Resolution d'un problème de déplacement avec le langage C
Utilisation de l'algorithme de LEE

Victor Vitcheff

6 avril 2022

Table des matières

Introduction

Je vais vous présenter ici la solution que j'ai adopté pour résoudre le problème suivant. Il existe une grille donc un tableau de taille N*M. Sur chaque case il peut y avoir du feu noté avec le caractère 'F', un arbre noté avec le caractère '♯', un terrain libre noté avec le caractère '.' une position de départ noté avec le caractère 'S' et des positions de sortie noté avec le caractère 'E'. Chaque case possède 2 à quatre voisins (un à gauche, un à droite, un en haut et un en bas). Un personnage se trouve à l'instant t=0 sur la case 'S'. Il doit atteindre une case noté 'E'. A chaque coup, il peut se déplacer sur une case voisine. Et a chaque tour, la case feu brûle toutes ses cases voisines même si c'est un arbre. Le personnage doit contourner les cases arbres. Son but est d'atteindre une case E sans brûler. Pour résoudre ce problème, j'ai utilisé le langage C. En effet, c'est le langage avec lequel je suis le plus à l'aise car je préfère la programmation modulaire. Je vais vous présenter ma solution puis je vous expliquerai mes choix quant à cette solution.

I Presentation de la solution

I.1 Le fichier

La solution est composé de 3 fichiers. 2 sont des modules, c'est à dire qu'il ne possède pas de fonction main et qu'on ne les lance pas dans le terminal. Le dernier se nomme "main.c" et c'est le programme principal. Dans ce fichier, les 2 modules sont intégrés. Les 2 modules sont aussi constitués de header, ces header permettent de décrire les fonctions et que les fonctions puissent s'appeler entre elles.

Ma solution a d'abord été de récupérer la grille sous la forme d'un tableau de caractères. Pour cela, j'ai récupéré le nombre de lignes et de colonnes du fichier "grid.txt". J'ai ensuite pu obtenir la taille de la grille (colonnes×lignes), allouer de la mémoire à un tableau de caractères avec la fonction malloc et initialiser le tableau que j'ai nommé plateau. Je récupère ensuite les éléments importants comme la position du personnage à l'instant t=0, les positions des différentes sorties et la position du feu à l'intant t=0. Une fois que cela est fait, j'ai décidé de transformer mon tableau en un tableau d'entiers car cela est pour moi plus facile à gérer. Donc je crée un nouveau plateau auquel j'alloue de la mémoire de même taille que le tableau de caractères. Voici comment chaque case est transformé :

- '.' → 0
- '♯' → 1
- 'S' → 2
- 'E' → 3
- 'F' → 4

I.2 Lee Algorithm

Après l'obtention de cette nouvelle grille, je mets en place l'algorithme de Lee. Cet algorithme démarre d'une position de départ noté 0 puis note tous les voisins par 1. Il se place ensuite sur les voisins notés 1 et notes tous les voisins de 1 par 2. Si la case 1 n'a pas de voisin et on ne peut plus avancer, alors il n'existe pas de chemin. A noter qu'en même temps que l'algorithme de Lee tourne à chaque pas de temps, le feu se propage. Après beaucoup de reflexion, j'ai calculé que le feu ne brûle pas sur le chemin si le personnage est déjà passé sur la case. La case peut brûler mais le personnage ne repassera pas sur la case. Donc cela ne sert à rien d'enflammer la case. Voici un exemple pour comprendre l'alogithme de Lee.

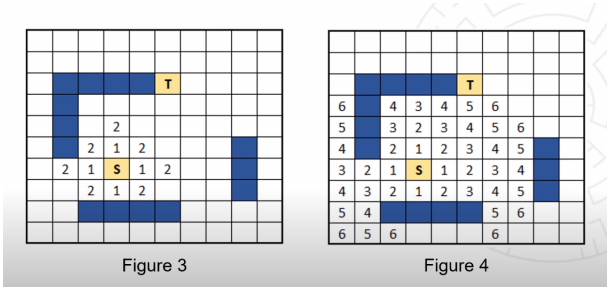

Figure 3 Figure 4

Sur la figure 3, on est parti de la case S. On ajoute ensuite 1 sur tous ses voisins. Puis on observe que sur toutes les cases 1, on met 2 sur les voisins accessibles. Et on continue jusqu'à arriver sur la case T. On voit que cette case est bien atteinte sur la figure 4.

La suite de l'algorithme consiste à partir de l'arrivée et de parcourir le minimum des points. Lorsque l'on parcourt le chemin, on va de voisin en voisin en choisissant la case avec le minimum à chaque fois. Une fois que l'on arrive sur la case de départ alors, on a récupéré le chemin avec toutes les positions. Il ne reste qu'à inverser le chemin, c'est à dire aller de la position de départ à la postion d'arrivée. La seule option que j'ai rajouté à cette algortihme c'est que juste avant de calculer les voisins, le feu se propage. Le feu se propage juste avant que l'algorithme ne se lance. C'est pour moi une des versions la plus simple pour résoudre le problème. J'ai du cependant chercher comment résoudre le problème et c'est cet algorithme qui m'a paru le plus performant.

Il m'a seulement fallu rajouter une option qui est celle du déplacement du feu. De plus, j'ai décidé de commencer mon chemin avec l'entier 5 car les entiers avant sont réservés dans la grille à des positions spécifiques comme un terrain libre ou un arbre par exemple. Donc le chemin va de la position de départ $2 \rightarrow 5 \rightarrow 6 \rightarrow 7$ ..... $\rightarrow 3$. Car 3 est la position de sortie du personnage.

Pour lancer le programme, il faut lancer dans le terminal la commande "gcc -o main main.c". Il faut avant de lancer la commande se placer dans le dossier ou se trouve le fichier "main.c". Une fois que cette commande est lancé, dans le même terminal, lancer la commande "./main grid1.txt" si vous voulez vérifier le chemin pour le fichier "grid1.txt". si vous désirez une autre grille, il suffit de la remplacer dans la ligne de commande directement. Il faut aussi modifier au sein même du code le répertoire dans lequelle se trouve tous vos grilles de jeu.

II Explication du choix

Je suis d'accord que mon choix peut paraître moins adapté qu'à d'autres langages comme python qui lui par exemple peut facilement implémenter des tableaux.

J'ai choisi le langage C car j'avais déjà réalisé des jeux similaires lors de ma scolarité. L'année dernière, par exemple, nous avions programmé un jeu de la vie. Ce jeu reposait aussi sur le concept de grille et c'est pour cette raison que j'ai décidé de choisir ce langage pour réaliser ce test.

Pour moi, l'un des plus gros défauts de C (et c'est aussi ce qui en fait sa force) est le fait de devoir utiliser des pointeurs et de ne pas pouvoir créer des tableaux dynamiques facilement. En effet, pour une fonction en python on peut en une boucle initialiser un tableau. En C, il faut utiliser 2 boucles. L'une pour récupérer la taille du tableau, ensuite on alloue de la mémoire au pointeur avec la fonction malloc puis on peut ensuite initialiser le tableau dans une nouvelle boucle. C'est une perte de temps quant à la mémoire. Mais comme le langage C est fait ainsi, je ne pense pas que cette perte soit très importante.

Une autre raison qui m'a poussée à utiliser le langage C est que je suis plus à l'aise avec la programmation modulaire et même la programmation à l'aide de fonctions que la programmation orientée objet. La plupart des projets que j'ai réalisé ont été en python mais pour ce projet, j'ai eu peur de me perdre avec les librairies du langage python. C'est pour cette raison que je suis rester sur le langage C. De plus, comme Python est un langage complet, je peux faire de la programmation à la fois avec des fonctions mais aussi avec des classes et cela m'aurait perdu.

Conclusion

Pour conclure, je pense que ce test m'a été d'une grande utilité. Il m'a permis de progresser quant à la création de jeu et de recherche de trajectoire. J'y ai en revanche passer beaucoup de temps. Je pense avoir mis 9h à coder le jeu en entier, environ 2h à rédiger le rapport ainsi que 3 heures sur la recherche de développement. Je pense qu'à cela je peux rajouter la recherche du bon algorithme pour trouver le chemin entre la position de départ et la sortie. Je pense avoir mis une demi heure à trouver cette algorithme et l'adapter à mon problème. Je pense donc avoir passer environ 14h30 à faire tout le test. Je ne me rends pas compte pour savoir si cela est peu ou non. Je pense aussi avoir passer du temps à me perdre dans le code, notamment lors de la programmation de l'algorithme de Lee.

Je voulais vous remercier aussi de m'avoir laissé le temps de réaliser mon test jusqu'au bout. J'ai mis plus de temps que je ne le pensais car un nombre de projets conséquents est aussi tombés pour la fin de notre semestre à l'école.