Rapport de projet - Systèmes Multi-Agents

Tri sélectif

M1 Informatique - S2 2021

Enseignante : Salima HASSAS

Titouan Knockaert & Gaspard Goupy

Sommaire

Modélisation

1. Grid

La grille sert de base au modèle. Elle se compose d'un tableau à 2 dimensions d'objet Cases et d'une liste de Robots.

Nb: dans la version 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 ramasse un objet, celui-ci est toujours considéré comme étant sur la case. On ne peut donc pas avoir à la fois un robot portant un objet et un objet libre sur la même case.

Nb: dans la version 2, les cases possèdent aussi un objet phéromone.

3. Objets

```
public class Objet{
    char type;

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

Dans la première version, il existe deux types d'objets : **A** et **B**. Ceux-ci peuvent être déplacés par un robot.

Dans la deuxième version, il existe un troisième type d'objets : **C**. Ces objets nécessitent l'intervention de deux robots pour les déplacer.

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 contenant 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 : $f = (\#T\ in\ memory + \#not\ T * error) / memory\ size$
- 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 : $f = (\#T\ in\ memory + \#not\ T * error) / memory\ size$
- 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)

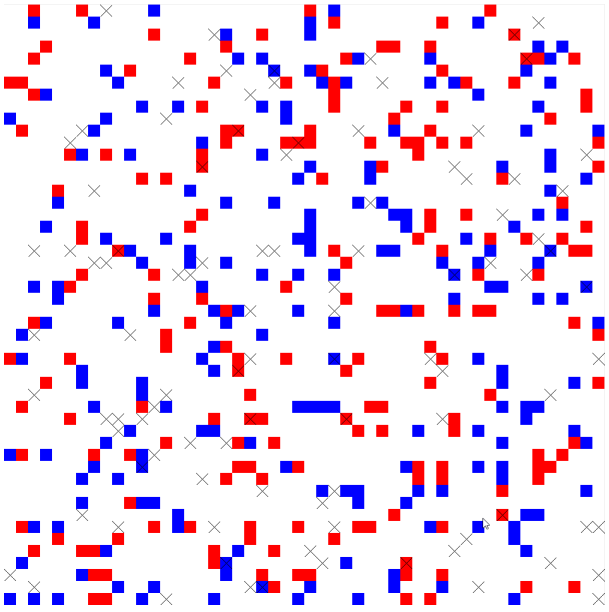
Résultats et observations

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.

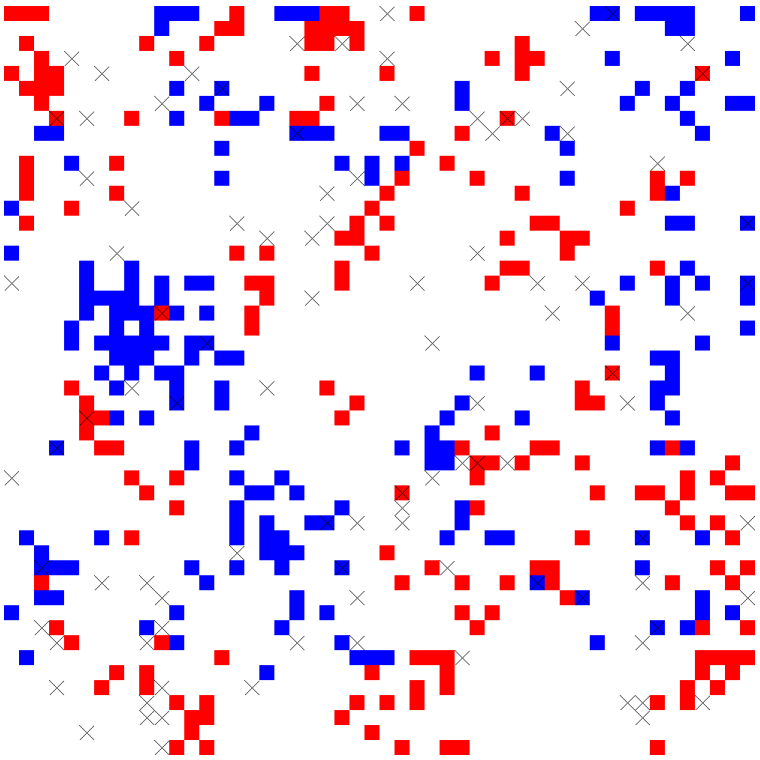
1. Essai initial



Pour ces paramètres :

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

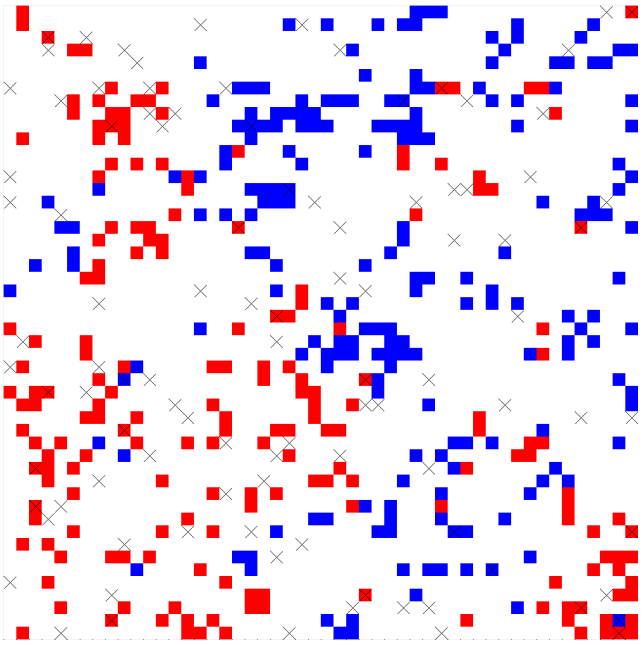
Avec une grille initiale ci-contre,



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

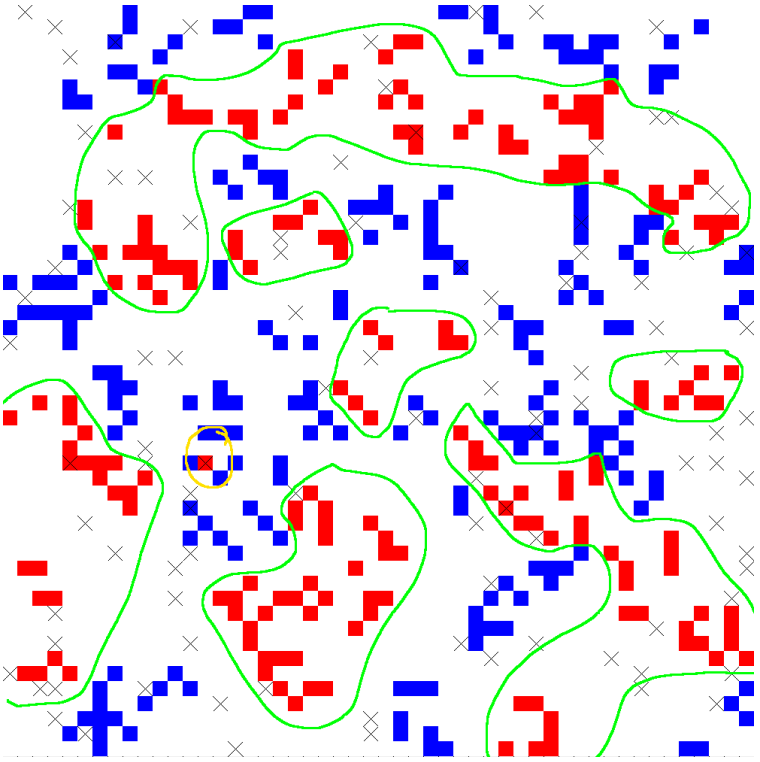


En multipliant le nombre d'itérations **par 10**, on obtient la grille ci-contre.

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 :

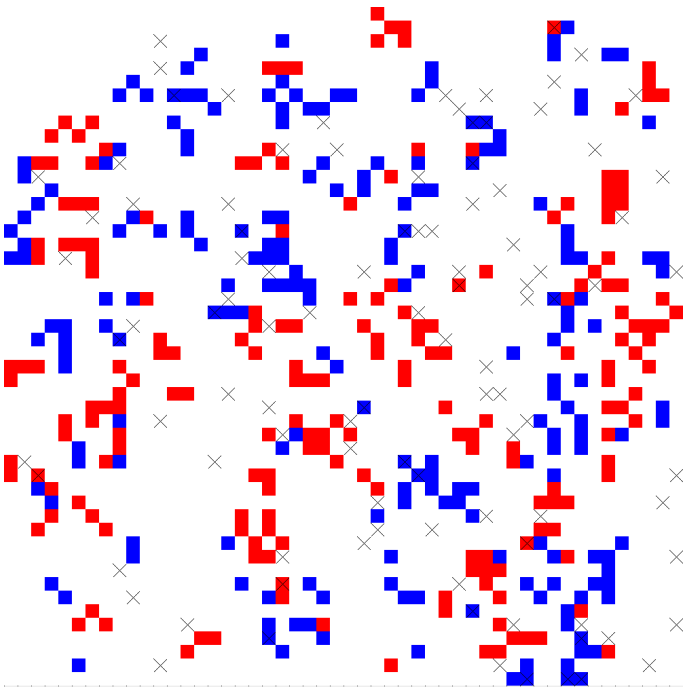


De cette manière, nous obtenons des clusters clairs et sans aucun bruit, à l'exception d'un objet rouge, qui était en cours de transport au moment final (entouré en jaune). Cependant, l'utilité de la mémoire devient moindre.

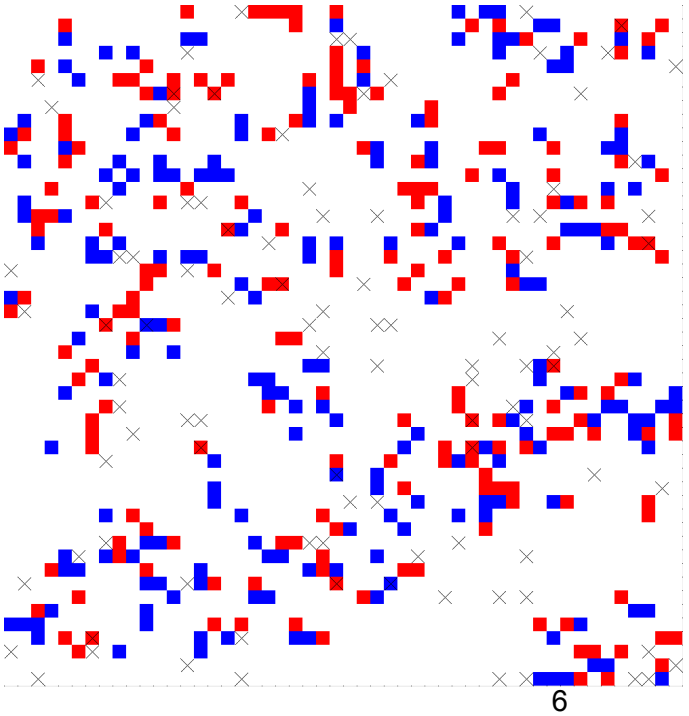
4. Ajout d'une erreur dans la perception

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%



Deuxième version

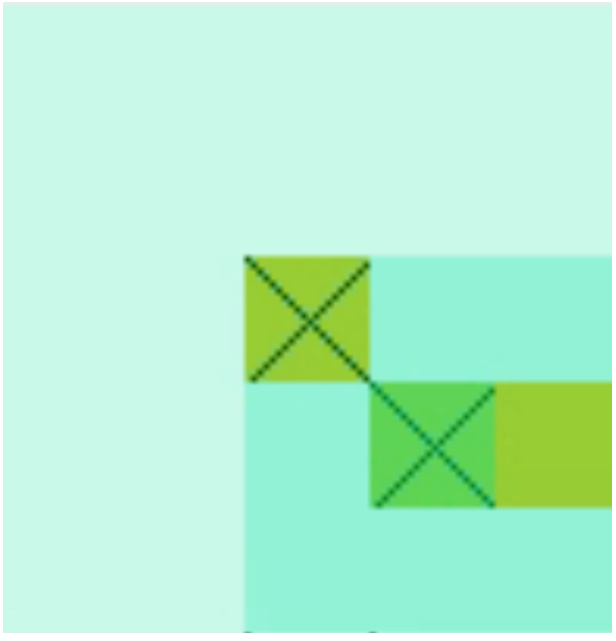
Dans la deuxième partie du projet, l'objectif était d'ajouter un nouveau type d'objet : les objets de type C. Ces objets nécessitent la collaboration de deux robots pour pouvoir être déplacés.

1. Mécanisme d'appel à l'aide

Quand un robot décide d'attraper un objet de type C, celui-ci arrête de se déplacer et doit demander de l'aide aux autres robots. Ainsi, nous avons implémenté un mécanisme d'émission de phéromone. Lorsqu'un phéromone est émis, il est diffusé récursivement sur les cases voisines. Son intensité diminue lors de sa diffusion : $0.75 * intensité\ précédente$. Si la nouvelle intensité est inférieure à 1, le phéromone s'estompera. Le robot est chargé de la valeur initiale de l'intensité, qui est fixée à 2.5. *Pour plus de détails, voir la méthode Grid.difusePheromone(Case).*

Lorsqu'un robot sent un phéromone, via sa perception, il modifie son comportement par sa façon de générer ses mouvements (méthode Robot.generateMove()) : il sent alors les cases adjacentes pour se diriger vers celle ayant l'intensité de phéromone la plus élevée.

A chaque itération de la simulation, l'intensité des phéromones sur chaque case est réduite afin de simuler un processus de dissipation avec le temps. *Pour plus de détails, voir la méthode Grid.updateCase().*



Ici, les cases jaunes représentent des objets de type C. Les croix représentent les robots. Les cases vertes représentent les phéromones.

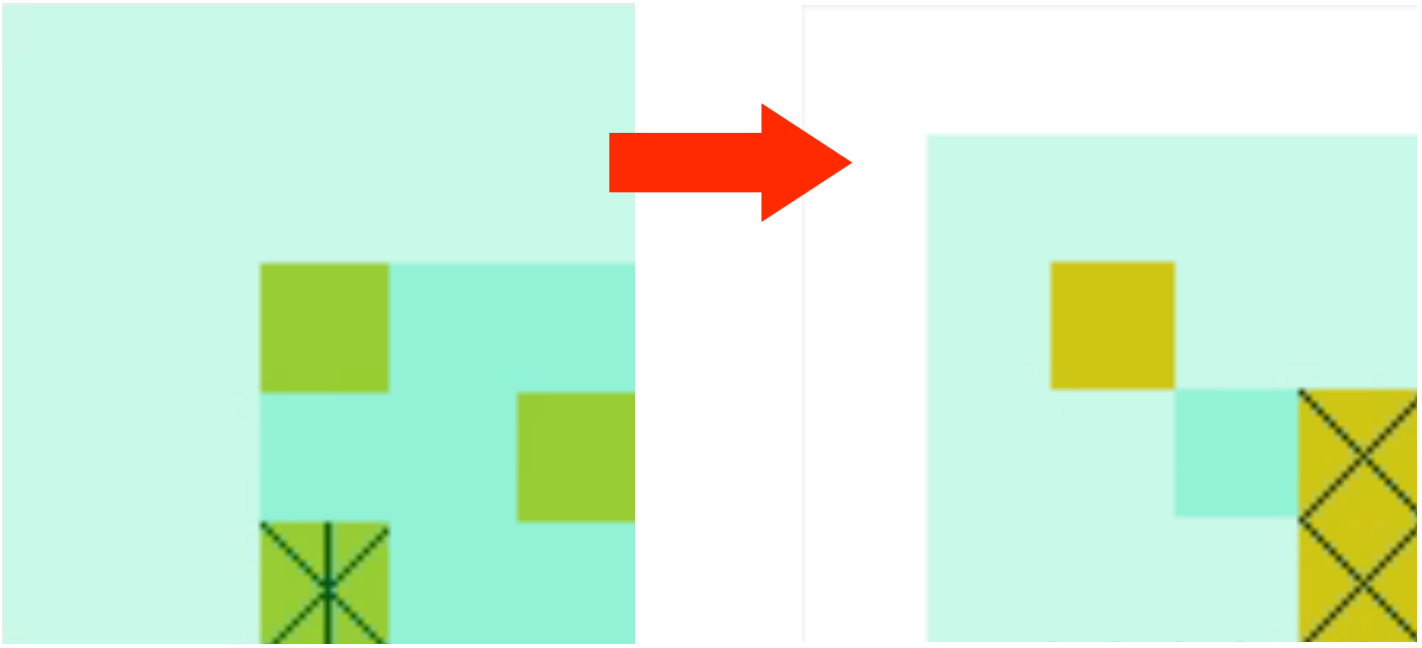
Le robot en bas à droite émet un phéromone qui va attirer le robot au milieu.

2. Mécanisme de collaboration - Fusion

Une fois qu'un robot appelant à l'aide a pour voisin un autre robot qui est venu l'aider, les deux fusionnent pour former un MegaRobot. Ce robot est un consensus des deux robots qui peut maintenant déplacer l'objet C. Il pourra alors se déplacer avec l'objet, et agira de la même manière qu'un robot classique, à l'exception près qu'il ne sera pas attiré par les phéromones. Une fois l'objet lâché, il défusionnera et chaque robot reprendra son fonctionnement normal.

Lors de la fusion les deux robots sont mis dans un état inactif et sont retirés de la grille et stockés dans le mégarobot, à la défusion le mégarobot replace les robots sur la grille et se place dans un état inactif et est retiré de la grille.

Pour plus de détails, voir les méthodes Robot.fusion et MegaRobot.defusion.



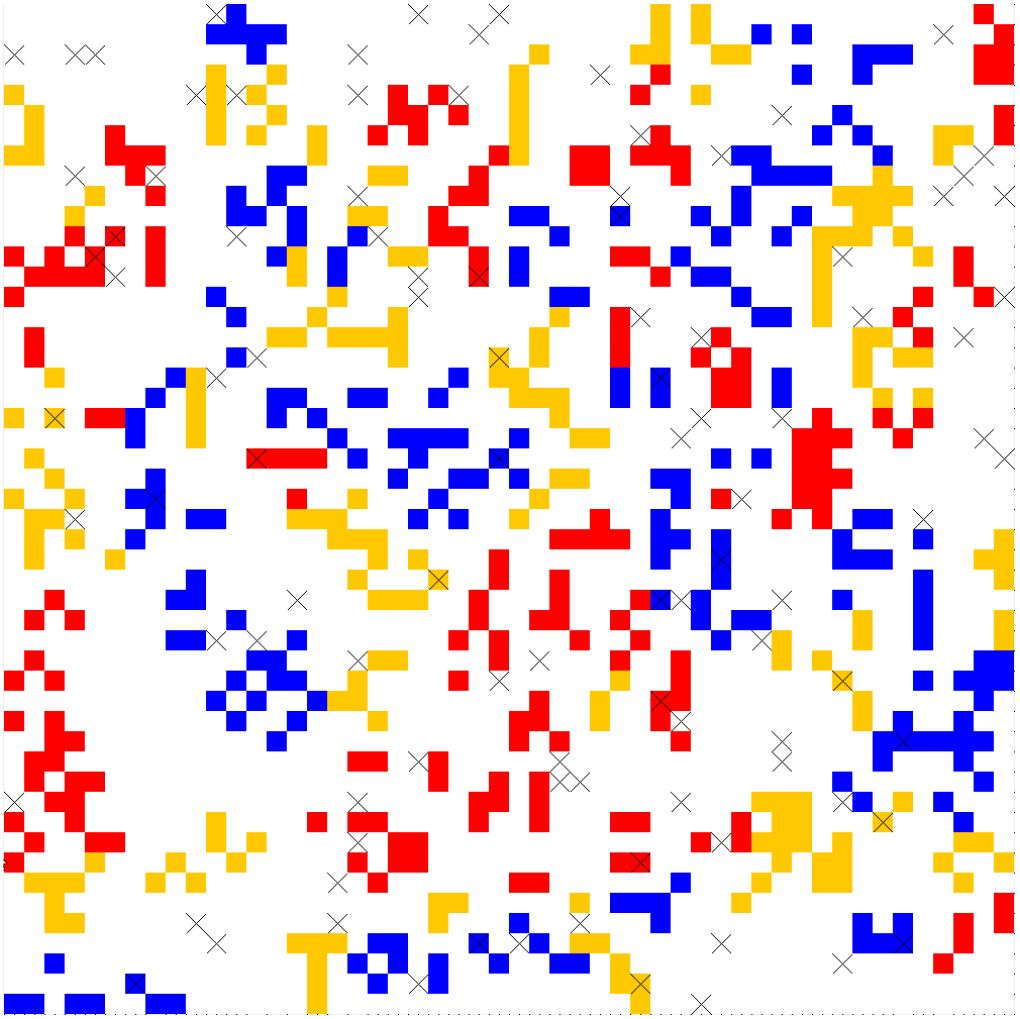
Jaune : Objet C
X : Robot
X + | = MegaRobot

Sur la première image, nos deux robots ont fusionné et ont commencé à déplacer l'objet.

Sur la deuxième image, les deux robots ont défusionné après avoir lâché l'objet.

3. Résultats obtenus

En gardant les paramètres optimaux obtenus à la fin de la partie précédente et en ajoutant 200 objets de type C :



On obtient donc des résultats identiques à ceux de la première partie : avec une mémoire de 1 on a des clusters sans bruit.