## Création Blocs

Pour la création des blocs, nous avions comme contrainte qu’un bloc soit composé de plusieurs autres sous-blocs (bloc élémentaire). Dans un premier temps, nous avons donc générer des cubes composer de plusieurs sous-cube.

Par la suite, nous voulions que nos blocs possèdent des formes aléatoires. Pour cela, nous n’avons plus qu’a enlever aléatoirement des parties du cube pour avoir des formes aléatoire, pour mettre en place ce fonctionnement on utilisera la méthode random() afin de déterminer si un sous-bloc doit être placer ou non.

De plus, on pourra jouer sur la probabilité des placements. Par exemple, un bloc possède plus de chance d’être rond soit il faut une plus grande probabilité lors du placement des sous-cube du centre de notre cube.

Enfin, on regroupera nos objets dans le graphe de scène de la manière suivante :   
rootNode -> sol  
 -> nœud camera  
 -> personnage (physique)  
 -> nœud des blocs (obj\_pierre) -> nœud d’un bloc -> géométries du bloc

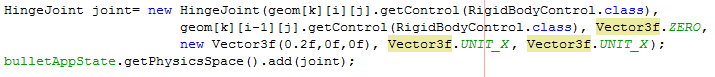
## Gestion Collision

Pour la gestion des collisions, Jmonkey utilise le moteur jbullet. Ce moteur implémente déjà tout un système de physique avec gravité et autres contraintes.

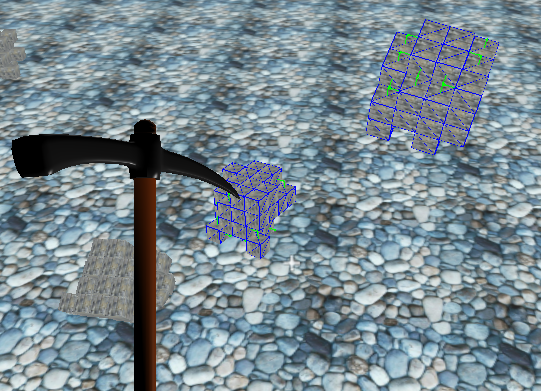
Dans notre cas, nous voulions que les blocs est un comportement dynamique (le bloc subit la gravité et les collisions). Il fallait donc appliquer la physique à notre cube. Cependant, nous avons fait face à un problème au niveau du moteur physique. Effectivement, si on applique la gravité sur le nœud celui-ci génère un volume englobant qui ne correspond pas au volume réel. De plus, le centre de gravité de notre cube est en générale mal placé lors de la mise en place de la physique sur le nœud. Au final, le comportement de nos blocs n’était pas réaliste : par exemple un cube en forme de triangle ne tomber pas sur le coté.

Nous avions pour solution d’appliquer la physique aux sous-blocs, puis dans un second temps de les liée par une contrainte. Ce système fonctionne, mais cela induit un grand nombre d’opération (vérification des contraintes de liaison + collisions entres les sous-blocs + gestion gravité sur tous les objets) ce qui à pour effet de ralentir de façon importante le fonctionnement de l’application.

Pour cela, il suffit donc d’appliquer la physique sur nos sous-blocs : 

On lui attribue donc un poids « 0.1f » et on ajoute le contrôle à l’espace physique. Par la suite, nous créons les liens entre les blocs pour que ceux-ci reste ensemble lors du mouvement du bloc. Jmonkey fournit les contraintes de jointure selon les axes appelé HingeJoint, ceux-ci permette de mettre des contrainte de distance entre des géométries en laissant un axe de liberté de rotation. On définie donc pour chaque sous-cube 3 contrainte de liaisons avec les blocs adjacent : 

Ces jonctions prennent en paramètre le type de contrôle des deux géométries, l’emplacement dans le repère local du premier objet et l’emplacement du deuxième objet dans le repère locale du premier et enfin l’axe de contrainte.

  
On obtient ainsi des cubes avec des sous-blocs liés

Ainsi nous pouvons créer des blocs sans problème, le seul souci restant étant que plus il y a de cube et plus le rendu est long à effectué. De plus, si nous voulons encore améliorer notre application, il est possible de mettre l’espace physique (la physique) dans un thread séparé du rendue en ajoutant la configuration suivante : 

Au final, nous avons retenue cette solution et nous l’avons amélioré en faisant en sorte de passer les blocs qui ne bouge plus (tomber au sol et ne bouge plus) en statique c'est-à-dire que nous enlevons la physique sur les blocs qui ne bouge plus afin de diminuer le nombre d’objet dynamique. Cela à pour effet de diminuer le nombre de calcul de l’espace physique.

Pour cela, on stock la position de notre cube à différent instant et on calcul la distance entre ces deux instants, si elle est nul alors on peut passer le bloc en statique. On va donc vérifier chaque nœud qui sont dynamique afin de voir si il faut les placer en statique :



On obtient ainsi des blocs statiques, ils sont fixés au sol. Au besoin, il est toujours possible de les repasser en dynamique si l’on souhaite appliquer la physique de nouveau.

Pour conclure, cette technique n’est peut être pas la meilleur et induit des latences dans le jeu due au grand nombre de face à représenter et à la dynamique de l’application. La technique la plus optimiser serait de gérer les blocs selon les arrêtes du mesh afin d’avoir le moins de face possible et de collisions mais cela est plus long et est plus compliquer à mettre en œuvre.