# Monde Procédural – Level Design

BERARD Quentin – CHARBONNIER Pierrick – FALLOURD Ségolène – SEVESTRE Antoine – TEXIER Florian

## Table des matières

# Analyse

La création d'un monde procédural requiert en premier lieu l'identification des besoins nécessaires à sa génération. Dans le contexte de l'exercice, la référence principale est Minecraft. Celui-ci utilise des blocs comme unité de base du monde, ce qui rend l'analyse de son fonctionnement plus simple que pour d'autres méthodes.

Différents types de blocs composent le terrain :

- Le gazon, qui correspond à la surface.
- La terre, présente sur une hauteur de 3 blocs sous la surface.
- La roche, présente sous la terre.

A partir de ces éléments, une hiérarchie est facilement identifiable, permettant de mettre en place les fondations d'une carte procédurale en trois dimensions. D'autres éléments supplémentaires sont à prendre en compte : l'eau et ses rives.

Le niveau de l'eau est déterminé de base à une hauteur de 62 blocs, ce qui permet de déterminer les endroits où l'océan créera des cours d'eau dans le terrain. L'eau peut également être présente sous la forme d'étangs indépendants des cours d'eau. Dans tous les cas, le cheminement des fleuves suit une certaine logique en respectant un angle et une largeur cohérente.

Après ça, le sable entre en jeu et cerne l'eau en prenant divers paramètres en compte avec des blocs plus ou moins dispersés, remplaçant ou s'ajoutant aux blocs adjacents. Si des blocs de terre sont présents en trop faible quantité, ils sont également remplacés par du sable. Ce dernier est également plus présent à proximité des très grandes étendues d'eau.

Il est également à noter qu'aucune ligne droite n'est présente dans l'environnement. Celui-ci dispose d'un relief plus ou moins varié selon son biome, permettant de créer un monde partiellement aléatoire mais cohérent, que ce soit pour des plaines, des forêts, des montagnes ou des pics enneigés.

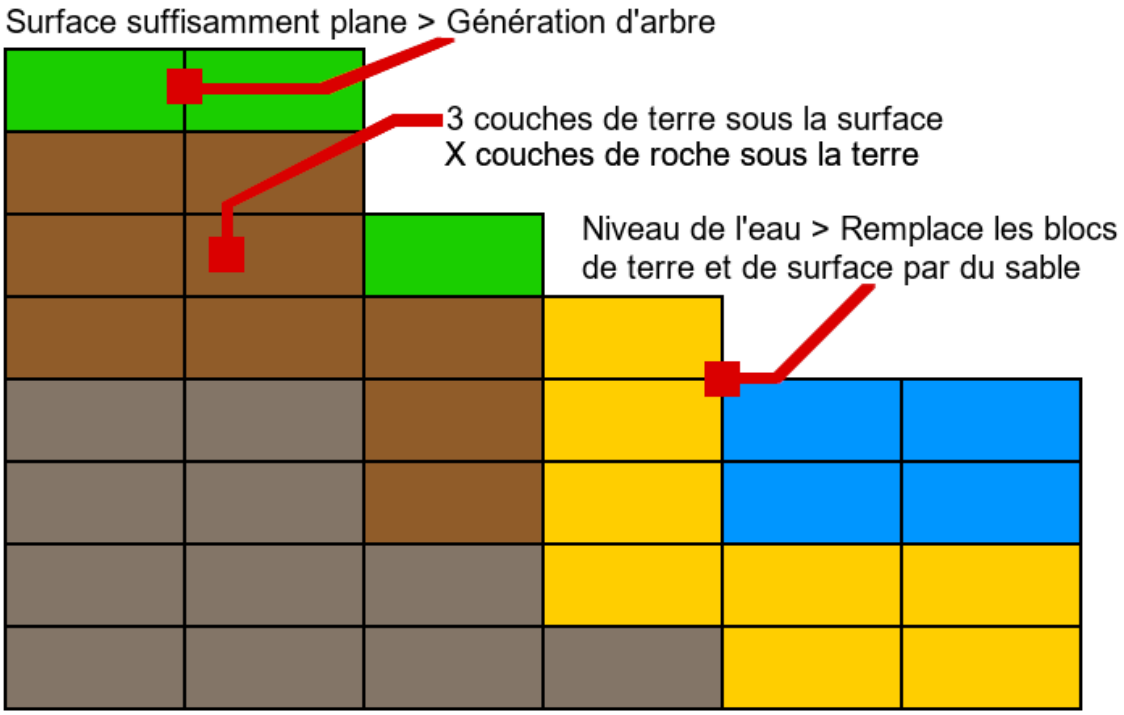
La nature est également à prendre en compte, avec des arbres répartis selon une certaine logique. De grandes plaines seront très peu boisées, tandis que d'autres zones seront criblées d'arbres. Leur placement varie selon le relief. Plusieurs types d'arbres sont disponibles pour amener en diversité tout en appuyant sur la cohérence des zones, en plaçant par exemple des sapins dans les zones enneigées. Leur placement peut être abordé de deux manières :

- Un arbre est posé selon la place disponible dans l'environnement.
- Un arbre crée sa place dans l'environnement en adaptant les blocs adjacents.

Ces paramètres peuvent aussi être modifiés selon le biome dans lequel la végétation s'inscrit. On peut imaginer un biome désert dans lequel logiquement il y aurait peu d'arbres. La densité d'arbres au sein d'un biome est un paramètre important pour la suite.

Le type d'arbre entre aussi en compte dans la création des biomes. Il faut rester un minimum réaliste et cohérent dans les environnements qui seront créés pour ne pas rencontrer de problèmes d'identification des différents biomes.

A partir de ces données, il est possible de commencer la création du monde procédural.

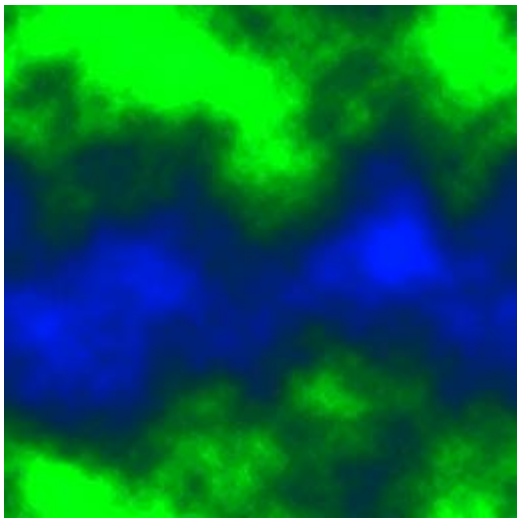


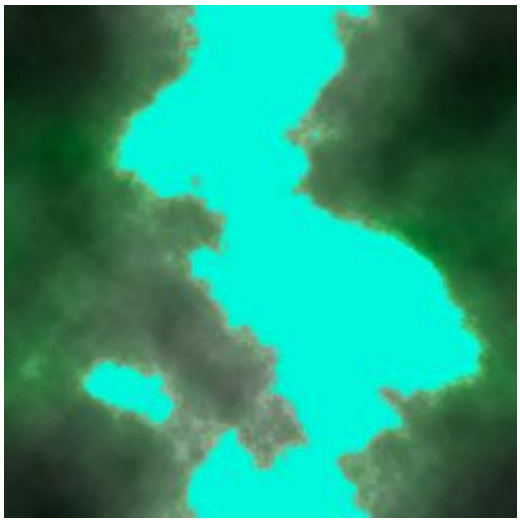

## Création du terrain

Pour créer un terrain procédural, la solution la plus abordable reste l'utilisation de bruit, ou noise. Celui-ci crée, d'une manière approchant l'aléatoire, une forme irrégulière semblable à un nuage. Basiquement, le noise ne se compose que de noir, de blanc, et de nuances de gris (pas de couleurs).

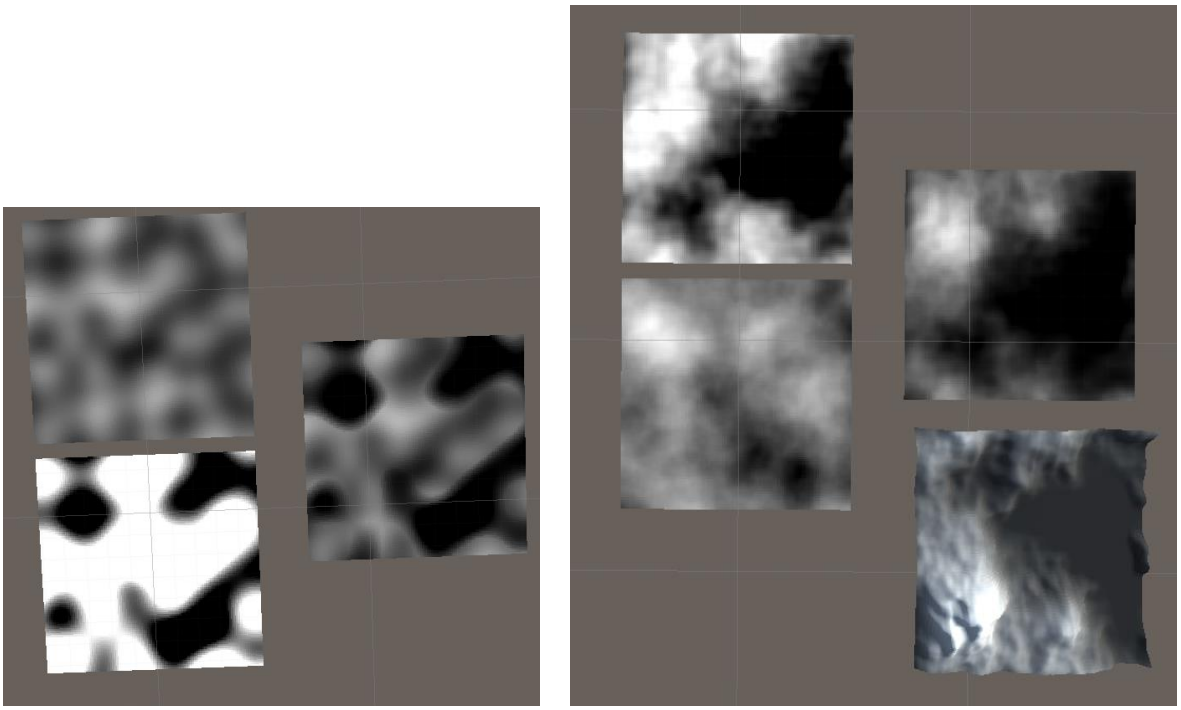
A l'aide d'un logiciel d'édition d'image, celui-ci peut être produit très rapidement. Le noise ainsi généré s'appuie sur des valeurs comprises entre 0 et 255, qu'il est possible de manipuler via le contraste pour produire des teintes de blanc et de noir plus ou moins nettes, donnant une représentation graphique en 2D de ce que peut donner la génération procédurale.

En attribuant des couleurs correspondant à la surface, à l'eau, au sable et aux arbres, on peut alors obtenir ce résultat :



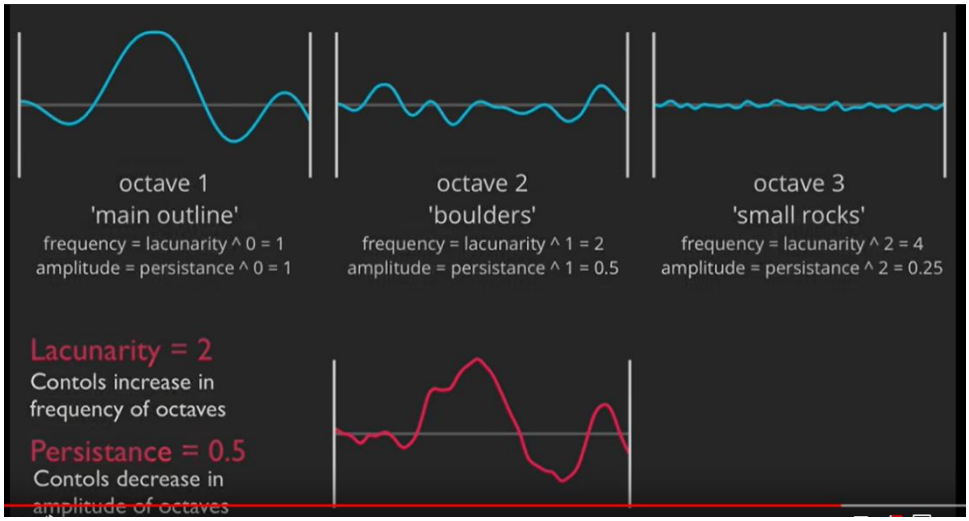


Avec cette logique, il est ainsi possible de reproduire la même chose sur Unity pour en obtenir une représentation en 3D. Avec un script générant du perlin noise sur un plane, puis déformer le mesh en fonction de ce noise, il est possible de récupérer ce résultat pour la surface :



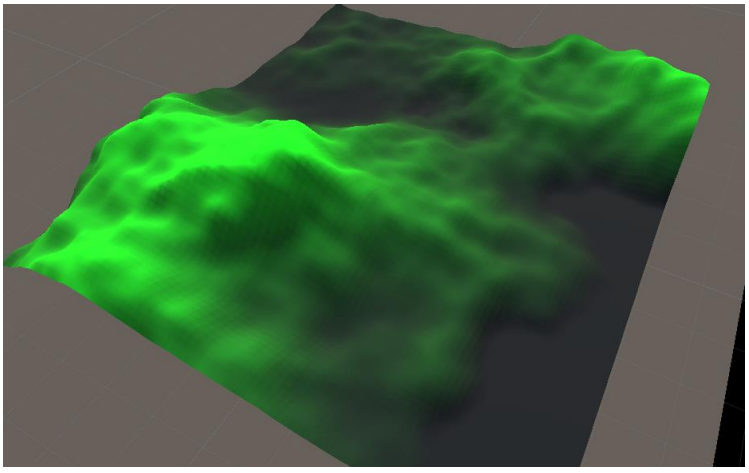
*A gauche, deux perlins noises générés et multipliés entre eux. A droite, les perlins noises ont été déformés et obtiennent une apparence nuageuse, réalisés en appliquant plusieurs paramètres supplémentaires.*

Pour mieux comprendre comment ce résultat peut se produire on peut se référer à [cette vidéo] qui explique très bien le principe. En résumé, il s'agit de combiner trois noises avec des paramètres différents, comme montré sur le schéma ci-contre.



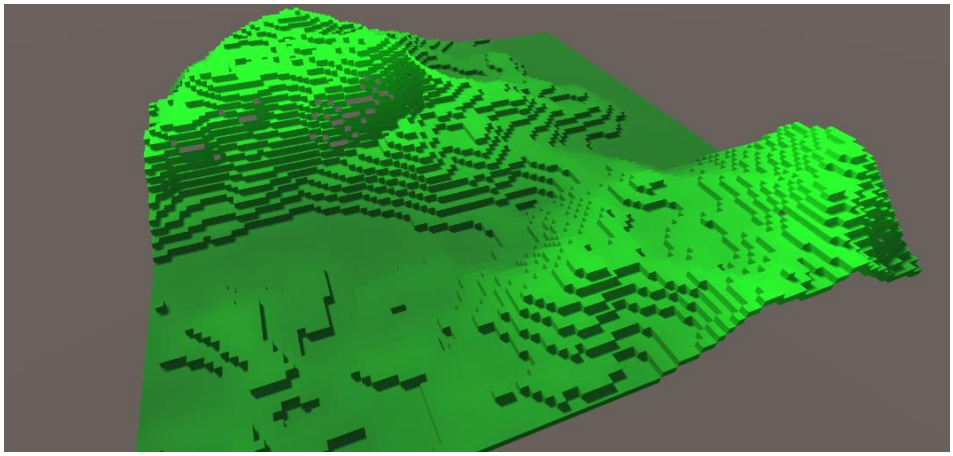
Pour mieux comprendre comment le système fonctionne, le noise est interprété comme une courbe dont la hauteur représente les valeurs des pixels, de 0 (noir) à 1 (blanc), et la longueur une portion de la longueur du noise. La hauteur est nommée amplitude, et la longueur fréquence (comme pour les sons). Sur le schéma, la courbe rouge correspond à l'addition des trois courbes bleues :

- L'octave 1 est la courbe qui agira le plus sur le résultat final car elle a la plus grande amplitude. Elle définira le profil du relief.
- L'octave 2 représente une variation dans le relief. Son amplitude étant deux fois moins importante que l'octave 1, elle a par conséquent moins d'impact sur le relief. Cependant, la fréquence de cette courbe est deux fois plus élevée, ce qui provoque une variation plus importante dans les valeurs et rend le profil de la courbe plus chaotique.
- L'octave 3 est nommé comme "petits cailloux" et est la courbe ayant le moins d'influence, qui a deux fois moins d'amplitude que l'octave 2, mais qui permet l'apport d'un chaos plus important, donnant un aspect granuleux ajoutant une variation minime au profil final. Cependant, son influence résulte en une apparence moins linéaire et plus cohérente du terrain.



Un noise dit "nuage" est en fait composé de trois noises. Par rapport au premier, le second et le troisième ont respectivement une fréquence deux et quatre fois plus élevée, ainsi qu'une amplitude deux et quatre fois plus faible. On parle de lacunarity pour la fréquence affectée par l'octave, et de persistance pour l'amplitude affectée par l'octave.

Une fois que le cube a été défini comme unité du monde, l'aspect recherché prend place. La surface d'une zone du monde est créée :



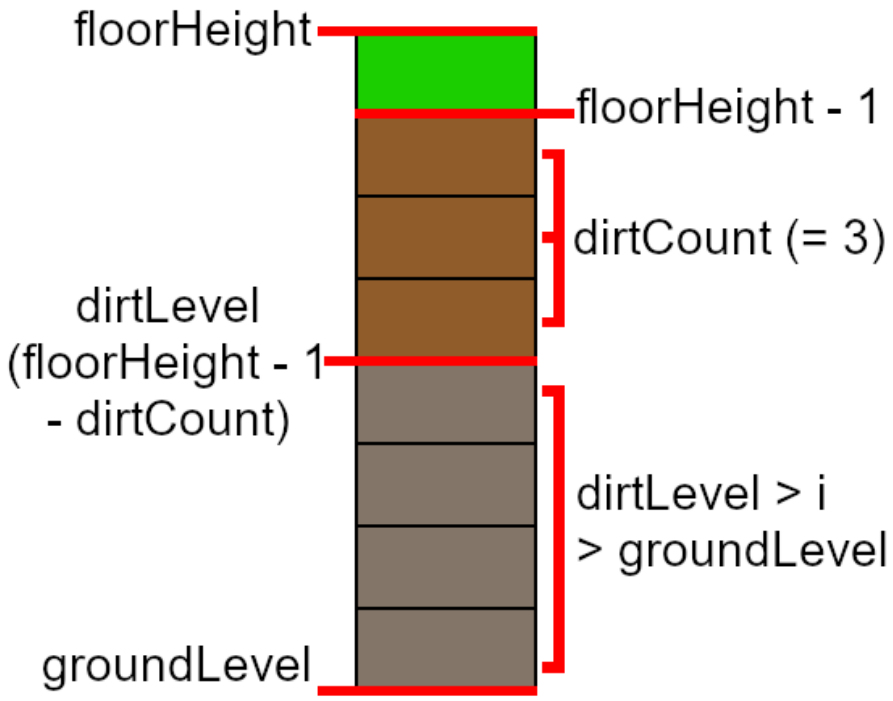
## Génération des couches sous la surface

La génération des couches présentes sous la surface nécessite plusieurs paramètres à prendre en compte : le niveau le plus bas sous la surface, la position de chaque bloc de la surface par rapport à ce niveau zéro, et la position à laquelle faire apparaître chaque cube par rapport à celui au-dessus.

Le niveau sous la surface peut être variable et n'aura pas de grande influence sur l'environnement, à partir du moment où la hauteur définie est suffisante pour remplir la différence entre les blocs ayant l'altitude la plus élevée et la plus faible.

La position des blocs de la surface étant définie par les paramètres de génération par rapport au noise, elle est accessible via une variable devant être utilisée pour créer un ensemble logique.

Pour pouvoir générer un bloc sous la surface, il faut prendre en compte sa position et le volume du bloc crée. Dans le cas présent, l'unité est un cube avec une origine présente au centre de son volume. Il est alors nécessaire de générer le premier cube de terre à la position du bloc de surface -1. Pour les suivants, la même logique est appliquée par rapport au bloc présent au-dessus, jusqu'à atteindre la niveau zéro avec un autre type de bloc (roche) une fois que les trois couches initiales (terre) sont placées.

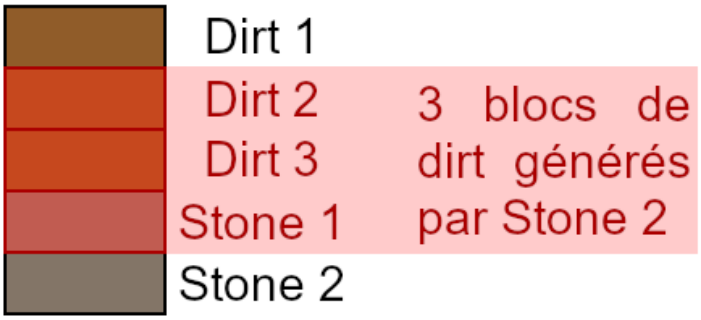



En guise de test de la théorie, une profondeur d'un bloc de terre a été générée et s'est révélée efficace. L'adaptation sur le code existant a été un peu plus complexe : dans un premier temps, il était possible de générer les blocs souhaités sous la surface, mais cela ne s'appliquait pas aux blocs de sable.

En attendant, la génération des couches de roches a été plus simple : à partir du niveau de la surface, il suffit alors de réduire de 1 pour la surface puis de 3 pour les couches de terre pour générer les blocs à partir des valeurs initiales du noise jusqu'au niveau zéro de la carte. Ceci en place, il était alors possible de ne pas toucher du tout au code initial pour générer la terre depuis la roche, en ajoutant 3 couches au-dessus du niveau le plus haut.

Cependant, cette approche comprenait une faille : chaque bloc de pierre générait 3 blocs de terre, ce qui provoquait une superposition et une génération de blocs inutiles.

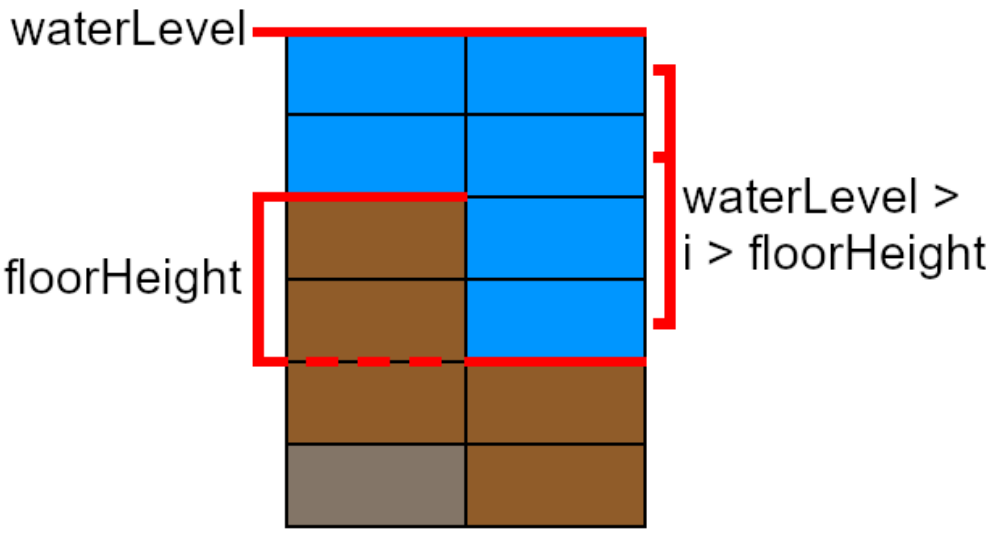
```
// Stone Generation
if ((int)Mathf.Floor(curveHeightMultiplier.Evaluate(noise[x, y]) * heightMultiplier) >= groundLevel)
{
    for (int i = (int)Mathf.Floor(curveHeightMultiplier.Evaluate(noise[x, y]) * heightMultiplier) - 4; i > groundLevel; i--)
    {
        GameObject stoneCube_clone = Instantiate(cubes[3]);
        stoneCube_clone.transform.position = new Vector3(x, i, y);

        // Dirt Generation - Under Every Block
        for (int j = 0; j < 3; j++)
        {
            GameObject dirtCube_clone = Instantiate(cubes[0]);
            dirtCube_clone.transform.position = new Vector3(x, i + j + 1, y);
        }
    }
}
```

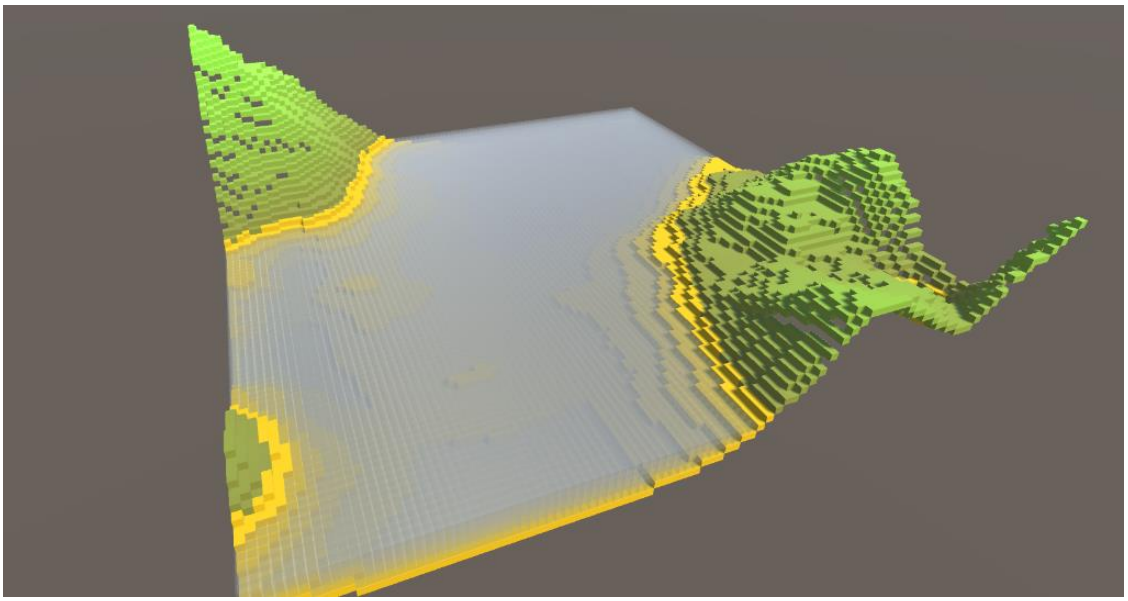


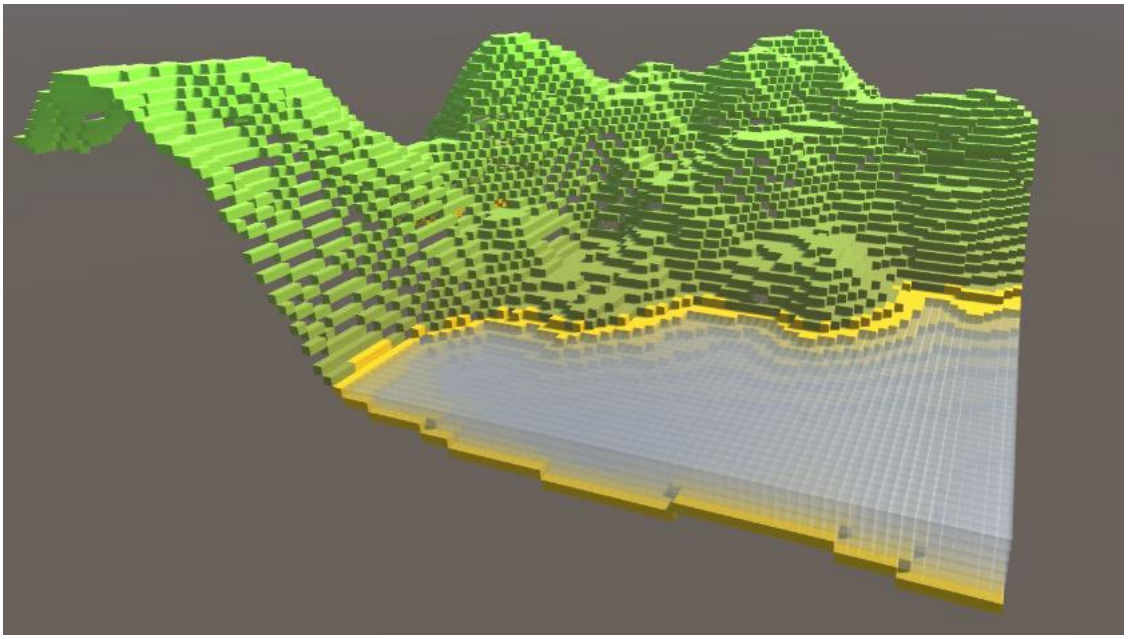

Cette solution n'étant pas optimale, le code a été légèrement modifié pour pouvoir générer la terre depuis la surface, ce qui permet également de modifier l'épaisseur de la couche via une variable.

## Génération de l'eau

La solution apportée a été de définir un niveau pour l'eau, et de générer les blocs correspondants en partant de la surface jusqu'au niveau défini. Le code a pu être optimisé très facilement après avoir établi celui des couches souterraines, étant donné qu'il s'agissait de la même logique à appliquer dans l'autre sens.



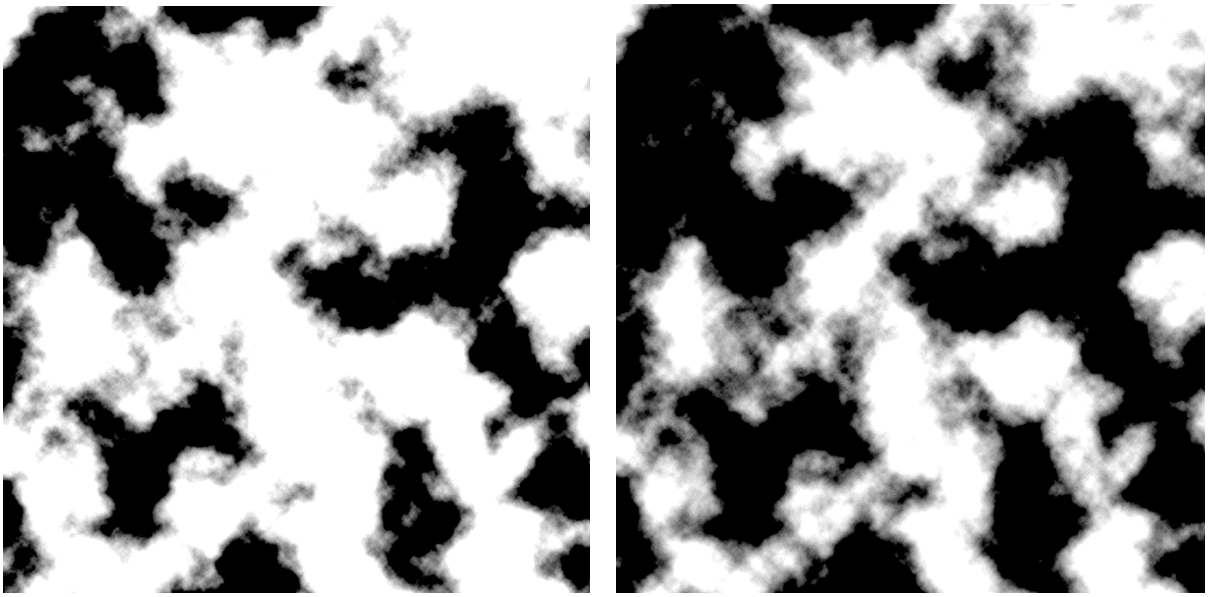




## Génération de sable et de neige

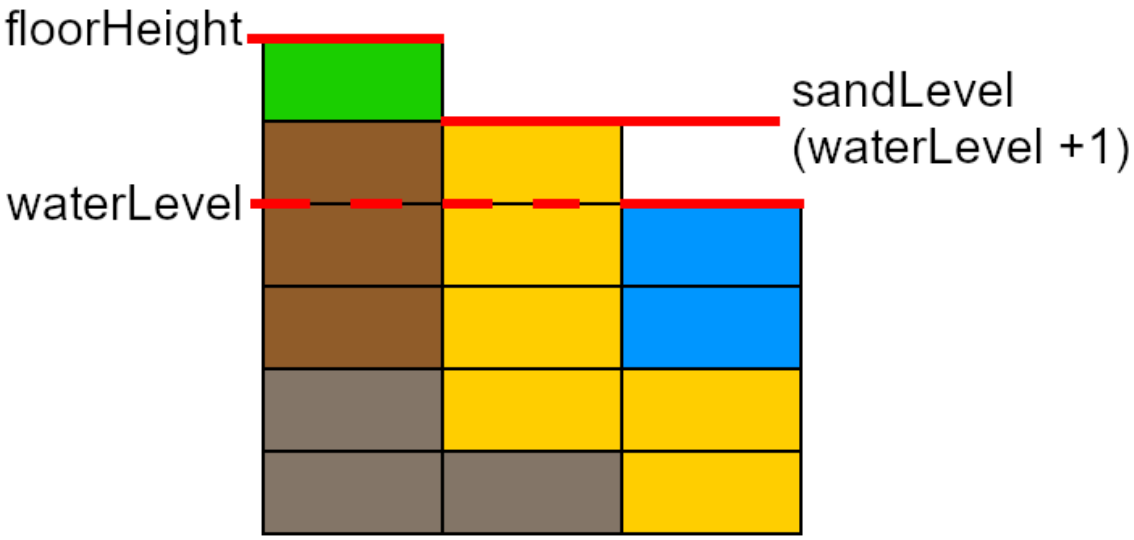
L'objectif était de définir, grâce au noise initial, la zone où le sable devait apparaître à la place des blocs de surface afin de l'intégrer pendant la génération du monde.

Ainsi, le noise a été légèrement modifié afin de créer plus de zones grises. En excluant le blanc, le sable peut être exploité via les teintes de noir et gris, formant un ensemble cohérent et moins linéaire.



*Noise continent (blanc) /océan (noir) à gauche ; Noise sable à droite (noir à gris, en excluant le blanc).*

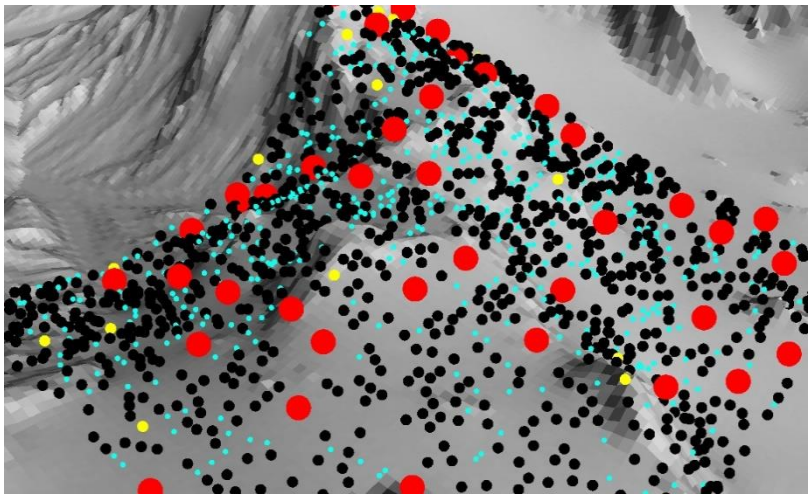
Cette manière de générer le sable n'a finalement pas apporté le résultat attendu, et nous avons choisi de fixer la valeur du sable au niveau de l'eau avec un ou deux blocs de différences.




Pour la neige, le même système a été reproduit. Au lieu de vérifier les blocs à partir de la hauteur de l'eau, celui-ci démarre d'une nouvelle variable déterminant l'altitude d'apparition de la neige puis remplace les blocs d'herbe par des blocs de neige.

# Génération de zones forestières

La première étape sera de placer l'emplacement de base des arbres, représenté par un block, mais son encombrement final doit être pensé également. Pour cela, je me suis rapproché du schéma suivant (source) :



## Théorie

Pour définir les zones dans lesquelles des arbres doivent apparaître, il est nécessaire de d'abord prendre en compte plusieurs paramètres.

Premièrement, la taille de l'objet généré. Ce paramètre influe directement sur les suivants, il faut donc définir la forme de l'objet et son échelle par rapport au terrain.

Ensuite son emplacement, les arbres devront se limiter à une zone précise. On ne veut logiquement pas d'arbre dans l'océan, il faudra donc le prendre en considération lors de la création de cette zone forestière, de même que pour les sommets des montagnes où il faudra définir une hauteur maximale pour le placement, afin de ne pas retrouver des arbres au sommet des montagnes enneigées.

Enfin, la densité de la zone, directement liée à l'encombrement de l'objet généré précédemment évoqué. Plus cette zone sera grande, plus la densité d'arbres sera faible. A l'inverse, plus elle sera petite, plus il y aura d'arbres. Pour obtenir un résultat convainquant, il faut également faire attention à ce que les modèles 3D ne se chevauchent pas.

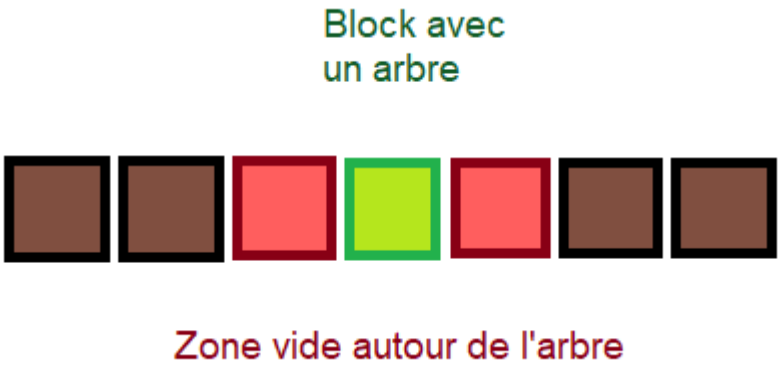
## Intentions

Tous les points seront ignorés à l'exception des points rouges (voir l'image ci-dessus), qui correspondent chacun à un arbre. Ceux-ci sont entourés d'une zone vide dans laquelle aucun autre arbre ne peut apparaître, représentant la distance minimale entre chaque individu. Leur placement est géré par cette distance et par la densité.

Dans un premier temps, le noise sera utilisé pour identifier des zones dans lesquelles les arbres pourront apparaître. Pour cela, on va regarder la valeur de chaque "case" générée et les comparer avec une valeur étalon. Sachant qu'un noise génère des valeurs entre 0 et 255, et en prenant par exemple une valeur étalon de 100, alors il a possibilité de faire apparaître un arbre si la valeur de la case est comprise entre 100 et 255. Il faut ensuite créer une zone autour de cet arbre pour empêcher d'autres arbres d'apparaître. Enfin, les arbres pourront être déployés.

### En pratique

Finalement, nous avons pu générer les arbres en fonction d'un noise définissant la disposition des forêts. Lorsque le noise indique une zone dans laquelle il est possible de générer des arbres, une valeur permet d'influer sur les chances qu'a un arbre d'apparaître à cet endroit. Ainsi, il y a un pourcentage de chances pour chaque bloc de chaque zone compatible pouvant générer un arbre.

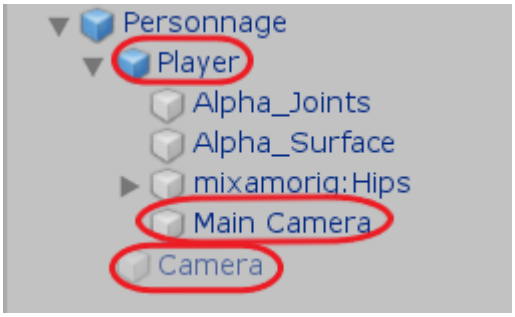


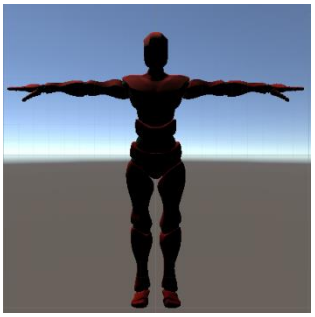

Les arbres sont des modèles 3D non-générés procéduralement. Seul leur placement dans le monde est procédural, utilisant un système de pourcentage qui peut varier selon plusieurs critères. La proximité avec d'autres arbres, qui réduit les chances d'apparition d'un autre arbre si il y en a un sur les cases alentours. Plus ces cases sont proches de l'arbre, plus les probabilités de génération sont faibles. Ainsi, cela permet d'avoir un environnement cohérent en laissant un espace suffisant entre chaque arbre, et en les disposant dans les milieux appropriés.

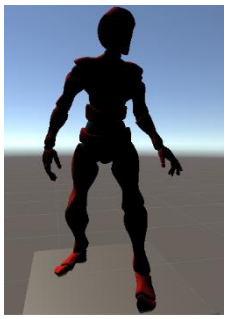
## Outil de navigation dans l'environnement

Pour visiter le monde généré, une capacité de déplacement doit être mise en place. Celle-ci doit permettre de naviguer n'importe où et par n'importe quel moyen : en marchant et prendre en compte les collisions, ou en volant et ignorer ces collisions. La deuxième solution permet la potentielle visite des souterrains de manière plus pratique.

Le personnage crée dispose des deux options : il s'agit d'un personnage physique à la troisième personne pouvant troquer sa caméra pour un point de vue libre.





Le personnage contient ainsi un prefab « Player », contenant un collider, une caméra et un modèle animé. Il peut se déplacer et sauter, orienter la caméra autour du modèle, et entrer en collision avec les blocs composant le terrain.

Toujours dans le personnage, mais à l'extérieur du prefab « Player », une caméra est créée et désactivée par défaut. Via un script, celle-ci peut être activée à la place du prefab « Player » pour pouvoir ensuite la déplacer librement dans la scène. En appuyant sur la touche définie dans l'éditeur, cette caméra s'active et se désactive à la volée.

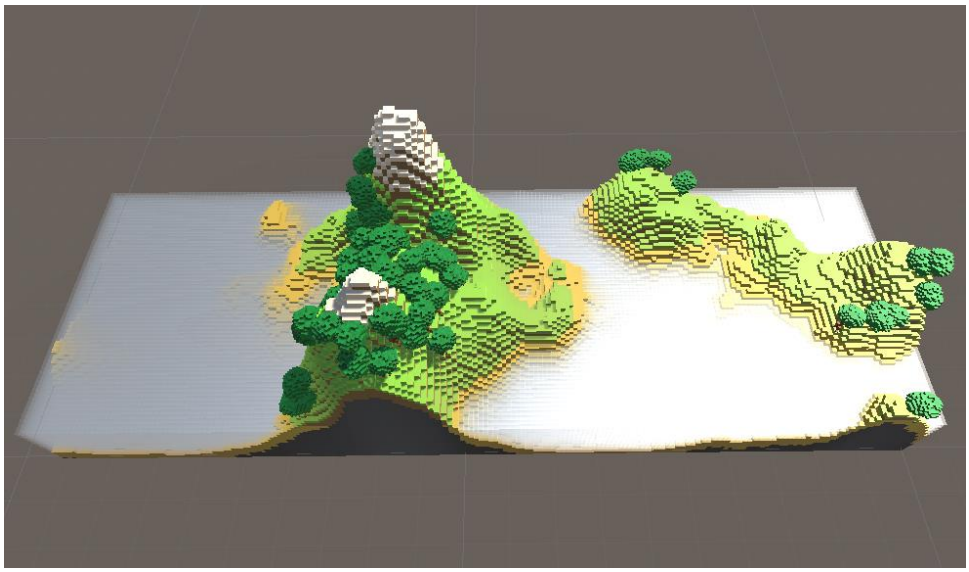
## Chunks

Le noise n'étant pas aléatoire, l'idée pour réaliser des chunks, et par la suite une génération infinie, est de générer le noise correspondant aux coordonnées adjacentes au noise de départ, puis de générer les blocs correspondants. En résumé, en ajoutant un offset fixe, il est possible de générer la continuité du noise de départ.

L'offset correspond alors à un Vector2 ne pouvant être qu'une valeur entière. Ainsi, il peut être interprété par les fonctions suivantes :

- Axe horizontal : offset.x * mapWidth
- Axe vertical : offset.y * mapHeight

La technique fonctionne mais présente un défaut : les hauteurs minimales et maximales des chunks sont calculées indépendamment. Ce point permettait une certaine lisibilité pour le prototypage d'une seule zone, mais demande alors de fixer ces valeurs pour permettre de générer plus d'un chunk. Dans le cas contraire, la hauteur définie par le noise est indépendante à chacun, générant un environnement incohérent. Après quelques réglages des différents paramètres de génération, et l'ajout de cette valeur fixe pour les hauteurs minimale et maximale, les chunks sont créés et forment un environnement cohérent :



*Trois chunks générés côte à côte.*

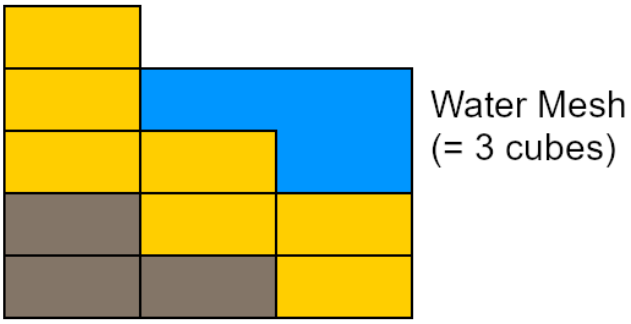
Pour en terminer avec cette partie, il faut maintenant faire en sorte de générer chaque chunk en fonction de la position du joueur dans le monde et désactiver les chunks non visibles, permettant ainsi d'améliorer les performances du système.

## Performances

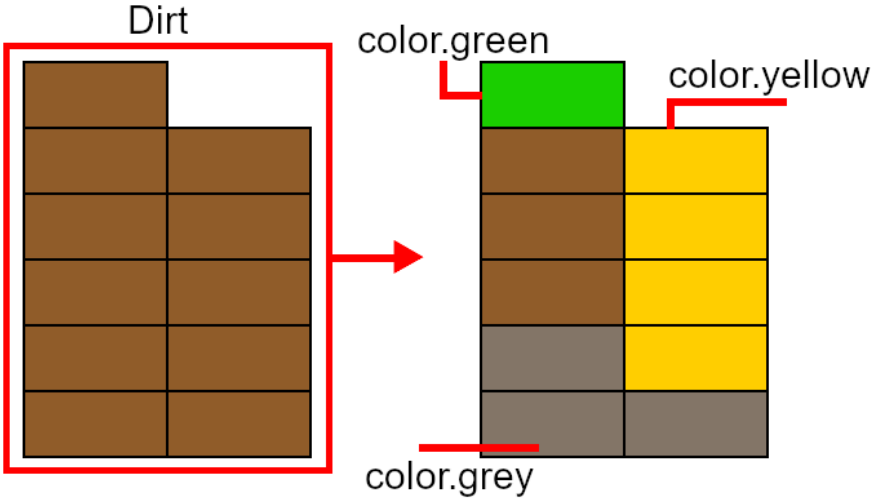
Les performances ont été l'un des points les plus compliqués à améliorer. La génération demandait beaucoup de temps à partir du moment où les blocs de terre, de roche et d'eau ont été intégrés, ajoutant ainsi un grand nombre de polygones supplémentaires à calculer et compliquant les manipulations nécessaires aux différents essais de génération.

Le système adopté a alors été de supprimer tous les blocs non-visibles, à l'exception des blocs d'eau. De cette façon, pendant que la génération des souterrains n'est pas prise en compte, le gain en performance est non négligeable. Les blocs d'eau, quant à eux, sont converties en meshes de 36 000 vertices, correspondant à une valeur proche de la limite gérée par Unity, toujours dans l'optique d'améliorer les performances. Cette approche permet au moteur de ne pas avoir à gérer un grand nombre de cubes inutilement.

De la même manière, les blocs d'eau ont été convertis en un seul mesh, permettant d'alléger encore le nombre de cubes. Les blocs sont tous générés puis placés dans un objet vide qui dispose du matériau de l'eau. Leurs informations sont ensuite récupérées et servent, à l'aide la fonction CombineMesh, à créer un mesh. Les performances durant la navigation sont ainsi améliorées, tandis que celles durant la génération sont sensiblement les mêmes.

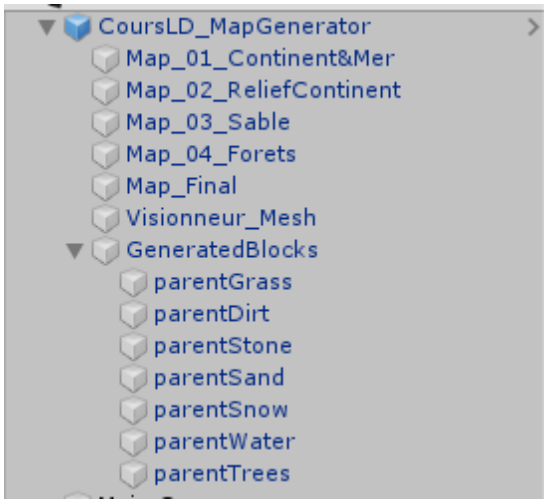



A la place d'utiliser un cube différent pour chaque matériau, seul le cube de terre, est utilisé pour toutes les surfaces à l'exception de l'eau. En utilisant les mêmes paramètres que pour définir les cubes séparément, il est possible d'influer sur la couleur du matériau du cube, permettant ainsi d'alléger les calculs. Unity n'ayant que deux objets différents à générer (terre et eau), les performances s'en trouvent améliorées. Dans l'hypothèse où des textures seraient utilisées pour chaque matériau, ce système peut s'y adapter très facilement. Par la suite, le bloc de terre a été remplacé par un bloc neutre, permettant l'uniformisation de tous les blocs solides.

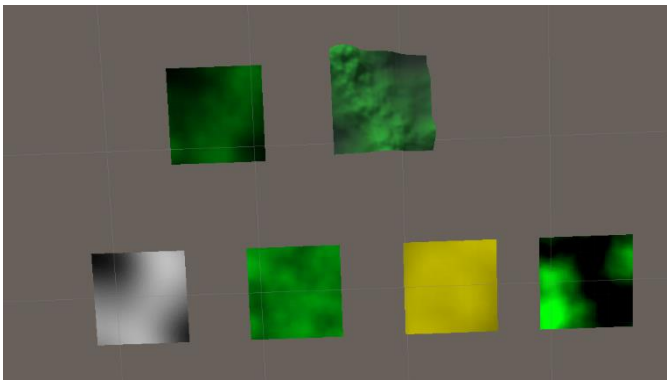


Toujours dans le but d'améliorer les performances, chaque bloc d'un chunk a été stocké dans un tableau à trois dimensions, contenant pour chaque case le nom du bloc en question. Une fois tous les calculs effectués, la génération se fera à partir des valeurs contenues dans ce tableau. L'inconvénient de cette technique est qu'il n'est plus possible de convertir plusieurs blocs en un seul mesh de la même manière, ce qui rend les manipulations précédemment décrites pour la génération du mesh de l'eau obsolètes.

## Etendues des possibilités du générateur




Le générateur et ses paramètres sont accessibles via un objet, « CoursLD_MapGenerator ». Celui-ci contient, en enfants, les différents planes sur lesquels sont dessinés les noises. Ils permettent de mieux visualiser l'apparence du monde à la fois en 2D et en 3D. L'objet se compose aussi d'un dossier « GeneratedBlocks » qui contient lui-même plusieurs objets nommés « parent[…] ». Il s'agit ici d'un système de rangement automatique permettant de ranger les blocs par type à leur génération.



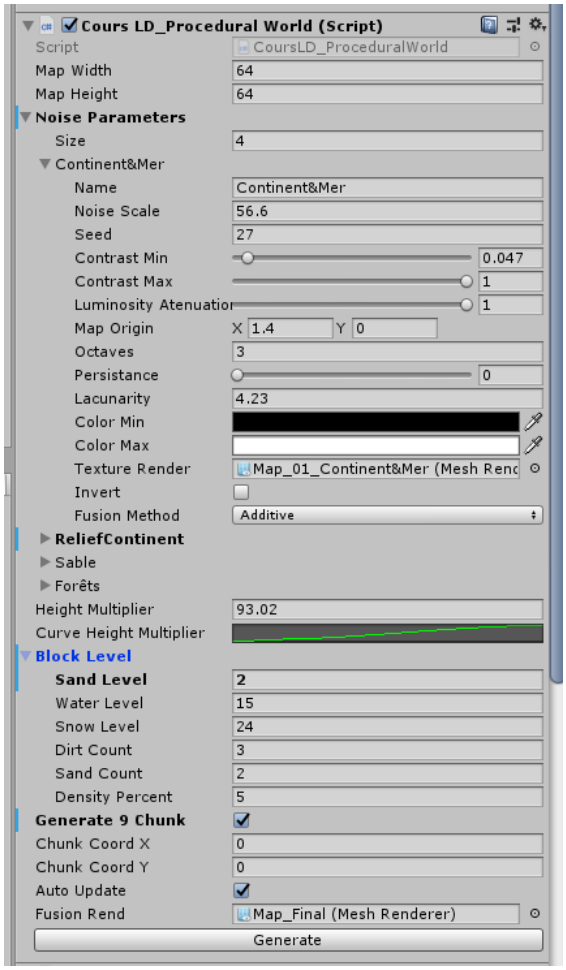
*Apparence du « CoursLD_MapGenerator » dans la scène*

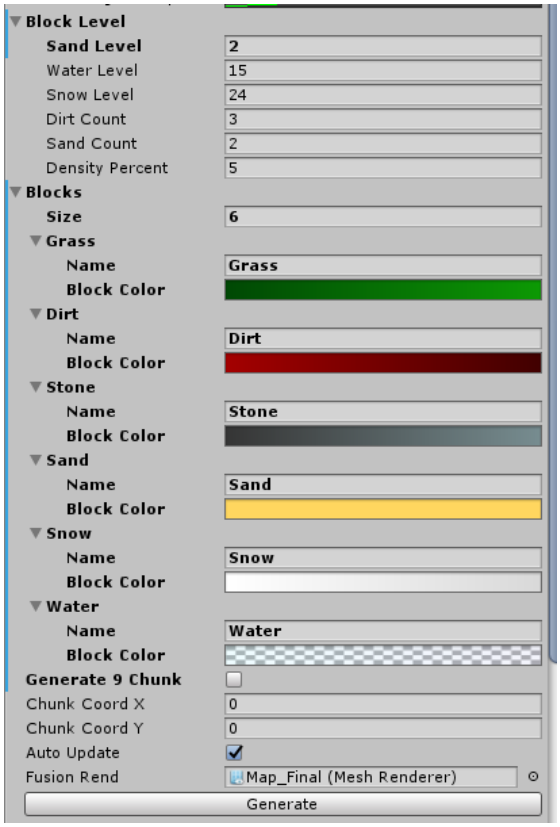
Le générateur contient plusieurs scripts, dont « CoursLD_ProceduralWorld » qui permet de paramétrer généreusement tous les aspects disponibles du monde procédural généré :

- *MapWidth* et *MapHeight* sont deux variables qui définissent la taille des planes de visualisation, influant par conséquent sur celle d'un chunk.
- *Noise Parameters* regroupe les paramètres relatifs aux différents noises, actuellement au nombre de 4.
    - *Name*: Le nom du noise.
    - *Noise Scale*: L'échelle du noise
    - *Seed*: La seed du noise, permettant de changer le profil du monde

- *ContrastMin*: Valeur minimale du contraste, permettant d'influencer la définition des teintes de noir.
- *ContrastMax*: Valeur maximum du contraste, permettant d'influencer la définition des teintes de blanc.
- *LuminosityAttenuation*: Atténuation du contraste, permettant de favoriser des teintes de gris.
- *Map Origin*: Modifie l'origine du noise.
- *Octaves*: Le nombre d'octaves de déformation du noise (voir section « Création du terrain ») : plus grandes est la valeur, plus les paramètres <persistances> et <lacunarity> auront d'effets, en définissant de nombre de noises superposés.
- *Persistance*: Permet d'augmenter l'impact des noises superposés, en définissant la diminution de l'amplitude liée à chaque noise pour chaque octave.
- *Lacunarity*: Permet de définir l'augmentation de la fréquence liée à chaque noise pour chaque octave.
- *ColorMin*: Couleur correspondant à la valeur minimale du noise (noir par défaut).
- ColorMax: Couleur correspondant à la valeur maximale du noise (blanc par défaut).
- *Texture Render*: Le renderer du plane sur lequel se génère le noise.
- *Invert*: Inverse *ColorMin* et *ColorMax*.
- *Fusion Method*: Méthode de fusion des noises par rapport aux autres, si désirée. Le premier noise sera toujours fusionné en additif.

- *HeightMultiplier*: La hauteur maximale du bloc le plus haut. La variable est un float pour contrôler aussi le visionneur 3D dans la scène.



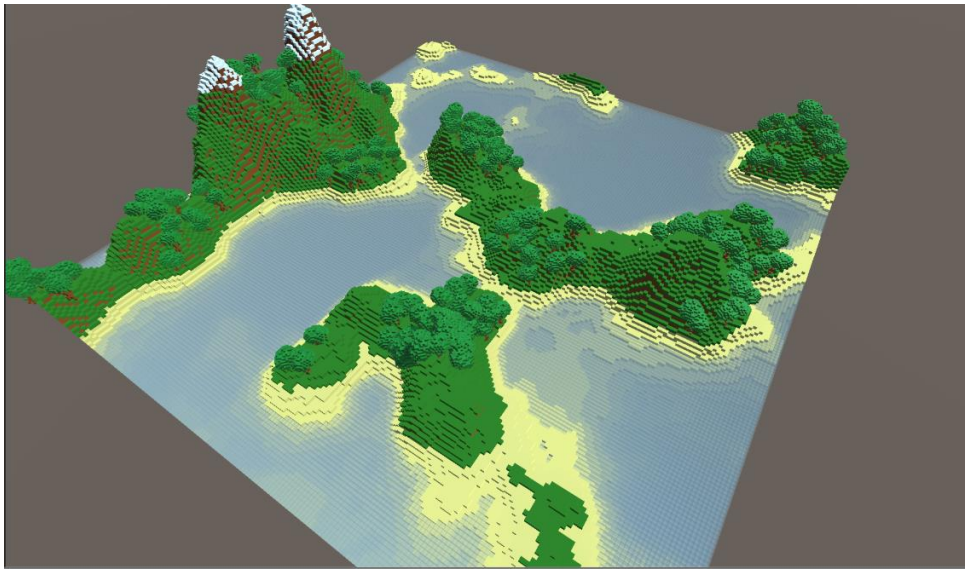



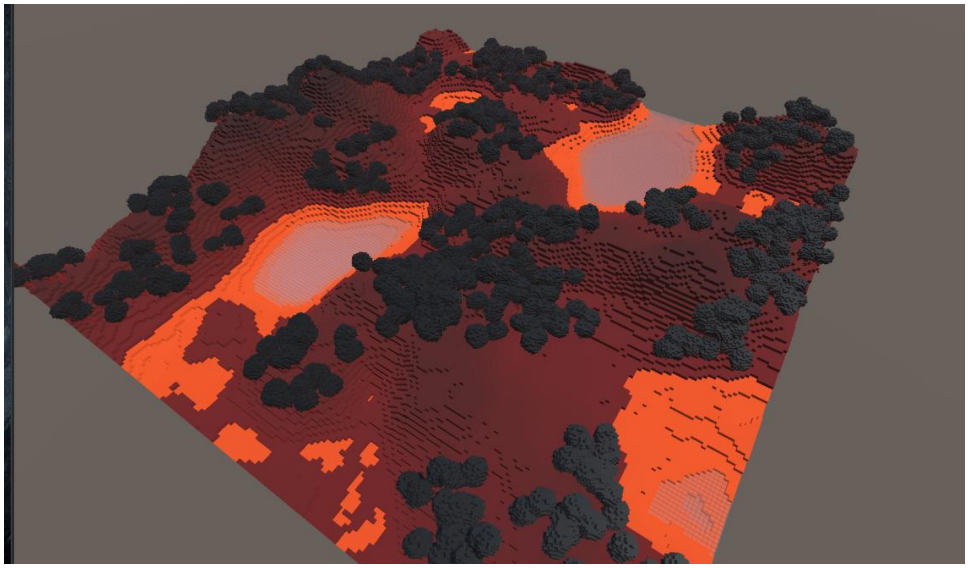

- *Curve Height Multiplier*: Courbe décrivant le profil du relief. N'accepte pas les valeurs négatives, et produit des résultats incohérents au-delà de 2 en abscisse.

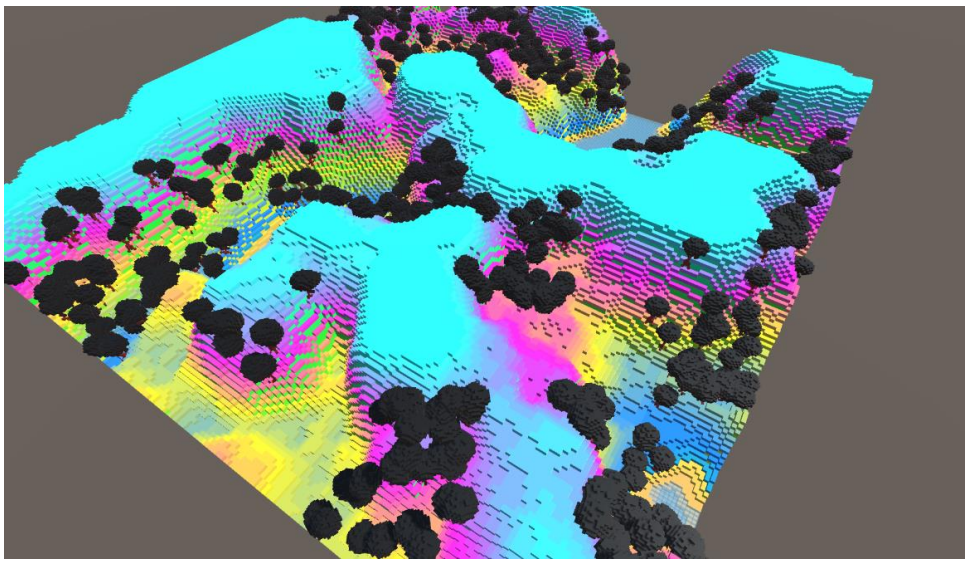
- *Block Level*: Paramétrage des niveaux d'apparition des blocs.
    - *SandLevel*: Nombre de blocs de sable au-dessus de l'eau.
    - *WaterLevel*: Niveau de l'eau par rapport au niveau zéro du monde.
    - *SnowLevel*: Nombre de blocs de neige depuis le point le plus haut.
    - *DirtCount*: Nombre de blocs de terre générés sous la surface.
    - *SandCount*: Nombre de blocs de sable générés sous une surface de sable.
    - *Density Percent*: Densité des arbres dans les biomes de forêts. Agit comme un pourcentage pour chaque bloc dans la forêt. 0 = Aucun arbre ; 100 = Un arbre sur chaque blocs.
- *Blocks*: Tous les types de blocs pouvant être générés, et leur couleur associée.
    - *Name*: Nom du bloc.
    - *Block Color*: Couleur du bloc en dégradé, permettant de varier sa teinte en fonction de sa hauteur, avec la valeur minimale vers le bas et la valeur maximale vers le haut.
- *Generate 9 Chunk*: Permet de générer 9 chunks au lancement. Cette fonction génère beaucoup de blocs et réduit drastiquement les performances.
- *Chunk Coord X & Chunk Coord Y:* Debug permettent de changer la visualisation du chunk.
- *Auto Update*: Met à jour automatiquement les noises et leur visualisation à chaque modification.
- *Fusion Rend*: Renderer du plane accueillant la représentation du résultat final.

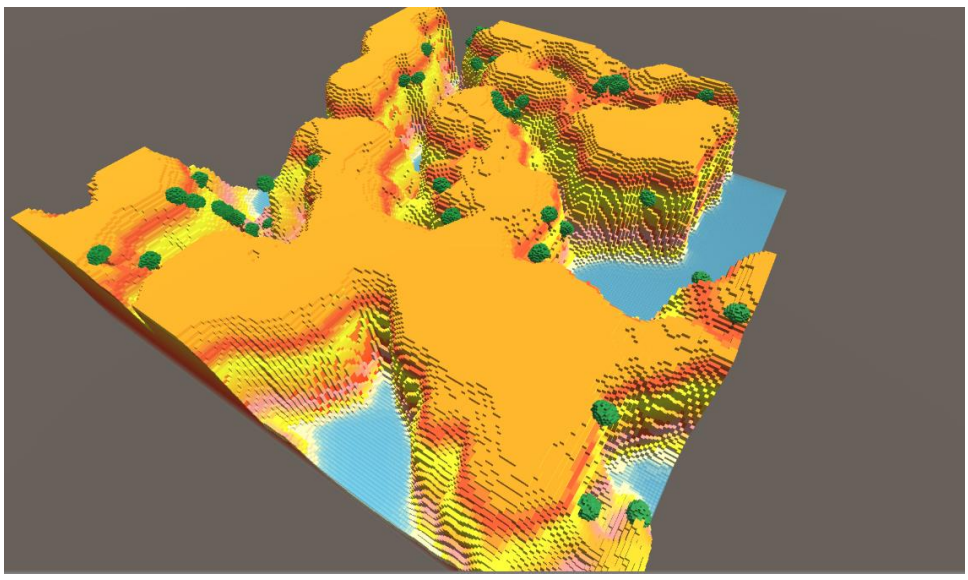
## Rendu final

A l'heure actuelle, le projet permet de générer un ou 9 chunks, avec tous les paramètres de configuration énumérés et expliqués plus haut qui ont permis d'obtenir divers résultats en les faisant varier, pouvant ainsi faire émerger des environnements très différents :









## Propositions d’améliorations

- Générer des chunks de manière infinie.
- Améliorer les performances en convertissant les valeurs du tableau en mesh.
- Générer des souterrains accessibles depuis la surface.
- Créer des variations dans la génération du sable et de la neige pour procurer un effet plus cohérent et réaliste.
- Convertir les blocs en un mesh pour améliorer les performances, pour l’eau puis pour les autres blocs.
- Créer différents biomes : forêt, désert, montagne.

## Tâches

### Tâches (22/01)

- Quentin : Génération des couches présentes sous la surface (terre et roche).
- Pierrick : Génération de blocs d’eau entre le niveau défini et les blocs solides en dessous.
- Ségolène : Génération des zones attribuées aux arbres.
- Antoine : Création d’une caméra permettant de naviguer dans l’environnement.
- Florian : Génération des blocs de sable par rapport aux paramètres de l’eau.

### Tâches (29/01)

- Quentin : Génération de la couche de neige sur les sommets et code review.
- Pierrick : Amélioration de la génération du sable.
- Ségolène & Florian : Poursuite de la génération des zones attribuées aux arbres.
- Antoine : Amélioration du personnage et sa documentation.