Rapport TP3 : SMA - Billes / Shelling / proies-prédateurs / Cinq ou plus

François Lepan - Alexis Linke

4 octobre 2013

1 Choix

1.1 Billes

Nous sommes partis sur un système de déplacement sur une grille case par case. Deux agents ne peuvent se retrouver sur une même case.

Nous avons fait ce choix car le système de collision ainsi que la représentation sur un `JPanel` est plus facile à gérer qu'un système de déplacement libre.

Les couleurs des Billes sont fixés lors de leurs création.

1.2 Schelling

Pour représenter le modèle de Schelling nous avons choisi une population divisée en deux type : la population bleue et la population jaune, d'égale importance. Chaque membre peut regarder dans le voisinage de Moore pour estimer son niveau de satisfaction et s'il n'est pas satisfait il se téléporte aléatoirement dans l'environnement. Un fichier *Schelling.txt* est créé lors de l'exécution du code donnant le taux de satisfaction sur la durée de la simulation.

1.3 Proies - Prédateur

Pour le modèle proies - prédateur nous avons avons choisis de créer deux sous classe de Agent l'un pour les proies et l'autre pour les prédateurs. Les positions ainsi que les directions sont défini aléatoirement lors de la création de ces deux Agents. L'environnement de déplacement est torique. Deux fichier sont créer lors de la simulation :

- *ages_prey_pred.txt* : contenant les ages de tous les agents à chaque tour de simulation.
- *population_prey_pred.txt* : contenant le nombre de proie et de prédateur à chaque tour de simulation.

1.4 5 ou plus

Pour le jeu du cinq ou plus nous avons créé quatre classes :

- *Plan* : Qui est l'environnement ou se déplace les Agents,
- *Player* : qui est l'agent qui demande à l'utilisateur d'entré des coordonnées,
- *Token* : qui est l'agent qui regarde si il forme une ligne ou non avec d'autre de ses congénère
- *MAS_FiveOrMore* : qui est Le System Multi-agent pour cette simulation.

Voici le déroulement pour ce jeu tout par tour :

- Trois jetons sont créés et positionnés sur l'environnement,
- les jetons regardent un à un si ils forment une ligne de cinq ou plus,
- on vérifie si le plateau de jeu est plein,
- on retire les jetons qui forment une ligne de cinq ou plus du même type,
- on met à jour la vue de l'environnement
- et enfin le joueur choisi de faire un déplacement.

Pour la vérification de présence d'une ligne de cinq ou plus, chaque jeton regarde dans quatre directions sur huit (en haut à droite, à droite, en bas a droite et en bas). Cela suffit car il y aura au moins un jeton sur cinq qui trouvera la ligne à enlever.

Pour la recherche du chemin le plus cour (*cf.* Fig. 1) :

```
On initialise un tableau de Integer de la meme taille que l'environnement avec :
        - 0 pour l'arrive
         integer.MAX_VALUE pour les agents déjà présent.

Pour (tous les indices des voisins trouvé)
        Pour les case autour de l'agent courant
                Si la case est vide
                        on met sa valeur +1
                        on rajoute cet agent dans la liste des voisins trouvé
                Sinon si l'agent présent est le depart
                        on sort de la boucle -> il y a un chemin
                fin si
        fin pour
fin pour

Si on est sortie de la boucle avant la fin -> il y a un chemin
sinon on redemande un chemin à l'utilisateur
```

2 UML

Fig. 2 (Il est aussi disponible en .png dans l'archive)

3 Compilation + fonctionnement

Compilation

Se mettre dans le dossier contenant le dossier src

```
mkdir bin
javac -sourcepath src -d bin src/core/Simulation.java
```

Ne pas bouger du dossier

Execution

Si on rentre un nombre de tour = -1 alors c'est infini

Pour la Simulation des Billes :

```
java -cp bin core.Simulation -b <taille> <nb agent> <nb tour> <delai entre chaque tour>
```

Pour la Simulation du modèle de Shelling :

```
java -cp bin core.Simulation -s <taille> <nb habitant> <seuil tolerance> <nb tour> <delai>
```

Avec le seuil de tolérance compris entre 0 et 1.

Pour la Simulation du modèle de Proies - Pédateur :

```
java -cp bin core.Simulation -w <taille> <nb proies> <nb pred> <tps reprod proies>
   <tps reprod pred> <tps faim> <delai>
```

Pour la Simulation du jeu Cinq ou plus :

```
java -cp bin core.Simulation -f
```

Exemples

Billes :

```
java -cp bin core.Simulation -b 100 50 -1 5
java -cp bin core.Simulation -b 10 5 100 5
java -cp bin core.Simulation -b 50 40 -1 5
```

Shelling :

```
java -cp bin core.Simulation -s 100 9750 0.3 -1 1
java -cp bin core.Simulation -s 100 9750 0.6 -1 1
java -cp bin core.Simulation -s 50 2000 0.7 -1 100
```

Proies-Prédateurs :

```
java -cp bin core.Simulation -w 50 1040 326 4 10 6 -1 50
java -cp bin core.Simulation -w 10 10 5 4 10 6 -1 50
java -cp bin core.Simulation -w 35 100 30 4 10 6 -1 50
```

Cinq ou plus :

```
java -cp bin core.Simulation -f
```

4 Problème

Nous n'avons plus de problèmes concernant proie - prédateur et Schelling. Les valeurs sont bonnes (*cf.* figure avec le rapport)

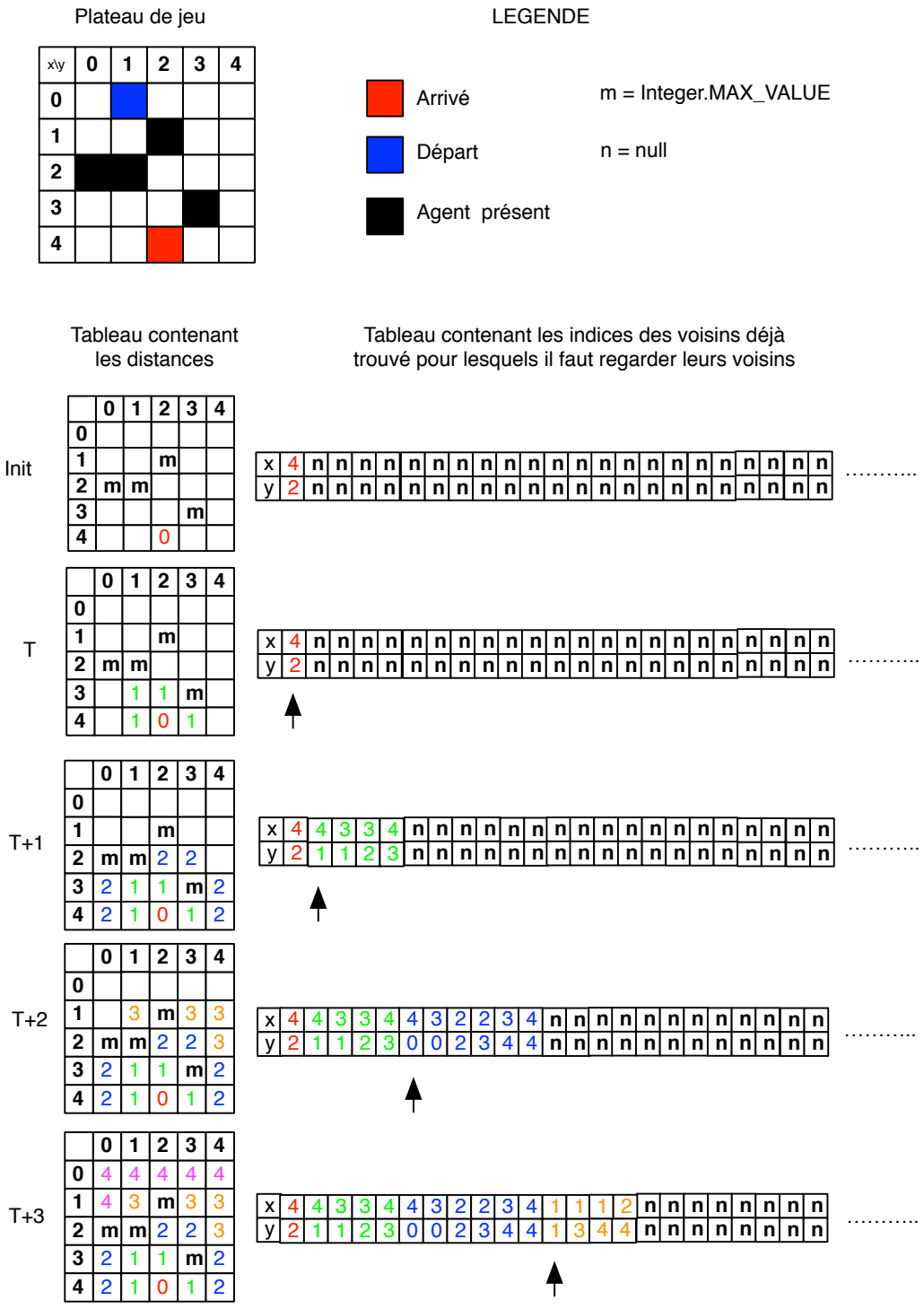

FIGURE 1 – Recherche du chemin le plus cour

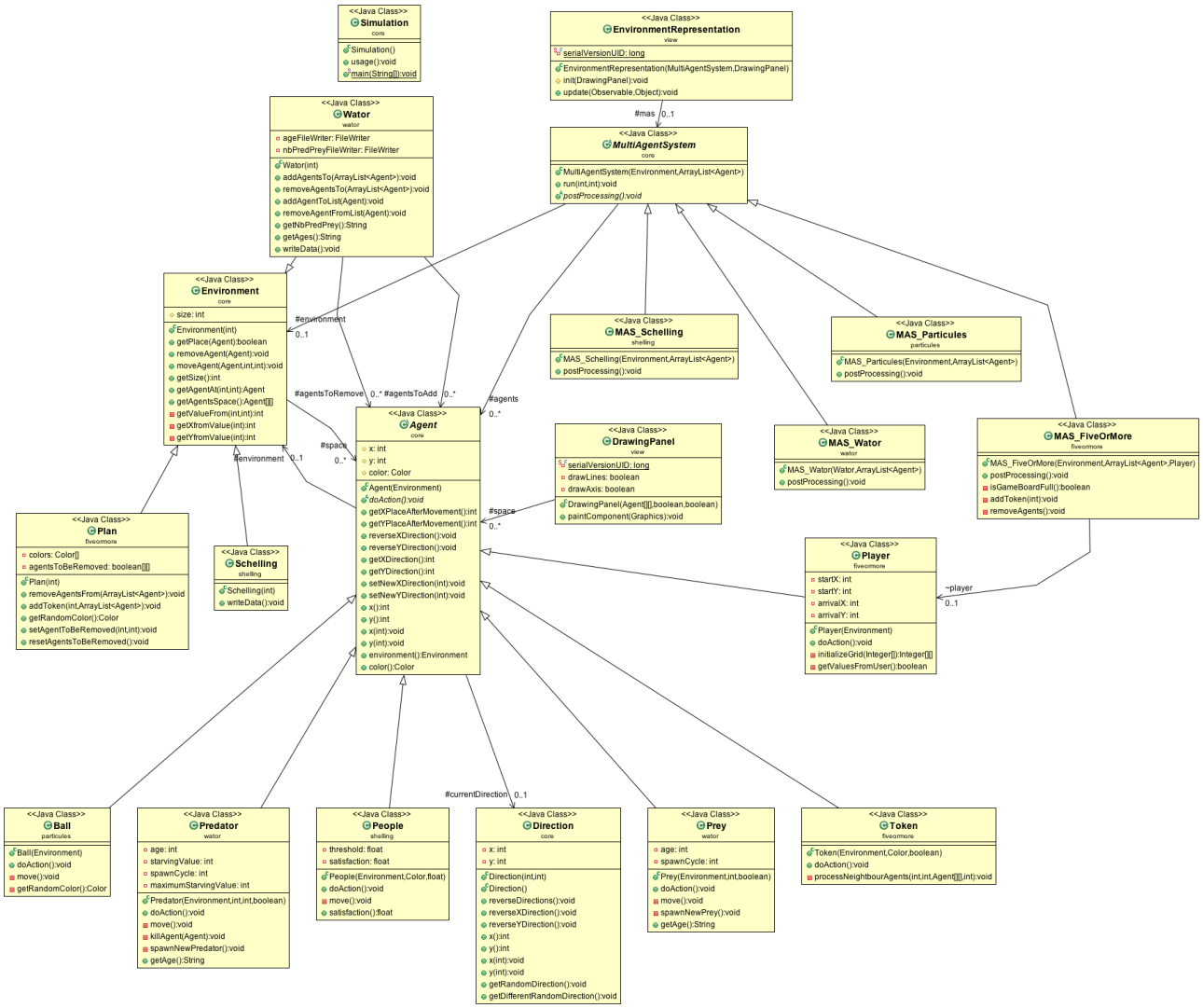

FIGURE 2 – L'UML du projet entier