Rapport POO space war
Romain FERRAND
Thaddeus LEONARD
Gr 319k

Dans ce rapport nous allons présenter dans un premier temps les classes, leurs attributs, leurs utilités et les relations qui existent entre elles, puis nous allons parler des quelques problèmes d'ordre généraux et leurs solutions, nous parlerons des outils utilisés pour tester le code pendant le développement enfin nous terminerons par le manuel d'utilisation.

Présentation des Classes :

Classe Entités :

Au cours de ce projet nous avons dans un premier temps essayé de déterminer les relations d'héritage et de compositions,

comme première relation d'héritage nous avons la classe Entites
qui est la classe mère de vaisseaux et planètes.
Entité est une classe abstraite du fait que construire un objet entité sans que cela soit un vaisseau, ni une planète ne nous permet pas de savoir ce qu'est l'entité son type est donc bien abstrait

au final nous avons dégagé le fait que les deux informations globales que partagent les planètes et les vaisseaux sont leurs positions et l'espèce à laquelle ils appartiennent.

Il a donc une relation de composition entre espèces et entités.

La classe entités gère les entités gère donc les entités en jeu,
ses classes filles sont vaisseaux et planètes
le polymorphisme est utilisé pour la méthode agir qui effectue des opérations radicalement différentes pour les vaisseau et planètes.
Pour le reste la classe fournie une liste de getter permettant de connaître la position et la faction de l'objet en question, et parfois à but de test, son identifiant unique.

Classe vaisseaux :

Comme nous l'avons vu précédemment la classe vaisseaux hérite d'entités
les informations spécifiques de cette classe se situent dans les relations de composition avec Propulsion et Équipement ainsi que dans des attributs plus basique qui définissent les « points de vie » actuel et maximal.
Lorsqu'on crée un vaisseau on doit lui fournir son espèce et sa position,
le reste est défini aléatoirement que ce soit sa propulsion, son équipement mais aussi la résistance suivant une plage de valeur un maximum et un minimum défini par des constantes.

La méthode agir hérité d'Entités permet de définir les actions du vaisseau à savoir par tour et dans

cet ordre un déplacement suivant sa propulsion, puis une interaction suivant son équipement le déplacement est choisi de manière aléatoire suivant une liste de positions valide fourni par sa propulsion, de même l'Équipement permet d'interagir sur une cible valide choisie de manière aléatoire

Classe équipement :

la classe équipement défini les interactions du vaisseau avec le reste de la grille galactique autre que le déplacement actuellement seul l'équipement Polyvalent est implémenté dans un premier temps l'équipement regarde avec quelles cibles un vaisseau peut interagir, les vaisseaux alliés ne sont pas considérés comme des cibles.
Puis il en choisit une de manière aléatoire, et lui applique une action suivant son type :
– planète vide : colonisation
– planète ennemie : attaque et colonisation si la population est réduite à 0
– plante allié : recharge en carburant

Classe propulsion :

la classe propulsion défini la façon dont le vaisseau se déplace, et également la portée maximale et le carburant disponible à la création
la classe propulsion est une classe abstraite, puisque crée une propulsion sans savoir son type de déplacement ne permet pas le déplacement donc propulsion est bien un type abstrait.
trois types de propulsion sont donc implémentés en utilisant l'héritage : linaire, diagonale, omnidirectionnel.
on utilise ici le polymorphisme pour définir les positions valides que retourne la propulsion en fonction du type de celle-ci.

Linaire et Diagonal sont implémentés de manière semblable :
par rapport à la portée maximale du vaisseau : min (carburant, portée), on calcule les positions cartésiennes qui correspondent qu'on calcule en positions réelles sur la grille puis une fois toutes les positions valide calculé on regarde si ces positions sont libres et on retourne la liste de celles qui le sont effectivement
la propulsion omnidirectionnelle effectue les mêmes calcule et vérifications, sauf que les positions sont prises dans un carré centré sur le vaisseau dans un diamètre qui est égale à 2 fois la portée maximale

Classe Planètes :

La classe Planètes est une classe qui hérite d'Entités.
Les informations spécifiques à cette classe sont :
sa taille qui est calculée aléatoirement,
sa population qui est calculée en fonction de sa taille,
et un Vaisseau qui est le vaisseau en construction sur la planète
Lorsqu'on crée une planète on doit lui fournir son espèce et sa position comme toutes entités.
Lors de la création d'une planète (avec une espèce qui n'est pas l'espèce neutre),
on crée également deux vaisseaux (jusqu'à 8 en fonction de la constante associé) qui sont

positionnés autour de cette planète, la production d'un vaisseau est lancé après cela.

La méthode agir hérité d'entité définie tout ce que la planète peut faire pendant un tour c'est-à-dire l'augmentation de la population la production d'un vaisseau et son lancement si toutes les conditions pour lancer le vaisseau sont remplies, ces actions se font dans cet ordre pendant le « tour » de la Planète.

Pour la construction de vaisseau, un vaisseau attaché à la planète est créé, sa position est celle de la planète et on choisit une résistance aléatoirement dans une plage de valeur choisie par rapport aux constantes et une intégrité de 0.
à chaque tour on incrémente l'intégrité son intégrité jusqu'à ce qu'elle soit égale à sa résistance.
À ce moment-là, la planète essaye de lancer le vaisseau,
pour cela on utilise la méthode lancement qui trouve une position valide pour le vaisseau grâce à la méthode invalides si cette position valide existe,
c'est-à-dire s'il y au moins une case libre autour de la planète, le vaisseau est lancé à cette position, puis est ajouté a la flotte de l'espèce.
Sinon le vaisseau reste dans le hangar de la planète qui retentera de le lancer au prochain tour.

La méthode colonisation permet à un vaisseau de coloniser une planète,
celle-ci prend alors l'espèce du vaisseau comme nouvelle espèce et cette planète est ajoutée à l'empire de l'espèce.
le vaisseau colonisateur est détruit par la planète et une nouvelle production de vaisseau est lancé
Planètes contient aussi une méthode bombardement qui permet d'infliger des dégâts a la planète donc en enlevant de la population, si la population est inférieure a 1 alors le vaisseau qui bombarde colonise la planète.

Ces méthodes sont vu comme des actions subis par la planète et non comme des actions effectuées par le vaisseau, dans ce sens le code est plus court que si les méthodes avaient été conçus depuis Équipement

Classe Espèces :

La classe Espèces est une classe qui permet de gérer des espèces. Ces espèces ont pour information un taux de reproduction, un taux de production, une couleur ainsi qu'une flotte et qu'un empire qui sont des l'ArrayList de Planètes ou de Vaisseaux.

La classe Espèces est surtout utilisée comme stockage de toutes les informations liés aux espèces, la classe est donc composée essentiellement de méthode de getter et de setter qui permettent de manipuler la flotte et l'empire des espèces
La méthode score permet de déterminer le score d'une espèce en fonction de son empire et de sa flotte.

Le classement est un ajout au projet, comme la partie peut se terminer avec plus d'une espèce en jeu, nous avons implémenté une méthode qui retourne le score de l'espèce en question,
Cela permet en fin de partie de faire un classement suivant le score des espèces,
pour pouvoir classé plus facilement les espèces, nous avons implémenté l'interface comparable dans espèces, la méthode compareTo est surchargé de façon à ce que le trie puisse se faire suivant le score des espèces, ainsi les Espèces sont triées de manière croissante dans la liste et sont affichées du dernier élément jusqu'au premier.

Classe Galaxie :

Pour la classe Galaxie nous avons décidé de passer de faire passer le constructeur en privé et de créer une méthode statique qui renvoie soit une nouvelle galaxie si aucune n'a encore été créé ou alors l'instance de la galaxie courante.
Cela permet de créer une unique galaxie pour ne pas avoir besoin de passer en paramètre La galaxie dans les méthodes des autres classes et pour être sûr que tout ce passe sur une même galaxie, le code est donc grandement facilité même si d'ordre général cette pratique est a évité puisqu'elle est limitante dans les évolutions qui peuvent être faite.

La Galaxie est composée d'un arrayList d'Espèces qui lui permet grâce à des foreach d'avoir accès a toutes les entités encore en jeu.
La classe galaxie implémente des méthodes qui servent à la création de la galaxie : la méthode « genese » prend en argument le nombre d'espèce ainsi que le nombre de planètes vide dans la galaxie.
Pour chaque espèce la méthode choisie une couleur, un taux de production et de reproduction aléatoire suivant les bornes définies dans les constantes, puis crée une flotte de deux vaisseaux.
Les planètes vides prennent comme espèce, l'espèce neutre qui se distingue par sa couleur blanche.
creePos, retourne une position aléatoire libre.
estLibre : regarde si la position passé en paramètre est libre
calculPos : retourne la position passé en paramètre modulo largeur pour le x et modulo hauteur pour le y.
calculDelta : prends deux positions passé en paramètre et qui regarde a combien de case elle se situe l'une de l'autre. Mais toujours en prenant en compte que la grille est un tore.
La méthode jouer quand a elle sert permet de regarder si les espèces sont encore en vie si ce n'est pas le cas on enlève l'espèce de l'ArrayList d'espèces de galaxie.
Cette méthode appel aussi la méthode agir de toutes les entités, elle enlève également les vaisseaux qui sont détruits (qui ont une intégrité est inférieur ou égal a 0)

Classe Simulation :

Comme nous l'avons vu la plupart des actions sont effectué dans la galaxie qui est unique. Dans la classe simulation seul deux méthodes sont implémentés le test de condition de victoire et le classement de fin de partie.
La méthode victoire est statique elle permet de savoir si seulement une espèce reste en jeu, en prenant en condition la taille de la liste des espèces de l'objet galaxie, si la taille vaut 1 une espèce a gagné c'est victoire renvoie vrai, faux sinon.

Classe Affichage :

La classe d'affichage ne change pas beaucoup de celle fourni, elle se contente de récupérer la liste des espèces et de copier la liste des vaisseaux et des planètes de chaque espèce dans deux arraylist différents puis parcourir ces listes pour en afficher les éléments, avant chaque copie le contenu des arraylist de l'affichage sont vidés.

Difficultés rencontrées :

Tout au long du projet nous avons rencontré des difficultés qui sont d'un ordre général à celui-ci :

Une difficulté du projet est la gestion de la distance entre les entités, la forme de tore de la grille galactique, ne nous permet pas de faire un simple calcule de distance dans un plan, nous avons créé une méthode qui calcule la distance de manière générale entre 2 positions en prenant en compte la forme de la grille galactique, la formule de ce calcul à été dégagé de manière empirique, grâce à la méthode « papier et crayon », cette méthode fonctionne indépendamment de la parité de la grille galactique, cela est utile pour savoir avec quelles entités un vaisseau peut interagir,
ce problème aurait pu être réglé plus simplement en scannant les positions dans un carré centré sur le vaisseau mais la méthode choisie est plus générale, par exemple cela permet de savoir simplement les planètes les plus proches pour un vaisseau

une autre difficulté était au niveau de la suppression de redondance des informations au début nous avions une liste d'entités et une liste d'espèce, la liste d'entité (qui contenait toutes les entités en jeu) semblait être en trop, car elle était utile seulement en début de partie pour stocker les planètes vide (sans espèces), mais une fois les planètes vide coloniser toutes les informations étaient stockées dans les listes empire et flotte de chaque espèce, on a donc créé une espèce spéciale l'espèce par défaut qui a un taux pour la production et la reproduction de 0 et une couleur blanche.

Bien que cette solution facilite le code et supprime la redondance, elle n'est pas idéale, car on doit faire des conditions sur cette partie de notre code pour vérifier que l'espèce manipulée n'est pas l'espèce par défaut, ce qui pourrait entraîner des exceptions du type nullpointer,
de même des méthodes comme (posLibre (int[] pos) : boolean se voient alourdie, car elle doit vérifier les cases libres par rapport à la flotte et à l'empire que chaque espèce plutôt que d'itérée directement dans les listes des entités en jeu.

Nous avons du également changé la façon dont on supprime les éléments d'un arraylist lorsqu'on le parcourt pour ne pas avoir des ConcurentModificationException, pour cela nous avons utilisé l'itérateur des Collections.

Méthodes de test :

Dans cette partie nous parlerons des méthodes de tests utilisées :
la plupart des classes possèdent une méthode affiche info construit dans ce sens :
la méthode dans galaxie affiches les informations de chaque espèce,
une espèce affiche tous ces attributs et appelle la méthode afficheInfo pour chaque vaisseau de sa flotte et chaque planète de son empire
ces éléments affichent à leurs tours tout leurs identifiants et leurs attributs
ces méthodes permettent de savoir à chaque instant comment se déroule la partie à la fois de manière générale car toutes les espèces sont affichées puisque l'on a un affichage détaillé de chaque objet.

La classe Équipement quant à elle possède un affichage de test spécifique qui affiche la liste des cibles valides, ce qui a permis de vérifier le bon fonctionnement des algorithmes utilisés.

Manuel d'utilisation :

le paramétrage est très simple, car les caractéristiques des éléments du jeu sont choisis aléatoirement
– on fournit à « genèse » un nombre d'espèce en premier paramètre, et un nombre de planètes vide pour le second
– et on lance le jeu.
On peut modifier toutes les constantes de jeu comme le nombre de vaisseau de bases des planètes dans les constantes.