Projet A.N.D.R.O.I.D.E

2018-2019

Le problème de la patrouille multi-agents

Etudiant:

IDRI Lila

PAVKOVIC Dragan

Encadrante:

Mme GUESSOUM Zahia

SOMMAIRE

1. Introduction

Les systèmes multi-agents SMA proposent une nouvelle approche de l'intelligence artificielle; L'accent est mis sur la coordination, l'interaction et la satisfaction des buts communs ou non à l'ensemble des agents.

L'une des tâches dans lesquelles les SMA sont utilisés est la tâche de la patrouille. Cette dernière, est la modélisation formelle de la situation dans laquelle une zone géographique contenant de multiples points d'intérêt doit être supervisée, contrôlée ou protégée continuellement et le plus fréquemment possible par un ensemble d'agents .

Le problème de la patrouille d'un ensemble de robots dans un environnement prédéfini ou inconnu a fait l'objet de plusieurs études depuis plusieurs années. Différentes hypothèses ont été posées sur ce problème:

- Patrouille en système fermé: basée sur l'hypothèse d'un nombre constant d'agents, aucun agent ne peut ni entrer ni sortir du système. Cette hypothèse est jugée trop forte et restreint l'applicabilité des modèles de patrouille de SMA [1].
- Patrouille en système ouvert: les agents sont libres d'entrer et de sortir du système à tout instant. Cette dynamité permet de modéliser des applications plus réelles et plus complexes [1].

Ces dernières années, deux variantes de ce problème ont été définies:

- Patrouille avec adversaire en système ouvert/fermé: le SMA doit faire face à d'éventuelles intrusions d'entités étrangères au système.
- Patrouille temporelle en système ouvert: le système doit rendre les visites aussi rapprochées que possible sur l'ensemble des points d'intérêts. Ce problème est un bon cas d'étude pour l'étude de la coordination dans les SMA [2].

2.Description

Notre notre projet, on se limite à une seule zone d'intérêt. Le territoire est subdivisé en trois zones graduées de la plus risquée à la moins risquée, commençant de l'intérieur vers l'extérieur. La zone centrale étant appelée zone d'intérêt, les agents ont pour mission la protection de celle-ci, en se positionnant d'une manière optimale pour couvrir la zone à protéger localement, puis en réadaptant leurs comportements lorsqu'un événement survient (entrer ou sortie d'un agent du système lors d'une attaque étrangère) .

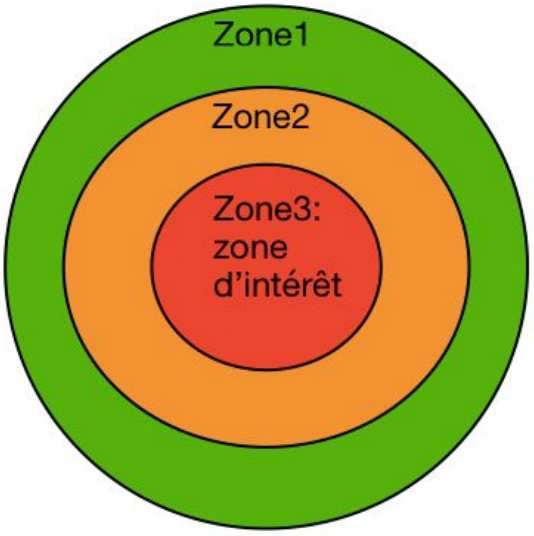


Figure 1 : répartition de l'environnement en trois zones.

Au départ, les agents se placent sur la zone la moins critique (zone 1), et lorsqu'un événement survient, ils doivent pouvoir enclencher un mécanisme de repli sur la zone 2. Si le mécanisme de repli et enclenché sur la zone 3, les agents donnent l'alerte.

L'objectif de ce projet et de construire un système multi-agents de la patrouille en environnement connu. Il consiste à coordonner le déploiement et l'interaction d'un ensemble d'agents qui se déplace sur un territoire prédéfini.

Pour respecter les contraintes horaires, on met en place un ensemble d'hypothèses que l'on suppose vraies:

- L'autonomie en énergie d'un agent est supposée infinie,
- Le rayon de chaque zone est connu et paramétrer en dur,
- Au départ, on dispose d'un nombre suffisant d'agents pour couvrir la zone 1.

Nous devons donc d'abord concevoir et implémenter un environnement de simulation qui respecte les hypothèses de départ et qui permet la réalisation de l'expérience. Ensuite, implémenter les algorithmes nécessaires au déploiement optimal des agents. Enfin, doter les robots des comportements leur permettant de mettre en place une stratégie de protection de la zone d'intérêt.

3.Etat de l'art

3.1. Cadre mathématique:

L'ensemble de travaux portant sur les stratégies de patrouille en système multi-agents[2][3][4][5][6][7][9][10] considèrent que l'environnement à patrouiller est connu, bidimensionnel et peut être modélisé avec un tuple <G,S,M>, dans lequel G est un graphe, S une société d'agents et M un ensemble de métriques.

G=<V, E>; V = { v_1, v_2, … , v_N } est l'ensemble des N positions(noeuds) à visiter. E = { e_{ij}} est l'ensemble des arêtes reliant deux noeuds vi et vj. Chaque noeud vi ∈ V est pourvu d'une priorité pi, et chaque arête ei possède une longueur li représentant la distance entre les deux extrémités de l'arête e_{ij}.

[4][5][6][7][9][10] proposent un ensemble de métrique M pour évaluer la qualité d'une stratégie de patrouille, qui devront être calculés au niveau de chaque noeud ou au niveau du graphe entier:

- L'oisiveté d'un noeud I (Idleness): nombre de pas de temps où un noeud est resté non visité.
- L'oisiveté du graphe IG: moyenne de l'oisiveté de tous les noeuds du graphe pour un instant donné.
- Temps de stabilisation dans un système ouvert: temps nécessaire à la reconfiguration des agents sur le graphe après un événement.
- Courbes de stabilisation dans un système ouvert: vitesse à laquelle à SMA atteint un certain seuil de la valeur stabilisée.

3.2. Travaux antérieurs:

De nombreuses stratégies ont été proposées ces dernières années. Les auteurs de [9] (2003) ont été les premiers à avoir proposer une étude comparative d'agents. En les divisant entre réactifs ou cognitifs, communicant ou non, avec un coordinateur ou non et par stratégie. Parmi les stratégies, on peut citer:

- Random Reactive agent (RR): agent réactif, non communicant, il choisit le noeud cible au hasard parmi ses noeuds voisins.
- Conscientious Reactive agent (CR): agent réactif, non communicant , il choisit le noeud cible son noeud voisin ayant la plus grande oisiveté instantanée.
- Cognitive Coordinated agent (CC): agent cognitif, communicant avec un coordinateur qui lui assigne son prochain noeud cible.

Chevaleyre [5, 6] (2003-2004) considère de problème de la patrouille comme un problème d'optimisation combinatoire. Il a donné des preuves mathématiques qu'avec un seul agent, le problème de la patrouille est équivalent à une variante du problème du voyageur de commerce

(TSP). Par conséquent, le problème de patrouille pouvait être résolu avec un algorithme permettant de résoudre une instance du TSP. Pour le problème multi-agents, il a montré que la stratégie à cycle unique(SC)[1] est moins intéressante qu'une stratégie rationalisée sur le critère AvrIG quand le graphe comporte d'importantes variations des valuations des arcs.

La stratégie suivante apparaît dans [10] (2004), elle propose l'utilisation de l'apprentissage par renforcement. Tel que le Grey-Box Learner Agent (GBLA)[2].

Les auteurs de [7] (2006) ont proposé des stratégies décentralisées à base de système d'enchères en Minimax[3] et en Minisum[4] tel que la stratégie FBA (Flexible Bidder Agent).

Plus récemment (2010), une stratégie basée sur l'approche gravitationnelle est proposé dans [11].

3.3. Résultats:

L'ensemble des travaux cités précédemment montre que les agents cognitifs sont plus adaptés à leurs homologues réactifs au problème de la patrouille en SMA; et plus les agents sont coordonnés, plus stratégies fournissent de meilleurs résultats. En particulier, sur le premier critère (l'oisiveté), on a l'ordre suivant:

$$SC > FBA = GBLA > CC > CR > RR$$

En système ouvert, sur les phases stables, les stratégies présentent des performances similaires qu'en système fermé.

[1] Un agent coordinateur calcule le cycle minimal du graphe, puis distribue uniformément les agents autour de ce chemin suivant leurs points de départ.

[2] Chaque agent connait le prochain noeud de chaque autre agent.

[3] Le Minimax est un critère de bien être égalitaire, maximise l'utilité de chaque agent en minimisant le chemin le plus long[8].

[4] Le Minisum est un critère de bien être utilitaire, c'est maximiser l'utilité moyennes des agents [8].

4.Réalisation

Pour développer notre système multi-agents, on a décidé d'utiliser un langage de programmation orienté agent qui est NetLogo.

Pour modéliser un SMA, NetLogo propose trois types d'agents:

- Les agents patches: ils représente le "sol" carré du monde NetLogo (le monde est en 2D)
- Les agents tortues (turtles): ils peuvent se déplacer dans le monde de NetLogo sur les patches.
- Les liens (links): agents qui relient deux tortues.
- L'observateur (observer): il n'a pas de position déterminé, c'est en quelque sorte le superviseur du monde NetLogo.

NetLogo a deux procédures nécessaires pour une simulation,la procédure "setup" qui définit un environnement à l'état initial, et la procédure "go" qui simule l'expérience.

NB: Le noms de ces procédures peut être changé.

4.1. Etape 1: Conception et implémentation de l'environnement

On utilise les agents patches pour définir notre environnement subdivisé en trois zones. le code correspondant est dans la procédure "setup"; le rayon de chaque zone est connu.
Chaque agent est modélisé par une tortue. Une fois les agents créés, on leur attribue un rayon de perception.
Initialement, les agents se positionnent aléatoirement sur la zone 3 (zone la moins critique)

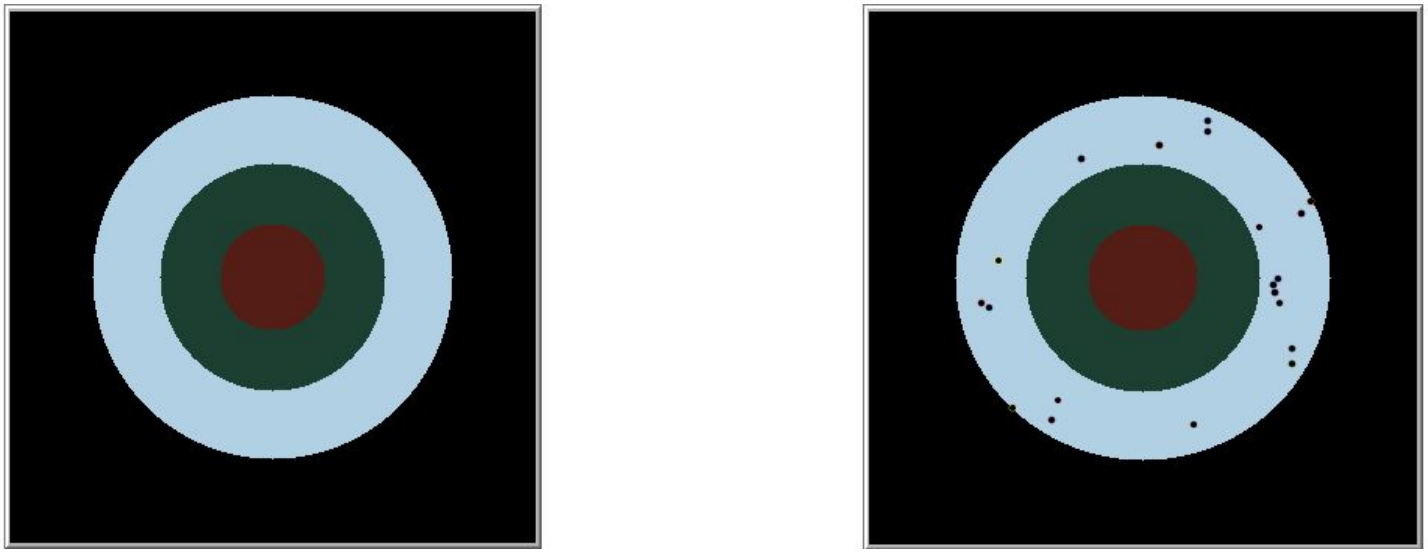


Figure 2 : Modélisation de l'environnement (zones + agents) en NetLogo.

4.2. Etape 1: Déploiement optimal des agents

Les agents doivent se répartir efficacement dans la zone à couvrir. Pour ce faire, on implémente l'algorithme de Voronoi qui génère des régions dites régions de Voronoi, dont le nombre est égale aux nombre d'agents du système. L'ensemble de ces régions forme le diagramme de Voronoi.

Définition du diagramme de Voronoi:

On appelle diagramme de **Voronoï** un découpage du plan (pavage) en régions de Voronoi, à partir d'un ensemble discret de points appelés « germes ». Chaque région enferme un seul germe, et forme l'ensemble des points du plan plus proches de ce germe que de tous les autres.

Figure 2 : Diagramme de Voronoi à 2 germes.

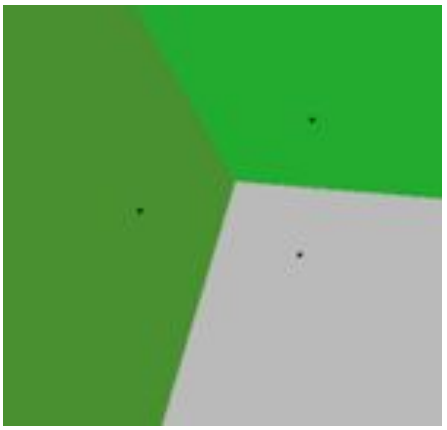


Figure 3 : Diagramme de Voronoi 2 germes.

Dans le cas de deux germes (Figure 2), la zone frontière des deux régions est simplement la médiatrice des deux germes, et dans celui de 3 germes (Figure 3), les zones frontières sont les médiatrices du triangle définie par les trois germes.

Algorithme de Voronoï

Pour chaque agent protecteurs faire :

Affecter son numéro dans **indice-agent** (variable globale)
Affecter sa couleur dans **couleur-agent** (variable globale)
Affecter sa **perception** à 2
Affecter à **voisins** l'ensemble des agents protecteurs(tortue) dans sa perception
Effectuer la procédure recolore dans sa perception (colorie les points de sa région de Voronoi perceptive, la partie de sa région de Voronoi située dans sa perception)
Affecter à **point** le patch de sa région de Voronoi perceptive le plus éloigné de l'agent.

Tant que **perception** est plus petite que 2* distance(agent, point) faire
doubler **perception**
Affecter à **voisins** l'ensemble des agents protecteurs dans sa perception
Effectuer la procédure recolor dans sa perception
Affecter à **point** le patch de sa région de Voronoi perceptive le plus éloigné de l'agent.
Fin faire tant que.

Fin faire pour chaque agent.

Déroulement de l'algorithme pour un seul agent:

l'agent a un rayon de perception initialement de longueur 2. Dans ce disque de perception, il perçoit les agents présents et construit l'ensemble des points appartenant à la région de Voronoï dans son disque de perception. L'agent considère ensuite le point de la région construite le plus éloigné de lui, se trouvant à une distance dmax. Tant que son rayon de perception est plus petit que le double de cette distance dmax, il double son rayon de perception et redéfinie sa région de Voronoï dans son nouveau disque de perception.
La procédure voronoï , qui fait appelle à la procédure recolor, exécute ce travail.

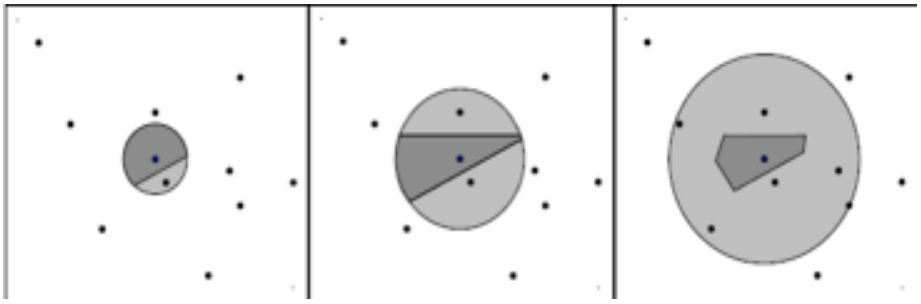


Figure 4 : Définition d'une région de Voronoï par un agent en utilisant l'algorithme Voronoï définit en dessus.

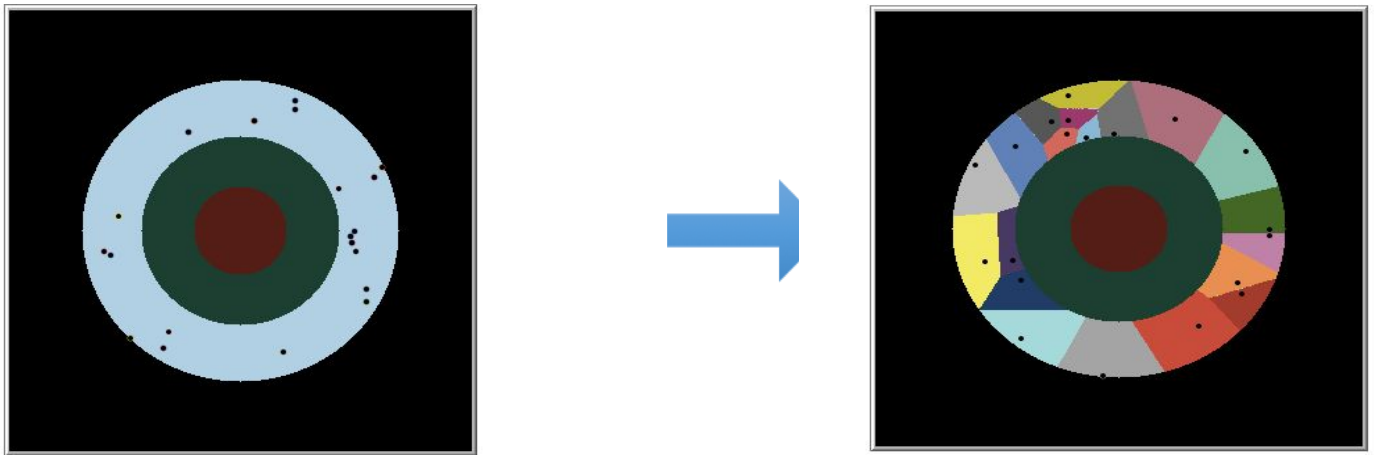


Figure 5 : Algorithme de Voronoï exécuté par 20 agents.

Les agents se positionnent aux points germes de leurs région Voronoï qui ne correspondent pas aux centroïdes [5]. Afin qu'ils se mettent sur les centroïdes, on implémente l'algorithme de Lloyd.

Code (algorithme Lloyd) pour trouver le centroïde de chaque région:

```
to positionner
  ask protectors [
    set xcor mean [pxcor] of patches with [pcolor = [color] of myself]
    set ycor mean [pycor] of patches with [pcolor = [color] of myself]
    let point max-one-of (patches with [pcolor = [color] of myself]) [distance myself] ; patche le plus eloigné de l'agent
```

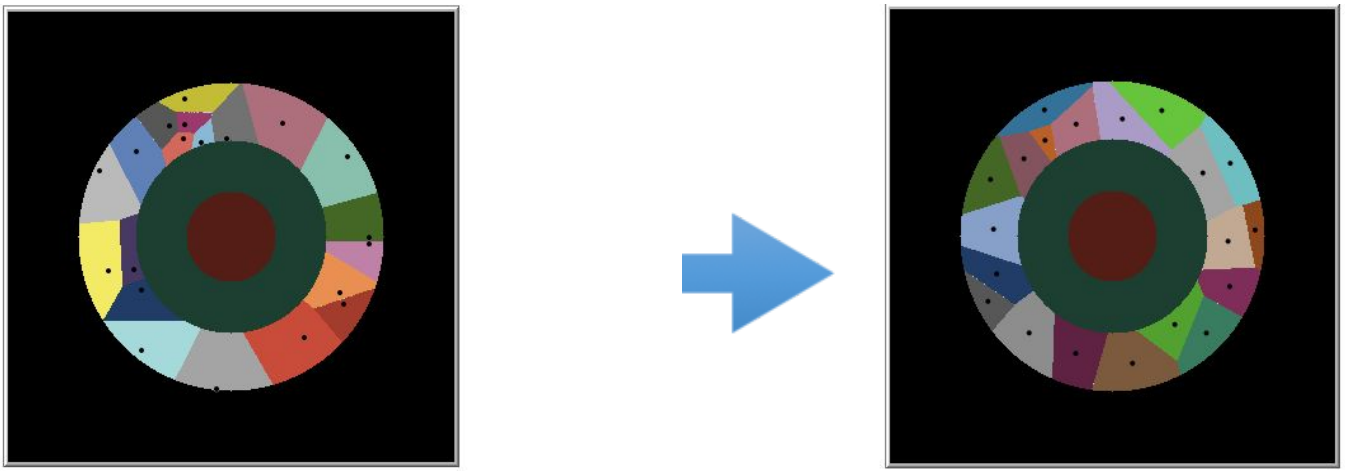


Figure 6 : Algorithme de Lloyd exécuté par 20 agents.

[5] le centre d'une région voronoï.

Une fois les agents placés sur leurs centroïdes, ils ne sont plus germe de leurs régions Voronoï. Il faut donc redéfinir la région de Voronoï qu'ils engendrent en ce nouvel endroit. Ainsi la procédure voronoï est réappliquée puis le recentrage est fait et ainsi de suite, en boucle tant que le système reste inchangé. Cet algorithme dit de Lloyd fait converger l'agent vers un point qui est à la fois centroïde et germe de la région de Voronoï qu'il définit. L'intérêt de ce point est qu'il est l'endroit où la somme des distances (élevées au carré) de l'agent aux points de sa région de Voronoï est minimale. C'est une manière d'optimiser l'emplacement de l'agent afin qu'il protège au mieux sa région.

Remarques :

1) La procédure setup ne place pas l'agent de façon optimal sur sa région. C'est la procédure positionner, effectuée en boucle, qui sans adversaire (le bouton attaque de la console), fera converger les agents vers la position optimum.

2) Si on note di la distance maximale de l'agent i aux points de sa région, un autre modèle est de minimiser le max sur i de ces distances d_i [12].

4.3. Etape 3: Mécanisme de repli

Les agents se replient sur la zone zone-actuelle +1 dans deux cas:

1) **Sortie d'un agent sans qu'il y est attaque d'un adversaire**: Chaque agent est le centre d'un disque de protection. Le rayon (le même pour tous) est défini à l'aide d'un curseur. Une fois les agents centrés sur leur région de Voronoï (Etape 2), on trouve pour chaque région le point le plus éloigné de l'agent germe. Si ce point n'est pas couvert par le disque de protection de l'agent le plus proche de lui (qui peut être autre que l'agent germe définissant la région auquel il appartient), on demande aux agents de se replier vers la zone inférieure (niveau de criticité= niveau(zone-actuelle) +1).

   La deuxième partie de la procédure positionner définit la condition de repli en affectant TRUE à la variable booléenne repli (variable globale). La procédure change-zone sera effectué dans ce cas.

   Code de la procédure Position:

```
ask point [
  let agent-proche min-one-of protectors [distance myself] ;protecteur le plus proche de point
  let d distance agent-proche   ; distance d entre point et le protecteur le plus proche
  if d > r-protection [ set repli true]  ; repli prend true si l'agent le plus proche de point ne protège pas point
]
```

Code de la procédure change_zone qui effectue le repli des agent sur la zone inférieure:

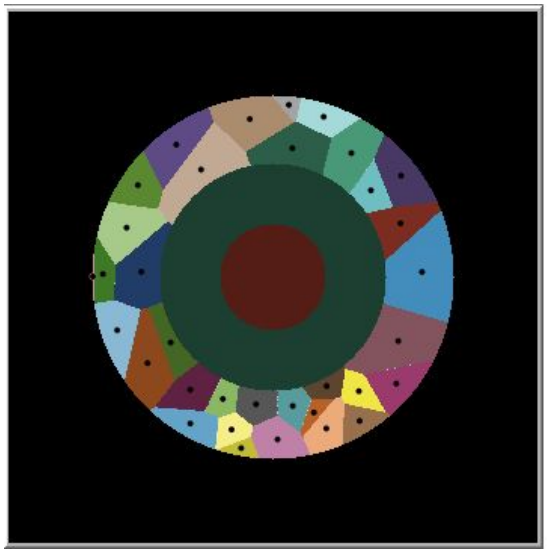
```
;;; procédure qui changent de zone les agents.
to change_zone
  set repli false
  ask protectors[
    facexy 0 0
    let destination patch-left-and-ahead 0 45
    move-to destination
  ]
  set zone-actuelle zone-actuelle + 1
end
```

2) **Sortie/destruction d'un agent suite à une attaque adverse:** une attaque est simulé de la manière suivante:un patch de la zone où les agents se situent est choisi aléatoirement. Les agents situés dans un rayon force-impact autour de ce point(patch) sont détruits.
La procédure missile avec comme paramètre le rayon (choisi via un curseur) effectue ce travail.

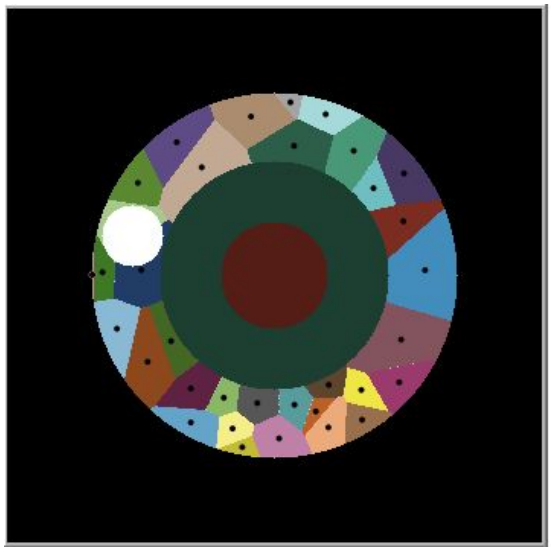
Code de la procédure missile:

```
;;; procédure qui lance un missile et qui détruit les agents dans un rayon puissance (paramètre) autour de l'impact
;;; un disque blanc s'affiche sur l'environnement
to missile [puissance]
  let impact n-of 1 (patches with [zone = zone-actuelle])
  ask impact [
    ask patches in-radius puissance [set pcolor 9.9]
    set morts protectors in-radius puissance
  ]
  ask morts [die]
end
```

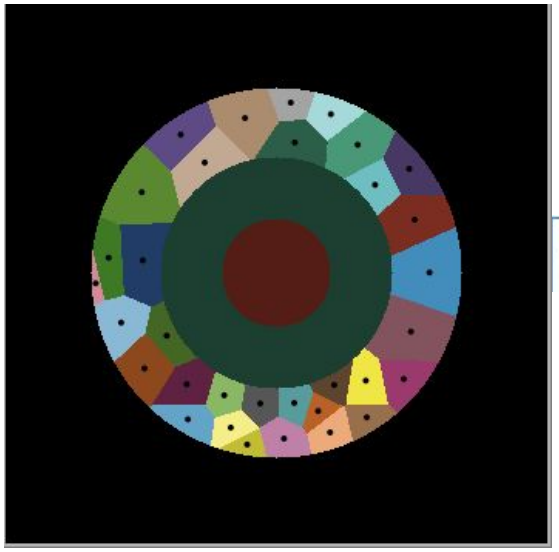
L'agent qui se trouve dans le rayon de l'impacte de l'attaque est détruit. Les agents restants tentent de trouver une reconfiguration pour couvrir la totalité de la zone (procédures define_environment, voronoï, positionner). Si leurs champs de protection ne le leurs permet pas (définie comme dans le cas 1), ils se replient sur la zone inférieure (procédure change_zone) puis réexécutent les procédures define_environment, voronoï, positionner; et ainsi de suite.



fin de l'exécution de "setup"



Exécution "go" puis "missile"




```
Command Center
observer: "Avant impact........"
observer: [22 5 14 26 18 23 31 33 21 2 27 8 12 17 24 25 4 29 11 19 13 0 10 7 3 16 1 15 32 20 30 6 28 9]
observer: "Apres impact........"
observer: [10 31 26 30 6 23 33 27 19 7 12 28 21 9 5 17 1 20 32 13 29 22 3 18 24 14 0 15 8 25 16 2 4]
```

Figure 7 : Reconfiguration des agents après une attaque ennemie.

Le nombre d'agent à baisser suite à la destruction d'un agent après l'attaque (procédure missile).

Leurs champs de protections leurs permet de rester sur la zone, les agents restants ont pu trouver une nouvelle configuration (define_environment, voronoï, positionner) pour couvrir la zone sans avoir recours au mécanisme de repli.

5.Résultats

La procédure setup :

Elle initialise la console, définie les 3 zones (les couronnes) à l'aide de define_environment, placent les agents d'une manière aléatoire sur une zone à l'aide de define_protectors, effectue les procédures voronoï et positionner.

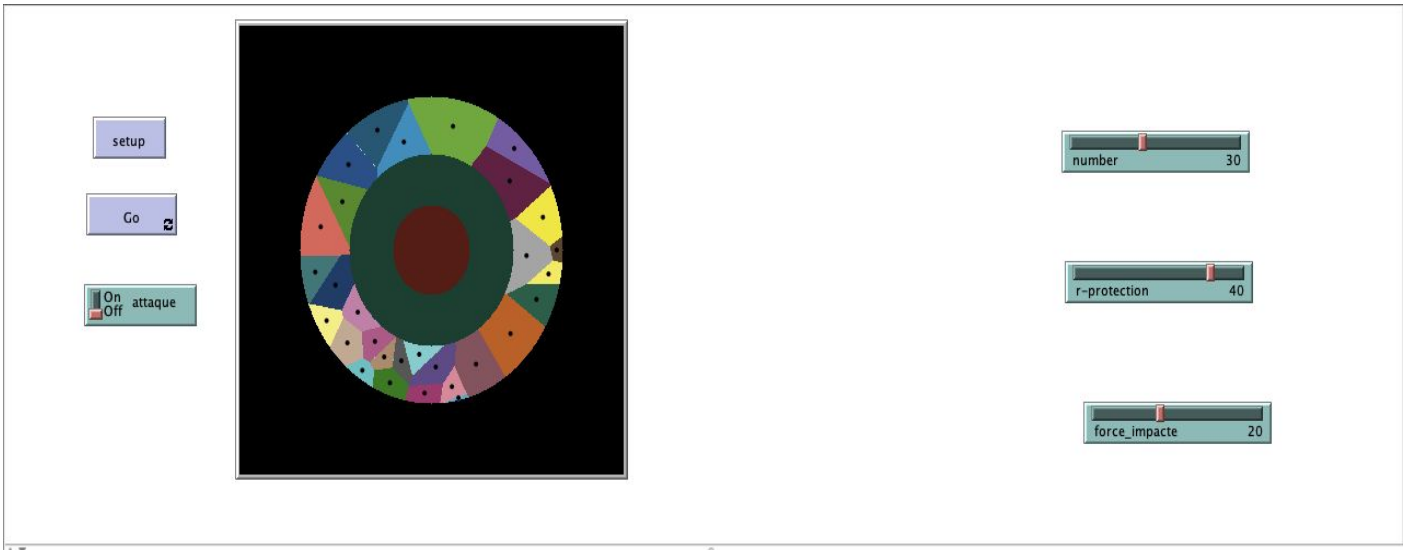


Figure 7 : Rendu de l'exécution de la procédure "setup": environnement de la simulation.

- Le bouton "attaque": permet de simuler ou non une attaque ennemie.
- Le bouton "number": permet de choisir le nombre d'agents du système.
- Le bouton "force_impacte": permet de choisir l'ampleur de l'attaque ennemie (choisir le rayon de la zone d'impacte).
- Le bouton "r-protection": permet de définir le rayon de protection des agents.

La procédure simuler.

Cette procédure effectue dans l’ordre et en boucle infinie : Missile, define_environment, voronoï, positionner et repli si besoin.

6.Conclusion

Pour rappel, ce projet a pour but la modélisation d’un problème de la patrouille multi-agents en environnement connu.

Par manque de temps, on a posé des hypothèses sur l'environnement de simulation.

On a réussi à représenter une version simplifiée d’un problème de la patrouille multi-agents. En réadaptant les algorithmes de résolution de de réels problèmes de la patrouille, on a pu mettre en place un mécanisme de patrouille qui permet de protéger une zone.

Dans l'idée d'améliorer notre implémentation, ainsi que les versions des algorithme utilisés, nous avons réfléchi à plusieurs approches pour coordonner les agnets. Celles ci n'ont pas put être explorées dans les délai imposé.

7. Bibliographie

[1] C. Poulet, « Étude du Problème de la Patrouille Multi-Agents en Système Ouvert », in *Rencontres des Jeunes Chercheurs en Intelligence Artificielle 2011*, Chambéry, France, 2011.
[2] « C. Poulet, V. Corruble, A.E.F. Seghrouchni, and G. Ramalho. The open system setting in timed multiagent patrolling. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM Int. Conf., IEEE, 2011. »
[3] A. ALMEIDA, « Patrulhamento Multiagente em Grafos com Pesos. », in MSc Dissertation, Universidade Federal de Pernambuco., 2003.
[4] A. ALMEIDA, G. Ramalho, et al, « Recent Advences on Multi-agent Patrolling. », In 17th Brazilian Symposium on AI, 2004.
[5] Y. Chevaleyre, « The Patrolling Problem », in International Report of University Paris 9, 2003.
[6] Y. Chevaleyre, « Theoretical Analysis of the Multi-Agent Patrolling Problem », in International Joint Confernce on Intelliget Agent Technology, 2004.
[7] T. Menezes, P. Tedesco, et G. Ramalho, *Negotiator agents for the patrolling task. Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*. 2006.
[8] C. Poulet, V. Corruble, et A. El Fallah Seghrouchni, « Travailler en équipe : le choix social appliqué au problème de la patrouille multi-agents », in *journées francophones sur les systèmes multi-agents (JFSMA '12)*, Honfleur, France, 2012, p. ?
[9] A. MACHADO, G. RAMALHO, J. ZUCKER, et A. DROGOUL, « Lecture notes in computer science, p. 155–170. », in *Multi- agent patrolling : An empirical analysis of alternative architectures*, 2003.
[10] H. SANTANA, G. RAMALHO, V. CORRUBLE, et B. RATITCH, « Multi- agent patrolling with reinforcement learning. In AAMAS-2004, Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems. », 2004.
[11] G. SAMPAIO, G. RAMALHO, et P. TEDESCO, « A technique inspired by the law of gravitation for the timed multi-agent patrolling. 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI) », 2010.
[12] Z. Drezner et A.Suzuki "the p-center location problem in an area" location Science, vol.4, no ½, pp 69-82, 1996.