LONGUEPEE Lubin
BAROUK Aidan
LDD2 IM
Groupe 2
2023/2024

Projet : Labyrinthe

17 janvier 2024

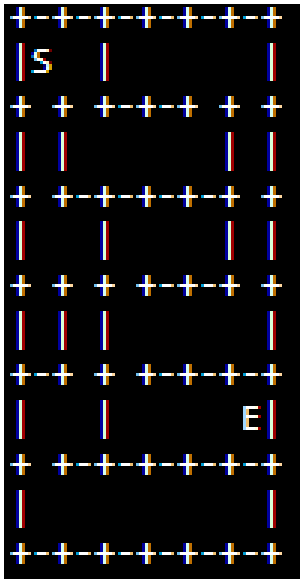
Répartition du travail

Nous avons fait le choix de travailler à deux sur les mêmes choses durant les séances de TP guidés pour pouvoir bien comprendre les enjeux du sujet et poser les bases du labyrinthe ensemble pour ne pas partir dans des directions qui auraient pu entrer en conflits. Une fois la dernière séance de TP guidée finie, nous en étions à la génération et à la résolution des labyrinthes. Nous avons choisi de continuer chacun de notre côté pour pouvoir avancer plus rapidement : Aidan a travaillé sur la résolution tandis que Lubin a travaillé sur la génération. Pour finir, nous avons pris le temps de faire la partie d'affichage avec ASCII Art ensemble afin de terminer ensemble pour par la suite résoudre les derniers bugs éventuels et s'informer mutuellement des changements de code effectués. Aidan, ayant eu des problèmes pour configurer gitlab sur sa machine, a envoyé le code des fonctions qu'il a écrites à Lubin afin qu'il les ajoute dans son fichier. C'est pour cela que tous les commit ont été fait par Lubin uniquement (preuves du travail en groupe disponibles si besoin).

Présentation générale de la structure de données

Pour représenter le labyrinthe, nous avons instinctivement pensé à utiliser un tableau à deux dimensions (2D array). Nous avons donc dû créer un type structuré contenant une grille (2D array) ainsi que le nombre de lignes et de colonnes de notre labyrinthe pour pouvoir faire des opérations dessus. Pour représenter chaque cellule du labyrinthe (ie le type des cases du 2D array) nous avons choisi de créer un nouveau type structuré contenant 3 informations : les coordonnées x

et y de la cellule en question, et une case. Une case étant également un type structuré permettant de savoir de quel type est la cellule en question (Vide, Mur, Entrée, Sortie ou Chemin). Les fichiers contenant les labyrinthes étant en .laby, nous avons dû créer des fonctions pour convertir un fichier .laby vers un labyrinthe représenté par notre type labyrinthe. Pour convertir un fichier textuel .laby en labyrinthe, il a suffit de lire tous les caractères et de les convertir vers le type par lequel ils doivent être représentés (exemple: un “+”, un “|” ou un “-” représentant un Mur, un “ “ représente du vide...). Nous avons également dû réfléchir à la manière permettant d’afficher un labyrinthe. Nous avons donc dû créer une fonction permettant d’afficher une case en particulier du labyrinthe, en prenant en compte les contraintes d’affichage puis nous avons appelé cette fonction sur toutes les cellules du 2D array du labyrinthe.



Affichage d’un labyrinthe (conversion du fichier textuel .laby en labyrinthe puis affichage de ce labyrinthe)

Présentation d’une difficulté

Nous allons présenter les difficultés que nous avons eu avec la fonction de génération de labyrinthe. La première difficulté à laquelle nous avons fait face a été de représenter les directions possibles autour d’une case. Pour cela nous avons dû créer un nouveau type représentant la direction (Haut, Bas, Gauche,

Droite). Nous avons également rencontré un autre problème : parfois, quand on génère un labyrinthe de taille assez faible, les coordonnées de la sortie et l'entrée générées aléatoirement sont les mêmes. On plaçait donc en premier l'entrée, et on écrasait ensuite cette entrée en y plaçant la sortie, ce qui rendait notre labyrinthe invalide (cf image ci-dessous).

```
(base) lubin.longuepee@universite-paris-saclay.fr@jupyterhub:~/Ocaml$ ./maze.exe random 5 5 200102
+-+-+-+-+-+
|   |     |
+ + + +-+ +
| | |   | |
+ + + +-+ +
| | | |S| |
+ + + + + +
| | | | | |
+ + + + + +
| |   |   |
+-+-+-+-+-+
```

Pour résoudre ce problème, nous avons dû changer notre code en rajoutant une fonction auxiliaire pour que l'on génère les coordonnées de l'entrée en premier, et que l'on génère ensuite les coordonnées de la sortie jusqu'à ce qu'elles soient différentes des coordonnées de l'entrée (cf image ci-dessous).

```
let (ligne_entree, colonne_entree) = (Random.int (lignes / 2)) * 2 + 1, (Random.int (colonnes / 2)) * 2 + 1 in

let rec position_aleatoire_differentes_entree () =
  let ligne = (Random.int (lignes / 2)) * 2 + 1 in
  let colonne = (Random.int (colonnes / 2)) * 2 + 1 in
  if (ligne, colonne) = (ligne_entree, colonne_entree) then
    position_aleatoire_differentes_entree ()
  else
    (ligne, colonne)
in

let (ligne_sortie, colonne_sortie) = position_aleatoire_differentes_entree () in
```

Cela a résolu le problème (cf image ci-dessous)

```
(base) lubin.longuepee@universite-paris-saclay.fr@jupyterhub:~/Ocaml$ ./maze.exe random 5 5 200102
+-+-+-+-+-+
|     |E  |
+ + + +-+ +
| | | |   |
+ + + + +-+
| | | |S| |
+ + + + + +
| | | | | |
+ + +-+ + +
| |       |
+-+-+-+-+-+
```

Mais résoudre ce bug en a entraîné un autre. En effet, on génère maintenant des nouvelles coordonnées pour la sortie jusqu'à ce qu'elles soient différentes de celles de l'entrée. Que se passe-t-il si on essaye de générer un labyrinthe de taille 1x1 ? Et bien le programme tourne à l'infini puisqu' il est impossible de trouver deux paires de coordonnées différentes pour l'entrée et la sortie. Nous avons donc dû ajouter une ligne de code au début de notre fonction de génération pour vérifier que l'on essaye pas de générer un labyrinthe de taille 1x1 (cf image ci-dessous).

```
if lignes = 1 && colonnes = 1 then failwith "Le labyrinthe ne peut pas être de taille 1x1";
```

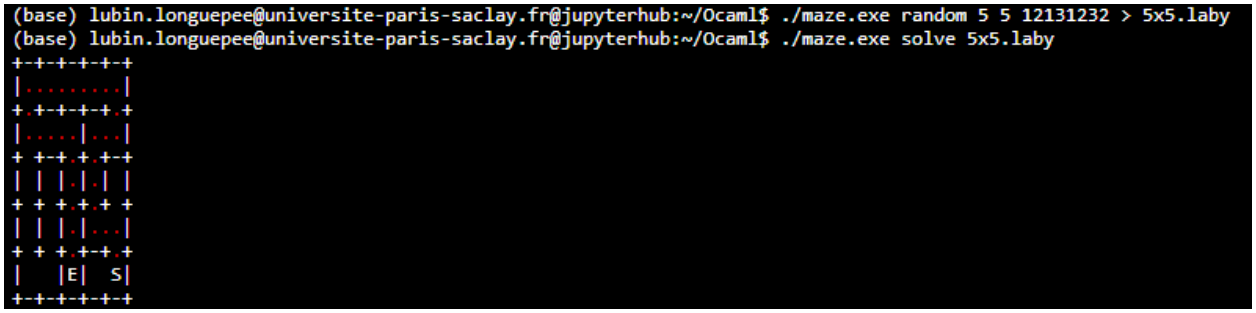
En exécutant notre code plusieurs fois, nous avons fait un autre constat, notre entrée et notre sortie n'étaient jamais placées sur la première ou la dernière ligne. Le problème venait d'une erreur sur les bornes de Random.int lors de la génération de coordonnées aléatoires.

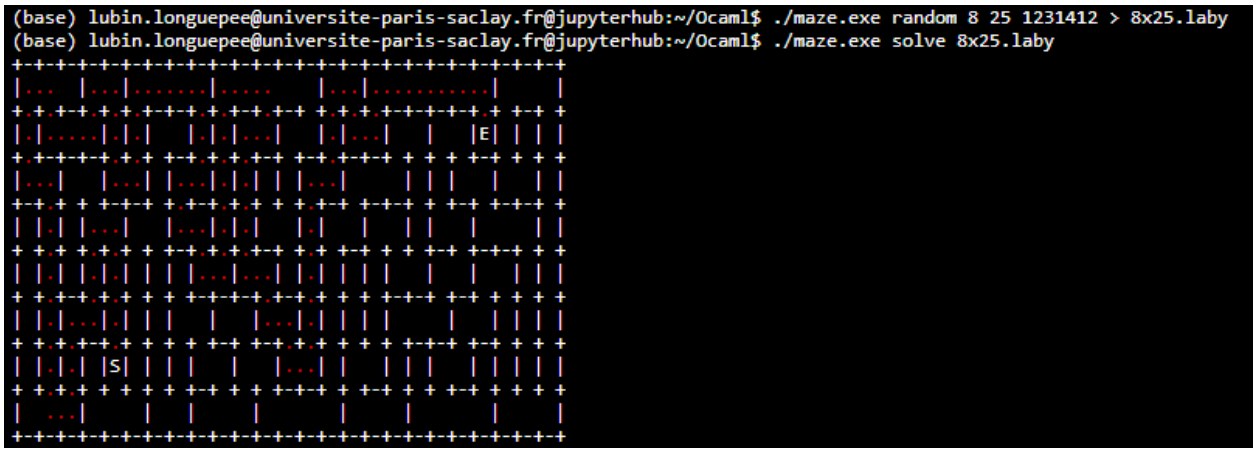
Description des tests

Pour tester notre code, il faut tester 3 situations:

-La première est de tester que l'affichage fonctionne. Pour cela, le plus simple est d'afficher plusieurs labyrinthes et de vérifier manuellement que l'affichage correspond au texte dans le fichier. Nous avons donc essayé d'afficher des labyrinthes de toute taille et nous avons vu que l'affichage était correct.

-La deuxième situation à tester est la résolution d'un labyrinthe. Pour cela, le plus simple est encore de le faire manuellement. En effet, on ne cherche pas à trouver le chemin le plus optimal mais seulement un chemin qui fonctionne et qui relie bien l'entrée à la sortie. On a donc appelé la fonction de résolution sur des labyrinthes de toute taille et avons constaté que l'algorithme trouvait toujours un chemin qui fonctionnait si le labyrinthe de départ était valide (cf images ci-dessous, les points rouge représentent le chemin. Les fichiers 5x5.laby et 8x25.laby sont dans le dépôt git).





-La troisième situation à tester est la génération. Pour cela, nous avons créé une fonction qui prend en argument trois entiers n, l, et L et qui génère n labyrinthes de taille l*l2 et qui vérifient qu'ils sont tous valides. Pour vérifier cela, nous avons créé une fonction est Valide qui prend en argument un labyrinthe et qui vérifie que celui-ci est valide (ie il possède une entrée et une sortie différentes et unique, il est bien entouré de murs et il n'y a pas de murs ou de chemin en dehors des murs extérieurs du labyrinthe). Nous avons également vérifié que si on exécute deux fois la fonction avec la même seed, le labyrinthe obtenu est le même, tandis que si on exécute deux fois la fonction avec des seed différentes, le labyrinthe obtenu est bien différent du premier.

Description du type d'affichage choisi

Nous avons choisi le type d'affichage amélioré sous forme d'ASCII Art car cet affichage avancé correspondait plus à ce que l'on avait envie de créer. Pour afficher le labyrinthe, la structure de la fonction de conversion de labyrinthe vers un affichage est la même. Toutefois, nous avons choisi de représenter les murs

grâce aux caractères suivants : ┼,┐ , ┌,┘ , └,─, │ ,┤ , ├,┴,┬. Pour utiliser ces caractères, nous avons dû tester, pour chaque mur, si d'autres murs étaient présents au-dessus, en dessous, à gauche et/ou à droite de lui, pour utiliser le caractère approprié pour ainsi relier chaque mur à ses éventuels voisins.

Nous avons choisi de représenter l'entrée et la sortie grâce à des emojis (🚪 pour l'entrée, 🚩 pour la sortie). Voici l'affichage d'un labyrinthe en version ASCII Art:

Nous avons également choisi de changer l'affichage des chemins. Dans la version classique, un chemin est représenté par une succession de points rouges. Dans la version avancée, nous avons choisi de représenter les morceaux de chemins avec des caractères similaires à ceux des murs, mais de taille plus petite pour que l'affichage du chemin ne soit pas trop gros et que le résultat soit esthétique. Voici un exemple d'affichage de résolution de labyrinthe: