# Didacticiel - Recréer *Game of Life* (John Conway) dans WebGPU Studio

Ce guide permet de recréer l'exemple **`GameOfLifeByJohnConway.wgstudio`** à partir d'un projet **`Nouveau`**.

---

## 1) Objectif et principe

On simule une grille **128×128** de cellules.

- `etatCourant` = état **courant** (0 = morte, 1 = vivante)
- `etatFutur` = état **futur** (résultat du calcul)
- `affichage` = **buffer d'affichage** (0 ou `0xFFFFFFFF` pour visualiser)

À chaque itération :

1. **Init** (une fois) : transforme le buffer aléatoire en 0/1 via `modulo 2`.
2. **Play** : calcule l'état futur (règles du Game of Life) dans `etatFutur`.
3. **Cpy** : copie `etatFutur` dans `etatCourant`.
4. **Render** : écrit `affichage` pour afficher.

---

## 2) Créer le projet

1. Lancer WebGPU Studio
2. Cliquer **`Nouveau`**

---

## 3) Créer les buffers (onglet `Buffers`)

Créer 3 buffers **etatCourant**, **etatFutur** et **affichage** avec chacun une taille de **128x128x1**, de type **Entier (i32)** et avec des valeurs initiales définies sur **Remplissage aléatoire**

À chaque création/modification : clique **`Appliquer`**.

---

## 4) Ajouter la bibliothèque de constantes (onglet `Fonctions`)

1. Aller dans l'onglet **`Fonctions`**
2. Cliquer **`+Ajouter`**
3. Nommer-la : `Bibliothèque 1` (ou autre)
4. Dans l'éditeur WGSL, coller :

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

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

5. Cliquer **`Appliquer`**

Note : ces constantes sont utilisées par les shaders.

---

## 5) Créer les 4 compute shaders (onglet `Compute Shaders`)

Pour chaque shader :

1. Cliquer **`+Ajouter`**
2. Donner le **Nom**
3. Cliquer **`Appliquer`**
4. Coller le code WGSL

## 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(&etatCourant)) {
        if (step == 0) {
            etatCourant[index] = etatCourant[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( etatCourant[index] == 0 ) { // dead cell
            if ( nb == 3 ) {
                etatFutur[index] = 1;
            } else {
                 etatFutur[index] = 0;
            }
        } else { // living cell
            if ( nb == 2 || nb == 3 ) {
              etatFutur[index] = 1;
            } else {
              etatFutur[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(&etatCourant)) {
        etatCourant[index] = etatFutur[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(&etatCourant)) {
        if ( etatCourant[index] == 0 ) {
            affichage[index] = 0;
        } else {
            affichage[index] = bitcast<i32>(0xFFFFFFFFu) ;
        }
    }
}
```

---

# 6) Configurer la Pass (onglet Pass)

On crée **4 étapes** (pipelines) dans cet ordre, toutes avec le même dispatch de 16x16x1.

## Pourquoi `dispatch = 16×16×1` ?

Chaque shader a `@workgroup_size(8,8,1)`.

- `16 * 8 = 128` Donc on couvre exactement une grille **128×128**, ce qui représente la taille du buffer.

Comment créer une pipeline :

- Sélectionner le shader correspondant.
- Appuyer sur **+Pipeline>>>**
- Donnez lui un titre
- Définissez le dispatch.
- Cliquer sur Appliquer

Ordre des pipelines :

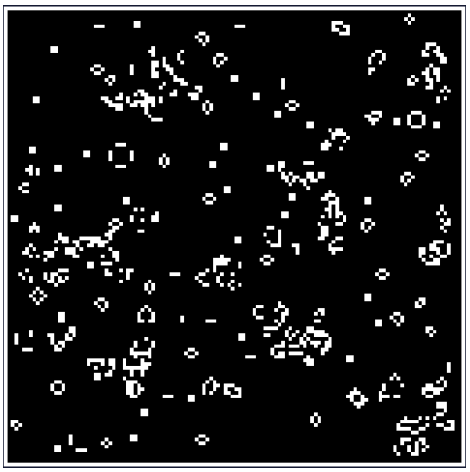
1. Initialisation : Shader **Init**
2. Jeux de la Vie : Shader **Play**
3. Mise à jour : Shader **Cpy**
4. Rendu : Shader **Render**

---

# 7) Compiler et exécuter

1. Aller dans l'onglet **`Buffers,`** `sur` **`affichage`**
2. Visualiser en **2D**
3. Cliquer **`Compile`**

4. Cliquer **Run** (ou avancer avec **`Step`**)



*Resultat attendu*

---

## 8) Vérifications rapides (si ça ne marche pas)

- Si la simulation semble “bloquée” → régénérer etatCourant et etatFutur
- **Console** : onglet `Console` → lire les erreurs WGSL.
- **Noms des buffers** : ils doivent être exactement `etatCourant`, `etatFutur`, `affichage`.
- **Taille** : doit être 128×128×1.
- **Dispatch** : doit être 16×16×1 (avec `workgroup_size(8,8,1)`).

---

## 9) Sauvegarder

Cliquer **`Sauver`** pour exporter le projet en `.wgstudio`.