Rapport de projet - Systèmes Multi-Agents

Tri sélectif - Partie 1

M1 Informatique - S2 2021

Enseignante : Salima HASSAS

Titouan Knockaert & Gaspard Goupy

Sommaire

Modélisation **2**

1.1 Grid 2

1.2 Cases 2

1.3 Objets 2

1.4 Robots 3

Expériences et observations **4**

Modélisation

1. Grid

Sert de base au modèle, se compose d'un tableau à 2 dimensions d'objet Cases et d'une liste de Robots.

nb: dans la partie 2, la grille a la responsabilité de la diffusion des phéromones de case en case.

2. Cases

Une case peut contenir un robot et un objet à la fois. Lorsqu'un robot prend un objet celui-ci est toujours considéré comme sur la case, on ne peut donc pas avoir à la fois un robot avec un objet porté et un objet libre sur la même case.

nb: dans la partie 2, les cases possèdent aussi un objet phéromone.

3. Objets

```
public class Objet{
    char type;

    public Objet(char t){
        type = t;
    }
}
```

4. Robots

Attributs

```
int nbCaseParPas; // Vitesse en cases parcourues/appel à move()
double kplus; // Facteur modifiant la chance de prise d'un objet
double kminus; // Facteur modifiant la chance de dépôt d'un objet
Queue<Character> memory; // Mémoire du robot
int memorySize; // Taille de la mémoire
Objet inHand; // Objet tenu par le robot
Case caseOn; // Case sur laquelle est le robot
Grid grid; // Grille sur laquelle est le robot
int x; // Position en abscisse du robot sur la grille
int y; // Position en ordonnée du robot sur la grille
int aInMemory = 0; // Nombre d'objets de type A en mémoire
int bInMemory = 0; // Nombre d'objets de type B en mémoire
```

```
double error = 0; // Partie 1 : ajout d'une erreur dans la perception
```

Fonctionnement

Les Robots suivent une boucle perception-action détaillée ci-dessous :

Perception :
À chaque passage dans la boucle, un Robot regarde le type de l'objet présent sur sa case (ou l'absence de celui-ci), et actualise sa mémoire en fonction. Dans notre modélisation un robot ne peut pas aller sur une case sur laquelle il y a déjà un objet alors qu'il a un objet en main. Nous avons donc ajouté une perception différente lorsque le robot a un objet en main : au lieu de regarder l'objet sur sa case, il regarde toutes les cases voisines et ajoute à sa mémoire le premier objet vu, s'il en voit un, sinon il note l'absence d'objet.

Action :
Si le Robot a un objet de type T dans les mains :

- Tire un nombre aléatoire entre 0 et 1
- Calcule f : $f = (\#T \text{ in memory} + (\#\text{not } T * error) / \text{memory size}$
- Si le nombre tiré est inférieur à $(f / (k^- + f))^2$, lâche l'objet

Si le robot a les mains vides et se trouve sur une case avec un objet de type T:

- Calcule f : $f = (\#T \text{ in memory} + (\#\text{not } T * error) / \text{memory size}$
- Tire un nombre aléatoire entre 0 et 1
- Si le nombre tiré est inférieur à $(k^+ / (k^+ + f))^2$ prend l'objet

Après ces deux étapes le Robot se déplace :

- Génère un nombre entre -1 et 1 qui correspond à la direction en abscisse
- Génère un nombre entre -1 et 1 qui correspond à la direction en ordonnée
- Se déplace d'autant de case que prévu par sa vitesse si le déplacement dans la direction obtenue est possible (case dans la grille et case libre)

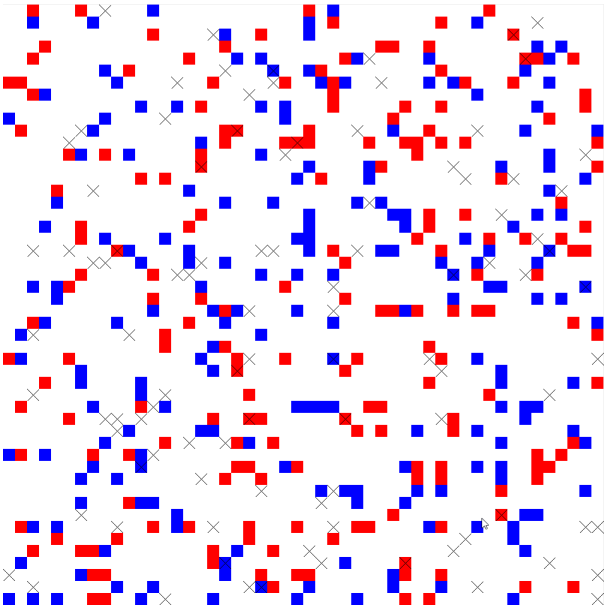
Résultats et observations

Pour cette partie nous avons fixé les paramètres suivant en accord avec le sujet :

```
int nbA = 200;
int nbB = 200;
int nbRobot = 100;
int width = 50;
int height =  50;
int speed = 1;
double kplus = 0.1;
double kminus = 0.3;
```

Nos expérimentations se feront sur des changements sur la taille de la mémoire, le nombre d'itérations, le taux d'erreur de perception, et le cooldown.

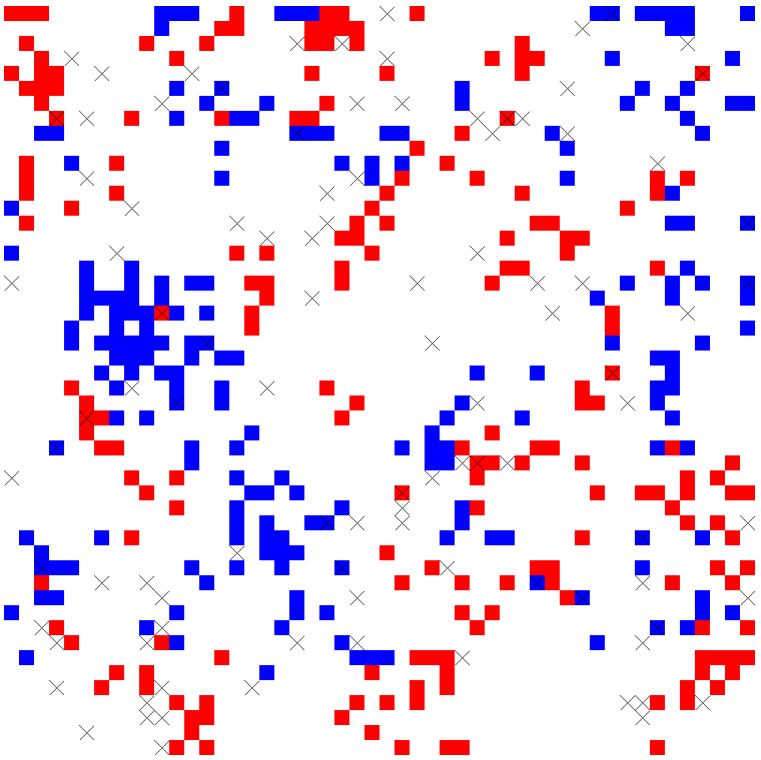
1. Essai initial



Pour ces paramètres :

```
int memorySize = 10 ;
int nbIterations = 1000000;
int cooldown = 0;
double errorPercent = 0.0;
```

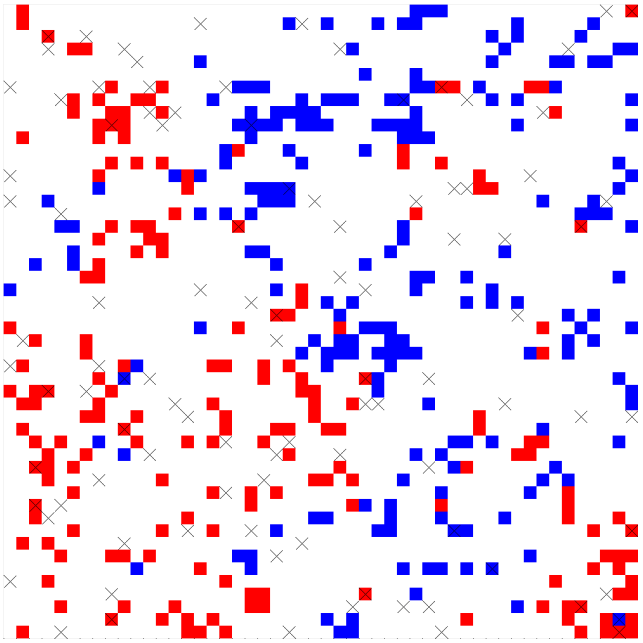
Avec une grille initiale ci-contre,



On obtient la grille ci-dessus.
On peut noter la formation de clusters, avec cependant du bruit à plusieurs endroits.

nb: Lorsque nous parlons de ce bruit dans cette partie, nous désignons des objets placés à l'intérieur d'un cluster d'objets d'un autre type.

2. Variation du nombre d'itérations

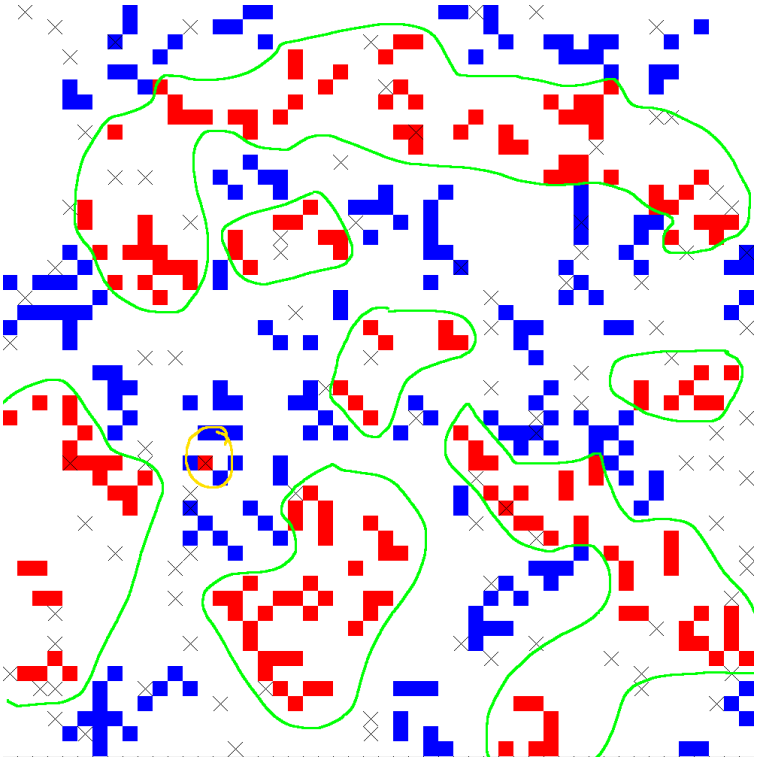


En multipliant le nombre d'itérations **par 10**, on obtient la grille ci-contre.

On observe toujours une formation de cluster avec un bruit semblable, ce qui laisse à penser que le problème ne converge pas.

3. Variation de la taille de la mémoire

Nous avons remarqué qu'il est possible de réduire presque totalement ce bruit en réduisant la taille de la mémoire à 1 :

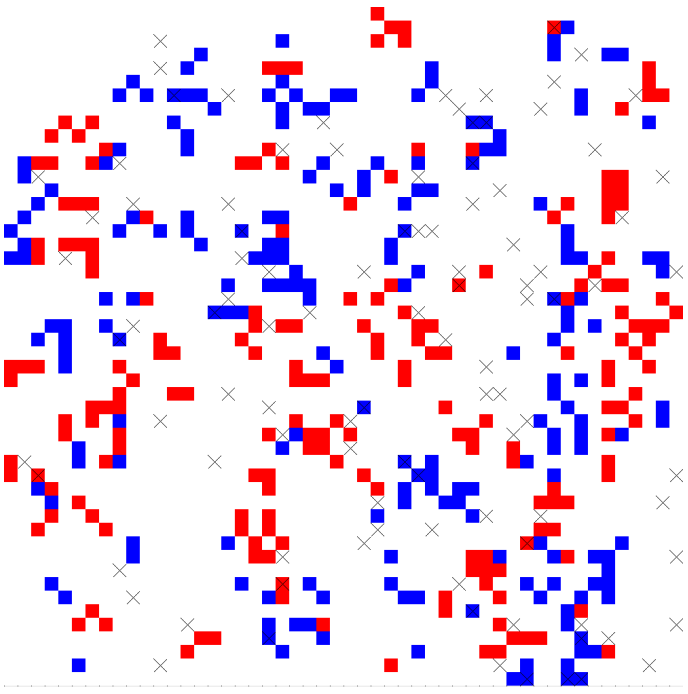


De cette manière, nous obtenons des clusters clairs et sans aucun bruit, à l'exception d'un objet rouge, qui était en cours de transport au moment final (entouré en jaune). Cependant, l'utilité de la mémoire devient moindre.

4. Ajout d'une erreur dans la perception

On peut maintenant ajouter un taux d'erreur, ce qui augmente logiquement le bruit dans les clusters, jusqu'à rendre le travail impossible :

Erreur = 10%



Erreur = 90%