PROG1 - projet de programmation 2 - 27 Novembre 2020
Date limite de rendu : vendredi 29 Novembre à 19h (dernier *push*).

L'objectif de ce projet est de concevoir un environnement de simulation, d'animation et de débogage pour le code neuronal des fourmis vus lors du 1er projet.

Le projet doit être programmé en Java. Prenez soin de bien présenter votre programme (types, noms de variables, tests élémentaires, commentaires) et d'utiliser un style orienté objet le plus élégant possible (privilégiez en particulier les appels virtuels et éviter l'utilisation des cast et des tests `instanceof`).

Le projet est séparé en trois parties. La dernière est la plus *créative*. Nous vous demandons d'accompagner votre projet d'un fichier `Readme.md` expliquant brièvement les choix d'implémentations suivis et les fonctionalités de votre interface graphique.

1 Simulation

Cette partie ne nécessite pas de réaliser une interface graphique. Vous devez créer une classe `ant.Simulator` qui prend en argument deux fichiers de code fourmis et un fichier de carte, puis affiche le résultat de la compétition. Cette partie devra pouvoir être testée avec une ligne de command de la forme `java ant.Simulator team1.brain team2.brain large.world`.

2 Animation

Cette partie nécessite d'afficher dans une interface graphique le duel entre deux fourmilières. Vous avez toute liberté sur les interactions proposés à l'utilisateur, mais votre classe `ant.Animation` doit à minima afficher une carte, et la position des rochers, des unités de nourritures et des fourmis, tour par tour, avec une vitesse d'animation raisonnable. Le lancement de l'interface graphique doit pouvoir se faire avec une commande `java ant.Animation team1.brain team2.brain large.world`.

3 Débogage

Cette partie est optionelle. Elle nécessite d'afficher dans une interface graphique un duel entre deux fourmilières mais en permettant de comprendre (et de déboguer) le code de la première fourmilière, le plus agréablement possible. En bref, l'outil dont vous rêviez (sans l'avoir...) pour mettre au point votre propore fourmilière lors du 1er projet !

Ce travail sera réalisé dans une classe `ant.Debug`. Le lancement de l'interface graphique doit pouvoir se faire avec une commande `java ant.Debug team1.brain team2.brain large.world`.

une machine à états finis simple et finie. Une machine est décrite par une liste de blocs de code. Chaque bloc commence par un label distinct, puis une liste de commande. De manière informelle, les commandes de cette machine d'état peuvent être décrites comme suit :

`Goto label`	saute au bloc `label`
`Sense sensedir label1 label2 cond`	saute au bloc `label1` si la condition `cond` est satisfaite dans la direction `sensedir`, saute au bloc `label2` sinon
`Mark i`	pose la marque `i` dans la cellule courante puis continue l'exécution du bloc courant
`Unmark i`	enlève la marque `i` (si elle existe) dans la cellule courante puis continue l'exécution du bloc courant
`PickUp label`	ramasse un élément de nourriture dans la cellule puis continue l'exécution du bloc courant; saute au bloc `label` si pas de nourriture
`Drop`	dépose un élément de nourriture (si la fourmie en porte un) dans la cellule courante, puis continue l'exécution du bloc courant
`Turn lr`	tourne à gauche ou à droite, puis continue l'exécution du bloc courant
`Move label`	avance d'une case dans la direction courante si c'est possible, puis continue l'exécution du bloc courant; si ce mouvement n'est pas possible, saute au bloc `lab_else`
`Flip p label1 label2`	tire un nombre aléatoire x entre 0 et `p`-1, puis saute au bloc `label1` si x=0, au bloc `label2` sinon

Voici un exemple de programme. La fourmi cherche de la nourriture en effectuant une marche aléatoire jusqu'à ce qu'elle trouve de la nourriture. Elle ramasse ensuite la nourriture et erre au hasard jusqu'à ce qu'elle se retrouve à la maison.

```
search:
  Sense Ahead pick_food flip Food  ; Y a-t-il de la nourriture devant moi ?
pick_food:
  Move search                      ; OUI : avance et prend la nourriture
  PickUp search                    ; repart au début si échec
  Goto go_home

flip:
  Flip 3 turn_left or              ; NON : tourne ou avance
turn_left:
  Turn Left                        ; tourne à gauche et continue la recherche
  Goto search
or:
  Flip 2 turn_right move
turn_right:
```

```
  Turn Right                        ; tourne à droite et continue la recherche
  Goto search
move:
  Move flip                         ; avance et continue la recherche
  Goto search

go_home:                            ; recherche la maison
  Sense Ahead home not_home Home    ; Est-ce que la cellule devant moi est la maison ?
home:
  Move go_home                      ; OUI : avance et dépose de la nourriture
  Drop                              ; puis repart en recherche
  Goto search
not_home:
  Flip 3 not_home_left not_home_or ; NON : tourne ou avance
not_home_left:
  Turn Left                         ; tourne à gauche et recherche la maison
  Goto go_home
not_home_or:
  Flip 2 not_home_right not_home_move
not_home_right:
  Turn Right                        ; tourne à droite et recherche la maison
  Goto go_home
not_home_move:
  Move not_home                     ; avance et recherche la maison
  Goto go_home
```

Notez la possibilité de mettre des commentaires après le symbole ';'. Un début de bloc commence par un label et termine forcément par un des instructions `Goto`, `Flip` ou `Sense`.

4 Syntaxe des instruction

```
instruction ::=   Sense  sensedir  label1  label2  cond
              |   Mark  i
              |   Unmark  i
              |   PickUp  label_error
              |   Drop
              |   Turn  lr
              |   Move  label_error
              |   Flip  p  label1  label2
              |   Goto  label
sensedir    ::=   Here
              |   Ahead
              |   LeftAhead
              |   RightAhead
cond        ::=   Friend
```

```
              |    Foe
              |    FriendWithFood
              |    FoeWithFood
              |    Food
              |    Rock
              |    Marker  i
              |    FoeMarker
              |    Home
              |    FoeHome
lr           ::=   Left | Right
i            ::=   0 | 1 | 2 | 3 | 4 | 5
p            ::=   1 | 2 | 3 | ...
```

5 Biologie

Une fourmi est rouge ou noir, ainsi que les caractéristiques suivantes

- Un numéro (identifiant) unique qui lui attribué en fonction de sa position initiale dans la carte. La fourmi le plus en haut à gauche possède l'identifiant 0, puis on numérote les autres fourmis par ordre croissant en parcourant la ligne de gauche à droite, puis la ligne suivante, etc... A chaque tour, les fourmis (quelque soient leurs couleurs) sont animées à tour de rôle en commençant les plus petits identifiants en premier.
- La position courante dans son code neurologique (une ligne du programme)
- Un compteur de *repos*. S'il est strictement positif, il est décrémenté au tour suivant, sans que la fourmi n'exécute d'autres actions. Il est placé à 14 quand la fourmi effectue une opération `Move`. On modélise ainsi le fait qu'un mouvement prendra plus de temps que les autres actions.
- Une direction courante. Il y a 6 directions possibles car les cellules sont des hexagones. Au départ toutes les fourmis regardent vers l'est.
- Une ou zéro unité de nourriture portée par la fourmi.

6 Carte

Une carte de jeu est spécifiée par un fichier ascii. Nous vous fournissons deux fichiers d'exemples. Les 3 premières lignes représente le zoom, la largeur et la hauteur de la carte. Le zoom agit sur l'affichage uniquement. Le reste du fichier spécifie ligne par ligne le contenu des cellules parmis les 5 possibilités suivantes :

- `#`: un rocher (infranchissable)
- `.`: une cellule vide
- `+`: une cellule de la maison des fourmis rouges
- `-`: une cellule de la maison des fourmis noires

- $x \in 0, \ldots, 9$: une cellule contenant x quantité de nourriture.

Il y a au plus une fourmi par cellule. Initialement elles sont toutes placées sur les cellules de leur maisons en occupant toutes les places disponibles. Les maisons ont des formes d'hexagones.

7 Arts martiaux

En plus de ramasser de la nourriture, les fourmis peuvent combattre avec d'autres fourmis. Les règles sont simples : si une fourmi se retrouve à un moment adjacente à 5 (ou 6) fourmis des autres espèces, elle meurt. Quand une fourmi meurt, elle se transforme en 3 particules de nourriture.

8 Marqueurs chimiques

Chaque fourmi peut placer et détecter 6 différents types de marqueurs chimiques, numérotés de 0 à 5.

Les marqueurs pour les deux couleurs des fourmis sont complètement séparés, c'est-à-dire que les marqueurs dans chaque cellule contiennent 12 bits d'information. Les fourmis d'une couleur donnée peuvent individuellement détecter, définir et effacer les 6 marqueurs de leur propre couleur, mais ne peuvent détecter que la présence d'un marqueur appartenant aux autres espèces.

Contrairement aux marqueurs chimiques utilisés par les vraies fourmis, les marqueurs de ce jeu persistent jusqu'à ce qu'ils soient explicitement effacés. Initialement, aucune cellule ne contient de marqueurs.

9 Scores

Les zones de maison rouge, noir, cailloux et nourritures sont initialement disjointes. A la fin des 100 000 tours de jeux, le score de chaque équipe est égal au nombre de nourritures déposées dans sa maison. La nourriture encore portée par une fourmi ne compte pas. L'équipe gagnante est celle avec le meilleure score.