Rapport de TP de SMA

Réalisé par:

Khalid OUHMAID & Med Amine ASRI

Encadré par:

Prof. Salima HASSAS

Introduction

Dans le cadre du cours de SMA, nous avons implémenté l'algorithme de tri collectif.

Dans ce système, plusieurs agents sont placés sur une grille et sont chargés de trier des blocs selon leur type (**A** ou **B**). L'objectif final est d'obtenir deux types de colonies distincts : des colonies de blocs A et des colonies de blocs B. Ainsi, les agents sont censés éviter au maximum les mélanges de blocs au sein d'une même colonie.

Dans un premier temps, nous avons procédé à l'implémentation même du système multi-agents que nous avons réalisé en Java.

Une fois le système implémenté, nous analyserons successivement l'influence des différents paramètres sur les résultats obtenus finaux obtenus en nous basant principalement les critères suivants :

- Nombre d'objets A et B dans la grille
- Taille de la grille
- Nombre d'agents dans l'environnement.
- Memoir des agents
- Le taux d'erreur dans la perception

Nous avons programe notre system base sur une programmation oriente objets

Notre programmation d'un agent consiste en une boucle perception (), action(). La perception, permet à l'agent de récupérer les informations de l'environnement et l'action, dans le cas du sujet est soit : le déplacement (aléatoire) d'un pas, la prise (suivant probabilité) d'un objet ou le dépôt (probabiliste) d'un objet et ça illustre dans le code suivant.

```
public static void main(String[] args) {
    //Environement environement =new Environement(50,50, 100, 100,1);
   Environement environement =new Environement( sizeM: 20, sizeN: 20,  numberOfAgents: 10,  numberOfObject: 50, agentMoveIndex: 1);

    System.out.println(environement);
    //environement.pushLeft();
    AppWindow f1;
    f1= new AppWindow(environement);
    f1.setVisible(true);

    for(int i = 0; i<100000; i++){
        List<Integer> range= IntStream.range(0, environement.getAgentsList().size()).boxed().collect(Collectors.toList());

        ArrayList<Agent> agents = environement.getAgentsList();
        for(int j = 0; j< environement.getNumberOfAgents(); j++) {
            int alea = (int) (Math.random() * range.size());

           environement.getAgentsList().get(range.get(alea)).perception();
            environement.getAgentsList().get(range.get(alea)).action();

            range.remove(alea);
        }
        environement.notifyView();
        try {
            //Thread.sleep(1000);
```

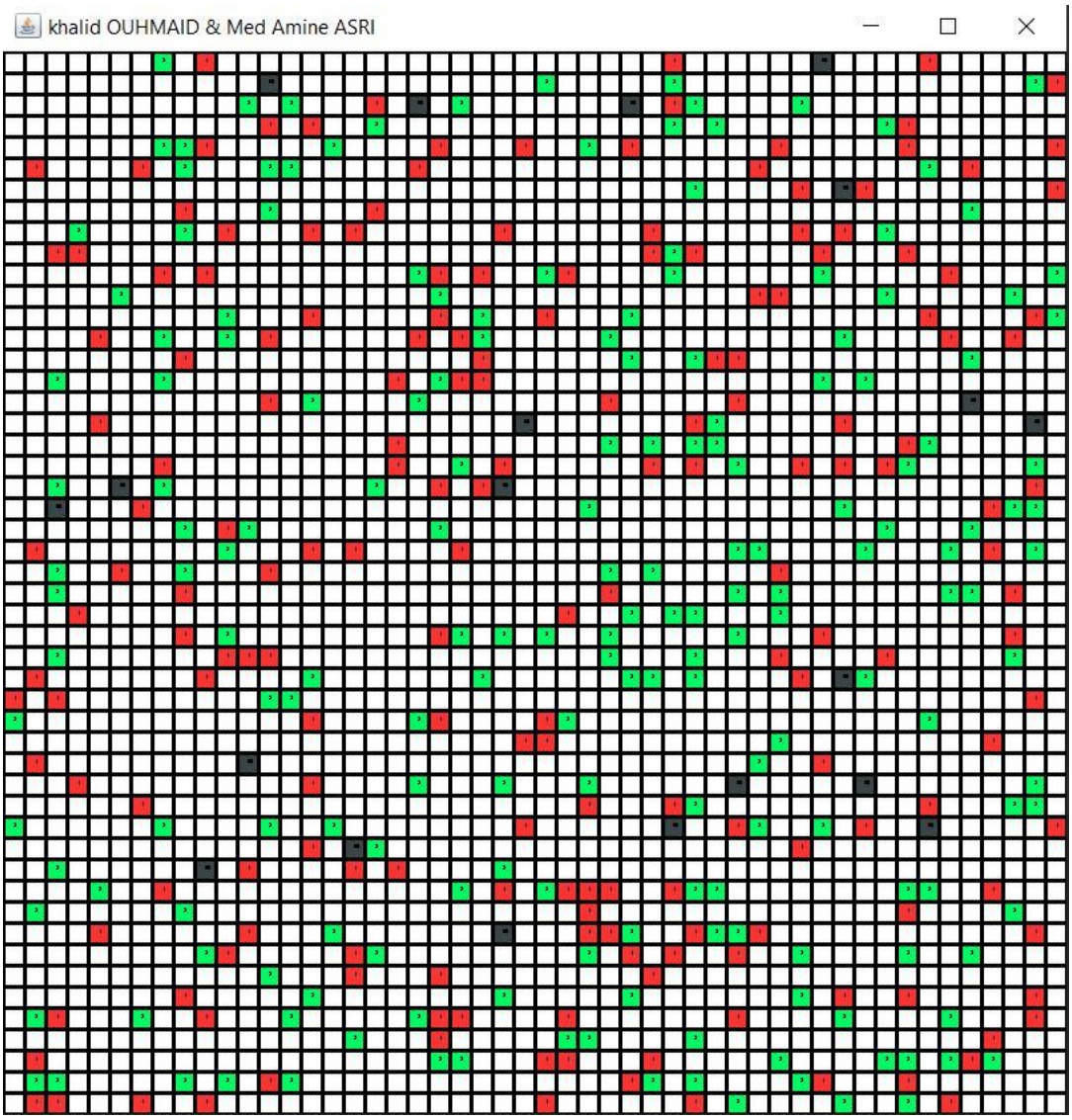
1 : Représente un Objet de type A

2 : Représente un Objet de type B

99 : Représente un Agent

Une grille 50x50, i=1, 20 agents, k+=0,1, k-=0,3, 200 objets de chaque type A ou B

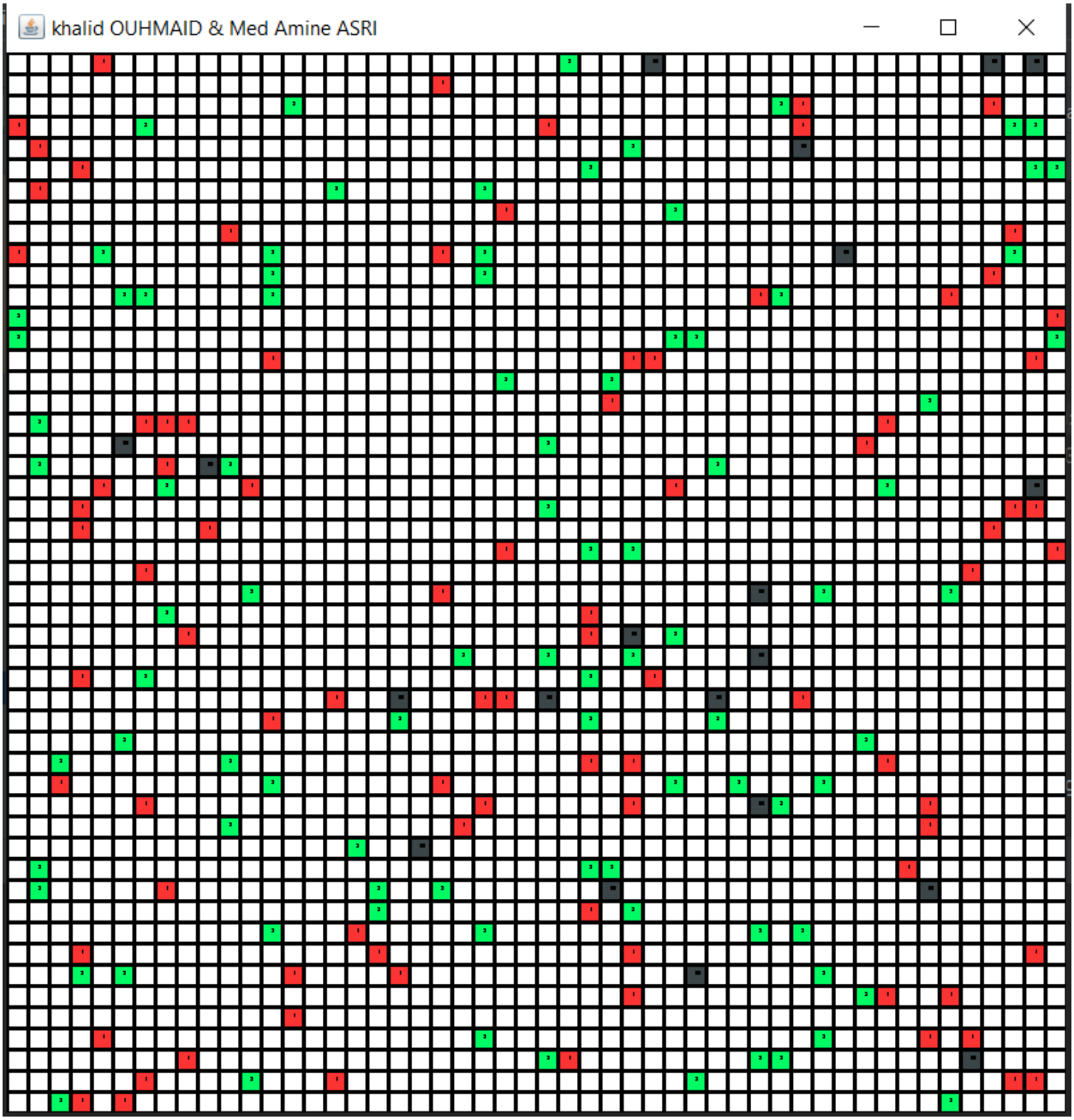
```
Environement environement =new Environement( sizeM: 50, sizeN: 50,  numberOfAgents: 20,  numberOfObject: 200, agentMoveIndex: 1);
```



Nombre d’objets dans la grille

Observations après plusieurs executions

Faisons tout d’abord varier le nombre d’objets présents dans l’environnement. Nous garderons dans un premier temps 50% d’objets A et 50% d’objets B, ce qui nous permet d’obtenir le schéma suivants :



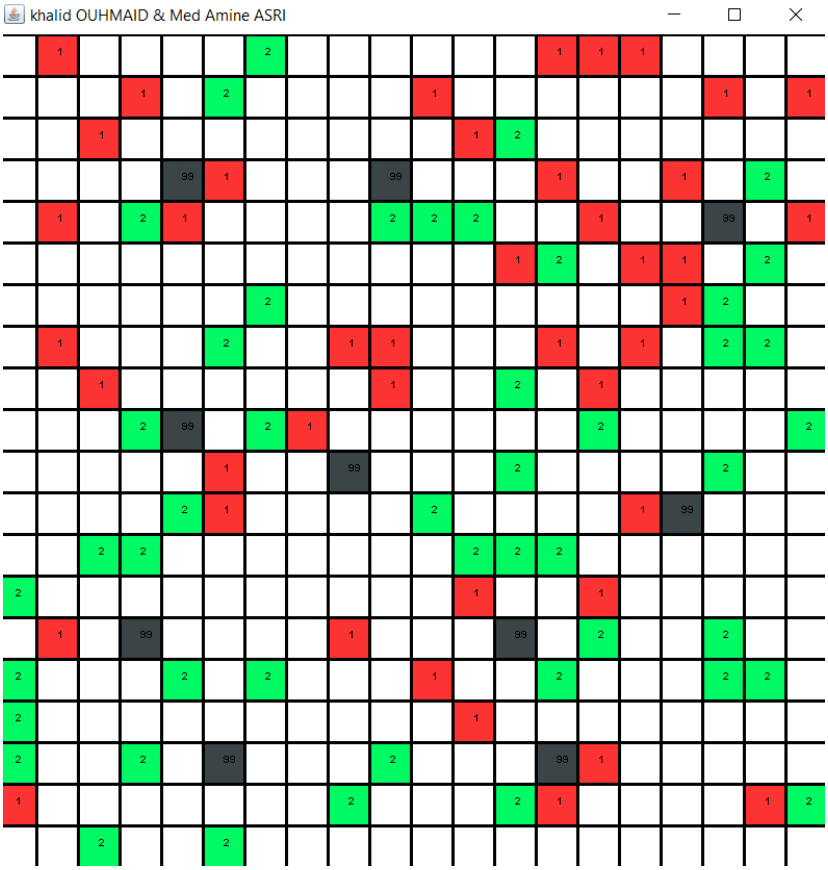
Analyse

Ajouter des blocs à la grille augmente la proportion d'erreurs de tri (c'est-à-dire le nombre de blocs A voisins de blocs B). Ceci est une conséquence logique : en effet, plus il y a de blocs sur la grille, plus un bloc A a de chances de se retrouver à côté d'un autre bloc, potentiellement de type B.

Taille de la grille

Observations après plusieurs executions

Intéressons-nous maintenant aux résultats obtenus lorsque nous faisons varier la taille de la grille sur laquelle évoluent les agents. En simulant l'exécution avec des grilles de tailles 20x20 (blocs recouvrant 50% de la grille), 20x20 (blocs recouvrant 50% de la grille) et 60x60 (blocs recouvrant 5,56% de la grille), nous avons généré les graphes suivants :



Nous remarquons que le nombre de colonies augmente avec la taille de la grille, ce qui est logique puisque plus l'environnement est vaste, plus les blocs sont libres d'être éparpillés et éloignés les uns des autres. De la même manière, comme il y a plus de colonies, ces dernières sont de plus petite taille. L'évolution du nombre et de la taille des colonies semble pouvoir être approchée par une fonction logarithmique.

Analyse

La taille de la grille utilisée pour représenter l'environnement impacte donc les colonies formées par les agents. En effet, plus la grille sera de petite taille, moins il y aura de colonies et plus elles comporteront de blocs. Par ailleurs, si nous réduisons trop les dimensions de la grille par rapport au nombre de blocs, les erreurs de voisinage augmenteront significativement, conséquence du manque de place dans l'environnement.

Nombre d'agents

Observations après plusieurs executions

Étudions maintenant l'influence du nombre d'agents utilisés dans l'environnement sur les résultats obtenus. Nous considérerons successivement l'emploi de 5, 20 puis 100 agents.

Tout d'abord, nous notons que plus le nombre d'agents est important, moins il y a de blocs posés en fin de simulation. Ce résultat s'explique par le fait que plus il y a d'agents, plus les blocs sont susceptibles d'être soulevés à un instant t. L'évolution de la proportion de blocs posés dans l'environnement en fonction de la quantité d'agents peut être approximée par une fonction linéaire

Analyse

À la finalité, même si accroître le nombre d'agents permet à première vue de réduire les erreurs de voisinage, il convient de nuancer ce résultat par le fait que le nombre de blocs posés, et donc analysables en fin de simulation, décroît avec la quantité d'agents. Il est donc difficile d'affirmer que les agents sont plus efficaces lorsqu'ils sont plus nombreux simplement avec ces observations.

Mémoire des agents

Observations après plusieurs executions

Faisons à présent varier la taille de la mémoire des agents. Nous prendrons pour nos tests des mémoires de taille 5, 15 et 50.

Dans un premier temps, nous remarquons que plus les agents ont une mémoire de taille importante, plus le nombre de colonies formées augmente. La conséquence directe de cette augmentation est une baisse du nombre moyen de blocs par colonies,

Analyse

Une mémoire de faible capacité permet de réduire à la fois les erreurs de voisinage et le nombre de colonies dans l'environnement. A l'inverse, des agents dotées d'une mémoire de grande capacité formeront davantage de colonies et produiront plus d'erreurs de voisinage.

En conséquence, pour répondre à notre problématique, il vaut mieux utiliser une mémoire de **faible capacité** pour les agents.

Erreur

Observations après plusieurs executions

Nous allons maintenant ajouter des erreurs dans la mémoire des agents. A chaque prise de bloc, l'agent aura alors une certaine probabilité de se tromper et d'ajouter le mauvais bloc à sa mémoire. Pour nos tests, nous considérerons des probabilités de 0, 0.1, 0.2, 0.3, 0.5, et 0.9.

Analyse

Ajouter la notion d'erreur ne s'est pas révélé bénéfique pour notre algorithme. En effet, nous avons considérablement augmenté le pourcentage d'erreurs de tri tout en créant plus de colonies. Il vaut donc mieux laisser l'erreur à **0** pour éviter une dégradation des résultats.

Conclusion

Pour conclure, l'analyse des différents paramètres nous a permis de tirer plusieurs conjectures quant aux valeurs à privilégier pour obtenir de bons résultats. Le tableau ci-dessous récapitule les observations faites paramètre par paramètre.

Nombre de blocs A	**Nombre de blocs B**	**Taille de la grille**	**Nombre d'agents**	**Taille de la mémoire des agents**	**Nombre de mouvements successifs des agents**	**K +**	**K -**	**Erreur**
100	Égal au nombre de blocs A	Telle qu'il y ait environ 12,5% de blocs	Inférieur à un dixième du nombre total de blocs	Petite (5 ou 15)	1	Proche de 0 (0,1)	0,3	0

D'un point de vue personnel, ce TP nous a permis d'approfondir nos connaissances en intelligence artificielle tout en implémentant un nouveau système multi-agents. Nous avons également pu voir, grâce à l'analyse des résultats obtenus lors des simulations, l'influence que pouvaient avoir les changements, en apparence anodins, des paramètres en entrée.