

Exercice 01- Enoncé

<u>1.</u>	<u>DÉFINITION DU BESOIN</u>	<u>2</u>
<u>2.</u>	<u>ENONCÉ DE L'EXERCICE</u>	<u>4</u>
<u>3.</u>	<u>Pour démarrer le codage (optionnel)</u>	<u>5</u>

1. Définition du besoin

Le jeu de la vie et de la mort.

Cette sorte de jeu consiste à exécuter des **Agents** (sorte de fourmis) qui se déplacent de case en case dans un espace infini, appelée **Espace Tore**.

Cet espace infini est modélisé sous la forme d'un tore : une grille rectangulaire fermée sur elle-même de taille NxM.

Chaque agent est autonome (le joueur n'intervient pas) et se déplace de case en case en fonction de sa politique de déplacement. A chaque déplacement, un agent se déplace toujours d'une seule case dans les 8 directions possibles.

Le rythme de déplacement des agents est le même pour tous les joueurs (sinon c'est triché).

Si la case de destination est occupée par un autre agent alors le déplacement ne se fait pas, et si la case est occupée par un ennemi alors la vie de l'agent ennemie est décrétementée de 1.

Si le compteur de vie d'un agent est à 0 alors il meurt.

Au départ le nombre de vie de tous les agents est de 10.

Chaque agent a une couleur qui est propre à chacun des joueurs.

On décide que tous les agents d'un joueur s'exécutent dans un conteneur spécifique, la **population** d'un joueur.

Dans un premier temps les agents ne se battent pas. Ils peuvent se déplacer, se mettre en rang de bataille, ...

Dans un deuxième temps, on déclenche le mode bataille. A partir de ce moment, les agents de couleur différente se battent entre eux.

Le joueur gagnant est celui qui reste seul avec au moins 1 de ses agents dans la grille.

Chaque joueur programme sa politique de déplacement de ses agents. Le joueur gagnant est donc, à priori, celui qui a réalisé les meilleurs algorithmes.

Pour cela, chaque joueur utilise, dans ses algorithmes, les fonctions prédéfinies suivantes :

- Faire arriver un de ses agents dans la grille.
- Demander de déplacer un de ses agents
- Savoir si le jeu est en mode combat
- Connaître la position (en x,y) de tous les agents d'une couleur donnée
- Connaître la position (en x,y) de tous les agents qui ne sont pas d'une couleur donnée
- Déterminer, dans un tore, la position la plus proche parmi une liste de position et à une distance supérieure à un seuil (qui peut être à 0)
- Déterminer, dans un tore, le sens de direction vers un point en suivant la distance la plus courte

Chaque joueur a sa propre IHM qui lui permet de créer ses agents dans la population et faire arriver ses agents dans la grille.

Théoriquement, le nombre d'agents au total doit être le même pour tous les joueurs, et un joueur n'a pas le droit de créer de nouveaux agents en cours de la partie.

Une autre IHM permet à chacun des joueurs de visualiser à distance le déplacement de tous les agents de toutes les populations.

L'organisateur du jeu à sa propre IHM qui lui permet de lancer le serveur de jeu en lui indiquant la taille de la grille et la taille de chaque cellule de la grille.

Cette IHM permet en cours de jeu de :

- Changer le rythme de déplacement de tous les agents
- Passer en mode combat
- Faire une remise à zéro du jeu afin de recommencer une partie.

Chaque composant logiciel est une JVM différente pouvant être répartie sur n'importe quelle machine suivant le principe de RMI. On impose le choix des 4 composants logiciels suivants :

- IHM JOUEUR (1 par joueur)
- POPULATION JOUEUR (1 par joueur)
- IHM GRILLE (1 par joueur)
- SERVEUR JEU (unique)

IHM JOUEUR

Ce composant logiciel est une IHM permettant, à un joueur, de créer ses agents. Il contient les algorithmes du joueur qui sont téléchargés sur la population du joueur (POPULATION JOUEUR).

POPULATION JOUEUR

Ce composant logiciel crée les agents d'un joueur. Il fait arriver les agents dans la grille du serveur de jeu.

IHM GRILLE

Ce composant logiciel visualise sous la forme d'une grille le déplacement des agents.

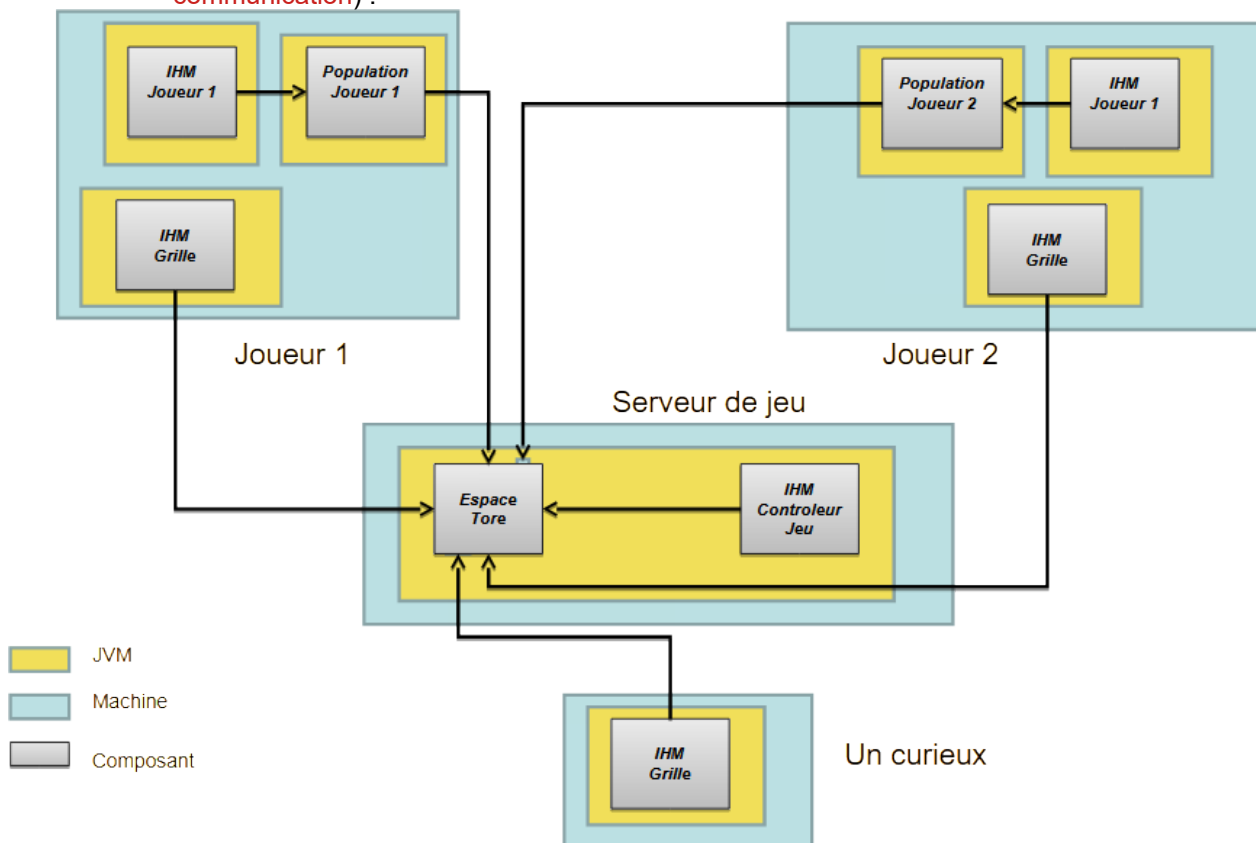
SERVEUR JEU

Ce composant logiciel centralise toutes les informations communes à tous les joueurs.

Il centralise aussi la demande de déplacement de tous les agents afin que deux agents ne puissent pas occuper en même temps la même position.

Il contient l'IHM de contrôle du jeu.

On obtient **l'architecture** physique suivante (ceci n'est pas un diagramme de communication) :



Remarque :

Dans ma correction toutes les JVM seront exécutés sur une même machine grâce à la propriété de RMI qui permet une indépendance de la localisation des composants.

2. Enoncé de l'exercice

1/ Faire le diagramme de communication de ce SI.

2/ Faire les diagrammes de classes de chacun des composants logiciels en mettant en évidence les Designs Patterns utilisés :

- IHM JOUEUR (1 par joueur)
- POPULATION JOUEUR (1 par joueur)
- IHM GRILLE (1 par joueur)
- SERVEUR JEU (unique)

Il y a donc 4 diagrammes de classes à produire.

Pour faire le diagramme de communication, il nous faut faire un choix d'architecture bien connu des architectures réparties.

Les agents sont répartis dans chacune des populations. Et chaque agent contient sa localisation dans la grille (attributs x, y).

Chaque joueur et l'IHM de visualisation ont besoin de connaître la position de tous les agents. Pour cela on a 2 possibilités :

- Interroger toutes les populations de chacun des joueurs pour obtenir la position des agents (attributs x et y de chaque Agent)
- Mémoriser la position de chaque agent dans une grille physique contenu dans le serveur. Le serveur centralise donc la position de tous les agents.

Dans la 1^{ère} solution, la grille est virtuelle. Optimisation mémoire mais performance moindre. Nécessite de connaître la liste des populations.

Dans la 2^{ème} solution la grille est réelle. Duplication de l'information des positions mais performance plus grande. Les populations sont indépendantes.

Nous faisons le choix de la 2^{ème} solution. L'espace de Tore déjà utilisé pour centraliser le déplacement de chaque agent afin que deux agents ne puissent occuper en même temps la même position, sera aussi utilisé pour centraliser la position de tous les agents dans une grille NxM. Chaque case contiendra la couleur (optimisation) et la référence à l'agent correspondant pour pouvoir décrémenter son nombre de vie.

3. Pour démarrer le codage (optionnel)

Il ne vous est pas demandé de réaliser le codage de ce SI. Mais si quelqu'un voudrait s'y lancer voici un petit codage permettant de montrer comment on peut coder le comportement d'un agent dans un espace de tore.

De plus, cela permet de montrer une propriété forte des algorithmes de ce type que l'on retrouve dans les [Systèmes multi-agents](#) :

L'originalité de la démarche est que bien que chaque agent de même nature ait la même politique de déplacement, on peut s'apercevoir que dans la vue d'ensemble ces agents semblent agir avec une certaine cohérence.

On appelle cela l'**émergence**. *"L'émergence est un concept qui intervient lorsque des systèmes simples interagissent en nombre suffisant pour faire apparaître un certain niveau de complexité qu'il était difficile de prévoir par l'analyse de ces systèmes pris séparément. C'est un phénomène qu'on trouve dans les domaines physiques, biologiques, écologiques, socioéconomiques, linguistiques et autres systèmes dynamiques comportant des rétroactions."* (Wikipédia)

Pour démarrer **le codage** de cet exercice, vous avez le code :



Disponible sur le site : **Exercice01_02_CodePourDemarrer**

Le lanceur : **runTesterGrilleIHM.bat** affiche une grille IHM dans laquelle des "agents" se déplacent dans un espace de Tore.

Le code des modes de déplacement de cet agent sera la base du développement des algorithmes de déplacement de vos agents dans le cadre de cet exercice.

