

UE Conception Orientée Objet

Jeu de l’oie

Exercice 1 : Jeu de l’oie

L’objectif de cet exercice est l’implémentation du “jeu de l’oie”, bien connu des plus jeunes. Ce jeu se présente sous la forme d’un plateau de 63 cases numérotées (plus une position de départ qui peut être considérée comme une 64ème case numérotée 0). Les différents joueurs, en nombre quelconque, jouent chacun leur tour. Ils lancent 2 dés à 6 faces et avancent leur pion d’un nombre de cases égal à la somme des dés. En fonction de la case d’arrivée il peut se produire plusieurs événements :

- si la case d’arrivée était déjà occupée, le pion du joueur remplace le pion déjà présent et ce dernier est placé sur la case précédemment occupée par le joueur. Il n’y a donc toujours qu’au plus un joueur par case, sauf pour la case de départ qui est un cas très particulier.
- si la case est une “case oie”, le joueur double le score du dé et avance donc encore son pion d’autant de cases qu’indiqué par les dés;
- si la case est une “case piège”, le joueur ne pourra plus jouer tant qu’il restera dans cette case (c’est-à-dire jusqu’à ce qu’un autre joueur tombe également sur cette case et que la première situation décrite se produise);
- si la case est une “case d’attente”, le joueur reste bloqué sans jouer pendant un nombre de tours prédéfini pour chaque case (sauf si un autre joueur arrive sur cette case et le renvoie donc vers sa case initiale, c’est au joueur arrivant de se retrouver en attente pour le nombre de tours initialement prévu pour la case);
- si la case est une “case de téléportation”, le pion est téléporté à une autre case prédéfinie;
- pour toutes les autres cases, il ne se produit rien de particulier.

Pour gagner et finir la partie, il faut qu’un joueur arrive exactement sur la case 63. Si un lancer de dés lui fait “dépasser” cette case, il revient en arrière du nombre de cases en excès. Par exemple, un joueur situé sur la case 57 fait 9 aux dés, il avance jusque la case 63 en dépensant 6 points et recule ensuite des 3 points restants pour arriver sur la case 60.

Dans la version originale du jeu, les “cases oie” sont les cases 9, 18, 27, 36, 45 et 54; les “cases piège” sont les cases 31 (le puits) et 52 (prison); la case 19 est une “case d’attente” pour 2 tours; les “cases de téléportation” sont les cases 6 (cheval) qui envoie en 12, 42 (labyrinthe) qui envoie en 30 et 58 (tête de mort) qui renvoie en 1. Cependant par la suite il ne faudra pas se focaliser sur ces cases, on veut avoir la possibilité d’éventuellement personnaliser le jeu de l’oie implémenté en modifiant les effets de chacune des cases, voire même le nombre de cases du plateau. Un exemple de partie est donné en annexe.

Nous nous intéressons donc maintenant à la conception objet de ce jeu. L’analyse a dégagé le besoin d’un certain nombre d’entités à modéliser :

- ▷ la classe **Game** : elle est caractérisée par un plateau et une liste de joueurs et permet de jouer une partie du jeu de l’oie, voici son diagramme UML :

Game
thePlayers : List<Player> # board : Board
<<constructor>> + Game(board : Board) <<methods>> + addPlayer(p :Player) +play()

La méthode `play()` déroule une partie jusqu’à son terme quand il y en a un (une partie peut être infinie si tous les joueurs sont dans des “cases piège”, on ne testera pas cette situation).
Jouer une partie consiste à faire jouer successivement chaque joueur selon les règles décrites ci-dessus.

- ▷ La classe **Board** permet de représenter le plateau de jeu contenant les différentes cases, en voici le diagramme UML :

<< abstract >> Board
final nbOfCells : int # theCells : Cell[]
<<constructor>> + Board(nbOfCells : int) <<methods>> # <i>initBoard()</i> + getCell(int numero) : Cell + getNbOfCells() : int

Un plateau est constitué d’un tableau de *nbOfCells* cases, plus une case numérotée 0 : la case de départ. On a donc en tout *nbOfCells+1* cases.
Chaque case numéro *i* du plateau est rangée dans la case d’indice *i* du tableau.
Le constructeur fait appel à la méthode `initBoard()`. C’est dans cette méthode que sont créées les différentes cases (`Cell`) qui constituent le plateau.

- ▷ L’interface `Cell` est donnée en annexe. Quand un lancer de dés amène un joueur sur une case, c’est le résultat de l’invocation de `consequence()` sur cette case qui indique quel est le numéro de la case réellement atteinte (par exemple dans le cas d’une “case de téléportation” c’est l’indice de la case destination de la téléportation qui est donné).

Toutes les entités seront à définir dans un paquetage appelé `oie`.

- Q 1 . Donnez l’algorithme de la méthode `play` de la classe `Game`.
- Q 2 . Donnez les diagrammes UML des entités nécessaires à la modélisation du jeu de l’oie. Comment gérer la “case 0” ?
- Q 3 . Ecrivez le programme correspondant.
En TP, utilisez l’environnement Eclipse pour coder ce projet :
 - commencez par les diagrammes UML,
 - puis saisissez la documentation,
 - en enfin (seulement) finalisez le code en complétant les corps des méthodes.

Annexes

Ces fichiers sont disponibles sur le portail.

La classe `oie.Player`

```
package oie;
import java.util.Random;
/** A player in the "jeu de l'oie" game */
public class Player {
    private static Random random = new Random();
    /** current cell of the player */
    protected Cell cell;
    /** name of the player*/
    protected String name;
    /**
     * @param name the name of this player
     * @param cell the strating cell of this player
     */
    public Player (String name, Cell cell){
        this.name = name;
        this.cell = cell;
    }
    /** */
    public String toString() {
        return this.name;
    }
    /** @return the current cell of the player */
    public Cell getCell() {
        return this.cell ;
    }
    /** changes the cell of the player
     * @param newCell the new cell
     */
    public void setCell(Cell newCell) {
        this.cell = newCell;
    }
    /** @return random result of a 1d6 throw*/
    private int oneDiceThrow() {
        return Player.random.nextInt(6)+ 1;
    }
    /** @return random result of a 2d6 thow */
    public int twoDicesThrow() {
        int result = oneDiceThrow() + oneDiceThrow();
        return result;
    }
}
} // Player
```

L'interface oie.Cell

```
package oie;

/**
 * Interface for the cells of the "jeu de l'oie" game.
 * Note that there can be only one player by cell,
 * the starting cell (index 0) excepted.
 */
public interface Cell {
    /**
     * @return true if and only if the player in this cell can freely
     * leaves the cell, else he must wait for another player to reach this cell
     * or wait a given number of turns
     */
    public boolean canBeLeaved();

    /** returns the index of this cell */
    public int getIndex();

    /**
     * returns the index of the cell really reached by a player when he reaches
     * this cell.
     * @param diceThrow the result of the dices when the player reaches this cell
     * @return the index of the cell effectively reached when a player reaches
     * this cell after the given throw of dices
     */
    public int consequence(int diceThrow);

    /** @return true iff a player is in this cell */
    public boolean isBusy();

    /** sets a player in this cell
     * @param player the new player in the sell
     */
    public void setPlayer(Player player);

    /** @return the player in this cell null if none */
    public Player getPlayer();
} // Cell
```

Un exemple de trace de partie

```
bilbo is in cell 0, bilbo throws 7, reaches cell 7
frodo is in cell 0, frodo throws 11, reaches cell 11
sam is in cell 0, sam throws 7, reaches cell 7
bilbo is sent to cell 0
bilbo is in cell 0, bilbo throws 3, reaches cell 3
frodo is in cell 11, frodo throws 3, reaches cell 14
sam is in cell 7, sam throws 5, reaches cell 12
bilbo is in cell 3, bilbo throws 6, reaches goose 9 and jumps to 15 goose = oie
frodo is in cell 14, frodo throws 5, reaches cell 19
sam is in cell 12, sam throws 8, reaches cell 20
bilbo is in cell 15, bilbo throws 6, reaches cell 21
frodo is in cell 19, frodo cannot play.
sam is in cell 20, sam throws 5, reaches cell 25
bilbo is in cell 21, bilbo throws 9, reaches cell 30
frodo is in cell 19, frodo cannot play.
sam is in cell 25, sam throws 5, reaches cell 30
bilbo is sent to cell 25
bilbo is in cell 25, bilbo throws 6, reaches cell 31
frodo is in cell 19, frodo throws 4, reaches cell 23
sam is in cell 30, sam throws 6, reaches goose 36 and jumps to 42
bilbo is in cell 31, bilbo cannot play
frodo is in cell 23, frodo throws 12, reaches cell 35
sam is in cell 42, sam throws 7, reaches cell 49
bilbo is in cell 31, bilbo cannot play
frodo is in cell 35, frodo throws 9, reaches cell 44
sam is in cell 49, sam throws 8, reaches cell 57
bilbo is in cell 31, bilbo cannot play
frodo is in cell 44, frodo throws 5, reaches cell 49
sam is in cell 57, sam throws 10, sam reaches cell 59.
bilbo is in cell 31, bilbo cannot play
frodo is in cell 49, frodo throws 9, reaches cell 58 and jumps to 1 téléportation
sam is in cell 59, sam throws 4, sam has won.
```

*même case que bilbo
bilbo repart à la case initiale de sam*

premier tour d'attente

*second tour d'attente
même case que bilbo
bilbo repart à la case initiale de sam*

*peut enfin jouer
jumps to 42
case 31 = case piège*

*il dépasse 63 donc retour arrière
case 63 atteinte partie finie*