Projet 1 NetLogo

Boudjémâa Salah

À noter : La numérotation des annexes correspond à la numérotation dans les souris fichiers de chaque exercice.

Exercice 1

Ce premier exercice fonctionne avec 6 boutons et un “**choicer**” (*Figure 1)*

Ordre d’utilisation :
clear → new-turtle → yellow-turtle → new-pcolor → new-patch → yellow-patch → move-turtle

Ces boutons sont associés à 6 méthodes permettant le bon fonctionnent de l’exercice.

Variables globales :
coo-**x/coo**-**y** coordonnées de taille du world
mx/**my** coordonnées du patch sélectionné

clear
La méthode permet dans un premier temps de redimensionner la zone
Dans un second temps, la méthode “**nettoie**” la zone.

- Recolorer tous les patchs en noir (couleur de base)

new-turtle *(Figure 2)*
Après avoir retiré la tortue présente,
la méthode crée une tortue blanche, de taille 5 au centre l’écran

yellow-turtle *(Figure 3)*
La méthode change la couleur de la tortue présente pour du jaune

new-patch *(Figure 4)*
La méthode enregistre le clic de la souris ainsi que ses coordonnées à ce moment-là.
Puis, elle nettoie les patchs afin de colorer le patch choisi uniquement.
La couleur peut être choisie via le **chooser**.
La méthode est de type “**forever**” elle tourne en boucle, jusqu'à ce que le patch soit sélectionné

yellow-patch *(Figure 5)*
La méthode colore le **patch** sélectionné en jaune

move-turtle *(Figure 6)*
Une fois la méthode activée, la tortue se tourne vers le patch sélectionné, puis avance en ligne droite jusqu'à ce que la distance les séparant atteigne le seuil limite.

Exercice 2

Pour cet exercice, on retrouve, sur l'interface, 9 boutons et un champ numérique. *Figure 1*

Le code ainsi que le fonctionnement général est basé sur le premier exercice, c'est pourquoi je ne parlerai que des éléments changeants.

Dans cet exercice, les méthodes **new-turtle** et **new-patch** génèrent une position aléatoire au lieu d'une sélection à la souris, ou d'une position centrale.

La méthode **move-turtle** est la même que précédemment, simplement, elle colore les patchs survolés au fur et à mesure de la course de la tortue. *Figure 2*

La méthode **resest-pos-turtle** permet de positionner la tortue au début du chemin parcouru. Pour cela, au moment de créer la tortue, on enregistre sa position dans une variable globale, qui sera récupérée plus tard par la méthode. *Figure 3*

La deuxième partie de l'exercice permet de dessiner différente forme grâce au chemin de la tortue. Cela est permis par des **méthodes/bouton** associé à chaque forme. Elles sont construites autour de répétition de rotations et lignes droites. *Figure 4 à 7*

Il faut noter que chacune des méthodes impliquant les tortues est réalisée du point de vue de ces dernières.

Exercice 3

Cet exercice a le même objectif que le précédent.
La différence se joue sur l'exécution des méthodes impliquant les tortues.
Les méthodes sont exécutées du point de vue de **l'Observer**. Pour que cela soit possible, on utilise le mot clé "**ask**" qui nous permet de demander à un groupe d'exécuter une tâche

qui lui est propre. Par exemple, on va demander aux **tortues**, via **l'Observer,** de déplacer les tortues vers le patch sélectionné.

Exercice 4

Pour cet exercice, on retrouve 2 **boutons** et 4 **sliders**. *Figure 1*

La méthode **setup** méthode permet de "nettoyer' et redimensionner la zone et mettre en place 30 **agents/turtles**. Ici, les agents sont des flèches, afin de bien observer le sens de déplacement. Les agents sont créés au centre orienté vers le haut.
La méthode **move** génère dans un premier temps un nombre aléatoire qui sera soit 0, soit 1. En fonction du résultat, le mouvement n'est pas le même : orientation et nombre de pas définis par les **slider**. Cette procédure est appelée pour **chaque agent**, afin qu'ils aient leur propre déplacement. Afin de mieux visualiser les choix de chacun, chaque changement de direction et accompagné d'un changement de couleur : rouge pour le premier mouvement, bleu pour le second. *Figure 2*

Exercice 5

Pour cet exercice, on retrouve 4 boutons et une fenêtre input *Figure 1*

La méthode setup est la même que précédemment

Pour le **randow-walk** on choisit de faire une procédure d'agent/tortue dont le fonctionnement est simple.
Pour chaque tortue, une rotation à gauche aléatoire est chose entre 0 et 360 (pareil si on avait choisi droite).
Une distance est choisie de la même manière, cette fois-ci entre 0 et l'hypoténuse de la taille du monde (distance max en ligne droite)
La méthode est en "**forever**" l'exécution est donc continue jusqu'à relâchement du bouton.

On peut utiliser différemment la méthode créée précédemment.

La méthode **random-repeat** va utiliser la fonction **repeat** native de NetLogo pour appeler la méthode random walk un certain nombre de fois. Nombre définit par la fenêtre **d'input** dédiée. Cette méthode reste une méthode de **turtle** afin que les tortues bougent simultanément, et non une à une.

La méthode **loop-random-walk** utilise la fonction native **loop** afin de lancer en boucle la méthode **random-walk** jusqu'à atteinte de la condition, ici le nombre de mouvements pour les tortues. 300 mouvements totaux pour 10 mouvements par tortue, par exemple. La fenêtre d'input utilisée est la même que la méthode précédente.

Exercice 6

Dans cet exercice, l'objectif est de simuler le comportement de termites, lorsqu'elles rassemblent des copeaux de bois afin de former leur termitière.
Ce que nous allons chercher à observer, c'est donc que malgré des mouvements initiaux aléatoires, on peut tout de même observer la formation de pile de copeaux.

Nous allons dans un premier temps créer un certain nombre de termites rouge au centre de notre **world**. *Figure 1*
Cette action est faite dans la méthode **setup** qui permet également de redimensionner le **world** et initialiser les variables globales.

Afin de créer un déplacement aléatoire aux termites, nous créons la méthode **wiggle**. Cette dernière va ordonner à chaque termite d'avancer d'un pas, puis de réaliser une rotation aléatoire.

Une fois que nos termites peuvent se déplacer, il faut qu'elles aient un objectif.
Pour cela, nous allons représenter les copeaux de bois à ramasser, par de patch jaunes. Ils seront générés à des positions aléatoires grâce à la méthode **patch-setup.** Le nombre de copeaux est choisi via un **chooser** sur l'interface principale. Pour être sûr d'avoir le nombre voulu (éviter les superpositions). La méthode utilise une boucle qui génère des copeaux tant que le nombre de copeaux jaunes est inférieur au nombre voulu. *Figure 2*

La boucle principale de notre programme est représentée par la fonction récursive **search-for-pellet.**
Chaque termite va avancer de manière aléatoire, jusqu'à avancer sur un patch jaune. Une fois un tel patch atteint, celui devient noir (le copeau est ramassé) *Figure 3*. Le termite lance alors la procédure de recherche d'une pile existante, la méthode **find-new-pile.**
Cette dernière va chercher une pile qui peut accueillir à ces cote, un nouveau copeau. Autrement dit, on cherche un patch jaune avec au moins voisin noir. Une fois qu'un tel patch est trouvé, le termite s'y dirige et dépose le copeau sur un des voisins encore noir.
De cette manière, on évite les entassements de copeau qui résultent en la disparition de certain.
Une fois le copeau déposé, on lance la procédure **get-away** qui nous permet de nous éloigner de la pile de copeau, afin de ne pas ramasser celui que l'on vient déposer.
Pour ce faire, le termite fait un demi-tour et avance de 5 pas, avant de reprendre un mouvement aléatoire (nouvelle exécution de la méthode **search-for-pellet**).
Finalement, pour différencier les termites qui cherchent un copeau et ceux qui cherchent une pile, les premiers sont représentés en jaune, les autres en bleu.

Afin d'étudier la formation de pile de copeaux, on peut ajuster les paramètres de nombre de termites et nombre de copeaux, mais aussi le nombre de "tours". On définit ici le nombre de

tours comme le nombre de déplacements de copeaux par termite. Le nombre déplacement total est donc ***nb-tour * nb-termites.***

*O*n définit comme une pile un ensemble d'au moins 3 patchs reliés par leurs voisins non diagonaux. Afin de les repérer, elles seront colorées en vert au fur et à mesure de la simulation. C'est le travail de la méthode **recolor-patch** qui sera appelée dès qu'un patch est retiré ou ajouté. *Figure 4*

On peut finalement visualiser ces données grâce à des **graphes** directement sur l'interface principale. On y ajoutera le nombre de **patchs** à récupérer (patch encore **jaune**) *Figure 5*

Finalement, la simulation fonctionne comme prévu. Plus il y a de **termites**, plus les termitières sont formées **vite**. Plus il y a de **tours max**, plus les structure formées sont **importantes**.