université PARIS DIDEROT PARIS 7

INITIATION À LA PROGRAMMATION 2 (IP2)

TP1 bis : Révisions du premier semestre et quelques objets

Ce TP n'est pas à rendre sauf demande contraire de votre enseignant qui pourrait vous demander d'en rendre une partie à la place d'une autre partie du TP1.

1 Révisions

Faites le QCM interactif qui est sur moodle. Si vous n'avez pas réussi à avoir 100% des points, revoyez vos cours d'IP1 et/ou demandez de l'aide à votre enseignant. Puis, refaites ce test plus tard.

Ce test n'est pas noté, il ne sert que pour votre auto-contrôle.

Bout qui manque Pour l'instant il n'y a que 2 questions, je vais faire en sorte qu'il y en ait entre 7 et 10.

2 Utiliser des objets pour faciliter la programmation

2.1 Fourmi de Langton

cf. `https://fr.wikipedia.org/wiki/Fourmi_de_Langton`

Présentation La fourmi de Langton se déplace sur une grille bicolore, noire et blanche. Elle peut modifier la couleur d'une case de la grille lors de son déplacement et elle se déplace en fonction des couleurs qu'elle rencontre.

Les couleurs seront modélisées par un booléen (false pour noir, true pour blanc). Au début, la grille sera entièrement noire. De plus, on supposera que la ligne 0 est en haut, la ligne 1 en-dessous, etc. et la colonne 0 est à gauche, la colonne 1 à sa droite, etc.

La fourmi a le comportement suivant. Si elle se trouve sur une case noire, elle tourne de 90 degrés vers la gauche, elle change la case où elle se trouve en blanc et avance à la case suivante. Si elle se trouve sur une case blanche, elle tourne de 90 degrés vers la droite, elle change la case où elle se trouve en noir et avance à la case suivante. L'orientation de la fourmi sera donnée par une valeur entière comprise entre 0 et 3 :

- 0 pour dire que la fourmi est tournée vers le nord (haut)
- 1 pour dire que la fourmi est tournée vers l'est (droite)
- 2 pour dire que la fourmi est tournée vers le sud (bas)
- 3 pour dire que la fourmi est tournée vers l'ouest (gauche)

Ainsi si l'orientation de la fourmi est 0 (vers le haut) et qu'elle se trouve sur une case noire, son orientation deviendra 3, en revanche si elle se trouve sur une case blanche son

orientation deviendra 1. Si son orientation est 1 (vers la droite) et qu'elle se trouve sur une case noire, son orientation deviendra 0.
Si la fourmi sort de la grille, il ne se passe plus rien.
Pour modéliser, ce "jeu", on utilisera deux classes en plus de celle contenant le `main`, qu'on appelera `Main`. [1]

2.1.1 La fourmi

On déclarera une classe Fourmi ayant comme attributs
- deux entiers `x` et `y` pour donner sa position ;
- un entier `orientation` pour dire vers où elle regarde ;
- une variable *gril* de type `Grille` qui sera un lien vers la grille où évolue la fourmi dont nous parlerons un peu plus tard. La grille ne sera pas au courant de la position de la fourmi.

1. Écrivez cette classe avec un constructeur qui prend comme argument la valeur de ces 4 attributs.
2. Écrivez deux méthodes `int getX()` et `int getY()`.
3. Écrivez une méthode `afficheToi()` qui écrira les différentes caractéristique de la fourmi et permettront de tester les méthodes au fur et à mesure.
4. Écrivez une méthode `void tourne(boolean aGauche)`, où `aGauche` vaut true si on veut tourner à gauche, false si on veut tourner à droite.
5. Écrivez une méthode `void unPas()`, qui selon l'orientation avance d'une case, i.e. change les coordonnées de la fourmi, la grille n'est pas impactée.

Pour faire le reste des méthodes nécessaires dans cette classe, nous avons besoin de la grille.

2.1.2 La grille

On déclarera donc une classe Grille ayant comme attributs
- deux entiers `largeur` et `longueur` pour donner les dimensions de ladite grille ;
- un tableau de booléens qui représentera les couleurs de chaque case.

1. Écrivez cette classe avec un constructeur qui prend comme argument la valeur de la largeur et de la longueur et créera le tableau de booléens initialisés à false (initialisation par défaut).
2. Écrivez une méthode `void dessine()` qui dessinera la grille : une case blanche sera dessinée à l'aide d'un espace, une case noire avec, par exemple, le caractère `#`.
3. Écrivez une méthode `void dessineAvecFourmi(Fourmi f)` qui dessinera la grille avec la position de la fourmi : on pourra la dessinez avec un `0`.
4. Écrivez une méthode `boolean getCouleur(int x, int y)` qui retourne la couleur à la case (x, y).

1. Une vraie modélisation objet comporterait plus de classes, mais pour l'instant nous nous contenterons de celles-là.

5. Écrivez une méthode `void changeCouleur(int x, int y)` qui change la couleur à la case (x, y) : noir devient blanc, blanc devient noir.
6. Écrivez une méthode `boolean estHorsGrille(int x, int y)` qui indique si la fourmi est sortie de la grille, i.e. si elle n'y est plus.

2.1.3 Finir le programme

Nous allons déjà compléter la classe `Fourmi` avec la méthode suivante :

1. Écrivez une méthode `boolean unMouvement()` qui va faire les choses suivantes :
   — on cherche la couleur de la case sur laquelle est la fourmi (appel à la méthode `boolean getCouleur(int x, int y)` de `Grille`.
   — Suivant la couleur en question, on tourne à droite ou à gauche.
   — On change la couleur de la case avant de la quitter.
   — on fait un pas.
   — si ce pas nous fait sortir de la grille, on retourne false, sinon true.
2. Ensuite, dans le main de la classe `Main`, on créera une grille (par exemple, de dimension 20, 20), et une fourmi, (on peut la mettre au milieu, orientée comme il vous plaît), on testera d'abord quelques mouvements en dessinant la grille à chaque mouvement. Puis, on fera une boucle de mouvements, dessins en faisant attention à tester la valeur de retour de `unMouvement()` pour éviter les sorties de tableau pas très agréables.
3. Convertissez cette boucle de test en une méthode (`public static`) qui prendra en argument, la taille de la grille, la position initiale de la fourmi, son orientation et le nombre de mouvements.
4. Testez aussi sur une petite grille pour justement vérifier que tout se passe bien quand on quitte sort de la grille.

2.1.4 Variante facile

Grâce à notre modélisation avec des objets, nous allons pouvoir facilement faire tourner le jeu avec plusieurs fourmis qui bougeront tour à tour (on considèrera que deux ou plusieurs fourmis peuvent être simultanément sur la même case). Faites une nouvelle méthode comme celle de la section précédente qui prendra en argument le nombre de fourmis désirées et la taille de la grille et placera les fourmis de manière aléatoire, leur orientation sera aussi choisie aléatoirement. Pour ne pas compliquer, on considèrera que

— dès qu'une fourmi sort de la grille, on arrête le jeu.
— on utilisera la méthode de dessin sans fourmi.

2.2 S'il vous reste du temps : La conjecture de Syracuse

Soit un entier (strictement) positif n, on calcule la suite u_i comme suit :

— $u_0 = n$

— si $i \geq 0$, $u_{i+1} = \begin{cases} u_i/2 & \text{si } u_i \text{ est pair} \\ 3u_i + 1 & \text{si } u_i \text{ est impair} \end{cases}$

Par exemple, si $u_0 = 10$, on a $u_1 = 5 (= 10/2)$, $u_2 = 16 (= 3 \times 5 = 1)$, $u_3 = 8$, $u_4 = 4$, $u_5 = 2$, $u_6 = 1$, $u_7 = 4$, $u_8 = 2$, $u_9 = 1$, ...

La conjecture de Syracuse (connue aussi sous divers autres noms, cf. `https://fr.wikipedia.org/wiki/Conjecture_de_Syracuse`) que la suite précedente a toujours un indice p tel que $u_p = 1$, et ceci quelque soit le n initial.

Cette conjecture est démontrée jusque des valeurs de n très grandes, mais le problème de savoir si elle valable pour tous les n est encore ouvert.

Pour limiter le temps de calcul, on va décider de calculer une suite en donnant une limite maximale à la valeur des u_i.

Ce que vous devez faire dans un premier temps

1. Créer une classe `Syracuse` qui contiendra un `main` pour tester.
2. Dans la classe `Syracuse`, écrivez une méthode `public static int suivant(int u)` qui calcule l'élément u_{i+1} si on lui fournit l'élément u_i de la suite.
3. Écrivez ensuite une méthode `public static void afficheSuite(int n, int lim)`, où n est l'élement 0 de la suite qu'on veut calculer et lim, la limite maximale des u_i. La méthode affiche les différents éléments de la suite et s'arrête soit quand la limite maximale est atteinte, soit, quand on atteint 1.
4. tester.

Tester plusieurs valeurs. Rendre un résultat complexe On voudrait tester le comportement de la suite pour des valeurs initiales (premiers éléments de la suite) de 1 à p pour une certaine valeur p. Utiliser la méthode `afficheSuite` précédente n'est pas très indiquée.

Ce que nous voulons c'est avoir une méthode qui, pour une valeur initiale donnée, donne un résultat permettant de savoir, si on a atteint 1 ou si on a arrêté le calcul à cause d'une valeur trop haute. (On suppose que le calcul ne peut pas boucler sans atteindre 1, ce qui n'est pas démontré, mais est vrai pour des valeurs raisonnables.)

Pour ce faire, on va déclarer une classe dont voici le début.

```
public class Resultat{
    bool limite; //est-ce que la limite a été dépassée
    int indice; //si limite non atteinte, pemier indice pour lequel u_i = 0
```

Ajoutez à cette classe deux constructeurs et une méthode comme suit :

— `Resultat(int indice)` qui crée un résultat donnant l'indice pour lequel on atteint 1.

— `public Resultat()` qui crée un résultat indiquant que la limite a étée dépassée.

— `public afficheToi()` qui affiche de manière intelligible pour un être humain le résultat.

Ensuite dans la classe `Syracuse`, vous pouvez faire une méthode qui prend en argument une limite et l'entier p jusqu'auquel on veut faire les calculs et affiche les résultats.