L2 Info POA

Mini projet 2024 : les tours infernales

Ce projet est noté, et à réaliser sur les dernières séances de TP en complément du temps de travail personnel. **Le projet sera réalisé en binôme**. Le code doit être compatible java 11. Une programmation objet mettant en œuvre les concepts d'héritage, de classe abstraite, d'interface, de généricité et de polymorphisme est exigée.

Présentation rapide

Ce projet est un exercice de programmation consistant à faire évoluer, automatiquement et au hasard, des personnages sur une grille où sont disposées des tours. Les personnages se déplacent d'une case à la fois en suivant une direction horizontale, verticale ou diagonale quand ils sont au sol. Ils rebondissent quand ils rencontrent un bord. Quand ils arrivent sur une case contenant une tour, ils entrent dedans se mettent à se déplacer en hauteur : ils peuvent monter les étages un à un jusqu'à arriver sur le toit. S'ils y parviennent ils deviennent propriétaires de la tour. Depuis le toit ils peuvent soit redescendre les étages un à un, soit sauter sur le toit d'une autre tour à condition d'en être propriétaire.

A minima, il est demandé d'implanter une version où les personnages évoluent complètement au hasard, en respectant les règles de fonctionnement détaillées ci-dessous. Vous devrez ensuite proposer d'autres types d'objets que les personnages, les tours et les bords, à placer sur la grille. En fonction du temps disponible, vous pourrez implanter une version plus évoluée où les personnages appliquent une priorité au choix de leur déplacement pour essayer de posséder plus de tours au détriment des autres personnages.

Règles de fonctionnement détaillées

Comme les personnages évoluent dans le plan (hors des tours) mais aussi en hauteur (dans les tours), la position est un triplet (x, y, z), ou plutôt $(row, col, height)$ pour *ligne, colonne, hauteur*. Un personnage p avance case par case à chaque tour, en suivant sa direction. Quand la case cible de p est libre il s'y déplace. Les autres cas sont énumérés ci-dessous.

Rencontre avec un autre personnage

La case cible de p est un autre personnage p'. p ne bouge pas mais sa direction est inversée (il repartira d'où il vient au prochain tour).

Rencontre avec un bord.

La case cible de p est un bord. p ne bouge pas mais sa direction change comme lors d'un rebond au billard. Il repartira conformément à ce rebond au prochain tour.

Plus précisément, quand p arrive perpendiculairement sur un bord, sa direction est inversée (il repartira en arrière). Mais quand p arrive en diagonale contre un bord, sa direction subit une rotation d'un quart de tour. Notons qu'en arrivant en diagonale contre un coin on repartira en arrière.

Rencontre avec une tour

La case cible de p contient une tour. L'idée générale est que p va pouvoir entrer dans la tour, et en monter les étages un à un, c'est à dire un étage à chaque tour.

S'il ne rencontre pas d'autre personnage au cours de sa montée, alors il atteint le toit et devient propriétaire de la tour. Depuis le toit, il peut (choix au hasard) :

- soit redescendre les étages un à un et ressortir de la tour (s'il n'a croisé personne dans sa descente)
- soit sauter sur le toit d'une autre tour dont il est propriétaire. Il pourra alors redescendre les étages un à un de cette autre tour (s'il ne croise personne dans sa descente).

Rencontre dans la tour avec un autre personnage p' : c'est la règle de rencontre avec un personnage qui s'applique, c'est à dire que la direction est inversée (s'il monte il redescendra, s'il descent il remontera).

Entrée dans la tour depuis le sol

Le rez-de-chaussée de la tour est une case spéciale qu'un personnage ne peut pas occuper (puisque cette case est déjà occupée). Pour que p puisse entrer dans la tour il faut que le premier étage soit libre, c'est à dire qu'il ne contienne pas déjà un autre personnage.
Supposons que le rez-de-chaussée de la tour occupe une position $(row, col, 0)$, alors le premier étage est à la position $(row, col, 1)$. S'il est libre cela devient la nouvelle position de p, et sa direction devient orientée vers le haut. Par contre si le premier étage est occupé par un autre personnage, alors p *ne peut pas entrer dans la tour*. Il reste alors à sa place, à l'extérieur de la tour, mais là encore c'est la règle de rencontre avec un personnage qui s'applique : la direction de p est inversée.

Sortie de la tour par le sol

En descente, p ne pas pas non plus occuper le rez-de-chaussée. Il passera donc de l'étage 1 à une case extérieure libre, voisine de la tour, choisie au hasard. Sa direction (dans le plan) est réinitialisée au hasard. Si aucune case voisine extérieure n'est libre p reste bloqué dans la tour en attendant le prochain tour.

Sortie de la tour par le toit

Comme le rez-de-chaussée, le toit de la tour est une case spéciale qu'un personnage ne peut pas occuper. Ainsi, si le toit occupe une position (row, col, h) alors le dernier étage qu'un personnage peut occuper a pour position $(row, col, h-1)$. S'il veut sauter sur une autre tour, dont le toit est en (row', col', h') alors il se retrouvera en position $(row', col', h'-1)$, *à condition que cette position ne soit pas déjà occupée par un autre personnage*. Si cette position est déjà occupée alors p ne pourra pas sauter sur la tour et devra redescendre sa tour actuelle. Sinon il peut sauter et redescendra ensuite les étages de sa nouvelle tour.

Consignes obligatoires d'implantation

Occupants, positions et directions

L'espace de déplacement est virtuellement un cube, mais seulement certaines des positions en hauteur peuvent être occupées : celles correspondant à des tours. On verra donc plutôt cet espace comme une grille avec certaines positions en hauteur occupables.

Les positions sont décrites par une classe `Position` décrivant un triplet $(row, col, height)$.

Les directions seront aussi des triplets de type $(rowDir, colDir, floorDir)$, mais dont la valeur de chaque élément ne peut être que -1, 0 ou 1. Ainsi la position future d'un personnage en mouvement est simplement la somme de son vecteur position et de son vecteur direction.

Tout ce qui peut occuper une position doit dériver d'une classe abstraite `Occupant`. Pour l'instant, les différents types d'occupants vus sont les bords, les personnages, les rez-de-chaussée et les toits des tours. Mais vous devrez proposer d'autres types d'occupants (obstacles, monstres, pièges, ... à votre imagination!).

Les occupants qui peuvent bouger (pour l'instant les personnages seulement) doivent tous dériver d'une classe abstraite `Moving`.

La liste des cases occupées de l'espace de déplacement sera une table associant une position à son occupant : `Map<Position, Occupant>`. N.B. le sujet est rédigé de telle sorte qu'une position ne peut recevoir qu'un seul occupant à la fois (pas de superposition). Si vous souhaitez être plus général que cela et autoriser les superpositions (mais c'est plus dur à gérer, et non recommandé) il vous est possible de préférer une table associant une position à une liste d'occupants : `Map<Position, List<Occupant>>`.

Redirection à la demande

Lorsque la case cible d'un personnage p est occupée, c'est **toujours cet occupant ciblé qui décide de la redirection à appliquer à** p. Il est exigé, à l'exclusion de toute autre solution, que chaque occupant implante une méthode `void redirect(Moving m)` de telle sorte que quand p a pour cible une case occupée par un occupant o, on applique à p la méthode de redirection de o : `o.redirect(p)`.

Par exemple, la méthode `redirect` d'un bord fait rebondir un mouvant. Celle du rez-de-chaussée d'une tour déplace (quand c'est possible) un personnage au premier étage, etc.

L'avantage est que cela rend le jeu évolutif. On peut introduire ensuite d'autres occupants avec d'autres régles de redirection, ils coderont leurs propres règles avec la méthode `redirect`.

Au final, faire évoluer le jeu d'un tour consistera simplement à prendre un à un chaque mouvant, à calculer sa position cible, à le déplacer si cette position est libre, sinon à demander à l'occupant de la case cible de le rediriger.

Interfaces à implanter

Tous les mouvants devront implanter l'interface `Moveable` suivante :

```
public interface Moveable {
    Position getPresentPosition();
    Position getTargetPosition();
    boolean moveTo(Position pos);
}
```

Pour pouvoir rediriger les mouvants, tous les occupants devront implanter l'interface `Redirector` suivante :

```
public interface Redirector {
    void redirect(Moving m);
}
```

Extensions du sujet

D'autres occupants

Les fonctionnements présentés ci-dessus correspondent au minimum exigé pour avoir la moyenne (si c'est bien codé). Vous devrez proposer des extensions avec d'autres types d'occupants et de mouvants. Par exemple, on peut imaginer des obstacles fixes, ou des trappes qui vous envoient sur une autre case. Ou encore d'autres types de mouvants tels que des fantômes qui vous figent de peur pendant trois tours, etc. Votre imagination est reine mais attention, dans tous les cas vous devrez respecter le principe de la case cible qui redirige.

Gestion de priorités

Tout se passe au hasard dans le sujet tel qu'il est décrit. On peut cependant imaginer mettre en place des stratégies sous la forme de priorité à appliquer pour les déplacements des personnages. Par exemple, un personnage peut choisir quand c'est possible de sauter sur le toit d'une tour dans laquelle un autre personnage est en train de monter; ou d'empêcher un autre personnage de se rapprocher d'une tour qu'on possède; ... Dans ce cas, on pourra placer les cases cibles potentielles dans une file de priorité (`PriorityQueue`) pour qu'à chaque fois la meilleure soit prise.

Visuels

Le jeu se déroule de manière entièrement automatique et ne demande pas d'interaction avec l'utilisateur.

Les dimensions du jeu et son état initial sont laissés à votre initiative.

Les visuels sont laissés à votre initiative et aucune interface graphique n'est exigée étant donné que cela n'a pas été enseigné. On attend donc une visualition console de la grille, avec des informations complémentaires sur le contenu des tours. Vous pouvez toutefois de manière optionnelle réaliser une interface graphique si vous savez déjà le faire.