Université de Bretagne Occidentale
Faculté des Sciences et Technologie
Master ILIADE deuxième année

Département d'Informatique
Premier Semestre
Année 2020-2021

Système Multi-Agents

Agents pour la résolution de problèmes exploration de carte

Réalisé par :

Khaled CHENOUF

Table des matières

I. SMA Réactif

1. Comportement d'un agent

Dans cet algorithme, chaque agent parcours toutes les directions présentes dans la **Map** retournée par la fonction **percevoirZones**, et ajoute les direction qui verifient l'ensembles des conditions dans une liste comme l'indique la figure ci-dessous

```
List<Direction> directions = new ArrayList<Direction>();
informations = percevoirZones();
for (Direction direction : this.informations.keySet()) {
    Zone zonecourante = informations.get(direction);
    // condition 1
    if (undiscovered().contains(direction) && zonecourante.isBordure())
        directions.add(direction.opposite());
    // condition 2
    if (undiscovered().contains(direction) && !zonecourante.isBordure())
        directions.add(direction);
    // condition 3
    if (undiscovered().contains(direction) && otherVehicules().contains(direction))
        directions.add(direction.next());
    // condition 4
    if (undiscovered().contains(direction) && !otherVehicules().contains(direction))
        directions.add(direction);
}
```

A la fin de la boucle, l'agent test :

- Si la liste est vide, il choisit une direction quelconque avec la fonction **getRandom** parce que aucune direction n'a vérifié l'une des conditions.
- Sinon il choisit une direction aléatoire depuis la liste

La figure ci-dessous représente la phase du test

```
if (directions.isEmpty()) {
    //pas de direction possible
    move(Direction.getRandom());
} else {
    // choix aléatoire depuis la liste
    move(directions.get(((int) (Math.random() * directions.size()))));
}
```

2. Résolution du problème de l'exploration

L'algorithme réactif permet au agents de parcourir la carte en choisissant les direction non découverte, et en évitant les zones qui contiennent des obstacles, des bordures et des véhicules.

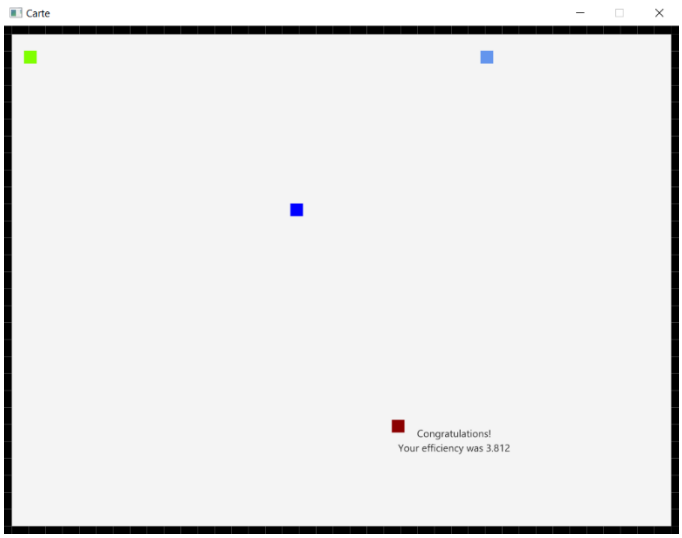
Le seul problème de cette exploration est dans le cas où l'agent ne perçoit aucune direction avec une zone non découverte autour de lui, son déplacement dans la carte sera que avec la fonction **Random**, cela démine l'efficacité des agents

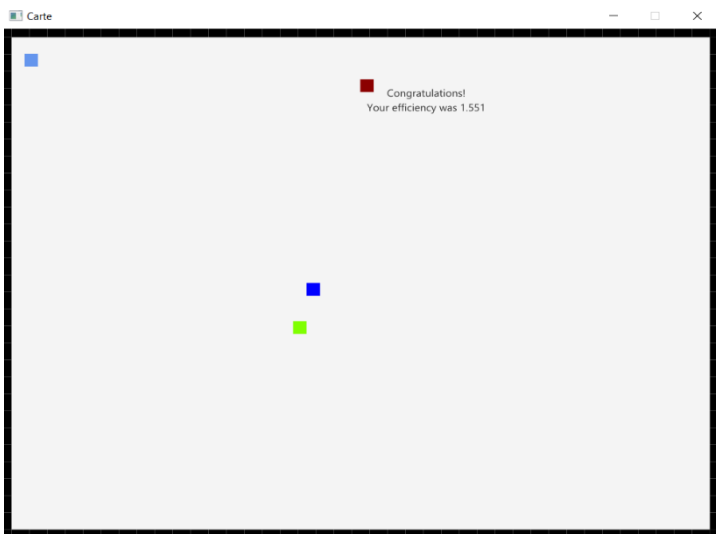
3. Amélioration du système

Pour améliorer le système, il faut traiter le cas où l'agent ne perçoit pas les directions avec des zones non découvertes, malgré qu'elles existent dans la carte.

Par exemple au lieu de choisir une direction aléatoirement, il choisit une direction bien définit, il la parcours jusqu'à arriver à la bordure ensuite il change de direction, il fait la même chose avec toutes les directions possible.

Les figures ci-dessous représentent les résultats obtenus par l'agent réactif





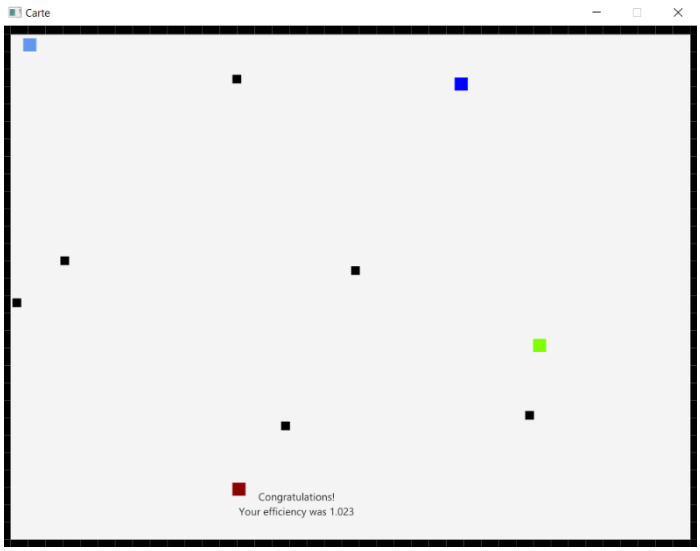
4. Traitement des obstacles

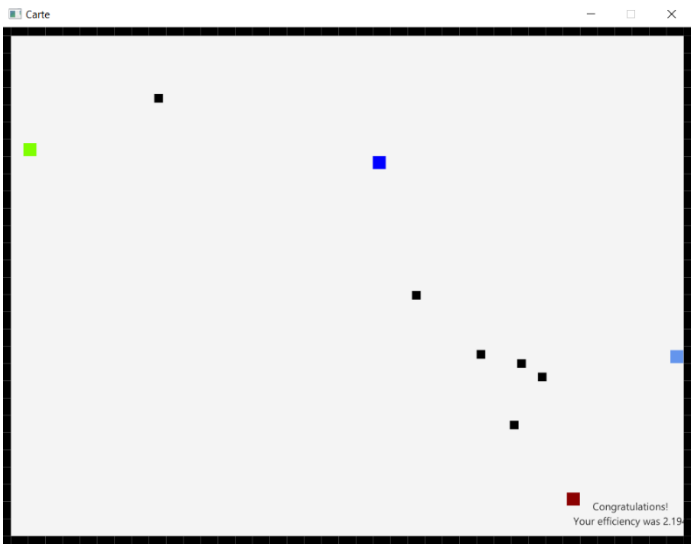
Pour traiter les obstacles dans le SMA réactif, lors de la récupération d'une position de la liste, l'agent test :

- Si la direction récupérée appartient à la liste retournée par la méthode **obstacles** alors l'agents prend la direction prochaine en utilisant la fonction **next**.

La figure ci-dessous représente la partie du traitement des obstacles

```
} else {
    Direction direction = directions.get(((int) (Math.random() * directions.size())));
    if (obstacles().contains(direction))
        move(direction.next());
    else
        move(direction);
}
```





5. *Résultat de l'algorithme*

L'algorithme réactif fonctionne quelque soit le nombre d'agent et la taille de la carte.

II. SMA Cognitif

1. Comportement d'un agent

Dans cet algorithme, chaque Agent suit un parcours bien définie, en utilisant leurs identifiants pour différencier entre eux.

le premier type d'agent : c'est les agents avec un id pair, le deuxième type d'agent : c'est les agents avec un id impair, et chacun de ces types est aussi diviser sur deux groupe.

La figure ci-dessous représente l'architecture de base du programme pour différencier entre les agents , c'est un exemple de 8 agents.

```
if (getAgentId() % 2 != 0) {
    if (((getAgentId() + 1) / 2) % 2 != 0) {

            // Agent 1 , Agent 5

    } else {

            // Agent 3 , Agent 7
    }
} else {
    if (((getAgentId()) / 2) % 2 == 0) {

            //Agent 2 , Agent 6

    } else {

            //Agent 4 , Agent 8

    }
}
```

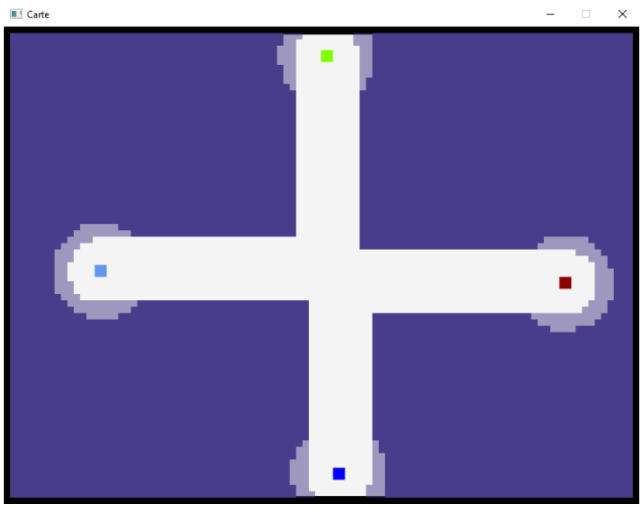
Chaque Agent récupère la direction depuis une liste qui contient une seule direction au début, tel que l'agent prend en premier cette direction est test :

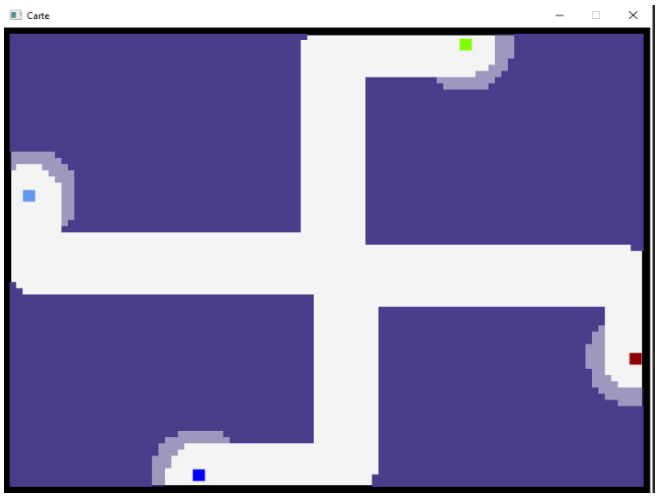
- Si la direction appartienne à la liste retournée par la fonction **undiscovered** alors il continue dans la même direction
- Sinon on ajoute sa prochaine direction dans le sens horaire avec la fonction **next** dans la liste et on incrémente une variable **i**.
- le prochaine tour l'agent récupère la **i ème** direction depuis la liste

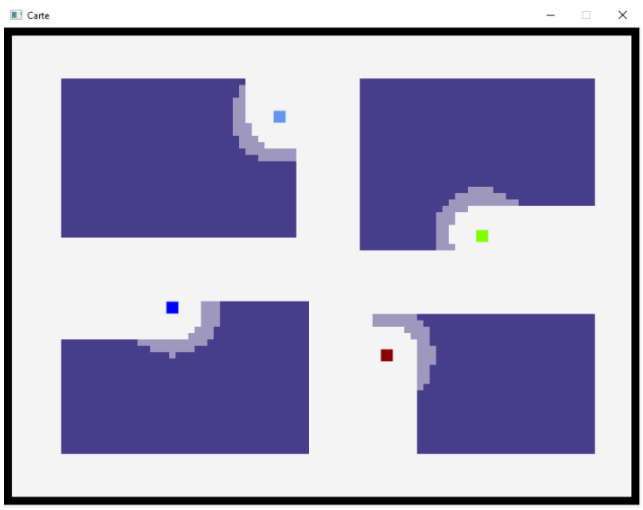
La figure ci-dessous représente l'algorithme suivi par les agents

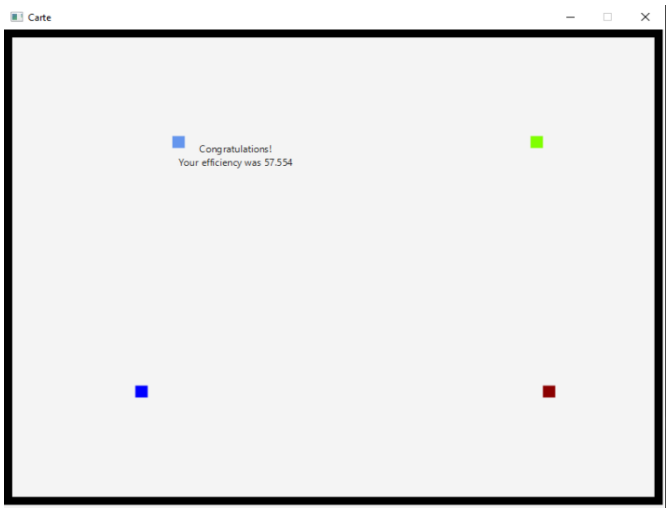
```
Direction directionCourante = null;
percevoirZones();
if (getAgentId() % 2 != 0) {
    if (((getAgentId() + 1) / 2) % 2 != 0) {
        directionCourante = List_dir1.get(i);
        if (undiscovered().contains(List_dir1.get(i)))
            move(List_dir1.get(i));
        else {
            i = i + 1;
            List_dir1.add(directionCourante.next());
        }
    } else {
        directionCourante = List_dir2.get(i);
        if (undiscovered().contains(List_dir2.get(i)))
            move(List_dir2.get(i));
        else {
            i = i + 1;
            List_dir2.add(directionCourante.next());
        }
    }
} else {
    if (((getAgentId()) / 2) % 2 == 0) {
        directionCourante = List_dir3.get(i);
        if (undiscovered().contains(List_dir3.get(i)))
            move(List_dir3.get(i));
        else {
            i = i + 1;
            List_dir3.add(directionCourante.next());
        }
    } else {
        directionCourante = List_dir4.get(i);
        if (undiscovered().contains(List_dir4.get(i)))
            move(List_dir4.get(i));
        else {
            i = i + 1;
            List_dir4.add(directionCourante.next());
        }
    }
}
```

Les figures ci-dessous représentent l'architecture du parcours effectué par les agents :









1. *Résolution du problème de l'exploration*

L'algorithme cognitif permet à chaque agents de parcourir la carte avec un parcours bien définit, tel que :

- La carte elle sera divisée sur le nombre d'agents,
- L'algorithme attribue une direction initiale à chaque agents
- Chaque agents va suivre que la direction avec des zones non découverte.
- Si la direction courante de l'agent n'aura pas des zones non découverte, alors l'agent passe à la prochaine direction dans le sens horaire avec la fonction **next.**

Le problème d'exploitation avec l'agent cognitif c'est que les agents ne partagent pas le travail équitablement , y'en a des agents qui parcours des surface plus importante que d'autres agents.

2. *Amélioration du système*

Pour améliorer le système faut trouver un moyen comment limiter les surfaces parcourues par les agents. Comme ça dés le départ les agents partagent la carte équitablement.

3. *Résultat de l'algorithme*

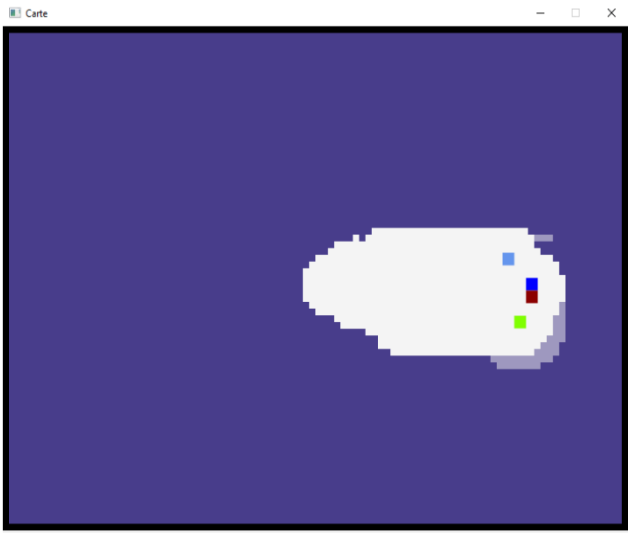
L'algorithme cognitif fonctionne quel que soit le nombre d'agent et la taille de la carte.

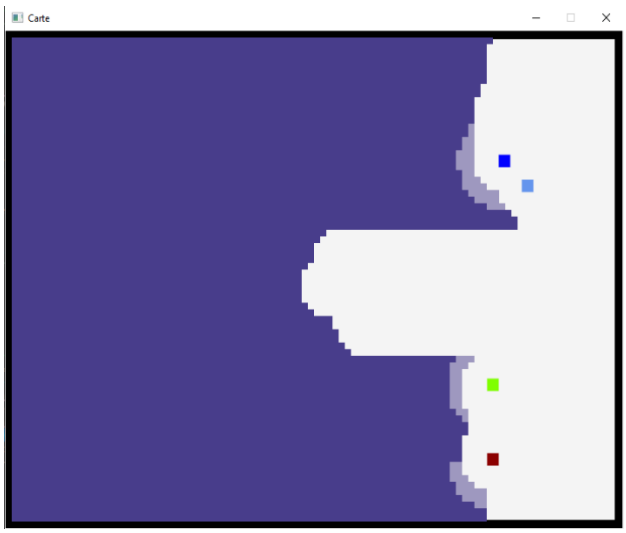
III. SMA Auto-organisé

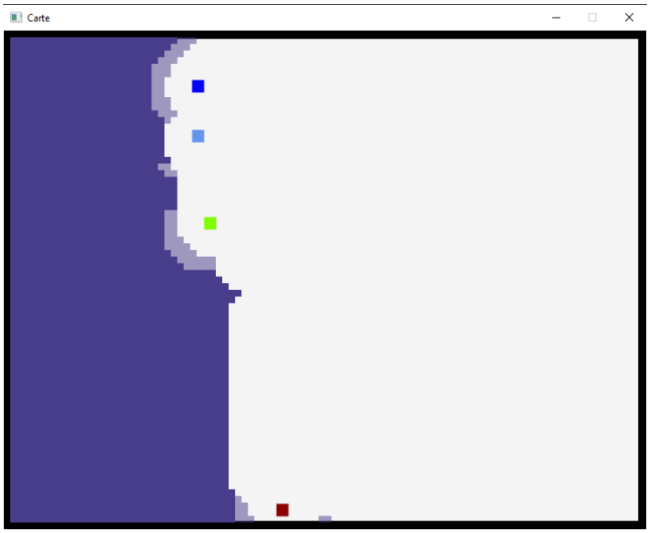
1. *Comportement d'un agent*

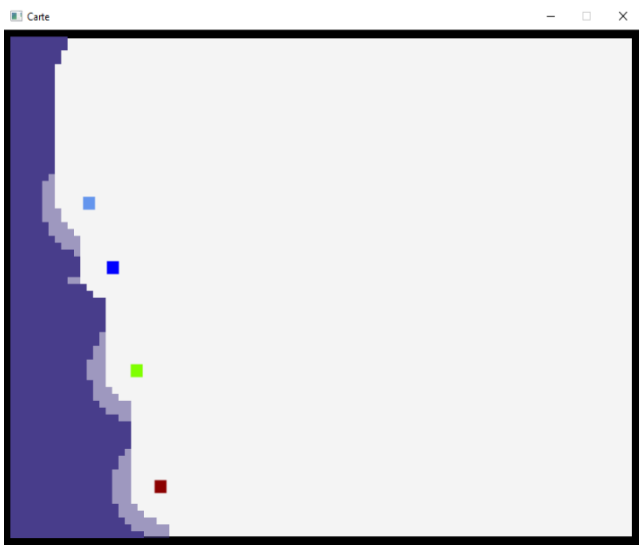
Le système que j'ai choisi pour la résolution de l'exploration, c'est la migration des oiseaux qui représente un déplacement régulier de nombreuses espèces d'oiseaux, tel que chaque agent représente un oiseau.

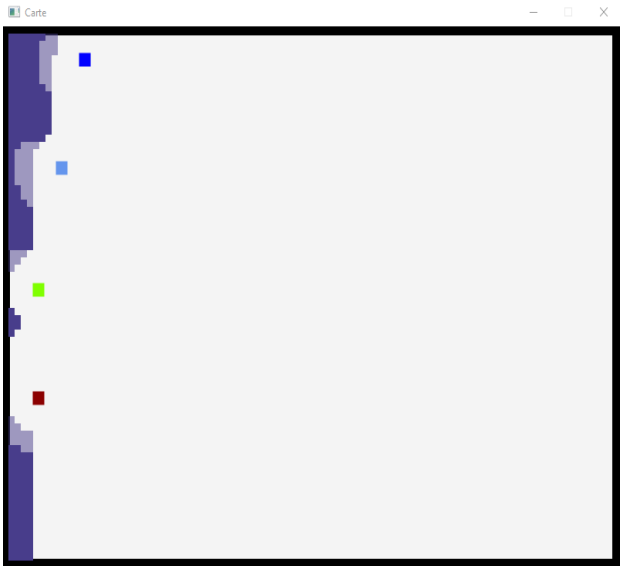
Tous les agents prennent la même direction en premier lieu , après il commence à balayer l'espace verticalement en allant de l'est vers l'ouest.

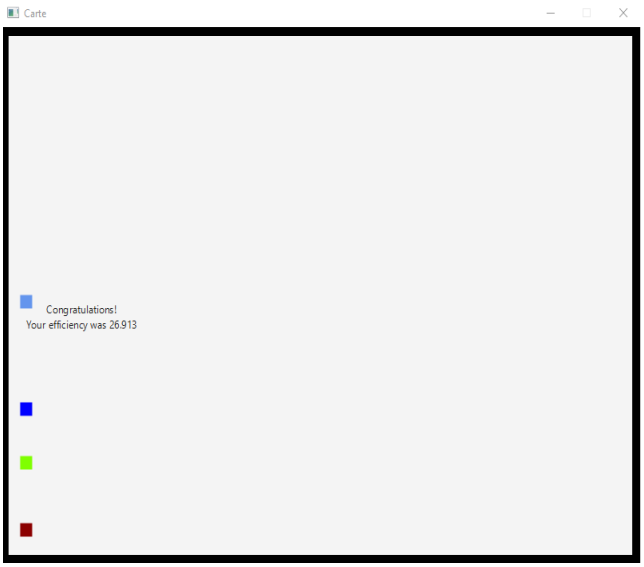












1. *Résolution du problème de l'exploration*

L'algorithme auto-organisé permet au agents de se déplacer dans la carte en groupe en allant d'une zone A initiale vers une zone B finale. Tel que à la fin les agents se retrouvent toujours dans la bordure de l'ouest.

Le problème d'exploitation avec l'agent auto-organisé c'est que lors du balayage de l'espace de l'est vers l'ouest, les agents oublient de temps en temps quelques zones et ils ne pouvant pas y revenir par la suite.

2. *Amélioration du système*

Pour améliorer le système, il faut trouver un moyen pour permettre les agents à retourner vers les zones oubliées lors du balayage.

1. *Résultat de l'algorithme*

L'algorithme auto-organisé fonctionne quel que soit le nombre d'agent et la taille de la carte.

IV. Test des solutions

Version	1	2	3	4	5	6	7	Moyenne	Meilleure
Réactive	3.81	1.55	4.87	2.36	3.89	1.63	2.01	2.87	4.87
Cognitive	57.55	57.55	57.55	57.55	57.55	57.55	57.55	57.55	57.55
AutoOrganisé	26.91	31.12	25.36	33.20	28.32	26.12	29.32	28.75	33.20
ObstacleRéactive	1.023	2.19	3.14	2.14	1.67	3.67	4.12	2.56	4.12