Multi-agents Bio-Inspirés

Valérian ACIER : *valerian.acier@hotmail.fr*

1 Modélisation du problème :

- Création d'une classe **Agent** ayant pour but de représenter les différentes actions et propriétés des agents. Ils possèdent une position ainsi qu'une mémoire.
- Création d'une classe **Board** représentant l'objet en charge de la position de chaque agent et leurs déplacements.
- Initialisation des **Agents** avec des positions aléatoires sur le **Board**.
- Initialisation des **Objets à récupérer** avec des positions aléatoires sur le **Board**.

1.1 Fonctionnement des agents

Chaque agent peut porter jusqu'à un objet et se déplace de manière aléatoire.

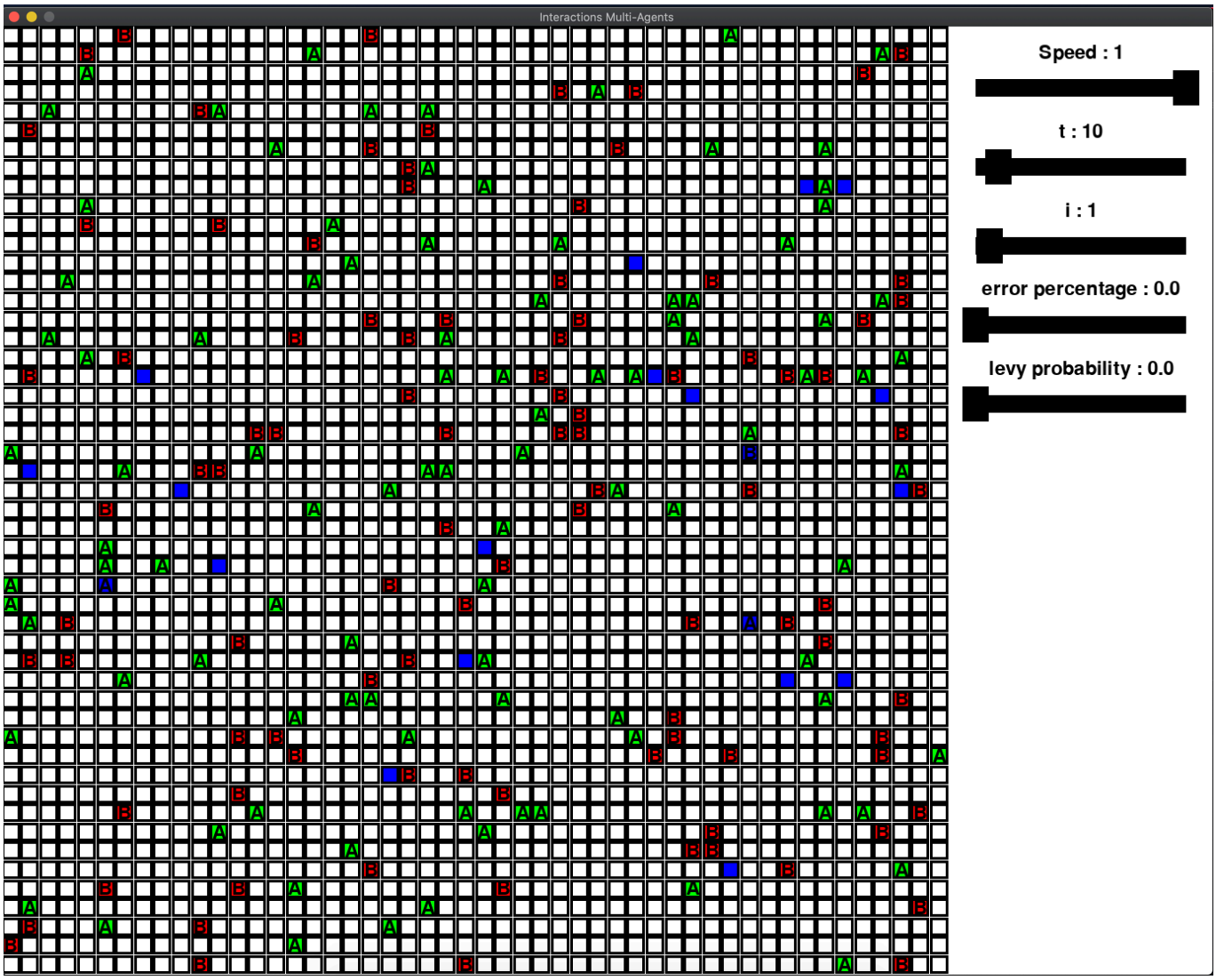

Figure 1: Fenêtre finale du TP

1.2 Les valeurs réglables

Sur la droite de l'interface des sliders sont présents pour permettre de régler les valeurs de certaines variables.

- Speed correspond au temps d'attente en seconde entre chaque action d'agent
- t correspond à la taille de la mémoire de l'agent
- i correspond aux nombres de déplacements avant de pouvoir récupérer ou déposer un objet
- error percentage introduit la notion d'erreur lors du repérage d'un objet
- levy probability permet aux agents de réaliser de grand déplacement avec une certaine probabilité (levy flight)

Influence des différents paramètres sur l'agent :

- T a pour effet d'augmenter la taille des clusters créée par les agents et permets la persistance du comportement d'un agent (un agent qui a vu beaucoup d'objet A va se focaliser plus longtemps sur l'objet A même s' il se trouve dans un cluster de B).
- I rend la création des clusters beaucoup plus compliqués voir les mélanges quand trop élevé, mais peut accélérer le rassemblement de deux clusters du même objet qui sont éloignés.
- Error percentage rend la clusterisation plus difficile, mais à moins de le mettre à une valeur élevée les clusters se forment toujours.
- Le vol de lévy accélère la création des clusters, car les agents vont avoir pour comportement de créer de petits clusters rapidement et ainsi favoriser l'émergence des agents dédiés à un type d'objet plus rapidement.
- Quand l'agent agit en fonction de son champ de vision ils vont alors créer des clusters basés sur l'environnement et non plus sur leurs propres mémoires et deux clusters d'un même objet vont être très difficiles à rassembler.

2 Le programme

J'ai implémenté la solution en Python pour mettre un code facilement compréhensible et facile à coder rapidement. J'ai utilisé la bibliothèque PyGame pour l'interface graphique.
Le programme accepte plusieurs arguments :

- **--width=1080** définit la largeur de la fenêtre du puzzle en pixels.
- **--height=1080** définit la hauteur de la fenêtre du puzzle en pixels.
- **--n=50** définit la taille du puzzle.
- **--agents=20** définit le nombre d'agents sur le puzzle.
- **--speed=1** définit le temps d'attente entre chaque action des agents en secondes.
- **--objects=200** définit le nombre d'objets présents sur le terrain.
- **--t=10** définit la taille de la mémoire des agents.
- **--kp=0.1** définit la variable Kp pour les probabilités.
- **--kd=0.3** définit la variable Kd pour les probabilités.
- **--i=1** nombre de déplacements entre chaque action (dépôt et récupération d'objet).
- **--error=0** probabilité qu'un agent se trompe dans la reconnaissance d'un objet.
- **--levy=0** probabilité que l'agent fasse un grand déplacement pour sortir de sa zone actuelle.
- **--view=0** pour que l'agent utilise sa vision à la place de sa mémoire (dans une zone de 6*6 autour de l'agent, mettre 1 pour utilise la vue plutôt que la mémoire.

Il est possible de faire varier certaines de ces valeurs via des sliders sur la droite de la fenêtre.
Les codes de couleurs de chaque case correspondent à un objet présent sur la cellule :

- **Rouge** objet B.
- **Vert** objet A.
- **Bleu** un Agent.