UFR ST - Besançon- L2 Info - Année 2015/16
Programmation Orientée Objets
Projet TP - La chasse au trésor

Ce TP noté est à réaliser en binôme sur les dernières séances de TP de POO. 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 un personnage (un aventurier) sur un damier, à la recherche d'un trésor. Le trésor se trouve dans une case du damier, inconnue du personnage. À chacun de ses déplacements, il interroge la case sur laquelle il arrive qui le réoriente en direction du trésor. Les cases peuvent être libres auquel cas le personnage peut s'y déplacer, ou être des pierres faisant partie d'un mur que le personnage devra contourner.

Le personnage se déplace tout seul, c'est à dire sans intervention de l'utilisateur. Son parcours est modifié par les cases qu'il rencontre.

Ce projet peut être enrichi en introduisant plusieurs personnages, ou encore en faisant se déplacer le trésor.

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 sous forme de démo, de votre conception et de votre programmation.

3 Vue d'ensemble des éléments du jeu

Les éléments immobiles du jeu (pierres, cases libres, trésor) occupent une position fixe sur le damier. Les personnages se déplacent sur le damier, selon une direction qui leur est propre et qui peut être horizontale, verticale ou diagonale. Les obstacles sont des murs horizontaux ou verticaux, constitués d'une collection de cases de type pierre. Enfin, un type spécial de case correspond au trésor recherché.

Les cases du damier sont soit libres, soit occupées (par un obstacle, le(s) personnage(s), ou le trésor). Quand une case est occupée, elle ne l'est que par *un seul* objet (obstacle, personnage ou trésor).

Les cases connaissent la position du trésor, mais elles ne l'indiquent pas directement à l'aventurier, ce serait trop facile ! En pratique, toutes les cases, au contraire du personnage, disposent d'une méthode distanceFromTreasure() qui donne leur distance au trésor.

Par ailleurs, tous les types de case ainsi que les personnages sont capables de changer la direction d'un personnage, au moyen d'une méthode redirect(Hunter p) prenant le personnage en paramètre.

4 Principe de codage des différents éléments

Tous les types de case héritent, directement ou indirectement, d'une classe abstraite Case (ou Cell en anglais).

Tous les types de case, mais *pas les personnages*, implantent une interface nommée par exemple Askable et proposant la méthode distanceFromTreasure().

Pour la redirection d'un personnage, toutes les cases mais aussi les personnages implantent la méthode redirect(...).

4.1 Les personnages : (Hunter, de symbole 'O')

Ils possèdent, contrairement aux pierres et aux cases libres, une direction. 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 n'avancent pas car la case qu'ils ont rencontrée était déjà occupée. Mais dans tous les cas, la case cible, qu'ils l'occupent ou non, change leur direction (par la méthode redirect()).

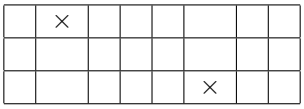
Les personnages n'implantent pas l'interface Askable. Mais ils implantent la méthode redirect() car si un autre personnage vient à leur rencontre, ils le redirigent. Mais ils le redirigent de manière aléatoire : cela modélise qu'ils délivrent une fausse information aux concurrents !

4.2 Les cases libres : (Free, de symbole '.')

Elles savent calculer leur propre distance au trésor, mais elles devront aussi savoir interroger leurs cases voisines (au maximum 8) pour connaître leurs distances respectives au trésor. En fonction de cela, elles réorienteront le personnage en direction de la case la plus proche du trésor.

Calcul de la distance

On la compte en nombre (entier) de cases. Il suffit de considérer le carré de la distance euclidienne en nombre de cases. Par exemple, la « distance » calculée de cette manière entre les deux cases marquées d'une croix dans l'exemple suivant est de 20 ($= 4^2 + 2^2$).



4.3 Les cases pierres : (Stone, de symbole '♯')

Elles calculent aussi leur propre distance au trésor, mais réorientent le personnage en fonction du mur auquel elles appartiennent, car elles sont infranchissables.

4.4 Les murs

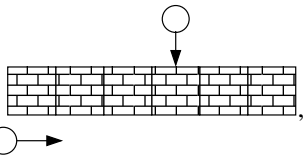
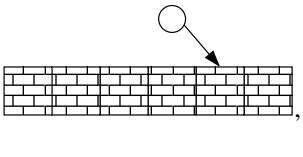
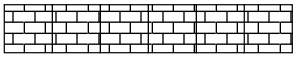
Les murs sont des séries continues horizontales ou verticales de cases pierres sur le damier.

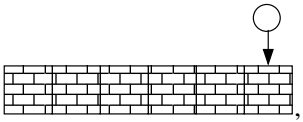
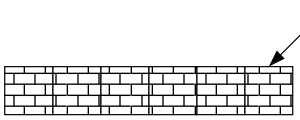
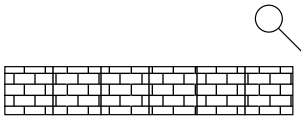
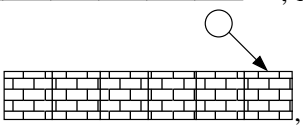
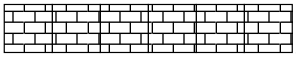
Un mur sera *un objet* (Wall) regroupant dans une collection de type ArrayList les cases (de type Stone) qui le constituent. Chaque mur possède deux bouts, qui sont les cases pierre aux extrémités du mur : leurs indices sont donc respectivement 0 et size()-1 dans la collection. La distance d'une pierre d'indice i à chacun des bouts est donc donnée par les formules i pour un bout, et size() $- i - 1$ pour l'autre bout (à implanter chacune dans une méthode de l'objet mur). L'objet mur possède aussi un attribut indiquant s'il est horizontal ou vertical.

Pour ne pas bloquer le personnage, les murs ne seront jamais collés les uns aux autres. De plus, ils ne sont jamais collés non plus aux bords du damier. Autrement dit, il y'a toujours un point de passage à chaque bout du mur qui va permettre de le contourner.

Quand le personnage vient heurter une pierre, celle-ci le réoriente vers le bout du mur le plus proche d'elle. Attention à bien gérer le cas particulier des bouts du mur qui doivent orienter vers le point de passage.

Voici des exemples de réorientation (le personnage est figuré par le cercle).

Quand le personnage arrive comme ceci  , ou comme cela  , il est réorienté comme suit :  .

Pour le bout du mur, quand le personnage arrive comme ceci  , ou comme cela  , il est réorienté comme suit :  . Mais s'il arrive comme ceci  , alors il est réorienté comme suit :  .

5 Consignes de conception et d'implantation

5.1 Affichage

L'affichage se fera en mode console. Le damier, avec les obstacles et les personnages 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.

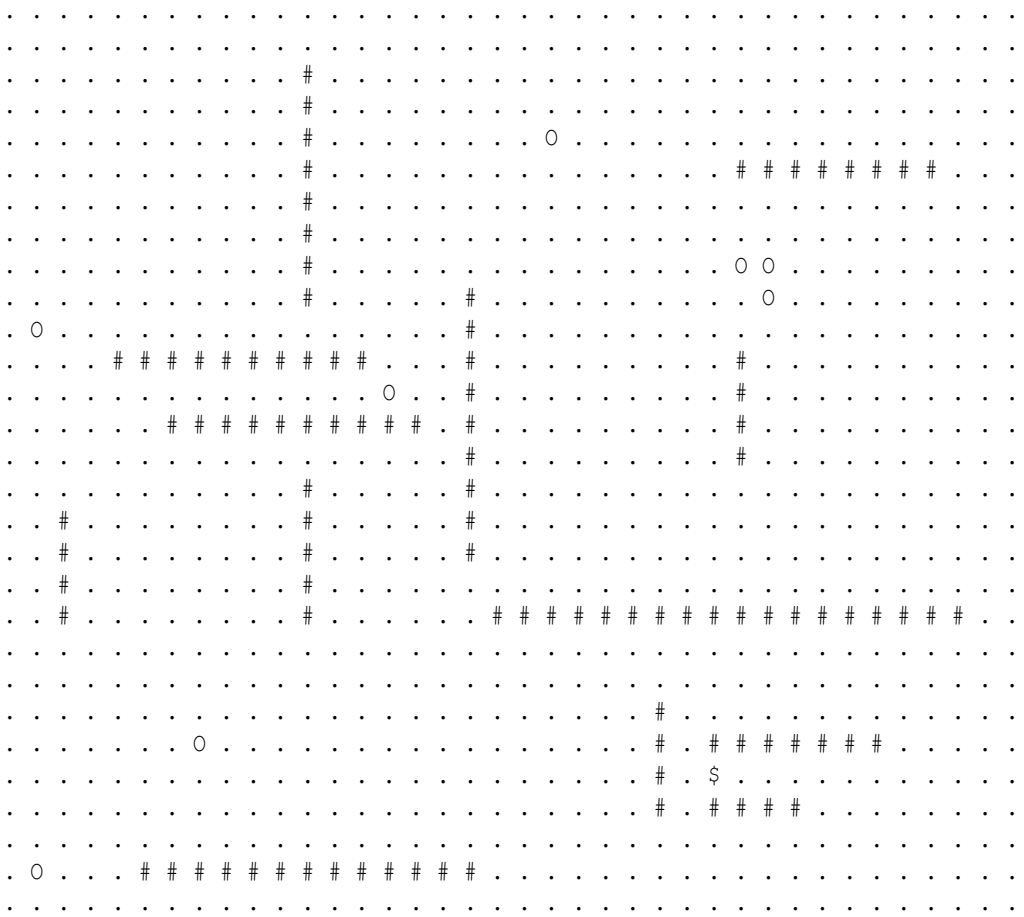
La représentation du damier sera assurée par une méthode toString() appropriée.

5.2 Damier et dimensions du jeu

Le damier est soit entouré de bords comme au TP précédent, soit il est sans bords. Dans tous les cas, les personnages ne doivent pas quitter le damier.

Les dimensions du damier, ainsi que le nombre de murs de personnages qu'il peut accueillir sont laissés à votre appréciation. Il est demandé de disposer tous les éléments du jeu (personnages, murs, trésor) au hasard sur le damier, mais en respectant la consigne qu'il y'a toujours un passage pour contourner un mur. 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 (le trésor est figuré par un $) :



5.3 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 un par un, 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 (futurePos). Si elle est libre, le personnage s'y déplace. Dans tous les cas, le personnage se fait rediriger par la case se trouvant en futurePos.

Le jeu s'arrête dès qu'un personnage a trouvé le trésor : à vous de gérer ce cas spécial d'arrivée d'un personnage sur la case trésor.

6 Évolutions possibles

On demande au minimum d'implanter ce « jeu » avec des murs, un trésor et un personnage, en version console. Toutefois, il est vivement recommandé d'avoir plusieurs personnages, en concurrence entre eux, sur le damier. Un raffinement peut consister à faire se déplacer le trésor (on imagine qu'il est à bord d'un véhicule par exemple). Les distances ne sont alors plus fixes mais changent à chaque tour.

On peut aussi introduire d'autres types d'obstacles : des miroirs cassés qui renvoient dans la direction symétrique une fois sur deux, des trappes qui téléportent sur une position (libre) aléatoire, des raccourcis, etc. À votre imagination ! Enfin, vous pouvez réaliser le jeu en version graphique mais uniquement à titre de bonus.