BARSONI Dorian

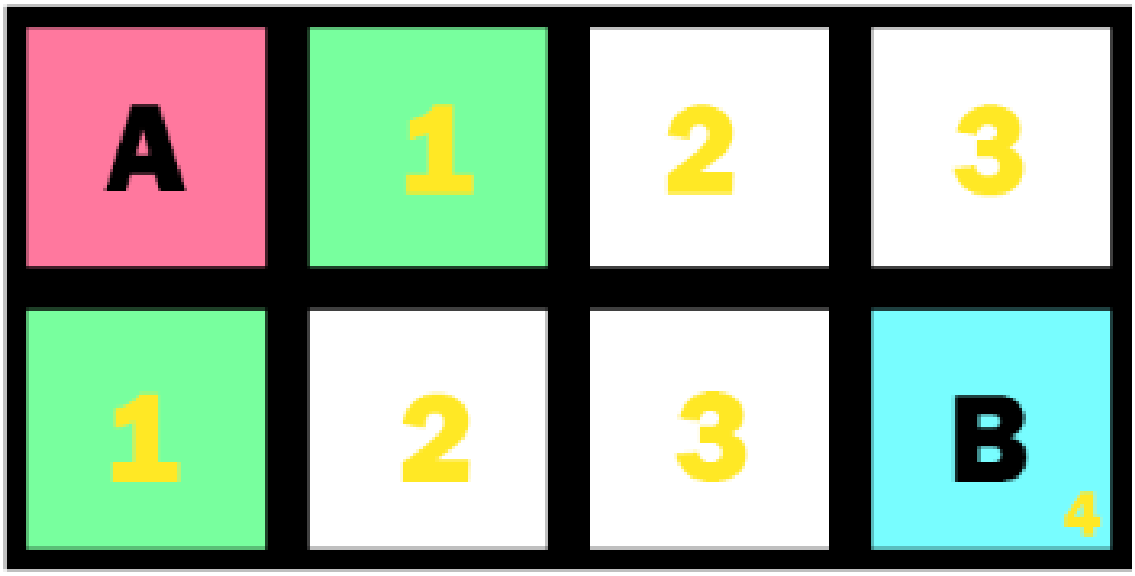
Projet Info4A : Partie 3

Dans la partie 3 du projet nous avions pour objectif d'implémenter les méthodes de déplacement des robots prédateur et proie. Ce compte rendu a pour but de vous détailler la manière dont je m'y suis pris. Nous verrons tout d'abord la description de l'algorithme du robot prédateur, puis ceux du robot proie. Enfin nous terminerons par une évaluation de ces algorithmes à l'aide de la méthode évalue.

1) Robot Prédateur

Concernant le robot prédateur la méthode que j'ai utilisé est celle décrite dans le document PDF du sujet, direct prédateur. Le robot prédateur a pour objectif de réduire la distance entre lui et le robot proie. On s'y prend donc de la manière suivante :

- On stock dans un tableau les cases blanches autour du robot prédateur
- On calcule le distMin pour chacune des cases
- On sélectionne la case blanche avec la valeur de distMin minimale, si 2 cases ont une valeur égale alors on en prend une aléatoirement
- On déplace le robot sur ladite case
- Un fois que le robot prédateur se retrouve sur la même case que le robot proie on retourne false



Ici le point A représente le robot prédateur et le point B le robot proie. Les cases vertes sont les cases de déplacement possible pour le robot prédateur. Étant donné que les 2 on un distMin égal l'algorithme direct prédateur choisira une des cases vertes aléatoirement.

BARSONI Dorian

Projet Info4A : Partie 3

2) Robot Proie

Concernant le robot proie j'ai tout d'abord implémenté l'algorithme random proie décrit dans l'énoncé du sujet du projet. Tout bêtement le robot proie se déplace aléatoirement sur une des cases blanches autour de lui.

J'ai ensuite développé un second algorithme appelé smart proie, le but étant de créer un algorithme plus performant que random proie. Une première idée et que j'ai implémenté était que le robot proie devait se déplacer sur la case maximisant la distance entre lui et le robot prédateur. Cependant comme on pouvait s'y attendre le robot proie se retrouve très souvent dans un endroit ou il n'y a aucune case blanche l'éloignant du robot prédateur. Il se retrouve donc à se déplacer sur 2 cases jusqu'à ce que le robot proie finisse par l'attraper.

J'ai donc tenté d'améliorer ce principe. Pour ce faire j'ai également calculé l'ID minimisant la distance avec le robot prédateur. Puis j'ai fait ceci :

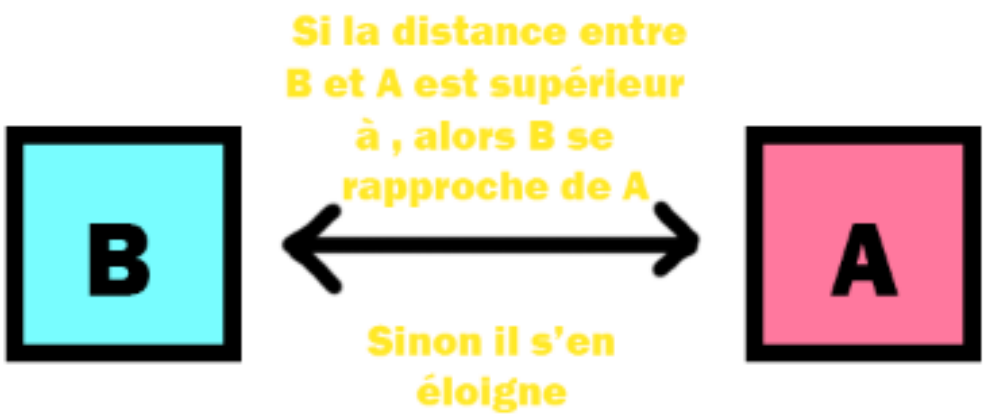
- Si la distance minimale entre le robot prédateur et le robot proie est inférieur à 15, alors le robot proie se rapproche du robot prédateur
- Sinon il s'en éloigne

En faisant cela on évite que le robot proie ne reste bloqué au début du labyrinthe et on peut constater qu'une course entre les 2 robots s'installe au bout d'un certain temps. Cependant cette solution n'est pas suffisante car il arrive très souvent que le robot proie se coince ailleurs dans le labyrinthe. Il arrive également que les 2 robots soient sur des cases en diagonales et répètent leurs mouvements de manière infini, l'un tentant de se rapprocher, l'autre de s'enfuir.

Malheureusement je n'ai pas trouvé de solution à ces 2 problèmes. L'une des idées que j'avais eu était de vérifier au préalable si l'ID de déplacement du robot proie dans 2 tours était la même que l'ID où il se situait actuellement (pour contrer la répétition des déplacements). Cette méthode ne fonctionnait pas non plus étant donné que certes, le robot, s'il détecte une répétition, allait alors l'éviter. Mais une fois cela fait il allait se redéplacer vers les 2 cases où il répète un mouvement infini, ce qui fait qu'au lieu d'avoir une répétition des déplacements sur 2 tours on en aurait eu une sur 3 tours.

BARSONI Dorian

Projet Info4A : Partie 3



3) Evaluation des algorithmes

Les tests suivants ont été effectué pour 100 courses dans les 4 descriptions de labyrinthes proposés dans le fichier trame.cpp avec un timeout de 100 dans une configuration initiale où les 2 robots se situent sur des cases blanches aléatoires dans le labyrinthe.

	Random Proie	Smart Proie
Labyrinthe 1	20	46
Labyrinthe 2	30	101
Labyrinthe 3	18	101
Labyrinthe 4	22	32

On remarque donc que globalement l'algorithme Smart Proie se débrouille mieux que l'algorithme Random Proie. On voit également quand pour les Labyrinthes 2 et 3 Smart Proie permet, en grande majorité, au robot proie de ne pas se faire attraper par le robot prédateur avant la fin du temps imparti. Cependant on se doute que la majeure partie des cas est du au fait que les 2 robots se soient retrouvés sur 2 cases en diagonales l'un de l'autre et qu'ils ont bouclés jusqu'à épuisement du temps.

Smart Proie est donc plus performant que Random Proie mais est loin d'être optimal. On peut également remettre en cause le fait que le robot prédateur ne cherche qu'à minimiser sa distance entre lui et le robot proie. Il est de ce fait également responsable de la répétition des déplacements.

BARSONI Dorian

Projet Info4A : Partie 3

Conclusion :

Dans cette partie j'ai programmé les déplacements des robots prédateurs et proie. Le robot prédateur n'a comme algorithme de déplacement que l'algorithme direct prédateur, très rudimentaire, qui minimise la distance entre lui et le robot proie. Le robot proie comme algorithme de déplacement random proie qui se déplace aléatoirement sur une des cases blanches autour de lui. Il possède également l'algorithme smart proie qui s'approche du robot prédateur si la distance qui les sépare est supérieure à 15 et qui sinon s'en éloigne.

Nous avons pu voir que globalement smart proie produisait de meilleurs résultats que random proie, cependant smart proie a tendance à induire une répétition des mouvements du robot proie conduisant soit à sa capture, soit à une boucle des déplacements des 2 robots. On peut également rejeter la faute sur les déplacements rudimentaires que produit direct prédateur.