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** et devra être **rendu sur la plateforme `Moodle` avant le vendredi 30 Octobre à 23h55**. Le code source se doit de respecter la spécification ci-dessous.

1 Préface

Dans ce projet, il sera parfois nécessaire de demander à l'utilisateur de rentrer des nombres, d'effectuer des choix, et de taper des noms de fichier.

Ces choix peuvent être effectués comme nous le faisions en TP: par le biais d'appels système (les syscalls 5,6,7,8), mais un point bonus sera accordé aux étudiants ayant réussi à implémenter de vrais arguments en ligne de commande.

Mars supporte l'option `pa` ("program arguments", sans tiret). Tout ce qui suit cette option sera accessible depuis l'assembleur.

Vous pouvez vous référer à cette page de documentation en savoir plus.
`http://courses.missouristate.edu/kenvollmar/mars/Help/MarsHelpCommand.html`

```
java -jar Mars4_5.jar 100000 projet.s pa 1 5 laby.txt
# Exécute projet.s avec un maximum de 100 000 étapes, mode 1, taille 5, écrit dans laby.txt
```

2 Spécification

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

Votre code sera contenu dans un fichier assembleur `.s`.

Votre programme devra demander à l'utilisateur de faire un choix entre les deux modes d'exécution :

1. Génération d'un nouveau labyrinthe. La taille du labyrinthe ($N \times N$) sera demandée à l'utilisateur.
2. Résoudre un labyrinthe.

Le choix de l'utilisateur se fera avec les chiffres 1 ou 2. En mode génération, la taille est un entier naturel.

Chaque binôme devra également écrire un rapport d'environ deux pages (expliquant les choix effectués durant la conception de votre projet, vos structures de données, qui a fait quoi, les difficultés rencontrées...).

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

Cette archive sera à déposer avant le vendredi 30 Octobre à 23h55 sur la plateforme `Moodle`.

Ce projet sera à présenter par chaque binôme lors de votre dernière séance de TP d'architecture des ordinateurs.

3 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 codée 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

3.1 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».

4 Génération et résolution de labyrinthe

4.1 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

Il n'est pas obligatoire que votre programme produise un fichier de sortie lors de la génération/résolution, un affichage dans la sortie standard est suffisant.

Le simulateur Mars supporte l'exécution de votre code directement depuis la ligne de commande. Il est donc possible de rediriger la sortie de votre projet dans un fichier, comme pour n'importe quelle autre commande.

`https://courses.missouristate.edu/kenvollmar/mars/Help/Help_4_5/MarsHelpCommand.html`

```
java -jar Mars4_5.jar 100000 projet.s > laby.txt
# Exécute projet.s avec un maximum de 100 000 étapes, stocke sa sorties dans laby.txt
```

Il est également possible pour un point bonus de programmer en assembleur l'écriture dans un fichier. Le nom du fichier sera alors demandé à l'utilisateur (par l'entrée standard, ou par argument CLI pour ceux qui ont choisi cette voie).

4.2 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.

Le labyrinthe doit être lu soit depuis l'entrée standard, soit directement dans un fichier pour un point bonus. Le nom du fichier sera demandé à l'utilisateur par l'entrée standard ou par argument CLI.

Le labyrinthe doit pouvoir être entré sous le format défini dans la section **??**. Aussi, si il ne peut en aucun cas être passé sous la forme d'arguments en ligne de commande, certains labyrinthes sont beaucoup trop volumineux.

Si le labyrinthe est lu depuis l'entrée standard, il est possible d'écrire chaque entier un par un, à la main, dans votre terminal. C'est évidement déconseillé. La méthode intelligente est de faire usage de votre ligne de commande en imitant l'exemple ci-dessous:

```
(printf '2\n' && cat 'laby.txt') | java -jar Mars4_5.jar 100000 projet.s
# Exécute projet.s avec un maximum de 100 000 étapes
# Ecrit d'abord "2" puis "\n" (retour chariot) dans son entrée standard afin de déterminer le mode
# Ecrit ensuite l'entièreté du contenu du fichier laby.txt
```

4.3 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