Rapport du projet Doom-like

Sommaire

Introduction

Commandes et utilisations du programme :

Commandes dans le moteur graphique :
- z : avancer
- s : reculer
- q : se déplacer sur la gauche
- d : se déplacer sur la droite
- a : pivoter sur la gauche
- e : pivoter sur la droite
- r : remettre le joueur à sa position initiale
- u : faire faire au joueur le chemin entre sa position initiale et l'emplacement final
- c : activer/désaciver le sprint
- v : colorier les murs
- b/n : changer la couleur avec laquelle on peint
- k : quitter le programme

Pour les touches avancées, telles « u », « c », « v », « b » et « n », se référer à leurs sections respectives pour plus d'explications.

Utilisation du programme :

faire la commande make à l'endroit du Makefile puis faire la commande suivante dans le terminal :

./bsp labyrinthes/labyrinthe.lab

labyrinthe.lab étant le labyrinthe qu'on va afficher.
Indication : pour la mini map et la touche « u », il faut que ce soit le labyrinthe « random_lab.lab » pour que ça marche parfaitement.

Partie I

Pour rappel, nous devions implémenter un moteur graphique à la doom, pour ce faire nous avons comme structure de données principale un arbre BSP qui se présente sous la forme suivante :

type t = E | N of Segment.t * t * t

On remarque que c'est un simple arbre dont les nœuds sont des segments qu'on affichera.

La question est donc : comment va ton utiliser l'arbre BSP pour afficher les segments ?

Premierement, on calcul les coordonnées des segments selon la position et l'angle du joueur.

Ensuite, on regarde si les segments sont dans le champ de vision du joueur, si il est derriere ce dernier, on n'applique aucun calculs dessus, dans le projet, cela s'appelle le clipping.

Pour les segments qui restent, nous les projetons sur un ecran puis on fait un deuxieme clipping pour afficher en 3D les segments qui sont projetés sur l'ecran (dans le code, les segments affichés sont ceux dont la projection est comprise entre les variables cmax et cmin dans le fichier render.ml).

Par rapport au sujet original, quelques modifications ont été faites.

Par exemple, nous n'utilisons pas de champs ci ou ce pour afficher nos segments.

Ensuite, nous n'utilisons pas l'algorithmes avec les calculs des hauteurs maximales et minimales d'affichages des extremités des segments.
A la place, nous calculons le rapport entre la distance entre le joueur et l'ecran avec la distance de chaque extremité avec la position du joueur.

Ce sont ces lignes de codes :

```
let calcul_p_y x y =
        let echelle = float_of_int(taille/4) in
        let rapport = float_of_int d_focale /. distance x y 0 0 in
        let calcul = int_of_float(echelle *. rapport)+hauteur_yeux in
        calcul
```

L'échelle étant prise de facon subjective, on voulait que lorsqu'un segment était à la meme distance que l'écran ou on le projette il prenne un quart de l'écran.

Partie II

Téléportation :

On peut créer un portail de téléportation, un segment à un champ « id_autre » qui est à 0 si il pointe sur aucun autre segment, si cet identifaint est différent de 0, il indique l'id du segment ou on saute quand on travers le téléporteur.

On remarque donc qu'un mur A peut téléporter vers un mur B sans pour autant que le mur B téléporte vers le mur A.

On remarquera qu'il y a un bug avec cette fonctionnalité, un mur téléporte que dans un sens, explications : si on arrive du mauvais coté du mur on ne se téléporte pas, on traverse simplement le mur, pour etre téléporté, il suffit donc de revenir sur le mur car on sera du bon sens.
Pas d'inquiétudes, cette foncitonalité ne génère pas de bug sur tout ce qui est collision.

Peintre :

Il y a trois commandes qui apparaissent avec le peintre, sur un clavier azerty, ces touches sont « v », « b » et « n ».

Commencons par les commandes « b » et « n », la commande « b » permet d'attribuer la couleur précédente de notre collection de couleurs au joueur/peintre, « n » quant à elle permet d'attribuer la couleur suivant.

Notre collection de couleurs étant celle ci (dans l'ordre) =

Blanc, Rouge, Vert, Bleu, Jaune, Cyan, Magenta

En ce qui concerne la touche « v », elle permet de dessiner aléatoirement les murs avec la couleur du joueur avec une probabilité de 1 sur 5.

Une amélioration qu'on pourrait apporter serait de dessiner uniquement le mur que le joueur/peintre vise.

Map :

Nous avons implémenté une mini map, elle ne marche bien que lorsque le labyrinthe sélectioné est le labyrinthe généré aléatoirement car la mini map utilise des attributs du Générateur.

Ennemi :

Un enemmi est représenté par un id et une position, il est ajouté ou enlevé au bsp en regardant sa position par rapport aux autres segments, la complexité est donc en O(log2 N).

Pour tuer un ennemi, le joueur doit tirer dessus, la balle est representée par un segment qui s'agrandi, elle detecte ainsi les collisions avec les ennemis, les murs et possede une distance limite.

Sprint :

Nous avons implémenté un sprint, l'implémentation a été facile, il suffit de multiplier par 2 le pas si on veut que le joueur se déplace 2 fois plus vite. Attention aux collisions avec le sprint, il faut prendre en compte que le pas est plus grand.

Générateur/solveur :

Nous avons implémenté un générateur qui fait un labyrinthe parfait de facon aléatoire.

Pour cela nous avions eu deux choix, utiliser union-find que nous conaissions ou utliser la recherche exhaustive.

Nous avons choisi de faire la recherche exhaustive car nous avons pensé que ce serait intéressant de l'implémenter.

Alors qu'est ce qu'est la recherche exhaustive ?

L'algorithme est le suivant :

On prend une case de départ, on regarde ses voisins, on en choisit un qui n'a pas encore été visité et on abbat le mur entre l'ancienne case et la nouvelle.
Si la case n'a pas de voisins de disponible, on remonte la pile d'appel jusqu'à trouver un voisin pas encore visité.

On fait ça jusqu'à avoir abbatu taille²-1 murs car c'est une propriété des labyrinthes parfaits d'avoir taille²-1 murs abbatus.

Pour avoir le chemin entre la case de départ et la case d'arrivée (coté solveur) , il suffit de sauvegarder la pile quand la case actuelle est égale à la case d'arrivée.

Attention, il faut qu'il y ait un fichier « random_lab.lab » (meme vide) pour ne pas avoir d'erreur.