



TP2 Version 2 - SMA

POLYTECH LYON

MANG Quentin BONNET Corentin | Système multi-agent | 08/12/2021

Table des matières

1. Fonctionnement du TP – version 2.....	1
2. Version 2 - Modélisation	2
3. Démonstration	5

1. Fonctionnement du TP – version 2

Notre version 2 du TP SMA est décomposé de la même manière que la partie 1 avec toujours 4 classes Java (*Main*, *Environnement*, *Agent* et *Direction*). La structure de l'environnement et des agents ne change pas et sont représentés par les classes *Environnement* et *Agent*. L'objet de type C a été codé de la même manière que les autres objets, à savoir, représenté par la lettre C dans notre map.

Voici les nouvelles fonctionnalités dans les classes intéressantes :

- *Environnement.java* : Dans cette version 2, la classe environnement va toujours avoir le rôle de perceuteur (conception environnement / agent) en renvoyant les différentes actions possibles à un agent. Ainsi, nous avons rajouter 3 fonctions principales qui sont :
 - *DiffusionSignal()* : Cette fonction permet de diffuser le signal à partir d'une case de l'environnement. Cette case représente l'endroit où un agent est bloqué et attend de l'aide pour prendre l'objet de type C. Pour se faire, cette fonction prend l'agent « bloqué » en entrée et récupère ces coordonnées. En fonction de la distance « ds », la fonction va parcourir un rayon de longueur « ds » autour de l'agent pour relever les agents disponibles (ne portant rien) autour de lui. Enfin, la fonction retourne un agent qui va aller aider l'agent bloqué. L'intensité du signal sera géré par la fonction *Tchebychev_function()*, expliqué ci-dessous.
 - *Tchebychev_function()* : Pour calculer la distance entre l'agent demandant de l'aide et le second agent, nous avons mis en place la distance de Tchebychev permettant de donnée la différence maximale entre 2 points sur un plan. Cela nous permet donc de calculer l'intensité du signal, selon la position d'un agent.
 - *SeDeplacerV2()* : Cette fonction prend en entrée l'agent bloqué et l'agent appelé. Avec cette fonction, l'agent appelé va se déplacer en direction de l'agent bloqué pour l'aider à porter l'objet C.

Cette fonction renvoie un booléen signifiant s'il se situe sur la même case que l'agent bloqué ou non.

- *Agent.java* : Cette classe va diriger les décisions de l'agent sans que ce dernier ne sache où il se trouve sur la carte, afin de respecter le concept environnement / agent. Dans la version 2, les agents doivent communiquer entre eux afin de se coordonner pour porter les objets de type C. En effet, les agents ne sont conscients que d'eux-mêmes et c'est l'environnement qui va permettre de les articuler ensemble.

Nous avons rajouté 2 variables booléennes globales : « bloque » et « appelle ».

La variable « bloque » indique que l'agent courant a besoin d'aide pour déplacer un objet C et ne peut plus se déplacer.

La variable « appelle » permet à l'agent qui vient aider l'agent courant de savoir qu'il a été appelé. Lorsque « appelle » est à « True », l'objectif de l'agent appelé est de venir aider l'agent bloqué.

Dans notre code, l'agent venant aider est déclaré comme « *AgentLié* » de type Agent.

Les fonctions qui ont été rajoutées sont :

- *PrendreObjetC()* : Cette fonction permet à l'agent courant de prendre l'objet C.
- *EnvoyerSignal()* : Cette fonction permet à l'agent courant d'envoyer un signal aux autres agents (par l'intermédiaire de l'environnement), afin de trouver un agent pour l'aider. Cette fonction prend en paramètre l'agent courant pour récupérer ses coordonnées sur la map.

Enfin, nous avons ajouté des morceaux de code pour le dépôt d'un objet C par un agent dans les fonctions de dépôt.

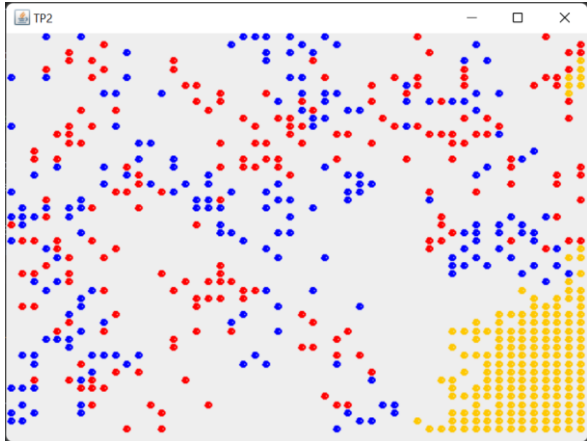
La gestion de l'intensité, du taux r et du renouvellement de l'appel se fait dans la fonction *action()* de la classe *Agent.java*.

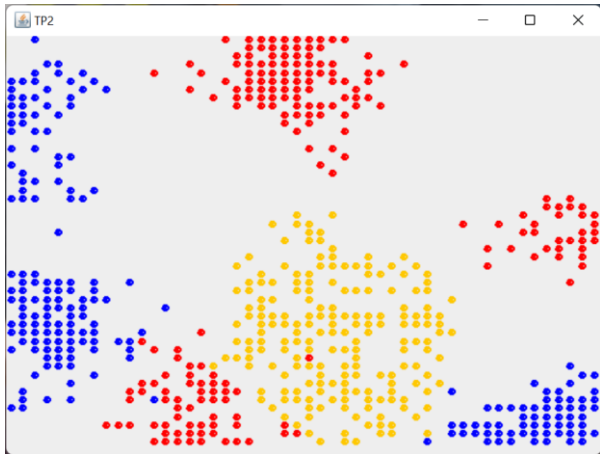
En effet, c'est ici que l'agent va prendre la décision d'une action. Par exemple, si l'intensité est assez forte pour que l'agent appelé vienne aider l'agent courant, ou encore si le temps d'attente de l'agent courant excède le renouvellement de l'appel alors l'agent courant se débloque...

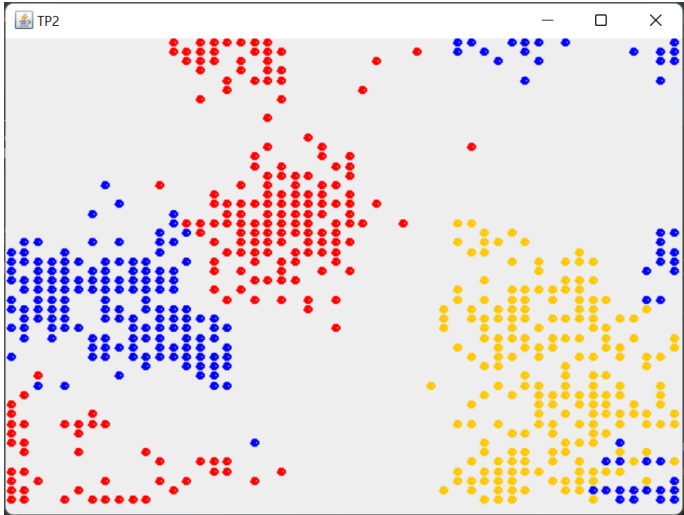
Toutes les variables supplémentaires pour la version 2 sont paramétrables dans le constructeur d'*Environnement*.

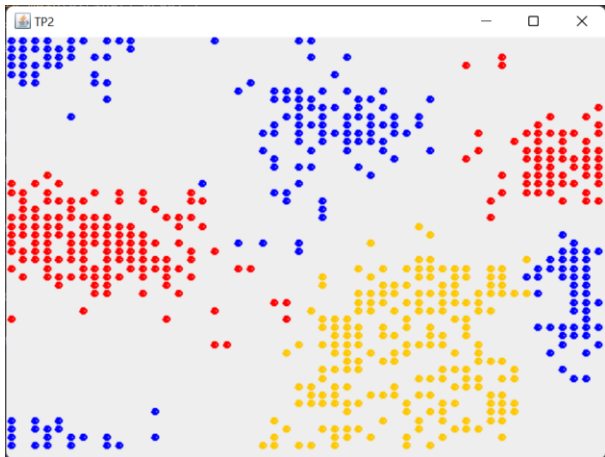
2. Version 2 - Modélisation

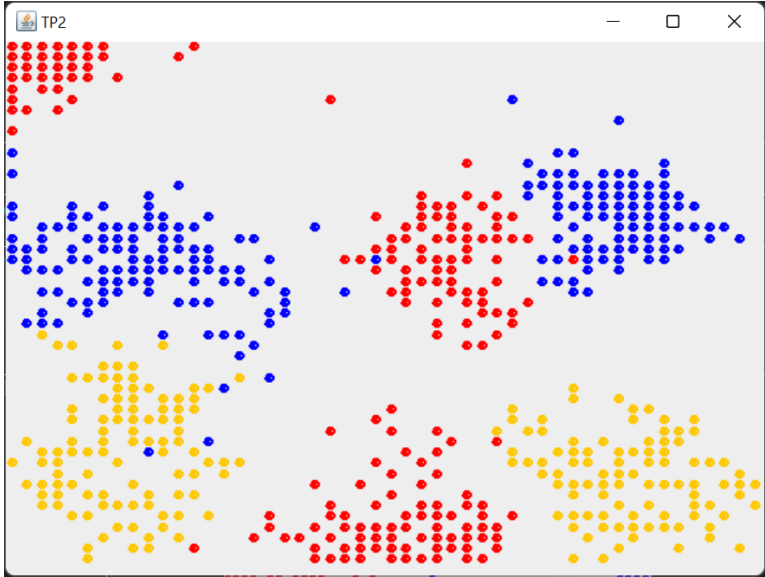
Voici notre système avec les valeurs suivantes pour une grille de 50 x 50 avec $k\text{-plus} = 0.1$ et $k\text{-mois} = 0.3$, taux d'erreur = 0.0, taille de la mémoire = 10 et avec 200 objets de chaque type :

	Itération : 1 000 000	Nb agent : 20	Distance (ds) : 2	Taux r : 0.1	Renouvellement appel : 3
Temps (ms)	5396 ms				
Description	On remarque la formation d'un cluster d'objet C mais que les objets A et B restent toujours éparpillés sur la carte, n'arrivant pas à former de cluster.				
Illustration					

	Itération : 5 000 000	Nb agent : 20	Distance (ds) : 2	Taux r : 0.1	Renouvellement appel : 3
Temps (ms)	18231 ms				
Description	On remarque la formation d'un cluster d'objet C au centre de la carte et que les objets A et B commencent à former plusieurs clusters séparés.				
Illustration					

	Itération : 1 000 000	Nb agent : 20	Distance (ds) : 10	Taux r : 0.1	Renouvellement appel : 3
Temps (ms)	5943 ms				
Description	On remarque qu'il y a la formation d'un cluster d'objet C qui se fait plus rapidement car la distance augmente et l'agent bloqué peut a plus de chance de porter un objet C. Les objets A et B forment des clusters séparés sur la carte.				
Illustration					

	Itération : 1 000 000	Nb agent : 20	Distance (ds) : 10	Taux r : 0.1	Renouvellement appel : 10
Temps (ms)	7328 ms				
Description	On remarque qu'il y a la formation d'un cluster d'objet C qui se fait encore plus rapidement qu'à l'essai précédent car l'agent bloqué renouvelle plus de fois son appel à l'aide pour porter un objet C. Les objets A et B forment des clusters séparés sur la carte, de manière plus éparses que précédemment.				
Illustration					

	Itération : 1 000 000	Nb agent : 20	Distance (ds) : 10	Taux r : 0.5	Renouvellement appel : 10
Temps (ms)	30672 ms				
Description	<p>On remarque que la modification du taux r va directement influencer sur la formation des clusters pour l'objet C. Plus r va être élevée, plus les phéromones vont s'évaporer rapidement sur la carte. Ainsi, les agents appelés ont peu de chance de retrouver la case de l'agent bloqué pour lui apporter de l'aide.</p> <p>On remarque ici deux clusters C et plusieurs clusters A et B sur la carte.</p>				
Illustration					

Nous sommes conscients que notre code n'est pas optimisé et que les temps de traitement sont relativement longs et que le déplacement des agents n'est pas optimal. Cependant, nous avons optimisé la détection des agents ainsi que leur intensité associée grâce à la distance de Tchebychev issue d'une distance euclidienne.

3. Démonstration

Une vidéo de démonstration est aussi disponible dans le répertoire du projet sous format « mp4 » s'intitulant : « TP2 version 2 - Demo.mp4 ». Cette démonstration est faite avec une map de 50x50, 20 agents, 200 objets A, 200 objets B et 200 objets C, une taille mémoire de 10, un taux d'erreur de 0, ds égale à 2, r égale à 0.1 et renouvellement appel égale à 3. Le nombre d'itération est de 1 000 000.