Conway's Game of Life - Project Explanation

# Introduction

This project is an implementation of Conway's Game of Life using React and JavaScript. The application simulates a 20x20 grid where each cell can be either alive or dead. By applying a set of predefined rules, the grid evolves from one generation to the next. In this document, I will explain how the project was implemented, covering the main components, logic, and features.

# Step 1: Setting up the Project

To begin, I set up a React environment by running the following commands in the terminal:

npx create-react-app conways-game  
cd conways-game  
npm start

This initialized a basic React application. From there, I began building the grid and implementing the functionality required for the Game of Life.

# Step 2: Creating the Game Board

The board is a 20x20 two-dimensional array. Each cell in the array can either be alive (represented by 1) or dead (represented by 0). To generate this board, I created a function called createEmptyBoard() that initializes a grid of dead cells.

The state of the board is managed using the useState hook in React. I also added the ability to toggle the state of each cell between alive and dead by clicking on it. This interaction is handled in the handleCellClick() function, which updates the board based on user input.

# Step 3: Applying the Rules of Life

The game follows these four rules, which determine how cells evolve:  
1. Any live cell with fewer than two live neighbors dies (underpopulation).  
2. Any live cell with two or three live neighbors survives.  
3. Any live cell with more than three live neighbors dies (overpopulation).  
4. Any dead cell with exactly three live neighbors becomes a live cell (reproduction).

I implemented these rules using the nextGeneration() function, which checks the neighbors of each cell and applies the appropriate changes. A helper function called countNeighbors() was used to count the number of live neighbors for any given cell.

# Step 4: Adding Play/Stop Controls

To control the simulation, I added Play and Stop buttons. These buttons toggle the isPlaying state, which determines whether the game is running or paused.  
When the Play button is clicked, the game advances by one generation every second (1 frame per second). This is achieved using the useEffect hook, which runs the nextGeneration() function repeatedly as long as the game is in the 'playing' state.  
If the Stop button is clicked, the game pauses, and the cells retain their current states until the game is resumed or reset.

# Step 5: Clearing the Board

In addition to the Play/Stop buttons, I implemented a Clear button that resets the entire board to its initial state (all dead cells). This functionality is handled in the handleClear() function, which resets the board using the createEmptyBoard() function.

# Step 6: CSS Styling

I used CSS to style the game board and cells, ensuring that the interface is easy to interact with. The board is displayed as a grid of square cells. Alive cells are displayed in black, while dead cells are displayed in white. Clicking on a cell toggles its color, providing visual feedback on its current state.

Here is an example of the CSS I used for styling the cells:  
  
.cell {  
 width: 20px;  
 height: 20px;  
 border: 1px solid #ddd;  
 cursor: pointer;  
}  
  
.cell.alive {  
 background-color: black;  
}  
  
.cell.dead {  
 background-color: white;  
}

# Step 7: Screenshots

I have attached screenshots in the ZIP file, demonstrating key stages of the project, including the following:  
1. Initial Board Setup: A 20x20 grid of dead cells.  
2. Toggling Cells: Cells change between alive and dead when clicked.  
3. Play/Stop Functionality: The game runs and stops as expected.  
4. Next Generation Updates: The grid evolves based on the rules of the game.

# Conclusion

This project successfully implements Conway’s Game of Life using React and JavaScript. The grid updates according to the game’s rules, and the Play/Stop buttons allow users to control the simulation. The project demonstrates the use of React hooks like useState and useEffect to manage component states and interactions.

# Files Submitted

• The entire src folder, which contains all the components and logic for the game.  
• Screenshots showing the board, cell interactions, and gameplay.  
• This Word document explaining the project in detail.  
  
Thank you for reviewing my work!