

École Polytechnique de l’Université de Tours

64, Avenue Jean Portalis

37200 TOURS, FRANCE

Tél. +33 (0)2 47 36 14 14

[www.polytech.univ-tours.fr](http://www.polytech.univ-tours.fr)

**Département Informatique**

Rapport de projet de Réalité virtuelle

**BlockBuster**

Etudiants :  
VANDENHOVE Pierre  
HAVARD Thibault

Encadrant :   
AUPETIT Sébastien

Sommaire

[Introduction 3](#_Toc355012333)

[Déroulement du projet 3](#_Toc355012334)

[Présentation du projet 3](#_Toc355012335)

[Découpage 3](#_Toc355012336)

[Déroulement des tâches 3](#_Toc355012337)

[Modélisation, Animation, Picking 4](#_Toc355012338)

[Modélisation d’une pioche et intégration 4](#_Toc355012339)

[Picking (Sculpture) 7](#_Toc355012340)

[Physique et Collision 8](#_Toc355012341)

[Création Blocs 8](#_Toc355012342)

[Gestion Collision 8](#_Toc355012343)

[Conclusion 11](#_Toc355012344)

# Introduction

Le projet BlockBuster s’inscrit dans le cadre des projets d’option de cinquième année, ici l’option Réalité Virtuelle, il nous permet de mettre en place les notions vues en cours et de développer une branche de notre métier peu ou pas aborder en cours : les jeux vidéos.

Ce rapport va développer, en première partie, le sujet ainsi que les choix effectués pour le mener à bien (découpage des tâches, choix des outils…). Ensuite dans les parties suivantes nous parlerons des différentes fonctions implémentées dans notre projet : leurs mises en place et leur fonctionnement. Enfin, nous concluerons sur les difficultés rencontrées et les possibles améliorations à apporter.

# Déroulement du projet

## Présentation du projet

BlockBuster est un projet de Mr Aupetit, l’objectif principal est de créer un jeu type Minecraft. Il s’agit de créer un monde de blocs ou ceux-ci sont générer aléatoirement par le moteur du jeu à la demande du joueur. C'est-à-dire que le jeu choisi la taille, la fragilité, poids… du bloc demandé par le joueur. Celui-ci peut, ensuite, s’amuser à sculpter ce bloc afin de lui donner la forme qu’il souhaite. Ces blocs peuvent être bougés et placés où le joueur le veut. Par contre chaque bloc subit la gravité et peut supporter un poids maximum ; si celui-ci est dépassé le bloc disparaît.

Ce jeu doit être développé en Java donc nous avons décidé d’utiliser Jmonkey et son SDK pour travailler. De plus le découpage en tâches nous permet d’utiliser un système de versionnage : Git. Nous avons donc créé un dépôt sur Github qui contient notre projet.

## Découpage

Suite à la présentation du projet par Mr Aupetit nous avons décidé de découper le projet en 2 parties :

* Collision et Gestion des blocs
* Sculpture

Ce découpage est surtout dû à l’avancement de chacun d’entre nous dans ses autres projets (surtout le PFE). En effet, Thibault ayant une bonne avance dans le sien, il a été décidé qu’il s’occuperait de la gestion des blocs et de la gestion des collisions. La sculpture des blocs étant un peu plus simple Pierre a décidé de s’en occuper.

## Déroulement des tâches

Comme dit dans le paragraphe précédent le développement s’est retrouvé en concurrence avec l’avancement de nos PFE respectifs. Il a donc fallu négocier cela, nous avons donc décidé de « négliger » le déplacement des blocs pour nous concentrer sur les collisions, la génération aléatoire de bloc et la sculpture.

Les collisions ont eu un peu de retard du à un manque d’information disponible sur Internet et donc Thibault a dû tâtonner pour trouver une solution viable. Nous verrons plus loin sa solution. La sculpture et la gestion aléatoire des blocs ce sont mieux passés.

# Modélisation, Animation, Picking

## Modélisation d’une pioche et intégration

Le jeu se rapprochant de Minecraft nous avons voulu modéliser une pioche afin de permettre au joueur de sculpter ses blocs. Pour cela nous avons utilisé un logiciel de modélisation : Blender.

Blender est un logiciel de modélisation, d’animation et de rendu 3D libre. Il permet de dessiner assez vite des formes simples et, ensuite de les animés. Sa communauté est très importante il est donc facile de trouver des tutoriels pour apprendre à l’utiliser ou plus généralement des modèles d’objet déjà fait.

Pour la pioche, suite à notre expérience de projet collectif, nous avons décidé de trouver un modèle tout fait et de l’animer par la suite.

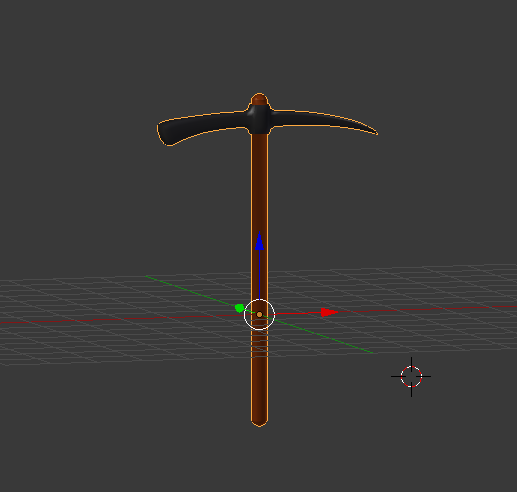


Figure : Pioche

Une fois le modèle trouvé et un peu transformé (ajout des deux matériaux), il a fallu mettre en place l’animation d’un coup de pioche, c'est-à-dire une rotation de la pioche selon l’axe y de 90°. Pour cela nous avons ajouté une armature à notre pioche composée d’un seul Bone (car une pioche ne se déforme pas il n’y a donc pas besoin de plus de Bone).

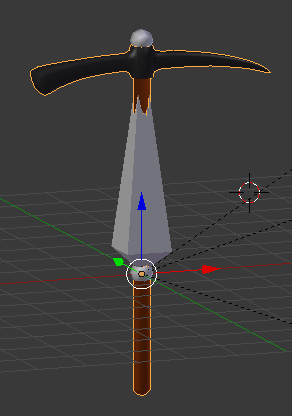


Figure : Pioche avec Bone

Une fois le bone ajouté et lié à la pioche il suffit d’ajouter les différentes phases de l’animation pour que Blender calcule les phases manquantes.

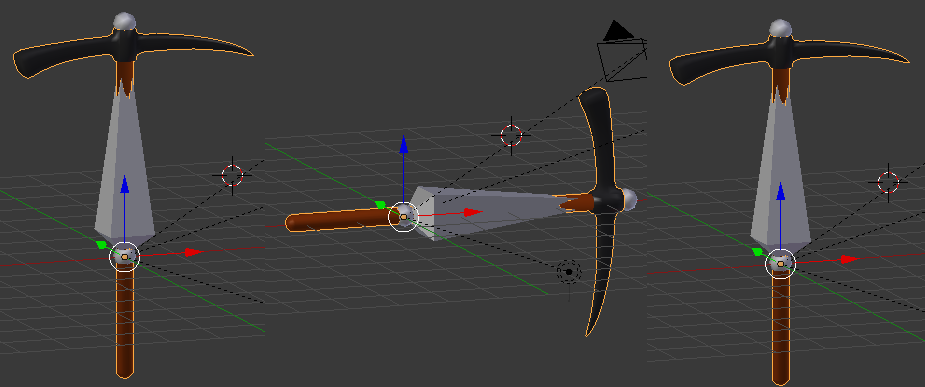
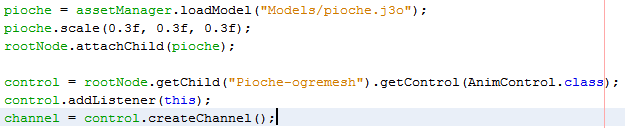


Figure : Les 3 phases de l'animation

Une fois cela terminé il faut exporter le modèle au format ogre pour permettre à Jmonkey de le convertir dans un format qu’il connait (j3o).

L’intégration dans notre jeu et l’animation se fait grâce à ses lignes de code :



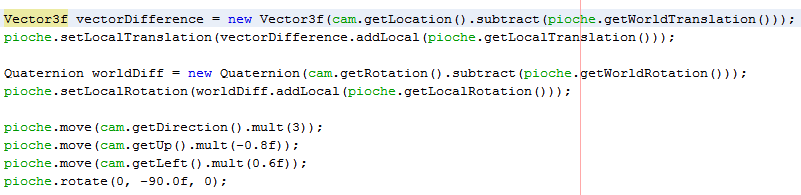
Le scale de la pioche permet de la mettre à l’échelle du jeu et ensuite on l’attache au nœud racine afin de la faire apparaître dans la scène.

Le control et le channel sont les deux variables utilisées pour lancer l’animation quand le joueur clique.

Ensuite il a fallu connecter la pioche à notre joueur, c'est-à-dire que celle-ci devait le suivre tout le temps. Cela n’a pas été facile car notre joueur n’est pas représenté par un nœud simple c’est un PhysicsCharacter. Il était donc impossible de lui attaché un objet. Pour palier à ce problème nous avons lié la position physique du joueur et la position de la pioche grâce à ce code :

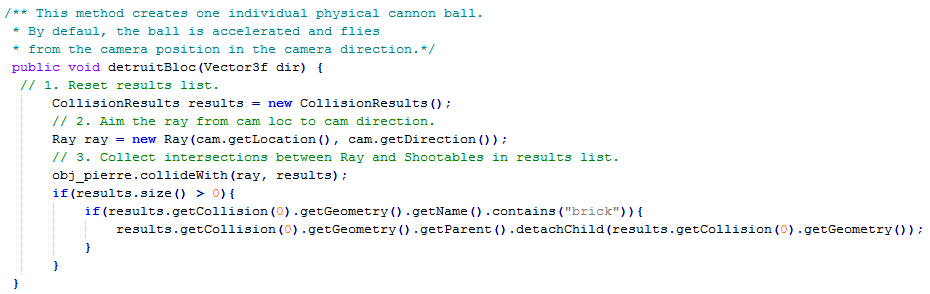


Mais cela a entrainé un autre problème : la pioche suivait bien le joueur mais pas les mouvements de rotation de la caméra. Il a donc fallu calculer les mouvements de rotation de la caméra et les retranscrire à la pioche :



## Picking (Sculpture)

Le picking avec Jmonkey est simple il suffit de ces lignes de codes :



Le principe est simple, un rayon est envoyé dans la direction du clic et le premier objet « brick » rencontré est détruit. Cela permet de sculpter nos blocs.

# Physique et Collision

## 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 qu’à enlever 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, pour qu’un bloc possède plus de chance d’être rond il faut une plus grande probabilité de placement des sous-cubes au 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

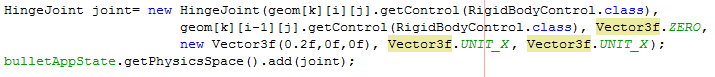
Pour la gestion des collisions, Jmonkey utilise le moteur jbullet. Ce moteur implémente déjà tout un système de physique avec gravité et autres contraintes.

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 va pas tomber sur le coté.

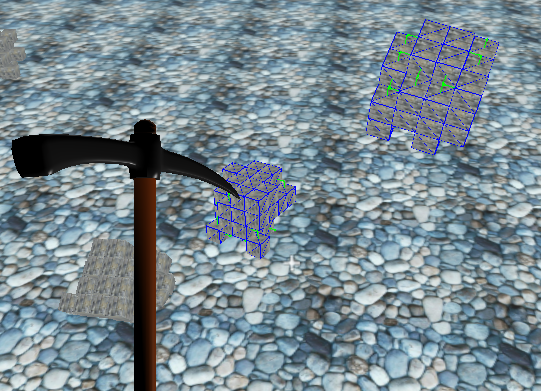
Nous avions pour solution d’appliquer la physique aux sous-blocs, puis dans un second temps de les liée par une contrainte. Ce système fonctionne, mais cela induit un grand nombre d’opération (vérification des contraintes de liaison + collisions entre les sous-blocs + gestion gravité sur tous les objets)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

Pour la physique sur nos sous-blocs :



On attribue 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 contraintes de liaisons avec les blocs adjacents : 

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 est dynamique afin de voir s’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.

# Conclusion

Pour conclure, cette technique n’est peut être pas la meilleur et induit des latences dans le jeu dues au grand nombre de faces à représenter et à la dynamique de l’application. La technique la plus optimisée 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 compliqué à mettre en œuvre.

Cependant, ce projet nous à permis de découvrir Jmonkey et de voir quels sont les moyens pour mettre en place une application de type minecraft. De plus, nous avons un aperçue des contraintes que ce type d’application possède notamment au niveau de la représentation d’un grand nombre de face.

Enfin, ce projet nous a permis malgré une perte de temps au début due à la prise en main de JmonkeyEngine de renforcer nos connaissances en matière de représentation virtuelle.