PROGRAMMATION OBJET

RAPPORT PROJET 2

Glenn PlOUHINEC
Mamadou DIALLO
GROUPE 303E

1-Introduction

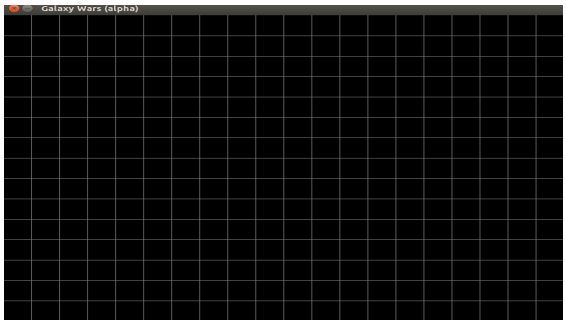
L'objectif de ce projet est de réaliser une simulation de conquête spatiale qui opposera différentes espèces de la galaxie chaque espèce étant constituées de plusieurs entitées « planete » ou « vaisseaux » , chaque espèce luttant pour sa survie défie les autres espèces dans des combat afin de les coloniser et d'accroître leur dominance .
Dans notre rapport nous parlerons dans un premier temps des différentes classes utilisés (objetctif de chaque classe , interface et structure de données).

2- Description des Classes

A - Galaxie

A.1 - Objectif

La galaxie est représentée comme une grille 2D de largeur×hauteur cases pouvant chacune être occupée par une entité, chaque case ne peut être occupée que par une seule entité. La grille de la galaxie est une planisphère, c'est-à-dire que les cotés opposés sont adjacents : une entité se déplaçant vers le haut de la grille se retrouvera en bas si elle atteint le bord de la grille et vis-versa, de même pour la gauche et la droite.



A.2 - Interface

getGalaxieLargeur() est un accesseur nous retournant la largeur de notre grille.

getGalaxieHauteur() est un accesseur nous retournant la hauteur de notre grille.

ajoutEntite(Entite ent) permet d'ajouter une entité sur la grille.

supprEntite(Entite ent) permet de supprimer une entité déjà existante sur la grille.

modifPosititionEntite(int xAnc,int yAnc,int xNouv,int yNouv) permet, après un déplacement de modifier la position d'une entité.

Entite getEntite(int id) cet accesseur nous retourne une entité, en connaissant son identifiant.

boolean occupee(int x, int y) cette méthode nous indique si une case de la grille est occupée par une entité ou non.

A.3 - Structure De Données

La grille est initialisée comme un tableau d'entités à deux dimensions, les entiers largeur et hauteur définissent la taille de ce tableau, ces entiers sont initialisées grâces aux constantes déclarées dans la classe Constantes.

B - Espece

B.1 – Objectif

Une espèce est caractérisée par un identifiant unique, un taux de natalité, un taux de productivité et d'une couleur unique. Chaque espèce dispose d'un empire qui recense toutes les possessions de cette même espèce.

B.2 – Interface

Color creerCouleur(int x) cette méthode affecte une couleur à une espèce en fonction de son identifiant passé en paramètre. Nous n'avons créé que 4 couleurs possible.

double getTauxNatalite() cet accesseur nous retourne le taux de natalité de l'espèce.

Color getCouleur() cet accesseur nous retourne la couleur de l'espèce.

Empire getEmpire() cet accesseur nous retourne le type Empire de l'espèce.

Double getTauxProductivite() cet accesseur nous retourne le taux de productivité de l'espèce.

B.3 - Structure De Données

L'entier idEspece caractérise le numéro d'identifiant permettant de reconnaître une espèce, il est unique et s'incrémente pour chaque nouvelle espèce créée. Le réel tauxNatalite est déterminé aléatoirement entre le taux de natalité minimum et le taux de natalité maximum (entre 5 et 10), puis multiplié par 0,01 pour avoir un résultat correspondant à un pourcentage. De même, le réel tauxProductivite est alors déterminé entre 1 et 5. L'empire représente globalement toutes les possessions de cette espèce (planètes et vaisseaux). La couleur coul sert à différencier visuellement les différentes espèces.

C – Empire

C.1 - Objectif

La classe Empire représente les «possessions» d'une espèce, ce qui englobe les planètes colonisées et tous les types de vaisseaux lui appartenant, elle a la même couleur que l'espèce qui lui est associée.

C.2 - Interface

void ajouterPlanete(Planetes p,Galaxie galaxie) cette méthode ajoute une planète dans la liste des planètes appartenant à l'empire. On y initialise également la création d'un nouveau vaisseau. Sert notamment lors de la colonisation d'une planète.

void supprPlanete(Planetes p) cette méthode retire une planète de la liste des planètes appartenant à l'empire.

void ajoutVaisseaux(Vaisseaux v,Galaxie galaxie) cette méthode ajoute un vaisseau dans la liste des vaisseaux appartenant à l'empire, et l'ajoute également sur la grille de jeu (galaxie).
void supprVaisseaux(Vaisseaux v,Galaxie galaxie) cette méthode retire un vaisseau de la liste des vaisseaux de l'empire, et le retire aussi de la galaxie.

void nouveauVaisseauxEnConstruction(Planetes p,Galaxie galaxie) cette méthode initialise la construction d'un vaisseau.

Color getEmpirColor() cet accesseur retourne la couleur de l'empire.

ArrayList<Planetes> getPlanetes() cet accesseur retourne la liste de toutes les planètes de l'empire.

void constructionVaisseaux(double t,Galaxie galaxie) cette méthode ajoute un vaisseau sur la grille une fois la construction de ce dernier terminée, et s'il est possible de le placer sur la grille.

void deplacementVaisseaux(ArrayList<Espece> esp,Galaxie galaxie) applique un déplacement à chaque vaisseau de l'empire.

void interaction(ArrayList<Espece>especes,Empire empire,PlanetesInoccupees pI,Galaxie galaxie) cette méthode permet aux vaisseaux de l'empire d'interagir avec les autres entités (attaque, colonisation, recharge de carburant)

void reproduction(double tauxNatalite) permet à toutes les planètes de l'empire de se reproduire en fonction du taux de natalité de l'espèce.

ArrayList<Vaisseaux> getVaisseaux() retourne laliste de tous les vaisseaux de l'empire.

C.3 - Structure De Données

L'empire possède une couleur de type Color une liste (ArrayList) des planètes qu'il possède ainsi qu'une liste des vaisseaux. L'empire est initialisé avec une planète et deux vaisseaux en début de jeu, qui sont ajoutés dans leur liste respective.

D - Entite

D.1 - Objectif

Entité est une classe abstraite qui permet de créer les entités «vaisseaux» et «planètes». Elle possède deux constructeurs avec des paramètres différents, l'un sert de constructeur pour planètes, et l'autre sert pour vaisseaux.

D.2 - Interface

void infligeDegat() est une méthode abstraite où les entités subiront des dégâts (baisse de l'intégrité ou de la population).

void modifCouleur(Color newColor) permet de modifier la couleur d'une entité. String getTypeEntite() retourne «vaisseaux» ou «planetes».

void caseAdjacente(Planetes p,Galaxie galaxie) permet de repérer une case inoccupée dans les 8 cases adjacentes, autour d'une planète.

Color getColorEntite() accesseur qui retourne la couleur d'une entité.

int getAbscisse() accesseur qui retourne l'abscisse d'une entité (position sur la grille/galaxie).

int getOrdonnee() accesseur qui retourne l'abscisse d'une entité.

int recadrerAbscisse(int x) recadre l'abscisse si une entité sort de la grille.

int recadrerOrdonnee(int y) recadre l'ordonnée si une entité sort de la grille.
void recadrerCoordonnees(int x,int y) recadre l'abscisse et l'ordonnée d'une entité qui est sorti de la grille : on la replace de l'autre côté de la planisphère.

int getNumeroEntite() retourne le numéro identifiant, calculé avec ses coordonnées.

double min(double a ,double b) détermine le minimum entre a et b.

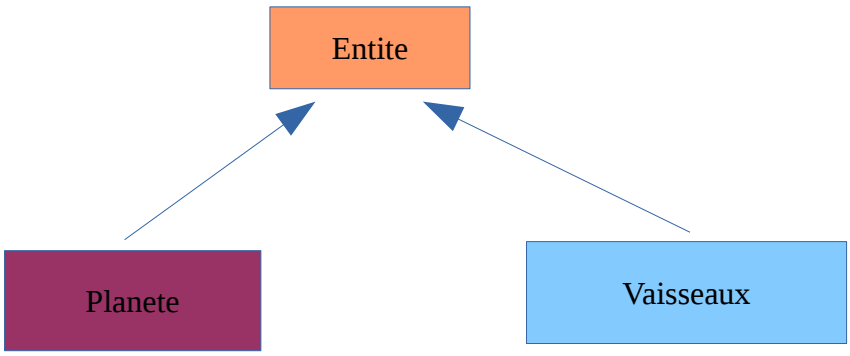
int retrouverAbs(int num) permet de retrouver l'abscisse d'une entité à partir de son numéro identifiant.

int retrouverOrd(int num) on retrouve l'ordonnée de la même manière.

D.3 - Structure De Données

Une entité possède une couleur permettant de déterminer son espèce (ou planète inoccupée), ainsi qu'une chaîne de caractères précisant son type (planète ou vaisseaux), et des coordonnées définies par une abscisse et une ordonnée.

D.4 Héritage



E -Planetes

E.1 - Objectif

La classe Planètes hérite de Entité et implémente ses méthodes abstraites. Il s'agit d'une entité qui possède une taille, possède une population, et qui peut créer des vaisseaux pour son empire.

E.2 - Interface

void infligeDegat() cette méthode implémente celle de Entité, la planète subit des dégâts, un entier entre 5 et 10 est choisi aléatoirement, puis est retranché à sa population.

int getTaillePlante() retourne un entier correspondant à la taille de la planète.

boolean constructionTerminee(double t) si l'intégrité du vaisseau est égal à sa résistance, on retourne un booléen indiquant que la construction est terminée.

void vaisseauxEnConstruction(Planetes p,Galaxie galaxie) initialisation de la construction d'un vaisseau.

Vaisseaux vaisseauxConstruit() retourne le nouveau vaisseau venant d'être créé.
void reproductionPlanete(double tauxNatalite) augmentation de la population en fonction du taux de natalité de l'espèce.

boolean populationNull() indique si une planète est inhabitée.

void modifPopulation(double nvPop) affecte une nouvelle population à la planète.

E.3 - Structure De Données

La taille d'une planète est un entier déterminé aléatoirement entre les constantes PlaneteTailleMax et PlaneteTailleMin, et sa population initiale vaut la moitié de sa taille.

F – Vaisseaux

F.1 - Objectif

Le classe Vaisseaux hérite de Entité et implémente ses méthodes abstraites. Un vaisseau est une entité se déplaçant sur la grille de jeu (galaxie) et pouvant interagir avec les autres entités. Un vaisseau possède une résistance déterminée aléatoirement lors de la construction, une intégrité (que nous comprenons comme des «points de vie»), une propulsion et une liste d'équipement, il est également caractérisé par son type de déplacement (linéaire, en diagonale ou omnidirectionnel).

La liste des équipements d'un vaisseau correspond aux 4 caractères imposés par le sujet: 's', 't', 'c', 'p'. Le caractère 's' permet à un vaisseau d'attaquer un vaisseau ennemi, le caractère 't' permet à un vaisseau d'attaquer une planète ennemi, le caractère 'c' permet à un vaisseau de coloniser une planète inoccupée, et le caractère 'p' permet à un vaisseau d'être polyvalent et faire toutes les interactions qu'il souhaite.

F.2 - Interface

void creerEquipements() permet de déterminer les équipements qu'un vaisseau pourra utiliser en fonction de la taille de la liste d'équipement qui lui aura été affecté, taille qui a été affecté aléatoirement entre 1 et 4. Il peut alors utiliser les équipements 's', 't', 'c', 'p'.

void avancerEquipement() on change l'équipement utilisé à chaque tour de jeu.

String equipementCourant() retourne le caractère correspondant à l'équipement actuellement utilisé.

void setIntegrite(int nouvelleIntegrite) ce mutateur permet d'affecter une nouvelle intégrité au vaisseau.

double getIntegrite() retourne l'intégrité du vaisseau.

void nouvelleIntegrite(double tauxProductivite , double pop) modifie l'intégrité du vaisseau lors de la construction du vaisseau par une planète.

char typeDeplacement() affecte aléatoirement un type de déplacement au vaisseau : '+' = linéaire, 'x' = diagonale, '*' = omnidirectionnel.

char getTypeDeplacement() retourne le type de déplacement.

int getResistance() retourne la résistance du vaisseau.

Propulsion getPropulsion() retourne la Propulsion du vaisseau.

boolean verifCarburant() retourne true s'il reste du carburant au vaisseau.

void sensHaut(Galaxie galaxie) permet à un vaisseau de se déplacer vers le haut de la grille.

void sensBas(Galaxie galaxie) un vaisseau de déplace vers le bas.

void sensGauche(Galaxie galaxie) un vaisseau se déplace vers la gauche.

void sensDroit(Galaxie galaxie) un vaisseau se déplace vers la droite.

void sensDiagonaleHD(Galaxie galaxie) un vaisseau se déplace en diagonale en haut à droite.

void sensDiagonaleBD(Galaxie galaxie) un vaisseau se déplace en diagonale en bas à droite.

sensDiagonaleBG(Galaxie galaxie) un vaisseau se déplace en diagonale en bas à gauche.

void sensDiagonaleHG(Galaxie galaxie) un vaisseau se déplace en diagonale en haut à gauche.

void lineaire(Galaxie galaxie) détermine aléatoirement la direction linéaire que va prendre le vaisseau.

void diagonale(Galaxie galaxie) détermine aléatoirement la direction diagonale que va prendre le vaisseau.

void omnidirectionnelle(Galaxie galaxie) détermine aléatoirement la direction omnidirectionnelle que va prendre le vaisseau.

void deplace(Galaxie galaxie) appelle l'une des trois méthodes précédentes en fonction du type de déplacement qui a été affecté au vaisseau ('+' ou 'x' ou '*').

void deplacement(ArrayList<Espece> esp,Galaxie galaxie) cette méthode vérifie si un vaisseau peut se déplacer pour le prochain tour de jeu, puis fait appel à la méthode autoDestruction(esp, galaxie)

int caseCible() on choisit aléatoirement une case adjacente au vaisseau.

void infligeDegat() implémentation de la méthode abstraite dans Entité, un vaisseau qui subit des dégâts se voit retrancher son intégrité d'un nombre entre 1 et 3.

void autoDestruction(ArrayList<Espece> esp,Galaxie galaxie) cette méthode permet de détruire un vaisseau si son intégrité ou son carburant atteint 0 ou moins.

void interactionVaisseaux(ArrayList<Espece> esp,Empire e , PlanetesInoccupees pI,Galaxie galaxie) cette méthode permet au vaisseau d'interagir avec les autres entités.

F.3 - Structure De Données

Lors de sa construction, un vaisseau possède une résistance déterminée aléatoirement entre les constantes VaisseauResistanceMax et VaisseauResistanceMin, soit entre 1 et 10. L'intégrité est déterminée aléatoirement entre 0 et la résistance précédemment affectée. La propulsion affectée dans le constructeur est un nouvel objet de type Propulsion. Le type de déplacement du vaisseau est déterminée grâce à la méthode typeDeplacement(). L'entier nb sert à définir le nombre d'équipements que le vaisseau aura à sa disposition (entre 1 et 4). Et la listeEquipement est un tableau de String qui définit la liste des équipements qu'un vaisseau aura à sa disposition.

G - Propulsion

G.1 - Objectif

Une propulsion est caractérisée par une portée (e.g., de 1 à 5 cases) et une quantité de carburant (e.g., de 10 à 20). Elle permet au vaisseau de se déplacer sur la grille galactique.

G.2 - Interface

int getPortee() est un accesseur qui retourne la portée affectée à la propulsion.

int getCarburant() cet accesseur retourne le carburant affecté à la propulsion.

int positionAtteinte() cette méthode nous retourne un entier entre 1 et la portée affectée précédemment pour qu'un vaisseau se déplace de tmp cases.

void diminuerCarburant() est un mutateur qui diminue le carburant du vaisseau de 1.

void rechargerCarburant() ce mutateur permet de recharger le carburant en l'augmentant de 5.

G.3 - Structure De Données

La portée de la classe propulsion est déterminée aléatoirement entre les constantes PorteeMax et PorteeMin, soit entre 1 et 5. De même le carburant est déterminé entre CarburantMax et CarburantMin, soit entre 10 et 20.

H – PlanetesInoccupees

H.1 - Objectif

Une planète inoccupée représente les planètes blanches pouvant être colonisées par les vaisseaux, elles ne possèdent, par définition, pas de population. Nous les avons définies comme des des éléments de type Planetes particuliers.

H.2 - Interface

ArrayList<Planetes> getPlanetesInoccupees() cette méthode retourne la liste des planètes inoccupées.

void ajoutPlaneteInoccupee(Planetes p) ajoute une planète à la liste des planètes inoccupées.

void supprPlaneteInoccupee(Planetes p) supprime une planète de la liste des planètes inoccupées.

H.3 - Structure De Données

Le constructeur de PlanetesInoccupees initialise une liste de Planetes et leur affecte à chacune la couleur blanche, une population fixée à 0 grâce à la méthode modifPopulation() de Planetes, l'ajoute à la liste des planetesInoccupees et l'ajoute sur la grille de jeu.

I - Constantes

I.1 - Objectif

Il s'agit de la classe référençant toutes les constantes réutilisées tout au long du projet.

I.2 - Interface

Cette classe ne possède aucune méthode.

I.3 - Structure De Données

Par rapport au fichier de base fourni, nous avons ajouté les constantes TauxNataliteMin, TauxNataliteMax, TauxProductiviteMin, TauxProductiviteMax, NombrePlaneteIni, NombreVaisseauxIni, NbPlanetesInocupee, PorteeMin, PorteeMax, CarburantMin, CarburantMax.

J - Affichage

J.1 - Objectif

Cette classe nous a été fournis , elle permet l'affichage graphique de la galaxie .

J.2 - Interface

Nous n'avons eu à rajouter aucune autres méthodes que celles déjà présente. Par ailleur nous avons modifié la signature de la méthode raffraichir et modifier les paramètres des planètes et vaisseaux dans la méthode paintComponent.

J.3 - Structure De Données

Nous n'avons rien modifier.

K - Fenetre

Cette classe n'a pas été touchée.

L - Simulation

L.1 - Objectif

Classe gérant la simulation de conquête galactique , elle contient une fonction principale « main » qui permet de faire des jeux de testes .

L.2 - Interface

Boolean victoire() : cette une méthode static , elle rétourne vrai si le jeu est terminé.
void couleurMot(Color c) : cette méthode est aussi static , elle determine le nom de la couleur qu'elle prend en paramètre en francais.
void afficheVainqueur() : cette méthode affiche le gagnant une fois la partie terminée.

L.3 - Structure De Données

Cette classe n'a pas de constructeur.

3-Conclusion

Ce second projet, Galaxy Wars, fut très intéressant à concevoir et à programmer en Java. Il nous a permis d'améliorer notre affinité avec ce langage, et la programmation objet en général, il nous a permis de comprendre un peu mieux les notions d'héritage abordées en cours, même si nous n'utilisons pas d'interface dans le cadre de ce projet.
Le principe même du jeu Galaxy Wars fut intéressant à analyser même s'il était souvent compliqué de programmer ce que nous voulions faire, étantdonné les très nombreuses informations à prendre en compte. Il était aussi intéressant d'observer les résultats sur une réelle sortie graphique, vu que nous sommes habitués à travailler avec la console. Au niveau des améliorations possibles du projet, il aurait été intéressant d'implémenter des classes Equipements, Construction, et Deplacement, cela aurait allégé le code de la classe Vaisseaux notamment, et on aurait eu une utilisation plus «logique» de la programmation objet.