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. L'objectif est de créer un labyrinthe aveugle. Le but du jeu est de trouver la sortie du labyrinthe en utilisant uniquement des sons. Ainsi, en fonction de son emplacement et des murs environnement, le joueur obtient un retour sonor dans un espace 3D. (Exemple : si le joueur se trouve à gauche d'un mur, il entend un son à sa gauche).

2 Mise en oeuvre

2.a Nos objets

Notre Labyrinthe Aveugle est une application codé dans le langage de programmation C++. Par le biais de la programmation orienté objet, nous avons créé plusieurs classes pour la gestion de notre jeu :

- main.cpp : classe principale
- Scene.cpp : classe de la scène
- MaterialTexture.cpp : classe des textures
- Labyrinthe.cpp : classe du labyrinthe
- Cubes.cpp : classe des cubes
- Ground.cpp : classe du sol

2.b Création du Labyrinthe

Le labyrinthe du jeu est généré selon l'algorithme Sidewinder. Concrètement, il s'agit d'un algorithme de génération de labyrinthe aléatoire. 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 supprimimons un seul mur du haut au hasard parmis les carrés. Ainsi, on obtient un labyrinthe dans lequel il n'y a aucune zone innaccessible.

2.c Gestion des colisions

Afin de garentir 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 colision avec cette marge et non pas directement avec le mur. Cela permet d'éviter le cas où position du joueur se retrouve dans le mur. Nous distingue deux scénarios pour la gestion des collisions : Premièrement, le joueurs 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 ainsi rester dans la zone de jeu. Deuxièmement, le joueurs 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.d Gestion du son

sfdh OCUOU

3 Tests du code

3.a Mode Debug

4 Conclusion