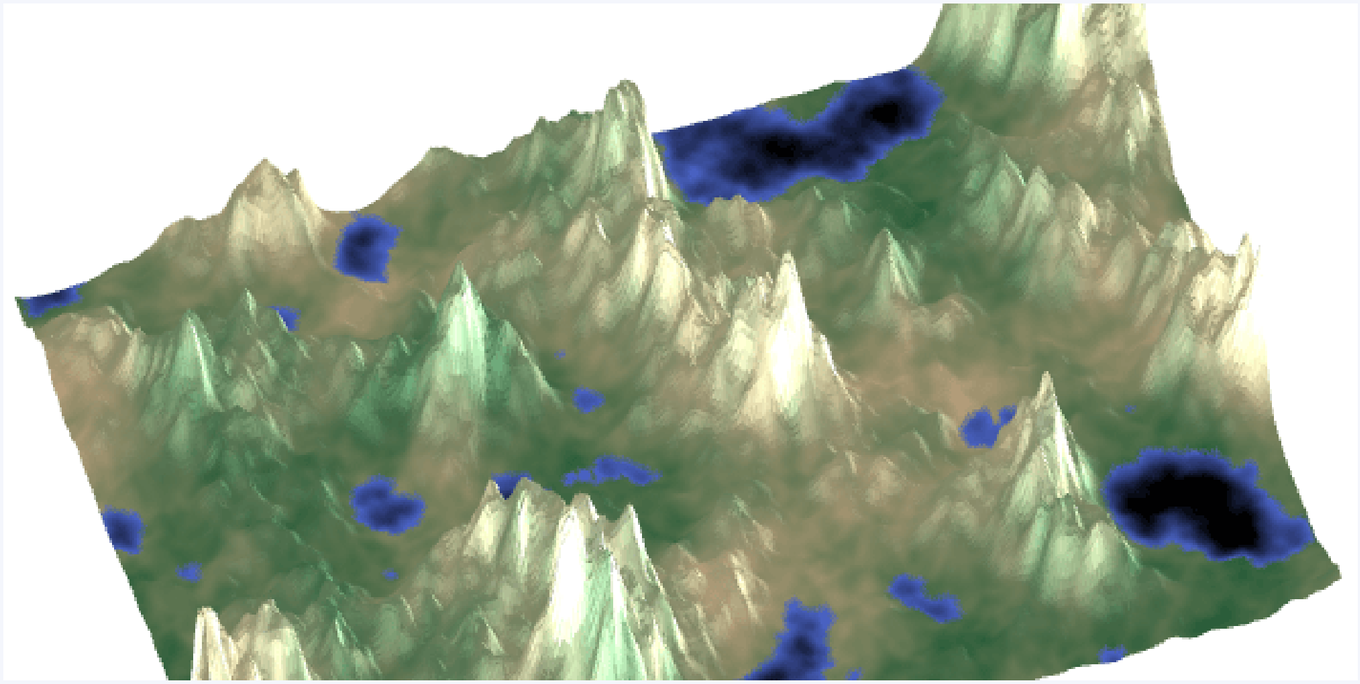
LogiCube

UNITY

Comment generer un monde ?



Dans la vraie vie…

- **Le sol n’est jamais tout plat**
- **Il y a :**
  - **des bosses**
  - **des creux**
  - **des collines**
  - **des montagnes**

Dans un jeu…

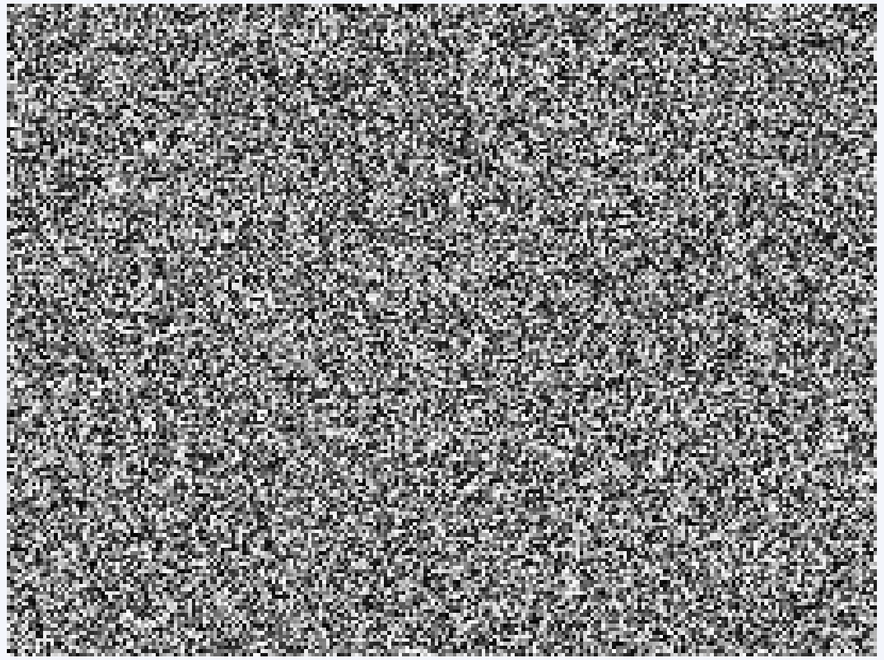
- **On ne dessine pas tout à la main**
- **On demande à l’ordinateur de créer le relief**
- **Pour ça, on utilise une astuce mathématique**

Cette astuce s’appelle : le bruit

C’est quoi du bruit ?

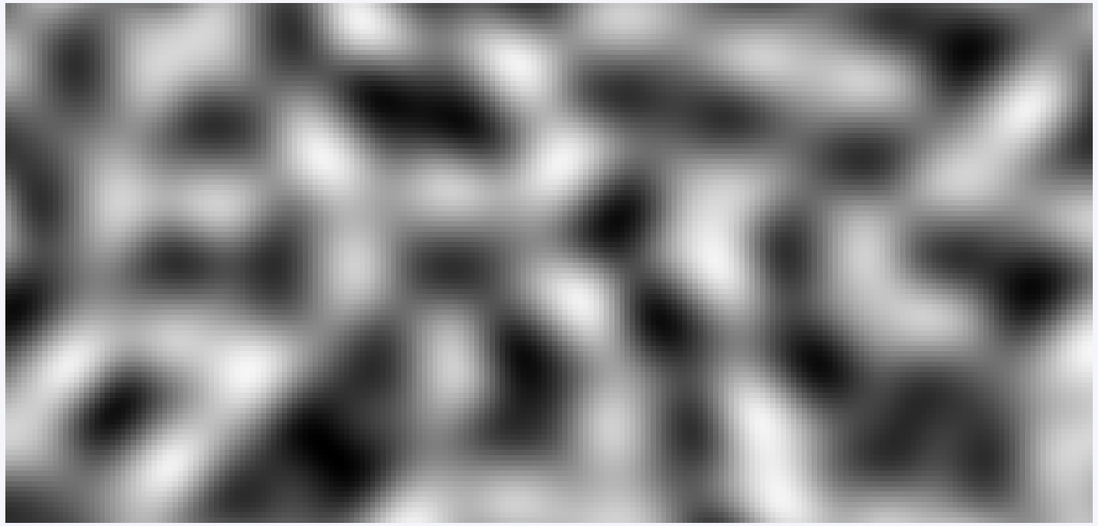
Le bruit aléatoire :

- **Comme le "static" quand la TV n'a pas de signal**
- **Chaque pixel est noir OU blanc, totalement au hasard**
- **Aucun lien entre les pixels voisins**



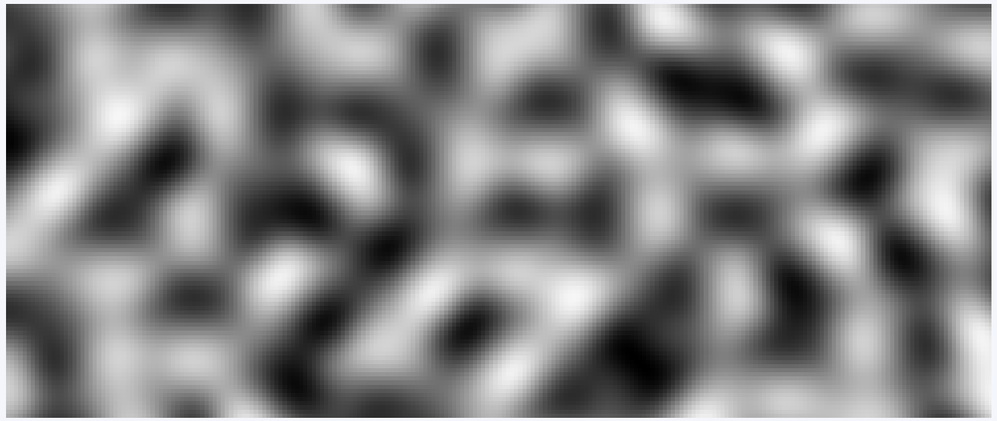
Le bruit de Perlin : du hasard... mais lissé

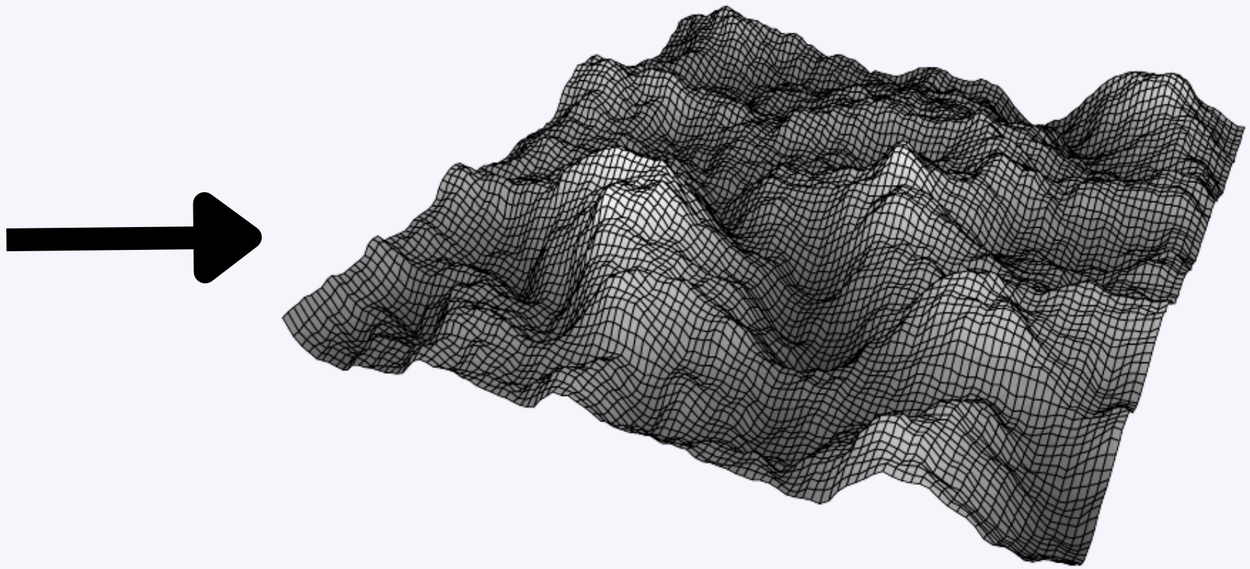
- **Le bruit totalement aléatoire :**
  - **Chaque pixel est blanc OU noir, sans aucun lien avec ses voisins**
- **Le bruit de Perlin :**
  - **Les valeurs changent PROGRESSIVEMENT d'un pixel à l'autre**
  - **Les pixels voisins ont des valeurs similaires**
  - **Résultat : on obtient des formes douces et naturelles**



Comment le bruit de Perlin crée notre monde

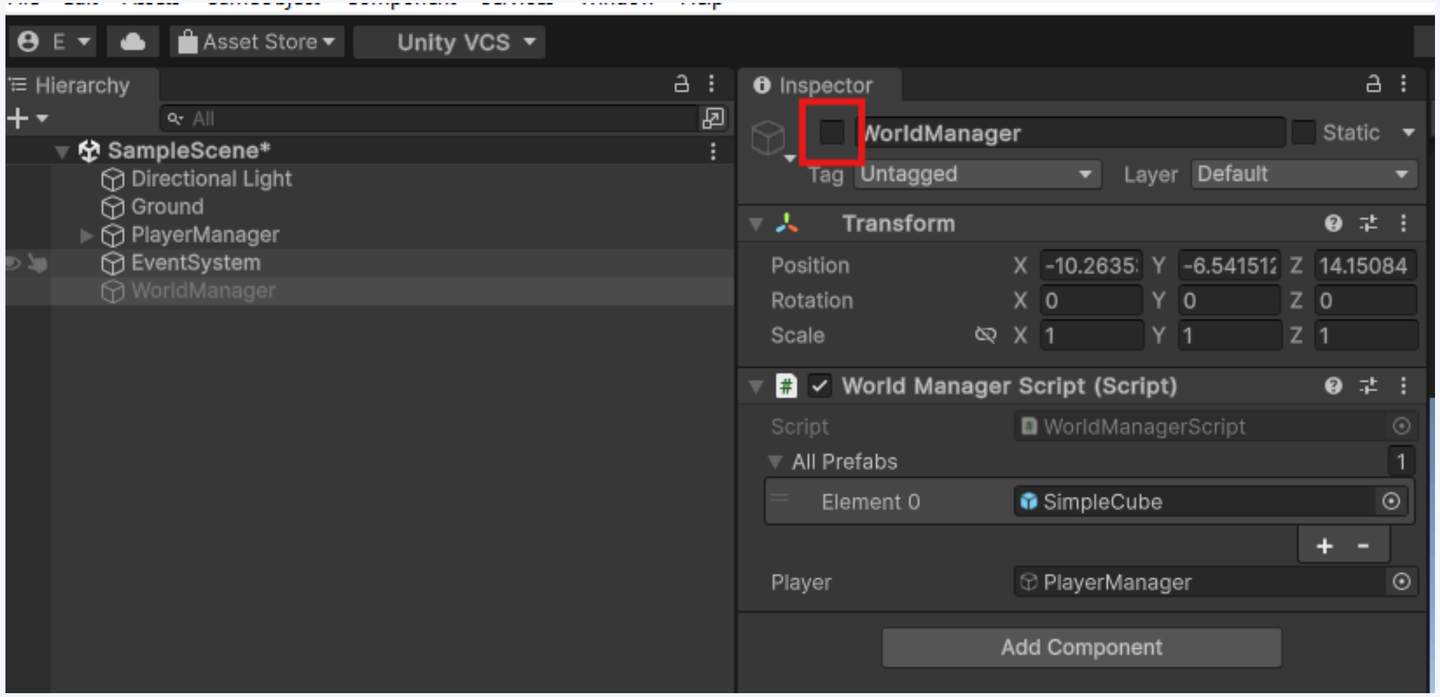
- **Le principe :**
  - **On génère une "carte" de bruit de Perlin**
  - **Chaque valeur (de 0 à 1) représente une hauteur**
  - **Valeurs basses = plaines et eau**
  - **Valeurs moyennes = collines**
  - **Valeurs hautes = montagnes**





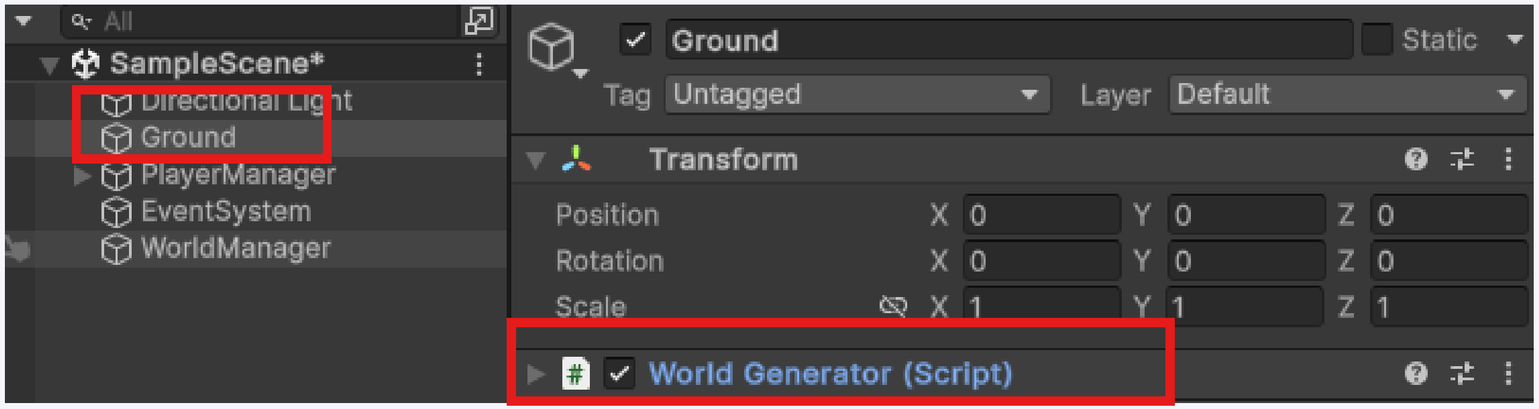
Générer le terrain

Pour commencer, désactivez le WorldManager pour éviter la sauvegarde du monde. Nous le réactiverons plus tard.



Générer le terrain

- **Crée un script WorldGenerator et mettez le sur votre ground**



Générer le terrain

- **Ajouter ces deux fonctions dans WorldGenerator**

```
void Start()
{
    GenerateTerrain();
}

void GenerateTerrain()
{
    int half = chunkSize / 2;

    for (int x = -half; x < half; x++)
    {
        for (int z = -half; z < half; z++)
        {
            // 1) On calcule les coordonnées pour le bruit
            float nx = (x + half) * noiseScale;
            float nz = (z + half) * noiseScale;

            // 2) On récupère une valeur entre 0 et 1
            float noise = Mathf.PerlinNoise(nx, nz);

            // 3) On transforme ça en hauteur en blocs
            int height = Mathf.FloorToInt(noise * terrainHeight);

            // 4) On empile les blocs de y=0 jusqu'à y=height
            for (int y = 0; y <= height; y++)
            {
                Vector3 pos = new Vector3(x, y, z);
                Instantiate(defaultBlockPrefab, pos, Quaternion.identity);
            }
        }
    }
}
```

- **On parcourt une grille :**
  - **x = gauche / droite**
  - **z = avant / arrière**
- **Chaque case correspond à une colonne de blocs**

PerlinNoise donne une valeur entre 0 et 1 pour une coordonné (bruit de perlin)

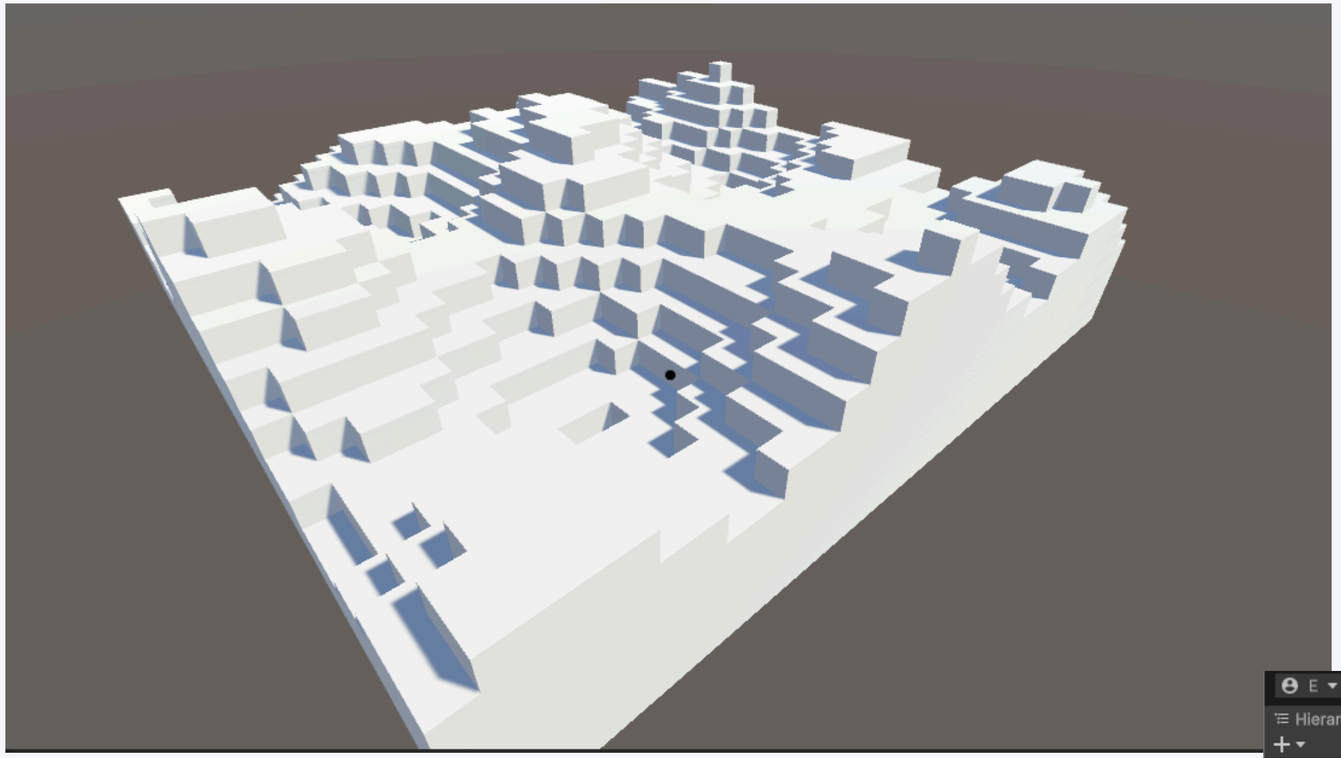
Transformation du bruit en hauteur.

Exemple

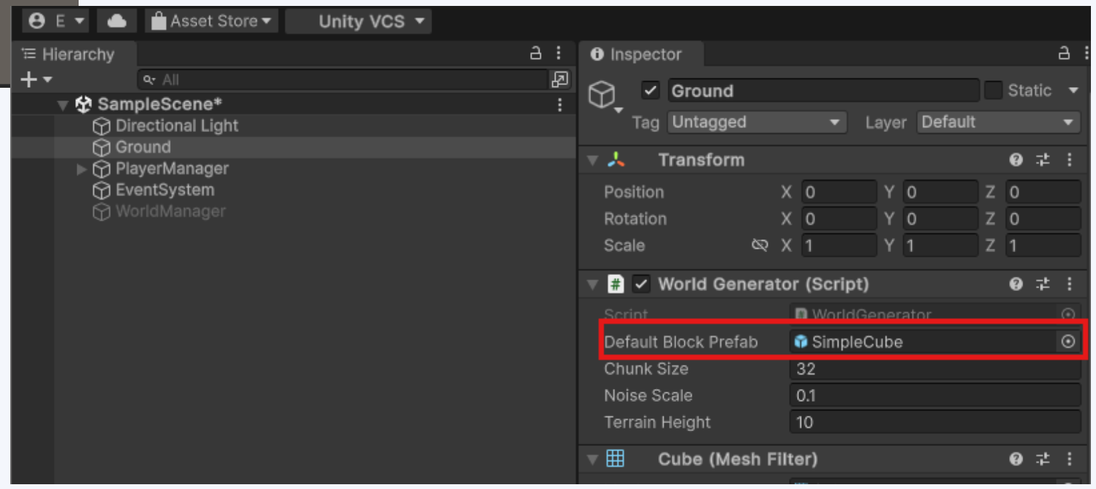
- **noise = 0 → height = 0**
- **noise = 0.5 → height ≈ terrainHeight / 2**
- **noise = 1 → height = terrainHeight**

nb: FloorToInt() covertit un nombre a virgule en un nombre entier

Générer le terrain



Lancez le jeu,
vous devriez obtenir ceci



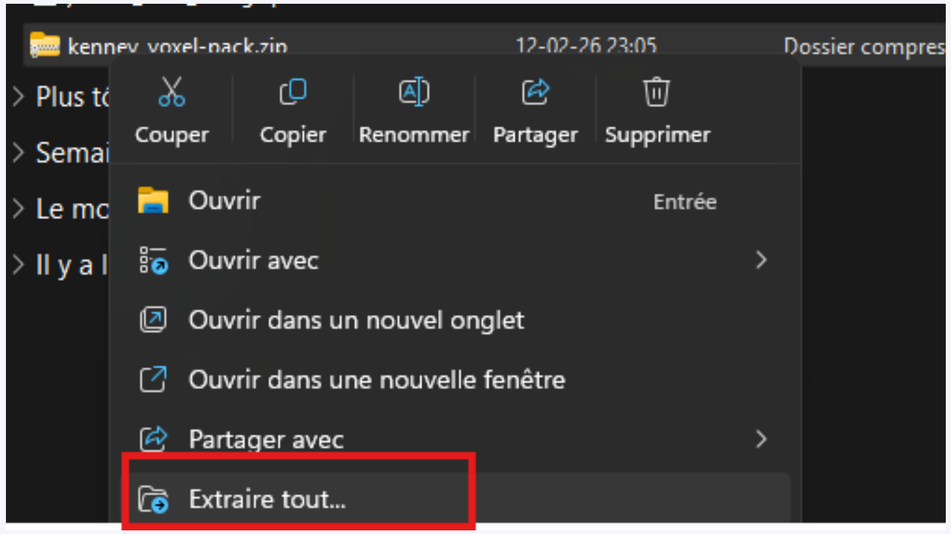
N’oubliez pas de sélectionner votre prefab de cube ;)

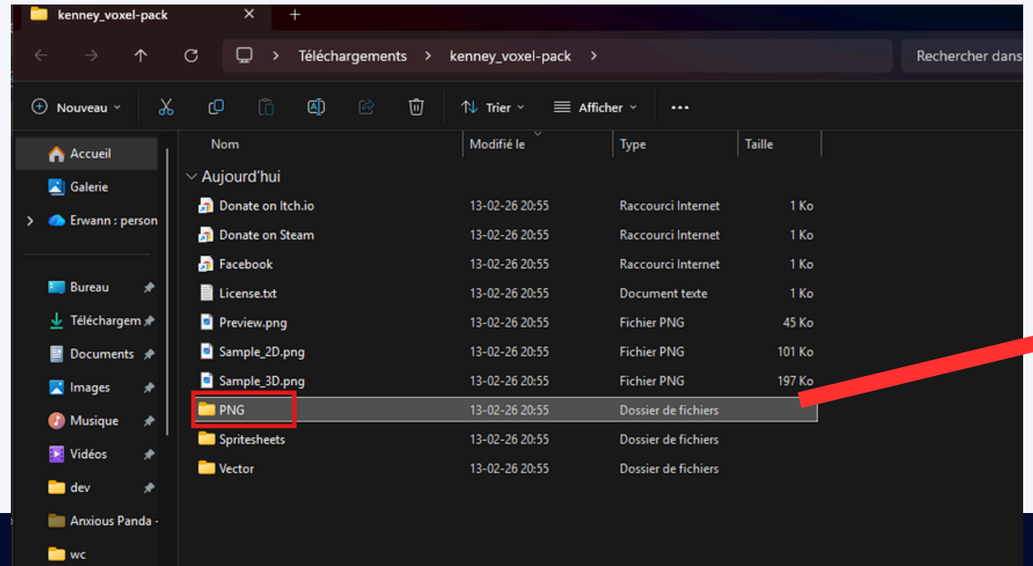
Textures

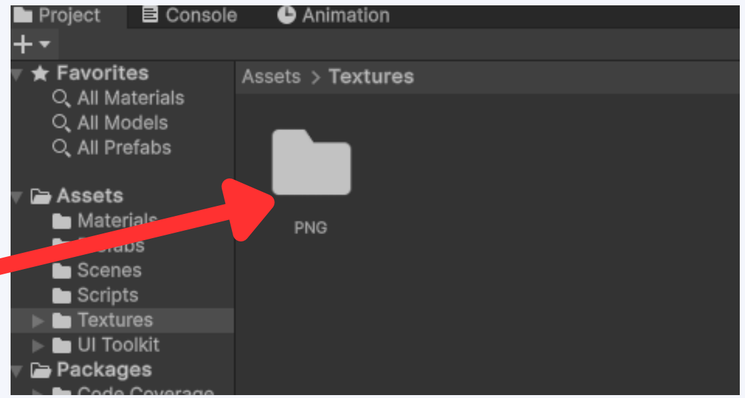
- On va ajouter des textures à nos blocs
- Créer un nouveau dossier Textures dans Unity
- Cliquez sur ce lien :

https://www.kenney.nl/media/pages/assets/voxel-pack/ee6f21f078-1677662501/kenney_voxel-pack.zip

- Dézippez le dossier et glissez le dossier PNG dans votre nouveau dossier Textures

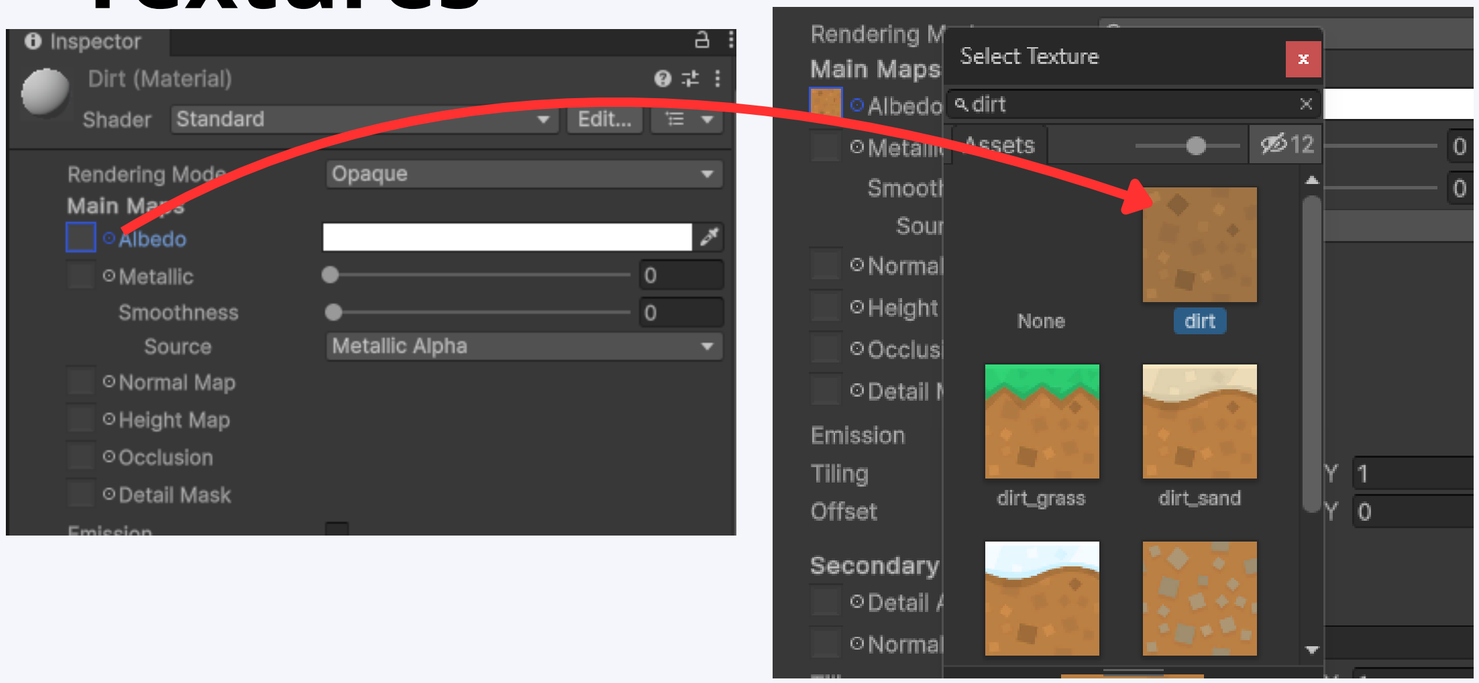


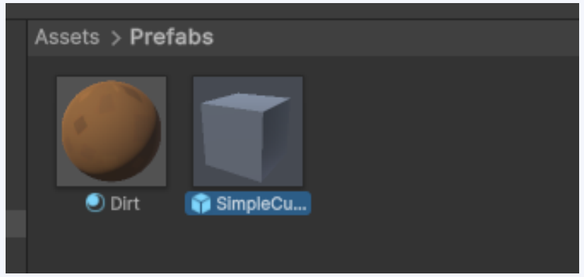


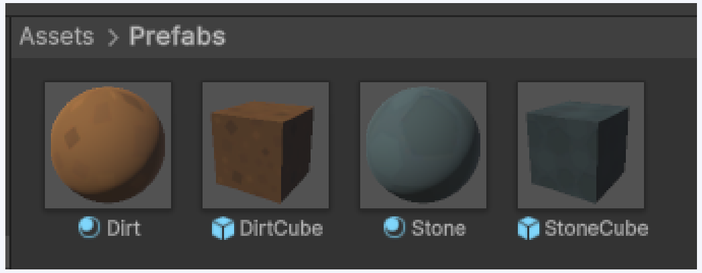


Textures

- Dans le dossier Prefabs, créez un nouveau matériau
- Nommez-le Dirt
- Réglez la Smoothness à 0
- Cliquez sur le rond à côté de Albedo
- Sélectionnez la texture dirt
- Renommez votre prefab en DirtCube
- Dupliquez et faites la même chose pour la pierre (Stone, StoneCube)







Textures

- Retournez dans le script WorldGenerator
- Modifiez les variables publiques

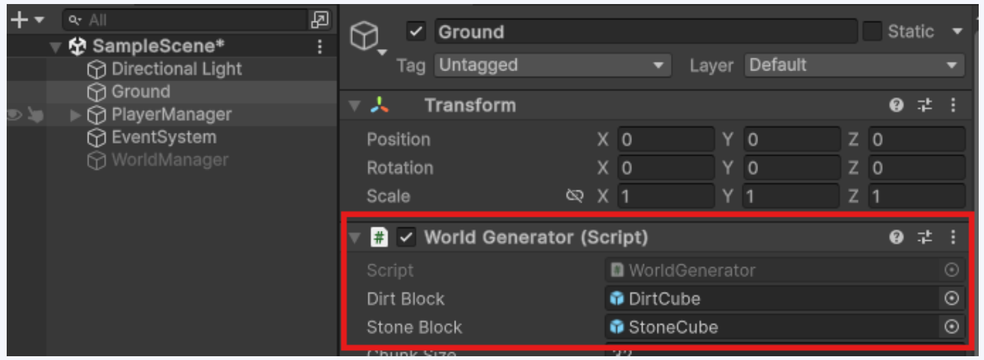
```
public class WorldGenerator : MonoBehaviour
{
    public GameObject dirtBlock;
    public GameObject stoneBlock;
    public int chunkSize = 32;
    public float noiseScale = 0.1f;
    public int terrainHeight = 10;
```

- Ajoutez cette logique dans la génération
- On choisit le type de bloc en fonction de la hauteur :
  - terre (dirt) en surface
  - pierre (stone) en profondeur

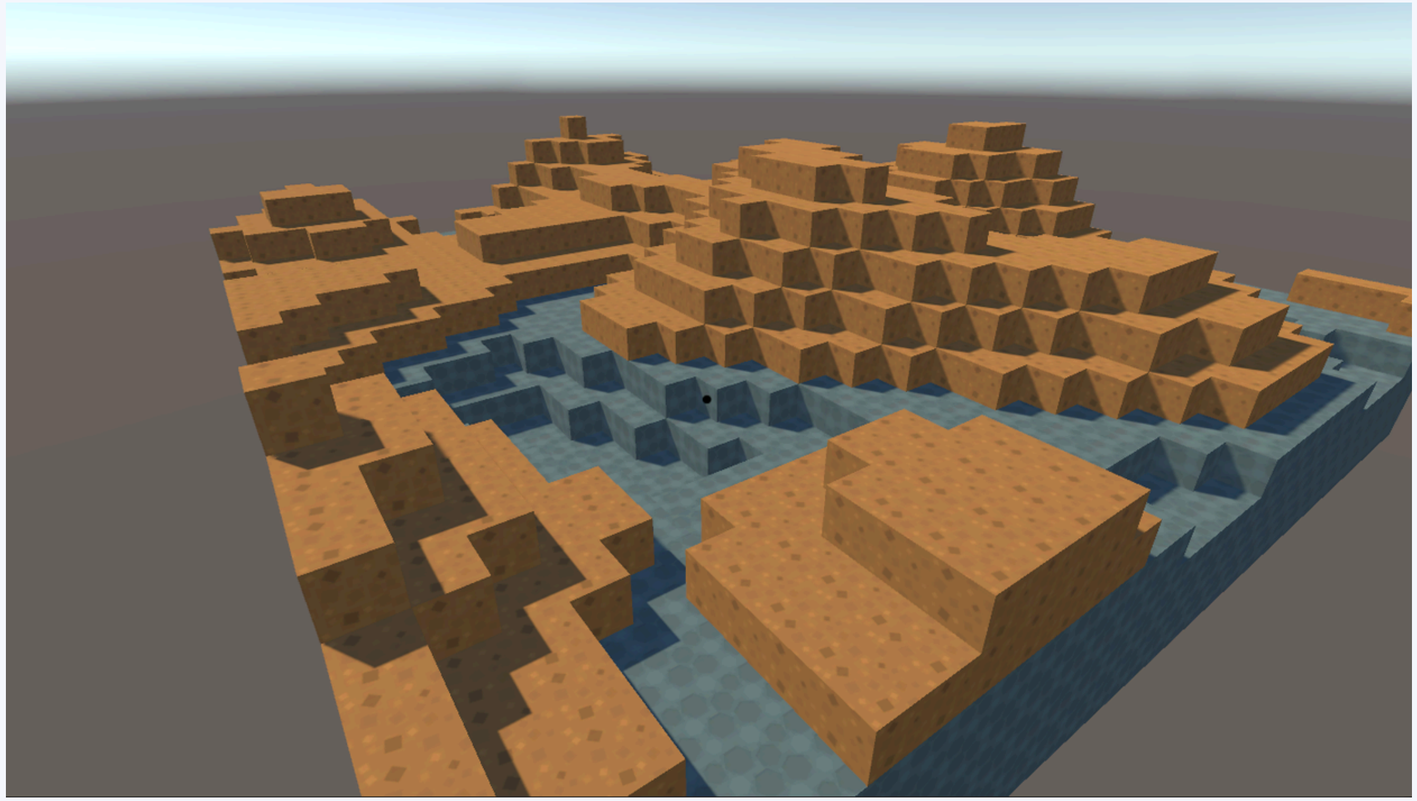
```
// 4) On empile les blocs de y=0 jusqu'à y=height
for (int y = 0; y <= height; y++)
{
    Vector3 pos = new Vector3(x, y, z);
    if(y > 3)
    {
        Instantiate(dirtBlock, pos, Quaternion.identity);
    }
    else
    {
        Instantiate(stoneBlock, pos, Quaternion.identity);
    }
}
```

Dans Unity

- Assignez les prefabs DirtCube et StoneCube



Générer le terrain



Lancez le jeu,
vous devriez obtenir ceci

Seed

- Actuellement, le monde généré est toujours le même.
- On voudrait un système qui nous permette de créer des mondes différents.
- On voudrait aussi pouvoir, simplement à partir d’un nombre (la seed), regénérer ce même monde.

C’est quoi une seed ?

- Une seed, c’est un nombre qui sert de point de départ au hasard.
- Exemples :
  - Seed = 42 → toujours le même monde
  - Seed = 123456 → un autre monde
  - Même seed = même terrain
- Minecraft fonctionne exactement comme ça.

Pourquoi utiliser une seed ?

- Cela nous permettra de regénérer le monde au lieu de devoir enregistrer la position de tous les blocs.
- On enregistrera seulement les blocs que l'utilisateur pose ou casse.

Comment utiliser une seed ?

- On garde le bruit de Perlin.
- Mais on modifie les coordonnées qu'on passe à la fonction.
- En pratique :
  - On ajoute un décalage (offset) au bruit.
  - Cet offset dépend de la seed.

Avant

```
float nx = (x + half) * noiseScale;
float nz = (z + half) * noiseScale;
```

Apres

```
float nx = (x + noiseOffset.x) * noiseScale;
float nz = (z + noiseOffset.y) * noiseScale;
```

implémentation de la seed

- Ouvrez le scipt WorldGenerator
- ajoutez ces lignes

- Notre offset (décalage)

- Seed (aléatoire pour la démonstration, mais sera ensuite enregistrée dans un fichier pour régénérer le même monde)

- Crée un générateur pseudo-aléatoire (aléatoire prévisible)
- permet de produire toujours la même suite de nombres pour une seed donnée

ps: Si ce concept vous semble compliqué, N'hésitez pas à poser des questions ;)

```
public class WorldGenerator : MonoBehaviour
{
    public GameObject dirtBlock;
    public GameObject stoneBlock;
    public int chunkSize = 32;
    public float noiseScale = 0.1f;
    public int terrainHeight = 10;

    private Vector2 noiseOffset;


    void Start()
    {
        int seed = Random.Range(1, 1000);
        var prng = new System.Random(seed);
        noiseOffset = new Vector2(
            prng.Next(-100000, 100000),
            prng.Next(-100000, 100000)
        );

        GenerateTerrain();
    }
```

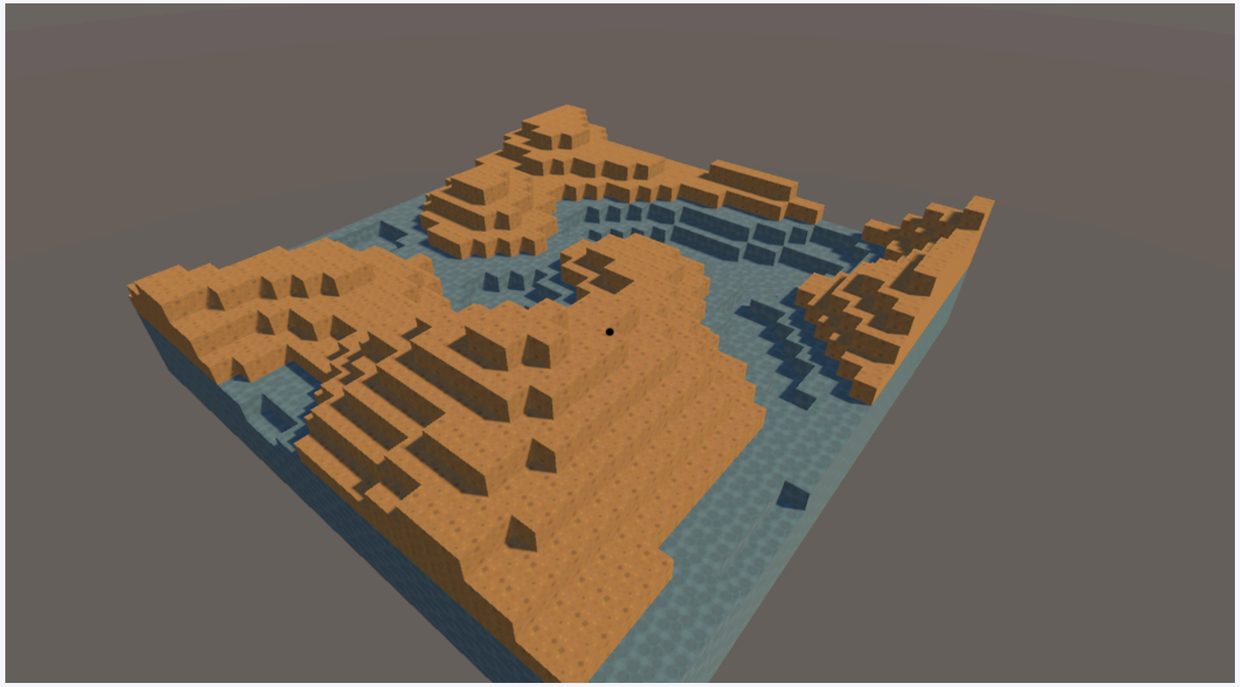
```
void GenerateTerrain()
{
    int half = chunkSize / 2;

    for (int x = -half; x < half; x++)
    {
        for (int z = -half; z < half; z++)
        {
            // 1) On calcule les coordonnées pour le bruit
            float nx = (x + noiseOffset.x) * noiseScale;
            float nz = (z + noiseOffset.y) * noiseScale;
```

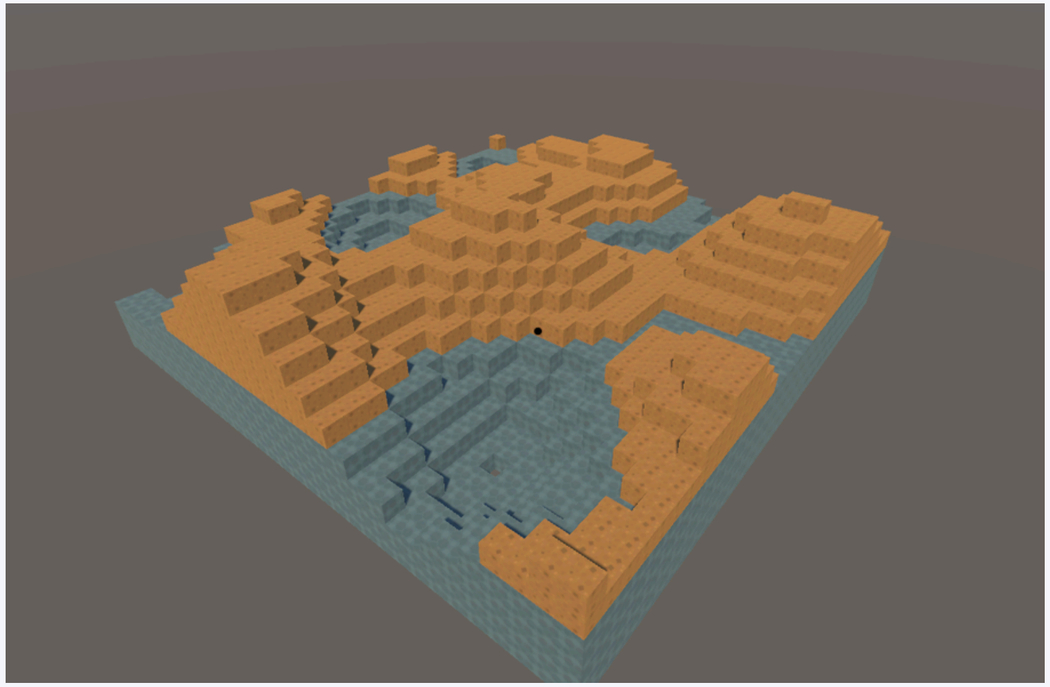
implémentation de la seed

Code du WorldGenerator jusqu’à présent (si besoin)
https://gist.github.com/ErwannCharlier/186642beff4a508de9d1a0864a6305d3

Testez le résultat

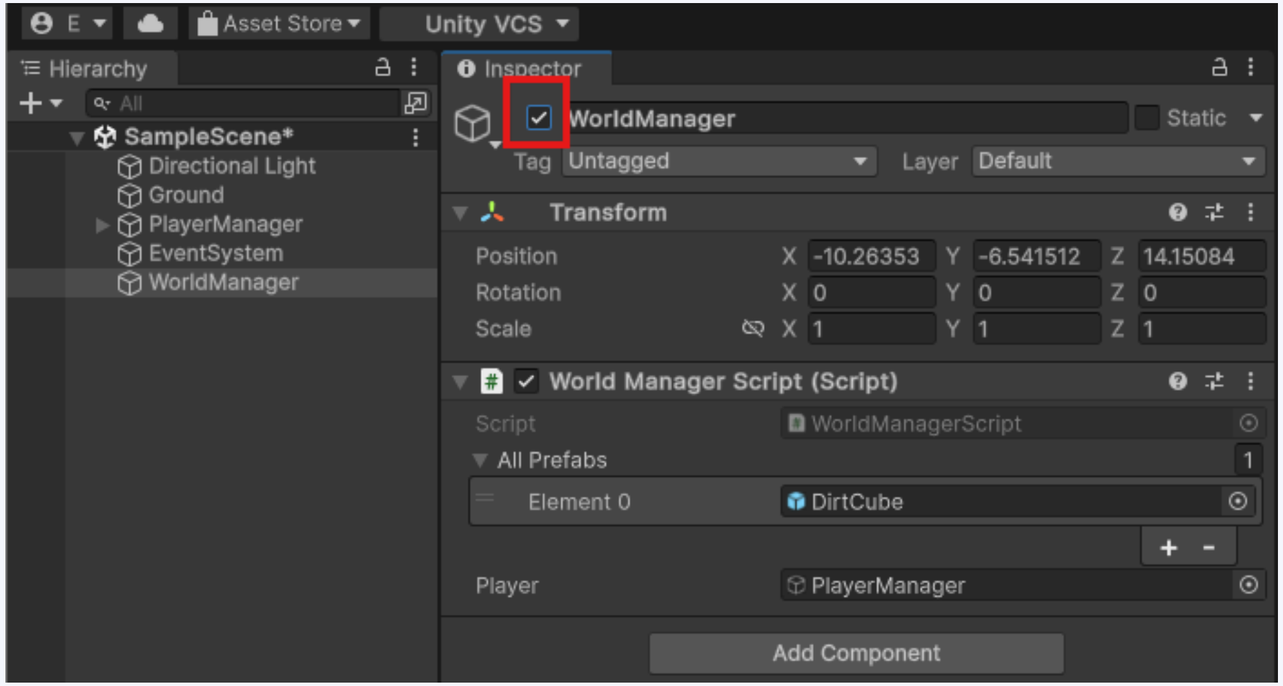
- **Relancez le jeu plusieurs fois**
- **Vous devriez obtenir un monde différent à chaque lancement**





Enregistrer le monde

- **Ré activer le WorldManager dans unity**



Enregistrer le monde

- Le script est assez long, donc je ne vous demande pas de l’écrire vous-même. En revanche, nous allons détailler les points importants.

- Ce qui est enregistré dans le fichier
  - les blocs posés par le joueur
  - les blocs cassés par le joueur
  - la position du joueur
  - la seed du monde

- Ce qui n’est pas enregistré
  - les blocs du monde auto-généré

- lien vers le script a copier/coller:
- https://github.com/Ayly-EXE/LogiCube/blob/Feat/worldGeneration/Assets/Scripts/WorldManagerScript.cs

Enregistrer le monde

Structures de données pour sauvegarder les modifs

- Sert à savoir quel bloc existe actuellement à une position.
- clé : position du bloc
- valeur : l'objet Unity (GameObject) correspondant

- Sert à mémoriser les blocs ajoutés par le joueur.
- clé : position
- valeur : type de bloc (ex: "DirtCube", "StoneCube")

- Sert à mémoriser les blocs cassés par le joueur.
- clé : position
- valeur : type du bloc cassé

```
public class WorldManagerScript : MonoBehaviour
{
    public GameObject[] allPrefabs;
    public GameObject player;
    public WorldGenerator generator;
    private string savePath;

    private readonly Dictionary<Vector3Int, GameObject> worldBlocks = new();
    private readonly Dictionary<Vector3Int, string> placed = new();
    private readonly Dictionary<Vector3Int, string> removed = new();
```

Enregistrer le monde

Comment le monde se reconstruit au chargement ?

- **Régénère le monde de base à partir de la seed sauvegardée dans le fichier**
- **Applique le delta (les différences par rapport au monde de base) :**
  - **enlever ce qui a été cassé**
  - **ajouter ce qui a été posé**

```
void Start()
{
    // 1) Regénère le monde de base
    generator.Generate(loaded.seed, this);

    // 2) Applique le delta
    ApplyDelta(loaded);

    // 3) Player
    if (loaded.player != null && player != null)
    {
        player.transform.position = loaded.player.position;
        player.transform.rotation = loaded.player.rotation;
    }
}
```

Enregistrer le monde

Dans PlayerAction:

- **Ajoutez ces lignes**
- **Le script ne crée plus directement les blocs**
  - **Il délègue le placement et la destruction des blocs au WorldManager**

```
public class PlayerAction : MonoBehaviour
{
    public GameObject blockPrefab;
    public WorldManagerScript worldManager;
    public RaycastHit? GetHit(float maxDistance = 100f)
    {
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, maxDistance))
        {
            return hit;
        }

        return null;
    }
```

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        RaycastHit? hit = GetHit();
        if (hit.HasValue)
        {
            Vector3 blockPos = GetBlockPlacementPosition(hit.Value);
            //Instantiate(blockPrefab, blockPos, Quaternion.identity);
            worldManager.PlayerPlacedBlock(blockPos, blockPrefab.name);
            Debug.Log("Block placed at: " + blockPos);
        }
        else
        {
            Debug.Log("Nothing hit");
        }
    }
```

```
    if (Input.GetMouseButtonDown(1))
    {
        RaycastHit? hit = GetHit();
        if (hit.HasValue)
        {
            GameObject hitObject = hit.Value.collider.gameObject;

            if (hitObject.CompareTag("Block"))
            {
                //Destroy(hitObject);
                worldManager.PlayerDestroyedBlock(hitObject);
                Debug.Log("Block destroyed: " + hitObject.name);
            }
            else
            {
```

Enregistrer le monde

Dans WorldGenerator:

- Ajoutez ces lignes
- Générer le monde à partir de la seed enregistrée dans le fichier
- Le WorldGenerator ne s’occupe plus de la sauvegarde
- Il informe le WorldManager de chaque bloc généré automatiquement

```
1 reference
public void Generate(int seed, WorldManagerScript manager)
{
    var prng = new System.Random(seed);
    noiseOffset = new Vector2(
        prng.Next(-100000, 100000),
        prng.Next(-100000, 100000)
    );

    GenerateTerrain(manager);
```

```
void GenerateTerrain(WorldManagerScript manager)
{
    int half = chunkSize / 2;

    for (int x = -half; x < half; x++)
    {
        for (int z = -half; z < half; z++)
        {
            // 1) On calcule les coordonnées pour le bruit
            float nx = (x + noiseOffset.x) * noiseScale;
            float nz = (z + noiseOffset.y) * noiseScale;

            // 2) On récupère une valeur entre 0 et 1
            float noise = Mathf.PerlinNoise(nx, nz);

            // 3) On transforme ça en hauteur en blocs
            int height = Mathf.FloorToInt(Mathf.PerlinNoise(nx, nz) * terrainHeight);

            // 4) On empile les blocs de y=0 jusqu'à y=height
            for (int y = 0; y <= height; y++)
            {
                Vector3Int gridPos = new Vector3Int(x, y, z);
                Vector3 worldPos = (Vector3)gridPos;
                GameObject bloc;
                if (y > 3)
                {
                    bloc = Instantiate(dirtBlock, worldPos, Quaternion.identity);
                }
                else
                {
                    bloc = Instantiate(stoneBlock, worldPos, Quaternion.identity);
                }

                manager.RegisterGeneratedBlock(bloc, gridPos);
            }
        }
    }
}
```

You have Docker instal
install the recommend

Enregistrer le monde

- **Sélectionnez l’objet WorldManager dans la scène**
- **Assignez les prefabs de blocs (DirtCube, StoneCube)**
- **Vérifiez les références :**
  - **Player**
  - **WorldGenerator**