AOO

Turtle trip

1 Objectifs

L'objectif de ce travail est de compléter l'application Turtle. Dans sa première version la tortue est capable d'avancer, de reculer, de tourner à droite ou à gauche. On souhaite faire évoluer cette spécification pour l'adapter ou l'enrichir. Ce travail doit permettre de maîtriser la modification de classes existantes, la compréhension et la mise en place d'assertion caractérisant les contrats entre objets. Il doit aussi permettre de réaliser des algorithmes simples.

1.0 Objectif 0

Utilisez l'application Turtle pour comprendre son fonctionnement. Cette application modélise la tortue LOGO capable d'effectuer des déplacements sur une surface plane en pouvant effectuer des tracés. Cette application est constituée d'un espace visible de l'espace de dessin où évolue la tortue et d'un panneau de contrôle permettant d'opérer sur la tortue. L'espace peut recueillir des figures qui seront affichées si elles font partie de l'espace de visibilité. Celles-ci sont réparties en trois catégories : les figures présentent de façon permanente et qui ne sont pas effacées par la commande « Clear », les figures standards qui sont effacées par la commande « Clear » et les figures temporaires qui ne restent visibles que le temps de l'interaction les concernant. Les Sliders permettent de fixer les valeurs des paramètres des commandes qui en ont besoin.

Dans un premier temps constituez un projet dans l'IDE Eclipse. Créez un projet de nom **jus.aoo.turtle** et importez les classes fournies dans l'archive Source dans celui-ci ainsi que le fichier turtle.gif. Ajouter à la librairie les archives .jar fournies. Si l'ensemble de ces opérations ont été réalisées correctement, le projet est opérationnel et vous pouvez exécuter le programme en demandant l'exécution de « java application » sur la classe TurtleTrip.

- Utilisez les différentes commandes à votre disposition en vérifiant le mode traçant ou non et en variant les valeurs des paramètres,
- Vérifiez le fait que la tortue peut sortir de l'espace de visibilité et y revenir.

L'espace de visibilité est représenté sur l'écran par une zone de dessin dont l'origine est le coin haut gauche de celle-ci, l'axe des ordonnées étant orienté vers le bas et celui des abscisses vers la droite. L'espace de la tortue est lui caractérisé par un repère plus traditionnel avec l'origine centré dans la zone de dessin et l'axe des ordonnées orienté vers le haut.

1.1 Objectif 1

Actuellement l'espace de déplacement de la tortue est infini, mais seule une fenêtre restreinte de cet espace est visible. La tortue peut donc sortir de l'espace visible puis y rentrer à nouveau. Pour améliorer cette situation, nous disposons de plusieurs solutions :

1. la première consiste à rendre visible l'ensemble de l'espace en permettant de déplacer la fenêtre de visibilité sur l'espace,
2. une seconde consiste à considérer que l'espace est borné et que toute tentative de sortie de cet espace est impossible en bloquant la tortue à la frontière de l'espace de visibilité,
3. la troisième est de considérer que toute tentative de rejoindre un lieu en dehors de la fenêtre de visibilité est interdite et se solde par un échec.

1.1.1 Question 1

On dispose, dans la classe DrawingSpace, des méthodes getWidth et getHeight pour accéder à la largeur (resp. la hauteur) de l'espace de visibilité.

1. En quoi les solutions 2 et 3 se différencient-elles ?
2. Peut-on envisager une autre solution, si oui laquelle ?
3. Quel invariant peut-on définir dans la classe Turtle pour les solutions 2 et 3.

1.1.2 Question 2

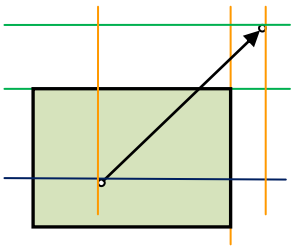
1. Faites le nécessaire pour définir cet invariant dans la classe Turtle.
2. Modifier le programme pour que cet invariant soit vérifié à l'exécution.

1.1.3 Question 3

- Réalisez la troisième solution proposée.
    1. Dans cette version qui est responsable d'un positionnement erroné ?
    2. Quel effet la solution 3 a-t-elle sur la chaîne de responsabilité et pourquoi ?
    3. Quelle différence il y a-t-il à engendrer une instance de la classe Exception ou une instance de la classe Require (jus.util.assertion.Require) ?
    4. Pourquoi doit-on choisir, dans l'ensemble des classes de violation d'assertion, la classe Require ?
    5. Quelle conséquence la mise en place de cette solution peut-elle avoir sur les vérifications dynamiques ?

  Pour afficher un message dans une popup-window vous pouvez utiliser le code suivant : `JOptionPane.showMessageDialog(null,"le texte à afficher");.`

1.1.4 Question 4



- Réalisez la seconde solution proposée. Le schéma ci-contre donne quelques indications sur façon de procéder pour calculer le vecteur réel de déplacement possible qui est le vecteur maximal sans sortir de l'espace de visibilité.
    1. Donnez la signature fonctionnelle (profil, prototype, …) de la méthode calculVecteurReel du calcul du vecteur réel de déplacement.
    2. Dans quelle classe cette méthode doit-elle être définie ? et pourquoi ?
    3. Fort de cette nouvelle méthode, modifier les méthodes concernées par la mise en place de cette nouvelle solution.
    4. Réalisez la méthode calculVecteurReel.

1.2 Objectif 2

On souhaite désormais que la tortue dispose d'un repère lui permettant de se situer dans l'espace. Elle sera dotée de 2 fonctionnalités supplémentaires qui lui permettront respectivement de se placer à un point précis de l'espace (allerA) et de s'orienter dans une direction précise (tournerVers). Dans les mêmes conditions que l'objectif 1, ajoutez ces 2 fonctionnalités à la tortue. Il vous faut ajouter 2 boutons supplémentaires dans l'interface pour les rendre accessible à l'utilisateur. Cette interface a été construite avec un builder d'interface, en l'occurrence celui d'Eclipse. Vous pouvez soir rééditer la classe TurtleTrip avec cet éditeur, soit ajoutez le code nécessaire par similitude au code produit pour les autres boutons (avancer par exemple).

1.2.1 Question 1

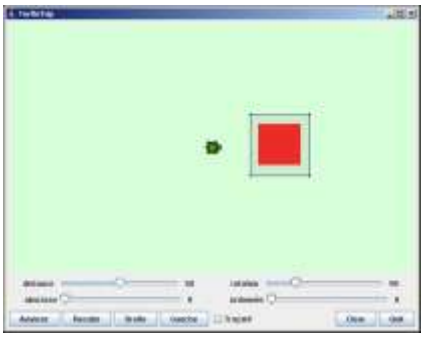
1. Modifiez les classes Turtle et TurtleTrip en conséquence. Vous prendrez soin de spécifier avez pertinence ces nouveaux services.

1.2.2 Question 2

On souhaite faire tourner en « bourique » la tortue en lui faisant accomplir un cercle caractérisé par un nombre de segments et une longueur de ces segments (quadrature du cercle). On considèrera que la tortue a une forte tendance l'entrainant à systématiquement aller vers la droite. Soit t la tortue, on pose la post-condition suivante :

@ensure intacte : old(t.position()).equals(t.position()) && old(t.cap()).equals(t.cap())

1. Quelles sont les différentes solutions pour répondre à ce problème ?
2. Réaliser une solution qui n'affecte pas la classe Turtle.



1.3 Objectif 3

On souhaite positionner dans l'espace de déplacement des obstacles qui pourront altérer les déplacements de la tortue (on part de l'objectif précédent). Pour cela on vous propose un canevas de la classe Obstacle qui représente un rectangle (cadre) dans l'espace de visibilité. Lorsque la tortue rencontre un tel obstacle elle bute sur celui-ci, ce qui interrompt sa trajectoire. La tortue est représentée par un cercle de 16 pixels de rayon[1]. *On admettra dans un premier temps que la construction d'un obstacle est correcte : il n'est pas positionné sur la tortue, etc*. L'espace inaccessible à la plume de la tortue est constitué du cadre de l'obstacle élargi par une bande dont la largeur est égale au rayon du cercle circonscrit à l'image de celle-ci. On repartira de l'objectif 1.

1.3.1 Question 1

Compléter les classes à votre disposition (Turtle, Obstacle) pour que l'on puisse mettre en place plusieurs obstacles dans l'espace de visibilité et que le comportement de la tortue corresponde à celui décrit ci-dessus. Pour cela vous utiliserez un tableau d'obstacles, *on admettra qu'il ne peut y avoir plus de 10 obstacles simultanément dans l'espace de visibilité*. On dispose d'une interaction utilisateur permettant de positionner un obstacle dans l'espace de visibilité en cliquant dans celui-ci puis en fixant la diagonale du rectangle souhaité.

1. Que représente l'ensemble des obstacles pour la classe Obstacle ?
2. En conséquence où et comment représenter cet ensemble, proposez plusieurs solutions ?
3. Choisissez la solution la plus pertinente dans le cas présent.

Le nombre d'obstacles étant limité dans cette version, il faut se prémunir contre les tentatives de créer plus d'obstacles qu'autorisé.

4. Quelle stratégie allez-vous mettre en place pour garantir ceci ?

Il existe une méthode, dans la classe DrawingSpace qui gère l'affichage, une fonctionnalité permettant d'éliminer les affichages temporaires (clearTemporaire). Attention cette question peut amener à devoir comprendre certains codes fournis.

1.3.2 Question 2

La trajectoire initialement prévue pour la tortue peut être interrompue prématurément lorsqu'elle rencontre un obstacle. Une solution pour résoudre ce problème consiste à calculer le vecteur maximal que l'on peut parcourir sur la trajectoire initiale sans rencontrer d'obstacle.

1. L'ordre d'énumération des obstacles est-il important et pourquoi ?
2. L'ordre d'énumération des segments d'un obstacle est-il important et pourquoi ?

Réalisez les modifications pour mettre en œuvre cette nouvelle spécification du système (voir §2), vous réaliserez l'ensemble du code nécessaire (pas d'utilisation de librairie). Vous disposez en annexe du système d'équations à résoudre pour déterminer le point d'intersection de 2 segments.

1.3.3 Question 3

Reprenez le problème de la question 1 en essayant d'assimiler la clôture de l'espace de déplacement à un problème d'obstacle.

[1] Si on veut tenir compte du fait que la position de la tortue est son centre, il faut translater la coordonnée concernée de la taille du rayon.

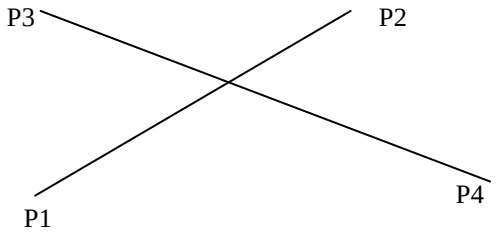
2 Annexes :

2.1 Intersection point of two lines (2 dimensions)

Written by Paul Bourke
April 1989

This note describes the technique and algorithm for determining the intersection point of two lines (or line segments) in 2 dimensions.

The equations of the lines are



Considering $\mathbf{P_a}$ a point of [$\mathbf{P_1}$ .. $\mathbf{P_2}$] and $\mathbf{P_b}$ a point of [$\mathbf{P_3}$ .. $\mathbf{P_4}$]

$$\mathbf{P_a = P_1 + u_a (P_2 - P_1)}$$
$$\mathbf{P_b = P_3 + u_b (P_4 - P_3)}$$

Solving for the point where $\mathbf{P_a} = \mathbf{P_b}$ gives the following two equations in two unknowns (u_a and u_b)

$$x1 + u_a (x2 - x1) = x3 + u_b (x4 - x3)$$
$$y1 + u_a (y2 - y1) = y3 + u_b (y4 - y3)$$

Solving gives the following expressions for u_a and u_b

$$u_a = \frac{(x4 - x3)(y1 - y3) - (y4 - y3)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}$$
$$u_b = \frac{(x2 - x1)(y1 - y3) - (y2 - y1)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}$$

Substituting either of these into the corresponding equation for the line gives the intersection point. For example the intersection point (x,y) where (x1,y1) is P1 and (x2,y2) is P2 :

$$x = x1 + u_a (x2 - x1)$$

$$y = y1 + u_a (y2 - y1)$$

Notes:

- The denominators for the equations for u_a and u_b are the same.
- If the denominator for u_a and u_b is 0 then the two lines are parallel.
- If the denominator and numerator for u_a and u_b are 0 then the two lines are coincident.
- These equations apply to lines; if the intersection of line segments is required then it is only necessary to test if u_a and u_b lie between 0 and 1. Whichever one lies within that range then the corresponding line segment contains the intersection point. If both lie within the range of 0 to 1 then the intersection point is within both line segments.