Projet Multi-Agent

11419808 – NGUYEN Van-Thanh-Tu

11419771 – NGUYEN Xuan-Huy

*git : https://forge.univ-lyon1.fr/p1419808/MultiAgents/project\_members*

**Sommaire**

1. Idée initiale
2. Algorithme de guidance
3. Système de communication
4. Problèmes
5. Résultat

**Idée initiale**

Implémenter des agents qui peuvent résoudre eux-mêmes le problème de l’orientation ou par communiquer avec des autres pour un mieux solution.

Des agents sont des Thread de Java8 et chacun est implémenté avec algorithme de guidance, ils peuvent communiquer avec des autres en utilisant une table de communication (attribut static dans la Class Agent).

**Algorithme de guidance**

Au début, nous voulons d’implémenter l’algorithme Dijkstra mais dans un environnement en constante évolution comme le nôtre, ce ne serait pas trop efficace. Alors, l’algorithme que nous avons utilisé : chaque tour, l’agent va calculer quelle est la meilleure carrée (par rapport à la distance à la position finale) qu’il doit visiter. Ensuite, il va le marquer cette carrée comme visité (pour éviter de revisiter 1 carrée plusieurs fois).

**Système de communication**

Pour la communication entre les Agents (entre les Thread), on va utiliser un structure partagé pour tous les Agents comme suivant :

Pseudo-code :

private static ConcurrentHashMap<Integer, ArrayList<Message>> table = new ConcurrentHashMap<>();

L’idee de base est qu’on va créer un membre « static » , nommé ***table***, partagé par les instances du Class Agent. En effet, cet variable contient un tableau des messages recu pour chaque Agent.

On a un Class Message contenant les informations que les Agents enverraient.

Pseudo-code :

enum Type {REQUEST, INFORM}

public class Message {

private Type type;

private int to;

private int from;

private Position fromPosition;

private Position toPosition;

}

Par exemple, si un Agent A est bloqué par un autre Agent B, A va utilise un message du type REQUEST pour demander B qui doit bouger. Et, B va utilise un message du type INFORM pour répondre les requetes.

En fait, chaque Agent a un fonction appelé **handleMessage()** pour traiter les messages avant de se bouger.

**Problèmes**

**Synchronisation**

Les threads sont généré en l’ordre hasard => difficile à gérer. Nous avons utilisé synchronized(this) pour que les threads marchent en l’ordre de index.