Projet – Génération de labyrinthe

Dans ce projet vous devrez implémenter les algorithmes nécessaires pour générer, résoudre et sauver/charger un labyrinthe. Le code sera à réaliser en assembleur MIPS (code source .s), et utilisera l'émulateur `MARS` pour l'exécution.

Le projet est à réaliser en **binôme ou seul** et devra être **rendu sur la plateforme `Moodle` avant le vendredi 09 décembre à 23h55**. Le code source se doit de respecter la spécification ci-dessous.

1 Spécification

Le projet devra impérativement être décomposé en fonctions et le code doit être **commenté** comme il se doit.

Votre code contenu dans un fichier assembleur `.s` devra demander à l'utilisateur de faire un choix entre les deux options suivantes :

— Génération d'un nouveau labyrinthe. La taille du labyrinthe ($N \times N$) sera demandée à l'utilisateur. Une fois ce labyrinthe généré, il sera sauvegardé dans un fichier texte en respectant la syntaxe décrite dans la section Représentation du labyrinthe.

— Résoudre un labyrinthe. Le labyrinthe sera lu dans un fichier dont le nom sera demandé à l'utilisateur. La solution sera sauvegardée dans un autre fichier ayant pour préfixe le nom précédemment entré concaténé avec ".resolu" (exemple : laby.txt -> laby.txt.resolu).

Chaque binôme devra également écrire un **bref** rapport contenant les explications des choix effectués (structure de donnée, qui a fait quoi ...).

Le rendu prendra la forme d'une archive nommée `Nom1_prénom1_Nom2_prénom2.tar.gz`, qui contiendra : le fichier `.s`, ainsi que votre rapport.

Cette archive sera déposer avant le vendredi 09 décembre à 23h55 sur la plateforme `Moodle`.

Ce projet sera à présenter par chaque binôme lors de vos séances respectives de TP durant la semaine du 12 décembre.

2 Représentation du labyrinthe

Le labyrinthe sera représenté sous format texte de la manière suivante :

```
05
29 03 09 03 43
11 10 10 10 10
10 10 10 10 10
10 10 10 10 10
12 04 06 12 06
```

Le chiffre sur la première ligne indique la taille du labyrinthe. Ici la valeur 5 indique un labyrinthe de taille 5×5 donc codé sur cinq lignes et cinq colonnes.

Chaque case a une valeur codé sur un caractère (8 bits). Chaque bit de cet entier a la correspondance suivante :

B7	B6	B5	B4	B3	B2	B1	B0
Ignoré	Chemin solution	Fin	Départ	Mur à gauche	Mur en bas	Mur à droite	Mur en haut

2.1 Décodage

Décodons la première ligne ensemble :

```
  A=29     B=03     C=09     D=03     E=43
00011101 00000011 00001001 00000011 00101011
```

On peux donc reconstituer les murs et attributs en décodant les entiers :

— La case `A` a un mur en haut (B0), en bas (B2), à gauche (B3) et démarre le labyrinthe (B5)

— La case `B` a un mur en haut (B0), et à droite (B1)

— La case `C` a un mur en haut (B0), et à gauche (B3)

— La case `D` a un mur en haut (B0), et à droite (B1)
— La case `E` a un mur en haut (B0), à droite (B1), à gauche (B3) et fini le labyrinthe (B5)
Ce qui peut se représenter en ascii art de la manière suivante :

```
@@@@@@@@@@@@
@A  B@C  D@E@
@@@ @ @ @ @
```

Et si on recommence pour toute les cases du tableau on obtient le labyrinthe suivant (X le départ et O la fin) :

```
@@@@@@@@@@@@
@X   @    @O@
@@@ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @
@     @    @
@@@@@@@@@@@@
```

Le script nommé «`print_maze.sh`» prends le nom d'un fichier contenant un labyrinthe et l'affiche de la même manière que ci-dessus. Il vous servira à confirmer que votre algorithme construit bien un labyrinthe «normalement constitué».

3 Génération et résolution de labyrinthe

3.1 Génération

L'algorithme utilisé est celui de la recherche en profondeur d'abord à la manière d'un graphe. La racine est la case correspondant à l'entrée du labyrinthe. Le graphe se construit de manière aléatoire en explorant des cases qui n'ont pas encore été visitées.

Données : Un labyrinthe dont tous les murs sont fermés.
Initialisation : Choisir aléatoirement deux cases sur des bords opposés et en considérer une comme la case de départ et l'autre comme la case de fin.
Initialisation : Faire de la case de départ la case courante et la marquer comme visitée
tant que *la case courante n'est pas la case de départ* `OU` *la case de départ a des voisins non visités* **faire**
  si *la case courante a au moins un voisin non visité* **alors**
    Choisir une case aléatoirement parmi les voisins non visités
    Détruire le mur entre la case courante et celle choisie
    Empiler la case courante
    Faire de la case choisie la case courante et la marquer comme visitée
  sinon
    Dépiler une case de la pile et en faire la case courante
  fin
fin

Algorithme 1 : Génération de labyrinthe

3.2 Résolution

La résolution du labyrinthe peut utiliser un algorithme similaire à la génération. On part de la case de départ et on visite tous les chemins possibles jusqu'à tomber sur la case de sortie. Une fois sur la case de sortie, la pile contient le chemin parcouru depuis le départ jusqu'à la fin. Il suffit alors de dépiler et marquer les cellules comme faisant partie du chemin résolvant le labyrinthe.

3.3 Outils mis à disposition

Nom du fichier	Description
Alea.s	Contient le code générant des nombres pseudo-aléatoires à utiliser pour votre projet
print_maze.sh	Prend en paramètre un fichier contenant un labyrinthe et l'affiche en ascii art
laby.txt	Un exemple de labyrinthe non résolu (après génération)
laby.txt.resolu	Le labyrinthe ci-dessus après résolution