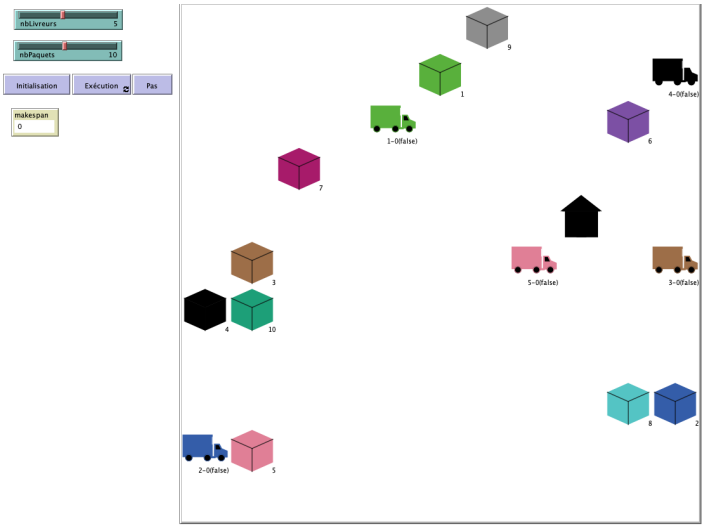

Figure 1 – Collecte de colis

Le problème de la collecte de colis par des agents coopératifs constitue un banc d'essai pour étudier, raffiner, expérimenter et évaluer les algorithmes multi-agents. Dans ce TP, vous devez concevoir des comportements d'agent pour réaliser la collecte.

Exercice 1 : Problème

Vous considérez un environnement qui est une grille de tuiles (cf. Fig. 1). Chacune contient au plus une tortue, qu'elle représente un livreur, une destination ou un paquet.

Q1. Créez un monde de 10 $\times$10 tuiles qui ne soit pas un tore. Ajoutez un curseur pour le nombre de livreurs (entre 1 et 10) et un autre pour le nombre de paquets (entre 1 et 20). Ajoutez des boutons pour l'initialisation, pour l'exécution continue et pour l'exécution d'un pas de simulation.

Q2. Créez trois espèces pour :
- la destination (de forme `house`) ;
- les livreurs (de forme `truck`) muni des propriétés `id` et `estCharge` ;
- les paquets (de forme `box`) muni de la propriété `id`.

Q3. Écrivez la procédure d'initialisation qui crée la destination, les livreurs et les paquets. Chaque individu de chaque espèce a une couleur différente. Ajoutez aux tortues une étiquette qui affiche leurs propriétés.

Exercice 2 : Actions

Les actions réalisables par un livreur sont :

1. le dépôt d'un paquet si le livreur chargé est sur une tuile adjacente à la destination ;
2. le chargement d'un paquet si le livreur, qui n'est pas chargé, est sur une tuile adjacente à ce paquet ;
3. le déplacement vers l'une des tuiles adjacentes. Si cette dernière est occupée par une tortue, les positions sont échangées.

Q1. Écrivez la procédure `deplacer [objectif]` qui permet à une tortue de se déplacer vers la prochaine tuile en direction de l'`objectif` quitte à échanger ses coordonnées à celle de la tortue sur la prochaine tuile. À cette intention :
- la primitive `face objectif` permet de se tourner dans la bonne direction ;
- la procédure `prochaine-tuile` retourne la prochaine tuile selon cette direction ;

— la primitive `move-to prochaine-tuile` permet de se placer sur la prochaine tuile.

```
to-report prochaine-tuile
if heading < towardsxy (pxcor + 0.5) (pycor + 0.5)
[ report patch-at 0 1 ]
if heading < towardsxy (pxcor + 0.5) (pycor - 0.5)
[ report patch-at 1 0 ]
if heading < towardsxy (pxcor - 0.5) (pycor - 0.5)
[ report patch-at 0 -1 ]
if heading < towardsxy (pxcor - 0.5) (pycor + 0.5)
[ report patch-at -1 0 ]
report patch-at 0 1
end
```

Q2. Créez un comportement réactif de livreur qui exécute ses actions en fonction de son état et de sa perception. Simulez ce comportement jusqu'à ce que tous les paquets soient livrés.

Exercice 3 : Effort

Arbitrairement, chaque action a un coût unitaire. L'effort d'un livreur correspond à la somme des coûts de ses actions déclenchées ou subies. On appelle *makespan*, l'effort consenti par le livreur le plus sollicité.

Q1. Associez à chaque livreur une propriété `effort` qui mesure le nombre d'actions réalisées. Ajoutez à l'interface un moniteur qui affiche, au cours de la simulation, le *makespan*. Quelle est sa valeur pour le comportement réactif ?

Exercice 4 : Tâches

Afin de réduire le *makespan*, vous allez mettre en œuvre un comportement d'agent proactif. Pour être tous collectés, chaque paquet doit être ciblé par un livreur.

Q1. Ajoutez à l'interface un sélecteur qui permet de choisir si le `comportement` est « `réactif` » ou « `proactif` ».

Q2. Associez à chaque livreur une propriété `cibles` qui est une liste ordonnée de paquets à collecter. Si le `comportement` est « `proactif` », ajoutez cette propriété à son étiquette et allouez aléatoirement les paquets aux livreurs lors de l'initialisation. Ces paquets doivent être de la couleur du livreur qui en est responsable.

Q3. Créez un comportement proactif de livreur qui :

1. ne fait rien s'il n'est pas chargé et qu'il n'a pas de cible ;
2. s'il est chargé,
   (a) dépose son paquet s'il est sur une tuile adjacente à la destination,
   (b) se déplace vers la destination sinon;
3. s'il n'est pas chargé mais qu'il a une cible,
   (a) charge le paquet s'il est sur une tuile adjacente à sa cible,
   (b) se déplace vers la cible sinon.

Exercice 5 : Planification

Le coût de la collecte d'un paquet par un livreur correspond (a) à la distance pour atteindre une case adjacente à ce colis, (b) au chargement, (c) à la distance pour atteindre une case adjacente à la destination, et (d) au dépôt du colis. Il est important de noter que le coût d'une tâche (i.e. un colis à collecter) ne correspondent pas nécessairement aux efforts observés. Certaines actions sont imprévues (la destination ou la cible a été déplacée) ou non sollicitées (e.g. le livreur est poussé par une autre).

Q1. Écrivez la procédure qui retourne le coût estimé de la collecte d'un paquet par un livreur à l'aide de la primitive `destination`. Ajoutez une propriété à chaque paquet dont la valeur correspond au coût pour le livreur qui en a la charge. Affichez cette propriété dans l'étiquette des paquets.

Q2. Écrivez la procédure qui estime le *makespan* de l'allocation courante. Affichez-le dans un moniteur lors de l'initialisation de l'allocation.

Q3. Écrivez la procédure qui retourne le coût estimé de la collecte d'un paquet par un livreur à l'aide de l'algorithme A*. Modifiez votre simulation pour que cette nouvelle procédure soit utilisée en lieu et place de la précédente.

Exercice 6 : Méthode d'affectation

Le problème d'ordonnancement sous-jacent à la collecte de colis consiste à minimiser la durée globale de réalisation (*makespan*) avec m exécutants multi-tâches et n tâches mono-exécutant dont les coûts dépendent de l'entité exécutante. Ce problème est NP-difficile. L'heuristique d'affectation la plus connue, intitulée ECT (*Earliest Completion Time*), consiste à assigner une-à-une chaque tâche à l'exécutant en minimisant le *makespan*.

Q1. Ajoutez à l'interface un sélecteur qui permet de choisir si l'allocation initiale est aléatoire ou calculée par ECT.

Réaffecation de tâches

Afin de réduire le *makespan*, chaque agent peut être amener à négocier une délégation de paquet qui n'est pas sa cible courante mais dont le coût s'avère inférieur pour l'un de ses pairs.

Q2. Ajoutez à l'interface un sélecteur qui permet (ou pas) une phase de réaffectation à chaque pas de simulation. Observez-vous des délégations ?

Exercice 7 : 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 déroulent en parallèle.

Q1. Fixez la taille du monde à 20 $\times$20 tuiles. Supprimez le curseur pour le nombre de paquets pour le fixer à 2 fois le nombre de livreurs. Créez une expérience qui vous permettra de comparer la *makespan* avec et sans réaffectation d'une allocation aléatoire en fonction du nombre de livreurs, de 2 à 10. Fixez à 100 le nombre d'exécution de chaque configuration. Tracez les courbes des *makespan* avec ou sans réallocation en fonction du nombre de livreurs.