Oxygen Not Included

Analyse Théorique :

1. De quel type de jeu s’agit-il ?

Oxygen Not Included est un jeu de gestion qui mélange city-builder et survival, où le joueur doit gérer efficacement ses ressources, matérielles comme humaines, afin de faire fonctionner et prospérer sa colonie.

1. Après une heure de jeu, nommez les différentes mécaniques de gameplay rencontrées.

Il y a énormément de mécaniques de gameplay dans Oxygen Not Included.

La plus importante est la **destruction de l’environnement**. Les “humains” (appelées “duplicants”, duplicata en français - ou clone, pour simplifier) peuvent détruire la majorité des blocs solides sur leur chemin, et récolter puis stocker les ressources. Certains blocs sont trop durs pour les duplicants normaux, ce qui requiert leur spécialisation, que l’on verra plus tard.

La seconde plus importante mécanique est la **construction**. Comme elle est dépendante de la récolte de ressources, je la classe comme étant très légèrement moins importante que la destruction de l’environnement. Les duplicants peuvent construire des appareils, des blocs, et bien plus encore, un peu partout, tant qu’ils ont les ressources nécessaires. Le jeu traque les ressources stockées et celles qui sont au sol, et le joueur est incapable de planifier la construction d’un bloc ou un appareil dont il ne possède pas les ressources nécessaires à la construction.

De la construction découle toutes les autres mécaniques ; la **gestion des duplicants**, par exemple, nécessite tout un tas de machines afin d’assurer leur prospérité. Les duplicants ont énormément de besoins : la faim, l’énergie, l’oxygène, la santé, la vessie, le moral … Chacun de ses besoins nécessitent une ou plusieurs machines pour en prendre soin : une cuisine pour la nourriture, des lits pour l’énergie, des générateurs d’oxygène …

Une des parties les plus importantes à gérer chez un Duplicant, à part sa survie, est son **moral**. Les duplicants peuvent gagner en expérience, et en niveau ; lorsqu’ils passent au niveau supérieur, ils peuvent choisir une compétence dans un arbre de compétence. Chaque duplicant à un intérêt, baissant d'un point le coût en moral de la compétence qui l’intéresse (en vérité, il augmente de un le moral du duplicant, mais le résultat est le même). Le moral n’est pas nécessaire afin de prendre une compétence, le joueur peut donner une balance de moral négative au duplicant, mais cela aura des effets négatifs sur son stress et son travail. Le moral peut être gagné de plein de façon différentes : avec de la meilleure nourriture, par exemple, ou des salles adaptées.

Les **salles** sont un autre aspect à gérer, même si il est assez mineur au début. Tout espace entouré de murs est une salle. Une salle n’a pas de forme spécifique. Cependant, le joueur peut adapter les salles afin de les transformer en salles spécialisées qui donnent des bonus aux duplicants ; par exemple, les *chambres communes* augmentent le moral de tous les duplicants qui y ont un lit assigné de 1 ; un *laboratoire* augmentera l’efficacité des machines de recherches qui sont à l’intérieur …

En parlant de **recherche**, il s’agit d’un autre aspect fondamental du jeu. Le joueur commence avec très peu de machines et de blocs débloqués, et doit construire la première machine de recherche afin d’assigner des intérêts de recherche ; à partir de la, tant que la machine est alimenté en électricité, un duplicant l’utilisera afin de gagner des points de recherche pour cette recherche spécifique. Les recherches sont divisées en plusieurs arbres de recherche, passant de la gestion d’animaux au management d’électricité ou de gaz …

L’**électricité** est une autre ressource fondamentale du jeu. Toutes les machines (pratiquement) fonctionnent à l’électricité. Certaines machines qui produisent de l’électricité demandent de l’électricité pour faire fonctionner les machines adjacentes qui assurent son bon fonctionnement ! Le joueur peut construire une variété de machines différentes, chacune ayant ses avantages et ses inconvénients. Certaines utilisent des ressources rares, et pas forcément facilement renouvelable ; d’autres ne génèrent tout simplement pas assez d’énergie … Le joueur a aussi accès à des batteries pour stocker le surplus d’électricité, ce qui peut être utile pour limiter la construction de fils électriques plus robustes mais plus coûteux en ressources, en stockant l’énergie et en la distribuant salles par salles via batteries.

Le **gaz** et les **fluides** vont de paire, et fonctionnent de manière relativement similaire. Les deux fonctionnent comme étant des blocs secondaires ; les gaz occupent toutes les cases qui ne sont pas remplies par des blocs solides ou fluides, et les fluides subissent la gravité et, similairement, occupent les cases qui ne sont pas remplies par des blocs solides ou des gaz. Les gaz et les fluides peuvent occasionnellement se bloquer entre eux, mais cela est plutôt rare. Les gaz et les fluides ont une valeur “maximum” (par exemple, 1t d’eau) pour occuper une case. La différence entre fluides et gaz est que les gaz tentent de se répandre de manière égale dans la pièce, et les fluides s’empilent, ligne après ligne. Certains fluides peuvent interagir avec d’autres, comme l’eau polluée qui contaminera l’eau classique.

La **température** est aussi une mécanique à gérer, permettant de préserver des aliments plus longtemps, affectant la santé des duplicants, l’intégrité de certains matériaux … Certaines machines produisent de la chaleur, et certains blocs/machines permettent aussi de la réduire. Certains biomes sont naturellement chauds, aussi, comme par exemple les biomes volcaniques, remplis de lave.

La **gestion du décor** est aussi une tâche importante, augmentant le moral des duplicants, et va de paire avec la **gestion de la lumière**, qui n’est pas nécessaire pour faire voir les duplicants mais augmente aussi le moral. La **gestion des matériaux** utilisés pour construire certains blocs et/ou machines permet de contrôler leur interaction avec certains éléments du monde extérieur, comme par exemple certains matériaux étant plus résistants que d’autres à la chaleur. Enfin, il y a aussi la **gestion des radiations**, si le joueur touche au nucléaire.

1. La gestion et la collecte des ressources dans ce jeu comportent une multitude de règles, définissez-en un maximum.

Lorsqu’un duplicant mine une ressource, ou détruit un bâtiment, ses composants minéraux tombent au sol ; les composants sont ensuite ramassés par les duplicants, soit quand ils en ont besoin pour construire quelque chose, soit pour le ranger dans un coffre si l’ordre à été donné.

Les coffres peuvent être filtrés afin de ne garder que certains types de ressources, voire une seule ressource spécifiquement.

1. La gestion des fluides/gaz dans ce jeu comporte une multitude de règles,

définissez-en un maximum.

Comme expliqué précédemment, les gaz s’étalent le plus possible dans l’espace disponible, tandis que les fluides vont vers le bas et s’étalent en longueur. Le joueur peut déplacer les fluides et les gaz à l’aide de pompes spécifiques, et de tuyaux adaptés. Les deux ont une contenance “maximale” par case, ce qui est plus facile à voir avec les fluides.

1. Le monde est généré d'une certaine façon ? Laquelle ? Expliquez les règles de cette

génération.

Chaque astéroïde possède des “biomes” qui régulent le type de ressource disponible. Les biomes sont arrangés aléatoirement, excepté pour le biome central dans lequel les duplicants spawnent au début. Chaque astéroïde à un type, qui définit quels biomes sont disponibles et à quelle fréquence.

1. Les déplacements des personnages ne se font pas au point and click mais plutôt à

l'aide de règles bien spécifiques. Lesquelles ?

A son niveau le plus basique, le joueur va assigner des tâches à accomplir, et les duplicants vont les accomplir dans l’ordre qu’ils souhaitent. Le joueur est incapable de prendre le contrôle d’un duplicant et de lui dire “va faire ça”. Cependant, il y a différents moyens de pousser ses duplicants à accomplir certaines tâches en priorité par rapport à d’autres.

Tout d’abord, le plus basique, la **gestion d’importance**. Le joueur peut appliquer 10 niveaux d’importances aux tâches, numérotés de 1 à 9 ainsi que l’alerte jaune. L’importance des tâches n’est pas figée, c'est-à-dire que le joueur peut l’ajuster comme il le souhaite, quand il le souhaite, pour chaque tâche. Elle est aussi applicable dès l’assignation de la tâche. Il est important de noter que les duplicants priorisent (note : ne font que) les tâches auxquelles ils ont accès. De plus, à part pour l’alerte jaune, les duplicants suivront leur planning à la lettre, même si une tâche d’importance 9 est assignée. Les duplicants ignoreront leur planning lorsqu’une tâche à une alerte jaune attribuée, excepté pour le sommeil.

Le **planning** est tangentiel à la gestion des tâches. Les duplicants ont des périodes de repos, de travail, de nettoyage, et de sommeil. Excepté dans le cas d’une alerte, les duplicants suivent toujours leur planning, même s' il y a une tâche que vous jugez cruciale en cours. Certains duplicants bénéficient de bonus lorsqu’ils travaillent la nuit (ceux qui ont le trait “oiseau de nuit”), et d’autres le matin, (ceux qui ont le trait “oiseau de jour”). La majorité n’ont pas ces bonus.

Un autre outil puissant dans la main du joueur est la **priorisation des types de tâches**. Le joueur peut assigner des priorités à chaques duplicants, par rapport à leurs compétences, afin qu’ils se focalisent sur certaines tâches et en délaissent d’autres. Ainsi, le minage sera assuré par ceux qui minent rapidement, la décoration par ceux qui sont créatifs, etc. Le joueur peut assigner 6 niveaux d’intérêt dans chaque tâche, basiquement un compteur numérique de 1 à 5 avec une option pour lui interdire tout simplement une tâche. A noter que certains traits de caractère font que certains duplicants sont littéralement incapables d’accomplir certaines tâches.

Enfin, lorsque plus rien ne va dans la colonie, le joueur peut déclencher une **alerte rouge**. Il s’agit d’un état de panique général, dans lequel les duplicants vont travailler jusqu’à mourir d’épuisement ou de faim, ignorant *tous* leurs besoins. Cet état d’urgence n’est pas sustainable, et résulte rapidement en la mort de vos duplicants, mais peut être utile par exemple si un duplicant se retrouve bloqué dans un liquide alors que tous vos duplicants sont en train de dormir.

1. Personnel : ce jeu vous a-t-il plu ? Pourquoi ?

Oui, car j’adore micro-manager une bande d’abrutis suicidaires et les pousser petit à petit à devenir capable de survivre dans le vide infini de l’espace. Et ensuite ils meurent tous un par un parce que je suis mauvais et j’ai une grosse issue de skill et je l’aime un peu moins.

Pratique :

~~Définissez, selon vous, la mécanique de gameplay principale du jeu, et reproduisez~~

~~là sous le moteur de jeu de votre choix (vous pouvez faire cette mécanique sur un projet en~~

~~2D ou 3D).~~

**Mécanique forcée : refaire la gestion soit de quatre liquides, soit de quatre gaz.**

Décrivez, étape par étape, comment vous avez mis en place cette mécanique.

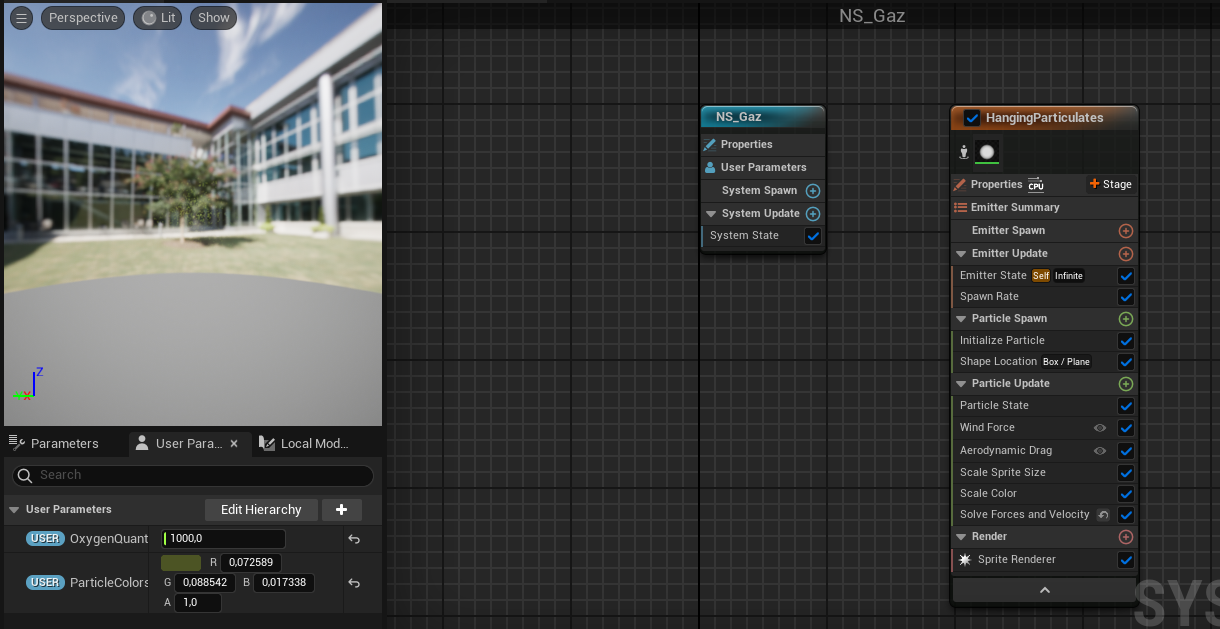
~~Expliquez en quoi cette mécanique est primordiale tout au long du jeu.~~

~~Pourquoi cette mécanique vous a-t-elle plu ?~~

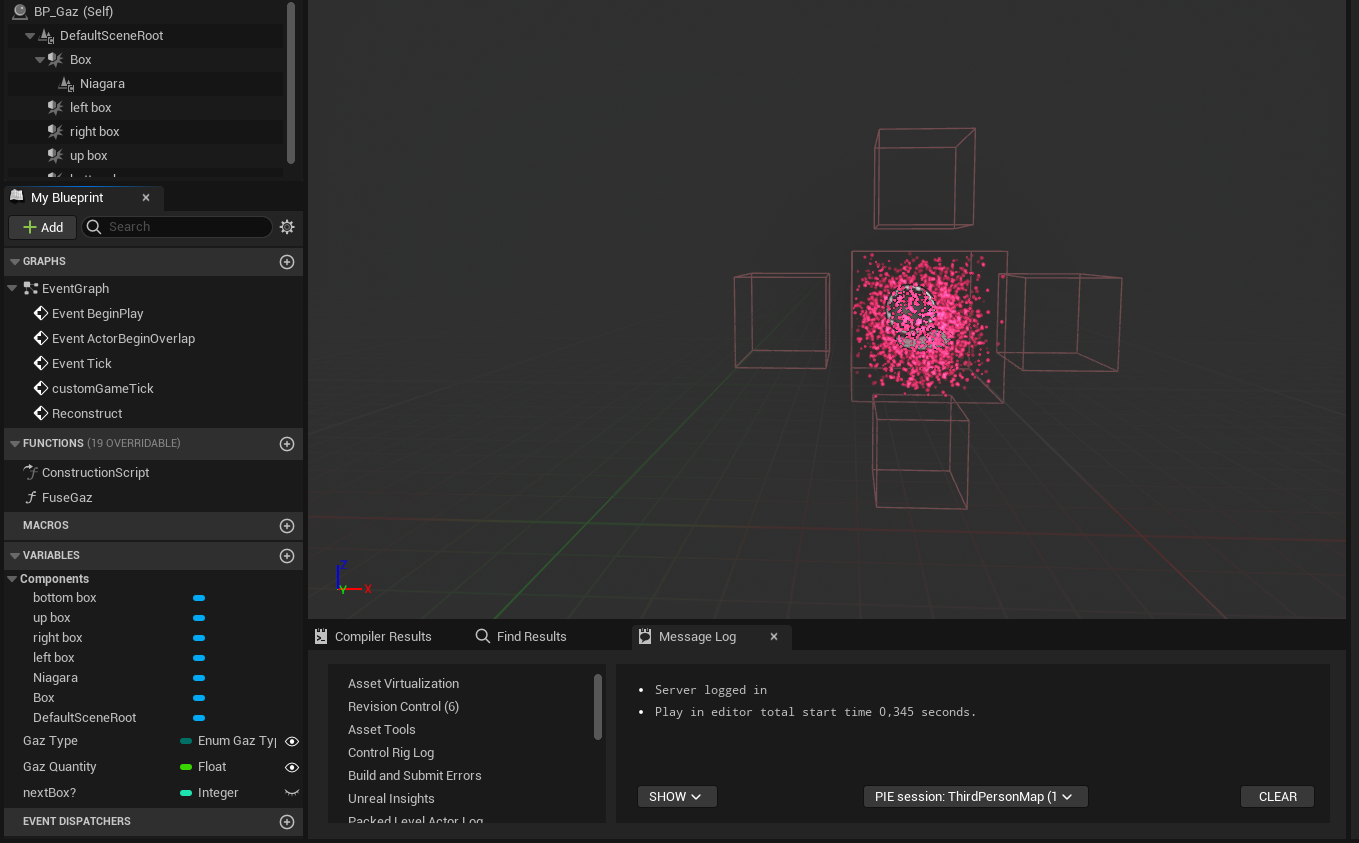
Tout d’abord, je vais définir comment je vais refaire la gestion des gaz : Je vais créer un Blueprint qui sera un cube, contenant comme information le type de gaz, et la quantité de gaz à l’intérieur. Chaque tick de jeu (1/5 seconde dans ce cas la), le cube de gaz regardera les cubes qui sont à côté de lui ; si il s’agit d’un gaz différent, ils changeront de place selon lequel est le plus “lourd” (défini uniquement selon son type)

Il y aura quatre types de gaz : oxygène, oxygène pollué, hydrogène, et dioxyde de carbone. Il y aura un cinquième type, pas vraiment un gaz, le “vide”, qui prendra du gaz autour de lui si il y en a et se transformera instantanément.

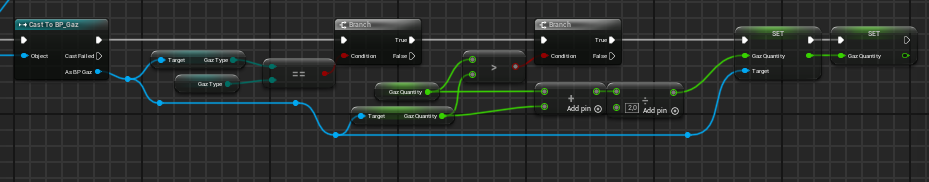
Pour commencer, j’ai décidé de créer une représentation simple visuelle des gaz, avec deux variables : une pour représenter la quantité (à quel point il y a de gaz dans le cube en question) et un autre pour représenter le type de gaz (juste la couleur).



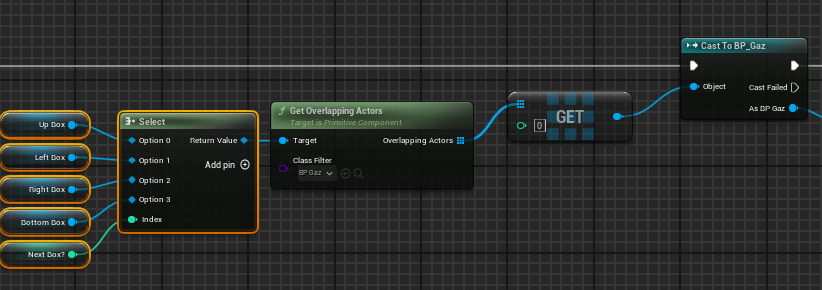
Ensuite, j’ai créé un Blueprint BP\_Gaz, contenant 5 box collisions et un niagara system. Le NS sert à afficher le gaz, une des box collision sert à délimiter la case, et les quatre autres servent à détecter les boîtes alentour. En outre, le BP contient 3 valeurs : le type de gaz, la quantité de gaz, et quelle est la prochaine boîte à vérifier.



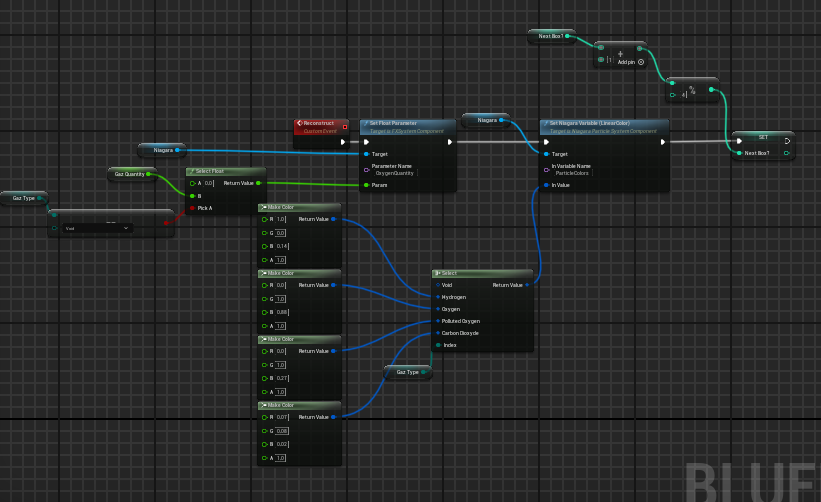
Ensuite, il faut commencer par le plus simple : La distribution du gaz entre cases contenant le même gaz. Pour simplifier, je fais en sorte que la transmission ne se fait que si c’est la case qui est analysée qui a moins de gaz ; cela me permet d’éviter les problèmes stupides, comme une double transmission au même moment.



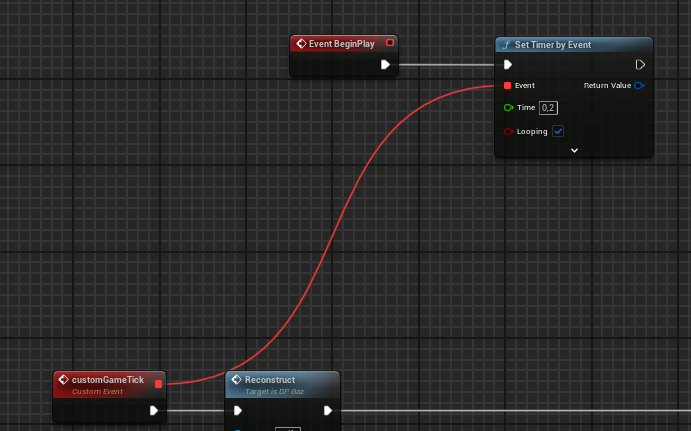
On ajoute un select au début, afin d’alterner entre chaques cases analysées :



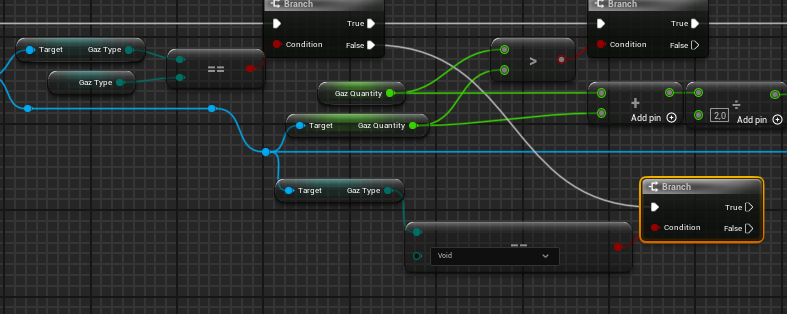
Et enfin, on s’assure de mettre à jour l’apparence de la case et de changer la variable qui sélectionne quelle box collision est analysée :



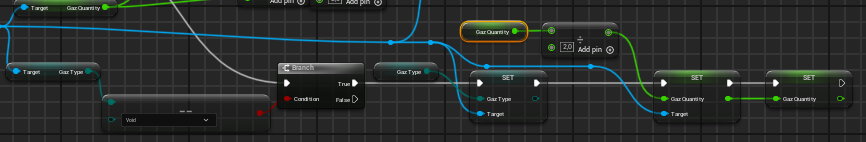
Et on s’assure que le tout se joue en boucle, comme on l’a précisé auparavant, toutes les 0.2 secondes :



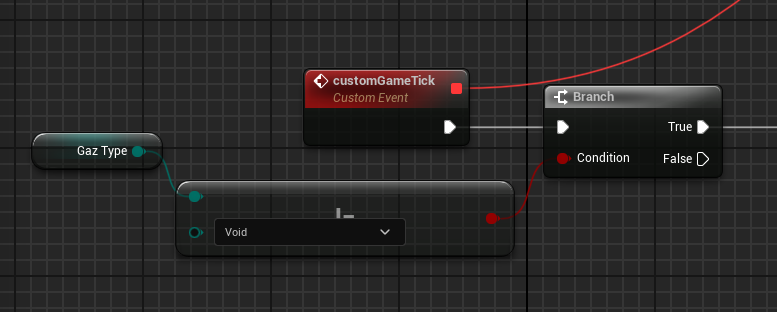
Maintenant que le gaz se répand sur toute la surface disponible, il est temps d’assurer l’interaction entre différents gaz. Pour commencer, nous assurons l’interaction avec le Vide :



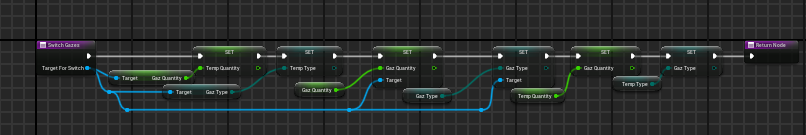
L’interaction est assez simple, simpliste même. La case donnera la moitié de sa valeur au Vide, et convertira le vide en son type de gaz.



Enfin, pour assurer qu’il n’y ait pas de problèmes, on empêche le vide d’effectuer la moindre action :

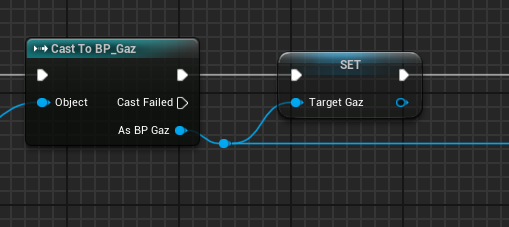


Enfin, il faut assurer le déplacement de plusieurs gaz dans le même environnement. Tout d’abord, nous allons coder un simple switch entre deux cases, en fonction :

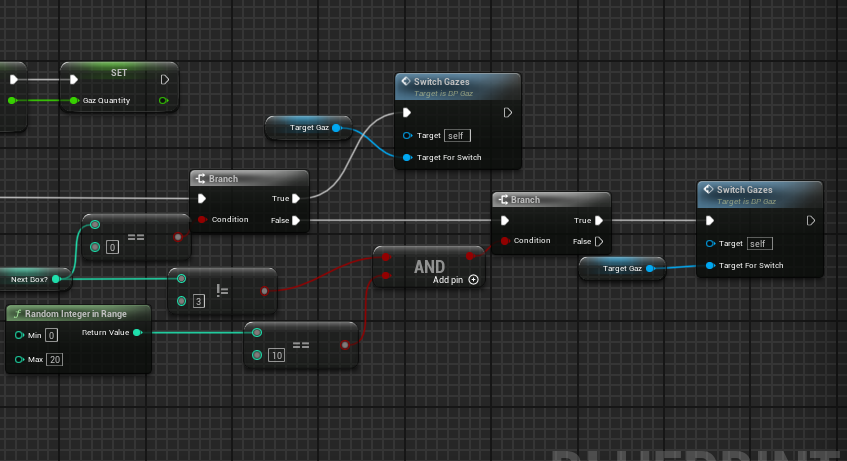


Ensuite, il nous faut assurer les interactions. Elles sont assez simples, et nous fonctionnerons selon deux règles simples : Seul le gaz le plus léger peut activer le switch entre deux cases, et le gaz ira forcément plus haut, mais tentera des fois (10% de chance) d’aller sur les côtés. La seule exception sera les deux oxygènes, qui, ayant la même masse, tentera toujours de switcher ; mais seulement avec une chance de 5%.

**Hydrogene :**



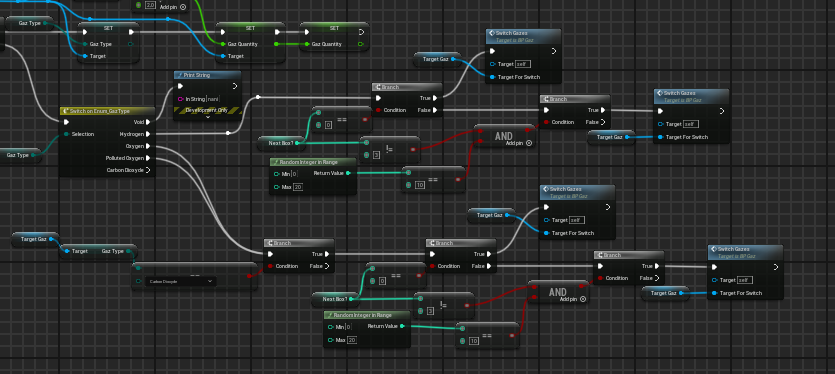
On ajoute une variable “target gaz” pour simplifier le code et faire moins de spaghettis



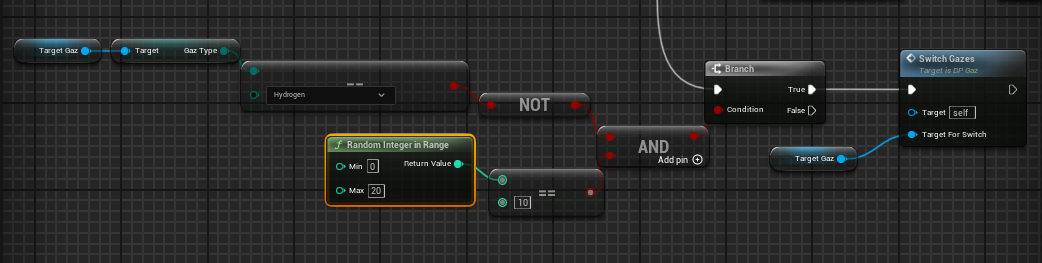
Et on met en place le switch de l’hydrogène : si il vise en haut, il switche forcément, (car il vise forcément autre chose que de l’hydrogène), sinon, si il ne vise pas en bas, il a 5% de chance de switcher à gauche ou à droite.

**Oxygène / Oxygène polluée :**

Tout d’abord, on met un check : si l’oxygène ou l’oxygène polluée rencontre du dioxyde de carbone, il fait la même manœuvre que l’hydrogène.



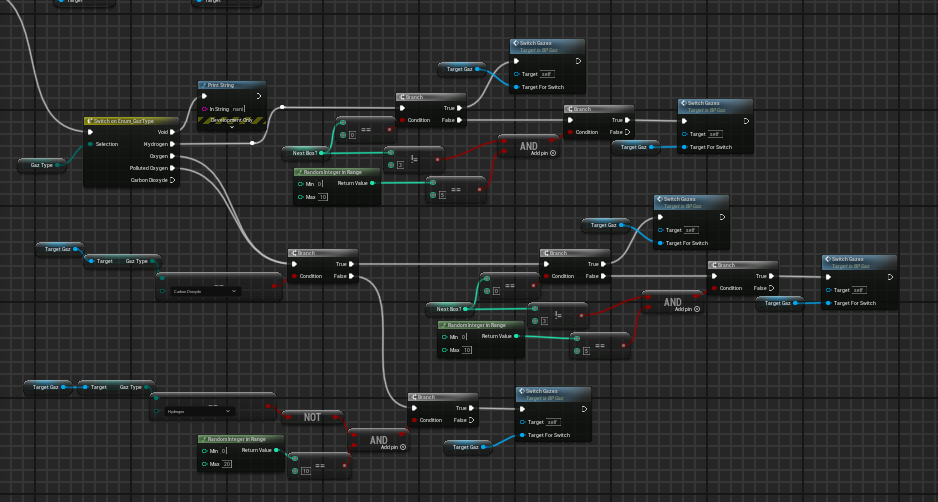
Si ce n’est pas du dioxyde de carbone, on fait une vérification pour voir si c’est de l’hydrogène ; si ça n’en est pas, on laisse la chance décider de si les deux types d’oxygène switcheront.



Et enfin, le plus dur pour la fin :

**Dioxyde de Carbone :**

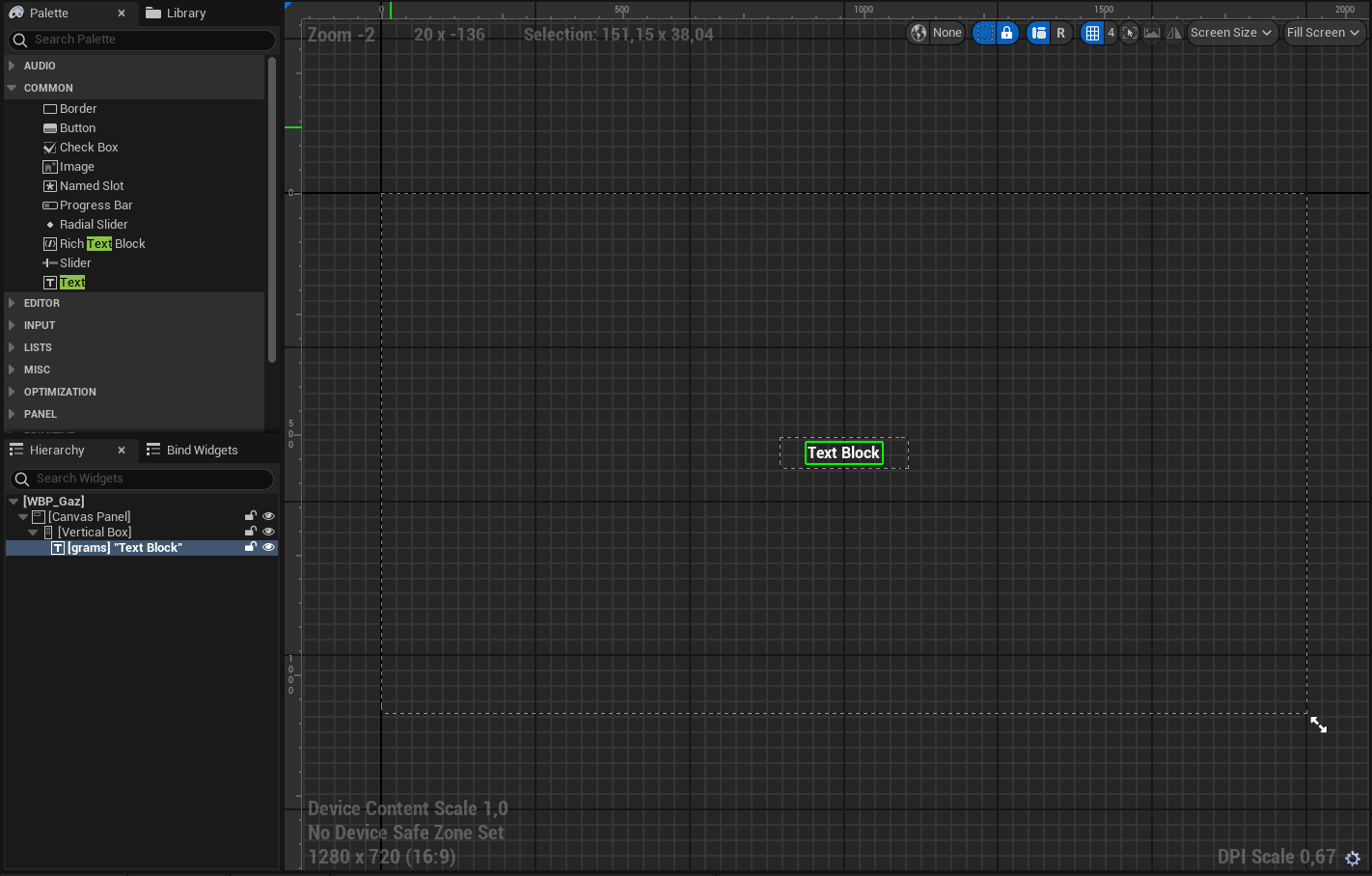
Le dioxyde de carbone est le plus lourd gaz, et donc il ne fait rien.

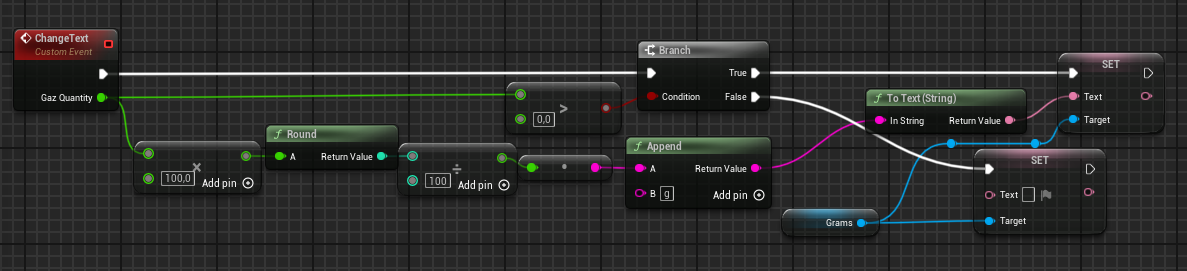


**Enfin, quelques améliorations rapides :**

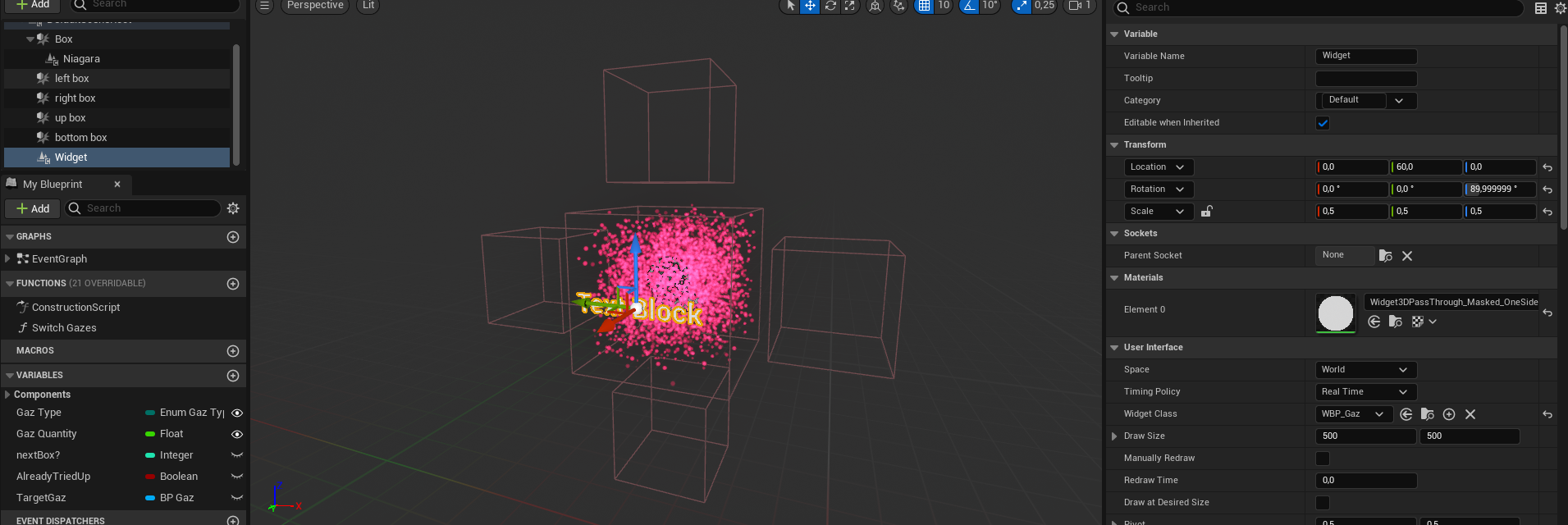
Tout d’abord, nous pouvons ajouter un widget pour afficher la quantité de gaz dans chaque cases.

On crée un widget rapide et simple :

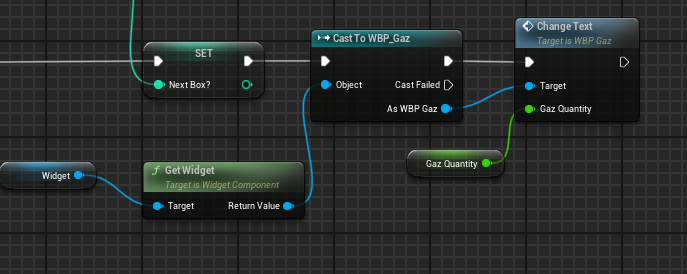


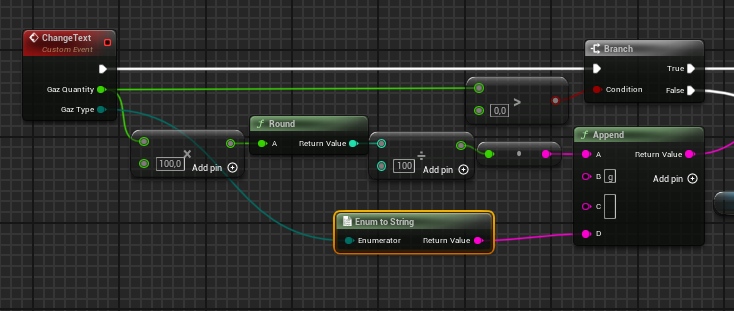


Puis on l’ajoute au bp\_gaz :



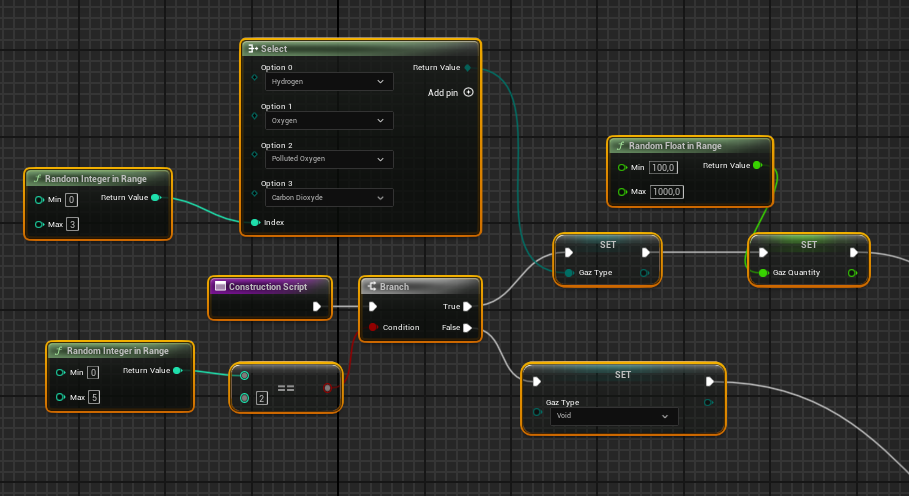
Et enfin, on lance le custom event changetext à chaque fois qu’on reconstruit la case :



Pour l’améliorer plus loin, nous pouvons ajouter le type de gaz : 

**Un mode aléatoire** :

Nous pouvons aussi ajouter une randomisation des cases au début de chaque lancements du programme, plutôt que de placer les cases nous-même.



Chaque case aura une chance sur 6 de ne pas être du void ; et ensuite, une chance égale d’être l’un des quatre gaz. La quantité est aussi randomisée, allant de 100 à 1000g.