Tutorial – Recreating Conway's Game of Life in WebGPU Studio

This guide explains how to recreate the GameOfLifeByJohnConway.wgstudio example starting from a new project.

---

1) Objective and Principle

We simulate a 128×128 grid of cells using three buffers:

- currentState = **current** state (0 = dead, 1 = alive)
- futureState = **future** state (result of the computation)
- display = **display buffer** (0 or 0xFFFFFFFF for visualization)

Each simulation iteration follows these stages:

1. **Init** (once) : transforms random initial data into binary 0/1 values using modulo 2.
2. **Play** : applies the Game of Life rules to compute the future state in futureState.
3. **Cpy** : copies futurState into currentState.
4. **Render** : writes to the display buffer for visualization.

---

2) Creating the Project

1. Launch WebGPU Studio.
2. Click "New" to start a fresh project.

---

3) Creating the Buffers (Buffers Tab)

Create three buffers, named **currentState**, **futurState** and **display** each with the following configuration:

- **Size**: 128×128×1
- **Type**: Integer (i32)
- **Initialization**: Random Fill

After each creation or modification, click "Apply".

---

4) Adding the Constants Library (Functions Tab)

1. Navigate to the Functions tab.
2. Click "+ Add".
3. Name it (e.g., "Library 1" or a relevant title).
4. In the WGSL editor, paste the following code:

```
const SX = 128 ;
const SY = 128 ;

fn countNeighbours(index: u32) -> i32 {
    let sx: u32 = u32(SX);
    return  currentState[index-1u] + currentState[index+1u] +
currentState[index-sx] + currentState[index+sx]
          + currentState[index-1u+sx] + currentState[index+1u+sx] +
currentState[index-sx-1u] + currentState[index-sx+1u];
}
```

1. Click **`Apply`**

Note: These constants are used by the shaders.

---

5) Create the 4 Compute Shaders (`Compute Shaders` Tab)

For each shader:

1. Click **`+ Add`**
2. Enter the **Name**
3. Click **`Apply`**
4. Paste the WGSL code

Shader `Init`

```
@compute @workgroup_size(8, 8, 1)
fn Init(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.y * 128u + gid.x;
    if (index < arrayLength(&currentState)) {
        if (step == 0) {
            currentState[index] = currentState[index] % 2;
        }
    }
}
```

Shader `Play`

```
@compute @workgroup_size(8, 8, 1)
fn Play(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.y * 128u + gid.x;
    if (step >= 1 && gid.x >= 1 && gid.x < SX-1 && gid.y >= 1 && gid.y <
SX-1) {
        let nb = countNeighbours(index);
        if( currentState[index] == 0 ) { // dead cell
            if ( nb == 3 ) {
                futurState[index] = 1;
            } else {
                 futurState[index] = 0;
            }
        } else { // living cell
            if ( nb == 2 || nb == 3 ) {
              futurState[index] = 1;
            } else {
              futurState[index] = 0;
            }
        }
    }
}
```

Shader Cpy

```
@compute @workgroup_size(8, 8, 1)
fn Cpy(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.y * 128u + gid.x;
    if (step >= 1 && index < arrayLength(&currentState)) {
        currentState[index] = futurState[index] ;
    }
}
```

Shader Render

```
@compute @workgroup_size(8, 8, 1)
fn Render(@builtin(global_invocation_id) gid : vec3<u32>) {
    let index = gid.y * 128u + gid.x;
    if (index < arrayLength(&currentState)) {
        if ( currentState[index] == 0 ) {
            display[index] = 0;
        } else {
            display[index] = bitcast<i32>(0xFFFFFFFFu) ;
        }
    }
}
```

---

6) Configure the Pass (Pass Tab)

We create **4 steps** (pipelines) in this order, all with the same dispatch of 16x16x1.

Why dispatch = 16×16×1?

Each shader has `@workgroup_size(8,8,1)`.

- `16 * 8 = 128` So we cover exactly a **128×128** grid.

How to Create a Pipeline:

1. Select the corresponding shader.
2. Click **+Pipeline>>>**
3. Assign a title.

4. Define the dispatch parameters.
5. Click **Apply**.

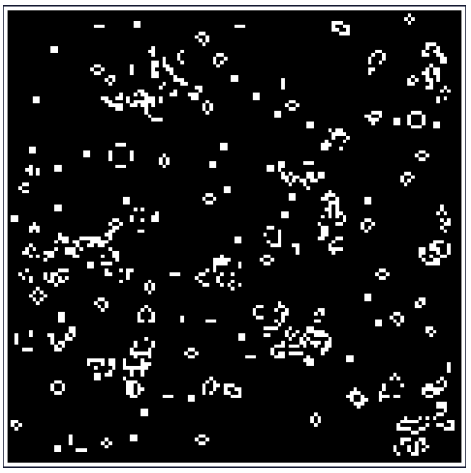
Pipeline Order:

1. Initialization : Shader **Init**
2. Game of Life : Shader **Play**
3. Update : Shader **Cpy**
4. Rendering : Shader **Render**

---

7) Compile and Execute

1. Go to the **`Buffers`** tab, on **display**
2. Visualize in **2D**
3. Click **`Compile`**
4. Click **Run** (or step with **`Step`**)



Expected result

---

8) Quick Checks (if it doesn't work)

- If the simulation look stuck → regenerate values of currentState and futurState
- **Console**: `Console` tab → read WGSL errors.
- **Buffer names**: they must be exactly `texture1`, `texture2`, `texture3`.
- **Size**: must be 128×128×1.
- **Dispatch**: must be 16×16×1 (with `workgroup_size(8,8,1)`).

---

9) Save Your Work

Click **Save** to export your project as a `.wgstudio` file.