Compte-rendu du projet d'informatique

Maxence Blanc

3 mars 2017

Table des matières

I Avant-Propos

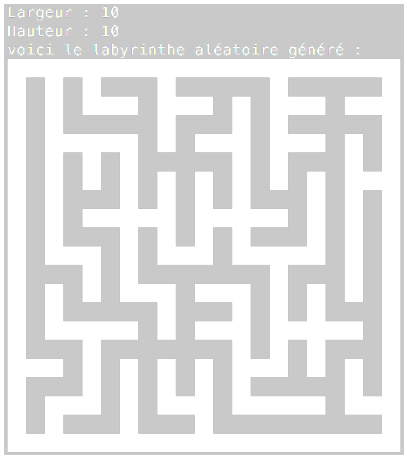
L'objectif de ce projet a été de créer à partir de dimensions entrées, un labyrinthe généré de manière aléatoire et de le résoudre.

Le labyrinthe créé est un labyrithe parfait, c'est à dire que chaque point est relié à tout autres par un unique chemin. J'ai choisi de rester sur ce type de labyrinthe pour des raisons de temps mais le programme est codé de manière à ce qu'il soit simple de générer et résoudre un labyrinthe imparfait.

Note : pour des questions de temps je considère que l'utilisateur ne tente pas de piéger le programme, je suis conscient qu'une multitude d'amélioration est possible pour protéger le prgramme mais bien que simples à rajouter, elles sont nombreuses et chronophages comme dit en classe.

II Création d'un labyrinthe

La première étape de ce projet est de générer un labyrinthe à partir de ses dimensions :



Generation de la matrice

Pour commencer, on a besoin d'une matrice, un tableau en 2 dimensions qui pourra contenir des valeurs correspondantes aux murs et cellules vides.

Afin d'être certain que l'utilisateur entre des données acceptables pour les dimensions du labyrinthe, on appelle une fonction de verification, qui n'accepte que les entiers.

Puis on créé la matrice, qui est en fait une liste de listes, et on lui ajoute des valeurs en commençant par 0 toutes les 2 cases. On créé ainsi des cellules (isolées par des murs de valeur -1) que l'on va relier par la suite.

Formatage de la matrice

Pour la suite, j'ai trouvé interressant d'établir une liste de tout les murs destructibles. C'est relativement simple en prenant toutes les colonnes puis en fonction, si une colonne est impaire on prend les lignes paires et inversement. J'utilise les range() de manière à éviter les cases qui ne m'interressent pas en prenant des pas de 2. Par exemple :

```
        for y in range(2, len(matrice)-1, 2):
                liste.append((x,y))
```

Enfin, on passe par une fonction qui, une fois le mur a détruire choisi dans la liste, récupère les valeurs autours et, si ces dernieres sont differentes, appelle une fonction récursive pour que la plus petite valeur se propage. On élimine des murs un nombre bien précis de fois : il s'agit de hauteur $\times$ largeur - 1. Une fois terminé, on aura obtenu un labyrinthe parfait. Mais encore faut il pouvoir l'afficher proprement.

Affichage du Labyrithe

Pour afficher le labyrinthe, on appelera une fonction qui va parcourir la matrice et en fonction de la valeur rencontrée, on affiche un certain caractere. Afin de rendre l'affichage plus attirant, j'ai choisi de colorer les caracteres : Blanc pour les murs et jaune pour le chemin le plus court.

III Recherche du plus court chemin

Positions Entrée/Sortie

Une fois encore, on va appeler deux fonctions chargées de vérifier que l'utilisateur indique au programme des coordonnées correctes pour la position de l'entrée et de la sortie.

Recherche de la sortie

La résolution est une partie très interressante de ce projet. On commence par ce servir d'une fonction qui va placer le premier point du chemin dans le labyrinthe.

Puis on appelle la fonction d'exploration, qui va partir du premier point et avancer dans toutes les directions possibles et s'arreter des qu'elle trouve la sortie, retournant ainsi une matrice contenant tout les chemins explorés. La fonction n'est pas récursive (bien qu'elle m'en semble très proche) mais néanmoins très rapide.

J'ai créé un système de liste qui contient les coordonnées de tout les points explorés au temps t-1 donc je ne re-parcours pas en permanence toute la liste. Cette liste se met à jour après chaque étape. Cette fonction s'arrete forcément une fois que le chemin le plus court a été trouvé que le labyrinthe soit parfait ou non.

Preuve : A chaque étape on avance d'une case, quelque soit la direction et à cette case on affecte une valeur plus grande que pour les cases explorées à l'étape d'avant. Cette valeur

correspond exactement à la longueur du chemin depuis le départ De plus le programme s'arrete dès qu'il a trouvé la sortie.

Ainsi, par l'absurde, considérons que le premier chemin trouvé n'est pas le plus court. Or d'après l'hypothèse initiale, on passe à l'étape suivante et on augmente la valeur de la longueur du chemin de 1. Par conséquent, tout chemin trouvé après le premier est plus long que celui-ci. Le premier chemin trouvé est donc bien le plus court possible.

Finalement, on appelle une fonction qui va retrouver le chemin la plus court à partir de la matrice contenant toutes les explorations. Pour ce faire, on par de l'arrivée et on rebrousse chemin en cherchant une case dont la valeur est inferieure à celle de la case actuelle.

Et on oublie pas d'afficher le labyrinthe résolu.

IV Tests complémentaires

J'ai choisi d'étudier l'efficacité de 2 fonctions à l'aide des outils fournis par python : la fonction qui propage les valeurs lorsque l'on casse les murs : propagation(), et celle qui explore les differents chemins possibles : recherche(). Ce sont d'après moi, les plus complexes du projet.

Grâce à la bibliothèque time, nous pouvons évaluer le temps d'éxecution des fonctions de la manière suivante. Biensur, on rajoute du code mais cela permet d'avoir une idée de la vitesse d'execution de la fonction.

```
import time
start_time = time.time()
fonction()
print("%s secondes" % (time.time() - start_time))
```

propagation() Pour un labyrinthe de 100 par 100, elle met environ 0,2 seconde à générer le labyrinthe.

recherche() Pour un labyrinthe de 100 par 100, elle met environ 0,02 seconde à trouver la sortie avec un point en (0,1) et l'autre en (200,199).

V Bilan du projet

J'ai rapidement décider d'organiser le programme et mon travail en 3 parties : tout d'abord, la création du labyrinthe et son affichage. Puis l'exploration des chemins possibles. Et enfin résolution ou l'on retrouve un unique chemin. Tout au long de ce projet j'ai cherché à rendre mon programme le plus modulable possible pour pouvoir le remodeler facilement en cas de problème et y ajouter des fonctionnalités de manière simple.
Par ailleurs j'ai trouvé ce projet intéressant tout d'abord pour l'algorithme de génération de labyrithe parfait, mais aussi la façon de le résoudre. Et je pense poursuivre, j'ai déja imaginé comment le modifier pour qu'il soit en 3 dimensions, qu'il demande à l'utilisateur combien de murs celui ci veut casser en plus (en faisant un labyrithe imparfait, qu'il peut deja résoudre.), et pourquoi pas utiliser pygame pour le rendre plus intéractif.