Version : 8 mars 2022

IUT Charlemagne - Dept info
Qualité de Développement

2021-2022
BUT 1A

TD 4 - Labyrinthe Ricochet

1 Description du sujet

Le jeu proposé est un jeu inspiré de `Ricochet Robot` [1].

Dans ce jeu, il s'agit de contrôler un personnage devant atteindre une sortie dans un labyrinthe dont les dalles sont gelées. La difficulté repose sur le fait que lorsque le personnage se déplace, il ne s'arrête que lorsqu'il rencontre un obstacle (cf. figure 1).

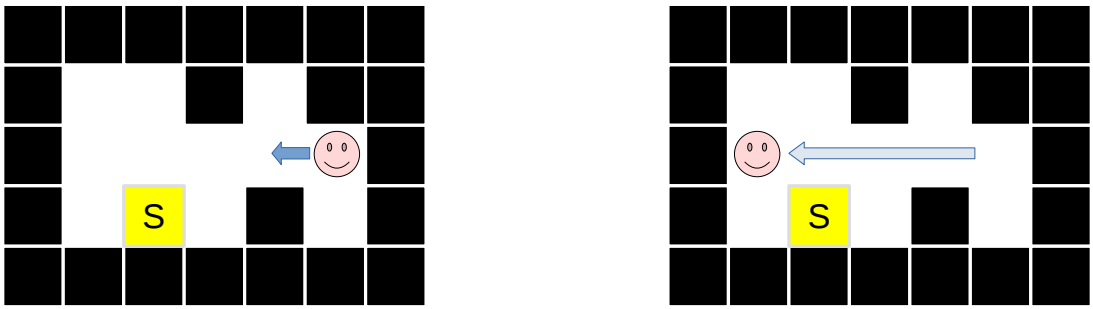

FIGURE 1 – Le personnage décide de se déplacer vers la gauche. Il avance alors de manière rectiligne jusqu'à rencontrer un obstacle.

Atteindre la sortie peut alors demander d'effectuer plusieurs rebonds (cf figure 2).

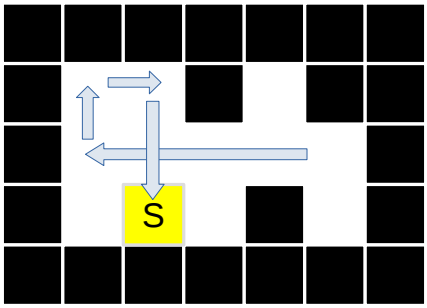

FIGURE 2 – Pour atteindre la sortie, le personnage doit utiliser les murs au mieux.

1. `https://fr.wikipedia.org/wiki/Ricochet_Robots`

Ce TP a pour objectif de développer une application permettant de charger des niveaux et de jouer à ce jeu. Le rendu de ce TP pourra constituer une base pour une des prochaines SAÉ du semestre 2.

Consigne

Le travail sera à faire sur `git` en binôme.

Le rendu constituant une base pour la suite, il sera évalué et une vérification de plagiat sera effectuée (comme pour tout rendu). Aucun plagiat ne sera toléré et nous serons amenés à sanctionner les groupes si il y a plagiat avéré (que ce soit sur le code complet ou une partie de celui-ci).

2 Mise en place git

Pour travailler en binôme, vous utiliserez un dépôt `git` comme vu en TP. Ce dépôt doit se nommer `Laby2022-NOM1-NOM2` où `NOM1` et `NOM2` désigne les deux noms du binôme.

Question 2.1

Dans un premier temps, créer un dépôt `git` commun et **privé** dans lequel vous ajouterez votre enseignant.

Question 2.2

Pensez à y ajouter un fichier `README.md` avec les membres du binôme ainsi qu'un fichier `.gitignore` pour ne pas pusher sur votre dépôt distant les fichiers `.class` et les fichiers de configuration `IntelliJ`.

3 Structure de la classe Labyrinthe

Un objet de type `Labyrinthe` est constitué d'un labyrinthe représenté par un tableau à deux dimension de booléens (représentant les murs du labyrinthe) et un personnage et une sortie qui sont tous deux caractérisés par des coordonnées entières.

Pour avoir les mêmes comportements sur tous les projets, on considérera que la première dimension (habituellement nommée `x`) correspond au numéro de ligne et que la seconde dimension (habituellement nommée `y`) correspond au le numéro de colonne. La manière dont les coordonnées sont représentées est illustré par la figure 3

3.1 Constantes

Pour représenter le labyrinthe à l'affichage et à la lecture du fichier, on utilisera les constantes suivantes de type `char` à déclarer dans la classe `Labyrinthe` :

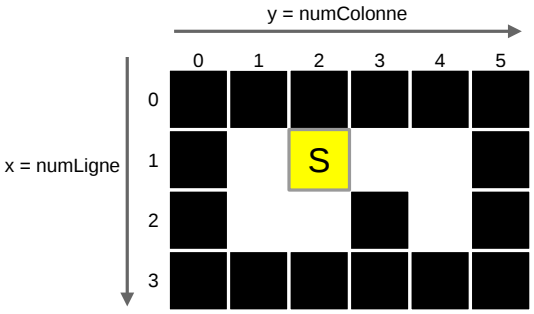

FIGURE 3 – Manière dont les coordonnées sont utilisées. Dans cet exemple, la sortie est située sur la case de coordonnées `(1,2)`

- `'X'` représente un mur et sera représenté par la constante `MUR` ;
- `'P'` représente le personnage et sera représenté par la constante `PJ` ;
- `'S'` représente une sortie et sera représenté par la constante `SORTIE` ;
- `'.'` représente une case vide sera représenté par la constante `VIDE`.

De la même manière, on représentera les actions du joueur par les chaînes constantes suivantes (à déclarer dans labyrinthe) :

- `"haut"` représenté par la constante nommée `HAUT`
- `"bas"` représenté par la constante nommée `BAS`
- `"gauche"` représenté par la constante nommée `GAUCHE`
- `"droite"` représenté par la constante nommée `DROITE`

3.2 Structure interne de la classe

On vous conseille de structurer la classe `Labyrinthe` de la manière suivante :

- un attribut `murs` de type `boolean[][]` (tableau deux dimensions de booléens stockant les murs du labyrinthe) ;
- un attribut `personnage` de type `Personnage` ;
- un attribut `sortie` de type `Sortie` représentant les coordonnées de la sortie du labyrinthe.

Les classes `Personnage` et `Sortie` pourront hériter d'une classe `Position` caractérisée par des coordonnées `(x,y)`. Il est ainsi possible de tirer parti de la classe `Position` pour factoriser des méthodes.

4 Méthodes publiques

La classe `Labyrinthe` doit posséder les méthodes publiques suivantes détaillées dans la suite du sujet :

- `char getChar(int x, int y)` retournant un caractère décrivant le contenu de la case `(x,y)` ;
- `static int[] getSuivant(int x, int y, String direction)` donnant les coordonnées de la case voisine de `(x,y)` selon la direction `direction` ;
- `void deplacerPerso(String action)` déplaçant le personnage lorsqu'il fait l'action `action` ;
- `String toString()` retournant une chaîne décrivant le labyrinthe actuel ;
- `boolean etreFini()` retournant `true` si et seulement si le jeu est fini ;
- `static Labyrinthe chargerLabyrinthe(String nom)` permettant de charger un labyrinthe à partir d'un fichier.

Cependant, rien ne vous empêche d'ajouter d'autres méthodes si vous le jugez utile.

4.1 Méthode `char getChar(int x, int y)`

La méthode `getChar` a pour objectif de retourner un caractère `char` décrivant le contenu de la case `(x,y)`. Ce caractère est une des constantes définies précédemment et représentant le statut de la case à la position `(x,y)` :

- `MUR` si la case est un mur du labyrinthe ;
- `PJ` si la case contient le personnage ;
- `SORTIE` si la case contient ne contient pas le personnage et est une sortie ;
- `VIDE` si la case contient ne contient pas le personnage et n'est pas une sortie.

On considérera que les positions en dehors du labyrinthe sont des murs.

4.2 Méthode `static int[] getSuivant(int x, int y, String direction)`

La méthode statique `getSuivant` a pour objectif de retourner les coordonnées de la case voisine de `(x,y)` selon la direction passée en paramètre. Cette direction sera une des constantes définies (`HAUT`, `BAS`, `GAUCHE`, `DROITE`).

Consigne

Vous ferez attention à bien respecter les dimensions : `x` désigne le numéro de ligne et `y` le numéro de colonne.

Ainsi, la case voisine de la case `(x,y)` vers le `HAUT` a pour coordonnées `(x-1,y)` (le numéro de ligne est réduit de 1).

De la même manière, si on passe en paramètre les coordonnées `1,2` et la direction `DROITE`, `getSuivant` retourne un tableau de deux entiers contenant les coordonnées de la case suivante, c'est-à-dire le tableau `[1,3]` (cf figure 4).

Cette méthode **NE DOIT PAS** vérifier si la case est libre ou occupée mais sera utile pour la méthode `deplacer`.

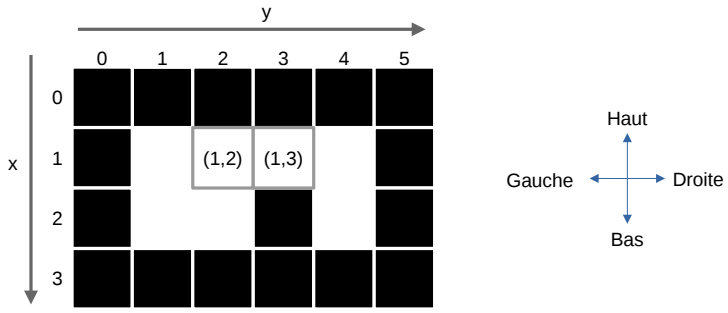

FIGURE 4 – Case voisine à droite de la case de coordonnées `(1,2)`

Si la direction transmise en paramètre ne fait pas partie des 4 actions connues, cette méthode doit retourner une exception de type `ActionInconnueException` à définir et possédant comme message le nom de l'action correspondante.

4.3 Méthode `void deplacerPerso(String action)`

La méthode `deplacerPerso` a pour objectif de modifier les coordonnées du personnage en fonction de l'action faite.

Cela implique de prendre en compte que le personnage glisse sur le sol et avance dans la direction passée en paramètre jusqu'à rencontrer un obstacle (en l'occurrence un mur).

À noter que si le personnage passe par la case de sortie et qu'il n'y a pas d'obstacle, il NE s'arrête PAS et continue son chemin.

Pour éviter d'avoir besoin de tester toutes les directions, vous utiliserez la méthode `getSuivant`.

Si l'action transmise en paramètre ne fait pas partie des 4 actions connues, cette méthode doit retourner une exception de type `ActionInconnueException` à définir et possédant comme message le nom de l'action correspondante.

4.4 Méthode `String toString()`

Cette méthode se charge simplement de retourner une chaîne de caractères décrivant l'état du labyrinthe. Vous penserez à utiliser la méthode `getChar`.

4.5 Méthode `boolean etreFini()`

La méthode `etreFini` retourne un booléen permettant d'arrêter le jeu lorsque le personnage se trouve sur la sortie. Lorsque c'est le cas, cette méthode doit retourner `true`. Elle retourne `false` sinon.

4.6 Méthode `void chargerLabyrinthe(String nom)`

Enfin la méthode `chargerLabyrinthe` a pour objectif de charger un labyrinthe représenté sous la forme d'un fichier texte (cf section 5).

5 Chargement de labyrinthe

Les fichiers décrivant un labyrinthe sont des fichiers textes structurés de la manière suivante (cf figure 5) :

- la première ligne du fichier contient le nombre de lignes du labyrinthe ;
- la seconde ligne du fichier contient le nombre de colonnes du labyrinthe ;
- les lignes suivantes du fichier (qui se finissent par un retour chariot) contiennent le descriptif de chaque ligne du labyrinthe caractère par caractère (sans espace). Chacun de ces caractères correspond à une des constantes déclarées.
  - `'X'` pour un mur ;
  - `'S'` pour la sortie ;
  - `'P'` pour la position initiale du personnage ;
  - `'.'` pour une case vide

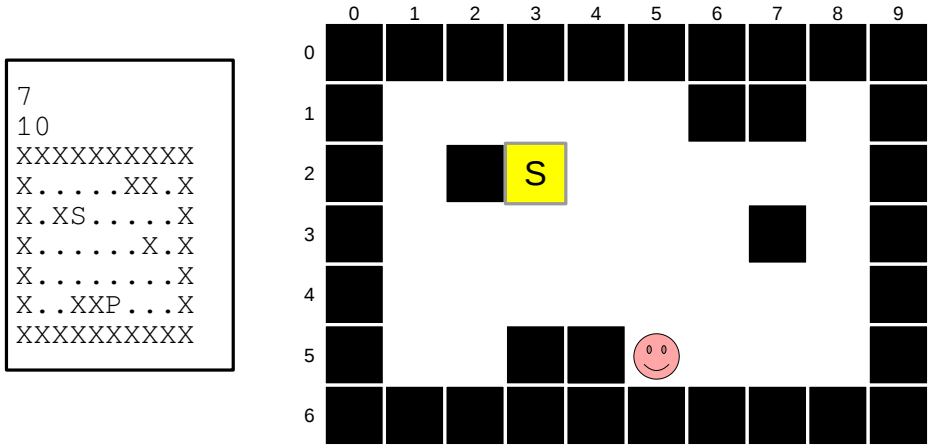

FIGURE 5 – Représentation graphique du fichier `"laby1.txt"` fourni sur arche. Le labyrinthe a 7 lignes, 10 colonnes. La sortie se trouve en `(2,3)` et la position initiale du personnage est `(5,5)`.

On supposera que la position initiale du personnage n'est jamais sur une case de sortie ou de mur.

En plus des exceptions de type `IOException`, plusieurs types d'erreurs peuvent apparaître à la lecture du fichier. Pour chacun de ces cas, on lèvera une exception de type `FichierIncorrectException` (à définir) avec le message correspondant.

- les numéros de ligne ou de colonnes ne sont pas des entiers (message : `"pb num ligne ou colonne"`) ;
- un caractère est inconnu dans le descriptif du labyrinthe (message : `"caractere inconnu <X>"` avec `'<X>'` le caractère correspondant) ;
- le nombre de colonnes ne correspond pas à ce qui a été annoncé (message : `"nbColonnes ne correspond pas"`) ;
- le nombre de lignes ne correspond pas à ce qui a été annoncé (message : `"nbLignes ne correspond pas"`) ;
- il manque la position du personnage (message : `"personnage inconnu"`) ;
- il manque la position de la sortie (message : `"sortie inconnue"`) ;
- il y a au moins deux positions de personnage (message : `"plusieurs personnages"`) ;
- il y a au moins deux positions de sortie (message : `"plusieurs sorties"`).

6 Classe principale

Une fois les différents éléments mis en place, on souhaite écrire une classe `MainLaby` permettant de jouer.

Cette classe doit fonctionner de la manière suivante tant que le jeu n'est pas fini :

- le programme se lance en précisant le nom du fichier contenant le nom du labyrinthe à utiliser ;
- le programme affiche le labyrinthe et demande une action à l'utilisateur :
  - si le joueur entre la chaîne `"exit"`, le jeu s'arrête.
  - si l'action est inconnue lors de l'exécution de l'action, le programme l'annonce à l'utilisateur et demande une nouvelle action ;
  - si l'action est connue, elle s'exécute et la position du personnage est modifiée en fonction de cette action.
- une fois que l'action s'est bien exécutée, le programme affiche le nouvel état du labyrinthe.

En cas de fichier absent ou mal formé, la méthode `main` devra afficher des messages d'erreur adaptés.

Vous pouvez bien sûr utiliser des méthodes intermédiaires si vous le jugez utile.

7 Tests unitaires

7.1 Classe de test fournie

La classe de test `JUnit` fournie `TestLabyProf` vérifie simplement que les noms des classes, des méthodes et des constantes sont corrects.

Consigne

Vous ne devez JAMAIS modifier la classe `TestLabyProf` par vous-même.

Si les consignes de l'énoncé ou la classe de test venaient à se contredire à cause d'une erreur de sujet, contactez directement votre enseignant.

Par défaut, la classe de test est prioritaire sur le sujet.

Question 7.1

Ajouter la classe de test fournie et vérifiez qu'elle fonctionne correctement (compilation et résultats du test).

7.2 Ecriture des tests

Cette partie a pour objectif de mettre en place des tests à l'aide de `JUnit` permettant de vérifier votre travail.

Question 7.2

Prenez le temps d'identifier les cas à tester et écrire les méthodes de test correspondant.

7.3 Debugger

Question 7.3

En cas d'erreur de test, utilisez le débugger pour localiser l'erreur et la corriger.

7.4 Couverture de tests

Lancer les tests avec vérification de la couverture de test.

Question 7.4

Ajouter dans votre compte rendu un retour rapide sur les résultats des tests et précisez leur couverture. Expliquez pourquoi certains morceaux de votre code sont non testés.

8 Rendu

Votre rendu se fera sous la forme de votre dépôt git. Votre enseignant y a accès et fera un clone de votre dépôt à la date spécifiée dans le sujet. Tout travail effectué ultérieurement ou non pushé sur votre dépôt ne sera pas pris en compte.

- un répertoire `src` contenant vos fichiers sources ;
- un répertoire `laby` contenant les fichiers des labyrinthe utiles (aux tests ou au `main`) ;
- un répertoire `test` contenant vos tests unitaires ;
- un fichier `.gitignore` adapté à votre projet ;
- un fichier `README.md` contenant
  - les noms des membres du binôme ;
  - un descriptif des difficultés rencontrées au cours du TP ;
  - une explication de vos choix de programmation ;
  - un descriptif expliquant comment lancer votre application ;
  - un résumé des résultats de vos tests (ce qui réussit / échoue éventuellement) ;
  - une présentation de votre couverture de test et des explications associées.

Vous ferez attention à ce que votre dépôt ne contienne pas les fichiers de configuration de votre projet `IntelliJ` ou des fichiers inutiles (tels que des `.class`).