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

Département Informatique
5e année
2010 - 2011

Projet de réalité virtuelle

Construction en Kapla

Encadrant
Sebastien Aupetit
aupetit@univ-tours.fr
Emmanuel Néron
emmanuel.neron@univ-tours.fr

Étudiants
Guillaume Smaha
guillaume.smaha@etu.univ-tours.fr
Pierre Vittet
pierre.vittet@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2010 - 2011

Version du 8 mai 2011

Table des matières

Table des figures

Liste des tableaux

Introduction

1.1 Présentation du projet

Ce projet se place dans le cadre de l'option Réalité Virtuelle pour les élèves de 5^{ième} année de l'école Polytech'Tours.

Ce projet a été proposé par Emmanuel Néron afin de répondre à la demande de concrétiser le problème d'ordonnancement pour un cas particulier du jeu de "Kapla". Le problème d'ordonnancement est, pour un nombre donné de briquettes, de trouver la meilleure architecture permettant d'avoir une structure stable et d'être éloignée le plus possible du support de départ.

1.2 Objectifs

L'objectif du projet est de proposer un environnement 3D dans lequel un utilisateur peut positionner un ensemble de briquettes afin de créer des structures. A partir d'une table, l'utilisateur doit essayer d'obtenir une structure qui ait la flèche la plus grande possible (c'est à dire qui soit éloignée autant que possible du bord de la table). Bien sûr, chaque briquette est soumise à la force de gravité rendant plus complexe l'élaboration de structures stables. Ce document décrit les technologies utilisées ainsi que le fonctionnement du logiciel.

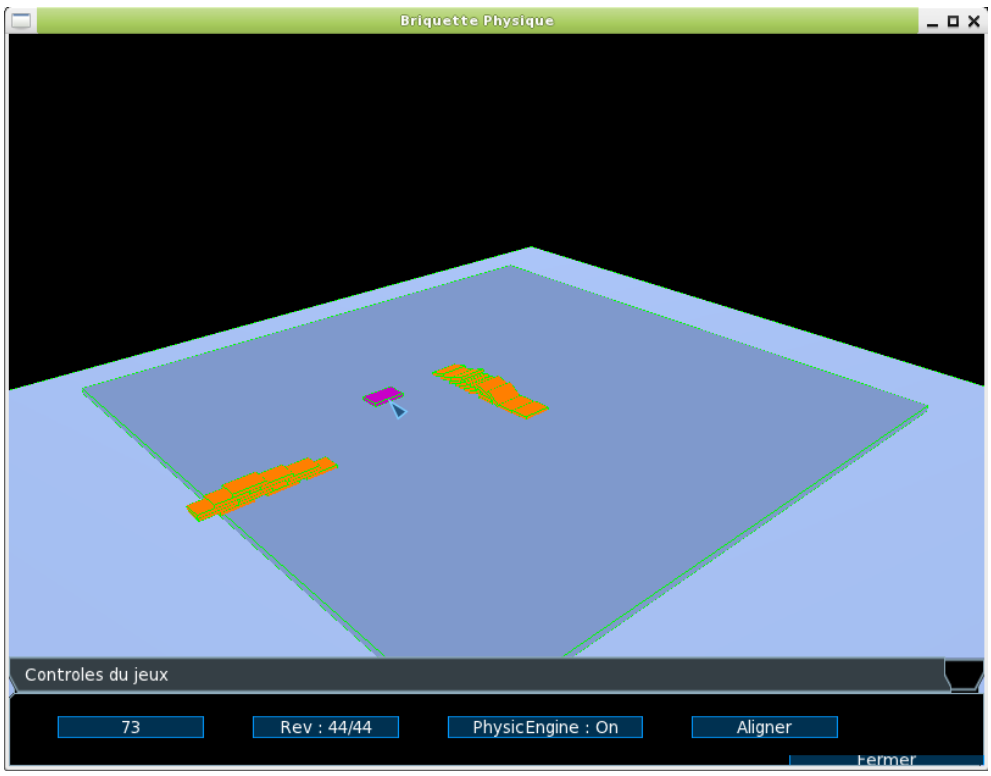

FIGURE 1.1 – Un exemple typique de l'application en cours de fonctionnment

1.3 Détail des attentes

Après avoir discuté avec notre encadrant, nous nous sommes donnés les objectifs suivants :

- Un menu permettant de gérer différents niveaux de difficulté. Dans un premier temps la seul différence viendra du nombre de briquettes disponibles, mais il est également possible de travailler sur le poids ou la taille des briquettes.
- Saisie des objets et déplacement des briquettes via la souris
- Possibilité de revenir à un état précédant ou sucesseur du jeu : Après chaque placement de briquette, l'état du jeu est mémorisé et on permet d'y revenir.
- Les briquettes ne doivent pas pouvoir être posées n'importe où mais seulement le long d'un axe. De même, leur orientation est vérouillée, l'utilisateur ne pouvant pas placer ces briquettes en travers. Cela permet de réduire la complexité du problème lorsque l'on recherche des solutions efficaces au problème. Le jeu n'est donc en réalité pas un jeu 3D mais 2D, seule la visualisation est en 3D.

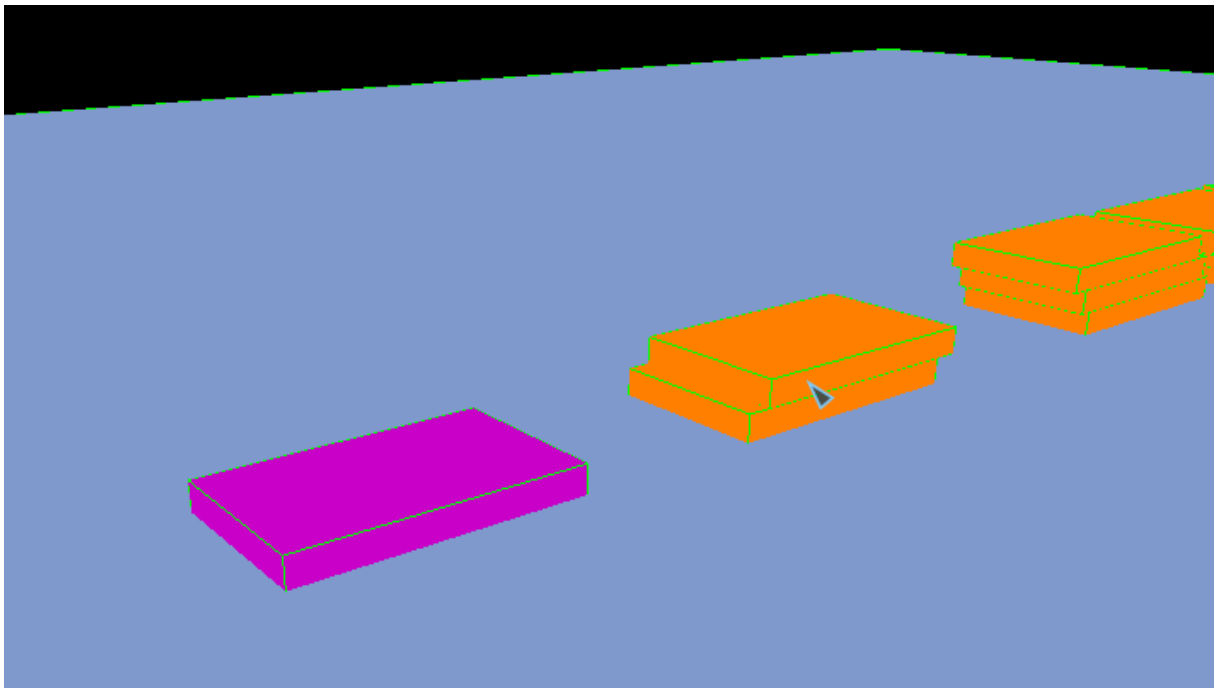

FIGURE 1.2 – Les briquettes sont toutes orientées de la même façon suivant le même axe.

Choix techniques

2.1 Contexte

L'encadrant de projet nous a laissé la possibilité de choisir les technologies et les outils que l'on souhaitait utiliser pour le projet. Nos choix techniques ont été pris de manière à offrir un logiciel facilement utilisable sur différentes plateformes mais également en vue d'aller aussi loin que possible dans le projet en considérant le temps imparti. Nous avons fait le choix de réutiliser autant que possible les outils que nous connaissions déjà. Cela nous a permis d'avoir une vue d'ensemble et une maîtrise que nous n'aurions pas eu autrement. Et nous avons pu améliorer nos idées mises en place pour le projet collectif de Réalité Virtuelle[8] .

2.2 Liste des outils

2.2.1 Librairie utilisée

Afin de faciliter la mise en œuvre, il est nécessaire d'utiliser des librairies. Nous avons donc utilisé les librairies suivantes.

- Ogre[2] : Moteur 3D open source (http ://www.ogre3d.org/) supportant aussi bien OpenGL que Direct3D.
- CEGUI[3] : Crazy Eddie's GUI System (http ://www.cegui.org.uk) fournissant des outils de créations de menus et de fenêtres dans un environnement 3D.
- Bullet[4] : Librairie de simulation de la physique http ://bulletphysics.org/

Notre expérience dans l'utilisation de ces librairies ayant été facilitée par leur utilisation dans le projet collectif[8] , il avons grandement amélioré la structure du programme.

2.2.2 Gestionnaire de version

Afin de sauvegarder le projet et les données liées à celui-ci, nous avons choisi d'utiliser le gestionnaire de version Git. Cela nous à été particulièrement utile, en particulier, nous avons apprécié la gestion fine des conflits. Avec Git, il n'est pas nécessaire d'utiliser un serveur de dépôt mais pour une question de sauvegarde des données, nous avons utilisé le site gitorious.org qui propose gratuitement des dépôts. Le notre est stocké à l'adresse suivante : https://gitorious.org/briquettephysique/briquettephysique

2.2.3 Documentation technique

Il est intéressant de concevoir une documentation technique, ce qui faciliterait son étude par une personne tierce dans le cas d'une reprise du projet.

Nous avons utilisé pour ce faire le logiciel Doxygen[7] . Ce logiciel permet à partir de commentaires spécifiques dans les fichiers d'en-tête de générer la documentation au format html et pdf. Le format html est intéressant puisqu'il permet d'être facilement diffusé sur Internet.

L'avantage majeur à utiliser Doxygen est qu'il permet la génération de graphique :

- Graphe de collaboration

- Graphe des dépendances par inclusion
- Graphe d'inclusion directe et indirecte
- Graphe d'appels d'une fonction

La légende est très simple à comprendre :

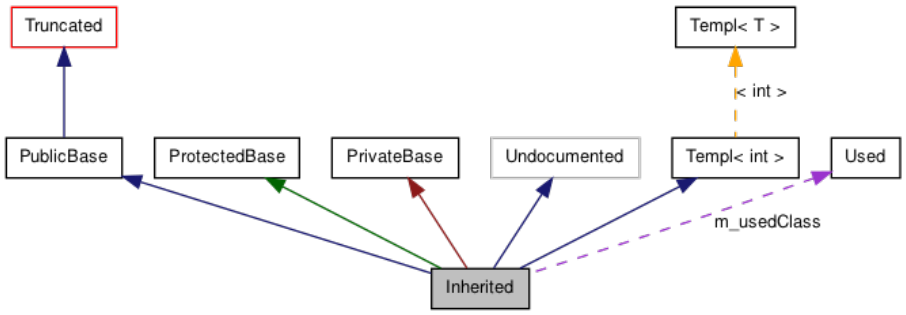

FIGURE 2.1 – Légende du graphe

Le code permettant de générer ce graphe :

```
/*! Invisible class because of truncation */
class Invisible { };

/*! Truncated class, inheritance relation is hidden */
class Truncated : public Invisible { };

/*! Class not documented with doxygen comments */
class Undocumented { };

/*! Class that is inherited using public inheritance */
class PublicBase : public Truncated { };

/*! A template class */
template<class T> class Templ { };

/*! Class that is inherited using protected inheritance */
class ProtectedBase { };

/*! Class that is inherited using private inheritance */
class PrivateBase { };

/*! Class that is used by the Inherited class */
class Used { };

/*! Super class that inherits a number of other classes */
class Inherited : public PublicBase,
                  protected ProtectedBase,
                  private PrivateBase,
                  public Undocumented,
                  public Templ<int>
{
  private:
    Used *m_usedClass;
};
```

FIGURE 2.2 – Le code permettant de générer le graphe de légende

Pour présenter les différents graphes, nous utiliserons la classe la moins utilisée dans le programme (GestViewport) car les graphes deviennent rapidement très grands et ils ne seraient pas correctement visibles dans le rapport.

Graphe de collaboration

Le graphe de collaboration de la classe GestViewport.

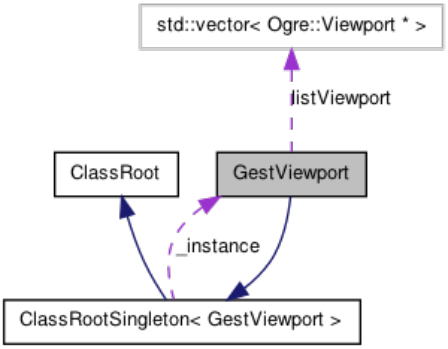

FIGURE 2.3 – Graphe de collaboration de la classe GestViewport

Graphe des dépendances par inclusion

Le graphe des dépendances par inclusion de la classe GestViewport.

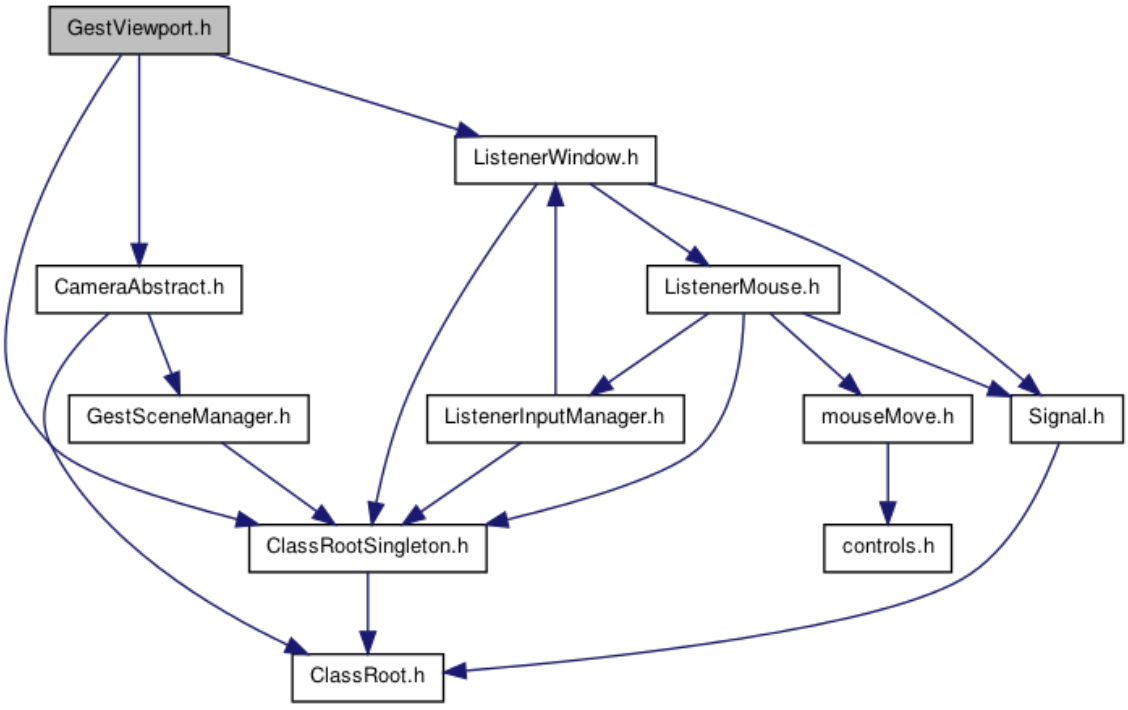

FIGURE 2.4 – Graphe des dépendances par inclusion de la classe GestViewport

Graphe d'inclusion directe et indirecte

Le graphe d'inclusion directe et indirecte de la classe GestViewport.

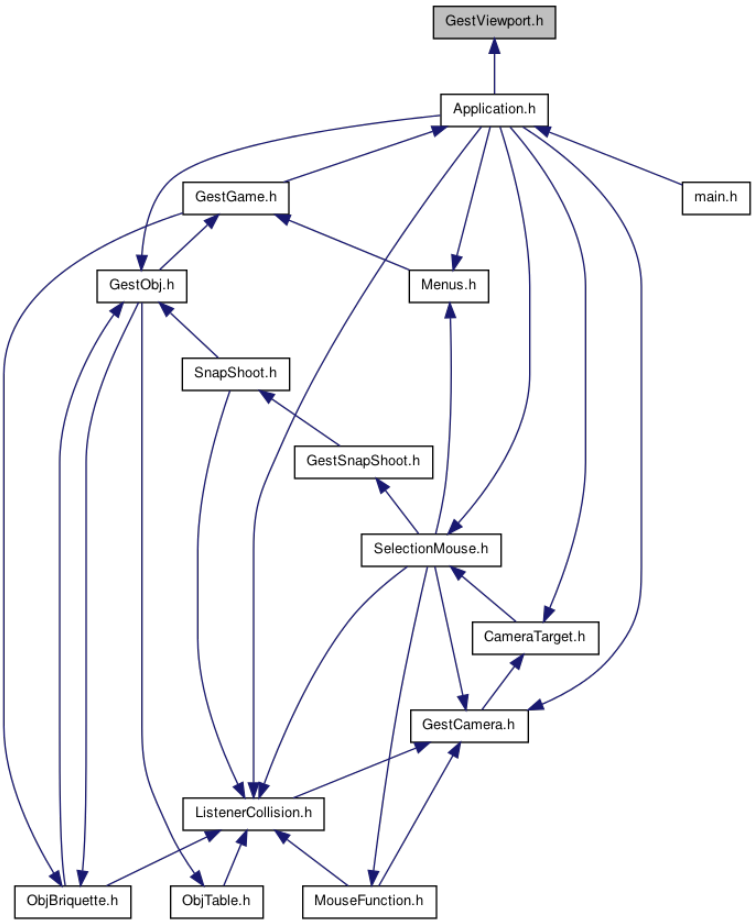

FIGURE 2.5 – Graphe d'inclusion direct et indirect de la classe GestViewport

Graphe d'appels d'une fonction

Le graphe d'appels de la méthode addViewport de la classe GestViewport.

FIGURE 2.6 – Graphe d'appels de la méthode addViewport de la classe GestViewport

Réalisation

3.1 Gestion des évènements claviers/souris

Nous utilisons une classe PlayerControl qui redirige l'ensemble des évènements claviers et souris en des évènements logiques. C'est cette classe qui permet de lier les touches à des actions, permettant d'abstraire la configuration des touches par la suite.

3.2 Gestion de la caméra

Dans ce projet, il n'y a pas besoin de fournir de multiples caméras à l'utilisateur. Nous avons convenu que la caméra la plus adaptée serait une caméra capable de pivoter autour de la table. La caméra est toujours orientée vers un point central, qui au départ est le centre de la table, cela permet d'avoir une orientation correcte, de plus l'axe de lacet de la caméra est fixé sur l'axe Z de façon à ce que la table soit toujours vue droite. Les rotations se font à l'aide de la souris, il est également possible de zoomer à l'aide des touches du clavier ou de la touche centrale de la souris (molette). Au départ, nous avons permis avec les touches directionnelles du clavier de déplacer le point central qui sert de repère à la caméra, en pratique ce n'est pas très utile car peu maîtrisable pour l'utilisateur. Par contre, un système beaucoup plus intéresant consiste à déplacer ce point central à la position d'une briquette en effectuant un clic droit sur celle-ci. La caméra se tourne alors vers la position de la briquette et les rotations se font autour de celle-ci.

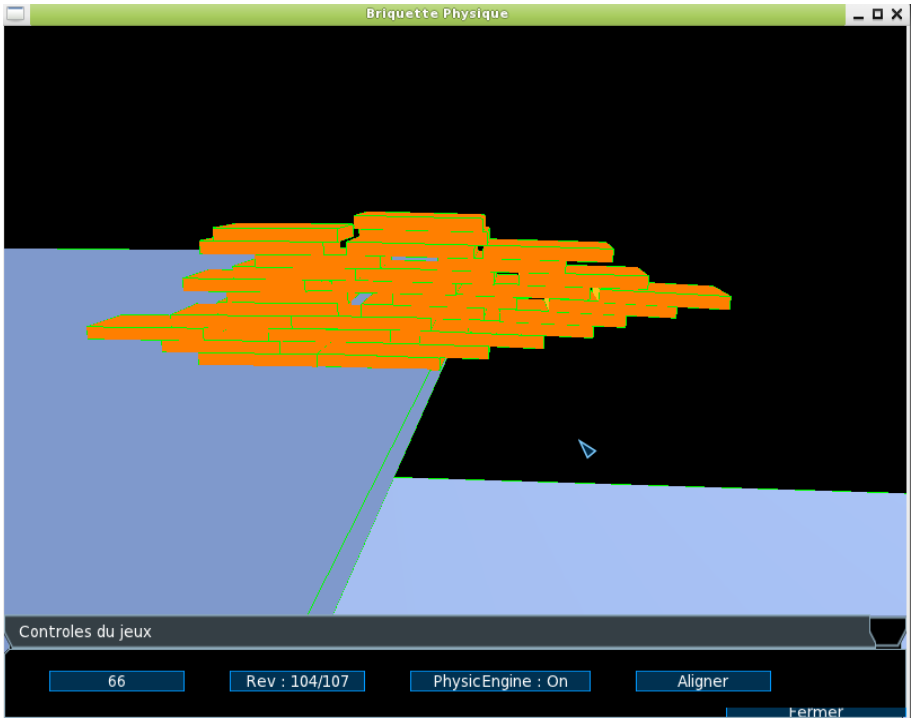

FIGURE 3.1 – Ici, on voit que la caméra n'est plus centrée autour du point central de la table mais sur une des briquettes de la structure, permettant d'étudier plus précisément celle-ci.

3.3 Gestion de la physique des briquettes

La physique est gérée avec la librairie Bullet. La technique consiste à lier à l'objet d'Ogre, un 'corps' particulier à Bullet sur lequel s'exerceront des forces et sur lequel on surveillera les possibles collisions. Bullet permet de rajouter une force d'attraction globale au monde et un poids à chacun des objets. Une des difficultés était de pouvoir outre-passer la gestion de la physique offerte par bullet lors de la sélection des briquettes pour les replacer. Pour cela il a fallu désactiver l'ensemble des forces s'exercant sur l'objet, permettre les déplacements de l'objet (lorsqu'un objet est stable, et qu'il n'y a pas de risque que celui-ci subisse de nouvelles collisions, bullet le bloque de façon à ne pas avoir à contrôler continuellement son état), et enfin empêcher la mise à jour de l'objet et l'application des nouvelles forces avant que l'utilisateur n'ait fini de positionner l'objet.

On utilise en réalité Bullet via OgreBullet pour l'intégrer avec Ogre, cependant OgreBullet est limité dans ses possibilités et il nous a parfois fallu travailler directement sur l'objet Bullet pour pouvoir obtenir le comportement souhaité.

Deplus l'activation de Bullet est très simple, il suffit d'appeler la méthode *stepSimulation* à chaque actualisation de la frame. Cette méthode a pour but d'effectuer toutes les actions du moteur physique : application de la gravité, mise à jour des forces sur les objets, gestion des collisions... Il nous est alors facile d'arrêter et de relancer le moteur physique à l'aide d'une condition avec un booléen.

Afin de gérer ce booléen, nous avons mis en place un système de mutex afin d'éviter que plusieurs threads modifient la valeur de ce booléen.

La méthode lock prend en paramètre un élément du type (void *) afin d'enregistrer quelle classe en fait l'appel. Il en est de même pour la méthode unlock. Et finalement la méthode de modification du booléen prend aussi en paramètre un pointeur sur la classe faisant l'appel pour savoir si elle a le droit d'effectuer la modification.

3.4 Sélection des briquettes

Pour sélectionner nos briquettes nous fournissons à l'utilisateur un curseur de souris. Lors du clic, un rayon est émis dans la direction de celui-ci, qui rapporte l'identifiant d'une briquette éventuellement atteinte. C'est également la librairie Bullet qui nous fournit les primitives nécessaires. Lorsqu'une briquette est sélectionnée, celle-ci prend une couleur violette pour avertir l'utilisateur.

3.5 Déplacement d'une briquette

Pour le déplacement d'une briquette, nous avons tout d'abord voulu utiliser la distance parcourue par la souris sur l'écran , puis directement déplacer la briquette en fonction de cette distance en X et Y. Le problème est que selon l'orientation de la caméra, la distance en X peut s'inverser et donc un déplacement de la souris vers la droite déplacerait la briquette vers la gauche.

Nous avons donc utilisé une autre méthode. Lors de la saisie d'une briquette, on crée un plan sur l'axe de positionnement des briquettes, puis nous effectuons un lancer de rayon depuis la position de la souris à l'écran. Ce plan possède la même largeur que celle d'une briquette. Le plan doit absolument être détruit après le lancer de rayon, sinon celui-ci intéragira avec les briquettes comme s'il y avait une collision, et les briquettes seraient alors projetées en dehors de leur axe commun Y.

Lors du déplacement de la souris, le rayon lancé percute alors un point du plan , il nous suffit alors de récupérer ce point et de déplacer la briquette aux mêmes coordonnées en X et Z que le point de collision (Y étant l'axe commun à toutes les briquettes). C'est pour cela, que le centre de la face latérale de la

briquette est toujours située à la même position que la souris. De plus, le plan ayant une taille limitée, il limite alors le déplacement de la briquette dans la scène.

Les briquettes tombant à côté de la table sont réinitialisées au centre de la table sur l'axe de construction.

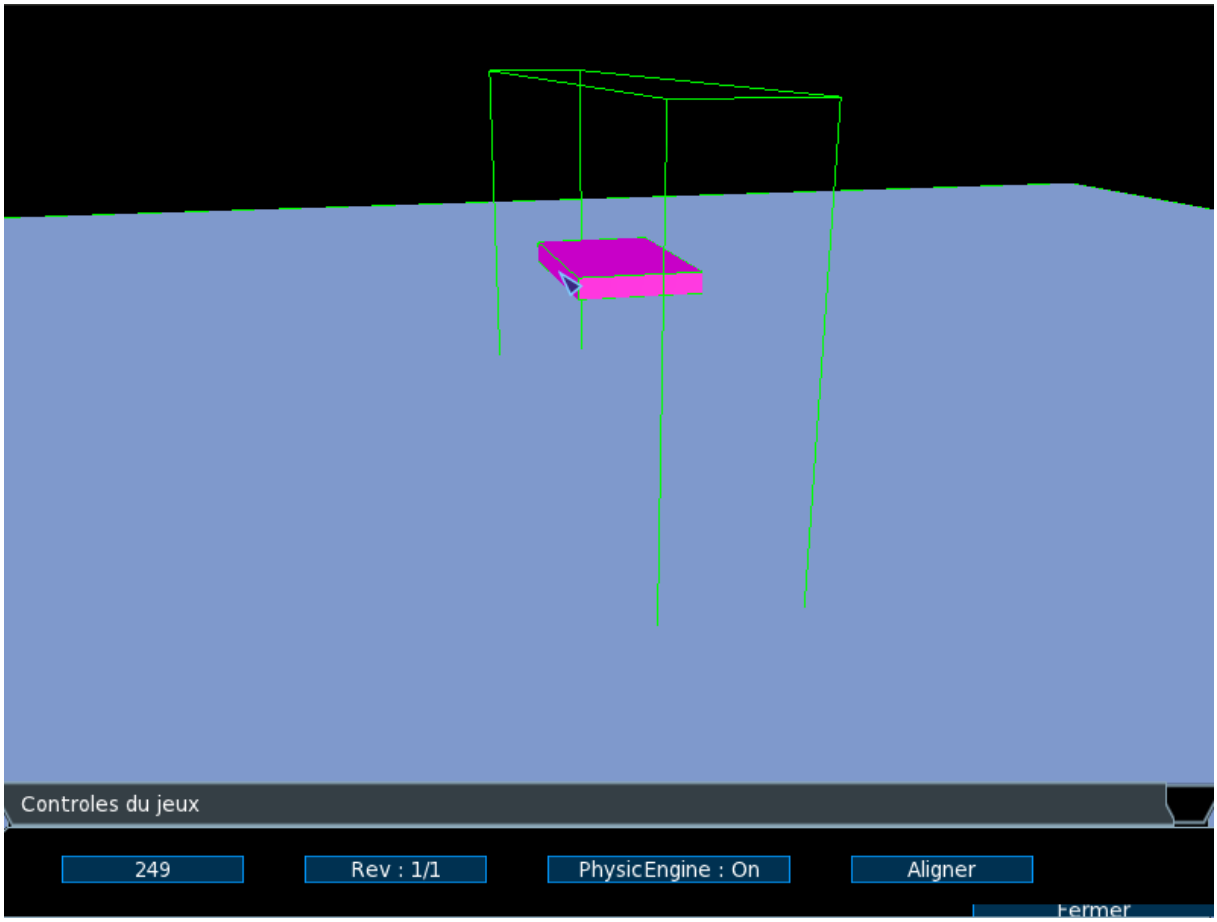

FIGURE 3.2 – On peut voir explicitement sur cette image le plan de collision qui est nécessaire au déplacement de la briquette.

3.6 Menus

Le menu à été réalisé en utilisant la bibliothèque CEGUI. Une classe virtuelle Fenetre à été utilisée pour permettre de créer différents types de Fenetre. Elle permet en particulier de mettre en place des mécanismes en gardant la même apparence graphique pour les différentes fenêtres que l'on crée, en offrant des méthodes pour les créer simplement. Il suffit d'insérer les boutons ou les éléments graphiques sur une fenêtre créée automatiquement.

La charte graphique provient de deux "skins" CEGUI pré-existants et placés sous licence libre. Il s'agit de "TaharezLook" qui est le thème par défaut de CEGUI et le thème "Sleekspace" que l'on peut télécharger sur le site. Ce thème est élégant mais n'est ni complet ni à jour. Je l'ai légèrement modifié pour le faire fonctionner (par exemple les labels n'apparaissaient plus correctement). Les thèmes utilisent un document xml pour décrire leur apparence qui est appelée le "Falagard looknfell system". J'ai donc travaillé dans le fichier xml présent dans media/CEGUI/looknfeel/TaharezLook.looknfeel. Le logiciel dispose d'un menu

d'introduction, simple qui permet de choisir le niveau de difficulté via 3 boutons : facile, intermédiaire, difficile.

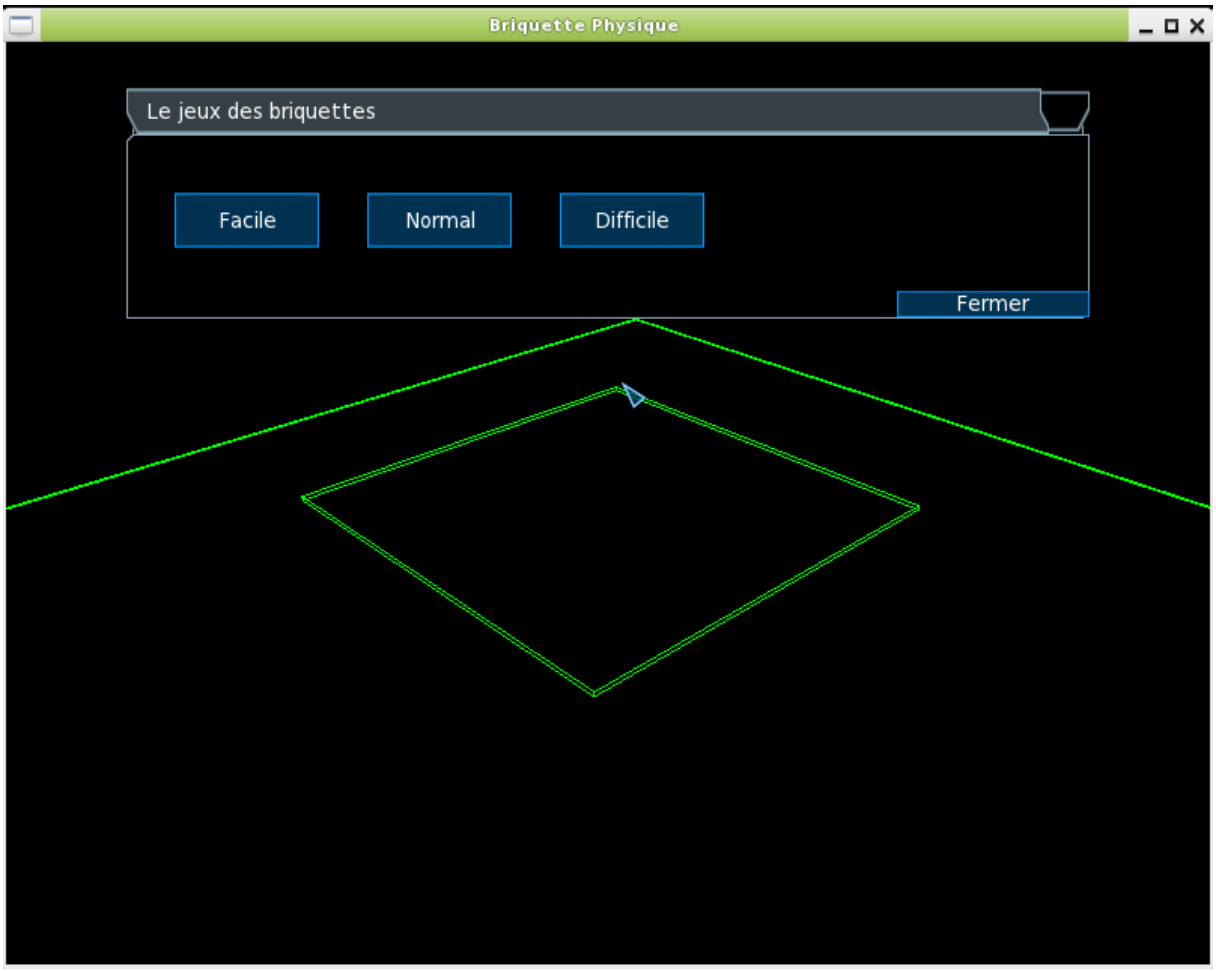

FIGURE 3.3 – Le menu d'introduction au jeu

Pendant le jeu, un menu est disponible en bas de l'écran. Il est également très simple. Il dispose de 4 boutons, le premier affiche le nombre de briquettes dont l'utilisateur peut disposer (qui ne sont pas encore en jeu). Un clic sur ce bouton permet de faire apparaître une briquette sur le plateau de jeu. Le second bouton permet de suivre les révisions (voir la partie sur les Snapshoots). Celui-ci est composé de 2 numéros, le premier annonce la révision actuelle sur laquelle on se trouve, le deuxième, le nombre de révisions accessibles. Les deux derniers boutons permettent respectivement d'activer ou de désactiver le moteur physique et d'aligner les briquettes sur l'axe de construction.

FIGURE 3.4 – Le menu en cours de jeu

Au départ, nous avions une souris gérée par Ogre (avec un overlay) qui permettait à l'utilisateur de viser les briquettes. CEGUI apporte sa propre souris. Le plus simple à été de ne conserver visible que la

souris de CEGUI tout au long du jeu. En pratique, on a conservé de l'ancienne souris d'Ogre qu'un jeu de coordonnées mis à jour à chaque déplacement de la souris de CEGUI afin de garder la synchronisation entre les deux.

3.7 Gestion des révisions

La plus grande difficulté d'utilisation du jeu était de ne pas faire d'erreurs pour ne pas devoir tout recommencer. Nous avons alors créer un système permettant de se déplacer parmi tous les changements effectués. Ces changements correspondent à l'ajout d'une nouvelle briquette , à l'alignement des briquettes, au déplacement d'une briquette ou tout simplement à l'ajout d'une révision par l'utilisateur.

Ce système est composé de 3 structures, comme on peut le voir sur le graphique de collaboration de la classe GestSnapShoot :

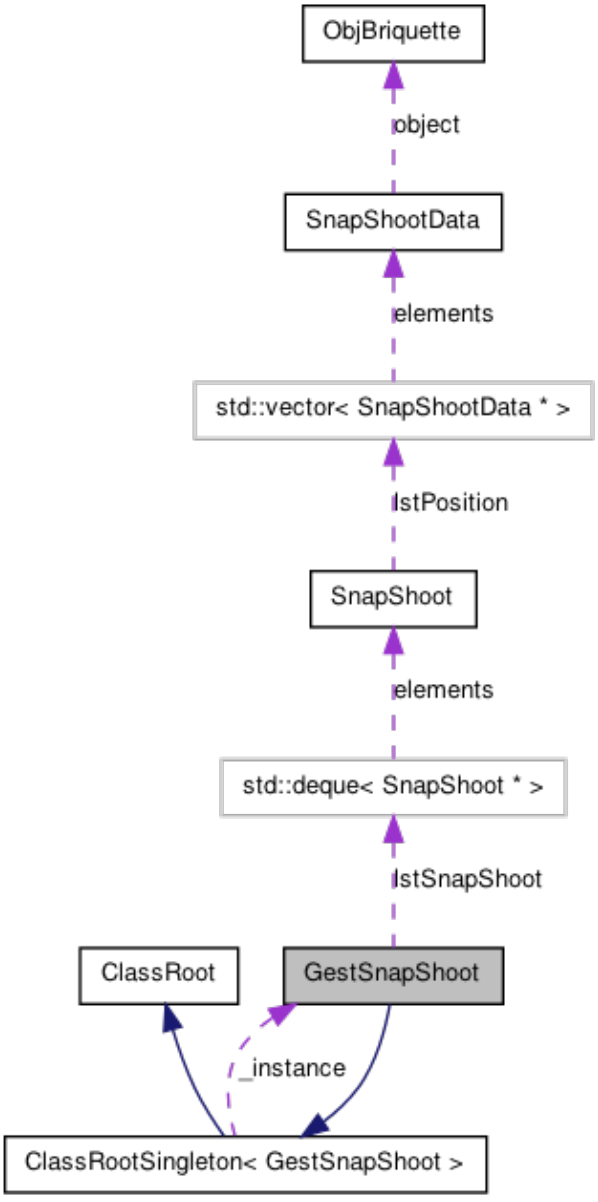

FIGURE 3.5 – Le graphique de collaboration de la classe GestSnapShoot

Maintenant, nous allons vous expliquer en détails le fonctionnement de ces 3 structures :

3.7.1 SnapShootData

Cette structure de données contient 3 éléments :
- Un pointeur du type ObjBriquette sur la briquette concernée par la donnée.
- Un vecteur de dimension 3 correspondant à la position de la briquette lors la capture.
- Un quaternion correspondant à l'orientation de la briquette.

3.7.2 SnapShoot

Cette classe correspond à une révision, un état du système qui contient la position et l'état des briquettes affichées dans la scène. Elle contient donc une liste d'élément de type SnapShootData.

Lors de la création d'une nouvelle instance de la classe (appel du constructeur) , le moteur physique de bullet est arrêté et la liste de l'instance est complété avec les briquettes affichées.

Cette classe contient donc entierement l'état du jeu à un instant donné. Il suffit donc de cacher toutes les briquettes stockées en mémoire et de ré-afficher en repositionnant les briquettes contenu dans la liste.

3.7.3 GestSnapShoot

Cette classe permet de gérer l'ensemble des révisions, elle contient une liste du type deque (liste "queue" accessible dans les deux sens). La classe permet d'ajouter une révision, d'en supprimer un ensemble depuis le début ou la fin, d'annuler une modification, de re-effectuer une modification et surtout d'appliquer une révision donnée.

3.8 Gestion du jeu

La plupart des actions effectuées par l'utilisateur, comme par exemple la gestion des révisions, n'appellent pas directement une méthode de GestSnapShoot. Elles appellent une méthode du même nom contenu par GestGame, cette sur-couche effectue l'action demandée par l'utilisateur mais elle permet de faire aux méthodes d'actualisation de l'affichage car au niveau structurel GestSnapShoot ne doit pas faire appel aux méthodes d'affichage. De plus, GestGame permet de gérer le niveau de difficulté et par conséquent le nombre maximum de briquettes en jeu.

3.9 Calcul du score

L'objectif est de construire la flèche la plus éloignée possible du bord de la table. Pour cela, nous avons mis en place un outil permettant de voir le score atteint. Durant le jeu, il suffit d'appuyer sur la touche "C" (Calcul) pour voir un menu donnant le score s'afficher. Celui-ci est calculé à partir de la briquette stable la plus éloignée de la table. Si la briquette vient d'être placée et qu'elle bouge encore un peu, elle ne sera pas comptabilisée immédiatement. Cela permet d'éviter d'avoir un score important en projettant une briquette à l'autre bout du jeu. Le calcul est fait entre le milieu de la briquette et le bord de la table.

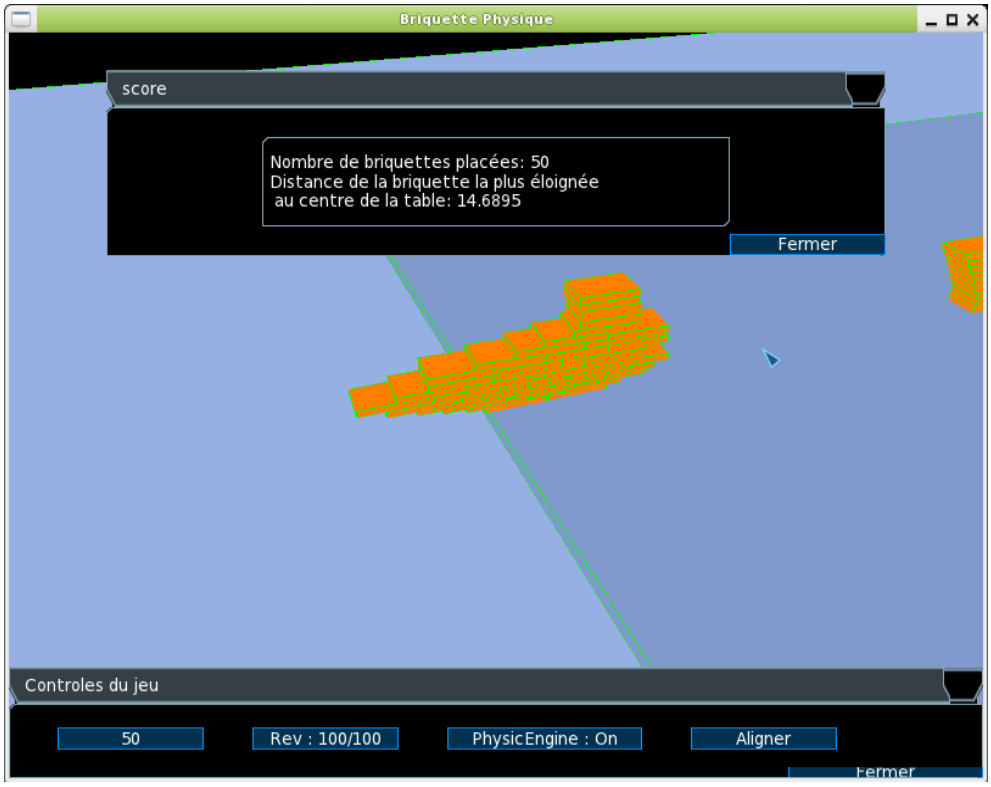

FIGURE 3.6 – Un exemple typique de l'application en cours de fonctionnment

Conclusion

La réutilisation d'outils que nous avions commencé à voir dans le projet collectif[8] nous a permis de gagner en efficacité et de mener le projet à bien. En particulier nous avons implémenté correctement la physique du jeu. Nous avons profité de cette aisance pour offrir de nombreuses améliorations, aidant la prise en mains du jeu (par exemple la possibilité d'arrêter le moteur physique, permettant de placer plusieurs briquettes avant de relancer la physique).

Le projet pourrait être continué de plusieurs manières, on pourrait mettre en place un langage permettant de placer les briquettes via des scripts et de tester les solutions ainsi obtenues. On pourrait également travailler pour perfectionner la stabilité des briquettes, une solution pourrait être de mettre en place un système de bords collants, pour placer la briquette au mieux, évitant ainsi un mouvement de chute pouvant mettre en péril la structure.

Bibliographie

[1] Ogre Wiki, *Utilisation des quaternions dans Ogre*, http://www.ogre3d.org/tikiwiki/Quaternion+and+Rotation+Primer

[2] Ogre Documentation, *Documentation Doxygen d'Ogre*, http://www.ogre3d.org/docs/api/html/index.html

[3] CEGUI Wiki, *Utilisation de CEGUI*, http://www.cegui.org.uk/wiki/index.php/Main_Page

[4] Bullet Wiki, *Utilisation de Bullet*, http://bulletphysics.org/wordpress/

[5] Ogre Wiki, *Tutorial d'utilisation d'OgreBullet*, http://www.ogre3d.org/tikiwiki/OgreBullet

[6] Site du Zéro, *Signaux avec QT*, http://www.siteduzero.com/tutoriel-3-11268-les-signaux-et-les-slots.html

[7] Doxygen, *Génération de documentation*, http://www.stack.nl/~dimitri/doxygen/

[8] Simulation Spatiale, *Projet Polytech'Tours de Réalité Virtuelle, 2011*, http://code.google.com/p/rv-simulation-interactive-ba

Construction en Kapla

Département Informatique
5e année
2010 - 2011

Projet de réalité virtuelle

Résumé : Ce rapport présentera notre projet de logiciel 3D de construction de Kapla. L'objectif de ce projet était de développer une simulation physique d'un jeu de Kapla. Nous y expliquerons les différentes techniques utilisées (signaux, déplacements, ...) pour obtenir le résultat le plus réaliste possible.

Mots clefs : Ogre3D, Bullet, Kapla, simulation

Abstract: This report will describe our project of a 3D software for building with Kapla. The main goal of this project was to develop a physic simulation of a Kapla game. We will describe the different techniques (signals, movement, ...) used to obtain the most realistic result.

Keywords: Ogre3D, Bullet, Kapla, simulation

Encadrant
Sebastien Aupetit
aupetit@univ-tours.fr
Emmanuel Néron
emmanuel.neron@univ-tours.fr

Étudiants
Guillaume Smaha
guillaume.smaha@etu.univ-tours.fr
Pierre Vittet
pierre.vittet@etu.univ-tours.fr

Université François-Rabelais, Tours