Comme le problème de la collecte de colis par des agents coopératifs, le problème de la patrouille multi-agents constitue un banc d'essai pour étudier, raffiner, expérimenter et évaluer les algorithmes multi-agents. Patrouiller consiste à parcourir une zone géographique pour la surveiller, par exemple un réseau local susceptible de subir des attaques informatiques, des pages web récemment modifiées devant être indexées par un moteur de recherche ou une zone pour localiser des personnes en danger devant être secourues. Dans ce TT, vous devez concevoir divers comportements d'agent pour patrouiller, les implémenter et les évaluer à l'aide de NetLogo.

Exercice 1 : Environnement

Nous considérons ici que la zone à surveiller est représentable par un graphe où n nœuds, qui sont les emplacements à visiter par m patrouilleurs, sont reliés par des arêtes. Nous considérons :

- qu'il n'y a pas d'obstacles mobiles sur le terrain (les arêtes sont statiques);
- que la distance entre deux nœuds connectés est égale à un (la longueur des arêtes est unitaire);
- que tous les emplacements ont la même priorité;
- que le nombre de patrouilleurs est constant au cours de la simulation.

Q1. Créez un monde de 20 $\times$20 tuiles qui ne soit pas un tore. Ajoutez des boutons pour l'initialisation, l'exécution continue et l'exécution d'un pas de simulation. Créer un curseur, pour fixer le nombre maximal de pas de simulation entre 3000 et 4000 ainsi qu'un menu pour déterminer le nombre de patrouilleurs (1, 2, 4, 8, 16, 32 ou 64).

Q2. Créez une espèce de tortue intitulée patrouilleur (de forme `airplane`) munie de la propriété `id`.

Q3. Écrivez la procédure d'initialisation qui :

- crée les patrouilleurs, chacun ayant une étiquette contenant son identifiant;
- positionne les patrouilleurs au centre du monde.

Q4. Écrivez la procédure d'exécution qui :

- se termine quand le nombre de pas de simulation est atteint;
- invoque le comportement individuel de chaque patrouilleur.

Q5. Implémentez un comportement individuel de patrouilleur qui consiste, à chaque pas de simulation, à se déplacer de manière aléatoire sur l'une des 4 tuiles adjacentes.

Exercice 2 : Objectif

Le problème de la patrouille consiste à définir un comportement global pour que chaque emplacement soit visité le plus souvent possible.

Considérons un graphe $G = (V, E)$ de n nœuds et qu'un pas de simulation est nécessaire à un patrouilleur pour aller à un nœud adjacent. Nous appelons oisiveté d'un noeud $i \in V$ au pas de simulation t (notée $Idle_t(i)$) l'intervalle de temps écoulé depuis la dernière visite d'un patrouilleur sur ce noeud. Par convention à l'instant 0, l'oisiveté de chaque emplacement est nulle.

Q1. Associez à chaque tuile une propriété d'oisiveté. Dans la procédure d'initialisation, chaque tuile se voit doter d'une oisiveté nulle. Étiquettez chaque tuile avec cette propriété. Dans la procédure d'exécution, incrémentez l'oisiveté de chaque tuile et mettez à jour son étiquette à chaque pas de simulation. Dans le comportement de patrouilleur, réinitialiser à 0 l'oisiveté de la tuile visitée et mettez à jour son étiquette.

Nous appelons :

- oisiveté moyenne du graphe G au pas de simulation t

$$AvgIdle_t(G) = \frac{1}{n} \sum_{i \in V} Idle_t(i) \tag{1}$$

— oisiveté maximale du graphe G au pas de simulation t

$$MaxIdle_t(G) = \max_{i \in V} Idle_t(i) \tag{2}$$

Dans un souci de simplicité, nous nous limiterons à l'oisiveté maximale du graphe.

Q2. À chaque pas de simulation, calculez et affichez l'oisiveté maximale dans un moniteur.

À partir de cette mesure instantanée, nous définissons un critère pour comparer les différentes stratégies multi-agents dans une simulation de t_f pas qui correspond au plus grand intervalle de temps durant lequel un emplacement ne reçoit la visite d'aucun patrouilleur :

$$MaxMaxIdle(G) = max_{t \leq t_f} MaxIdle_t(G) \tag{3}$$

Pour mesurer la contribution individuelle des patrouilleurs, ce critère peut être normalisé :

$$\widehat{MaxMaxIdle}(G) = \frac{m}{n} \times MaxMaxIdle(G) \tag{4}$$

Q3. Ajoutez à l'interface un graphique qui affiche au cours de la simulation $\widehat{MaxMaxIdle}(G)$.

Exercice 3 : Comportement réactif

La seule action réalisable par un agent est le déplacement vers l'une des tuiles adjacentes si cette dernière est accessible. Deux comportements réactifs sont possibles. Selon si la prise de décision est aléatoire ou heuristique, le déplacement est arbitraire ou se fonde sur l'oisiveté des emplacements adjacents.

Q1. Créez un interrupteur pour fixer la stratégie des patrouilleurs. Coder un comportement individuel réactif d'agent qui se déplace en fonction de sa stratégie. Simulez 3000 pas de simulation avec X agents. Comparer l'efficacité de ces deux stratégies à l'aide de $\widehat{MaxMaxIdle}(G)$.

Exercice 4 : Comportement cognitif

On considère que la perception de l'environnement par les patrouilleurs est globale, c'est-à-dire qu'un patrouilleur peut percevoir l'oisiveté de toutes les tuiles. Un patrouilleur cognitif se dirige alors vers la tuile qui n'a pas été visitée depuis le plus longtemps, en utilisant un algorithme de plus court chemin. Pour ce faire, une fonction d'utilité évalue chaque tuile en prenant en compte son coût (distance) et son gain (oisiveté).

Q1. Associez à chaque tuile les propriétés de gain et de coût. Implementez la procédure qui réinitialise ses propriétés pour toutes les tuiles.

Q2. Créez un interrupteur pour sélectionner le comportement cognitif ou réactif des patrouilleurs. Modifiez la procédure de simulation avec l'extrait de code suivant :

```
; exécute le comportement des patrouilleurs
ifelse cognitif [
foreach (list patrouilleurs) [
patrouilleurCourant ->
reinitialiser-utilite
ask patrouilleurCourant [comportement-cognitif]
]
][
ask patrouilleurs [comportement-reactif]
]
```

Q3. Implémentez les procédures requises pour le comportement d'agent suivant :

```
to comportement-cognitif
; Calculer l'utilité des emplacements
calculer-utilité
; Trouver la tuile avec la meilleur utilité
let cible max-one-of patches [gain - coût]
; Trouver le chemin le plus court vers la cible
let chemin (bfs-to cible)
; Se déplacer vers la première tuile du chemin
ifelse is-list? chemin [
move-to first chemin
] [
; Se déplacer vers la tuile cible directement
move-to chemin
]
reinitialiser-oisivete-courante
end
```

Exercice 5 : Évaluation

BehaviorSpace est un outil (menu Tools) intégré à NetLogo qui vous permet de réaliser des expériences avec votre modèle à base d'agents. Il exécute un modèle plusieurs fois, en faisant varier les paramètres du modèle et en enregistrant les résultats de chaque exécution du modèle. Si votre ordinateur possède plusieurs cœurs, les exécutions se déroulent en parallèle.

Q1. Créez une expérience qui vous permettra de comparer l'efficacité des comportements avec 1, 2, 4, 8, 16, 32 ou 64 patrouilleurs pour 3000 pas de simulations. Fixez à 10 le nombre d'exécution de chaque configuration. Tracez les courbes des $\widehat{MaxMaxIdle}(G)$ pour les différents comportements en fonction du nombre de patrouilleurs.