Labyrinthe

15 octobre 2016

1 Sujet

La première étape de ce projet consiste à créer un labyrinthe à deux dimensions dans un rectangle, ayant la propriété qu'il y ait un seul chemin connectant deux points quelconques.

Ensuite, on peut rajouter une sortie, et inventer un jeu comme par exemple: un trésor est caché quelque part. Le héros rentre dans le labyrinthe, se déplace avec 4 touches, va chercher le trésor, et ressort avant que des monstres ambulants et le suivant à l'odeur ne l'aient atteint ...

L'algoritme pour construire le labyrinthe peut être le suivant (à *améliorer lorsque vous aurez vu ses défauts.* voir l'article de D.Matuzsek dans le journal "Byte" de décembre 1981). Il ressemble à la croissance d'un cristal de neige.

On dispose d'un réseau de 20×20 cellules. On pose $N = 20$.

(1) Au départ les cellules sont toutes "non traitées". Seule la cellule du centre est "traitée".

(2) On choisit une cellule X' traitée au hasard qui possède au moins une cellule voisine non traitée. On choisit au hasard une cellule X'' non traitée, voisine de X'.

(3) On connecte les cellules X' et X'' et on rajoute X'' à la liste des cellules traitées.

(4) On repart à l'étape 2 jusqu'à ce que toutes les cellules du réseau soient traitées.

A la fin, les connections sont les chemins du labyrinthe pour se déplacer d'une cellule à une autre.

2 Indications plus précises sur l'algorithme

2.1 Structures utilisées et codage

Soit $N \geq 1$ la taille du labyrinthe.

On crée:

- Un tableau d'entiers (matrice d'entiers) de taille $N \times N$, notée $M(i,j)$, $i,j = 0 \rightarrow N-1$, contenant les connections du labyrinthe:

- $M(i,j) = -1$ si la case n'est pas encore "traitée".
- $M(i,j) = 0$ si sans connexion avec ses voisins.
- $M(i,j) = 1$ si connexion avec voisin de droite: $\boxed{\rightarrow}$
- $M(i,j) = 2$ si connexion avec voisin du haut : $\boxed{\uparrow}$
- $M(i,j) = 3$ si connexion avec voisin de droite: $\boxed{\rightarrow}$ et du haut $\boxed{\uparrow}$.

- Codes des directions d et variations de coordonnées de (i,j):

d :	0	1	2	3
direction	$\rightarrow$	$\uparrow$	$\leftarrow$	$\downarrow$
D:	$(1,0)$	$(0,1)$	$(-1,0)$	$0,-1$

- On note $L = (L_i(k), L_j(k))_{k=0\rightarrow l-1}$ deux listes L_i, L_j d'entiers de taille $l \geq 0$:
L_i : $\begin{array}{|c|c|c|c|c|}\hline i_0 & i_1 & & & i_{l-1} \\ \hline 0 & 1 & & & l-1\end{array}$, et L_j : $\begin{array}{|c|c|c|c|c|}\hline j_0 & j_1 & & & j_{l-1} \\ \hline 0 & 1 & & & l-1\end{array}$, où $(i_k, j_k) = (L_i(k), L_j(k))$ sont les coordonnées des cases $k = 0 \rightarrow l-1$ "traitées" et qui ont des voisins libres.

2.2 Algo

(1) On choisit un point de départ $(i_0, j_0) = (0,0)$. On initialise M: $M(i,j) = -1$ pour tout (i,j) sauf $M(i_0, j_0) = 0$ et on initialise L contenant le seul point (i_0, j_0) .

(2) Choisir un élément k au hasard dans l'intervalle $0 \leq k < l$ où $l = taille(L)$. On pose $X' = (i', j') = (L_i(k), L_j(k))$ qui est la coordonnées d'une case traitée.

(3) On choisit une direction $d \in \{0,1,2,3\}$ au hasard.

(4) Soit $(i'', j'') = (i', j')$ +décalage en direction d, c'est à dire $X'' = X' + D(d)$. On teste si X'' ne sort pas du tableau et si la case X'' est libre, cad $M(X'') == -1$. (faire une fonction `test()` pour cela).
Si test précédent pas OK, faire $d = d + 1$ modulo 4 et revenir en (4).
Si le test précédent est OK, alors on s'y connecte c'est à dire:

(a) rajoute X'' à la liste L et on pose $M(X'') = 0$.

(b) Créer la connection: il y a 4 cas:

i. Si $d = 0$ (X'' à droite de X') faire $M(X') = M(X') + 1$
ii. Si $d = 1$ (X'' au dessus de X') faire $M(X') = M(X') + 2$
iii. Si $d = 2$ (X'' à gauche de X') faire $M(X'') = M(X'') + 1$
iv. Si $d = 3$ (X'' au dessous de X') faire $M(X'') = M(X'') + 2$

(5) On nettoye la liste L, c'est à dire on parcourt L et pour chaque case $X \in L$, on l'enlève si elle n'a pas de voisin libre, cad que l'on fait

(a) boucle $d = 0 \rightarrow 3$,

i. $X' = X + D(d)$,

ii. On teste si $X' \in Tableau$ et si X' est libre.

(6) Si $taille(L) = 0$ c'est la fin, sinon retour en 2.

2.3 Remarques sur le dessin

Le tableau $M(i,j) \in \{0,1,2,3\}$ obtenu contient le codage des connections possibles entre les cases. Pour le dessin il faut représenter ces connections. Il y a plusieures possibilités, en voici une (les traits fins ne doivent pas être dessiné):

code M: 0 1 2 3

dessin:

2.4 Amélioration de l'algorithme

On pourra aux étapes 2 et 3, choisir les valeurs k, d avec d'autres règles afin d'obtenir d'autres géométries pour le labyrinthe. Par exemple favoriser les longs chemins...

2.5 Indications de programmation en C++

2.5.1 Concernant les types de données

Matrice M: la matrice M sera déclarée en utilisant la librairie armadillo:

```
int N=3;
Mat<int> M;
M.zeros(N,N); // remplit la matrice de zeros
```

Listes L_i,L_j: on utilisera la classe vector:

```
vector<int> L_i, L_j; // declaration
L_i.push_back(X); // rajoute X a la fin de la liste
L_i.size(); // taille de la liste
L_i[k]; // element, 0 <= k < L_i.size()
L_i.erase(L_i.begin() + k); // elimine element
```

2.5.2 Etapes:

(1) Commencer par écrire la fonction `test()` et vérifier son fonctionnement.

(2) Ecrire l'algo et tester son fonctionnement pas à pas sur un tableau de petite taille, par exemple $N = 3$.

(3) Ecrire une fonction qui dessine le labyrinthe à partir du tableau M des connections.

2.6 Indications de programmation en python

2.6.1 Concernant les types de données

Matrice M: la matrice M sera déclarée:

```
N=3
M=zeros([N,N]) # matrice N × N remplit de zeros
```

Listes L_i,L_j: on utilisera:

```
L_i, L_j = [],[] # declaration de listes vides
L_i.append(X) # rajoute X a la fin de la liste
len(L_i) # taille de la liste
L_i[k] # element, 0 <= k < L_i.size()
del L_i[k] # elimine element a la position k
```

2.6.2 Etapes:

(1) Commencer par écrire la fonction `test()` et vérifier son fonctionnement.

(2) Ecrire l'algo et tester son fonctionnement pas à pas sur un tableau de petite taille, par exemple $N = 3$.

(3) Ecrire une fonction qui dessine le labyrinthe à partir du tableau M des connections.

3 Représentation du labyrinthe en 3D

3.1 Calcul préalable

Dans l'espace $(x, y, z) \in \mathbb{R}^3$, on suppose l'observateur au point $O = (0, 0, 0)$ et qu'il regarde dans la direction $\vec{v} = (1, 0, 0)$.

L'"écran" est représenté par le plan $P := \{(x', y', z') \text{ t.q.} x' = 1\}$.

Soit un point $M\,(x, y, z) \in \mathbb{R}^3$. On cherche l'intersection de la droite (O, M) avec le plan P, donnant un point $M'\,(x' = 1, y', z')$. On posera

$$X = -y', \quad Y = z'$$

Question 3.1. *Etant donné $M\,(x, y, z) \in \mathbb{R}^3$, montrer que*

$$X = -\frac{y}{x}, \quad Y = \frac{z}{x} \tag{3.1}$$

Solution 3.2. On écrit $\vec{OM'} = \lambda \vec{OM}$ et $M' \in P$, cad

$$x' = 1 = \lambda x$$
$$y' = \lambda y = \frac{y}{x}$$
$$z' = \lambda z = \frac{z}{x}$$

donc $X = -\frac{y}{x}$, $Y = \frac{z}{x}$.

3.2 Représentation 3D du labyrinthe

On note $(i_0, j_0, k_0) \in \mathbb{R}^2$ la position du joueur dans les coordonnées du labyrinthe. Au départ, $i_0 = 0.25$, $j_0 = 0.25$, $k_0 = 0.5$.

On note $\theta \in \mathbb{R}$ l'angle avec l'axe i donnant la direction du regard.

Formule de changement de coordonnées pour un point M de l'espace,

si (i, j, k) sont les coordonnées d'un point M dans le repère du labyrinthe alors ses coordonnées (x, y, z) dans le repère du joueur sont

$$\begin{aligned} x &= \cos\theta\,(i - i_0) + \sin\theta\,(j - j_0) \\ y &= -\sin\theta\,(i - i_0) + \cos\theta\,(j - j_0) \\ z &= k - k_0 \end{aligned} \tag{3.2}$$

Soit $R > 0$, par exemple $R = 4$ la "profondeur d'affichage".

(1) On parcourt les cases de coordonnées (i_c, j_c) avec $i_c = i_0 - R \to i_0 + R$ et $j_c = j_0 - R \to j_0 + R$

(a) La case (i_c, j_c) contient (ou pas) des cubes aux positions (i_{c2}, j_{c2}). Précisément,

- si $M(i_c, j_c) \in \{0, 1\}$ alors on considère $(i_{c2}, j_{c2}) = (i_c, j_c + 0.5)$.
- si $M(i_c, j_c) \in \{0, 2\}$ alors on considère $(i_{c2}, j_{c2}) = (i_c + 0.5, j_c)$.
- Toujours, on considère $(i_{c2}, j_{c2}) = (i_c + 0.5, j_c + 0.5)$.

(b) Pour chaque case (i_{c2}, j_{c2}) ci-dessus, il y a 4 faces verticales à considérer. Chaque face A, B, C, D est un carré avec 4 sommets $s = 0, 1, 2, 3$ de coordonnées $(i_s, j_s, k_s)_{s=0,1,2,3}$:

i. Pour face A, $(i_s, j_s, k_s) \in \{(i_{c2}, j_{c2}, 0), (i_{c2}, j_{c2}, 1), (i_{c2} + 0.25, j_{c2}, 1), (i_{c2} + 0.25, j_{c2}, 0)\}$

ii. Pour face B, $(i_s, j_s, k_s) \in \{(i_{c2} + 0.25, j_{c2}, 1), (i_{c2} + 0.25, j_{c2}, 0), (i_{c2} + 0.25, j_{c2} + 0.25, 0),$

iii. Pour face C, $(i_s, j_s, k_s) \in \{(i_{c2}, j_{c2} + 0.25, 0), (i_{c2}, j_{c2} + 0.25, 1), (i_{c2} + 0.25, j_{c2} + 0.25, 1),$

iv. Pour face D, $(i_s, j_s, k_s) \in \{(i_{c2}, j_{c2}, 1), (i_{c2}, j_{c2}, 0), (i_{c2}, j_{c2} + 0.25, 0), (i_{c2}, j_{c2} + 0.25, 1)\}$

(c) Pour chaque face, on a une liste de ces 4 sommets de coordonnées (i_s, j_s, k_s). On transforme $(i_s, j_s, k_s) \to (x_s, y_s, z_s)$ avec (3.2). Si $x < 0$ pour au moins un des 4 sommets, on ne fait rien car l'objet est derrière le joueur. Sinon on transforme $(x_s, y_s, z_s) \to (X_s, Y_s)$ avec (3.1). On obtient donc une liste de 4 points (X_s, Y_s). On stoke cela dans une liste L_{faces}. De plus on stocke la valeur maximale x_s rencontrée.

(d) On ordonne la liste L_{faces} selon les valeurs x_s décroissantes.

(e) On parcourt la liste $L_{faces,}$ on dessine les polygones ayant les 4 sommets (X_s, Y_s).