Edimap

Documentation

Synnaxium Studio
Head office
99 A Boulevard Constantin Descat
59200 Tourcoing
(+33)3.74.09.69.80

Synnaxium Studio

SOMMAIRE

Présentation

Edimap est à la fois un éditeur de carte et un éditeur de niveau. Quelle est la différence, pensez-vous ?
C'est très simple : Edimap gère à la fois la conception de l'environnement et l'intégration du gameplay. Pour cela, Edimap propose une gestion entièrement centrée autour du préfab.

Chez Synnaxium Studio, Edimap est notre outil de production fondamental, utilisé au quotidien pour accélérer la production de nos divers jeux (principalement du Serious game).

Edimap est capable de gérer la plupart des problématiques que vous rencontrerez lors de l'élaboration de jeux basés sur un système de cases, qu'il s'agisse d'un plateformer, d'un jeu isométrique, de 2D ou de 3D.

Nous vous souhaitons une bonne lecture !

Retrouvez-nous sur Discord !

https://discord.gg/tHWvyraN3B

Pour commencer

Introduction

L'objectif de cette partie est que vous puissiez utiliser Edimap rapidement sans forcément s'arrêter sur les détails techniques.

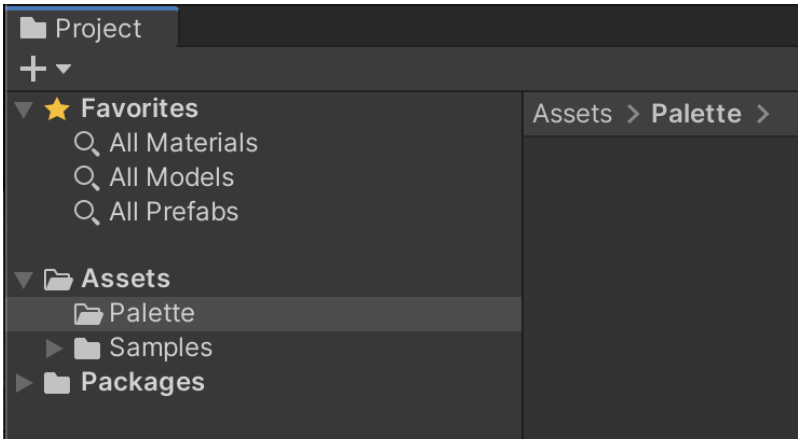
Une fois que vous aurez pris en main Edimap, la suite de la documentation viendra expliquer chaque fonctionnalité.

⚠ **Si vous constatez une erreur sur "group.Name", pensez à passer en C# 4.x ou supérieur. Le futur c'est maintenant !**

⚠ **Dans le cas ou votre projet est en URP ou en HDRP, il se peut que vous deviez modifier les matériaux des previews (cf.** Configuration globale**).**

Palette

Commençons par initialiser la palette. La palette contient la liste des préfabs que vous pourrez utiliser pendant la création de votre carte. Cette palette est matérialisée par un dossier au sein de votre projet.



Créez tout d'abord un dossier, là où vous le souhaitez, au sein de votre projet. Vous n'êtes pas obligé que ce dossier soit à la racine du projet.

i Notez bien que Edimap ne vous impose pas de chemin particulier, vous pouvez donc créer d'autres sous-dossiers pour des raisons d'organisation.

Premiers préfabs

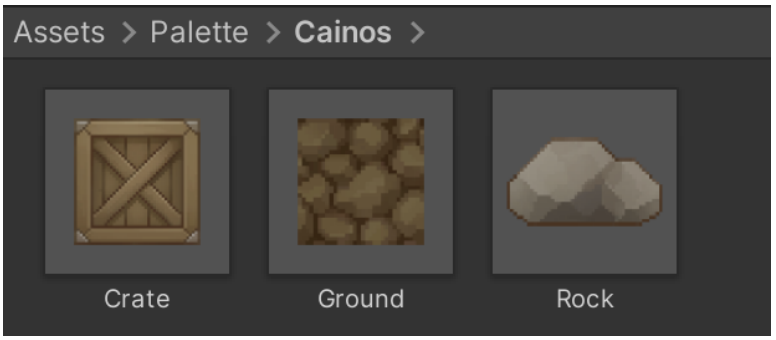
Pour cette partie, nous allons nous limiter au cas d'usage le plus simple pour rapidement pouvoir utiliser l'outil, à savoir les jeux 2D.

Créez des préfabs avec un composant *SpriteRenderer* et placez-les dans un sous-dossier de la palette (dans notre exemple le sous-dossier a le chemin *Assets/Palette/Cainos*).

⚠ **Le sous-dossier est important, car la palette est constituée de dossiers de préfabs.**

Nous remercions Cainos de nous autoriser à utiliser leurs sprites. Il sont disponibles dans le dossier suivant :

Assets
↳ Synnaxium
↳ Edimap
↳ Examples
↳ Platformer 2D
↳ Graphics
↳ Cainos
↳ Pixel Art Platformer - Village Props
↳ Texture



Votre première palette est maintenant créée. Bien évidemment, cette première installation est très naïve et ne nous emmènera pas très loin, mais chaque chose en son temps.

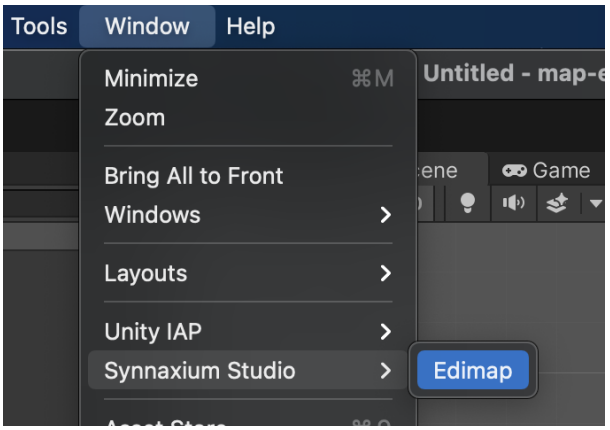
Dans notre exemple, nous avons choisi les sprites :

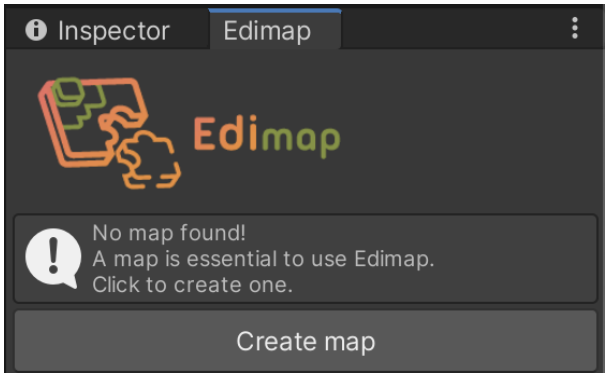
- TX Tileset Ground C TL
- TX Village Props Rock B
- TX Village Props Crate Large

Fenêtre Edimap

Ouvrez une nouvelle fenêtre Edimap à l'aide du menu :

Window ➜ Synnaxium Studio ➜ Edimap

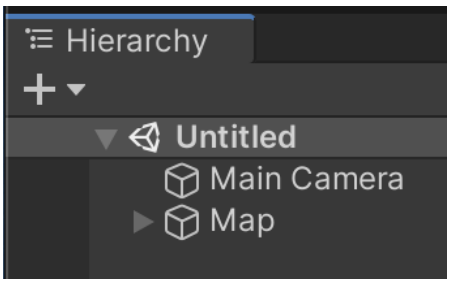




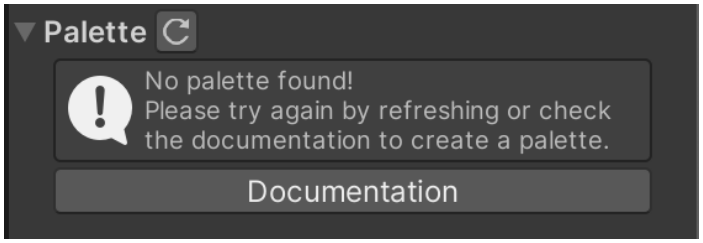
Edimap apparaîtra au même niveau que la fenêtre *Inspector*, et vous pourrez constater qu'il y a un message d'information vous indiquant qu'aucune carte n'est présente dans votre scène.

On commence donc par en créer une nouvelle en cliquant sur le bouton **Create map**.

Dès lors, un *GameObject* nommé “Map” apparaît dans votre scène.

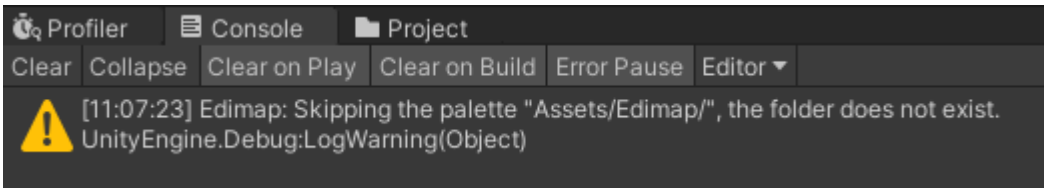


i Il s’agit du principal objet utilisé par Edimap pour créer une carte, les tuiles que vous ajouterez seront comprises dans cet objet.



Par défaut, Edimap utilise un chemin prédéfini pour localiser votre palette, or s’il n’y a aucune palette dans le dossier en question, un message d’information apparaîtra dans la fenêtre Edimap.

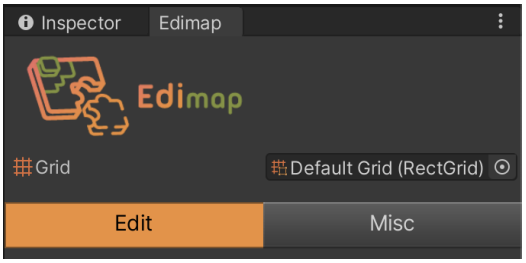
En parallèle, un warning devrait apparaître au sein de votre console :



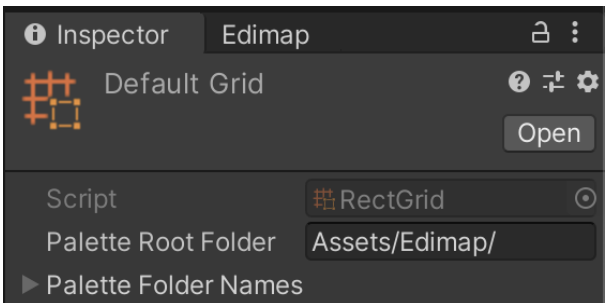
Dans notre exemple, nous préférons créer notre propre configuration.

La grille

Une fois l’étape précédente réalisée, vous pourrez remarquer qu’une nouvelle référence vers un *ScriptableObject* est apparue.



Il s'agit de la grille par défaut utilisée par Edimap, celle-ci pointe vers le dossier par défaut.

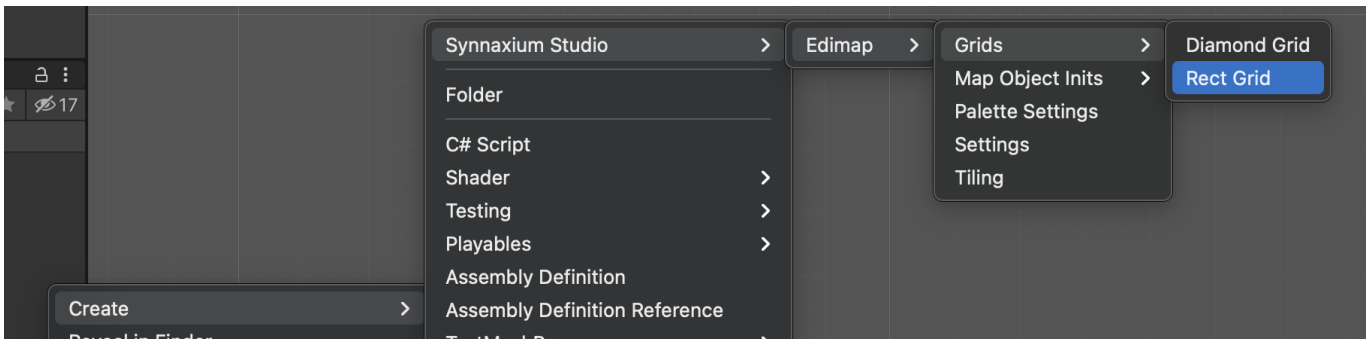


Nous allons remplacer cette grille par une nouvelle créée spécialement pour notre exemple.

Pour notre première grille, nous utiliserons une grille permettant de faire des formes de types rectangulaires.

Commencez par choisir le dossier dans lequel vous souhaitez créer la grille, puis :

Clic droit ➜ Create ➜ Synnaxium Studio ➜ Edimap ➜ Grids ➜ RectGrid

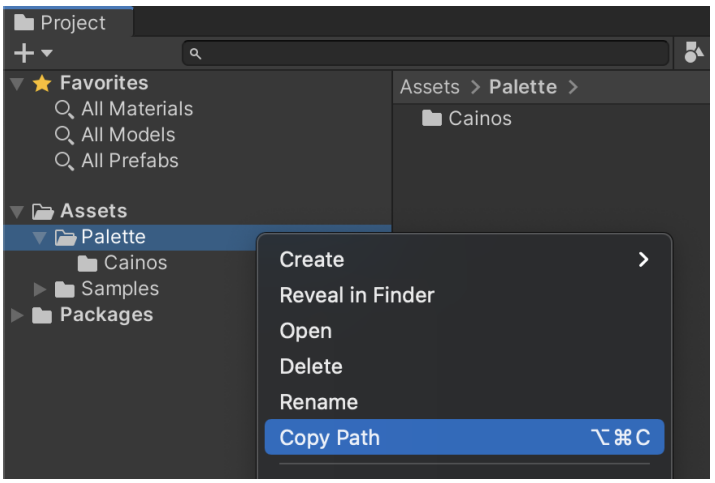


i Vous pouvez créer votre grille à l'emplacement de votre choix dans le projet.

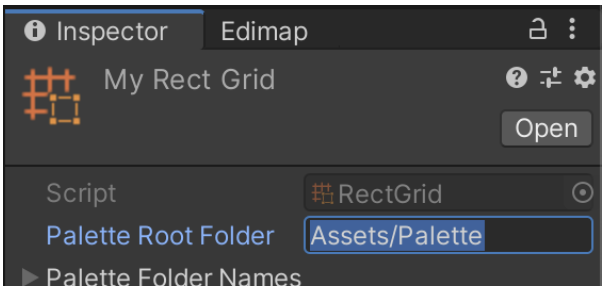
i Une grille peut être partagée entre plusieurs cartes et chaque grille peut pointer sur sa propre palette. Cette fonctionnalité s'avère très pratique lorsqu'il s'agit de conserver des tailles raisonnables de palette. Ainsi, on pourrait avoir une palette pour le biome "Désert" et une autre pour le biome "Forêt".

Une fois notre grille créée nous allons la configurer à l'aide de son *Inspector*.

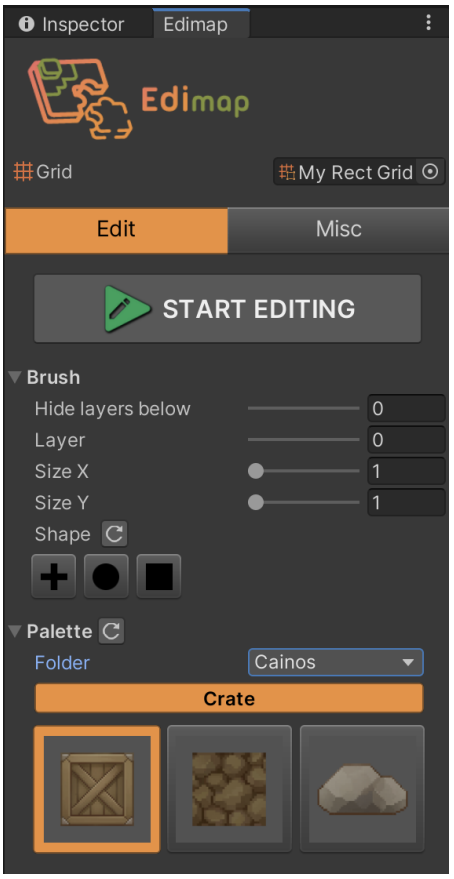
Copiez le chemin d'accès du dossier de votre palette (à ne pas confondre avec le sous-dossier Cainos).



Ensuite, coller ce chemin dans le champ **Palette Root Folder** de votre grille.



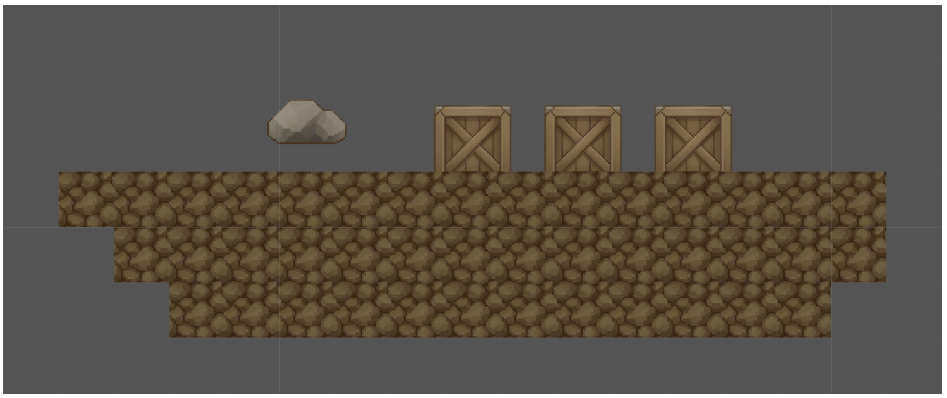
Retournez maintenant sur Edimap et sélectionnez votre grille fraîchement créée dans le champ **Grid**.



Si tout a été réalisé correctement, la palette devrait maintenant s'afficher. Si ce n'est pas le cas vérifiez que le chemin renseigné dans votre grille correspond et que le dossier de votre palette contient bien vos préfabs avec des *SpriteRenderer*, puis cliquez sur le bouton ⟳ au niveau de la palette.

Vous êtes maintenant prêt à démarrer.

Cliquez sur **START EDITING** et commencez à placer vos différents prefabs dans la scène en effectuant des clics gauches. Edimap place dans la scène l'objet actuellement sélectionné au sein de la palette.

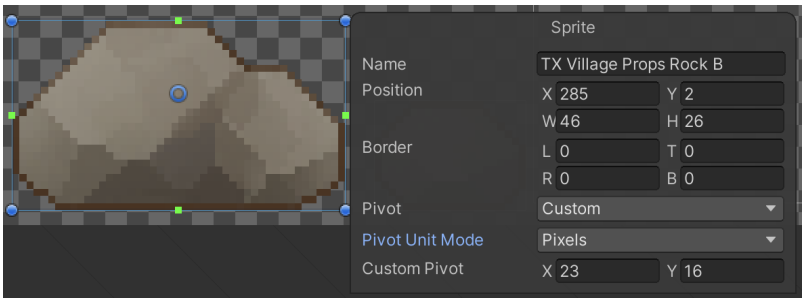


Pivot

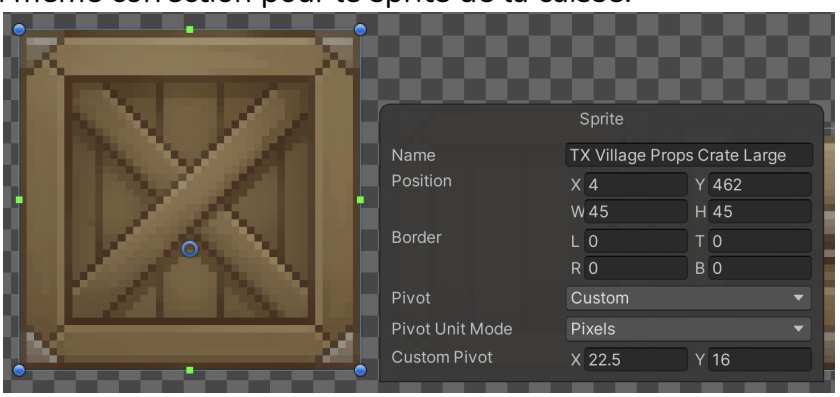
Les objets créés par Edimap sont placés au centre de leur case. Il faut donc en tenir compte pour le pivot de nos sprites.

Dans notre cas, nous remarquons que le rocher n'est pas au sol. Pour corriger le pivot il vous suffit de :

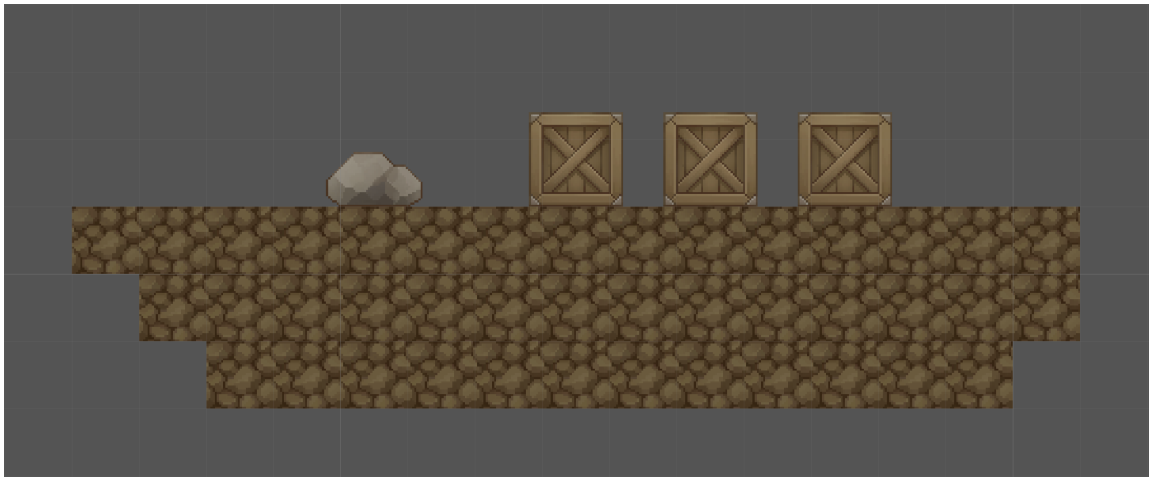
➔ sélectionnez le sprite de votre rocher,
➔ éditez en cliquant sur le bouton **Sprite Editor** de son *Inspector*,
➔ remplacez sa valeur de **Y** par **16 pixels**, car la case fait 1 unité de haut par défaut, et le sprite est configuré à 32 pixels par unité (16 pixels représente donc la moitié de la case).



Procédez à la même correction pour le sprite de la caisse.



Vous pourrez constater que les rochers et les caisses sont désormais correctement positionnés sur le sol.



Suppression (Ctrl + clic gauche)

Pour supprimer un objet placé sur votre carte, il vous suffit de maintenir enfoncé la touche **Ctrl** de votre clavier et de faire un **clic gauche** sur l'objet en question, tout étant dans le mode édition.

i Par défaut, si vous créez un objet à un emplacement où un autre objet du même type est déjà présent, le plus ancien est alors supprimé. Pour gérer les superpositions, veuillez consulter la section Map Object Type.

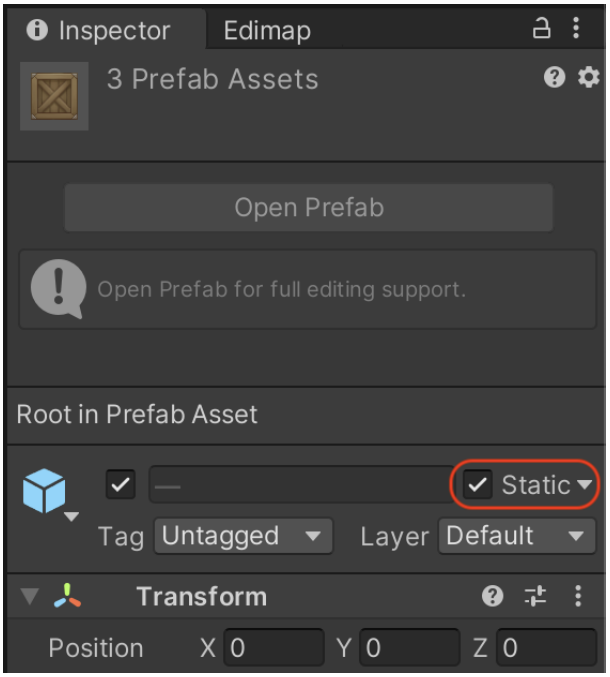
i Il vous est également possible de supprimer directement un *GameObject* enfant de l'objet "Map" depuis la *SceneView*, toutefois nous vous conseillons de bien identifier l'élément que vous souhaitez effacer.

⚠ **Si la carte a été optimisée (cf. section suivante), il est nécessaire de la "désoptimiser" avant de pouvoir supprimer une quelconque tuile qui y soit liée. Les paramètres Unoptimize On Edit et Optimize Automatically de votre grille s'avèrent très utiles pour cela.**

Optimisation

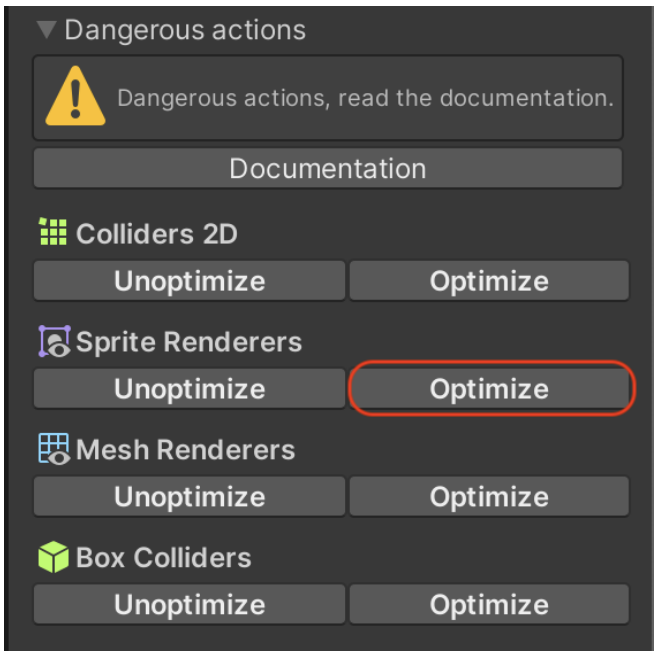
Maintenant que vous avez réussi à réaliser votre première carte avec Edimap, nous allons voir rapidement comment optimiser le rendu de cette dernière qui, en l'état, utilise beaucoup trop de *SpriteRenderer*.

On commence par activer l'option **Static** de tous nos préfabs.

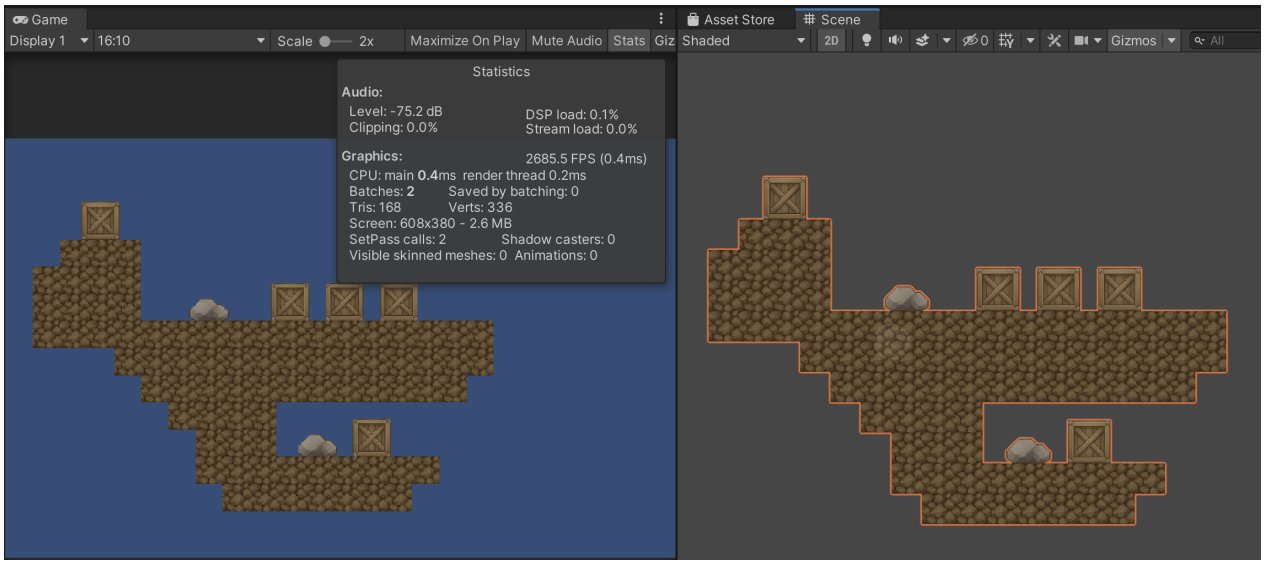


Ensuite on retourne sur notre fenêtre Edimap :

➔ sélectionnez l'onglet **Misc**,
➔ dépliez la partie **Dangerous actions**,
➔ cliquez sur le bouton **Optimize** de la section **Sprite Renderers**.



Votre carte est maintenant optimisée, vous pouvez observer dans **Stats** que désormais il n'y a plus que 2 batches et 0 dynamic batching.

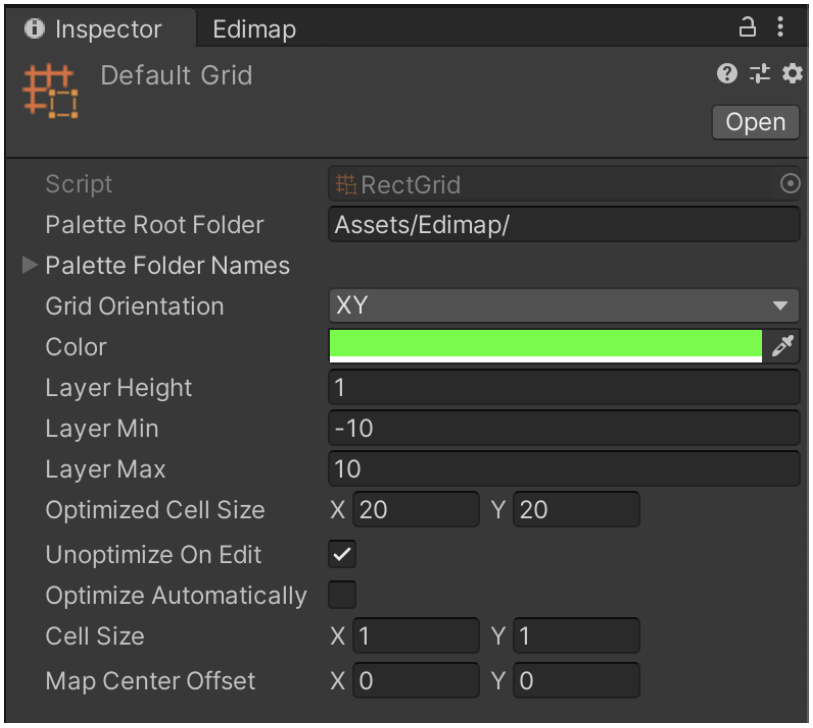


Félicitations, vous venez de créer votre première carte optimisée avec Edimap.

Néanmoins celle-ci reste très basique, la suite de la documentation vous permettra de réaliser des cartes plus complexes en vous présentant des options qui, nous l'espérons, répondront à vos attentes.

MapGrid : Configuration de la grille

Pour mieux comprendre l'édition de carte avec Edimap, nous allons analyser pas à pas le *ScriptableObject* **RectGrid**.



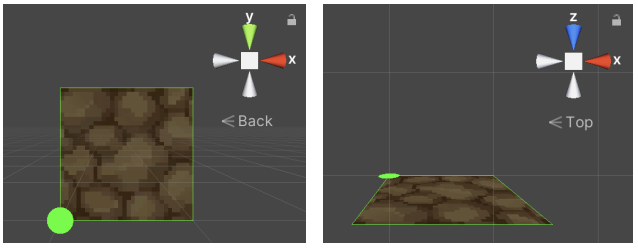
i Si vous avez l'intention d'utiliser une grille isométrique ou une grille personnalisée, nous vous conseillons vivement de suivre cette analyse dans la mesure où les bases sont identiques entre les différents types de grilles.

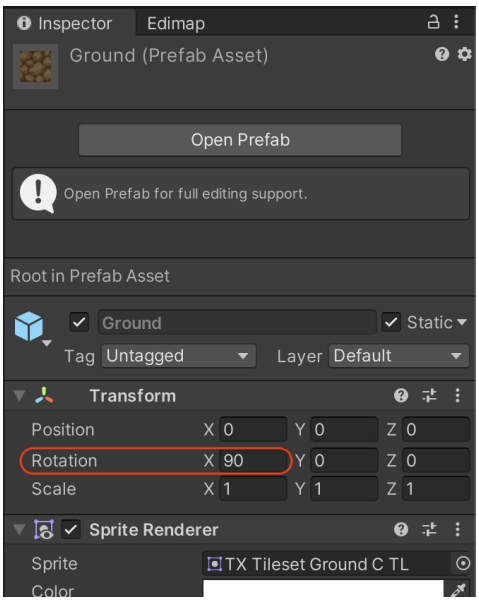
Grid Orientation

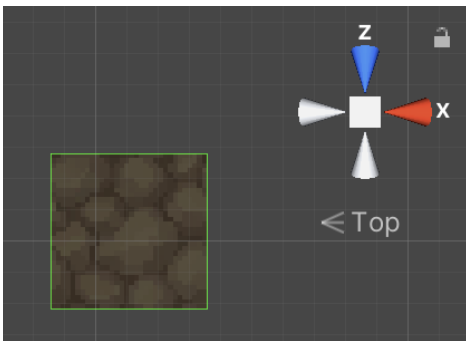
L'orientation de la grille suit un système de coordonnées. Deux systèmes de coordonnées sont disponibles :

- **XY** ➔ associé le plus souvent aux cartes 2D.
- **XZ** ➔ généralement utilisé pour les cartes 3D.

⚠ **Par défaut, Unity aligne les sprites en XY. Par conséquent, pour les sprites en orientation XZ, il est nécessaire d'ajouter une rotation de 90° en X dans la *Transform* du préfab, afin d'aligner le rendu correctement.**

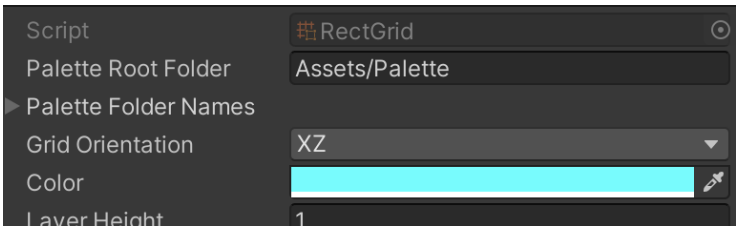


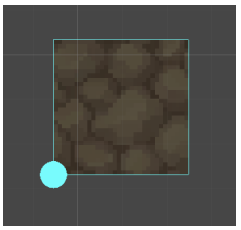




Color

Pour des raisons de visibilité, il est possible de définir une couleur personnalisée pour la grille.





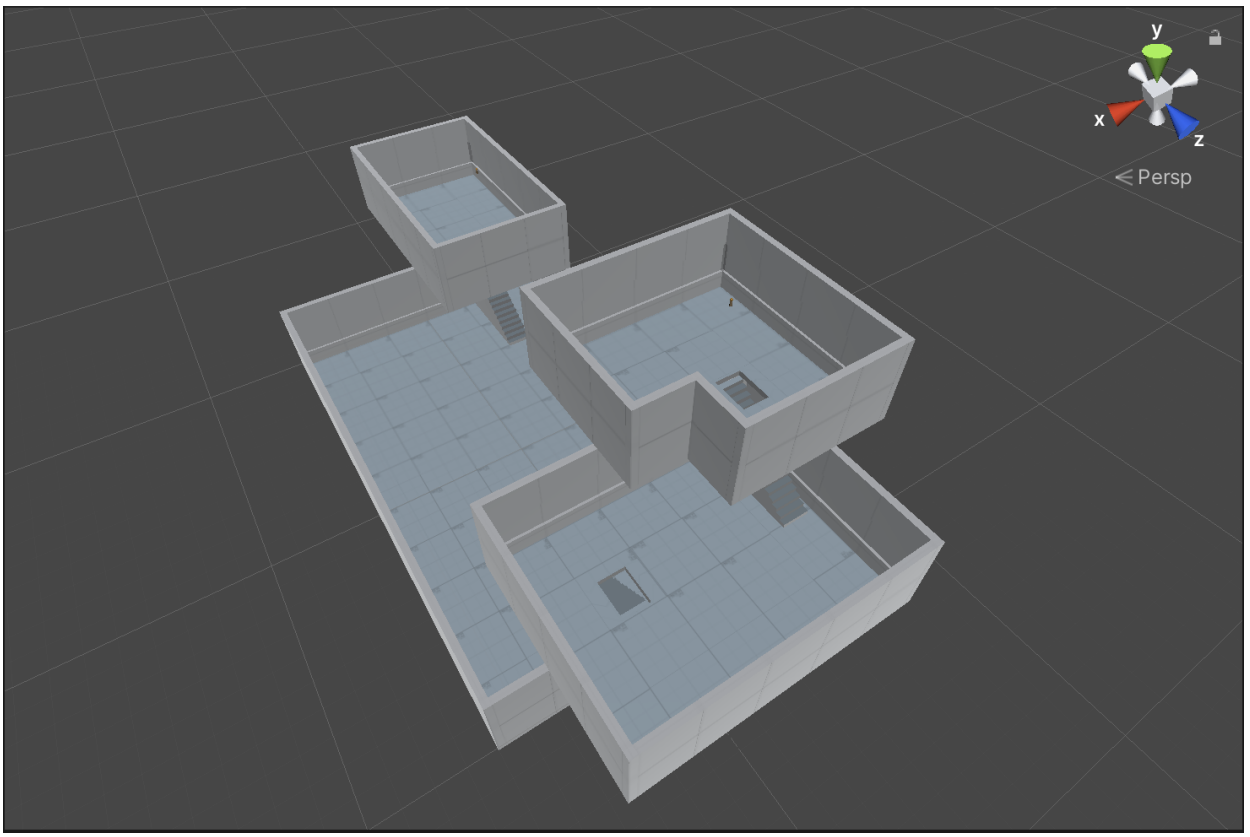
Layer Height, Layer Min & Layer Max

Lorsque vous travaillez avec Edimap, vous pouvez placer les objets sur différents niveaux de profondeurs, appelés “layers”.

Chaque layer possède sa propre profondeur dans l’espace de jeu, qui est égale à son indice multiplié par la valeur de **Layer Height**.

Dans le cas d’une grille XY, la profondeur se situe en Z. Au contraire, sur une grille en XZ, la profondeur se trouve le long de l’axe Y.

Le paramètre **Layer Height** peut être ajusté sur des cartes existantes sans aucun danger.



La profondeur servira à construire des étages à votre carte de jeu, pour des élévations de sol ou des bâtiments.

⚠ **Il est important de noter que l'élévation des décorations tels que des portraits accrochés à un mur ne doit pas être résolue par la profondeur. Il est préférable d'avoir un préfab qui inclut cette élévation directement.**

Il y a trois paramètres de configuration du "layer" au sein de la grille :

- **Layer Height** ➔ définit l'écart entre deux étages.
- **Layer Min** ➔ définit l'étage minimum.
- **Layer Max** ➔ définit l'étage maximum.

i Les champs **Layer Min** et **Layer Max** délimitent le nombre d'étages disponibles dans une carte. Ils peuvent être modifiés à loisir puisqu'ils ne sont pas destructeurs. Leur principal intérêt est d'éviter de manipuler un slider de 20 valeurs alors que la carte ne se compose que de 3 étages.

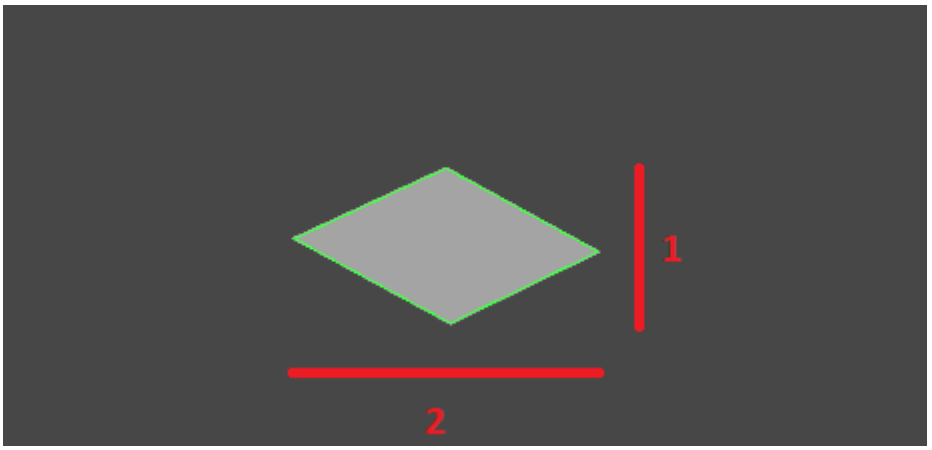
Cell Size

Une grille se compose de cases et la taille d'une case est définie par le paramètre **Cell Size**. La valeur par défaut de 1x1 signifie qu'une case mesure 1 unité par unité dans la *SceneView*.

Ce paramètre façonne la géométrie de la grille en modifiant la taille des cases.

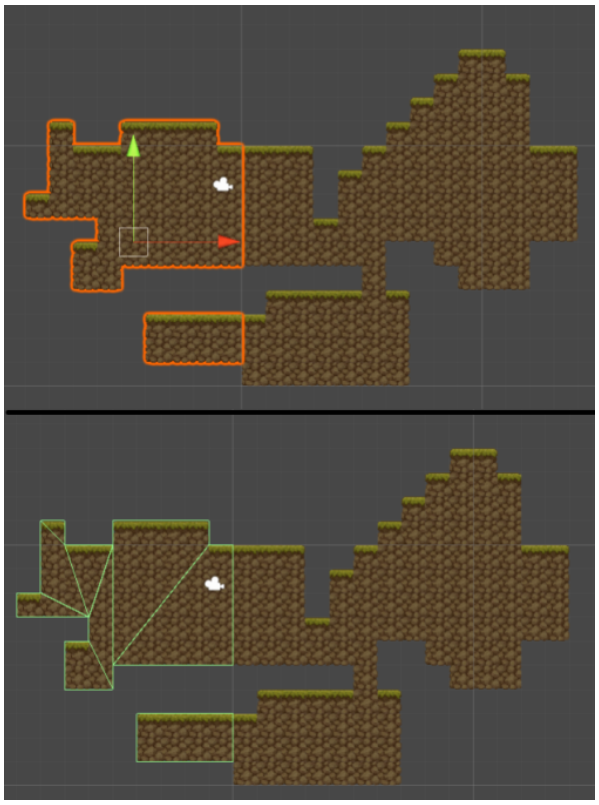
⚠ **Attention à ne pas confondre la taille des cases de la grille avec la taille d'un objet qui lui peut occuper plusieurs cases, il s'agit là de deux concepts bien distincts.**

Par exemple, une grille isométrique est une grille de losanges où chaque losange, autrement dit chaque case, a une taille X de 2 et une taille Y de 1.



Optimized Cell Size

Lorsque Edimap optimise les renderers et colliders d'une carte de jeu, il n'est pas forcément judicieux de tout fusionner. Ce paramètre sert à définir la taille d'une zone à fusionner, en terme de nombre de cases.



Dans l'exemple ci-contre, les sprites et colliders sont optimisés sur des surfaces de 10x10 cases.

i Par défaut, ce paramètre vaut 20 en X et en Y.

i Idéalement, il faut choisir une taille qui représente un quart d'un écran de jeu.

Unoptimize On Edit

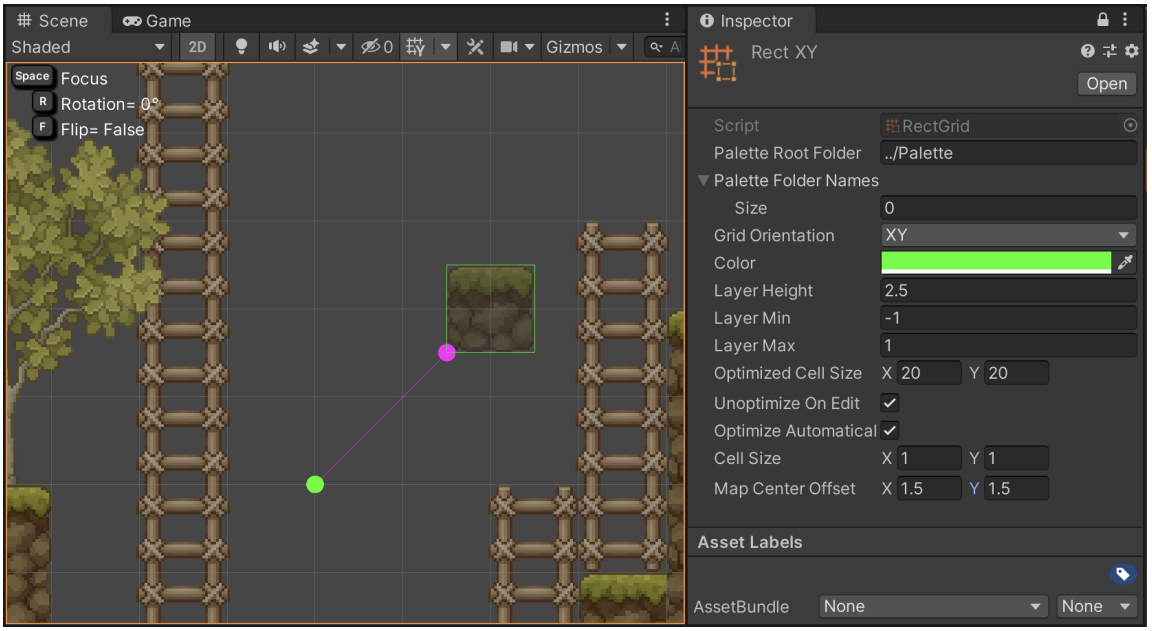
Cette option permet de retirer les optimisations des renderers et des colliders automatiquement lorsque le mode édition est activé.

Optimize Automatically

Optimise automatiquement les renderers et les colliders lorsque le mode édition est désactivé.

Map Center Offset

Ce paramètre sert à décaler arbitrairement l'origine de la carte.



i En mode édition, un point de la couleur de la grille représente l'origine de la carte par rapport au layer sélectionné. Le point d'origine décalé est représenté par un point de la couleur inverse de la grille, et une ligne est tracée entre les deux points pour prendre pleinement conscience de l'écart .

i Par défaut, la coordonnée (0, 0) est le coin bas gauche de la case d'indice (0, 0).

⚠ **Ce paramètre ne permet pas de corriger l'origine d'une carte déjà existante, il faut donc penser à bien choisir la valeur avant de commencer à éditer votre carte. Une solution pour remédier à ce problème serait de rédiger un script permettant de décaler tous les préfabs de la carte.**

Configuration des préfabs

Jusqu'ici, nous avons utilisé seulement des sprites avec la configuration par défaut.

Mais qu'en est-il si l'on souhaite avoir un objet qui ne se place pas au milieu de la case ? Et si nous souhaitons mettre plusieurs objets sur la même case ? Ou encore si nous souhaitons placer une grande maison qui représente un seul bloc ?

Aucun souci, tout est faisable. Nous allons voir en détail comment réaliser tout cela.

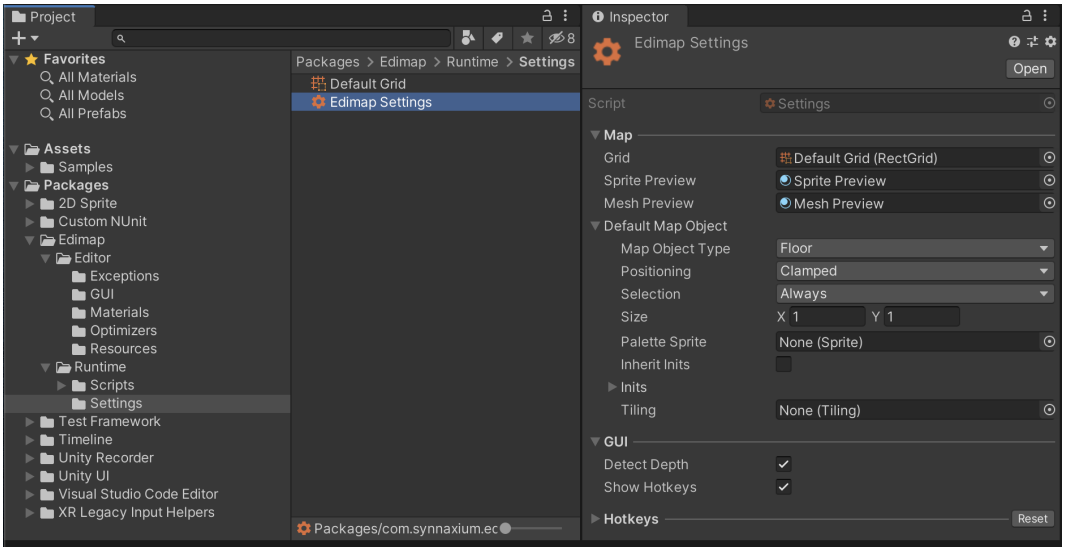
Niveaux de configuration

Edimap vous propose de configurer vos objets dans une granularité macro, puis en ajoutant des exceptions de plus en plus fines.

Dans le cas où tous vos objets ont le même comportement, vous pouvez tout configurer en une seule fois. Si au contraire vos éléments sont complexes et hétérogènes, vous pourrez configurer Edimap avec autant de finesse que nécessaire.

Configuration globale

Edimap possède une configuration globale qui se trouve au sein du package.



i Il est possible de créer une nouvelle instance de cette configuration dans le dossier *Assets*. Cette instance sera prioritaire sur la configuration du dossier package.

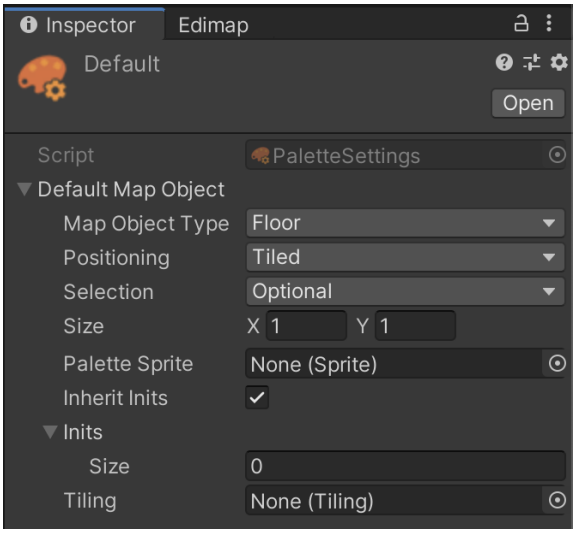
Au sein de cette configuration, vous trouverez une partie **Default Map Object** qui est la configuration par défaut utilisée à la création d'un préfab au sein d'une palette.

Cette configuration indique les valeurs par défaut, elle est donc la moins prioritaire, mais reste toujours présente.

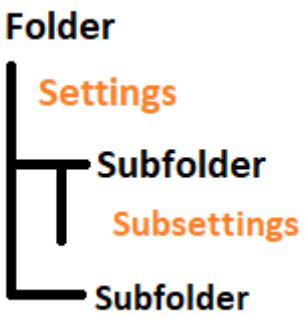
Configuration à la palette

Il est possible de surcharger la configuration globale pour un dossier et ses sous-dossiers. Pour cela, il suffit d'ajouter un *ScriptableObject* **Palette Settings** au sein de celui-ci.

Clic droit dans le dossier ➜ Create ➜ Synnaxium Studio ➜ Edimap ➜ Palette Settings



Il est également possible de surcharger la configuration d'une palette au sein de l'un de ses sous-dossiers. Pour cela, il suffit d'ajouter un nouveau **Palette Settings** au sein du sous-dossier concerné.



Vous pouvez donc avoir un dossier "Props", et un sous-dossier "Bâtiments" qui contiendra une configuration pour les larges objets.

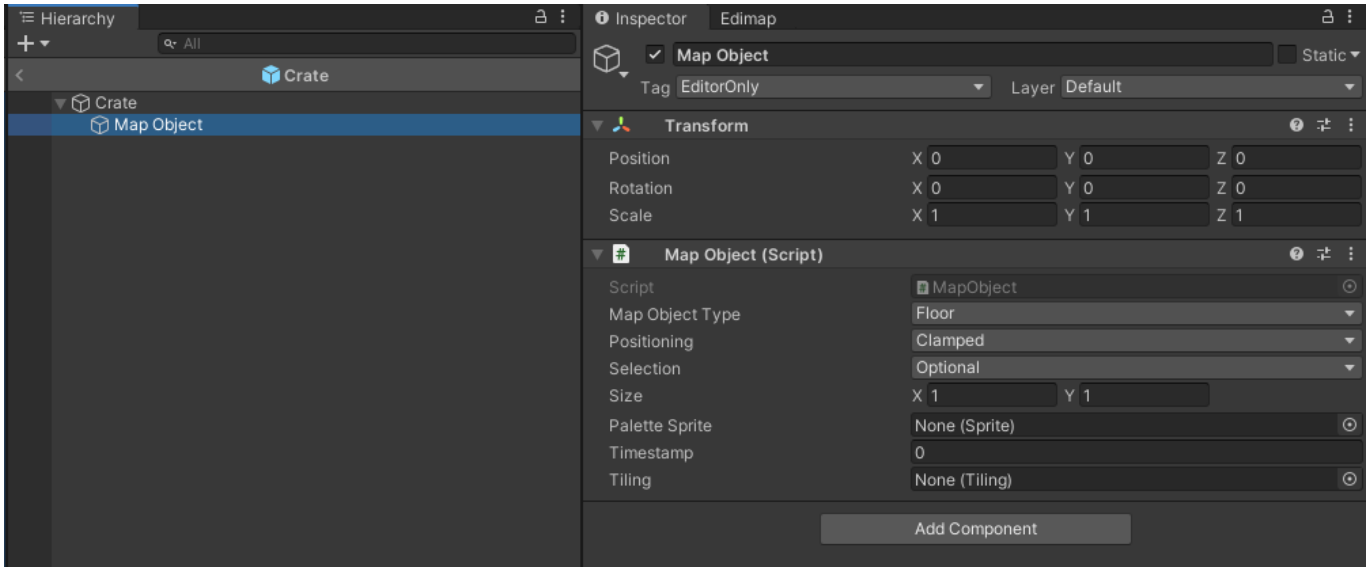
Cette notion de configuration au dossier, et héritable des dossiers parents, vous permet d'intégrer 90% de vos objets instantanément en vous contentant de les mettre dans les bons dossiers.

Configuration par objet

Parfois certains objets sont trop spécifiques, et cela peut devenir rapidement pénible de devoir créer une armée de sous-dossiers contenant chacun un seul préfab.
C'est souvent le cas pour certains concepts très orientés gameplay tels que les checkpoints et les spawns.

Heureusement, il est possible de surcharger la configuration directement au sein du préfab :

➔ créer un *GameObject* enfant au préfab,
➔ renseignez lui le tag *EditorOnly*,
➔ ajoutez lui un composant **MapObject**.

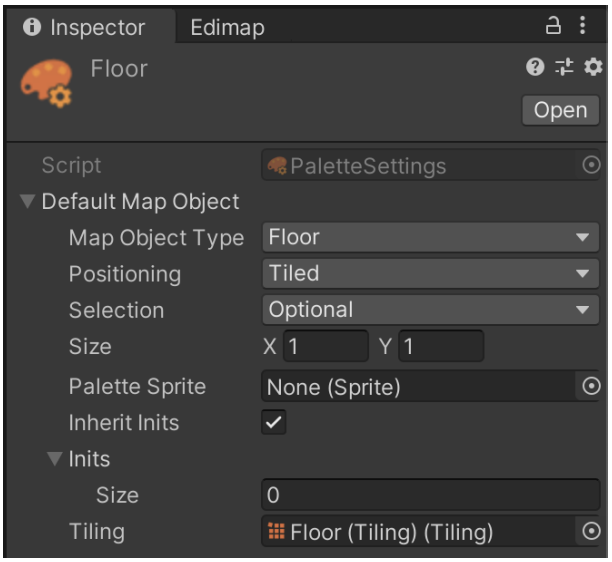


i Le tag *EditorOnly* nous assure que cette approche n'aura pas d'impact au sein de la version exécutable.

⚠ **Il y a toutefois une limitation à cette approche : si la carte de jeu est instanciée via un préfab au runtime, alors le tag *EditorOnly* ne fonctionnera pas et l'objet sera présent dans le build. Ce cas de figure se produira si vous travaillez sur un jeu contenant une génération procédurale pour un donjon.**

⚠ **La configuration par le composant MapObject est toujours prioritaire.**

Default Map Object

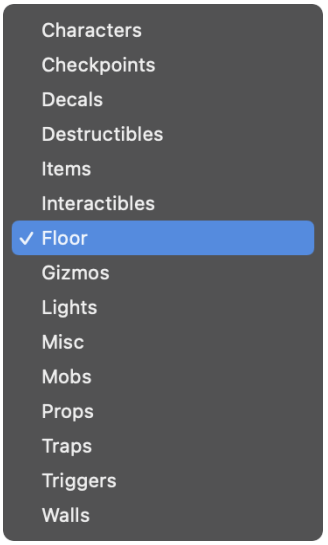


Map Object Type

Le type d'objet est défini par une énumération nommée **MapObjectType**.

Ces types sont utilisés essentiellement pour résoudre les superpositions d'objets. Si vous posez un caillou ou un arbre sur le sol, vous n'avez pas envie que le sol soit supprimé.

Le type est aussi utilisé lors de la suppression manuelle (ctrl + clic gauche), Edimap retire un objet du même type que l'objet actuellement sélectionné dans la palette.



i Il est fortement conseillé d'utiliser le moins de types possibles et un seul type par dossier au sein de la palette pour vous faciliter la vie lors de l'édition d'une carte.

Size

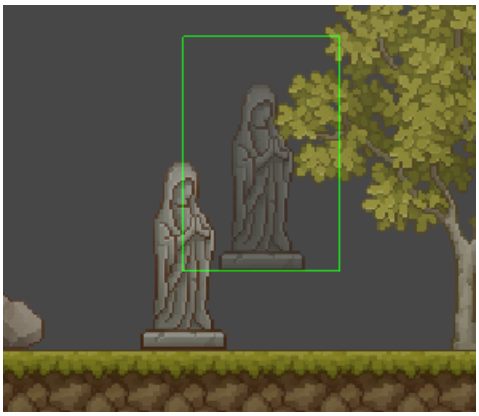
Ce paramètre définit le nombre de cases occupées par l'objet.

La taille de l'objet permet de gérer les superpositions et le tiling pour les sols composés de larges tuiles. Cette taille est également utilisée lorsque vous placez plusieurs préfabs à la fois.

Une armée de statues posées en une seule fois, sans superposition.

i Pour des raisons de performances, l'objet est en réalité ancrée sur sa case la plus en bas à gauche.

Dans ce cas, la nouvelle statue va remplacer l'ancienne car le coin bas-gauche de l'ancienne est compris dans la tuile de la nouvelle.



Dans c'est autre cas, la nouvelle statue va se superposer à l'ancienne puisqu'elle ne touche pas le coin bas-gauche de l'ancienne.

i En pratique, cette subtilité ne devrait pas vous déranger. Si malgré tout ce comportement pose de réels problèmes au sein de votre production, n'hésitez pas à nous contacter : la roadmap de Edimap reste active !

Positioning

Edimap gère trois types de positionnement :

- **Normal** ➔ crée l'objet à l'emplacement du curseur.
- **Clamped** ➔ positionne l'objet au milieu de la case.
- **Tiled** ➔ permet de positionner l'objet au centre d'une case tout en évitant les superpositions.

Dans le cas d'objet de taille 1x1, les types **Clamped** et **Tiled** auront le même comportement, ce qui ne sera pas le cas pour des objets plus grands.

En type **Clamped** pour des objets de 2x2, Edimap permet la superposition.

En type **Tiled** pour des objets de 2x2, Edimap ne permet pas la superposition.

Pour le type **Normal**, Edimap considère que vos préfabs ont la même topologie que des décals : la superposition est donc autorisée.

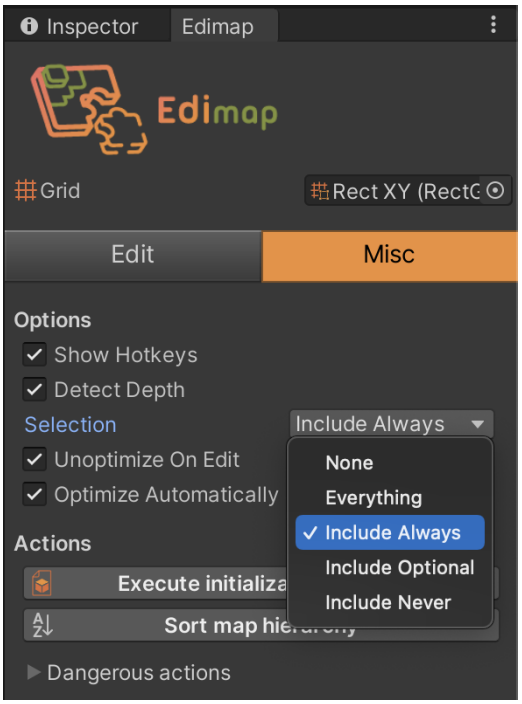
L'exemple ci-contre, montre deux rochers qui sont sur la même case et qui se superposent.

Selecting

Au cours de vos tests avec Edimap, vous avez potentiellement remarqué que vous ne pouviez pas sélectionner tous les objets au sein de la scène.

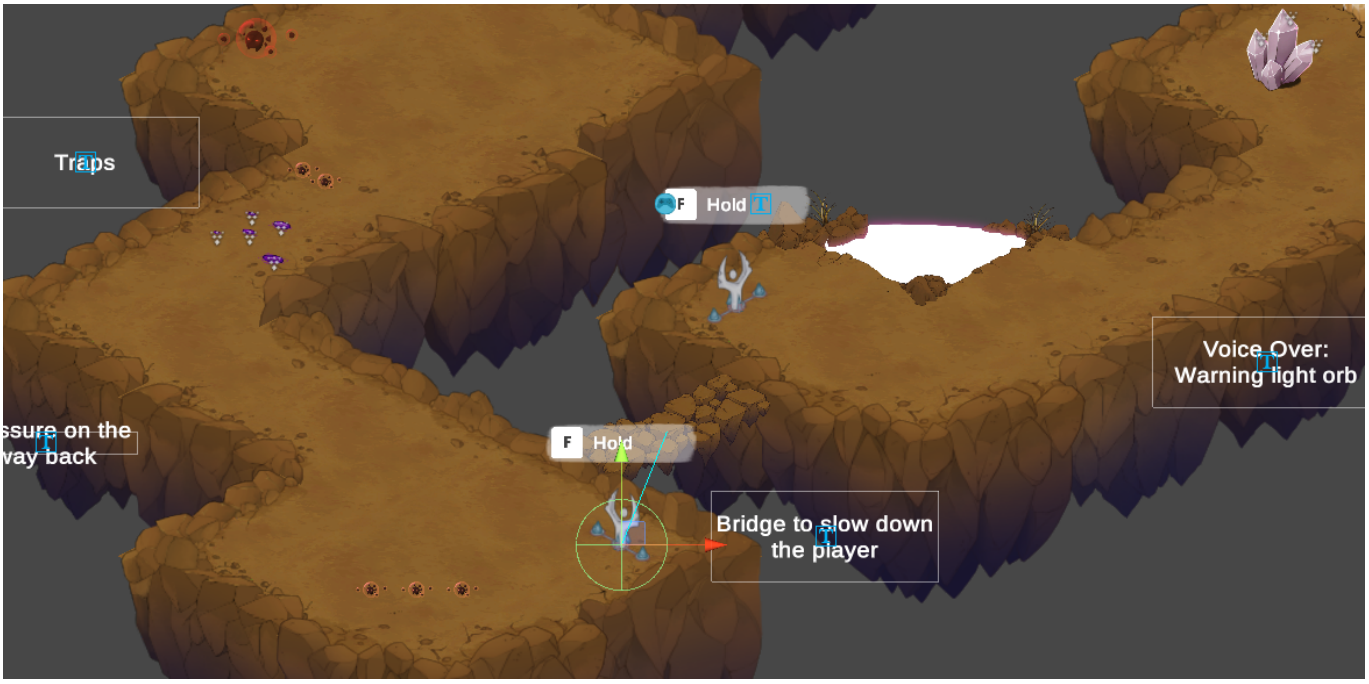
Rien ne se passe si l'on essaie de sélectionner une boîte

Edimap propose une méthode de travail par sélection. Au sein de l'onglet **Misc**, vous pouvez remarquer une option **Selection** avec plusieurs options possibles.



A côté de cela, chaque objet possède un type de sélection parmis trois possibilité :

- **Always** ➔ pour les objets orientés gameplay souvent sélectionnés par le level designer.
- **Optional** ➔ pour les éléments configurables très peu manipulés.
- **Never** ➔ pour les objets purement décoratifs faisant partie du décor.



Lorsque vous créez un niveau jouable, vous allez rapidement vous retrouver avec une hiérarchie encombrée et une scène où la sélection est imprécise.

En prenant soin de bien choisir le type de sélection pour vos préfabs, Edimap s'assure que l'étape de level design soit un véritable plaisir.

Palette Sprite

Edimap utilise *AssetPreview* pour avoir une prévisualisation de votre préfab. Parfois, l'image générée par celui-ci ne fait pas forcément sens pour certains concepts abstraits tels que les checkpoints.

Ce paramètre sert à ajouter une image pour la visualisation du préfab au sein de la palette de Edimap.

Tiling

Introduction

Lorsque l'on travaille avec certains types de tuile, comme des murs, des échelles, des clôtures ou encore du terrain, nous avons besoin de tiling pour créer un schéma répété.

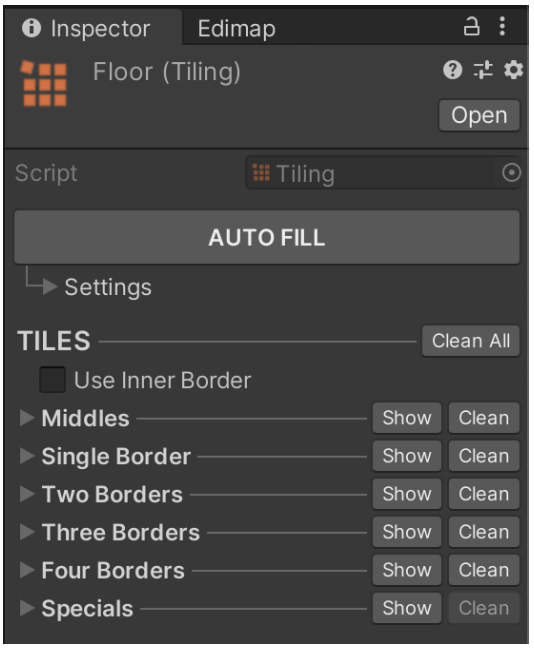
Exemples de tiling pour un platformer 2D.

Pour configurer le tiling, nous utilisons un *ScriptableObject* **Tiling**.

Clic droit ➜ Create ➜ Synnaxium Studio ➜ Edimap ➜ Tiling

Comme vous pouvez le remarquer les tuiles sont séparées en plusieurs groupes en fonction du nombre de bordures qu'elles contiennent. Il y a également un groupe **Specials** mais nous y reviendrons plus tard.

Un gros bouton **AUTO FILL** est également disponible. Il va nous permettre de remplir automatiquement le tiling avec les bonnes tuiles en fonction d'une certaine codification rendue possible par l'utilisation de préfixes numériques.



Préfixe

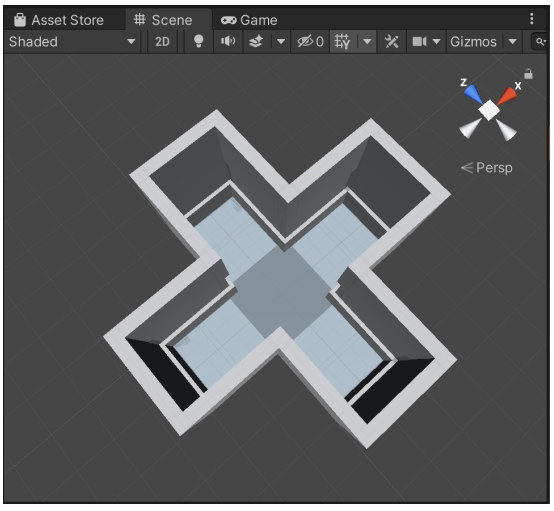
Afin de pouvoir utiliser la fonctionnalité **AUTO FILL**, il est nécessaire de respecter une normalisation s'appuyant sur le pavé numérique d'un clavier.

Cette normalisation nous semble être la plus logique et la plus simple.

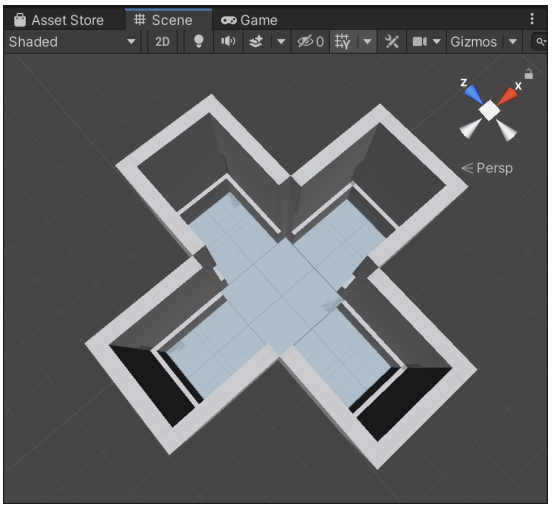
1	coin bas gauche
2	bordure bas
3	coin bas droite
4	bordure gauche
5	ni bordure, ni coin
6	bordure droite
7	coin haut gauche
8	bordure haut
9	coin haut droite

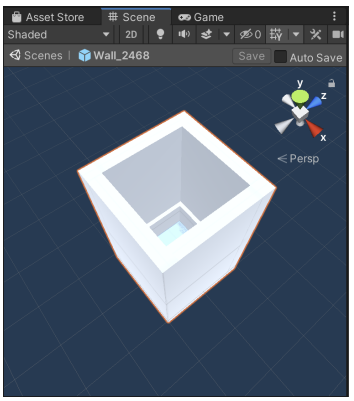
Les coins sont utilisés uniquement si l'option **Use Inner Border** est activée. Cette dernière est le plus souvent utilisée en 3D, notamment lorsqu'une bordure est représentée par un mur.

***Use Inner Border** activé.*

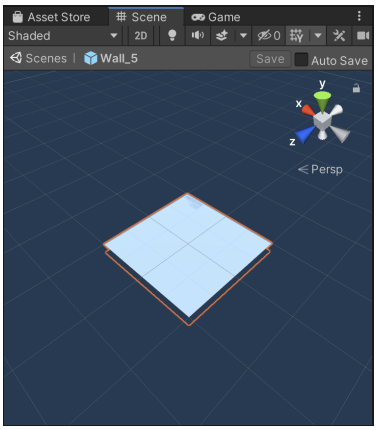


***Use Inner Border** désactivé.*





Un préfix commence toujours par « _ » et peut se composer de plusieurs chiffres, ainsi une tuile avec un nom contenant le préfix « _2468 » sera considérée comme une tuile avec 4 bordures.

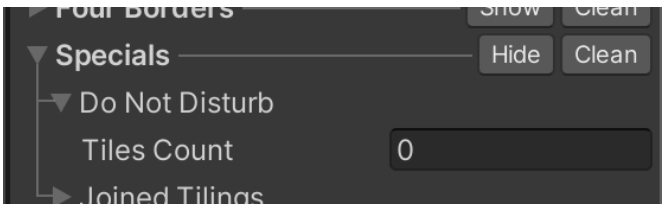


A contrario, une tuile ayant le préfix « _5 » dans son nom est considérée comme une tuile sans aucune bordure, autrement dit la tuile qui sera au centre de 4 autres tuiles.

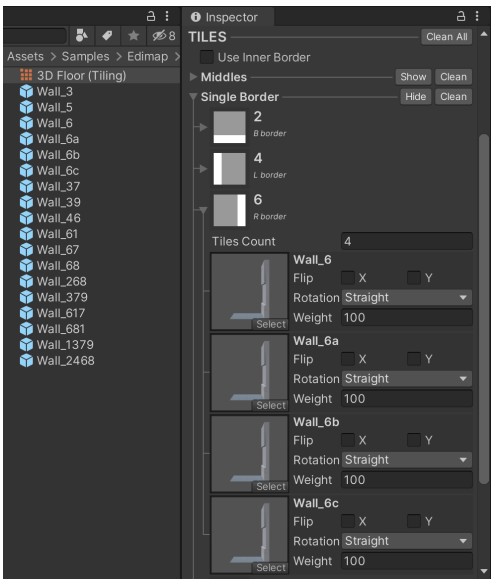
Les chiffres représentant les préfixes, ainsi qu'un schéma et un mini descriptif, sont visibles dans chaque groupe.

Parmi ces groupes il y en a un intitulé **Specials** dans lequel se trouve le sous-groupe **Do Not Disturb**. Ce dernier doit être rempli directement à la main par l'utilisateur.

Les tuiles renseignées dans ce tableau seront considérées comme voisines du tiling et ne seront pas modifiées par celui-ci. Cela peut s'avérer très utile pour ajouter des exceptions au tiling, notamment avec une porte par exemple.



Comme vous l'avez peut être déjà remarqué, il est possible d'avoir plusieurs tuiles du même type, autrement dit plusieurs tuiles avec une bordure droite par exemple (avec le préfixe « _6 »). Toutefois si toutes ces tuiles ont exactement le même préfixe, une seule sera prise en compte par **AUTO FILL.**

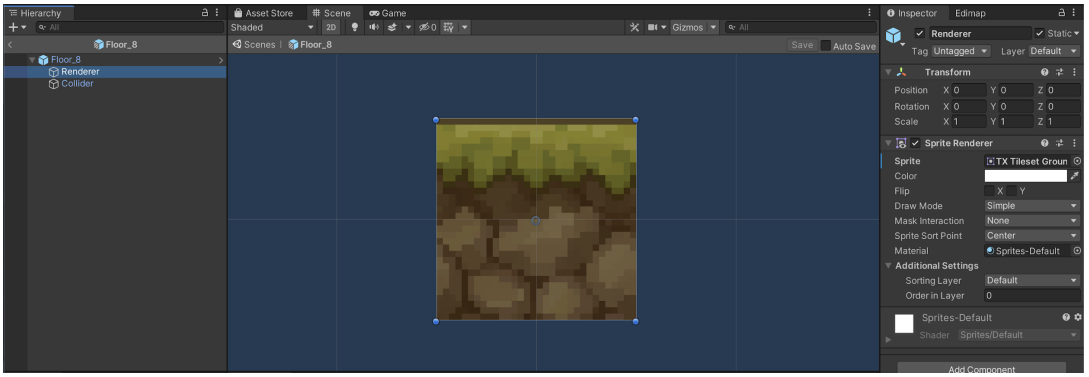


Afin que chaque tuile soit prise en compte il vous suffira d'ajouter une lettre en minuscule, ce qui donnera des noms de tuiles de ce genre :

➔ *NomTuile_**6***
➔ *NomTuile_**6a***
➔ *NomTuile_**6b***
➔ *NomTuile_**6c***
➔ *etc.*

Les tuiles (tiles)

Une tuile n'est ni plus ni moins qu'une simple référence à un préfab.



En plus de la référence au préfab, les tuiles utilisées par le tiling ont plusieurs paramètres plus ou moins importants.

Les paramètres **Prefab**, **Flip X**, **Flip Y** et **Rotation** sont généralement compléter automatiquement par **AUTO FILL**.

Prefab

Référence au préfab utilisé par la tuile. Le nom du préfab et une prévisualisation sont visibles lorsque ce paramètre est renseigné.

i Il se peut que le rendu de la prévisualisation ne soit pas pertinent du fait de l'utilisation de *AssetPreview*.

Flip

Ce paramètre indique si la tuile est inversée sur l'axe horizontal (X) ou vertical (Y).

Rotation

Rotation de la tuile par rapport au point de pivot de son renderer.

Weight (Pondération des éléments)

Lorsque vous avez plusieurs tuiles possibles pour une même configuration, Edimap choisira aléatoirement une des tuiles possibles, en pondérant les chances par la valeur de ce champ. Si une tuile a une valeur de 200 et une autre tuile une valeur de 100, elles auront respectivement 66% et 33% de chances d'être sélectionnées.

i La valeur par défaut est 100 pour toutes les tuiles.

Créer un tiling (Utilisation de **AUTO FILL**)

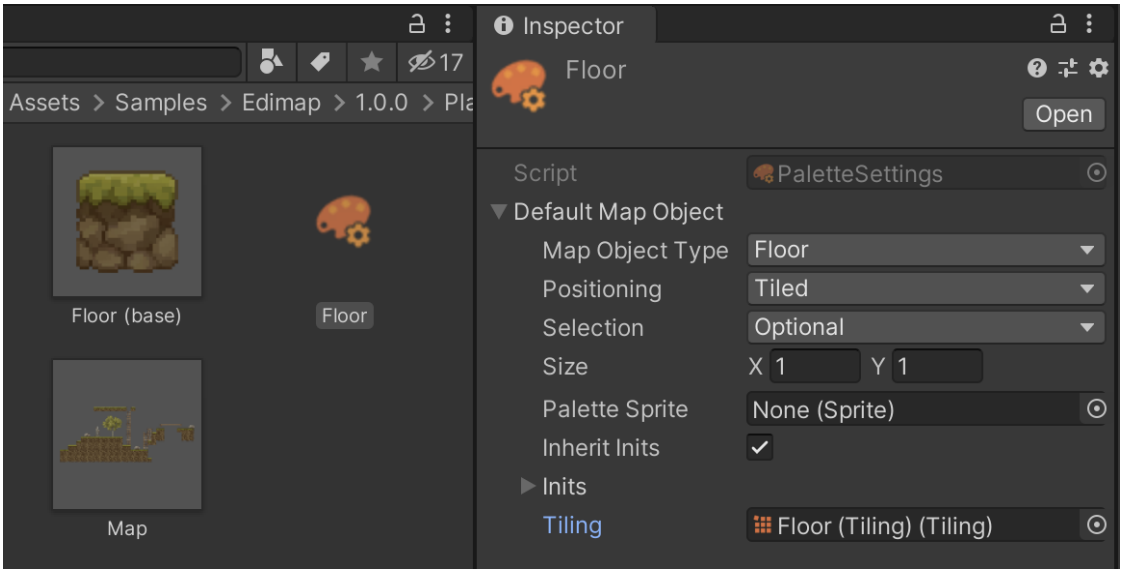
Pour créer un tiling, la méthode la plus simple consiste à préparer un répertoire de préfab contenant toutes les variations.

Chaque variation doit être normalisée correctement à l'aide d'un préfixe comme indiqué dans la partie Préfixe.

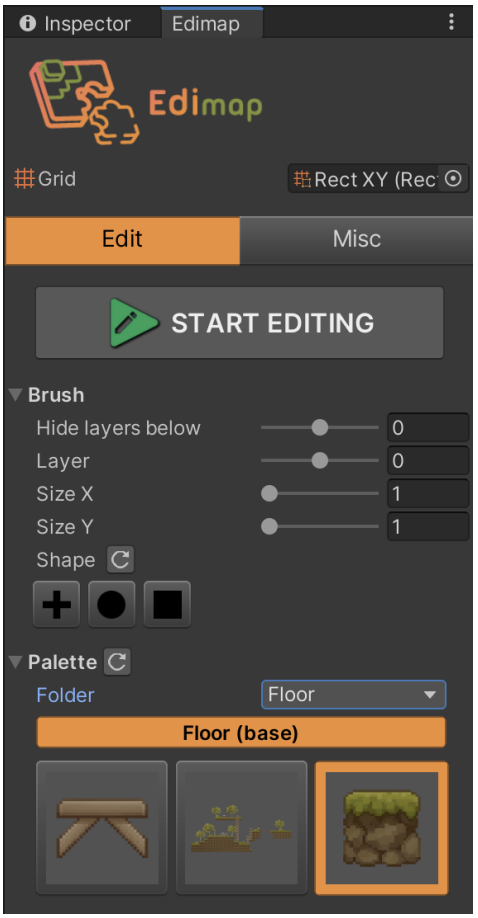
Après avoir créé et bien nommé vos préfabs, ajoutez un *ScriptableObject* **Tiling** au sein du dossier, et cliquez sur le bouton **AUTO FILL**. Il vous sera demandé de sélectionner le répertoire dans lequel se trouvent vos préfabs (le chemin par défaut est celui du **Tiling**). Cette manipulation permet de pré-remplir automatiquement votre **Tiling**.

Pour que le tiling soit opérationnel il doit être référencé sur une palette :

➔ Créez un nouveau répertoire dans le dossier de la palette de votre choix.
➔ Ajoutez y un préfab qui sera utilisé seulement pour être affiché dans la palette.
➔ Ajoutez également un *ScriptableObject* **Palette Settings**.
➔ Sélectionnez votre tiling dans le champ **Tiling** de ce *ScriptableObject*.

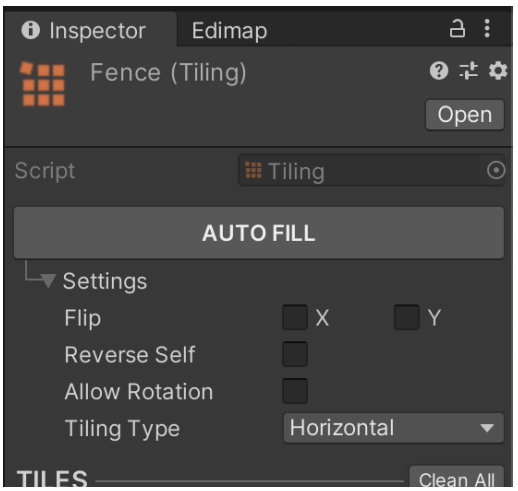


Si vous avez procédé correctement vous verrez alors votre préfab dans la palette correspondante. Il ne vous reste plus qu'à tester votre tiling afin de vérifier si tout a bien été configuré.



i Deux types de tiling de même **Map Object Type** peuvent être placés côte à côte au sein de la carte.

Configuration de **AUTO FILL**



Vous remarquerez qu'il y a un menu déroulant nommé **Settings** sous le bouton **AUTO FILL** du *ScriptableObject* **Tiling**. Il s'agit des paramètres utilisés pour la fonctionnalité d'auto-remplissage.

i Par défaut, le menu déroulant **Settings** est déployé. Il suffit de cliquer sur **Settings** pour cacher ou afficher son contenu.

Flip X et Y

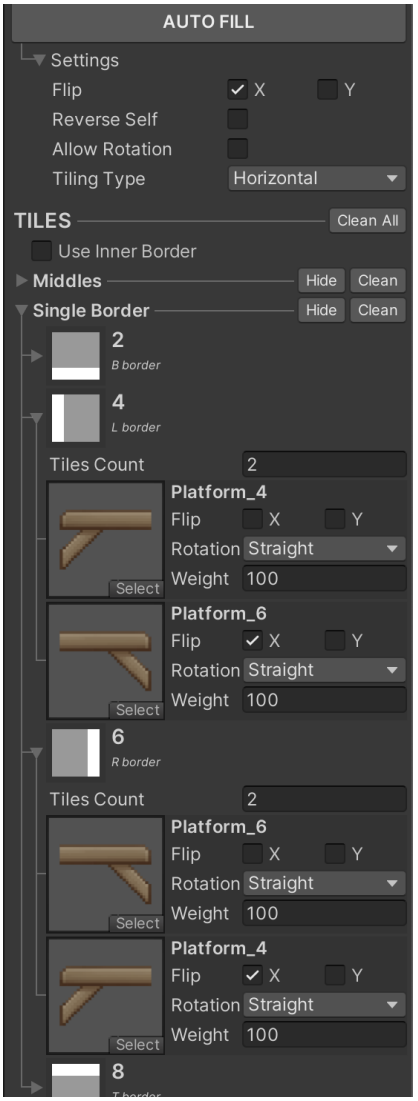
Lorsque vous réalisez un tiling, il est possible que votre préfab de bord gauche fonctionne également pour le bord droit, à une inversion près.

Si la case **Flip X** est cochée, alors **AUTO FILL** inclut les inversions horizontales pour remplir le tiling.

Ainsi dans l'exemple ci-contre on remarque que la tuile avec le préfixe _4 apparaît dans la catégorie de tuile 6, et son paramètre **Flip X** est activé. On peut également observer l'inverse, la tuile _6 dans la catégorie 4.

Le fonctionnement est le même pour le paramètre **Flip Y** mais les inversions sont verticales.

i Le véritable intérêt de cette option est de n'avoir qu'un seul préfab pour compléter deux catégories de tuile dans le tiling.



Reverse Self

Ce paramètre est activé lorsqu'un même préfab peut être présent plusieurs fois pour la même configuration en s'inversant.

i Il est utilisé principalement pour les préfabs "corridor", c'est-à-dire pour les préfabs avec seulement deux bordures.

Allow Rotation

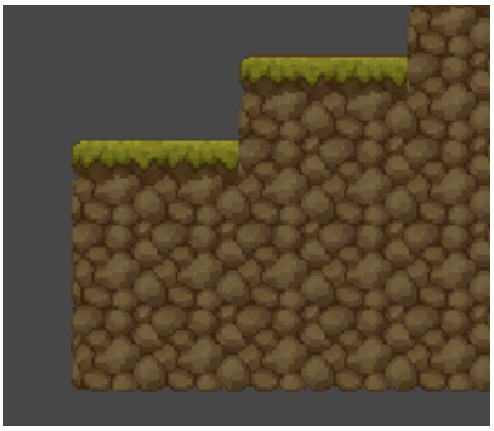
Cela fonctionne comme le **Flip** mais avec des rotations. De ce fait, un seul préfab de tuile peut compléter les catégories 2, 4, 6 et 8. Le même préfab sera dans chaque catégorie mais avec une **Rotation** différente.

Tiling Type

Un type de tiling est nécessaire à **AUTO FILL** pour que celui-ci puisse déterminer quelles tuiles ajouter automatiquement. Il y a 3 types de tiling :

Full

Il est utilisé lorsque votre schéma se répète à la fois verticalement et horizontalement, ce qui souvent le cas pour le sol.

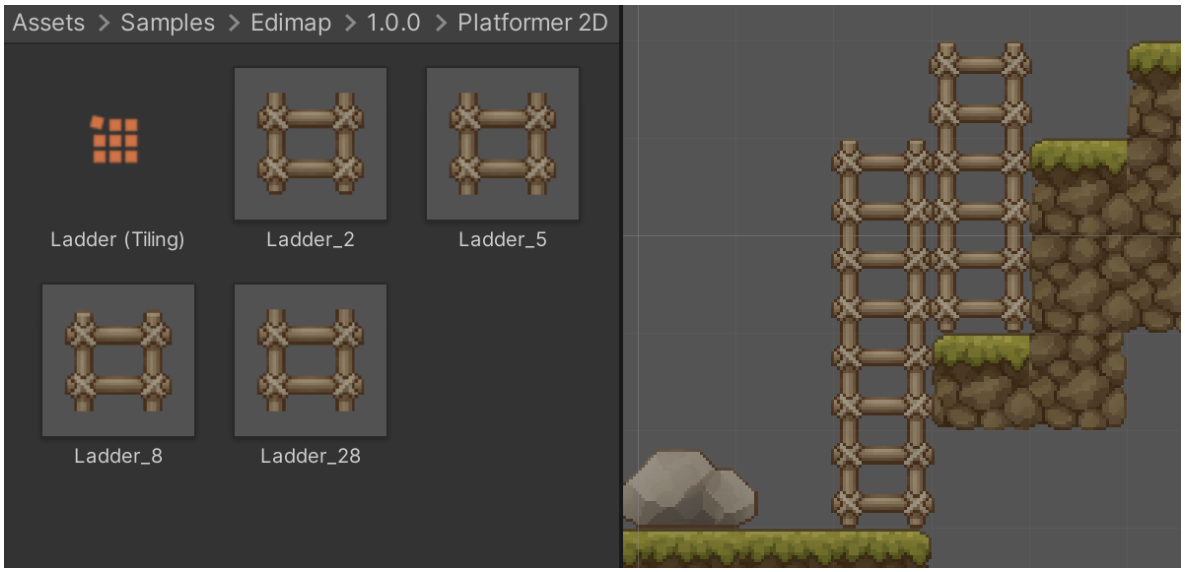


Horizontal

Ce type est utilisé pour les répétitions horizontales, dans notre exemple 2D nous l'utilisons pour les plateformes.

Vertical

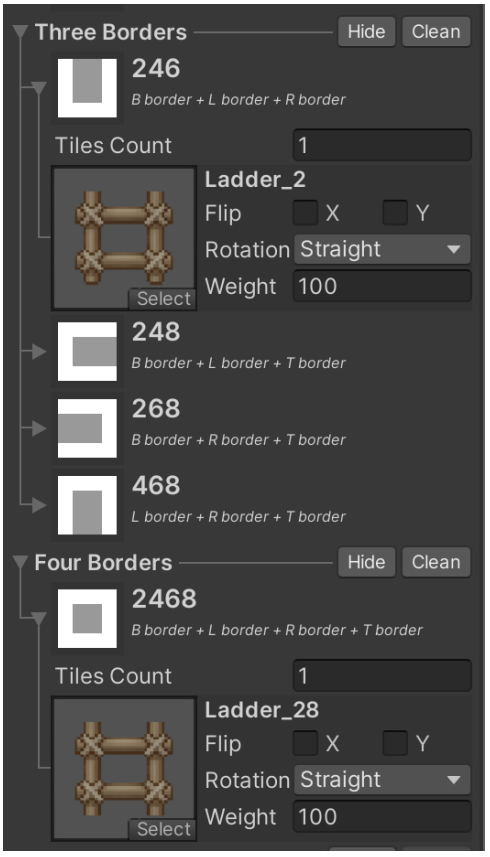
Il est associé aux répétitions verticales, notamment des échelles ou des cordes.



Comme précisé précédemment, **AUTO FILL** va venir compléter les tuiles manquantes de certaines catégories, encore faut-il que les préfabs utilisés soient corrects.

À titre d'exemple, dans notre cas nous n'avons besoin que d'un bord bas (**2**), un bord haut (**8**), une tuile pour le milieu (**5**) et optionnellement une tuile pour les échelles d'une seule case de haut (**28**).

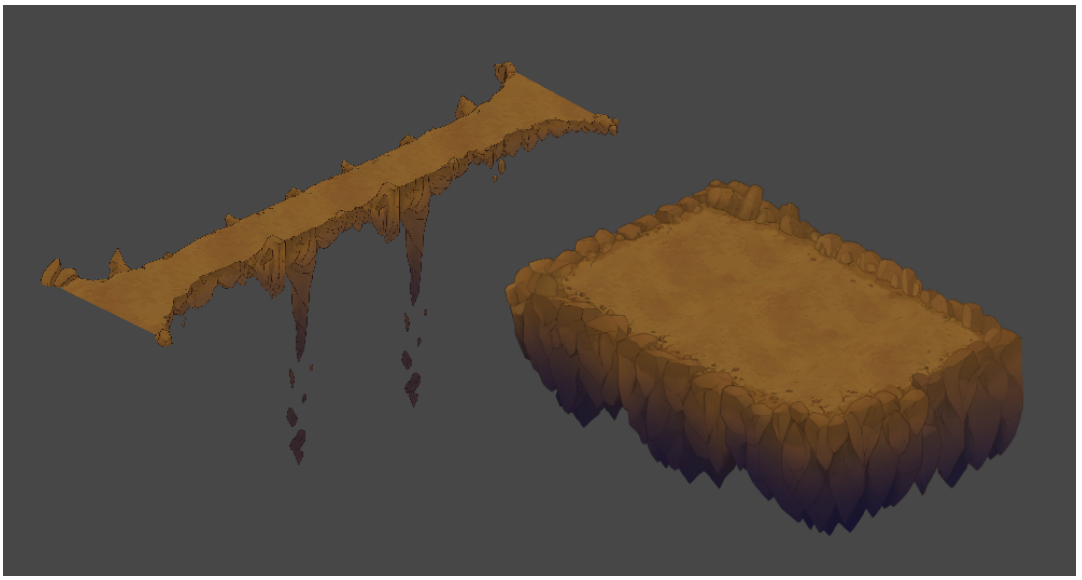
De cette manière les configurations manquantes seront complétées par **AUTO FILL**, ce qui permet d'avoir deux échelles côte à côte sans que cela ne brise le schéma.



Joined tilings

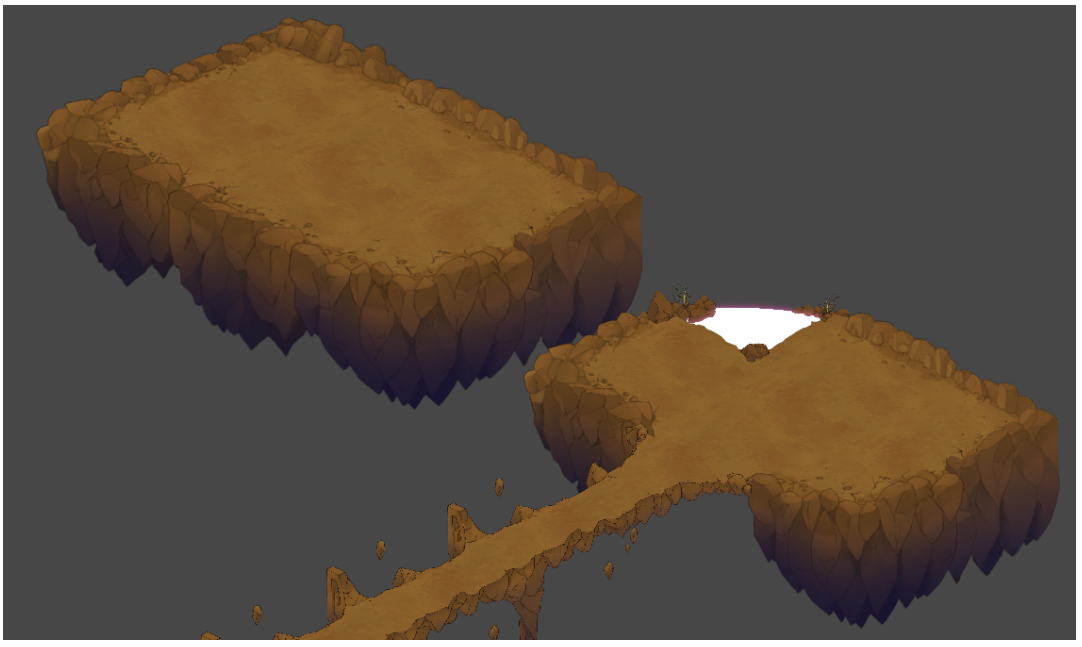
Par défaut, les tilings au sein de Edimap n'interagissent pas entre eux. Autrement dit, si deux types de tilings sont côte à côte, chacun fermera ses bords respectifs.

Dans certains cas de figure, ce comportement est inapproprié. Considérons cet exemple avec deux tilings :



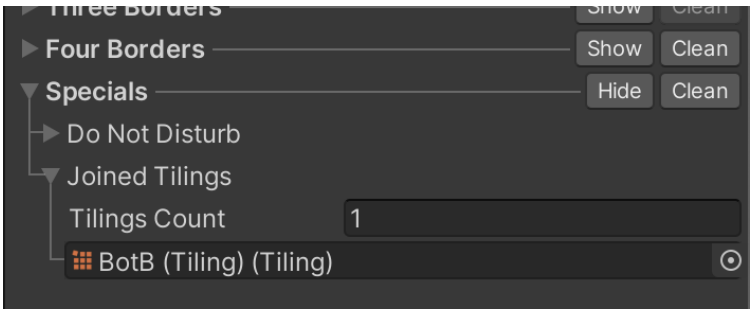
Tiling de pont horizontal et tiling de sol complet.

Le pont a été conçu pour venir étendre naturellement la zone de sol, sans coupure :



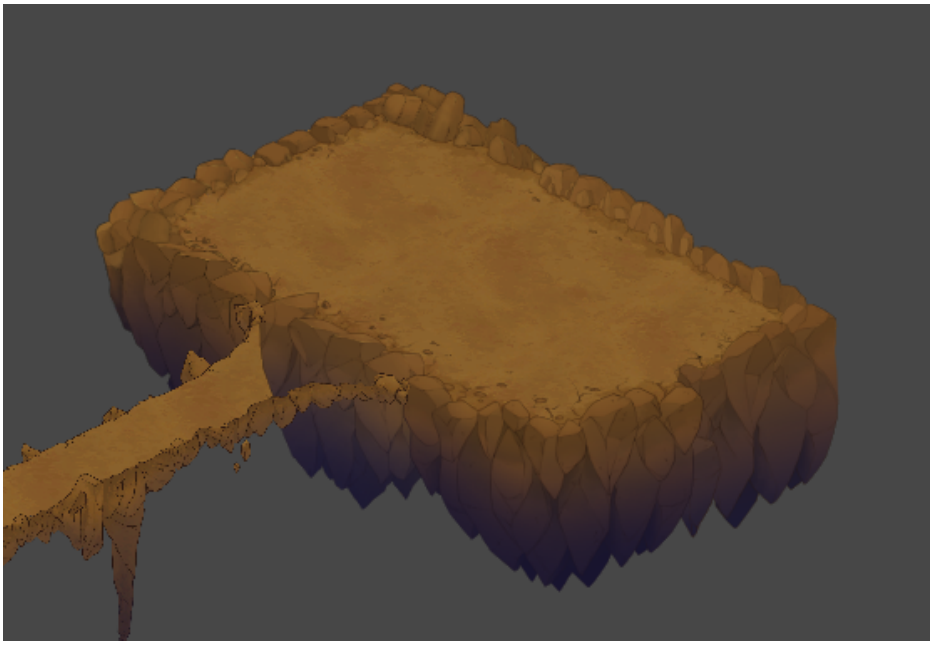
Le pont et le sol s'emboîtent pour créer un ensemble homogène.

Dans ce cas de figure, le tiling du sol contient une “jointure” vers le tiling du pont, ainsi le tiling du sol n'assigne pas de tuile avec un bord lorsque cette dernière est en contact avec une tuile du tiling du pont. Le pont quant à lui est parfaitement indépendant et place son bord malgré la présence du sol à côté.



Le tiling du sol a une référence vers le tiling du pont dans le champ **Joined Tilings**.

Sans cette option paramétrée, le rendu aurait été le suivant :



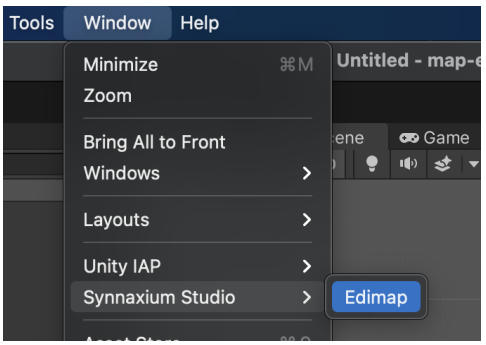
Sans jointure, le tiling du sol va créer un bord qui n’est ici pas souhaité

i La jointure est un outil très puissant qui vous permettra de créer des murs intérieurs, des routes, des ponts, et tout un tas de schémas répétés complexes.

⚠ **Si un tiling A comprend une référence au tiling B :**

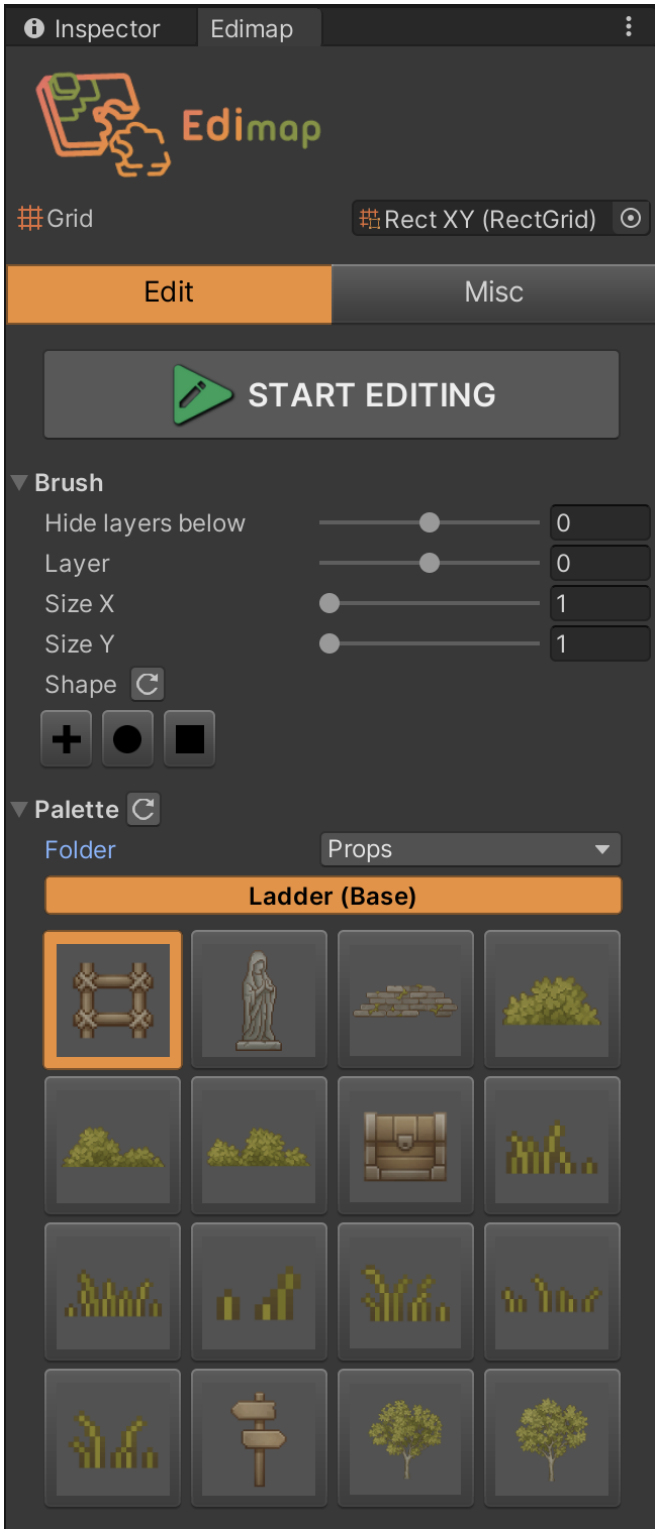
➔ la tuile du tiling A n’aura pas de bord,
➔ la tuile du tiling B aura un bord (si le tiling B n’a pas de référence au tiling A).

La fenêtre Edimap



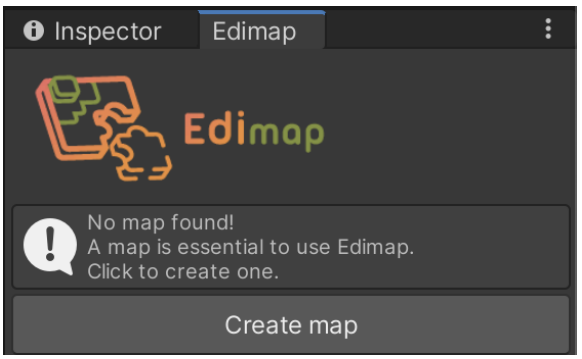
Vous pouvez ouvrir une nouvelle fenêtre de Edimap depuis le menu **Window** :

Window ➜ Synnaxium Studio ➜ Edimap



ℹ Une unique fenêtre peut être affichée à la fois.

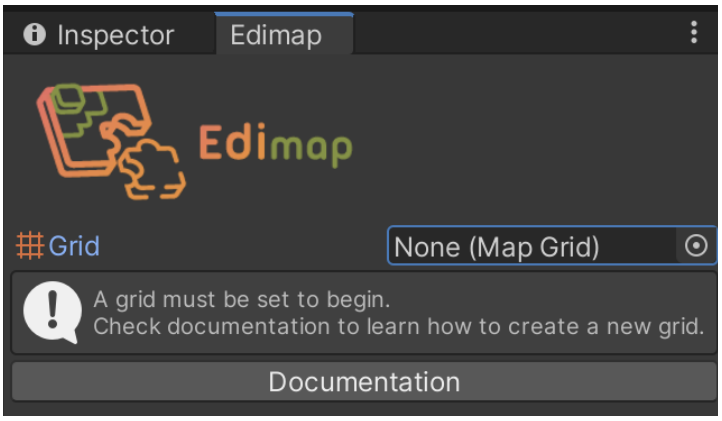
Si aucune carte n'a été créé dans la scène, un bouton **Create map** apparaîtra dans la fenêtre.



Ce dernier permet de créer un nouveau *GameObject* "Map" dans la hiérarchie de la *SceneView*. Cet objet sera utilisé comme référence de la carte par Edimap, il comprendra tous les préfabs et configurations ajoutés par le mode édition.

Grid

Ce champ est indispensable pour pouvoir utiliser Edimap. Si aucune référence n'a été renseignée, alors un message d'information vous indique que ce champ doit être rempli obligatoirement.



i Une grille par défaut est référencée dans ce champ lors de la création d'une nouvelle carte.

La grille doit être instaurée lorsque la carte est créée pour la première fois et avant l'édition de celle-ci. Cette grille ne doit pas être modifiée par la suite. Pour en savoir plus sur la configuration d'une grille, consultez le chapitre MapGrid : Configuration de la grille.

Edit

Cet onglet permet d'accéder au paramètres d'édition ainsi qu'aux palettes définies par la grille, mais elle permet aussi d'activer ou de désactiver le mode édition.

Brush

Ces options définissent où et comment éditer la carte, elles permettent de gérer les paramètres du pinceau.

Hide Layer Below

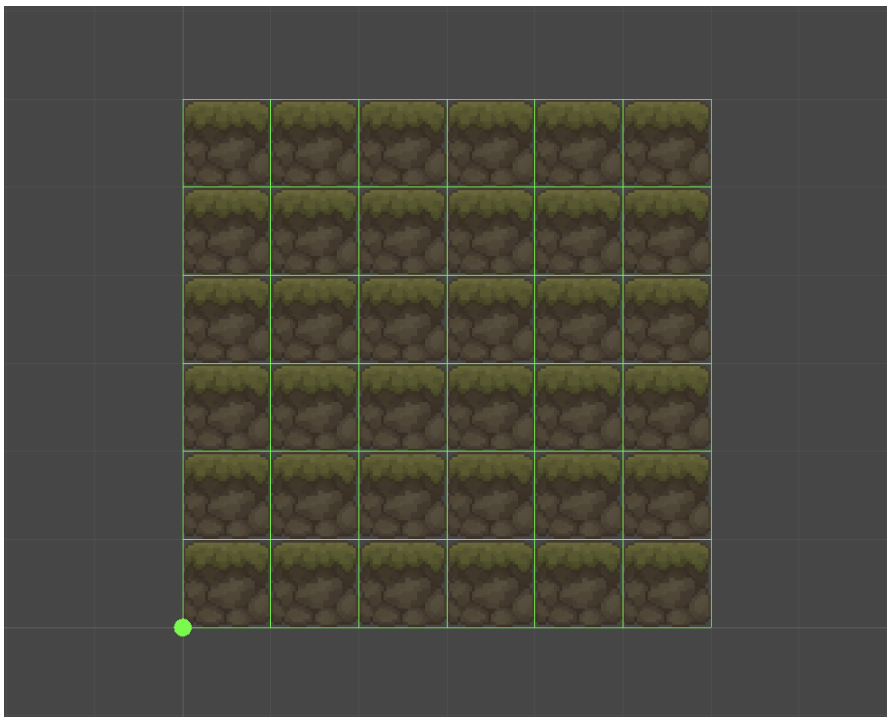
Les layers inférieurs à la valeur définie par ce champ seront invisibles. Cela permet d'éviter certaines situations incommodantes comme par exemple lorsque l'on souhaite ajouter un objet au rez-de-chaussée alors que le premier étage bloque la vue.

Layer

Définit la profondeur souhaitée de l'objet que l'on souhaite placer sur la carte.

Size X & Size Y

Il s'agit de la taille du pinceau utilisé pour placer ou supprimer plusieurs objets à la fois.



i Lorsqu'un pinceau de plusieurs cases est utilisé, le tiling n'est pas résolu au sein de la prévisualisation.

Shape

Spécifie la forme du pinceau que l'on souhaite utiliser.

i Il est possible d'implémenter de nouvelles formes en héritant de la classe **BrushShape**. Les formes de bases sont disponibles dans le dossier :
Assets/Synnaxium/Edimap/Scripts/Impl/BrushShapes

⚠ **Une fois une forme ajoutée ou supprimée, il est conseillé de cliquer sur le bouton ⟳ pour rafraîchir les formes disponibles.**

Palette

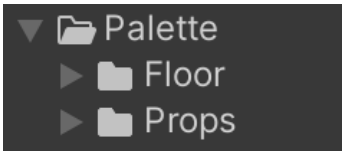
Cette partie regroupe toutes les palettes et leurs éléments pouvant être placés sur la carte.

i Un bouton pour rafraîchir la palette est disponible, il permet de prendre en compte les modifications d'une palette au cas où celle-ci aurait été modifiée.

Folder

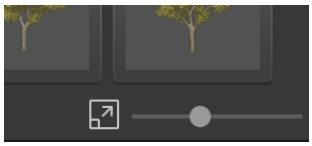
Il s'agit d'une liste des différentes palettes disponibles. Il suffit d'en sélectionner une pour avoir accès à son contenu.

Pour rappel, il s'agit des sous-dossiers directement sous le dossier principal de la palette.



La grille des objets

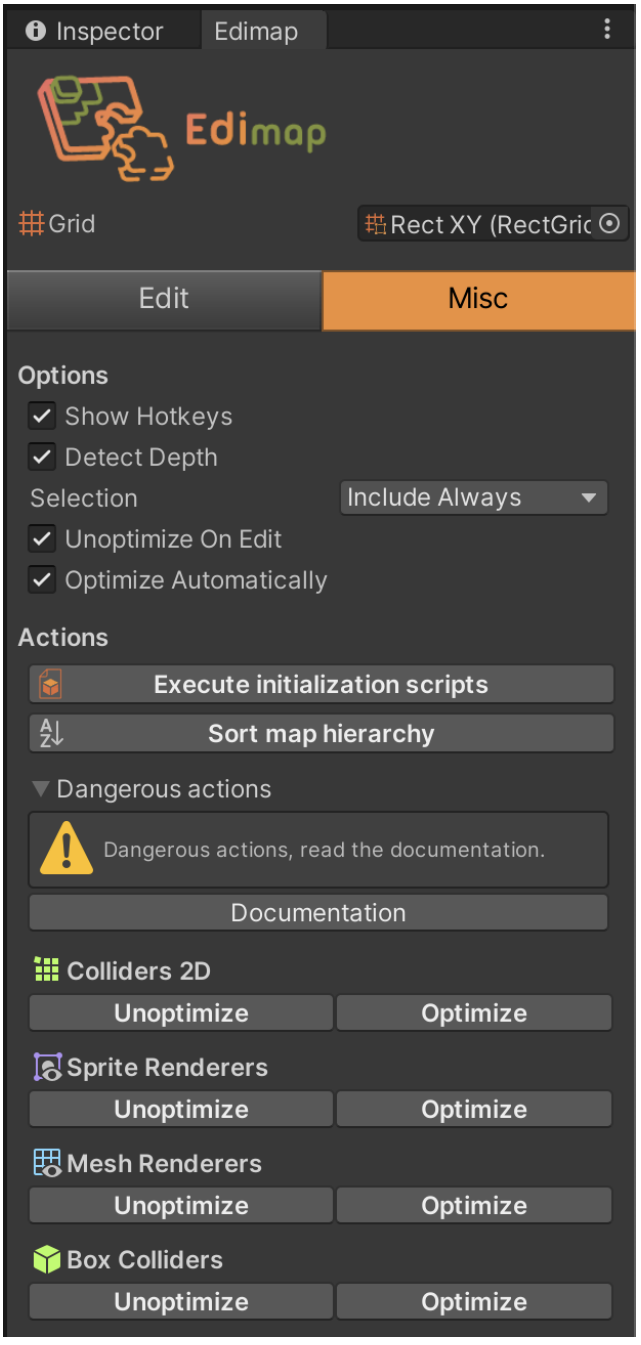
La liste des préfabs disponibles de la palette courante qui peuvent être sélectionné pour être placé sur la carte.



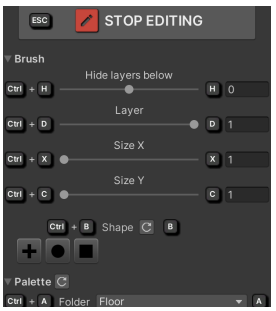
i En bas à droite, un slider permet de varier la taille des cases de prévisualisation des préfabs dans la grille.

Misc

Permet d'accéder à des options et des outils supplémentaires.



Show Hotkeys



Si cette option est activée, les icônes des raccourcis sont visibles sur l'interface lorsque le mode édition est lancé, afin de les rappeler à l'utilisateur.

i Les raccourcis sont personnalisables, si vous souhaitez en savoir plus consultez le chapitre Raccourcis.

Detect Depth

Active la détection automatique du layer sur lequel ajouter le préfab sélectionné lorsque le mode édition est enclenché. Toutefois, dès lors que la valeur de **Layer** est modifiée, cette option est temporairement désactivée jusqu'à ce qu'une nouvelle édition soit démarrée.

i Cette fonctionnalité est utilisée lorsque plusieurs layers sont disponibles dans la grille utilisée, autrement dit lors de l'édition de carte 3D principalement.

Selection

Permet de choisir quel type d'objet sont autorisés à être sélectionnés dans la *SceneView*. Il est possible de choisir plusieurs types, ou bien aucun en sélectionnant **None**.

Unoptimize On Edit

Renvoie au paramètre Unoptimize On Edit de la grille actuellement référencée dans Edimap.

Optimize Automatically

Renvoie au paramètre Optimize Automatically de la grille actuellement référencée dans Edimap.

Execute initialization scripts

Ce bouton exécute tous les scripts d'initialisation présent dans la *SceneView*. Pour en savoir plus sur les scripts d'initialisation, reportez vous au chapitre Exécuter un script lors de l'édition.

Sort map hierarchy

Réorganise toute la hiérarchie du *GameObject* "Map" par ordre alphabétique. Cela permet de retrouver plus facilement un objet présent dans ce *GameObject*.

Dangerous actions

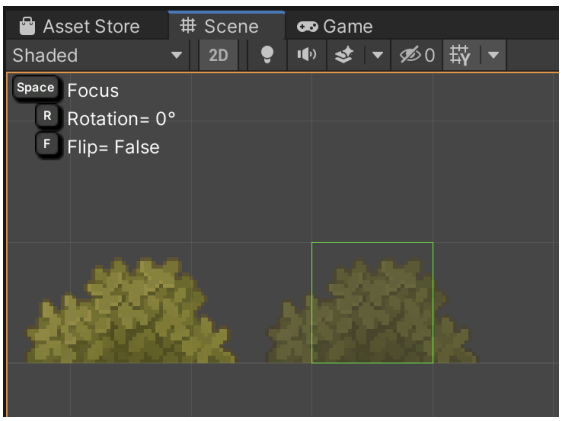
Ces actions peuvent potentiellement affecter les performances de votre carte. Actuellement, cette section comporte uniquement des actions d'optimisation et leurs inverses, c'est pourquoi nous vous renvoyons vers le paragraphe Optimisation.

La scène

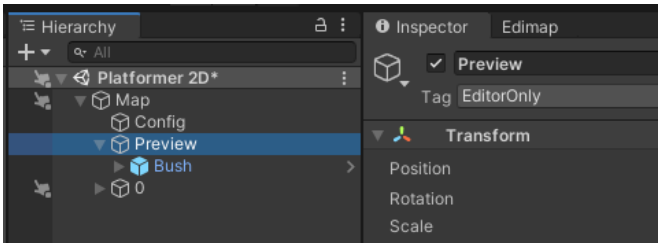
Prévisualisation

Lorsque le mode édition est activé, on peut remarquer plusieurs changement :

- ➔ un contour orange autour de la *SceneView*,
- ➔ des indications sur le placement du préfab,
- ➔ un contour autour du préfab de la taille définie,
- ➔ une version transparente du préfab pour visualiser le futur rendu.

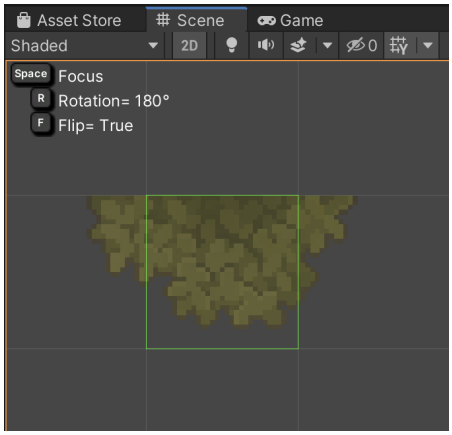


i Le visuel du préfab est réalisé en créant une copie de celui-ci mais sans aucun composant *MonoBehaviour*. Ainsi les éventuels méthode *Awake*, appelés en script éditeur via l'attribut *ExecuteInEditorMode*, ne seront pas appelés.



Focus, Rotations & Flip

Lors de l'édition de la carte, il est possible d'effectuer un focus sur le pinceau, une rotation ou une inversion du préfab.



Comme on peut le voir dans l'exemple ci-contre, illustrant une rotation de 180° et une inversion, des raccourcis clavier sont utilisés pour procéder à ces actions.

Raccourcis

Edimap fournit un ensemble de raccourcis pour diverses actions afin d'accélérer l'édition de la carte de jeu.

⚠ **Ces raccourcis sont fonctionnels uniquement lorsque le mode édition est activé.**

La liste des raccourcis par défaut est la suivante :

Raccourci	Action	Mémotechnique
Escape	Quitter le mode édition	
Space	Focus sur la prévisualisation	
D	Augmenter la profondeur (positif)	D pour Depth
Ctrl+D	Diminuer profondeur (négatif)	D pour Depth
H	Augmenter la profondeur cachée	H pour Hide
Ctrl+H	Diminuer la profondeur cachée	H pour Hide
X	Augmenter la taille du pinceau en X	X pour l'axe X
Ctrl+X	Diminuer la taille du pinceau en X	X pour l'axe X
C	Augmenter la taille du pinceau en Y	Lettre à côté de X
Ctrl+C	Diminuer la taille du pinceau en Y	Lettre à côté de X
B	Sélectionne la forme du pinceau suivante	B pour Brush
Ctrl+B	Sélectionne la forme du pinceau précédente	B pour Brush
V	Sélectionne le préfab suivant	
Ctrl+V	Sélectionne le préfab précédent	
A	Sélectionne le dossier de préfab suivant	
Ctrl+A	Sélectionne le dossier de préfab précédent	
F	Inverser le préfab	F pour Flip
R	Tourner le préfab	R pour Rotation

i Il vous est possible de personnaliser ces raccourcis en créant votre propre configuration globale de Edimap (cf. Configuration globale). Le groupe **Hotkeys** de cette dernière contient l'ensemble des raccourcis disponibles ainsi qu'un bouton pour rétablir les valeurs par défaut.

Exécuter un script lors de l'édition

Lorsqu'un nouveau préfab est placé sur une carte via Edimap, il est possible d'exécuter un script pour gérer une configuration automatique, comme par exemple :

- Ordonner les sprites dans un jeu 2D top-down.
- Connecter automatiquement une paire de téléporteurs.
- Connecter automatiquement un levier à une porte.
- etc.

Puisque Edimap se charge d'instancier les préfabs, nous avons par la même occasion l'opportunité d'exécuter du code sans dépendre de l'attribut *ExecuteInEditMode*.

MapObjectInit : Configuration au dossier

Au sein de vos **PaletteSettings**, vous trouverez un tableau **Inits** pouvant contenir des *ScriptableObject* de type **MapObjectInit**.

Vous pouvez créer votre propre implémentation en héritant de ce script. Par exemple, le code ci-dessous modifie la couleur des tuiles en fonction de leur profondeur :

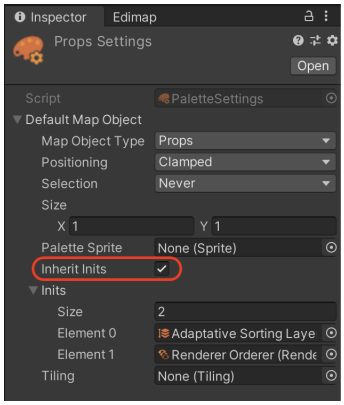
```
using UnityEngine;

namespace Synnaxium.Edimap
{
    [CreateAssetMenu(menuName = "Synnaxium Studio/Edimap/Map Object Inits/Depth Color")]
    public class DepthColor : MapObjectInit
    {
        [SerializeField] private float distance = 5f;

        public override void InitGameObject(MapGrid grid, MapObjectSettings settings, GameObject gameObject)
        {
            var renderers = gameObject.GetComponentsInChildren<SpriteRenderer>();

            foreach (var renderer in renderers)
            {
                renderer.color = Color.Lerp(renderer.color, Color.black, renderer.transform.position.z / distance);
#if UNITY_EDITOR
                UnityEditor.PrefabUtility.RecordPrefabInstancePropertyModifications(renderer);
#endif
            }
        }
    }
}
```

L'argument *grid* est un descriptif de la grille actuellement utilisée, tandis que *settings* contient la configuration de l'objet (positionnement, nombre de cases utilisées, type d'objet, etc.).



i Les scripts seront exécutés dans l'ordre de présence au sein du tableau.

i Vous remarquez la présence d'une option **Inherit Inits** au sein de votre **PaletteSettings**. Si cette option est cochée, le dossier hérite alors des initialisations configurées dans les **PaletteSettings** des dossiers parents.

IMapObjectInit : Configuration par objet

Il vous est possible d'hériter de l'interface **IMapObjectInit** sur l'un de vos *MonoBehaviour* :

```
public class MyCustomCode : MonoBehaviour, IMapObjectInit
{
    public void MapObjectInit(MapGrid grid, MapObjectSettings settings)
    {

    }
}
```

Attachez ce *MonoBehaviour* à votre préfab dans un *GameObject* enfant, en ajoutant le tag *EditorOnly*, et le tour est joué.

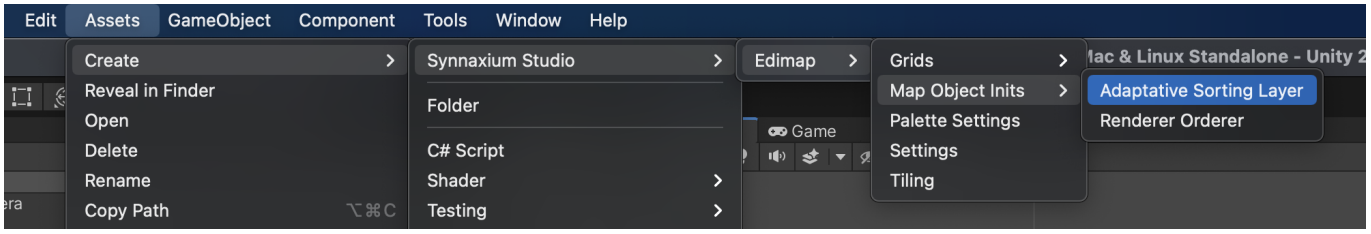
i Ce mode de fonctionnement est plus intrusif car une modification du préfab est nécessaire, mais l'option est disponible si vous avez des besoins spécifiques pour un préfab en particulier.

AdaptativeSortingLayer

Edimap propose un processus d'initialisation pour les objets dédiés aux cartes 2D avec plusieurs profondeurs, il s'agit de **AdaptativeSortingLayer**.

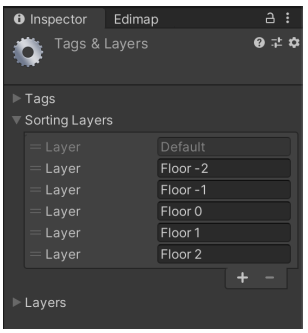
Ce script d'initialisation est disponible dans le menu contextuel de création d'asset :

Create ➜ Synnaxium Studio ➜ Edimap ➜ Map Object Inits ➜ Adaptative Sorting Layer



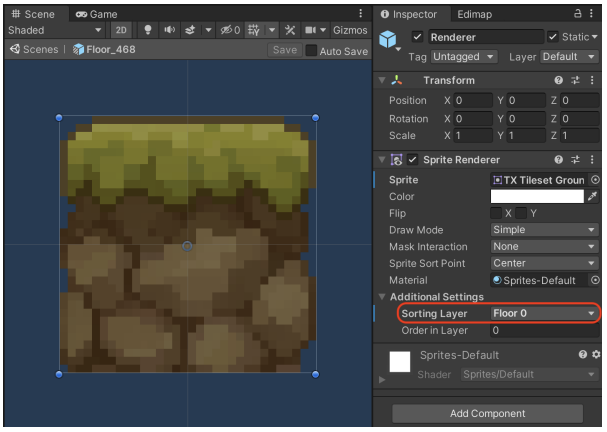
L'asset possède une expression régulière (regex) en paramètre qui permet de définir la syntaxe de vos *Sorting Layers*.

Par défaut, cette regex va permettre d'interpréter les *Sorting Layers* qui terminent par un numéro de profondeur.



i La regex doit toujours récupérer l'indice de profondeur sous le nom de groupe "layer".

Référencez cette initialisation dans la tableau **Inits** de votre **PaletteSettings** et assurez-vous que vos préfabs utilisent un des *Sorting Layers* numérotés.



Ainsi, **AdaptativeSortingLayer** s'occupera de faire correspondre les *Sorting Layers* en fonction du layer dans lequel le préfab sera placé.

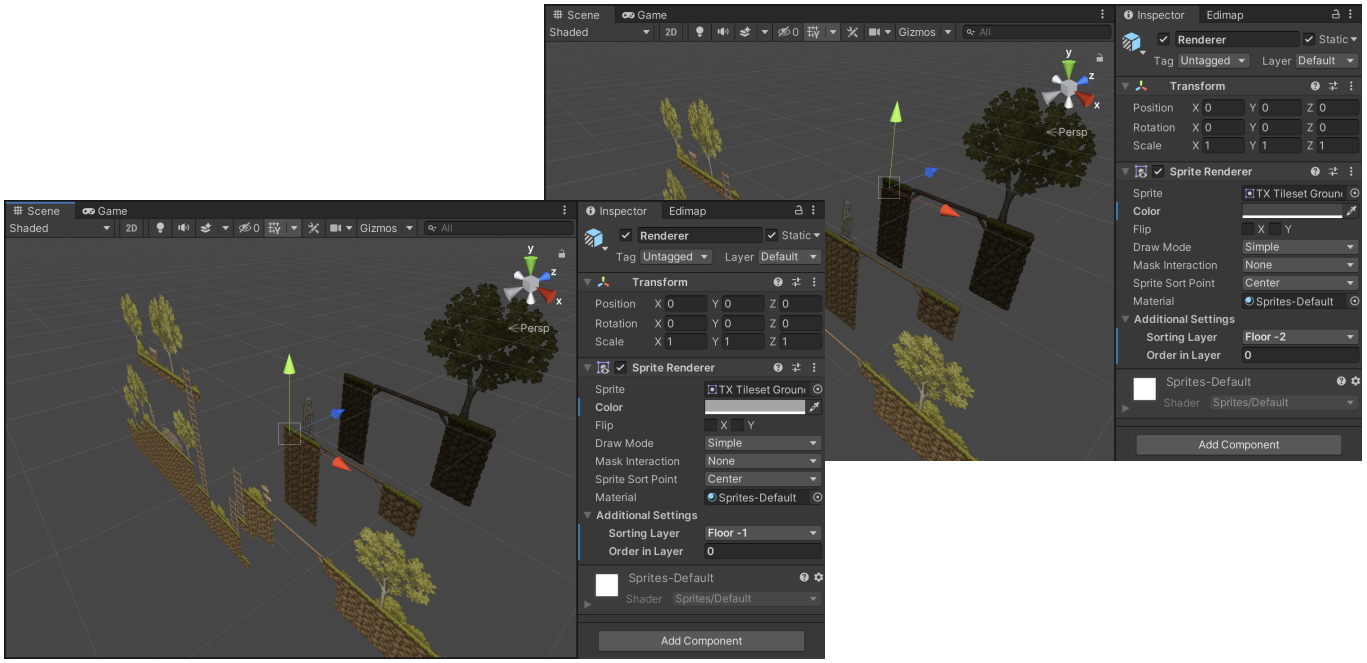


Illustration de l'utilisation du script d'exemple DepthColor et de AdaptativeSortingLayer.

RendererOrder

Ce script d'initialisation est utile aux cartes 2D sans utilisation de profondeur, notamment pour les jeux 2D en vue du dessus ou isométrique. Il permet de définir automatiquement le paramètre **Order in Layer** des renderers des tuiles en fonction de leur position.

Comme le script précédent, celui-ci est disponible dans le menu contextuel de création d'assets :

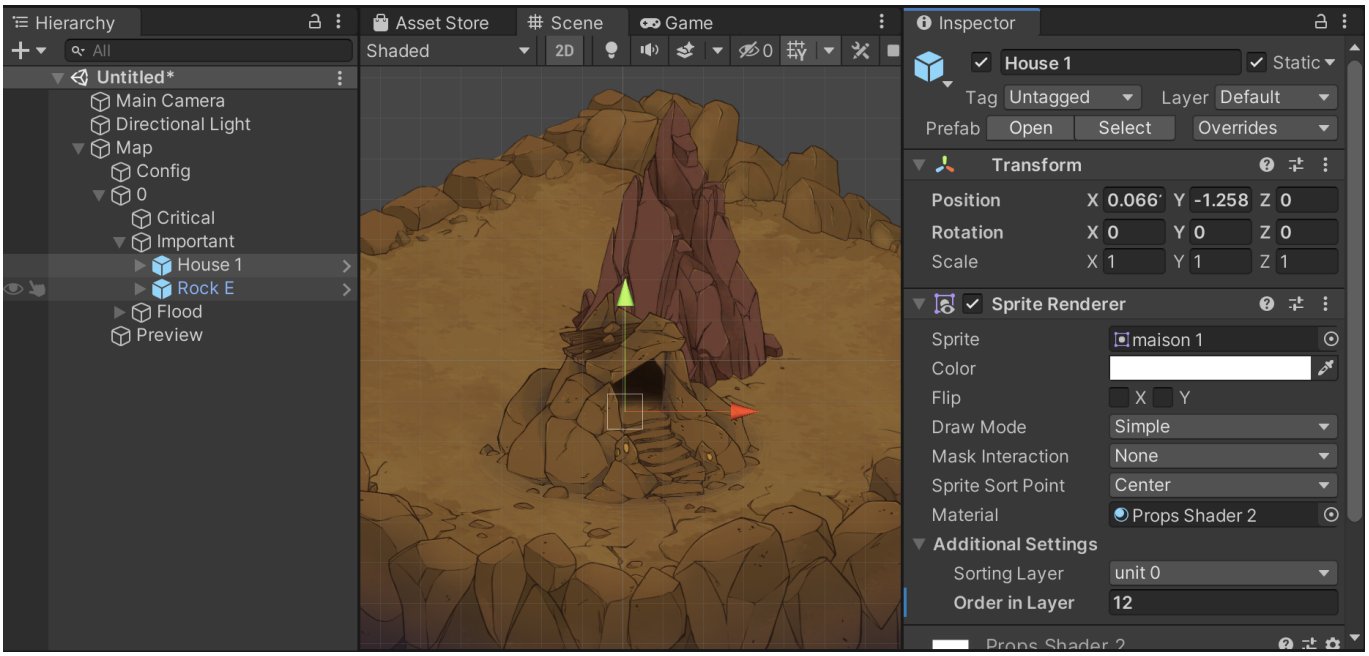
Create ➜ Synnaxium Studio ➜ Edimap ➜ Map Object Inits ➜ RendererOrder

Y Factor

Il s'agit du facteur par lequel la position du sprite sera multipliée pour obtenir l'ordre de tri.

Pivot Child Name

Si un enfant existe et que le nom de l'enfant correspond, alors le pivot du sprite sera remplacé par la *Transform* de cet enfant.



Dans l'exemple ci-dessus, *House 1* a un **Order In Layer** plus élevé que *Rock E*, ainsi il s'affiche au-dessus de ce dernier. Pourtant ces deux objets ont le même **Sorting Layer**.

Optimisation

Edimap utilise un workflow dit par préfab, ce qui signifie qu'un préfab est instancié pour chaque case de votre jeu.

L'énorme avantage de l'approche est de vous permettre d'avoir des concepts concrets pour chaque partie de votre carte : la lave, les portes, les créatures, les checkpoints, etc.

Une tilemap classique ne vous donnerait qu'une représentation abstraite de l'environnement, et il vous resterait encore pas mal de travail pour avoir une carte jouable. Ce n'est pas le cas avec Edimap, l'édition d'une carte vous donne directement le produit fini.

Évidemment, il n'est pas possible de livrer un jeu où chaque case est un préfab, il y aurait des dizaines de milliers de renderer et de colliders, cela ferait chuter vos performances.

La solution est de travailler en préfab, mais d'optimiser le rendu final en venant fusionner les renderers et les colliders ensemble dans un second temps.

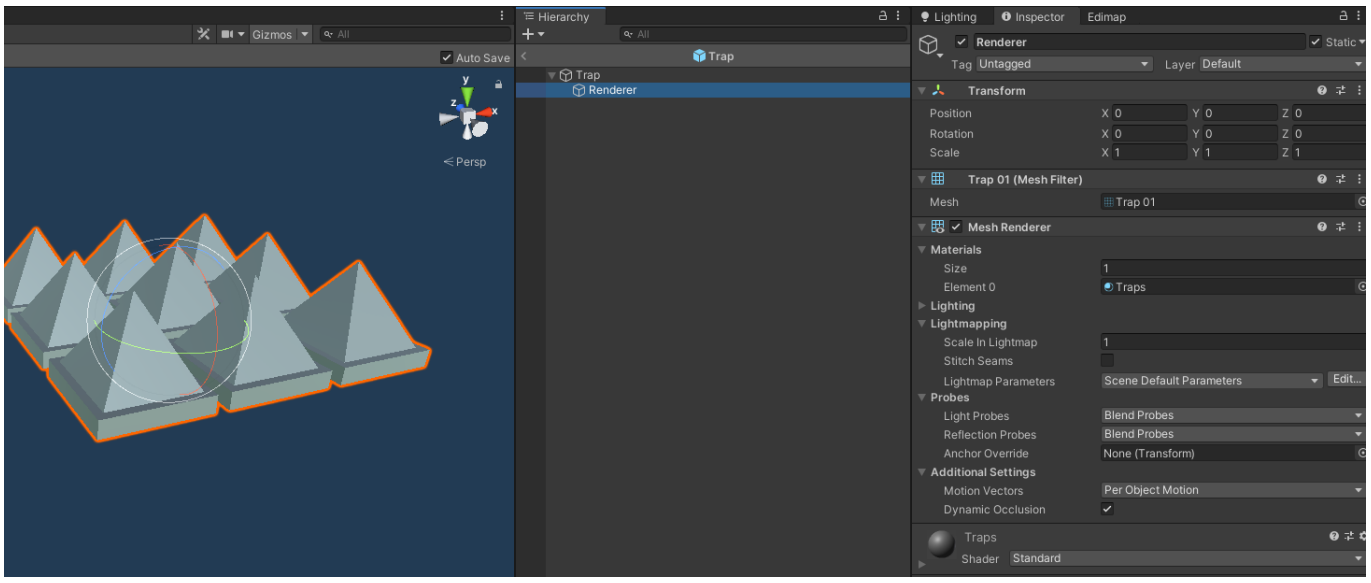
Design des préfabs

Lorsque vous concevez des préfabs, plusieurs cas d'usage classiques peuvent apparaître :

- Le préfab est composé d'un seul renderer ou d'un seul collider.
- Le préfab mixe plusieurs notions entre le rendu, la collision et le gameplay.

Dans le cas simple, il n'y a pas de considération particulière à avoir, si ce n'est de s'assurer que le préfab soit noté comme **Static**.

Pour les préfabs mixant plusieurs responsabilités, il va falloir décomposer en *GameObject* enfants pour permettre à Edimap d'isoler les composants.



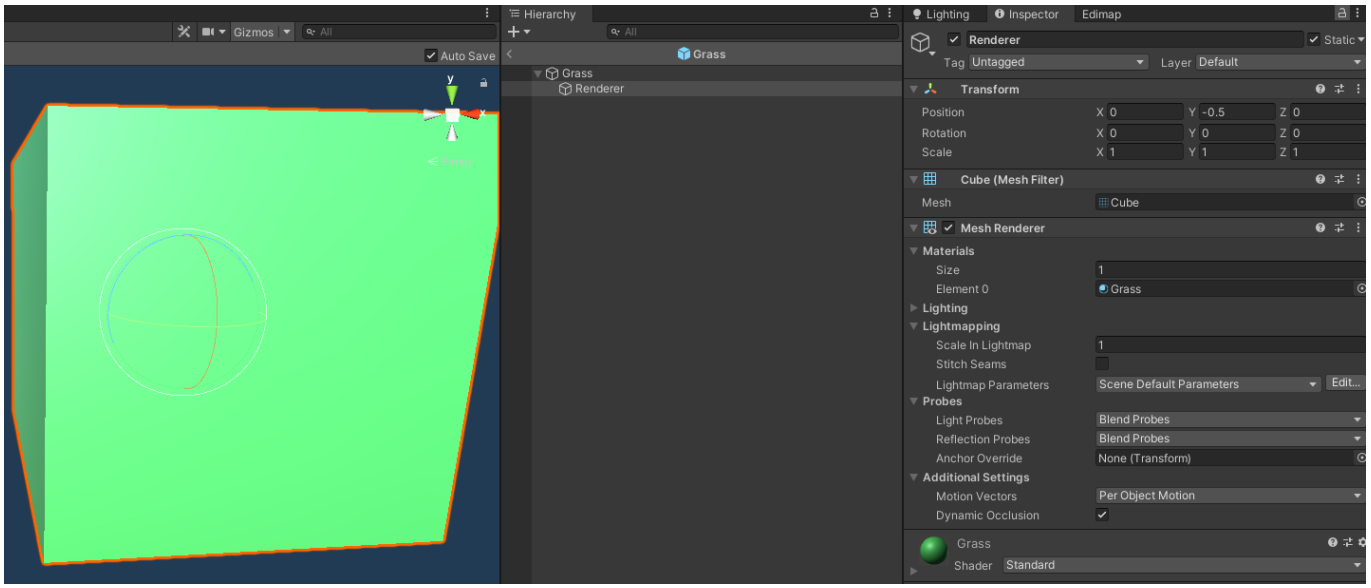
Par exemple, avec les pièges de l'exemple précédent, nous allons avoir deux fonctionnalités. Ils ont un rendu avec un *MeshRenderer* et vont blesser le joueur si celui-ci marche dessus.

Lorsque Edimap va optimiser ce préfab, le tag du *GameObject* contenant le *MeshRenderer* deviendra *EditorOnly*, ainsi cet objet ne sera pas présent au sein du build.

Toutefois, nous ne souhaitons pas perdre la partie qui inflige des dégâts au joueur.

Pour gérer cela, il nous faut donc deux *GameObject* :

- un parent qui s'occupera des dégâts,
- un enfant qui s'occupera du rendu et qui sera potentiellement éliminé du build par Edimap.

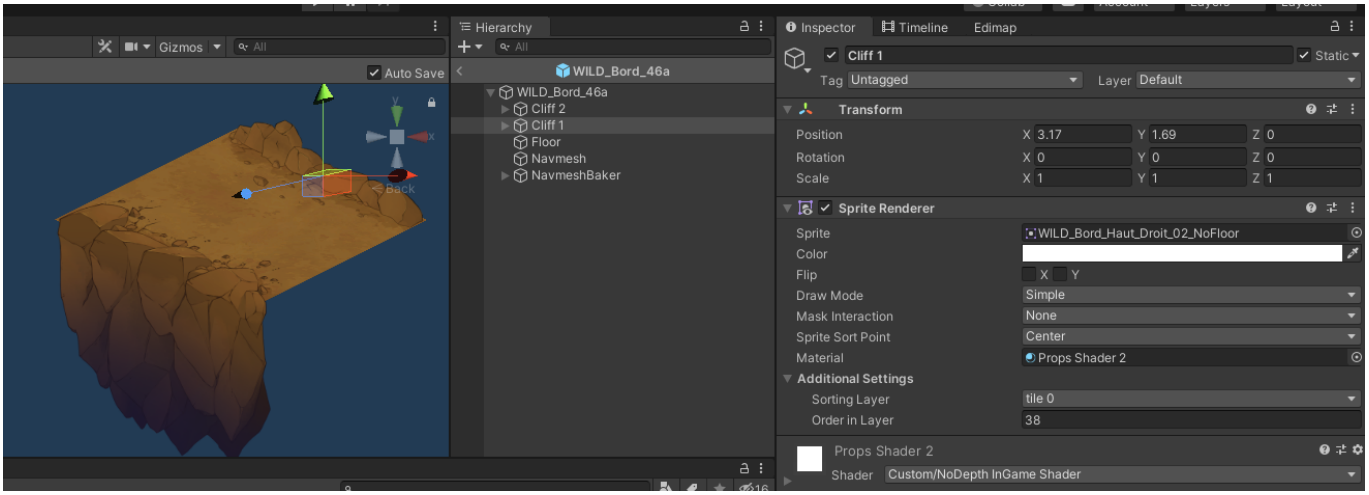


Un avantage inhérent à cette approche est la gestion de vos pivots.

Ici cet objet "Grass" va être éliminé du build entièrement, car la racine est le collider qui sera optimisé.

Même sans l'excuse du collider sur l'objet parent, il aurait été tout à fait possible de simplement mettre la racine en *EditorOnly* malgré tout, sachant que nous comptons optimiser le *MeshRenderer* dans tous les cas.

Comme notre objet sera converti en un autre mesh par Edimap, nous avons l'avantage de pouvoir ajouter des *GameObject* superflus simplement pour corriger le positionnement de nos objets, ce qui nous permet de limiter les aller-retours entre les équipes d'intégrations et les artistes.



Dans l'exemple ci-dessus, nos sols pourraient rapidement devenir infernaux à intégrer. Ici le sol est composé de trois parties qui utilisent deux materials, ce qui nous oblige à les séparer.

Le rendu isométrique nous impose également quelques contraintes particulières sur les positions des sprites pour pouvoir générer automatiquement un ordre de rendu correct.

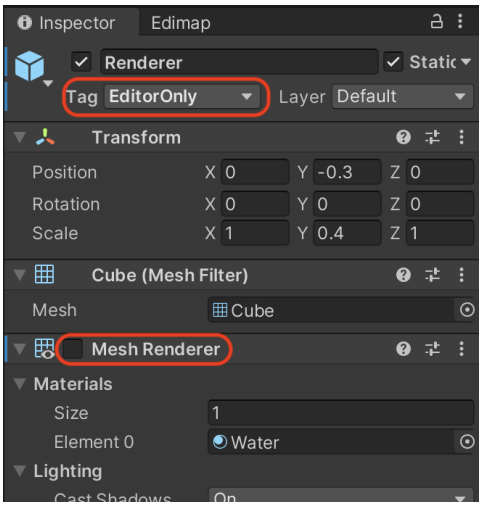
Une solution possible serait d'imposer aux graphistes des exports dans lesquels le pivot est automatiquement placé au au bon endroit pour chaque partie. Cependant, cette manière de faire engendre beaucoup de transparence inutile dans les fichiers image, et fait souvent perdre un temps substantiel aux graphistes qui n'ont pas forcément une compréhension parfaite des contraintes techniques.

Ici, nous pouvons simplement placer les parties dans le préfab nous-même et sans surcoût en jeu.

Détails techniques

Lorsque vous optimisez une partie de votre carte, qu'il s'agisse de *SpriteRenderer* ou de *BoxCollider*, Edimap procède à l'optimisation de manière similaire.

Dans un premier temps, les objets concernés sont passés en *EditorOnly* et les composants optimisés sont désactivés.

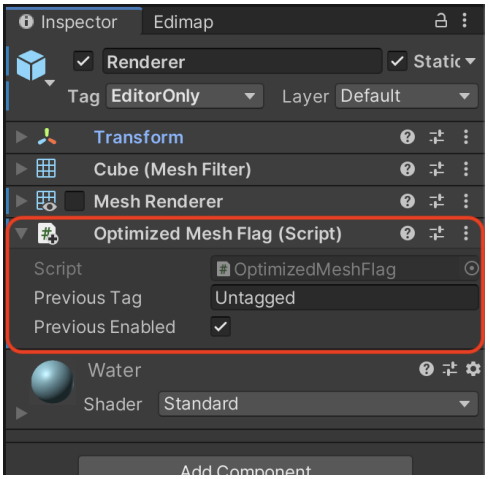


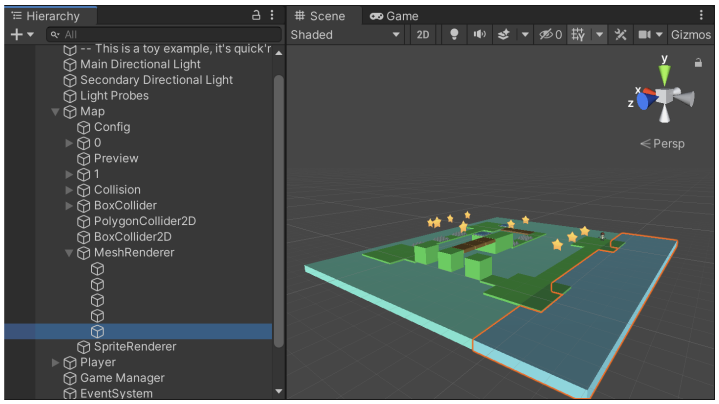
En second temps, un composant est également attaché à l'objet pour que Edimap puisse se souvenir que l'objet a été optimisé.

Dans le cas où vous retirez l'optimisation, cela permet à Edimap de restituer les anciennes valeurs.

⚠ **Ne supprimez jamais ce composant vous-même.**

i Nous avons choisi de le laisser visible pour que vous puissiez rapidement identifier pourquoi un de vos composants est désactivé.



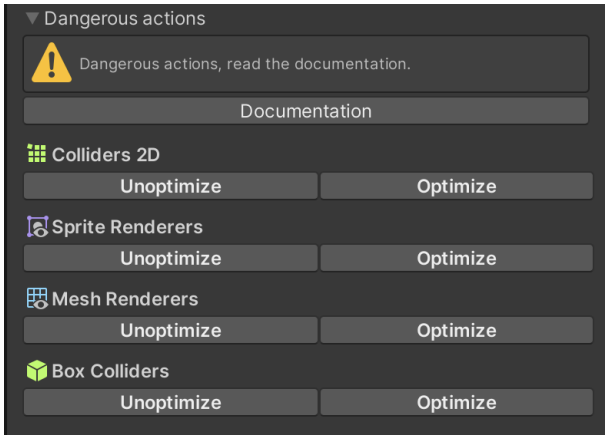


Enfin, Edimap va ajouter de nouveaux objets au sein de votre carte, dans une section du même nom que le composant optimisé.

Vous constaterez alors un ensemble de *GameObject* sans nom, où chaque objet est un regroupement de vos composants.

Utilisation

Depuis la fenêtre Edimap, il est possible d'accéder à la section **Dangerous actions** avec l'onglet **Misc**. Celle-ci regroupe les différentes optimisations possibles avec deux boutons, un pour optimiser la carte et un autre pour son inverse.



i Des paramètres d'optimisation ou de désoptimisation automatique sont disponibles via le *ScriptableObject* de la grille ou dans la partie Misc de la fenêtre Edimap.

i Pour automatiser le processus sur un ensemble de scènes, vous pouvez utiliser l'API de Edimap via les classes **BoxColliderOptimizer**, **SpriteRendererOptimizer**, etc.

SpriteRenderer

L'optimisation des *SpriteRenderer* suit les mêmes règles que le dynamic batching de Unity.

Les conditions à respecter pour fusionner deux *SpriteRenderer* sont :

- Ils utilisent le même material avec les mêmes paramètres.
- Ils ont un ordre de rendu qui n'est pas séparé par un troisième *SpriteRenderer* incompatible.

Colliders 2D

Pour les *BoxCollider2D*, l'optimisation est actuellement gloutonne, ce qui signifie que la solution n'est pas strictement optimale, mais largement suffisante.

Deux *BoxCollider2D* seront probablement fusionnés si ils peuvent être parfaitement englobés par un seul *BoxCollider2D* en retouchant les paramètres du centre et de la taille.

Pour les *PolygonCollider2D*, nous utilisons l'implémentation *Clipper* garantissant ainsi une solution optimale. Toutefois, les *PolygonCollider2D* ont inhéremment de mauvaises performances au sein de Unity. Il est donc à utiliser avec modération, il est préférable de privilégier les *BoxCollider2D* autant que possible.

MeshRenderer

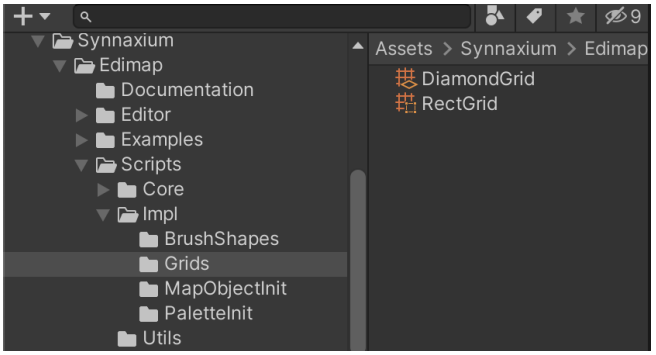
Les *MeshRenderer* seront optimisés ensemble s'ils utilisent les mêmes material avec les mêmes paramètres.

BoxCollider

À l'instar des *BoxCollider2D*, deux *BoxCollider* seront fusionnés si il est possible de les englober exactement avec un *BoxCollider* plus grand. La géométrie est parfaitement préservée, mais la solution n'est pas forcément optimale. Toutefois, la qualité de l'algorithme est largement suffisante.

Grilles personnalisées

Fonctionnement



Edimap vous propose deux types de grille par défaut : la grille rectangulaire et la grille losange.

La grille rectangulaire vous permet de réaliser des quadrillages et toute forme rectangulaire, tandis que la losange vous permet de faire de l'isométrie (une grille isométrique est composée de losanges de taille de case de 2 par 1).

Ces deux grilles peuvent ne pas correspondre à vos attentes, aussi Edimap met à disposition une API pour créer votre propre implémentation. Pour cela, il vous suffit de créer un script de *ScriptableObject* héritant de **MapGrid**, et d'y remplir les méthodes abstraites. Une fois votre grille créée, il ne vous reste plus qu'à la référencer dans la fenêtre Edimap pour l'utiliser.

i N'hésitez pas à copier le code des deux grilles basiques de Edimap pour vous inspirer.

Coordonnées

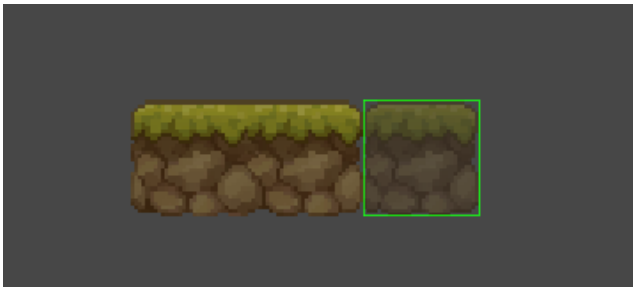
La grille de Edimap fonctionne toujours par coordonnées discrètes qui sont représentées par des *Vector2Int*.

Chaque case au sein de votre grille doit donc être uniquement identifiée par un couple d'entiers x et y. Votre implémentation de grille va principalement consister à établir une fonction pour passer de cette paire d'identifiants à une coordonnée dans l'espace et vis-versa.

i À noter que la notion de grille XY et XZ n'est pas présente dans cette API. Pour vous simplifier la vie, cette transposition est réalisée à part.

OnWorldToCell

Lorsque vous pointez avec votre curseur au sein de la scène, Edimap en déduit une cellule sélectionnée. C'est ce qui vous permet de placer des objets côte à côte sans superposition.



Paramètre	Type	Description
position	*Vector2*	La position spatiale à traduire en coordonnées.
retour	*Vector2Int*	La coordonnée permettant d'identifier cette cellule.

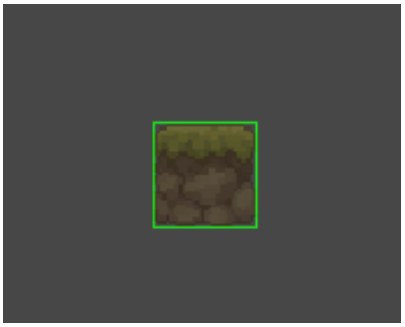
OnCellToWorld

Lorsque Edimap souhaite placer un objet dans la scène, il va placer cet objet sur la cellule sélectionnée. Il est ensuite nécessaire de traduire cette coordonnée en un point dans l'espace au sein de la scène.

Paramètre	Type	Description
cell	*Vector2Int*	La coordonnée de la cellule.
retour	*Vector2*	Le centre de la cellule.

OnDraw

Appelé lorsque Edimap souhaite afficher une prévisualisation de la grille pour l'utilisateur. Cette méthode a pour but de retourner une liste de points utilisés pour tracer la grille d'une cellule.



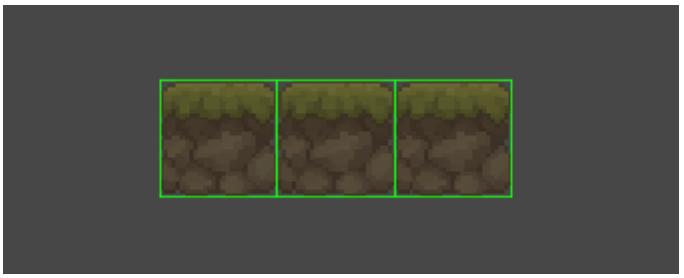
***OnDraw** décrit les lignes vertes de la prévisualisation*

Les points sont à fournir deux par deux au sein de la liste. Chaque paire de points forme une ligne dans la scène.

Paramètre	Type	Description
objectSize	*Vector2Int*	La taille de la cellule à afficher (pour les objets qui s'étendent sur plusieurs cases)
cell	*Vector2Int*	La cellule à prévisualiser. Si la taille de l'objet fait plus de une case, il s'agit de la cellule la plus en bas à gauche (plus petits indices x et y).
z	*float*	La profondeur à laquelle la prévisualisation a lieu. Il faut injecter cette valeur dans le composant z des points au sein de la liste.
retour	*Vector3[]*	Un tableau de points. Chaque paire de points donnera une ligne.

OnCellDelta

OnCellDelta sert à définir l'écart entre deux cellules lorsque l'on incrémente les coordonnées.



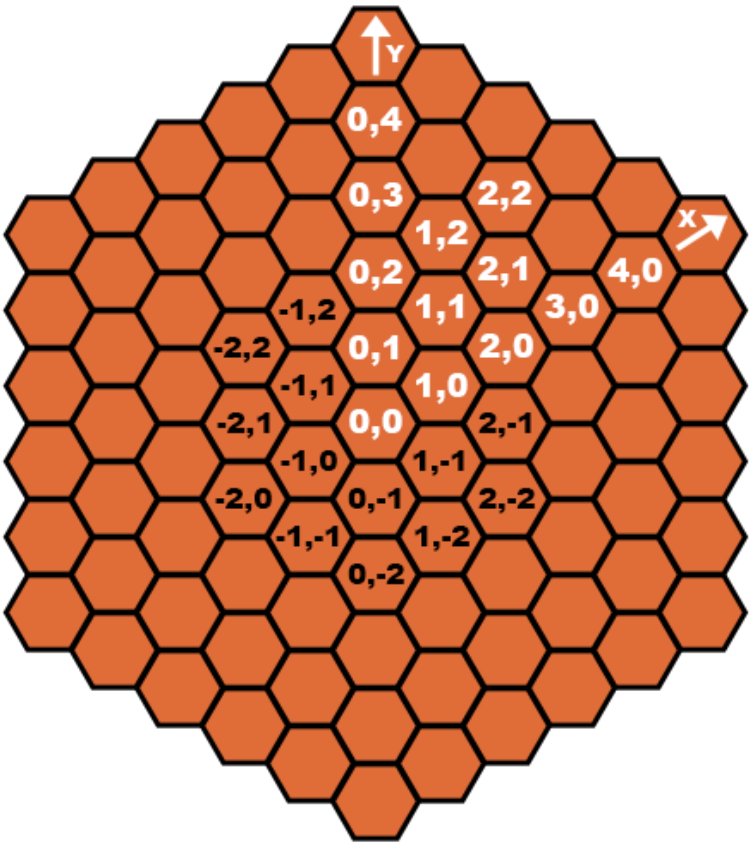
*La brosse se sert de **OnCellDelta** pour décaler les prévisualisations.*

Paramètre	Type	Description
x	*int*	Le décalage de coordonnée en X
y	*int*	Le décalage de coordonnée en Y
retour	*Vector2*	Le décalage résultant en coordonnées spatiales

Exemple : Grille hexagonale

Actuellement, la grille hexagonale n'est pas encore implémentée. Son intégration est prévue dans un futur proche en fonction de l'engouement de Edimap par la communauté Unity.

En attendant, nous pouvons tout de même vous expliquer rapidement la démarche en schématisant la grille avec ses axes et quelques coordonnées :



La grille hexagonale peut parfois effrayer, avec ses 6 voisins par case.

Mathématiquement, du point de vue de Edimap, il s'agit d'une grille 2D tout à fait banale. La seule véritable subtilité est le positionnement de l'axe X qui est à 30° par rapport à la direction habituelle.