UNIVERSITE DE FRANCHE-COMTE

Stoners : les pétrifieurs

Licence informatique, semestre 3

Programmation orientée objet

Rédacteurs : DUMONTET Sylvain et OSSETE GOMBE Bérenger

Groupe TP 2B

Enseignants : MASSON Pierre-Alain, GREFFIER Françoise

Décembre 2014

Remerciements

Nous tenons à remercier nos enseignants de Programmation Orientée Objet, Pierre-Alain Masson et Françoise Greffier pour leur direction et leur aide au cours de la réalisation de ce projet.

Sommaire

Table des illustrations

Glossaire

NPC : Non Playable Character (personnage non joueur). Désigne un personnage contrôlé par le programme (dans notre cas, tous les personnages sont des NPC).

Introduction

Dans le cadre de l'unité d'enseignement de Programmation Orientée Objet, il nous est demandé de réaliser un projet lors du semestre 3 de licence parcours informatique.

Ce projet nous demande de mettre en jeu les notions de classes, d'héritage, d'interface et de polymorphisme abordées en cours. C'est en binôme que nous avons réaliser ce projet, en utilisant le langage java.

1. Résumé de l'application

1.1 Le sujet

Le sujet peut être décomposé en trois parties :

- Générer un damier contenant des obstacles de différents types, des cases vides et différents personnages
- Afficher le damier avec les personnages
- Faire évoluer l'état des personnages sur le damier

1.2 Les améliorations apportées au sujet

- Pour ajouter de la visibilité vis à vis de l'évolution de l'état des personnages, nous avons décidé d'afficher ces derniers différemment s'ils sont pétrifiés ou non : les personnages pétrifiés sont affichés en minuscule (x, r, o,...) tandis que les personnages non pétrifiés sont affichés en majuscule (X, R, O,...).
- Nous avons ajouté également deux types de NPC afin d'illustrer la facilité d'extension de notre projet :
  - Les Wizards (magiciens) qui pétrifient les Stoners et dépétrifient les autres NPC.
  - Les Vampires pétrifient leur cible puis se téléportent.

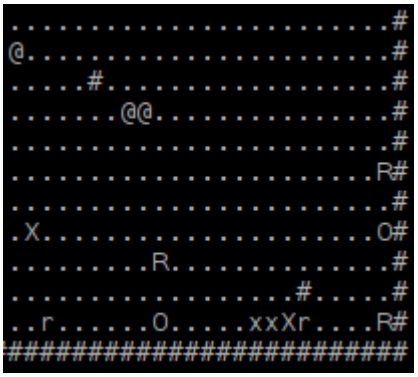

Figure 1. Affichage personnages pétrifiés ou non

- En plus du code source java, nous avons fourni un package exécutable de notre application.

2. Phase de conception

1.3 Diagramme de classe

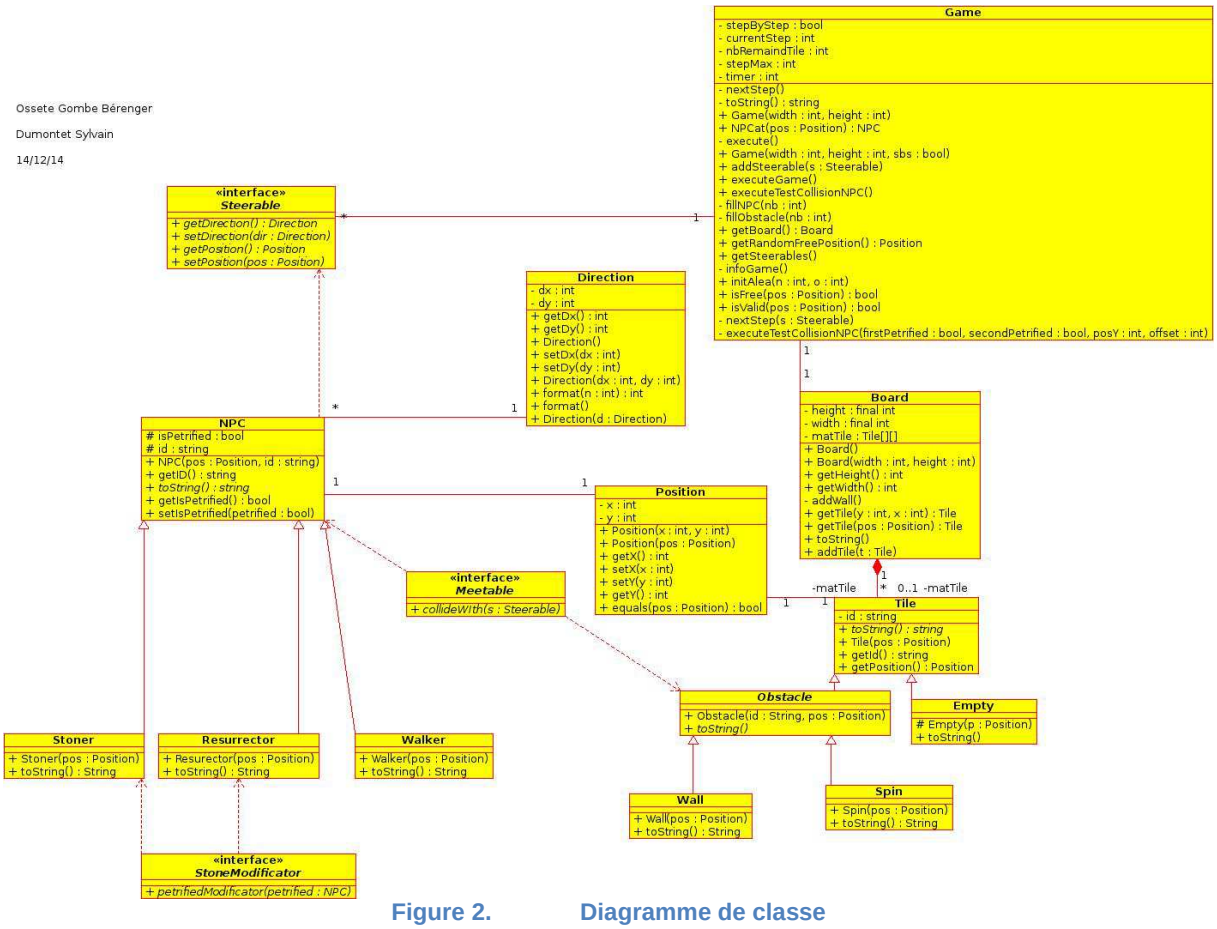

Figure 2. Diagramme de classe

Notre diagramme de classe est composé de 13 classes et de 3 interfaces.

Game (le jeu), permet la construction du jeu et son exécution. Game comporte d'une part **Board** (le damier), et d'autre part une ArrayList de **Steerable** (dirigeables).

Board est constitué de **Tile** (de cases) rangées dans une matrice. Ces **Tile** ont une **Position** (position) et peuvent être soit des **Obstacle**

(cases comportant un obstacle), soit des **Empty** (des cases sans obstacles). **Obstacle** implémente l'interface **Meetable** (rencontrable), ce qui signifie qu'ils pourront agir sur la direction d'un **Steerable** qui rentrera en collision avec eux. Les **Obstacle** peuvent être de plusieurs nature : les **Wall** (murs) et les **Spin** (tourniquets).

Le programme est conçu de telle sorte qu'il soit simple d'ajouter d'autres types d'obstacles : seuls l'affichage via la méthode toString() et l'interaction avec les personnages via la méthode collideWith(Steerable s) changeront des autres obstacles existants.

Game comporte également une ArrayList de **Steerable**, une interface implémentée par tout élément possédant une **Direction** (direction) susceptible d'être modifiée. Ce qui est le cas des **NPC** (personnages non jouables), qui possèdent une **Direction** et une **Position**. Tout comme les **Obstacle**, les **NPC** peuvent être rencontrés par d'autres éléments, c'est pourquoi ils implémentent la méthode **Meetable**.

Les **NPC** peuvent être de plusieurs nature : les **Stoner** (pétrifieur), les **Resurrector** (ressusciteur), et les **Walker** (marcheur). Les **Stoner** et les **Resurrector** peuvent respectivement pétrifier et dépétrifier d'autres personnages qu'ils rencontrent. Pour cela ils agissent sur l'attribut isPetrified possédé par tous les **NPC**, cette interaction ce fait via l'interface **StoneModificator**.

Le programme est conçu de telle sorte qu'il soit simple d'ajouter d'autres types de **NPC** : seuls l'affichage via la méthode toString() et l'interaction avec les personnages via la méthode petrifiedModificator(NPC petrified) de l'interface StoneModificator devront être modifiés par rapport aux **NPC** déjà existants.

1.4 Algorithmes intéressants

Algorithme permettant de détecter l'appui sur la touche entrée

```
while( running && this.currentStep < this.stepMax)
    {
        System.out.println(this);
        nextStep();

        if( this.stepByStep )
            {

                while(keyValue == 0)
                    {
                        try
                            {
                                keyValue = lecteur.read();

                            }
                        catch(Exception e){keyValue=0;}
                    }

                //on lit keyValue
                if(keyValue == (int)('q')) running = false;

                //
                keyValue = 0;

            }
        else
            {
                try{

                    Thread.sleep(this.timer);
                }catch(Exception e){e.printStackTrace();}
            }

        this.currentStep ++;

    }
```

Figure 3. surveillance de l'entrée clavier

3. Les points intéressants du programme

- L'algorithme permettant de retourner le NPC présent à une position donnée.

```
/**
 * retourne le NPC a la position pos ou retourne null
 * @param pos Position a tester
 * @return NPC reference du npc a la position pos
 */
public NPC NPCat(Position pos)
{
    NPC res = null;
    Steerable temp = null;

    int i = 0;

    while(temp == null && i < this.steerable.size() && this.steerable.size() !=0 )
        {

            if( this.steerable.get(i).getPosition().equals(pos) )
                {
                    temp = this.steerable.get(i);
                }

            i++;
        }



    res = ((NPC)temp);

    return res;
}
```

Figure 4. méthode NPCat

- L'interface permettant de changer l'état de pétrification des personnages. Il est intéressant de voir que ce ne sont pas les attributs de l'instance courante du NPC qui sont modifiés, mais les attributs du personnage qui rencontre l'instance courante.

```php
/**
 * @author Sylvain DUMONTET
 * @author Berenger OSSETE GOMBE
 */
public interface StoneModificator
{

    /**
     * @param petrified NPC dont l'etat de petrification va etre modifie par le StoneModificator
     */
    abstract void petrifiedModificator(NPC petrified);
}
```

Figure 5. interface StoneModificator

- Le formatage des données dans la classe direction. Pour laisser plus de liberté dans la façon de modifier la direction des personnages (par exemple ajouter un obstacle qui dévie de 45° au lieu de 90°), nous avons ajouté une méthode qui formate les attributs d'une Direction pour être sur qu'elle soit toujours valide, quelque soit les modifications qu'on lui apporte.

```php
/**
 * assure la coherence de dx et dy pour le constructeur
 * @param n int a formater
 * @return int formate a 0, 1 ou -1 selon le signe de n
 */
public int format(int n)
{
    if(n > 0) return 1;
    if(n < 0) return -1;
    return 0;
}

/**
 * assure la coherence de la direction en applicant format(int n) sur chacune des composantes de l'instance courante
 */
public void format()
{
    this.dx = format(this.dx);
    this.dy = format(this.dy);
}
```

Figure 6. méthode format

Conclusion

Le programme fonctionne avec toutes les fonctionnalités demandées dans le sujet. En outre, nous nous sommes efforcés d'orienter la conception de ce programme de façon à le rendre maintenable et qu'il puisse facilement évolué.

Ce projet nous aura permis d'aborder la conception d'un programme en binôme, nous obligeant à utiliser des outils de communication explicites comme le diagramme uml. Cela nous aura également permis de ce familiariser davantage avec les interfaces et de les mettre en œuvre.