UFR ST - Besançon- L2 Info - Année 2014/15

Programmation Orientée Objets

Projet TP - STONERS : les pétrifieurs

Ce TP noté est à réaliser en binôme sur les dernières séances de TP de programmation par objets. Il sera nécessaire d'y consacrer du temps supplémentaire, au titre des heures de travail personnel, en dehors des créneaux de TP pour le terminer.

1 Présentation rapide du projet

Ce projet consiste à faire se déplacer des personnages sur un damier, où ils peuvent rencontrer des obstacles contre lesquels ils peuvent rebondir. Ils peuvent aussi rencontrer d'autres personnages susceptibles de les « pétrifier » : ils ne bougent plus, jusqu'à ce qu'un autre personnage vienne les « dépétrifier ». Ils pourront alors reprendre leur promenade sur le damier.

Les personnages se déplacent tout seuls, c'est à dire sans intervention de l'utilisateur, et leur parcours est modifié par les obstacles ou personnages qu'ils rencontrent.

Ce projet est à réaliser en java, et une programmation objet mettant en œuvre les concepts d'héritage, de classe abstraite, d'interface et de polymorphisme est exigée.

2 Réalisations

Une analyse devra être rendue au bout d'une semaine, pour validation par votre chargé de TP, avant de commencer le développement.

Le code complet de votre projet sera rendu en fin de semestre, avec une présentation à l'oral en TP de votre conception et de votre programmation.

3 Description des éléments du jeu

Tous les éléments du jeu (obstacles, personnages, cases vides si nécessaire) possèdent une *position* dans le damier, comme décrit dans le sujet de TP précédent, et rappelé dans la partie 4.3. Les personnages peuvent se déplacer, et donc ils possèdent en plus une *direction*, également décrite dans le sujet de TP précédent et dans la partie 4.3.

Les obstacles sont de deux types : des murs, ou des tourniquets. Les personnages sont de trois types : des promeneurs, des pétrifieurs ou des réssusciteurs.

Les cases du damier sont soit libres (cases vides), soit occupées (par un obstacle ou un personnage). Il n'y a pas de superposition sur une case : une case occupée ne l'est que par *un seul* objet (obstacle ou personnage).

3.1 Les obstacles

3.1.1 Les murs (*Wall*) : ♯

Ils modifient la direction des personnages qui viennent les heurter en les faisant tourner d'un quart de tour dans le sens des aiguilles d'une montre. On pourra les visualiser par le symbole '`#`'.

3.1.2 Les tourniquets (*Spin*) : @

Ils modifient la direction des personnages qui viennent les heurter en leur donnant une nouvelle direction aléatoire. On pourra les visualiser par le symbole '`@`'.

3.2 Les personnages

Les personnages se déplacent sur le damier d'une case à la fois, selon la direction qui est la leur. Quand ils rencontrent une case libre, ils l'occupent. Sinon, ils changent simplement de direction, mais sans avancer car la case qu'ils ont rencontrée était déjà occupée.

Quand un personnage vient en heurter un autre, il fait demi-tour : au prochain tour, il repartira en sens inverse. Mais il peut aussi rester pétrifié s'il se heurte à un pétrifieur. Un personnage pétrifié devra attendre d'être heurté par un réssusciteur pour pouvoir bouger de nouveau.

3.2.1 Les promeneurs (*Walker*) : O

On pourra les visualiser par le symbole '`O`'.

Ils n'ont pas le pouvoir de pétrifier ou de dépétrifier les personnages qui les croisent. Ils se contentent donc de se promener sur le damier, au hasard des rencontres.

3.2.2 Les pétrifieurs (*Stoner*) : X

On pourra les visualiser par le symbole '`X`'.

Les pétrifieurs figent les objets qu'ils rencontrent, c'est à dire qu'ils les immobilisent (leur position ne change plus) sur le damier. Les pétrifieurs peuvent se pétrifier entre eux.

3.2.3 Les réssusciteurs (*Resurrector*) : R

On pourra les visualiser par le symbole '`R`'.

Les réssusciteurs, quand ils viennent heurter un personnage figé, lui redonnent vie : il pourra à nouveau se déplacer. Les réssusciteurs peuvent eux mêmes être pétrifiés, puis ressuscités.

4 Consignes de conception et d'implantation

4.1 Affichage

L'affichage se fera en mode console. Le damier, avec les obstacles et les balles qu'il contient, sera affiché après chaque tour. L'utilisateur pourra choisir soit de voir s'afficher les différents tours les uns à la suite des autres sans intervention de sa part, soit de saisir une touche après chaque tour.

La représentation du damier sera assurée par une méthode toString() appropriée.

4.2 Damier et dimensions du jeu

Le tour du damier est constitué de murs comme au TP précédent.

Les dimensions du damier, ainsi que le nombre maximum d'obstacles et de personnages qu'il peut accueillir sont laissés à votre appréciation, et devront être définis par des constantes, éventuellement interdépendantes. Leur nombre peut être demandé à l'utilisateur au lancement du programme. Il est conseillé de ne pas trop saturer le damier pour mieux observer les mouvements.

Vous pouvez choisir d'attribuer des positions déterminées à l'avance à vos obstacles et à vos personnages, ou au contraire de les disposer au hasard sur le damier. Dans tous les cas, on doit s'assurer qu'une position est libre avant d'y déposer un obstacle ou un personnage.

Voici un exemple de damier :

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
# # # . . . . # . . . . . . . . . . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . . . . . . . . . O . . . . . . #
# . . . . . . . . . . . R . . . . . X . . . . . @ X R . . . O . #
# . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . #
# . . . . . # . . . . . . . . . . . . . . . . . . . . . . . . . . . . #
# . . @ . . # # . . . . @ . . . . # # # . . . . . # . . # . . #
# . . . . . # . . . . . . . . . . @ . . . O . . O . . . . . . . . #
# . . . . . # . . . . . R . . . . . . . . . . . O . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . # #
# . . . X . . . . @ . . . . . . . . . O O . . . . . O . . . . . #
# # . . R . . @ . . . . . . . . . . . . # # . . . . . # . . . . #
# . . . . . . . . . . . . . . . . . . . . . . . . . . . # . . # #
# . . . . . . . . . . @ . . X X . . . . . . . . . . # . . . . . # #
# X . . . . . . . . # . . . . . R . . X . . . . . X . . . . . . #
# . . # . . . . . . . . . . . . . . . . . . . . . . . . . . . . # #
# R . . . . . . . . . . R . . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . # . O X . . . . . # . O @ . . . #
# . . # . . . # # . . . . . . . . . . . . . . . # . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . . . . . # # . . . . . . . . . #
# . O . . . . . # . . . . . . . . . . . . . # . . . . . . . . . . #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

4.3 Position et direction

Vous devez coder une classe Position et une classe Direction.

Position définit la position d'une pièce sur un damier par un couple d'entiers.

Direction définit la direction d'une *pièce en mouvement*. Une direction peut être codée par un couple d'entiers, chacun d'eux valant soit 1, 0 ou -1, pour coder la variation à appliquer à la composante correspondante de la position : la valeur 1 indique que la valeur (en x ou en y) de la position doit être augmentée, la valeur -1 qu'elle doit être diminuée, et la valeur 0 qu'elle doit rester inchangée. Par exemple, la direction *nord* est codée par le couple $(0, 1)$, et la direction *sud* par le couple $(0, -1)$. La direction *ouest* est codée par le couple $(-1, 0)$. La direction *sud-est* est codée par le couple $(1, -1)$. Etc.

4.4 Évolution : gestion d'une collection de personnages en mouvement

Les personnages en mouvement sur le damier seront rassemblés au sein d'une collection de type ArrayList. Faire évoluer le jeu d'un tour consistera alors à parcourir la collection au moyen d'un itérateur pour déplacer chaque personnage, en gérant les collisions avec les autres objets sur le damier. Le principe du déplacement d'un personnage est de calculer sa position potentielle future. Si elle est libre, le personnage s'y déplace. Sinon, il interagit avec l'objet rencontré (obstacle ou personnage) : sa direction change, et il peut éventuellement rester pétrifié.

4.5 Comportements des obstacles et des personnages : implantation d'interfaces

Les comportements des différents objets (personnages ou obstacles) de votre jeu devront être définis au moyen d'interfaces appropriées. Vous avez pour contrainte de *définir* des interfaces, et de les faire *implanter* par vos objets. Les comportements spécifiés par interfaces concernent les interactions entre objets : comment un objet est pétrifiable, comment il redirige un personnage qui vient le heurter, etc.

Le but de cette approche est de pouvoir définir après coup de nouveaux obstacles ou personnages. Du moment qu'ils implantent les interfaces décrivant leurs interactions avec les autres objets, votre programme saura les manipuler.

Une interface imposée : Steerable

L'interface suivante vous est fournie et doit être implantée par tous les personnages. Ils sont tous dirigeables (*steerable* en anglais), c'est à dire que leur direction peut être modifiée : ils doivent implanter les méthodes de l'interface Steerable. getDirection() permet de récupérer la direction d'un personnage et setDirection(Direction dir) de la modifier.

```
public interface Steerable {
    public Direction getDirection() ;
    public void setDirection(Direction dir) ;
}
```

5 Calendrier

semaine 47	distribution du sujet
semaine 48	analyse (diagramme des classes) à rendre
semaine 50	rendu TP (vendredi 12/12 jusqu'à 23h55)
semaine 51	présentation orale et démonstration