IFT1025 - Travail Pratique 1 (Énoncé partiel)

RobotFindsKitten : Super Dungeon Master 3000^{Ultra}_{Turbo} Edition

Concepts appliqués : Programmation orientée objet, héritage, développement d'application de taille minimalement intéressante

Contexte

RobotFindsKitten (traduction : RobotTrouveChaton) est un jeu pour `MS-DOS` conçu en 1997 par Leonard Richardson.

Il s'agit d'une *simulation zen* : vous incarnez un Robot (`#`) qui cherche Chaton (affectueusement surnommé *Kitten* dans le reste du TP) et qui examine chaque objet sur la grille, jusqu'à ce qu'il trouve chaton (pas de contrainte de temps, pas de barre de vie, pas de score, . . .)

L'objectif principal est de passer un bon moment en contemplant les divers objets non-chatons (*Non-Kitten-Items*) qui se trouvent dans le niveau.

Exemple de jeu

Le jeu est représenté sous forme de grille en caractères ASCII :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        [j                            f           Y       %
%                                F                            Z            %
%   I A       Oc  k                                         Y              %
%    _    ]                             sC         #       A            G  %
%                   a            t         X                        b      %
%  k     o                         `                                 A     %
%           j                             '         Y    ;                 %
%         T Y   l   y             j                                 J I    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Chaque fois que le robot marche sur un item qui n'est pas le chaton, une description de l'item en question est affichée.

Quelques exemples :

> This is a tasty-looking banana creme pie.
>
> It's a business plan for a new startup, kitten.net.
>
> An automated robot-liker. It smiles at you.
>
> A book with "Don't Panic" in large friendly letters across the cover.

Chaque item sur la grille a une description fixe tirée au hasard en début de partie depuis une banque de descripteurs de *Non-Kitten-Items*.

Le jeu se termine lorsque le Robot trouve Chaton.

Une version web du jeu original est disponible ici : http://robotfindskitten.org/play/robotfindskitten/

Votre travail

Vous devrez recréer ce jeu, en y ajoutant quelques éléments :

- Plutôt que d'être une grande salle ouverte, la grille de jeu est divisée en pièces fermées à clé
- Le robot peut collecter des clés pour ouvrir les portes et passer à une prochaine pièce
- Chaque pièce contient quelques *Non-Kitten-Items* et une clé (représentée par une apostrophe)

De plus :

- Un des *Non-Kitten-Items* sur la grille est un *Téléporteur*, qui permet de se rendre à une position aléatoire de la grille sur demande

Le programme final attendu vous est donné en exemple avec le fichier `rfk.jar`.

Vous pouvez le télécharger et l'exécuter en ligne de commande avec :

```
java -jar rfk.jar
```

Déroulement du jeu

1. La partie commence par un message de bienvenue :

```
      Bienvenue dans RobotFindsKitten
Super Dungeon Master 3000 Ultra Turbo Edition !
```

2. La grille de jeu est générée semi-aléatoirement :
   - Des murs (`%`) sont installés autour de chaque pièce avec une porte au milieu (`!`) pour passer d'une salle à l'autre
   - Une clé est déposée sur une case au hasard dans chaque pièce
   - Les *Non-Kitten-Items* et le *Kitten* sont disposés sur la grille de façon aléatoire (la représentation des *Non-Kitten-Items* et du *Kitten* est déterminée aléatoirement parmi un ensemble de caractères ASCII)
   - En plus des *Non-Kitten-Items* un *Téléporteur* est caché quelque part sur la grille de jeu

La grille une fois générée devrait avoir l'air de quelque chose comme :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  F       e %        P    %  ‘         % T @        %            ’%
%            %Y       V    %         M’ %      d     %             %
%    gf      !           v!    j        !  x     z   !     #     @ %
% ’          %       m    %            %  |  ’      %             %
%            % e’ ] p     %g         | % i    o      %     g      %
%%%%%%!%%%%%%%%%%%%!%%%%%%%%%%%%!%%%%%%%%%%%%!%%%%%%%%%%%%!%%%%%%
%    j       %  }      o  %           w%            %             %
%F           %         g  %            %            %          u %
%        W   !       [    !’   d       !        e  t! ?          %
%’           % N    z     %            %           % ’      v    %
%    @   t   %’      @     %    T S  h %       ’    %^       E    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. La grille est affichée, puis un prompt demande à l'utilisateur d'entrer son mouvement :

```
R.O.B. [0]>
```

Le prompt représente le status du robot :

`{Nom du robot} [{nombre de clés en sa possession}]`

4. Le robot peut explorer la carte en entrant les commandes `w`, `a`, `s` ou `d`, respectivement pour se déplacer dans les directions *Haut*, *Gauche*, *Bas* ou *Droite*

5. Lorsque le robot ramasse une clé, il peut s'en servir pour débarrer une des portes (qui restera débarrée pour le reste de la partie)

6. Si le robot possède le téléporteur, il peut entrer la commande `t` pour se téléporter sur une case libre aléatoire sur la grille. De plus, un `T` est ajouté à la fin du prompt pour indiquer qu'il peut se téléporter :

```
R.O.B. [0]T>
```

La grille est affichée et le prochain mouvement est demandé jusqu'à ce que Robot trouve Chaton

6. Lorsque Robot trouve Chaton, le message suivant est affiché :

```
You found kitten! Way to go, robot.
Caramel <3 R.O.B.
```

Où `Caramel <3 R.O.B.` correspond à `{Nom du Kitten} <3 {Nom du Robot}`

Et le programme se termine.

Changement par rapport au jeu original

Notez que contrairement au jeu original, il est possible de marcher sur les NonKittenItems. Cela simplifiera l'implantation et évitera des cas où le robot pourrait rester coincé dans sa position de départ.

Référez-vous au programme `rfk.jar` si vous doutez de la façon dont certaines choses devraient fonctionner.

NonKittenItems

Vous êtes libres d'utiliser les messages de NonKittenItems que vous voulez. Une liste de messages de NonKittenItems provenant du jeu original vous est fournie sur StudiUM.

Hiérarchie de classes

On vous fournit la classe helper `Point` :

```
public class Point {

    private final int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    boolean egal(int x, int y) {
        return x == this.x && y == this.y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

Il s'agit d'une *classe immuable*, c'est-à-dire une classe dont les attributs ne peuvent pas être modifiés après l'instanciation.

Utilisez cette classe pour simplifier la manipulation des coordonnées x & y lorsque nécessaire.

On vous fournit également la classe abstraite Case :

```
public abstract class Case {

    protected char representation;
```

```java
    /**
     * Retourne la représentation de la case (un seul caractère)
     *
     * @return la représentation de la case
     */
    public char getRepresentation() {
        return representation;
    }

    /**
     * Indique si une interaction entre la case et le robot est
     * possible (ex.: le robot peut interagir avec un NonKittenItem
     * en tout temps, mais ne peut pas interagir avec un mur, le robot
     * peut interagir avec une porte seulement s'il possède une clé,
     * etc.)
     *
     * @param robot Le robot qui interagirait avec la case
     * @return true si une interaction entre le robot et la case est possible
     */
    public abstract boolean interactionPossible(Robot robot);


    /**
     * Interaction entre la case et le robot
     *
     * @param robot
     */
    public abstract void interagir(Robot robot);

    /**
     * Génère un symbole aléatoire
     *
     * @return Un symbole ASCII compris entre ':' et '~'
     */
    public static char getRandomSymbole() {
        return (char) (Math.random() * (126 - 58) + 58);
    }
}
```

Pour vos classes, utilisez la nomenclature et la structure qui suivent :

```
public class RobotFindsKitten
============================
  Classe principale
  - Contient la fonction main() et la logique de base du jeu
    C'est dans cette fonction que se fait la création d'une new
    Grille(...) et d'un new Robot(...)

public class Robot
==================
  Classe qui représente le robot
```

```
  Le robot a minimalement les attributs suivants :
  - (String) nom du robot
  - (Point) position x, y sur la grille
  - (int) nombre de clés collectées (et pas encore utilisées)
  - (boolean) possède le téléporteur ou non

public class Grille
====================
  Classe qui représente la grille de jeu

  private Case[][] grille
    un tableau 2D de cases

  Constructeur : public Grille(int nbrPiecesX, int nbrPiecesY,
                               int largeurPiece, int hauteurPiece, int nbrNonKitten)
    Cette fonction initialise la grille en créant les pièces, les portes, les murs
    les clés et les items (le téléporteur, les NonKittenItems et le Kitten)
    Il y a ‘nbrNonKitten‘ NonKittenItems au total sur tout le jeu

  public Point randomEmptyCell() :
    Retourne une coordonnée de cellule qui ne contient rien

  public boolean deplacementPossible(Robot robot, int x, int y)
    indique si c’est possible pour le robot robot de marcher sur la
    cellule de coordonnée (x, y)

  public void afficher(Robot robot)
    Affiche la grille dans la console à coups de System.out.println(...)

  void interagir(Robot robot)
    Lance l’interaction entre le Robot robot et la case de la grille sur
    laquelle il se trouve

    L’"interaction" peut être l’affichage d’un message (pour les NonKittenItems),
    l’ouverture d’une Porte, le fait de ramasser une clé ou un téléporteur, ou encore
    le fait de gagner la partie en trouvant le Kitten

Les classes suivantes héritent de Case :

- public class Kitten extends Case
- public class NonKitten extends Case
- public class Cle extends Case
- public class Porte extends Case
- public class Teleporteur extends Case
- public class Mur extends Case
```

Vous pouvez bien sûr ajouter des méthodes et des attributs aux classes demandées ou créer d’autres classes au besoin.

Indications supplémentaires

- La date de remise est le *6 mars 2018 à 23h55*. Il y a une pénalité de 25% pour chaque jour de retard.
- Vous devez faire le travail par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code, dans le fichier principal (`RobotFindsKitten`).
- Une seule personne par équipe doit remettre le travail sur StudiUM, l'autre doit remettre un fichier nommé `equipe.txt` contenant uniquement le code d'identification de l'autre personne (`p1234...`, soit ce que vous utilisez pour vous logguer sur StudiUM)
- Un travail fait seul engendrera une pénalité. Les équipes de plus de deux ne sont pas acceptées.
- Basez-vous sur le code fourni et remettez tous les fichiers
- L'évaluation du code se fera principalement sur les critères suivants :
  - L'exactitude (respect de la spécification)
  - L'élégance et la lisibilité du code
  - La présence de commentaires explicatifs lorsque nécessaire
  - Le choix des identificateurs
  - La décomposition fonctionnelle
- De plus :
  - La performance de votre code doit être raisonnable
  - Chaque fonction devrait être documentée en suivant le standard `JavaDoc`
  - Il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense (utilisez votre bon sens pour arriver à un code facile à lire)
  - Les identificateurs doivent être bien choisis pour être compréhensibles (évitez les noms à une lettre, à l'exception de i, j, ... pour les variables d'itérations des boucles for)
  - Vous devez respecter le standard de code pour ce projet, soit les noms de variables et de méthodes en camelCase