Benjamin BEAUCAMP
Muruo WANG

TP4 – Programmation Objet

Gestion d'une multitude de protagonistes dans notre jeu

Introduction

Dans ce TP, nous allons :

- Manipuler les conteneurs génériques Java pour gérer un grand nombre de personnages, monstres et objets
- Comparer les performances de différents conteneurs
- Etudier les différentes manières de manipuler ces conteneurs et les impacts potentiels de nos choix d'implémentation en termes de performances

1) Retour sur le TP précédent

Nous avons modifié le calcul de la norme euclidienne par la méthode **distance** dans **Point2D** en remplaçant Math.pow(x, 2) par x * x (plus efficace que le calcul d'une exponentielle qui est fait dans Math.pow)

2) Gestion de multiples protagonistes

Pour gérer un nombre non connu à l'avance de créatures, nous allons utiliser des collections (de taille variable). Comme nous allons devoir ajouter, supprimer ou modifier régulièrement les créatures, nous privilégierons les collections **LinkedList** ou **ArrayList**. On propose deux solutions :

- Créer un attribut **listCreatures** de type LinkedList<Creature> ou ArrayList<Creature> dans **World** qui contient toutes les créatures que l'on souhaite manipuler (archers, guerriers, loups, …) et un attribut **listObjets** de type LinkedList<Objet> ou ArrayList<Objet> qui contient tous les objets (potions de soin, de mana, …)
- Créer un attribut de type LinkedList ou ArrayList pour chaque classe fille que l'on veut manipuler (archer, guerrier, mage, paysan, loup, lapin, potions de soin et potions de mana)

La première solution nous permet bien de stocker des archers, loups, lapins, … dans un même conteneur car ils héritent tous de **Creature**. Cependant, lorsqu'on accède à un élément du tableau, on ne sait pas si c'est un archer, un loup, un lapin, …
Pour résoudre ce problème, on pourrait utiliser une table de hachage contenant des listes de créatures et dont les clés seraient les noms des classes ("Archer", "Paysan", …). Lorsqu'on récupère une liste de créatures, on connaîtrait alors le type plus spécifique des éléments contenus dans cette liste.

Nous avons choisi d'implémenter la deuxième solution avec des ArrayList car on peut accéder à un élément en O(1) (O(n) pour une LinkedList) et en ajouter un nouveau en O(1) ou O(n) quelques fois.

Changements à apporter à World :

On commence par créer les attributs suivant (qui sont des ArrayList) : **ListArchers**, **ListGuerriers**, **ListMages**, **ListPaysans**, **ListLoups**, **ListLapins**, **ListSoins**, **ListManas**.

Il faut aussi changer la méthode **afficheWorld()** qui s'appuyait sur les attributs merlin, peon, guillaumeT, … On décide de n'afficher que le nombre de créatures/d'objets de chaque classe ainsi que la taille du monde (attribut ajouté plus tard).

Enfin, il faudra modifier **creeMondeAlea()** (fait plus tard dans le TP).

Test :

Pour illustrer le bon fonctionnement de notre solution, on change temporairement notre méthode **creeMondeAlea** pour qu'elle ajoute entre 0 et 10 Lapins, Guerriers et Paysans à leur listes respectives. On affiche la taille de la liste ainsi que le premier élément pour montrer qu'il est bien initialisé. Le résultat est le suivant :

```
La taille de liste des Lapins : 4
Permier élément dans la liste :
Points de vie : 10
Position : [0 ; 0]
Points d'attaque : 5
Pourcentages d'attaque : 50
Points de parade : 2
Pourcentages de parade : 20

La taille de liste des Guerriers : 5
Permier élément dans la liste :
Nom : Guerrier
Points de vie : 120
Position : [0 ; 0]
Points d'attaque : 20
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 1

La taille de liste des Paysans : 3
Permier élément dans la liste :
Nom : Paysan
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 1
```

3) Parcours des conteneurs Java

Pour notre premier parcours de conteneur, nous avons crée une méthode dans TestSeance3 qui initialise un ArrayList avec 10 archers. Ensuite, nous avons effectué un parcours de cet ArrayList basé sur sa taille (boucle **for (int i=0 ; i<listArchers.size() ; i++){...}**). Comme la sortie est trop longue, nous ne présentons que la première et la dernière itération :

```
Itération 0
Nom : Archer
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
Nombre de flèches  5
```

```
Itération 9
Nom : Archer
Points de vie : 100
Position : [0 ; 0]
Points d'attaque : 15
Pourcentages d'attaque : 70
Points de parade : 5
Pourcentages de parade : 60
Points de mana : 5
Dégâts de magie : 10
Pourcentage de magie : 30
Pourcentage de résistance à la magie : 30
Distance d'attaque maximale : 5
-Nombre de flèches  5
```

4) On ne se marche pas sur les pieds (ni sur les pattes) !

On commence par ajouter un attribut entier **tailleMonde** à World. On utilise la méthode **contains(Object o)** pour vérifier si le point est déjà dans la liste ou pas. (On a parcouru toute la liste pour vérifier dans le dernier TP3 car on ne connaissait pas cette méthode). Mais on a rencontré un bug car notre méthode **equals(Point2D p)** de **Point2D** ne redéfinit pas celle de **Object**. **contains** comparait donc les adresses mémoires de nos positions. **equals** a été redéfini de la manière suivante :

```
public boolean equals(Object o) {
    if (o == null || o.getClass() != this.getClass()) {
        return false;
    } else {
        Point2D p = (Point2D) o;
        return (this.x == p.getX() && this.y == p.getY());
    }
}
```

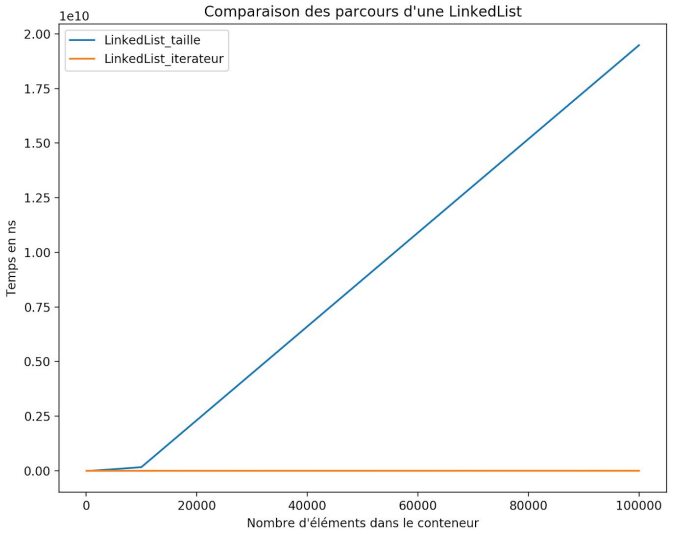
La génération des positions est faite dans la méthode **generePositionsAleatoires** de **World.**

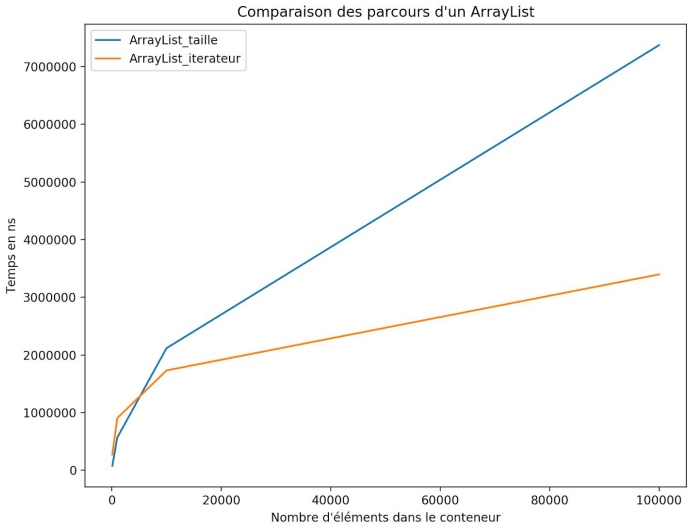
On peut ensuite générer 100 positions aléatoirement dans un monde de taille 50x50. Voici le résultat:

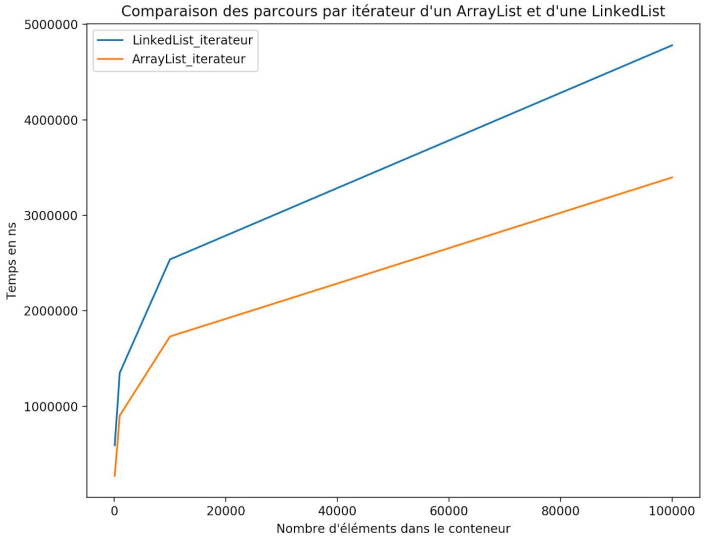
```
[16 ; 17][6 ; 28][14 ; 7][14 ; 2][11 ; 0][6 ; 48][7 ; 0][27 ; 16][28 ; 29][25 ; 31]
[41 ; 13][34 ; 14][2 ; 29][9 ; 23][0 ; 31][43 ; 27][24 ; 33][19 ; 25][29 ; 47][15 ; 8]
[35 ; 44][26 ; 8][33 ; 0][24 ; 1][9 ; 6][15 ; 1][39 ; 33][29 ; 2][48 ; 24][46 ; 23]
[21 ; 47][9 ; 32][23 ; 15][29 ; 15][12 ; 32][46 ; 16][4 ; 40][8 ; 33][31 ; 45][16 ; 22]
[27 ; 28][36 ; 27][13 ; 15][45 ; 49][21 ; 3][27 ; 10][6 ; 10][35 ; 23][10 ; 26][49 ; 30]
[13 ; 12][49 ; 10][6 ; 25][31 ; 15][42 ; 36][31 ; 13][11 ; 46][14 ; 28][2 ; 33][5 ; 46]
[31 ; 49][41 ; 39][39 ; 36][3 ; 33][22 ; 15][0 ; 46][13 ; 22][22 ; 28][41 ; 41][38 ; 17]
[23 ; 20][2 ; 34][10 ; 31][16 ; 40][48 ; 22][8 ; 48][4 ; 26][14 ; 12][17 ; 20][19 ; 5]
[13 ; 31][39 ; 42][22 ; 6][9 ; 12][9 ; 15][14 ; 5][21 ; 38][14 ; 19][9 ; 4][35 ; 27]
[47 ; 46][2 ; 25][41 ; 32][11 ; 32][31 ; 28][27 ; 0][38 ; 1][25 ; 5][6 ; 26][11 ; 39]
```

5) Comparaison des différents conteneurs Java

Pour comparer les différents conteneurs, nous avons choisi d'utiliser seulement **generePositionsAleatoires** car l'initialisation de personnages n'était pas utile. On ne manipule donc que des ArrayList ou des LinkedList de Point2D. Nous n'avons pas de temps pour des conteneurs de taille 1 000 000 car l'exécution était trop longue. Il n'a pas été difficile de passer d'une LinkedList à une ArrayList car les méthodes sont les mêmes (il suffit de changer le type et le constructeur lors de la déclaration de la variable). Pour d'autres conteneurs, le résultat ne serait pas Les résultats sont les suivants :


Comparaison des parcours d'une LinkedList
1e10
LinkedList_taille
LinkedList_iterateur
Temps en ns
Nombre d'éléments dans le conteneur


Comparaison des parcours d'un ArrayList
ArrayList_taille
ArrayList_iterateur
Temps en ns
Nombre d'éléments dans le conteneur


Comparaison des parcours par itérateur d'un ArrayList et d'une LinkedList
LinkedList_iterateur
ArrayList_iterateur
Temps en ns
Nombre d'éléments dans le conteneur

On remarque que les parcours sont plus efficaces avec un itérateur (surtout quand le nombre d'éléments est grand). Par exemple pour un ArrayList à 100 éléments, le temps de création de l'itérateur est trop long par rapport au parcours effectif, ce qui fait qu'utiliser l'itérateur est plus lent.

En comparant ArrayList et LinkedList, ArrayList est plus efficace ce qui confirme notre choix initial.

6) Retour à WoE

Pour ajouter la condition de distance supérieure à 3, il faut modifier notre fonction **generePositionsAleatoires**. Nous n'avons pas eu le temps de le faire à cette séance car cela implique de parcourir le tableau positions alors que nous utilisons actuellement la méthode **contains**.

Pour que les entités ne soient pas à la même position, on peut gérer une ArrayList de positions occupées comme un attribut de **World**. La méthode **deplace** peut être modifiée pour prendre un **World** en paramètre (ou seulement la liste des positions occupées) afin de vérifier que la nouvelle positions est bien libre.

7) Conclusion

Dans ce TP, on a utilisé les structures de donnée ArrayList et LinkedList pour générer un grand nombre d'entités. En comparant les performances de différents conteneurs, on trouve que :

- ArrayList est plus efficace que LinkedList pour le parcours
- Si on va parcourir un conteneur avec un grand nombre d'éléments, il faut utiliser l'itérateur.

Nous avons aussi généralisé notre méthode permettant d'initialiser les positions des protagonistes aléatoirement à des positions distinctes. Le code est aussi plus simple grâce à la méthode **contains** que nous avons trouvée dans la Javadoc.