ENSSAT
LANNION

Projet Blind Maze

Labyrinthe en 2d avec une écholocation

C programming language

20-05-2023

Réalisé par :
Ferhat Saidoun
Abir Hsaine
Zakaria Raji

Objectifs

1. Echo-location simple : le joueur doit appuyer sur des touches pour savoir s'il y a un mur à droite/gauche/devant.
2. Echo-location avancée : le joueur obtient automatiquement un retour en fonction de son environnement.
3. Ajout d'un mode spectateur : pour des raisons de debug, vous pouvez avoir besoin d'afficher le labyrinthe dans une fenêtre et la rendre visible aux spectateurs.

Introduction

Nous avons développé une application de jeu dans laquelle le joueur doit trouver la sortie d'un labyrinthe en deux dimensions qui n'est pas éclairé. L'application permet un déplacement libre au clavier dans le but de rechercher un artiste qui joue de la guitare. Lorsque le joueur s'approche suffisamment de l'artiste (le point de sortie du labyrinthe), le volume sonore augmente de manière significative. Une fois que le joueur pénètre dans la zone de sortie, le jeu se termine et affiche "won" dans le terminal. L'objectif est de trouver la sortie afin d'obtenir un autographe du guitariste.
Note : Tous les objectifs sont réalisés.

Côté visuel : 2D

La bibliothèque SDL a été utilisée pour générer le labyrinthe visuellement en mode debug.
Le labyrinthe est composé de plusieurs cases, toutes de la même taille, dans notre cas une case 32x32. Supposons que le nombre total de cases soit de 19x19. En C, le labyrinthe est représenté par une matrice 19x19 où chaque valeur supérieure à 0 désigne une case vide, sinon il y a un mur (Note : On peut ajouter d'autres objets en précisant d'autres numéros, par exemple 5 pour un ennemi).
La position du joueur, sa taille, sa vitesse de marche et sa vitesse de rotation sont initialisées dans le fichier player_setup.c. Le joueur à la liberté de se déplacer sans restriction dans les zones vides. Une fonction movePlayer() a été mise en place pour déterminer la position du joueur et décider de son déplacement dans une direction donnée, tout en vérifiant la présence d'un mur dans cette direction. Une collision se produit avec un mur lorsque la nouvelle position du joueur se trouve en intersection avec la surface du mur. En calculant les coordonnées de la nouvelle position par rapport à la carte (matrice), il est possible de déterminer si la case correspond à un mur ou à un chemin libre.

Côté audio : 3D

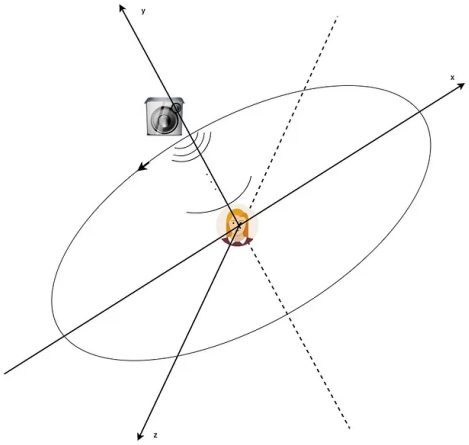
Avec l'API OpenAL, nous aurons la possibilité de configurer la scène sonore suivante (son spatiale) :

- Une source de son.
- Un auditeur positionné au milieu de la scène (0,0).
- La source sonore tourne autour de l'auditeur sur les axes x et y (version 2d)

Comme indiqué dans le diagramme suivant (sans prendre en compte l'axe z, car on est en version 2d) :

Dans l'application, trois sons différents sont joués :

- Le son de la marche : Il s'agit d'un son ordinaire sans spatialisation. Il accompagne les mouvements du joueur dans le labyrinthe.



- Le son de la sortie et de la collision avec le mur : Lorsque le joueur s'approche de la sortie du labyrinthe ou lorsqu'il y a une collision avec un mur, un son spatial est joué. Ce son est positionné en fonction de la position et de l'angle du joueur, ainsi que de la position et de l'angle de la source audio correspondant au mur ou à la sortie.

Cela crée une sensation d'immersion sonore réaliste, indiquant au joueur qu'il se rapproche de la sortie ou qu'il a rencontré un obstacle.

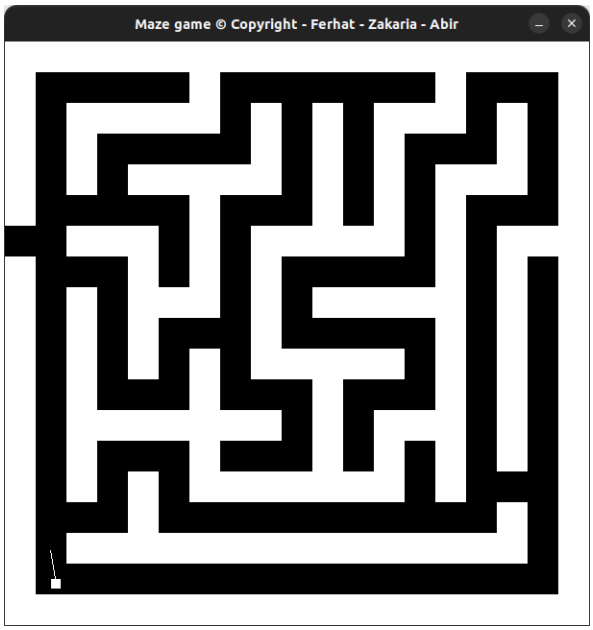
Pour tenir compte de la position réelle du joueur par rapport à la carte du labyrinthe, les coordonnées de la source audio ont été normalisées. Cela signifie que les coordonnées de la source audio ont été ajustées de manière à ce que la position du joueur soit considérée comme l'origine du système de coordonnées (0,0). En normalisant les coordonnées, on peut ensuite positionner la source audio en fonction de la position relative du joueur par rapport à la carte, ce qui permet une spatialisation correcte du son.

Pour s'y faire :

1. Calculer la différence horizontale et verticale (dx & dy) entre la position de la source et celle du joueur.
2. Calcul de la direction du joueur (dirX & dirY) vers la source correspondent à la normalisation du vecteur formé par la question 1. Cela signifie que les valeurs sont ajustées pour représenter une direction unitaire, indépendamment de la distance réelle entre le joueur et la sortie.
3. Calcul de l'angle entre le joueur et la source :
    - *anglePlayerSource* est calculé en utilisant la fonction *acos*() pour trouver l'arc cosinus de dirX, qui représente la projection horizontale de la direction du joueur vers la sortie. La condition dirY < 0 permet de gérer les angles dans les quadrants supérieurs.
    - L'angle *anglePlayerSource* est utilisé pour déterminer l'angle du son *angleSound* en ajoutant l'angle de rotation du joueur (*p.rotationAngle*) et π/2 pour compenser l'orientation de l'écran (le coin supérieur gauche étant le point de référence).
4. Rotation de la position du son en fonction de l'orientation du joueur :
    - *SoundX* et *SoundY* représentent les nouvelles coordonnées de la source sonore après avoir tourné les coordonnées initiales selon l'angle *angleSound*. Ces coordonnées sont calculées en utilisant les fonctions trigonométriques cos() et sin() respectivement, multipliées par la distance entre le joueur et la sortie.

En résumé, ce calcul permet de déterminer la position spatiale de la source sonore de manière à ce qu'elle corresponde à la position relative du joueur par rapport à la sortie, en tenant compte de l'orientation du joueur. Cela permet de créer une spatialisation réaliste du son dans le jeu.

Annexes



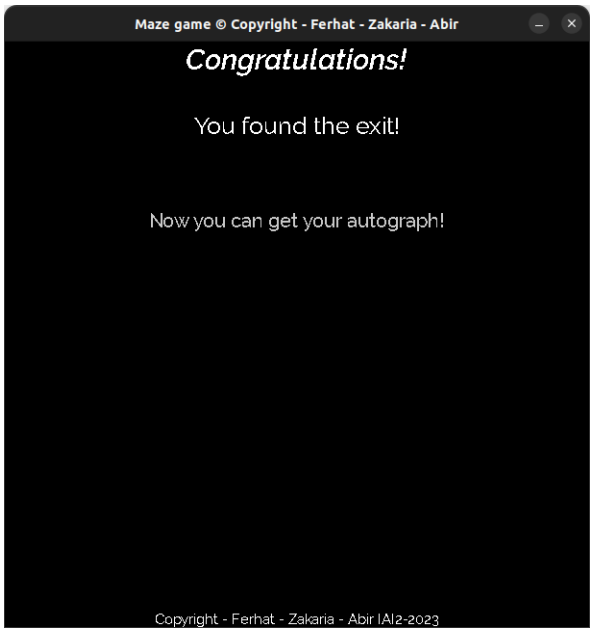


Fig - Blind Maze game - Visuel - en C[OBJ]

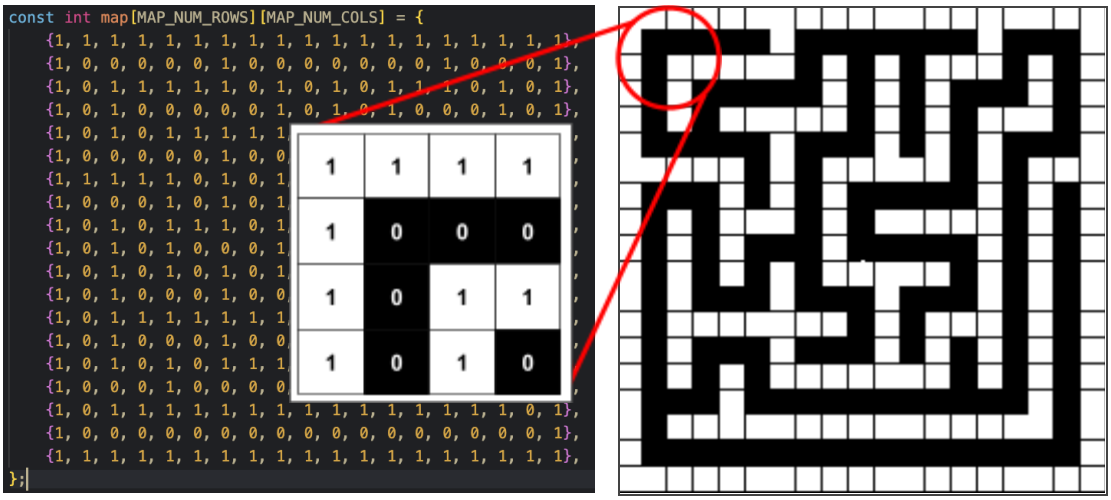


Fig - Maze - Visuel - Matrice en C