliderGun() {
                // Write your function here
                const metrix = [
                    [26, 1],
                    [24, 2], [26, 2],
                    [14, 3], [15, 3], [22, 3], [23, 3], [36, 3], [37, 3],
                    [13, 4], [17, 4], [22, 4], [23, 4], [36, 4], [37, 4],
                    [2, 5], [3, 5], [12, 5], [18, 5], [22, 5], [23, 5],
                    [2, 6], [3, 6], [12, 6], [16, 6], [18, 6], [19, 6], [24
, 6], [26, 6],
                    [12, 7], [18, 7], [26, 7],
                    [13, 8], [17, 8],
                    [14, 9], [15, 9]
                ]

                metrix.forEach(([x, y]) => {
                    this.grid[y][x] = true;
//the cell exist so renderer will fill it
                });
            }
```

Sets certain cells value to true based on a predefined pattern called “GliderGun”

These two methods are similar to previous one except first creates Pulsar pattern, second creates Penta Decathlon pattern:

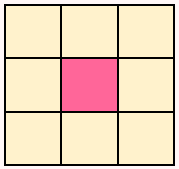
```
drawPulsar() {
                const p = [[4,2],[5,2],[6,2],[10,2],[11,2],[12,2],[2,4],[7,4
],[9,4],[14,4],[2,5],[7,5],[9,5],[14,5],[2,6],[7,6],[9,6],[14,6],[4,7],[5,7
],[6,7],[10,7],[11,7],[12,7],[4,9],[5,9],[6,9],[10,9],[11,9],[12,9],[2,10],[
7,10],[9,10],[14,10],[2,11],[7,11],[9,11],[14,11],[2,12],[7,12],[9,12],[14,
12],[4,14],[5,14],[6,14],[10,14],[11,14],[12,14]];
                p.forEach(([x, y]) => {
                    this.grid[y][x] = true;
//the cell exist so renderer will fill it
                });
            }
```

```
drawPentaDecathlon() {
                const p = [
                [5, 4], [6, 4], [7, 3], [7, 5], [8, 4], [9, 4], [10, 4], [11
, 4], [12, 3], [12, 5], [13, 4], [14, 4]
                ];

                p.forEach(([x, y]) => {
                    this.grid[y][x] = true;
                });
            }
```

```
countNeighbors(x, y) {
                //check how many filled (existed) neighbord a cell has
                let count = 0;
                for (let dy = -1; dy <= 1; dy++) {
                    for (let dx = -1; dx <= 1; dx++) {
                        if (dx === 0 && dy === 0) continue;
                        const nx = (x + dx + this.cols) % this.cols;
                        const ny = (y + dy + this.rows) % this.rows;
                        if (this.grid[ny][nx]) count++;
                    }
                }
                return count;
            }
```

This method counts number of neighbors of a specific cell, it iterates throw all possible cells (yellow ones) around a single cell (the pink one), it checks if each one of these cells are inside the grid or not, when **dx = dy = 0** this means we are in current cell so we skip it using continue statement, it calculates coordinates of those 8 possible cells, if they exist in the grid it increments **count++,** otherwise does nothing.



```
update() {
                const newGrid = this.createEmptyGrid();
//empty the grid first
                for (let y = 0; y < this.rows; y++) {
                    for (let x = 0; x < this.cols; x++) {
                        const neighbors = this.countNeighbors(x, y);
                        const isAlive = this.grid[y][x];
                        newGrid[y][x] = (isAlive && (neighbors === 2 ||
 neighbors === 3)) ||
                                                (!isAlive && neighbors === 3);

                    }
                }
                this.grid = newGrid;
            }
```

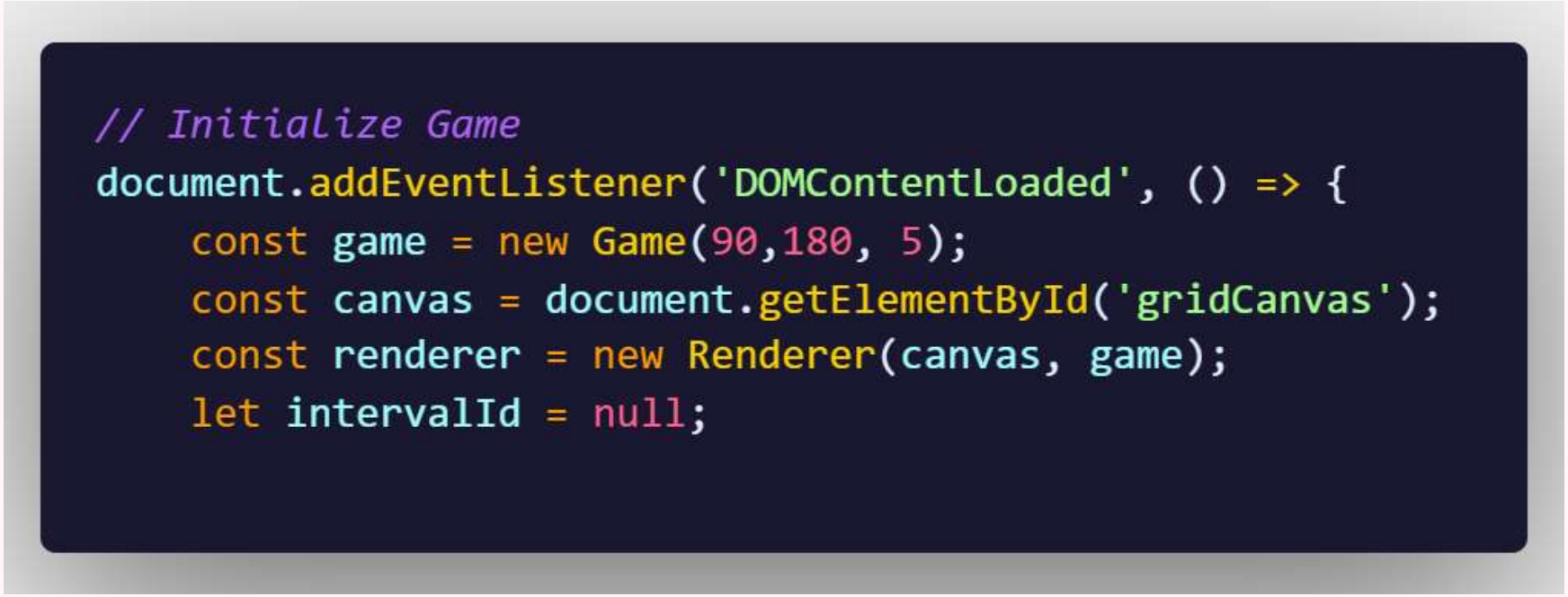
This method calculates number of neighbors of a cell (by calling countneighbor method), then sets its status to alive **if** the cell is alive and has two or three neighbors, **or if** it is dead and has exactly 3 neighbors.

```
class Renderer {
            constructor(canvas, game) {
                this.canvas = canvas;
                this.ctx = canvas.getContext('2d');
                this.game = game;
                canvas.width = game.cols * game.cellSize;
                canvas.height = game.rows * game.cellSize;
            }
```

This constructor initializes the canvas with game parameters previously set.

```
draw() {
                this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height
);//clear all rect of canvas
                for (let y = 0; y < this.game.rows; y++) {
                    for (let x = 0; x < this.game.cols; x++) {
                        if (this.game.grid[y][x]) {
//if cell value is true then fill it with color
                            this.ctx.fillStyle = '#AC1754';//bg color of cell
                            this.ctx.fillRect(
                                x * this.game.cellSize, //start x of cell
                                y * this.game.cellSize,//start y of cell
                                this.game.cellSize - 1,//width
                                this.game.cellSize - 1//height
                            );
                        }
                    }
                }
            }
```

This method iterates throw rows and columns of game's grid, if cell status is true it calculates its coordinates (x and y, height and width) and fill it with color, otherwise does nothing.



This event is performed after all html script is loaded, it creates a game and renderer instance to start the game.

```
// Button Event Handlers
            document.getElementById('startBtn').
addEventListener('click', () => {
                game.isRunning = !game.isRunning;
                document.getElementById('startBtn').
textContent =
                    game.isRunning ? 'Stop' : 'Start';
                if (game.isRunning) {
                    intervalId = setInterval(() => {
                        game.update();
                        renderer.draw();
                    }, 1);
                } else {
                    clearInterval(intervalId);
                }
            });
```

This method checks if game is running, if yes it updates the grid by calling the update method discussed previously.

```
document.getElementById('clearBtn').addEventListener('click',
 () => {
                game.grid = game.createEmptyGrid();//done
                renderer.draw();
            });
```

When clear button is clicked this method is called, it clears the grid by calling the “create empty grid” method discussed previously.

```
document.getElementById('randomBtn').addEventListener('click'
, () => {
                game.randomize();
                renderer.draw();
            });
```

When random button is clicked this method is called, it generates random cells by calling randomize method.

```
document.getElementById('gliderGunBtn').addEventListener('
click', () => {
                game.drawGliderGun();
                renderer.draw();
            });
```

When glider gun button is clicked, this method is called, it draws glider gun pattern by calling its method.

```
document.getElementById('pulsarBtn').addEventListener('click'
, () => {
                game.drawPulsar();
                renderer.draw();
            });
```

When pulsar button is clicked, this method is called, it draws pulsar pattern by calling its method.

```
document.getElementById('pentaDecathlonBtn').addEventListener
('click', () => {
                game.drawPentaDecathlon();
                renderer.draw();
            });
```

When Penta decathlon button is called this method is called, it draws Penta decathlon pattern by calling its method

```
// Initial Draw
            renderer.draw();
```

Initialize the game.