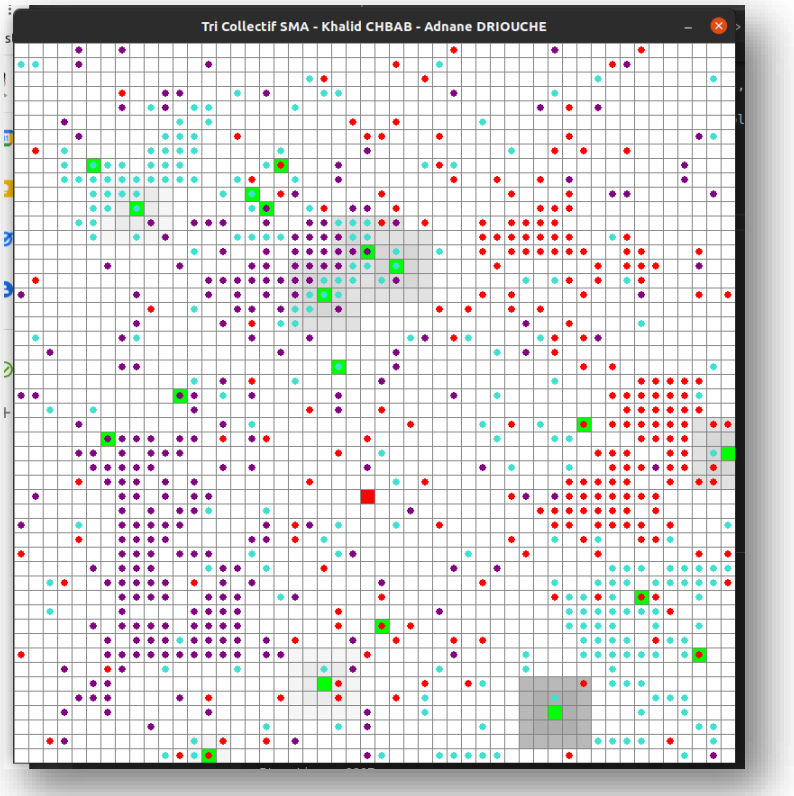
TRI COLLECTIF – VERSION 2

SMA



A l'intention de : Mme Salima HASSAS
Réalisé par : CHBAB Khalid & DRIOUCHE Adnane

I. Introduction

Après la réalisation de la première version du TP, qui consiste à programmer à l'aide de règles de comportements simples, le comportement de tri collectif de deux types d'objets :

- Type 'A'
- Type 'B'

Dans cette version, on a ajouté un 3ème objet de type 'C' qui nécessite la collaboration de 2 robots pour être portés.

Dans ce cas, quand un robot tombe sur ce type d'objet dans son environnement, il émet un signal sous forme de phéromone qui se code par une information propagée dans son voisinage (8 cases autour) sur une distance de diffusion du signal ***ds***. Parfois, nous nous sommes trouvés dans un cas où tous les agents envoient du signal et attendent de l'aide, pour éviter cet interblocage, nous avons doté nos agents d'un comportement qui consiste à laisser l'objet à sa place après l'envoi de 10 phéromones (10 itérations).

Pour réaliser ce projet, nous nous sommes basé sur un tri collectif qui va suivre les règles suivantes :

- L'agent collecte des informations sur l'environnement dans les 8 directions .
- L'agent se déplace aléatoirement à chaque itération dans les 8 directions.
- L'agent peut prendre ou déposer selon des probabilités.

$$P_{prise} = (k^+ / (k^+ + f))^2 \qquad P_{dépôt} = (f/(k^- + f))^2$$

- l'agent envoie signal lorsqu'il rencontre l'objet 'C'.
- L'agent le plus proche au phéromone aide à déplacer l'objet 'C'.

k + et k - des constantes et f représentant la proportion d'un objet de même type A ou B dans la planette (voisinage de l'agent). Le voisinage de l'agent est défini par les cases atteignables en 1 pas de temps par l'agent dans les 8 directions. l'agent est doté d'une mémoire de taille variable représentant les objets déjà rencontrés sur les N derniers pas.

II. Méthodologie

i. Définition des Agents :

Dans notre simulation, l'agent est représenté par un carré vert qui se déplace dans les 8 directions sur la grille. Les agents qui entrent en jeu sont tous des objets de la classe ***Agent***.

Pour réaliser la deuxième version, nous avons doté nos agents avec quelques fonctionnalités :

- ***Send_phero_signal()*** : permet de déclencher l'attribut ***waiting*** lorsque l'agent tombe sur un objet de type 'C'.
- ***Move_to_phero()*** : permet à l'agent de se déplacer vers le plus proche phéromone.
- ***Help()*** : fait appel à l'aide lorsque l'agent trouve un objet 'C'.

Figure 1 Classe Agent - Version 2

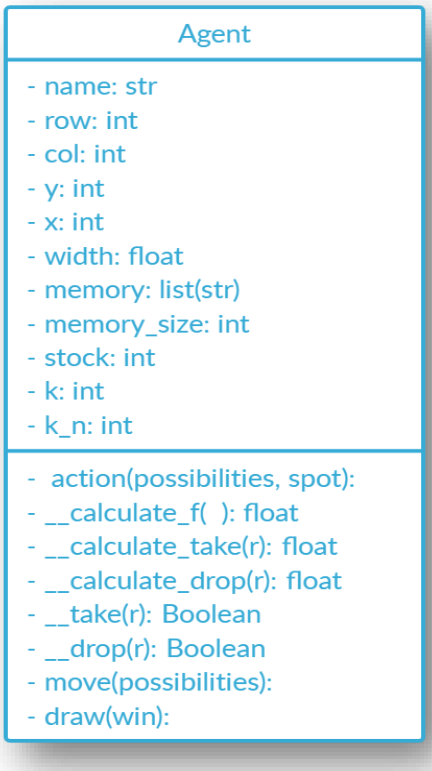


Figure 2 Classe Agent - Version 1

ii. Comportements des agents :

Le programme d'un agent consiste en une boucle perception (), action().

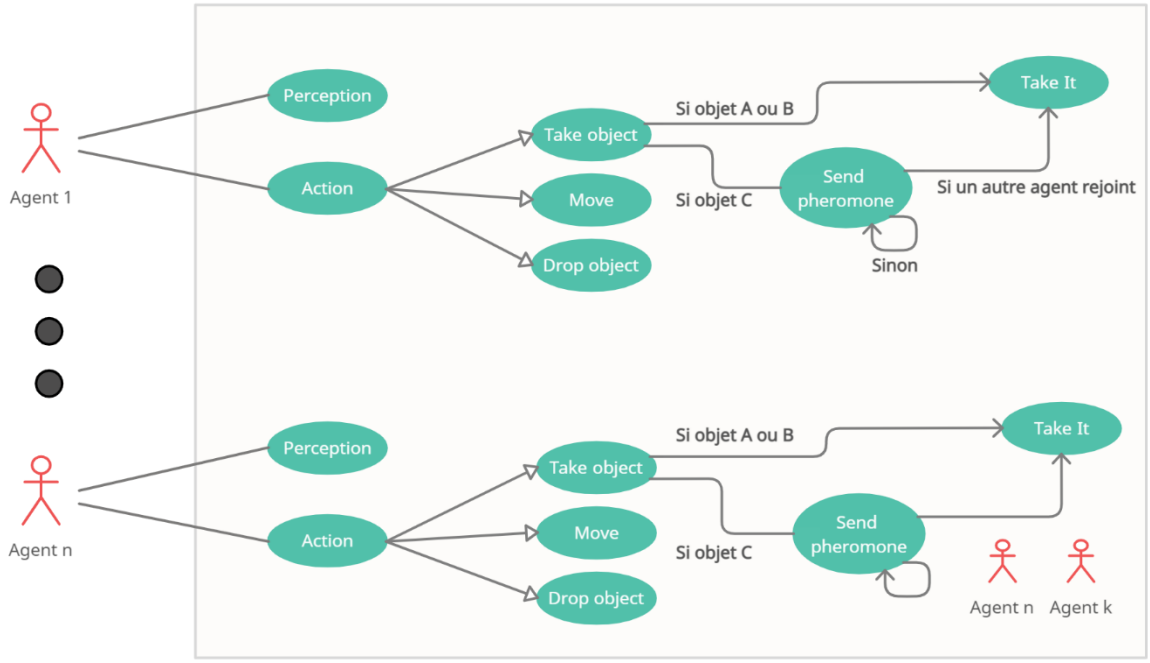


Figure 3 Diagramme de cas d'utilisation

Dans cette version :

- La perception, permet à l'agent de récupérer les informations de l'environnement.
- L'action, dans le cas du sujet est soit :

  - le déplacement (Move) d'un pas.
  - la prise d'objet (Take object) : On a deux cas, soit l'agent rencontre un objet du type A ou B (Version 1 du TP), soit un objet du type C. Dans ce cas, il est obligé d'envoyer des signals à l'aide de sa phéromone pour qu'il puisse recevoir l'aide et par la suite prendre l'objet. Afin d'éviter les cas d'interblocage, l'agent va envoyer la phéromone juste pendant une durée t que nous avons défini.
  - le dépôt (drop object) d'un objet.

III. Implémentation de la solution

i. Structuration du code :

Cette simulation est construite au-dessus de la bibliothèque Pygame, elle utilise le concepte POO pour représenter le monde et est développée en utilisant python. Aucun multithreading n'est implémenté et les agents se relaient pour déplacer chaque iteration.

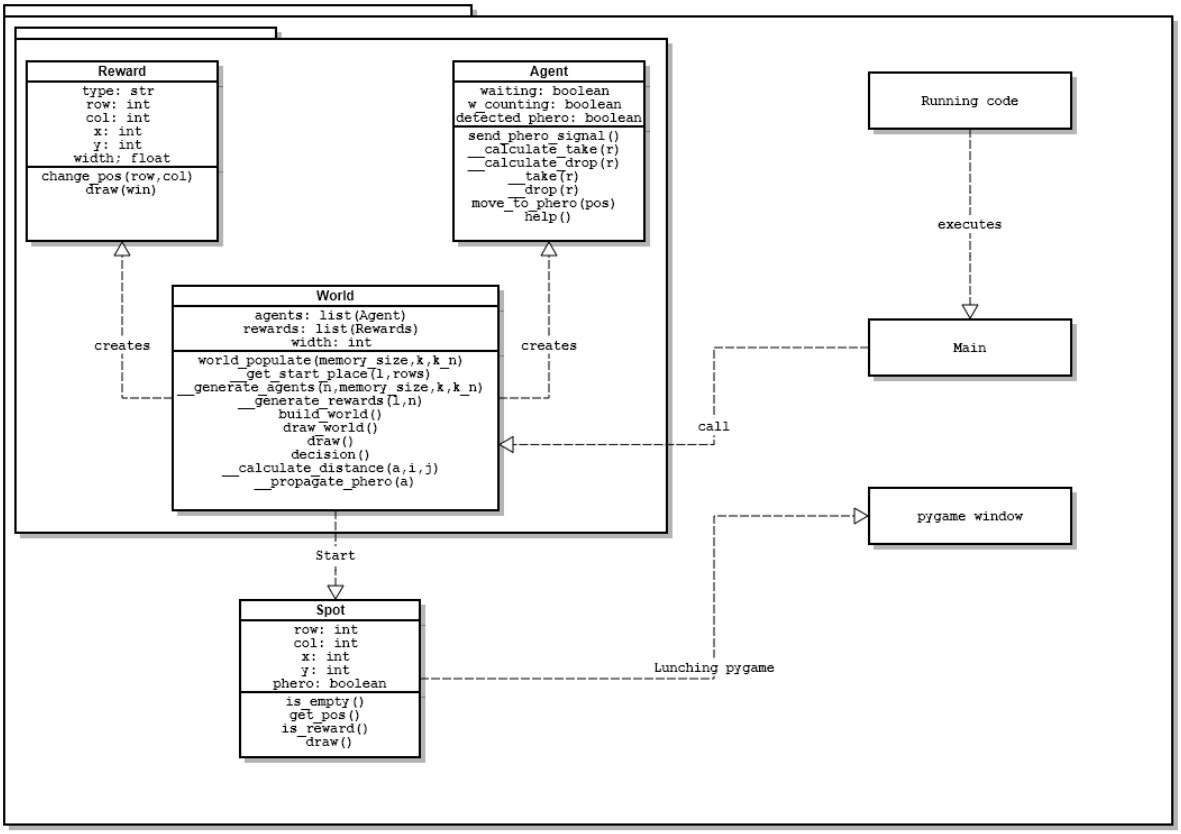


Figure 4 Structuration du code

ii. Les Classes et leurs relations :

Afin de réaliser la 2ème version, nous avons agi sur notre diagramme de classe et précisément les classes ***Agent***, ***Spot*** et ***Reward*** :

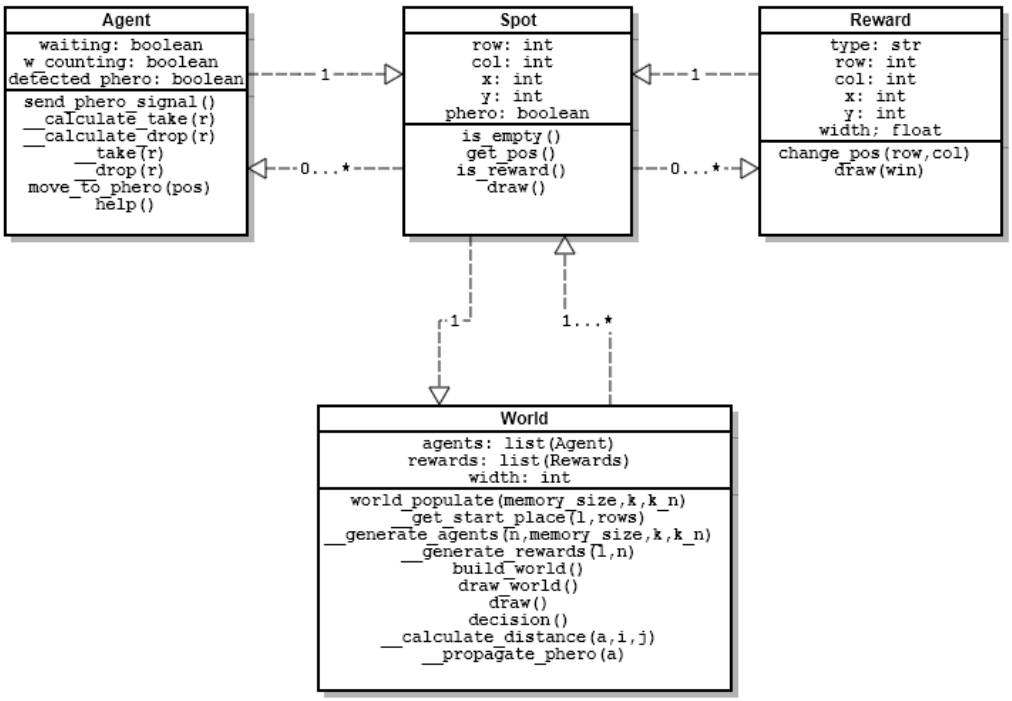


Figure 5 Diagramme de classe

- **Agent** : la classe d'agent contient tous les opérations liées à l'agent notamment :
  - Initialisation des propriétés d'agent.
  - Action on se basent sur les informations envoyées par l'environnement.
  - Déplacement aléatoire en suivant les possibilitées fourni par l'environnement ou en suivant le phéromone d'un agent.
  - Demande de l'aide (quand il trouve un objet C).
  - Control d'état interne comme la mémoire.
- **World** : la classe World ou simulation englobe l'environnement et affichage(GUI)
  - Initialisation des propriétés d'agent.
  - Génération des agents, objets, grille et les cellules.
  - Réaliser la perception et envois aux agents.
  - Mise à jour de GUI par itération.
- **Reward** : contient l'abstraction de caractéristiques d'un objet.
- **Spot** : le bloc minimal de construction de terrain de simulation/world.
- **Main** : point d'entrée de programme et exécution de simulation.

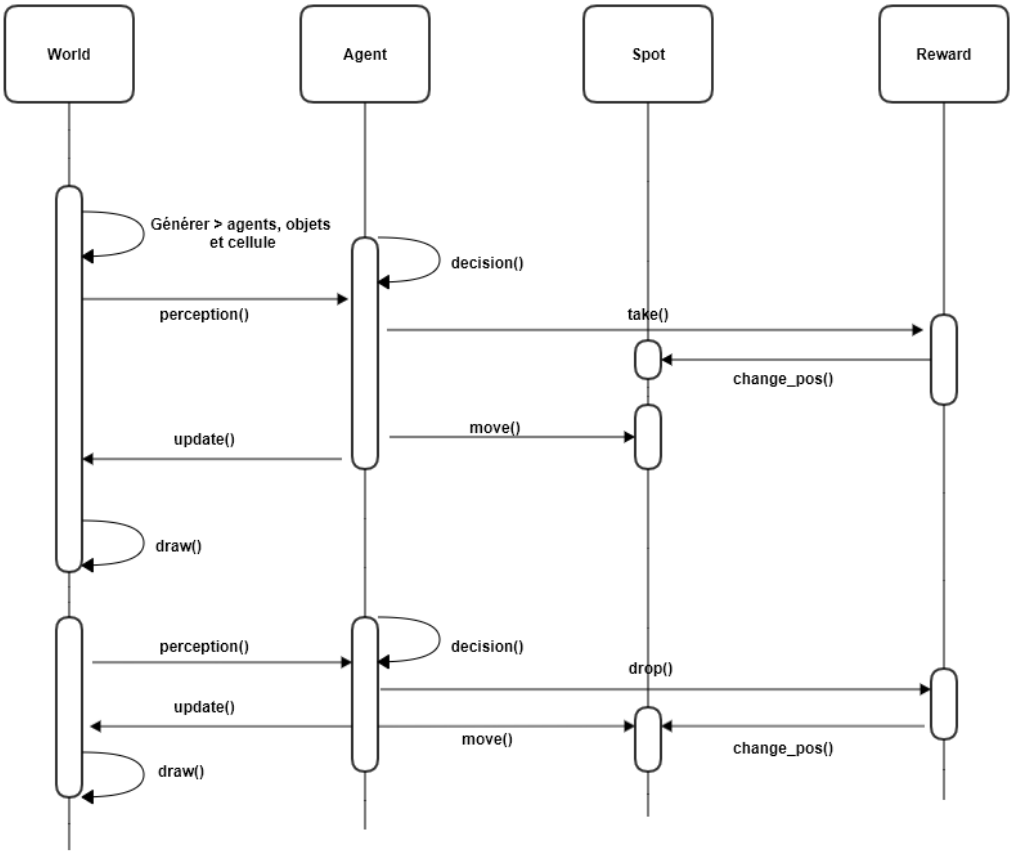


Figure 6 Diagramme de conséquence

La simulation commence par initialisation du monde et GUI par instanciation de class World, puis la génération des entités comme les agents, les objets et les cellules. Une fois le monde initialisé, la simulation commence par récupérer des informations pour chaque agent dans les 8 directions qui les entourent.

Le calcul des possibilités d'agent se fait par interrogation de 8 cellules, on leur demande si elles sont vides ou pas. l'agent se base sur ces informations pour alimenter sa mémoire et calcule des probabilités de dépôt et prise.

La décision de chaque agent va dépendre de type d'objet rencontré :

- Si : objet de type A ou B ➔ Il va le prendre.
- Sinon : envoyer le phéromone ➔ attendre l'aide d'un autre agent.

On note que :

- Si un agent porte un objet sa couleur change.
- L'agent se déplace et sa position sur la grille change.
- La phéromone est représentée par une grille de taille 4*4, qui disparaît avec le temps.

IV. Résultats d'exécution

i. Version 1:

Pour l'exécution, nous avons pris comme exemple :

Un environnement sous forme de grille de Taille = (50,50) qui contient :

- 20 agents : représenté par des carrés verre(vide) rouge(porte un objet)
- 400 objets : 200 de type A de couleur VIOLETTE et 200 type B de couleur ROUGE représenté par des petites cercles.

Chaque agent se déplace aléatoirement dans l'environnement et il prend ou dépose l'objet en se basant sur ces probabilités suivantes :

$P_{prise} = (k^+ / (k^+ + f))^2$ $P_{dépôt} = (f/(k^-+f))^2$

Avec : k+ et k- des constantes et f représentant la proportion d'objet de même type A ou B dans l'environnement immédiat.

Les captures d'écran ci-dessous sont prise respectivement après : 357, 11718, 100783 et 423897 itérations (1 minute, 15 minutes, 255 minutes et 9 570 minutes).

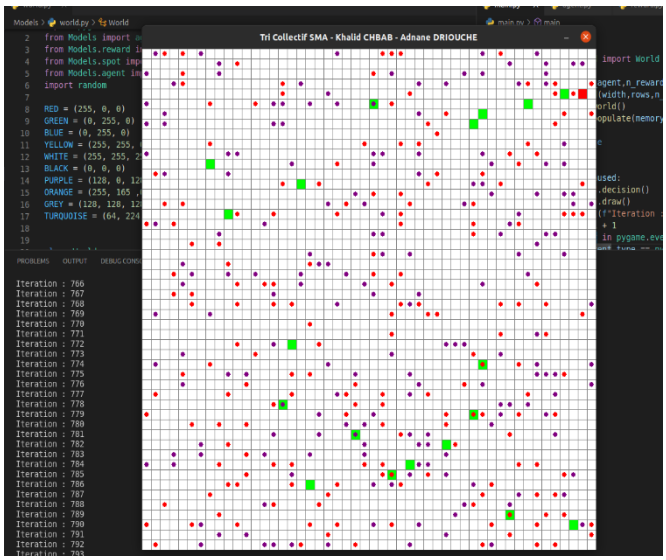


Figure 7 une minutes après l'exécution

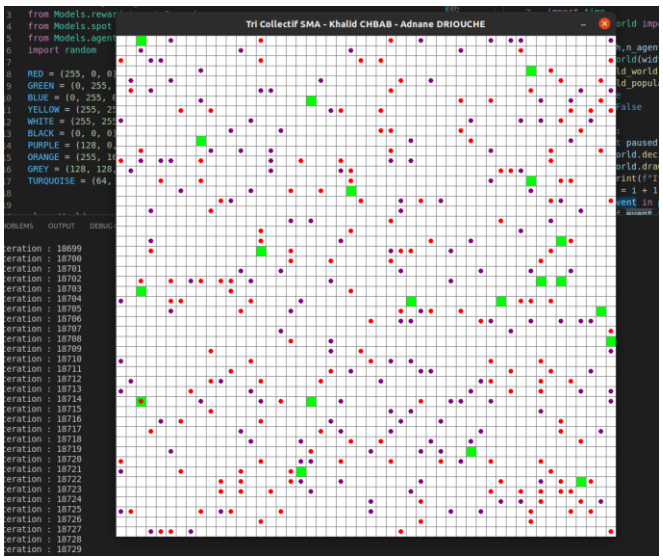


Figure 8 15 minutes après l'exécution

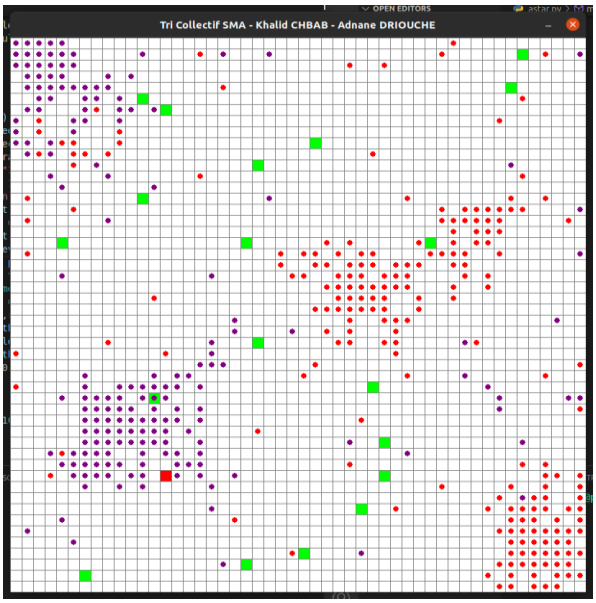


Figure 11 255 minutes après l'exécution

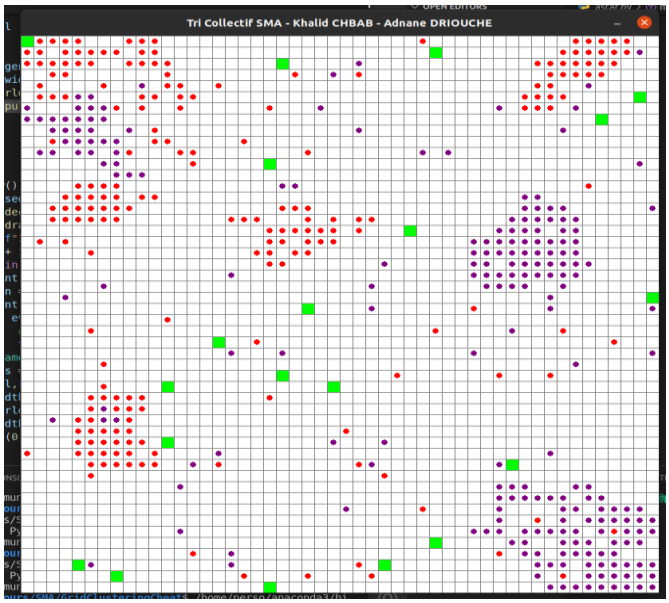


Figure 10 570 minutes après deuxième exécution

Étant donné que les agents n'ont aucune intelligence sur l'endroit où mettre ou prendre des objets, la convergence vers un monde semi-cluster ou un monde clusterisé peut prendre beaucoup de temps. Il sera intéressant de voir si le changement de variables comme la capacité de mémoire ou le mécanisme de l'environnement peut améliorer la simulation.

ii. Version 2 :

On a gardé le même environnement sous forme de grille de Taille = (50,50) qui contient :

- 20 agents : représenté par des carrés verre(vide) rouge(porte un objet)
- 600 objets représenté par des petites cercles:
  - 200 de type A de couleur VIOLETTE
  - 200 type B de couleur ROUGE
  - 200 de type C de couleur TURQUOISE

Les captures d'écran ci-dessous sont prise respectivement après : 16718, 360783 et 723897 itérations (15 minutes, 300 minutes et 750 minutes).

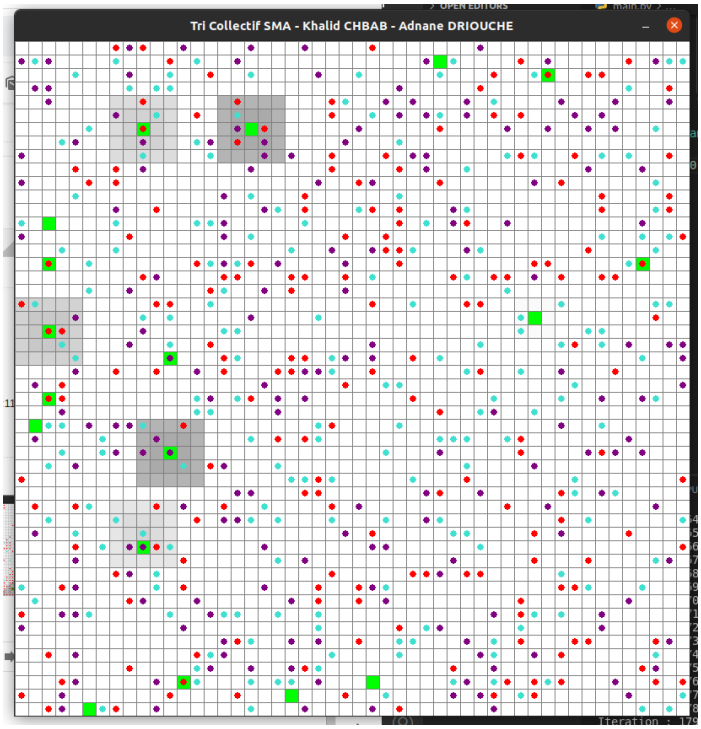


Figure 13 15 minutes après l'exécution

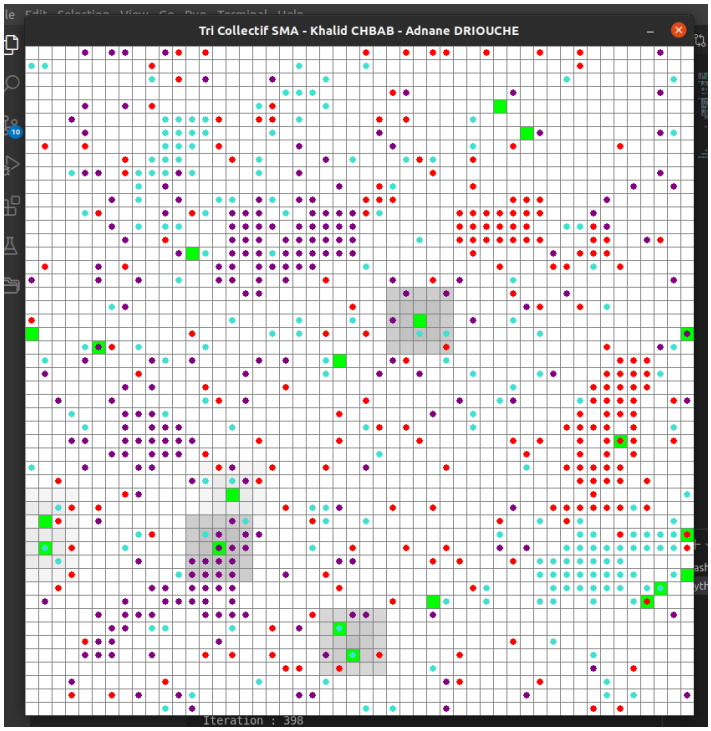


Figure 12 300 minutes après l'exécution

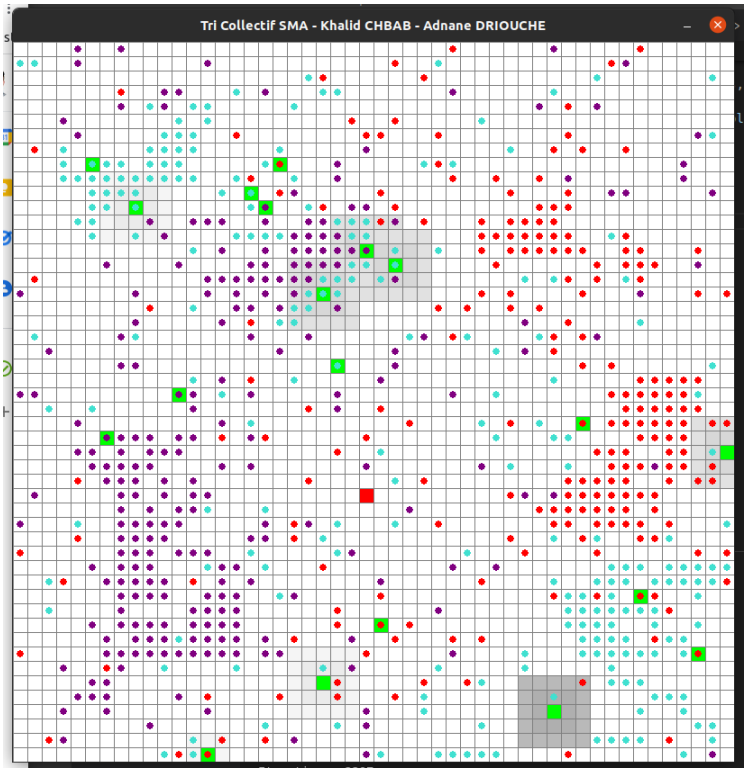


Figure 14 750 minutes après l'exécution