Faculté des Sciences et des Techniques Nantes
X3I0020 : Programmation objet
Groupe 302C
Irena Rusu

Projet de programmation objet n°2

Galaxy Wars

Rapport

fait par

Silvan Habenicht

12 décembre 2016

Table des matières

1 Structure des objets

1.1 Galaxie

```
class Galaxie
```

Variables d'instance

```
static int            hauteur     // Constantes.hauteur
static int            largeur     // Constantes.largeur
ArrayList<Planete>    planetes    // tous les planetes dans la galaxie
ArrayList<Vaisseau>   vaisseaux   // tous les vaisseaux dans la galaxie
ArrayList<Espece>     especes     // tous les especes dans la galaxie
```

Constructeur

```
Galaxie()
```

Le constructeur crée un nombre de planètes dépendant de `Constantes.NombrePlanetes` et un nombre d'espèces dépendant de `Constantes.NombreEspeces`. Il initialise une planète natale pour chaque espèce et crée deux vaisseaux qui sont placés autour.

API

`ArrayList<Planete> getPlanetes()`	retourner **`planetes`**
`ArrayList<Vaisseau> getVaisseaux()`	retourner **`vaisseaux`**
`ArrayList<Especes> getEspeces()`	retourner **`especes`**
`static int` `getHauteur()`	retourner *`hauteur`*
`static int` `getLargeur()`	retourner *`largeur`*
`ArrayList<Planete> getPlanetesVoisins(Entite entite)`	retourne une liste avec tous les `Planetes` qui sont autour d'`entite`
`ArrayList<Vaisseau> getVaisseauxVoisins(Entite entite)`	retourne une liste avec tous les `Vaisseaux` qui sont autour d'`entite`
`void` `ajouter(Vaisseau vaisseau)`	ajouter `vaisseau` dans **`vaisseaux`**
`void` `supprimer(Vaisseau vaisseau)`	supprimer `vaisseau` dans **`vaisseaux`**

1.2 Especе

```
class Especе
```

Variables d'instance

```
Color                color          // sert à la représentation graphique
                                    // et comme identifiant unique
double               natalite       // taux entre 0.05 et 0.1
double               productivite   // taux entre 0.01 et 0.05
ArrayList<Planete>   planetes       // planetes possédées par cette espèce
ArrayList<Vaisseau>  vaisseaux      // vaisseaux possédés par cette espèce
```

Constructeur

```
Especе(Color color)
```

Dans le constructeur, une valeur aléatoire est générée pour **natalite** et **productivite**.

API

`Color getColor()`	retourner **color**
`double getNatalite()`	retourner **natalite**
`double getProductivite()`	retourner **productivite**
`ArrayList<Planete> getPlanetes()`	retourner **planetes**
`ArrayList<Vaisseau> getVaisseaux()`	retourner **vaisseaux**
`void addPlanete(Planete p)`	ajouter p dans **planetes**
`void addVaisseau(Vaisseau v)`	ajouter v dans **vaisseaux**
`void removePlanete(Planete p)`	supprimer p dans **planetes**
`void removeVaisseau(Vaisseau v)`	supprimer v dans **vaisseaux**

1.3 Entite

```
abstract class Entite
```

Variables d'instance

```
Galaxie       galaxie
Espece        proprietaire
int           yPosition     // ordonnée
int           xPosition     // abscisse
final Espece  inoccupe      // une espece avec Color.WHITE
```

Constructeur

```
Entite(Galaxie galaxie, int yPosition, int xPosition)

Entite(Galaxie galaxie, int yPosition, int xPosition, Espece proprietaire)
```

Dans le premier constructeur, **proprietaire** est initialisé avec **inoccupe**.
Dans le deuxième avec l'`Espece` qui était rendue comme paramètre.

API

`int getY()`	retourner **yPosition**
`int getX()`	retourner **xPosition**
`Espece getProprietaire()`	retourner **proprietaire**
`Galaxie getGalaxie()`	retourner **galaxie**
`void setY(int y)`	**yPosition** reçoit le valeur de `y`
`void setX(int x)`	**xPosition** reçoit le valeur de `x`
`void setProprietaire(Espece espece)`	`espece` devient **proprietaire**
`boolean estOccupe()`	retourne **true** ssi **inoccupe** et **proprietaire** sont différents
`void setInoccupe()`	**inoccupe** devient **proprietaire**

1.3.1 Planete

```
class Planete extends Entite
```

Variables d'instance

```
int       taille        // population maximale
int       population    // population réelle
Vaisseau  construction  // le vaisseau qui est construit sur cette planète
```

Constructeur

```
Planete(Galaxie galaxie, int yPosition, int xPosition)
```

Le constructeur initialise **taille** avec une valeur aléatoire entre `Constantes.`**`PlaneteTailleMin`** et `Constantes.`**`PlaneteTailleMax`**.

API

`int getTaille()`	retourner **taille**
`int getPopulation()`	retourner **population**
`void multiplierPopulation()`	**population** est multipliée par le taux de natalité du propriétaire
`void creerVaisseau()`	augmenter l'intégrité de **construction** en appelant la fonction `construire(`**`population`**`)`, si la construction est complet le lancer et créer un nouveau vaisseau
`void recharge(Vaisseau vaisseau)`	augmente le carburant de **vaisseau** de 5
`void colonise(Vaisseau vaisseau)`	colonisation de la planète avec les ressources de **vaisseau**
`void colonise(Espece espece, int population)`	colonisation de la planète avec les ressources rendues comme paramètres
`void attaquer()`	**population** est diminuée d'un valeur aléatoire entre 5 et 10, la planète devient inoccupé si sa population soit 0

1.3.2 Vaisseau

```
class Vaisseau extends Entite
```

Variables d'instance

```
int         resistance      // intégrité maximale
int         integrite
Propulsion  propulsion
Equipement  equipement
boolean     enConstruction  // vrai pendant le procès de la construction
```

Constructeur

```
Vaisseau(Galaxie galaxie, int yPosition, int xPosition, Espece espece,
          boolean enConstruction)
```

Le constructeur initialise **resistance** avec une valeur aléatoire entre `Constantes.`**VaisseauResistanceMin** et `Constantes.`**VaisseauResistanceMax**. Il assigne une Propulsion de type aléatoire et crée un nouveau Equipement. Si **enConstruction** est vrai, **integrite** est initialisé avec 0, sinon avec la valeur de **resistance**.

API

`int getTaille()`	retourner **resistance**
`int getIntegrite()`	retourner **integrite**
`Propulsion getPropulsion()`	retourner **propulsion**
`boolean estEnConstruction()`	retourner **enConstruction**
`String getStringRepresentation()`	retourne un String représentant le type de propulsion et le type d'équipement
`void construire(int population)`	augmente **integrite** d'un vaisseau qui est en construction, dépendant de `population` et le taux de productivité
`void lancer()`	commence le mouvement et met **enConstruction** `false`
`void bouger()`	mouvement du vaisseau et effectuer des actions de l'**equipement**
`void attaquer()`	**integrite** est diminuée d'un valeur aléatoire entre 1 et 2, le vaisseau se détruit, si l'intégrité devient 0

1.4 Composant

```
abstract class Composant
```

Variables d'instance

```
Vaisseau  vaisseau  // propriétaire du composant
```

Constructeur

```
Composant(Vaisseau vaisseau)
```

API

`Vaisseau getVaisseau()`	retourner **vaisseau**

1.4.1 Equipement

```
class Equipement extends Composant
```

Variables d'instance

```
boolean  estPolyvalent
```

Constructeur

```
Equipement(Vaisseau vaisseau)
```

API

`boolean estPolyvalent()`	retourner **estPolyvalent**
`void accomplir()`	cherche pour des voisins du vaisseau correspondant et effectue des actions dépendant de leur type et leur propriétaire

1.4.2 Propulsion

```
abstract class Propulsion extends Composant
```

Variables d'instance

```
int         portee         // intégrité maximale
int         carburant      // intégrité maximale
final int   carburantMax   // intégrité maximale
String      string
```

Constructeur

```
Propulsion(Vaisseau vaisseau, String string)
```

Le constructeur de Propulsion initialise **portee** avec une valeur aléatoire entre `Constantes.`**PropulsionPorteeMin** et `Constantes.`**PropulsionPorteeMax**. En outre, **carburant** et **carburantMax** sont initialisés avec un valeur entre `Constantes.`**PropulsionCarburantMin** et `Constantes.`**PropulsionCarburantMax**.

API (+ sous-classes)

`String getString()`	retourner **string**
void `ajouteCarburant(int c)`	changer **carburant** de la valeur c, s'il soit plus petit que 0, détruire le vaisseau
void `mouvementDiagonal()`	consommer carburant ; faire un mouvement diagonal vers un point cardinal aléatoire, la distance dépend de **portee**
void `mouvementLineaire()`	consommer carburant ; faire un mouvement lineaire vers un point cardinal aléatoire, la distance dépend de **portee**
class `PropulsionDiagonal extends Propulsion`	
void `mouvement()`	exécuter `mouvementDiagonal()`
class `PropulsionLineaire extends Propulsion`	
void `mouvement()`	exécuter `mouvementLineaire()`
class `PropulsionOmnidirectionnel extends Propulsion`	
void `mouvement()`	exécuter `mouvementDiagonal()` ou `mouvementLineaire()` aléatoirement

2 Modifications dans les classes données

Pour éviter trop de changement dans les classes `Simulation` et `Affichage`, j'ai décidé de gérer les entités dans la galaxie avec l'aide de deux listes qui contiennent tous les vaisseaux et tous les planètes. Un désavantage de cette manière est une procédure plutôt défavorable pour déterminer les voisins d'une entité. Il est actuellement nécessaire d'itérer tous les entités et tester pour chacun si les coordonnées sont avoisinants. Pour optimiser, on aurait également pu créer un tableau bidimensionnel d'entités qui représente la galaxie, pour pouvoir déterminer les voisins par leur position dans le tableau.

2.1 Simulation

Pour la classe `Simulation`, j'ai ajouté une variable de classe `Galaxie galaxie` pour pouvoir accéder tous les informations sur la galaxie comme la classe `Galaxie` n'est pas statique.

`victoire()`

Le jeu/la simulation termine après un certain nombre de tours ou en cas de victoire. D'après mon implémentation, la victoire est équivalent avec l'état ou il reste une seule espèce survivant. Victoire retourne donc vrai, si et seulement si la population de tous les espèces sauf une, soit 0.

`main()`

`// Exécution des étapes du tour courant`
Cette partie a été implémenté dans une fonction privée `executeTour()`.

`// Affichage d'un bref rapport textuel`
Cette partie a été implémenté dans une fonction privée `affichage(int tour)`.

`// rafraîchissement de la grille`
Pour cette partie j'ai supprimé tout le code donné, sauf l'appel de la fonction `rafraichir(ArrayList<Planete> planetes, ArrayList<Vaisseau> vaisseaux))` qui prend maintenant deux listes des objets `Planete` et `Vaisseau` à la place des listes de tableaux d'entiers. Les modifications nécessaires sont décrit dans la section pour la classe `Affichage`.

`executeTour()`

Cette fonction fait une copie de la liste de planètes et de vaisseaux dans `galaxie`. Pour chaque planète occupé elle appelle les fonctions `multiplierPopulation()` et `creerVaisseau()`, pour chaque vaisseau la fonction `bouger()`.

`affichage(int tour))`

La méthode `affichage()` calcule pour chaque espèce les vaisseaux, les planètes et la population entière. Les résultats sont affichés par `System.out`.

2.2 Affichage

Dans `Affichage`, des variables **`lesPlanetes`** et **`lesVaisseaux`** sont maintenant des listes de leur objet correspondant et pas plus de tableaux d'entiers.

`affichePlanete()` et `afficheVaisseau()`

Les deux fonctions ne prennent plus chaque spécification comme paramètre, car toutes les informations sont contenues dans un seul paramètre du type `Planete` resp. `Vaisseau`. Les variables `x, y, r, ...` sont remplacées par les « getters » correspondantes.

2.3 Constantes

Les constantes suivantes ont été ajouté dans la classe :

```
NombrePlanetes // nobre de planètes dans la galaxie
NombreEspeces // nobre d'especes dans la galaxie
PropulsionCarburantMin // carburant minimal d'une propulsion
PropulsionCarburantMax // carburant maximal d'une propulsion
```

3 Utilisation

Le programme peut être lancé comme d'habitude. La fonction principale est toujours dans la classe `Simulation` et tous les paramètres sont déjà définis dans la classe `Constantes`. La simulation, en standard, a trois espèces et dix planètes. Chaque espèce commence avec deux vaisseaux qui sont placés au nord et à l'ouest de la planète natale. La construction des vaisseaux se passe sur une planète, c'est le seul cas ou il y a deux entités à la même position. Si un mouvement d'un vaisseau arriverait à une position qui est déjà occupé, le mouvement n'est pas effectué. La simulation termine après cent tours ou dès qu'une espèce a exterminée tous les autres.