**Topic - Conway's Game of Life**

Implement the Game of Life using HTML/CSS, php, MySQL, Java Script, along with any other libraries

**PLAGIARISM DISCLAIMER**

**We are fully aware of many sources are available and we will check**

**Acknowledgement should always be made when secondary sources are used. Failure to acknowledge sources will be penalized and any substantial plagiarism – for example, unacknowledged quotation that extends for more than a few lines, or extended close paraphrasing of a critical work – will automatically be reported to the university disciplinary board and can lead to suspension.**

**Objective:** The game of life is a grid of cells where each cell is in the state of being alive or not. The next generation of the game depends on the current generation and the following rules:

The game consists of a grid of cells, each of which can be alive or dead. For every cycle of the game, the cells can be turned on or off based on the following rules:

1. Any live cell with fewer than two live neighbors dies, as if caused by under population.

2. Any live cell with more than three live neighbors dies, as if by overcrowding.

3. Any live cell with two or three live neighbors’ lives on to the next generation.

4. Any dead cell with exactly three live neighbors becomes a live cell.

5. If a dead cell has exactly three live neighbors, it comes to

6. If a live cell has less than two live neighbors, it dies

7. If a live cell has more than three live neighbors, it dies

8. If a live cell has two or three live neighbors, it continues living. life - Therefore by repeating the cycle over and over, these simple rules create interesting, often unpredictable patterns.

9. Create an Administrator account who can control all the users

records for the table using the FORM PROCESSING functionalities.

Design Specifications

1. Create a variable sized table. (Grads use DHTML)

2. The background color of the cells will determine life.

3. The cells can be turned on or off with the mouse.

4. Display the current population (Grads only) and generation.

5. Create a button for each of the following functions (using mouse events such as clicks, drags i.e.):

1. Start the game
2. Stop the game
3. Increment 1 generation
4. Increment 23 generations
5. Reset the game (Population=0 or other, Generation=0).
6. Pattern button/Drop down selection to (Animate selection)
7. Create a login profile for each user using registration and auto link relocation to the sign-in.

* HTTP basic **authentication**.
* Form-based **login authentication**.
* Client certificate **authentication**.
* Mutual **authentication**.
* Digest **authentication**.

1. Store the name of players in a DB table and keep track of any user events stats, i.e., time spent, number of games played.

6. Undergraduates (Please choose 3 patterns --- (1) pattern from the Still life and Two Oscillators, Gliders are welcome and will be considered as a Bonus)

7. **Graduates only - Please create Random Populations (choose 1 pattern from the Still life (2) patterns, Oscillators and (1) pattern Gliders). Graduates only use any demo patterns that will demonstrate the following behavior: {Create any type of animated lit points that will turn off if there are fewer than two or more than three surrounding lit points. An unlit point turns on if there are exactly three lit neighbors}**

8. Implement the use of the React.js and develop a component that will display the beginning state of the GRID and the ending state of the GIRD (real-time update)

Component states are what give React its name. Any time a component’s state changes, its “render” function is called, updating the output. Essentially, each component “reacts” to changes, which is handy for any user interface. Data stored in a state should be information that will be updated by the component’s event handlers (changes that should update in real time as the interface is being used).

**Conway's Game of Life — Project Features Overview**

**🔹 1. Interactive Game Grid**

* Create a dynamic, resizable grid using HTML/CSS/JavaScript.
* Each cell has two states: **alive** (colored) or **dead** (blank/gray).
* Mouse click toggles cell state.
* Responsive design using **Bootstrap Grid System** or **Flexbox**.

**🔹 2. Game Control Panel**

**Buttons include:**

* ▶️ **Start Game** – Runs the game continuously by applying rules.
* ⏸ **Stop Game** – Halts the game loop.
* ⏭ **Next Generation** – Manually run one generation.
* 🔁 **+23 Generations** – Run a fast forward of 23 generations.
* ♻️ **Reset** – Resets grid, population, and generation counter.
* 🧬 **Load Pattern** – Dropdown to load:
  + Still Life: Block, Boat, Beehive
  + Oscillators: Blinker, Beacon
  + Bonus: Glider, Gosper Glider Gun (for extra credit)

**🔹 3. Form Processing & User Authentication (Required)**

* Registration and Login forms implemented in **PHP**.
* Use **MySQL database** to store:
  + User ID
  + Username/Password (hashed)
  + Game session info (start time, score, generations)
* Session management to maintain login state.
* Logout function.

**🔹 4. Database Integration**

**MySQL Tables:**

* users: ID, name, email, password
* Game Sessions: Session ID, User ID, Start Time, End Time, Generations.
* patterns: optional for saved patterns
* admin\_users: for admin access

**Features:**

* Store and retrieve, play history, pause states.

**🔹 5. Admin Panel**

* Admin account login.
* CRUD interface to:
  + View all users
  + Edit user info
  + Delete users
  + View all game sessions with filters
* Admin actions performed through PHP forms (validated).

**🔹 6. Design Specifications**

| **Requirement** | **Implementation** |
| --- | --- |
| **Grid System** | Dynamic table for cells |
| **Layout & Design** | Bootstrap/Flexbox layout, styled buttons |
| **Color/Text** | Alive/dead color schemes; readable fonts |
| **Navigation** | Nav bar with links: Home, Game, Login, Dashboard |
| **Forms & Buttons** | Bootstrap-styled; client/server-side validation |
| **Responsive** | Mobile-first design with media queries |
| **Style Guide** | Organized with reusable classes/components |
| **Delivery** | Clean UI, functional logic, and good design |

**🔹 7. Gameplay Features**

* Implements all 4 Conway rules accurately.
* Handles edge cases (e.g., grid edges).
* Keeps track of:
  + **Generation number**
  + **Current population count** (for grads)
* Optional animation speed control.

**Milestones for the project**

**🚀 Milestone 1: Project Setup & Team Coordination**

* Choose team members and assign roles (SCRUM master, frontend dev, backend dev, etc.)
* Decide on your game concept and platform name
* Create project repository on GitHub and set up collaboration
* Design wireframes or sketches for the homepage, dashboards, and forms

**🧑‍💻 Milestone 2: User Registration, Login & Roles**

* Implement user registration and login pages
* Add password hashing and form validation
* Create sessions for user login
* Implement role-based redirects (Player vs Admin)
* Create basic dashboards for each role

**🎮 Milestone 3: Player Dashboard & Game Features**

* Build Player Dashboard UI
* Display game stats, username, avatar.
* Implement start/join game logic (mocked or functional)
* Add a matchmaking or friend invitation feature
* Allow players to add items to a Wishlist

**📊 Milestone 4: Admin Dashboard & Analytics**

* Create Admin Dashboard layout
* Show some type of game analytics (users, games played, top users)
* Use data visualization (charts/graphs)
* Add admin actions: suspend/ban user

**📽️ Milestone 6: Final Polish, Testing & Presentation**

* Finalize all pages and features
* Conduct group testing & fix bugs
* Create presentation slides (with screenshots & code snippets)
* Record and upload a YouTube presentation
* Upload code to GitHub and deploy on CODD
* Submit all required links and documentation on iCollege

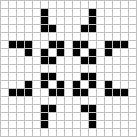
**Fundamental Rules**   
  
The game is designed around the following simple rules:  
  
1) Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.  
2) Any live cell with more than three live neighbors dies, as if by overcrowding.

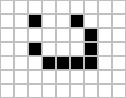
http://web.stanford.edu/~cdebs/GameOfLife/img/dead.png  
3) Any live cell with two or three live neighbors’ lives on to the next generation.http://web.stanford.edu/~cdebs/GameOfLife/img/livecell.png  
4) Any dead cell with exactly three live neighbors becomes a live cell.

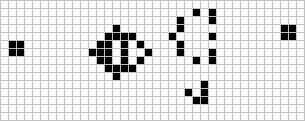
http://web.stanford.edu/~cdebs/GameOfLife/img/deadcell.png  
The operation of the game starts with an initial configuration on a two-dimensional grid. This infinite square grid consists of cells with two possible states, alive or dead. Each cell has eight neighbors, namely the eight cells that touch it. The game operates in iterations, called ticks. Each tick applies the four rules of the game to every cell on the board simultaneously.

**PATTERNS ARE AS FOLLOWS-**

**Still Life**   
  
These are stable patterns that do not change and can be used to build critical solid parts of complex patterns. These patterns stay in one state which enables them to store information or act as solid bumpers to stop other patterns or keep other unstable patterns stable. Examples of still life include:   
  
1) **The Block**  
2) **The Boat**  
3) **The Loaf**  
4) **The Beehive**  
     
  
**Block, Boat, Loaf, Beehive** (from left to right).

Oscillators   
These patterns are more complex and change over a specific number of ticks. They repeat their pattern infinitely. The basic oscillators have periods of two or three ticks, but complex oscillators have been discovered with periods of twenty or more ticks. These oscillators are very useful for setting off other reactions of bumping stable patterns to set off a chain reaction of instability. The most common period-2 oscillators include:  
  
1) The Blinker   
2) The Beacon   
3) The Toad   
4) The Pulsar  
     
  
**Blinker, Beacon, Toad, Pulsar (from left to right).**

Gliders and Spaceships   
  
The spaceship is a pattern that moves, returning to the same configuration but shifted after a finite number of generations. The glider is an example of a simple spaceship and its generations each consist of five live cells. The glider has a period of four and moves diagonally one cell every four generations. It moves at one-quarter the speed of light.   
  
Other examples of simple spaceships include lightweight, medium weight, and heavyweight spaceships. They each move in a straight line at half the speed of light.  
  
      
Glider, Lightweight Spaceship (from left to right).

**Guns**   
  
Guns are repeating patterns which produce a spaceship after a finite number of generations. The simplest gun, called the Gosper glider gun, produces a glider every 30 generations. This fascinating pattern was discovered in 1970 by Bill Gosper. Through careful analysis and experimental testing he developed a pattern which emitted a continuous stream of gliders. Since 1970 researchers and freelance experimenters have discovered hundreds of new patterns and have built thousands of intricate machines and devices using these and other simple parts.  
  
Theoretically the many different possibilities of these four simple rules allow the development of any kind of computing. A Turing machine has been implemented in Conway's Game of Life. There are hundreds of other amazing patterns.  
  
****  
  
Gosper Glider Gun.

**Design requirements**

1. Grid system
2. Awesome Layout designs are just beautiful.
3. Colors - Fonts and Texts
4. Links and navigation
5. Images / Icons
6. Forms and buttons
7. Validation
8. Responsive Web Design
9. Style Guide and Component Approach
10. Delivery — Analysis and pre-work phases

**Remember these guidelines for a proper design**

* Ensure the functions are all working logically
* Ensure proper access by everyone regardless of user.
* Use valid CSS and follow CSS best practices.

**Additional Resources:**

* 🛠 [Bootstrap Grid System](https://getbootstrap.com/docs/4.0/layout/grid/) (v4)
* 🛠 [Flexbox Grid](http://flexboxgrid.com/)
* 📖 [Don’t Overthink It Grids | CSS-Tricks](https://css-tricks.com/dont-overthink-it-grids/)