Ronan Renoux
rrenoux@enssat.fr

Malo Louboutin
mloubout@enssat.fr

Maxime Cordier
mcordier@enssat.fr

Module :
Contenu Multimédia

Promotion :
Fisa Informatique 2

Projet « Labyrinthe aveugle »

ENSSAT LANNION | Université de Rennes

1 Introduction

Dans le cadre du module Contenu Multimédia, nous avons réalisé un projet d'application de jeu. Il s'agit d'un labyrinthe aveugle. L'objectif du jeu est de parcourir le labyrinthe en partant d'un coin pour atteindre le coin inverse en diagonale. Cependant, le joueur doit évoluer dans le noir sans lumière pour se repérer. Il ne va utiliser que des sons. En fait, en fonction de son emplacement et des murs environnents, le joueur obtient un retour sonore dans un espace 3D. (Exemple : si le joueur se trouve à gauche d'un mur, il entend un son à sa gauche). Le joueur devient donc capable de se repérer dans le labyrinthe grâce à ses oreilles.

2 Mise en oeuvre

2.a Création du Labyrinthe

Le labyrinthe du jeu est généré selon l'algorithme Sidewinder. Concrètement, il s'agit d'un algorithme de génération de labyrinthes aléatoires. Il consiste à créer un labyrinthe en partant d'une grille de carrés. La première ligne de la grille est remplie de murs, puis on parcourt les lignes suivantes. Pour chaque ligne, on réalise des groupes de carrés pouvant aller de 1 carrés à N carrés (avec N le nombre de carrés par ligne). Puis, dans chaque groupe de carrés, nous supprimons un seul mur du haut au hasard parmi les carrés. Ainsi, on obtient un labyrinthe dans lequel il n'y a aucune zone inaccessible. Pour chaques carrés du labyrinthe nous enregistrons la configuration ouvert ou fermé des murs. Ces informations nous sont utiles pour la gestion des collisions et pour la gestion des sources sonores.

2.b Gestion des colisions

Afin de garantir le bon fonctionnement du jeu, nous avons implémenté une gestion des collisions. Ainsi, lorsque le joueur se déplace, il ne peut pas traverser les murs. De plus, nous avons ajouté « une marge » pour les murs de notre labyrinthe. On réalise finalement la collision avec cette marge et non pas directement avec le mur. Cela permet d'éviter le cas où la position du joueur se retrouve dans le mur. Nous distinguons deux scénarios pour la gestion des collisions : Premièrement, le joueur peut atteindre le grand **mur entourant le labyrinthe**. Suite à un calcul de nouvelle position, il se peut que le joueur se retrouve dans la zone entre la marge et le mur ou aussi hors de la zone de jeu. Dans ces cas, nous application un décalage en arrière pour revenir à la position précédente et rester dans la zone de jeu.

Deuxièmement, le joueur peut atteindre un **mur interne** dans le labyrinthe. Autrement dit, le joueur rencontre un mur d'un autre carré de labyrinthe. Nous avons créé la fonction hasWallBetweenCells pour savoir si un mur est réellement présent entre deux carrés internes au labyrinthe. Si oui, le mouvement n'est pas possible donc nous appliquons un décalage en arrière pour revenir à la position précédente et ainsi rester dans la zone de jeu.

2.c Gestion du son

Pour s'orienter dans le labyrinthe le joueur doit ce fier aux sons qu'il enttend. Pour cela, trois sources sonore existent : la source de droite, la source de gauche et la source de davant. Ces sources s'activent lorsqu'on appuis sur la touche « **X** » mais une source produit un son uniquement si un mur est présent dans la direction correspondant à chaque source. Pour savoir cela nous récupérerons les informations du carré courrant (configurations des murs ouverts ou fermés). Dans le cas d'un mur fermé la source sonore est disposée à la même distance que le mur et perpendicualire au joueur. Pour simplifier la gestion du son nous avons choisi d'autoriser des rotations du joueurs à 90° uniquement. Ainsi, le joueur sera constament perpendiculaire aux murs.

3 Tests du code

3.a Mode Debug

Un mode debug a été implémenté pour faciliter le développement du jeu. Ce mode est accessible en appuyant sur la touche « **Y** » du clavier. Avec le mode debug, nous avons ajouté la lumière pour pouvoir mieux se repérer. Il faut savoir qu'initialement la vision en profondeur maximale de la caméra est de 1. Cette valeur très faible et égale à la vision en profondeur minimale. Cela permet d'obtenir un écran completement noir pour jouer. En mode debug, cette valeur est de 1000. Cela permet de voir la scène et donc de se repérer dans le labyrinthe.

Par défaut, nous pouvons utiliser les touches « **Z** », « **Q** », « **S** », « **D** » pour se déplacer dans le labyrinthe. Le mode debug nous permet également d'autoriser plus de possibilité de déplacement. Les mouvements de souris deviennent possibles pour orienter la caméra. Il est ainsi possible de se déplacer sur un axe z (hauteur) pour observer le labyrinthe depuis l'extérieur. De plus, les collisions avec les murs sont également déactivées.

Ce mode debug nous a notamment permis de vérifier le bon fonctionnement de l'algorithme de génération du labyrinthe. Celui-ci pourrait également nous être utile pour mettre en place le mode spectateur du jeu. Ce mode n'a pas pu être implémenté dans le temps imparti.

4 Conclusion

Ce projet nous a permis de mettre en pratique les connaissances acquises en cours. Nous avons pu mettre en oeuvre les notions de programmation objet avec le language de programmation C++. Nous avons égelement découvert la bibliothèque OpenAL nous permettant de faire de la programation audio. Nous avons du revoir certains de nos objectifs comme les déplacements du joueurs. Avec un déplacement à 90℃ nous avons pu simplifier la gestion du son mais nous savons aussi qu'il s'agit d'un point d'amélioration possible de notre projet. Pour compléter notre application nous aurions également voulu mettre en place ….