Devoir1 : Labyrinthe Invisible

À faire en équipe de 2 personnes

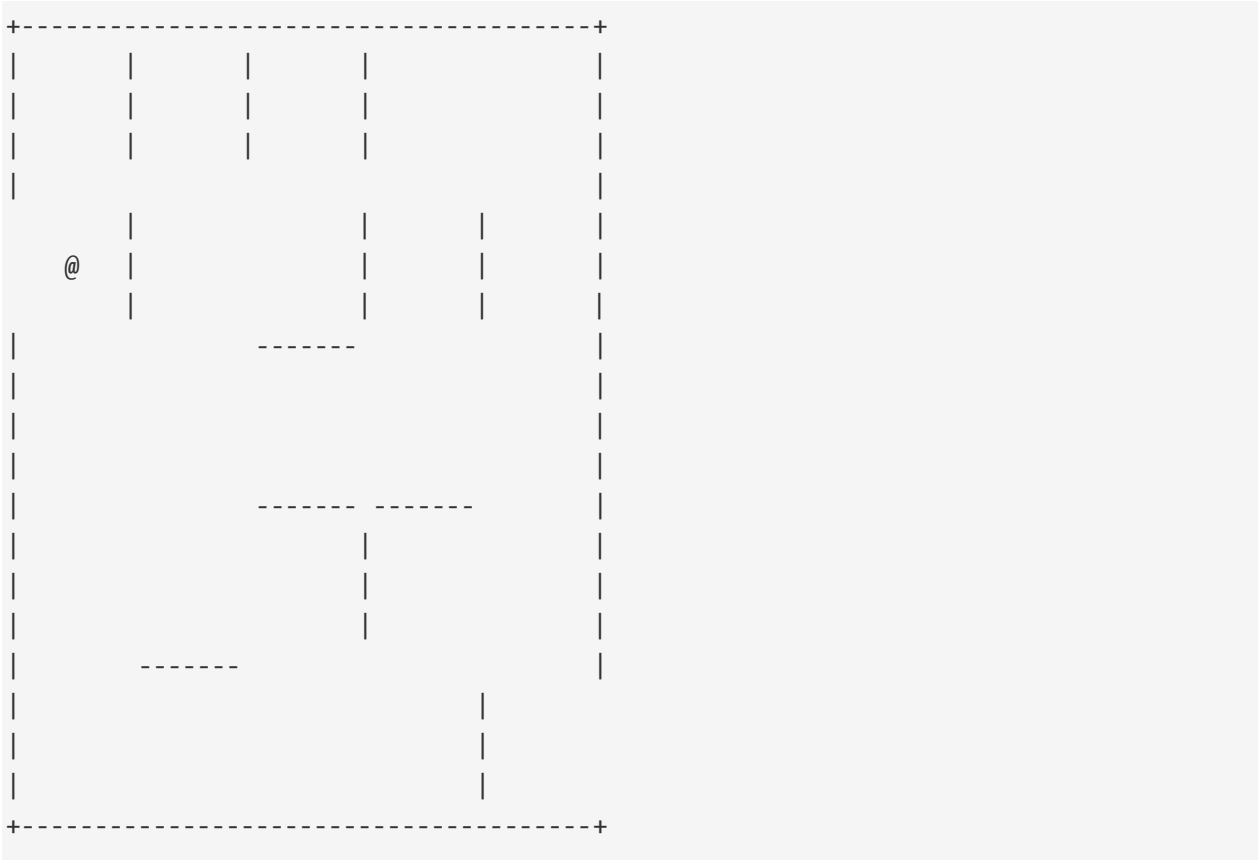
Date de remise : 15 novembre 2017 (23:55)

Principe du jeu

Le jeu de labyrinthe invisible est un jeu de mémoire qui fonctionne selon le principe suivant:

Au début du jeu, l'ordinateur crée un labyrinthe au hasard. Il crée au hasard une entrée par la gauche et une sortie par la droite. Des murets sont construits à l'intérieur. Un personnage (représenté par @) est placé juste à l'entrée du labyrinthe dans l'extrême gauche du labyrinthe.

Il vous affiche ce labyrinthe pendant quelques secondes à l'écran. Exemple pour un petit labyrinthe de 5x5 cases:



Au bout de ces quelques secondes, tous les murets intérieurs du labyrinthe deviennent invisibles, ce qui donnera, pour notre exemple, l'affichage ci-dessous:

```
+--------------------------------------+
|                                      |
|                                      |
|                                      |
|                                      |
                                       |
    @                                  |
                                       |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|
|
|
+--------------------------------------+
Il vous reste 5 vies sur 5.

Quelle direction souhaitez-vous prendre?
(droite: d; gauche: g ou s; haut: h ou e; bas: b ou x)
```

En plus l'ordinateur affiche combien de vies il reste à votre personnage.

C'est alors à vous de guider votre personnage vers la sortie au travers du labyrinthe aux murs invisibles (et électriques), en choisissant de vous déplacer à chaque fois d'une case vers la droite, la gauche, le haut et le bas.

Il est interdit de sortir par l'entrée. Ainsi, le déplacement vers la gauche dans cette configuration initiale est invalide.

A chaque fois que vous choisissez une direction qui vous amène à toucher un muret, le muret devient visible de façon permanente, mais vous êtes électrocuté, et vous perdez une vie.

Ainsi, dans notre exemple, si on choisissait d'aller à droite (d), on obtiendrait l'affichage suivant (notez que votre personnage n'a pas changé de case, puisque son mouvement a été bloqué par un muret. Notez aussi que le nombre de vies est passé à 4):

```
+--------------------------------------+
|                                      |
|                                      |
|                                      |
|                                      |
        |                             |
    @   |                             |
        |                             |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|
|
|
+--------------------------------------+
Il vous reste 4 vies sur 5.

Quelle direction souhaitez-vous prendre?
(droite: d; gauche: g ou s; haut: h ou e; bas: b ou x)
```

Si vous allez ensuite dans la direction bas, vous obtiendrez l'affichage suivant:

```
+--------------------------------------+
|                                      |
|                                      |
|                                      |
|                                      |
        |                              |
        |                              |
        |                              |
|                                      |
|                                      |
|   @                                  |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|
|
|
+--------------------------------------+
Il vous reste 4 vies sur 5.

Quelle direction souhaitez-vous prendre?
(droite: d; gauche: g ou s; haut: h ou e; bas: b ou x)
```

Si vous continuez avec bas, droite, droite, comme il n'y a aucun muret sur ce chemin, vous devriez voir l'affichage suivant:

```
+--------------------------------------+
|                                      |
|                                      |
|                                      |
|                                      |
        |                              |
        |                              |
        |                              |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                   @                  |
|                                      |
|                                      |
|
|
|
+--------------------------------------+
Il vous reste 4 vies sur 5.

Quelle direction souhaitez-vous prendre?
(droite: d; gauche: g ou s; haut: h ou e; bas: b ou x)
```

Si de ce point, vous essayez d'aller à droite puis vers le haut, comme il y a un muret à droite et aussi au dessus, vous perdrez 2 vies supplémentaires et votre plateau de jeu apparaîtra comme ceci:

```
+--------------------------------------+
|                                      |
|                                      |
|                                      |
|                                      |
|        |                             |
|        |                             |
|        |                             |
|                                      |
|                                      |
|                                      |
|                                      |
|                                      |
|                -------               |
|                       |              |
|                   @   |              |
|                       |              |
|                                      |
|
|
|
+--------------------------------------+
Il vous reste 2 vies sur 5.

Quelle direction souhaitez-vous prendre?
(droite: d; gauche: g ou s; haut: h ou e; bas: b ou x)
```

Si votre personnage épuise toutes ses vies (s'il reste 0 vies) alors vous avez perdu la partie, et l'ordinateur devra afficher le message suivant.

```
Vous avez perdu, vous avez épuisé vos 5 vies!
```

Si votre personnage parvient jusqu'à la sortie, l'ordinateur devra afficher:

```
Bravo, vous êtes parvenu jusqu'à la sortie en commettant seulement 3 erreurs.
```

Dans un cas comme dans l'autre, cela termine la partie, et l'ordinateur devra vous demander si vous voulez jouer une nouvelle partie ou non.

```
Voulez-vous jouer une nouvelle partie?
```

A quoi vous pouvez répondre non (auquel cas le programme se termine) ou oui (auquel cas une nouvelle partie recommence avec un nouveau labyrinthe).

Paramètres du jeu

Vous appellerez votre programme Laby.java et il devra prendre 5 paramètres sur la ligne de commande qui sont respectivement:

1. la hauteur du labyrinthe en nombre de cases
2. la largeur du labyrinthe en nombre de cases
3. la "densité" sous forme de probabilité de générer un muret (il s'agir d'un nombre entre 0 et 1: plus il est élevé, plus il y aura de murets, ainsi 0 génèrera un labyrinthe sans aucun muret intérieur (il y a quand même les murs du contour), et 1 génèrera un labyrinthe avec tous les murets intérieurs (voir exemple et explications plus bas).
4. le nombre de secondes d'affichage du labyrinthe complet avant qu'il ne devienne invisible.
5. le nombre de vies dont dispose votre personnage

Ainsi l'exemple ci-haut a été généré avec l'appel:

```
java Laby 5 5 0.30 7 5
```

Ces paramètres vous permettront de rendre le jeu plus intéressant, en contrôlant le niveau de difficulté, par exemple en créant un labyrinthe plus grand, plus ou moins dense, etc... A vous d'expérimenter des valeurs pour voir ce qui rend le jeu plus intéressant.

Si votre programme est appelé sans le bon nombre de paramètres, affichez un message d'erreur expliquant les paramètres nécessaires et suggérez à l'utilisateur comment démarrer le jeu avec des paramètres intéressants.

```
Nombre de paramètres incorrects.
Utilisation: java Laby <hauteur> <largeur> <densite> <duree visible> <nb vies>
Ex: java Laby 10 20 0.20 10 5
```

Format de l'affichage

Nous allons préciser ici le format de l'affichage. Comme nous venons de le voir, le labyrinthe a une certaine hauteur et une certaine largeur exprimée en nombre de cases ou de "murets".

Pour clarifier les choses, voici un exemple d'un labyrinthe de hauteur 3 et de largeur 5, avec tous les murets dessinés (sauf l'ouverture de la porte de sortie), tel que pourrait le montrer l'appel à

```
java Laby 3 5 1 100 5


+---------------------------------------+
|       |       |       |       |       |
|       |       |       |       |       |
|       |       |       |       |       |
|------- ------- ------- ------- -------|
|       |       |       |       |       |
|       |       |       |       |       |
|       |       |       |       |       |
|------- ------- ------- ------- -------|
        |       |       |       |       |
    @   |       |       |       |       |
        |       |       |       |       |
+---------------------------------------+
```

Le personnage est représenté par le caractère '@' et devra toujours être affiché au centre d'une case.

On choisira la convention selon laquelle le système de coordonnées a son origine en haut à gauche. Les grosses cases sont repérées par une paire de coordonnées (ligne, colonne) (en numérotées à partir de 0. Ainsi le personnage sur l'exemple ci-dessus est à la position (2,0).

Un labyrinthe possède un mur d'enceinte complètement fermé à l'exception d'une "entrée" sur le mur gauche et une "sortie" sur le mur de droite, à la hauteur d'une certaine case. Ces positions de portes sont choisies au hasard lors de la construction du labyrinthe.

À l'intérieur de l'enceinte, il y a deux sortes de murets: les murets verticaux sont construits avec des "briques" | et les horizontaux dessinés avec des "briques" -. Remarquez que les murets verticaux sont dessinés avec HMURET-1 briques et les horizontaux avec LMURET-1 briques (où HMURET et LMURET sont des constantes définies dans votre programme), de sorte qu'il y a toujours un petit interstice (espace) entre deux murets consécutifs (ce qui permet de plus facilement voir la limite entre les cases).

Classes à définir

Un tel labyrinthe sera représenté par une instance d'une classe **Labyrinthe**, qui possèdera notamment les attributs private suivants:

- un tableau à 2 dimension de char représentant le dessin du labyrinthe (tel qu'affiché dans les ex. ci haut).

  Il s'agira d'un tableau de char à 2 dimensions de taille hauteur*HMURET+1 par largeur*LMURET+1

Elle aura aussi les constantes static de classe définissant les distances en largeur et en hauteur entre 2 murets (exprimées en nombre de caractères).

```
    private static final int LMURET = 8;
    private static final int HMURET = 4;
```

La classe **Personnage** comportera notamment des propriétés indiquant sa position et son nombre de vies restantes.

Finalement, une classe **Laby** est le jeu à proprement dit.
Toutes les *propriétés* déclarées dans ces classes devront être déclarées comme *private*

Méthodes des classes

Méthodes à définir **dans la classe Labyrinthe**:

- un constructeur *Labyrinthe(int h, int w)* qui prend en paramètre hauteur et largeur (en nombre de cases) et initialise le tableau de caractères
- *creeTableau(int hauteur, int largeur)* prenant en paramètre hauteur et largeur et qui crée un tel tableau, et le remplit de caractères espace (' ').
- *effaceTableau()* qui remplit le tableau de caractères espace (' ')
- *dessineMurdEnceinte()* dessine un mur d'enceinte complètement fermé.
- *dessineOuverture(int j)* prend en paramètre la position verticale j (en nombre de cases) de l'ouverture de droite et la crée en effaçant la portion du mur d'enceinte correspondante.
- *dessineMuretVertical(int i, int j)* reçoit en paramètre la position i et j de la case où on veut dessiner un muret vertical (sur son bord gauche)
- *dessineMuretHorizontal(int i, int j)* reçoit en paramètre la position i et j de la case où on veut dessiner un muret horizontal (sur son bord haut)
- *aMuretAGauche(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord gauche de cette case, apparaît un muret ou un mur d'enceinte
- *aMuretADroite(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord droit de cette case, apparaît un muret ou un mur d'enceinte
- *aMuretEnHaut(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord en haut de cette case, apparaît un muret ou un mur d'enceinte
- *aMuretEnBas(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord en bas de cette case, apparaît un muret ou un mur d'enceinte
- *aEntreeAGauche(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord à gauche de cette case, c'est l'entrée du labyrinthe
- *aSortieADroite(int i, int j)* reçoit en paramètre la position i et j d'une case. Retourne *true* si sur le bord à droite de cette case, apparaît la sortie
- *dessinePersonnage(Personnage p)* **prendra en paramètre un objet de type Personnage** (voir ci-dessous), et le dessinera (avec un caractère '@') au centre de la case correspondant à sa position
- *effacePersonnage(Personnage p)* prendra en paramètre un objet de type Personnage (voir ci-dessous), et effacera (en y mettant un caractère ' ') le centre de la case correspondant à sa position

- fonction static *effaceEcran()* affiche *200* lignes vides pour évacuer vers le haut ce qui était auparavant visible sur le terminal.
- *affiche()* qui affiche le tableau de caractères à l'écran (suggestion, construisez une String contenant votre tableau puis affichez-la)
- *construitLabyrintheAleatoire(double densite)* une méthode qui prend en paramètre la "densite" et construit des murets aléatoirement (voir indication sur comment procéder plus bas), et pratique une ouverture (au hasard) sur le mur d'enceinte de droite.
- un constructeur de copie *Labyrinthe(Labyrinthe l)* qui prend en paramètre un autre Labyrinthe et le "recopie" dans celui qui est créé.

Notez que les fonctions *dessine...* et *efface...* n'affichent rien à l'écran, mais se contentent de "dessiner" et "effacer" dans le tableau de char, où effacer signifie simplement remplacer par le caractère ' '. Ce n'est pas parce que vous modifiez le tableau de char qu'il s'affiche! Appelez régulièrement la fonction *affiche()* pour afficher votre tableau de jeu à l'écran (après chaque changement pertinent). Aussi il vous est recommandé d'effacer l'écran avant tout nouvel affichage (ceci devrait permette au tableau de jeu de toujours apparaître au même endroit, plutôt qu'en dessous du précédent...).

Votre affichage devrait fonctionner même si on change les valeurs de HMURET et LMURET pour d'autres valeurs (valeurs paires supérieures ou égales à 2).

L**a classe Personnage**: un objet personnage devra comporter les coordonnées de la case où il se trouve et son nombre de vies restantes. Il devra aussi avoir une propriété qui sera une référence vers l'objet Labyrinthe dans lequel il se trouve. Les méthodes seront:

- constructeur *Personnage(Labyrinthe l, int i, int j, int vie)*, prenant en paramètre un Labyrinthe, les coordonnées de la case où devra se trouver le personnage dans ce Labyrinthe, et le nombre de vies
- méthodes pour déplacer le personnage dans les 4 directions

Ce que vous devez faire

Vous devez écrire un programme **Laby.java** qui correspond au jeu de labyrinthe invisible tel que décrit ci-dessus en **respectant les contraintes** suivantes:

- Vous devez respecter le format d'affichage indiqué ci-dessus.
- Vous ne devez utiliser aucune autre fonctionnalité des librairies Java que les fonctionnalités de base vues en cours (notamment entrées/sorties et tableaux), et celles explicitement mentionnées dans le présent énoncé.
- Vous ne devez utiliser aucune variable static de classe (à l'exception des constantes HMURET et LMURET définies ci-haut).

Plus précisément ce que vous devez écrire:

- Une fonction main qui parse les arguments de la ligne de commande et appelle la fonction jeuLabyrintheInvisible (avec les bons arguments) tant que le joueur veut faire une nouvelle partie.
- Une fonction jeuLabyrintheInvisible qui gère une partie au complet du début à la fin.

La fonction jeuLabyrintheInvisible devra notamment:

- Créer un labyrinthe de la bonne taille avec des murets au hasard (voir indications ci-dessous pour comment procéder) ;
- Placer le personnage dans l'une des cases de gauche au hasard et pratiquer une ouverture dans la partie droite au hasard ;
- Afficher le labyrinthe "visible" pendant le temps voulu, avant de l'afficher "invisible" ;
- Gérer les déplacements du joueur (si le joueur entre une commande de direction invalide, on ne fait rien de spécial, à part réafficher le jeu) ;
- Gérer les heurts de murs invisibles, leur affichage, et le nombre de vies du joueur
- Détecter la fin de partie si le joueur parvient à sortir du labyrinthe ou s'il a épuisé toutes ses vies.

Écrivez et utilisez au mieux des fonctions! La plupart des fonctions ne devraient pas dépasser une dizaine de lignes. En règle générale, si une de vos fonctions fait plus d'une vingtaine de lignes, elle est probablement trop longue! Regardez alors si vous ne pouvez pas en extraire certaines opérations et en faire d'autres fonctions plus élémentaires à appeler. Cela aidera à la modularité et à la lisibilité de votre code. Indentez correctement et documentez votre code (avec des commentaires)!

Une fois que vous aurez écrit le jeu de Labyrinthe invisible de base tel que décrit ci-dessus, on vous demande d'ajouter les fonctionnalités suivantes, pour certaines touches de commandes entrées par l'utilisateur qui ne sont pas des touches de direction:

- Touche 'q': quitter le programme
- Touche 'p': commencer une nouvelle partie en regénérant un nouveau labyrinthe (cela peut être utile si il n'y a pas de solution pour le labyrinthe qui vient d'être créé et affiché).
- Touche 'v': rendre tout le labyrinthe visible et redonner le contrôle au joueur (pour les tricheurs!)

Quelques indications

Indications diverses

IMPORTANT: Pour que l'affichage se fasse correctement, assurez-vous que votre terminal utilise une police de taille fixe (ex: *Courier*).

Dans un terminal, vous pouvez interrompre votre programme à tout moment avec la touche Ctrl C. Concernant la saisie au clavier, utilisez la classe Scanner pour saisir un caractère de commande. Attention, les fonctions de Scanner attendent toujours que vous appuyez sur Entrée (ou Enter ou Return selon votre clavier) après avoir saisi le caractère indiquant la direction.

Concernant la création d'un labyrinthe au hasard

La seule méthode de Math que vous devez utiliser est Math.random() qui retourne un nombre aléatoire entre 0 et 1 tiré d'une loi uniforme.

Pour créer un labyrinthe au hasard, on vous suggère de considérer tour à tour tous les murets internes possibles, et de faire un tirage aléatoire d'un boolean pour voir si on devrait construire le muret ou pas. Le tirage aléatoire peut se faire en générant un nombre aléatoire entre 0 et 1 et en le comparant au paramètre *densite* passé sur la ligne de commande (qui est lui aussi entre 0 et 1 et représente la probabilité de construire un muret).

Il vous faut aussi choisir au hasard la position (indice de ligne) où placer le personnage au départ (sur la gauche de l'écran), et la position (indice de ligne) où placer la sortie (sur le mur droite du labyrinthe).

Notez que comme tout est construit au hasard, il se peut parfois (surtout si vous indiquez une densité élevée) qu'il n'y ait pas de possibilité pour votre personnage de sortir du labyrinthe.

Pour faire une pause

Pour faire effectuer une pause pendant un certain temps à votre programme (après avoir affiché le labyrinthe "visible"), vous pouvez utiliser la fonction suivante:

```
public static void sleep(long millisecondes)
{
    try {
        Thread.sleep(millisecondes);
    }
    catch(InterruptedException e){
        System.out.println("Sleep interrompu");
    }
}
```

Comment détecter qu'un mouvement entraînerait une collision avec un muret (possiblement invisible...)

Suggestion: conservez 2 objets de type *Labyrinthe*, un contenant le labyrinthe visible au complet, l'autre contenant le tableau de jeu courant (partiellement invisible). Ainsi vous pourrez aller voir dans le tableau où tout est "visible" s'il y a oui ou non un muret...

Remise

Remettez tous vos fichiers .java sur StudiUM.

Barème

- Fonction main, parsing correct des arguments de ligne de commande et affichage d'un message d'usage: 5 pts
- Création laby, affichage correct du labyrinthe initial et du personnage: 20 pts
- Déplacements du personnage et gestion correcte des collisions de murs: 20 pts
- Gestion des vies, et détection de mort et de gain de partie: 20 pts
- Touches spéciales fonctionnelles q, p, v, o: 15 pts
- Qualité et clarté du code: 5pts
- Commentaires: 5 pts
- Respect des contraintes de l'énoncé: 10 pts