Conways Game of Life implemented in React by Jonathan Jamel Holloway

The hardest part I believe is the start. How will the data structure that holds the grid be implemented? What type of data structure will it be? My approach is just that and maybe not the best practice however, I decided to use objects. To get to the previous node I could decrement the index as long as the index was not at 0. To get to the next node I could increment as long as the index isn’t the max which is 14 (if I have a 15 by 15 grid). I could use these same techniques to get to the top and bottom locations as long as I use integers as my keys. The reason why I am thinking of a direct access to these other locations is because of the rules.

From <https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life> the universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states alive or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

I also consider what we need done, initially there has to be a way for a grid cell to be turned on or off and for this to be represented.

In css I immediately create two classes:

.onDiv {

background: yellow;

}

.offDiv {

background: white;

}

For my state I am using arrays inside of a the state like so:

class App extends Component {

state = {

0: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

1: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

2: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

3: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

4: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

5: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

6: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

7: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

8: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

9: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

10: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

11: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

12: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

13: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

14: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],

}

render() {

return (

<div className="container">

</div>

);

}

}

export default App;

The next step is to create the grid by looping over the state. This is where I consider there to be a problem. What if I want more items on the state? It might be better if I made the arrays sit inside its own property on the state. So I made the following adjustment:

state = {

matrix: {

0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

1: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

2: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

3: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

5: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

6: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

7: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

8: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

10: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

11: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

12: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

13: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

14: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

}

};

The grid in my opinion just like most of this project could be implemented in many ways however I am going to use div’s initially. Div’s with borders to show that it is a grid that feature a conditional check and a function. First I will implement the grid.