Filière Informatique

Rapport projet d'algorithmique de mobilité

Membres du groupe :
Tanguy Pemeja
Maxime Rosay

Encadrant :
Arnaud Casteigts

Version du 22 décembre 2019

1 Présentation du sujet

Le sujet se présente sous la forme d'une archive où l'on retrouve une petite fourmilière devant récolter de la nourriture pour ne pas s'éteindre. Les fourmis ne sont dotés d'aucune forme d'intelligence et se déplacent aléatoirement dans une zone prédéfinie.
L'objectif de ce projet est donc de mettre en place diverses stratégies algorithmiques afin de permettre aux fourmis d'acquérir une certaine intelligence pour ramener de la nourriture de manière optimisée. Il y a aussi quelques contraintes pour compliquer cette récolte : les blocs possèdent une dureté proportionnelle à leur profondeur et doivent donc être cassé avant de pouvoir se déplacer ;
Les trois points selon lesquels notre implémentation sera évaluée sont les suivants :

— Le temps de survie de la fourmilière
— La clarté du code et de ce rapport
— Les idées mises en oeuvre

2 Stratégie algorithmique mise en place

Dans cette partie, nous présenterons quelles sont nos idées mises en place afin d'optimiser la récolte de nourriture, et ainsi permettre à la fourmilière de survivre indéfiniment.

2.1 Connaissez-vous Minecraft ?

En se basant sur le fait que la reine peut apparaître seulement dans la partie supérieure de la carte et que la nourriture se trouve plus facilement dans la partie inférieure cette fois-ci, nous avons effectué un parallèle entre la nourriture des fourmis et les diamants de Minecraft.
Dans le jeu Minecraft, la ressource la plus convoitée est le diamant et se trouve aux couches basses du monde. Afin d'augmenter ses chances d'en trouver, il existe une stratégie dite *minage optimisé* permettant aux joueurs de couvrir la quasi-totalité des blocs pour vérifier si cette précieuse ressource ne s'y trouverait pas.
Nous avons donc pris l'initiative d'adapter cette fameuse technique à ce projet et se décompose en plusieurs étapes :

1. Au spawn de la fourmi (sur la reine), elle détecte si un tunnel a été creusé à gauche et/ou à droite de la reine.
2. Si aucun ou seul un tunnel a été creusé, elle décide de creuser son propre tunnel, en prenant celui de gauche en priorité. Puis effectue les actions suivantes :
   (a) Creuse le bloc, puis se déplace sur ce dernier
   (b) Pose une phéromone [1] de type *tunnel*
   (c) Continue tout droit jusqu'à buter sur le bord de carte où elle fera des aller-retours entre la phéromone *tunnel*, l'incrémentatant de 0.1 à chaque passage, et ce bord en question

1. issue de la classe PheroNode

3. Si il existe déjà deux tunnels, alors la fourmi descend trois blocs plus bas et recommence à l'étape 1 en remplaçant la reine par sa position actuelle.
4. Une fois le bord de carte inférieur touché, la fourmi repart en sens inverse

Cette stratégie permet la couverture la plus large possible du terrain sachant que les fourmis détectent la nourriture à au plus un bloc d'elles.

Afin d'éviter plusieurs bugs il faut dissocier bon nombre de cas et mettre des priorités sur chacunes de ces situations. Par exemple, quand une fourmi détecte de la nourriture grâce à la méthode `onSensingIn` il faut :

— Soit rediriger la fourmi vers ce bloc contenant de la nourriture (et le creuser si besoin) puis
   1. Si la fourmi se trouve dans un tunnel, ramener la nourriture vers la colonne principale qui se fait en deux étapes : retourner sur la case contenant la phéromone *tunnel* et jeter la nourriture d'une case vers l'avant
   2. Sinon, retourner vers la reine puisque la fourni n'est pas affectée à un tunnel (dans ce cas-ci, la fourmi se trouve dans la colonne principale)

— Soit ne rien faire lorsque la fourmi est en train de creuser son tunnel, dans le cas contraire on perdrait l'information que la fourmi s'est affectée à un tunnel et cela désorganiserait toutes les autres

Afin de mieux comprendre notre projet, voici ce à quoi correspondent les nouvelles images :

FIGURE 1 – Légende des images

2.2 Problèmes connus

Tout d'abord, nous avons constaté un autre bug dans notre code peu impactant sur son fonctionnement mais empêchant un visuel réaliste. En effet, les fourmis héritent de la classe `CellLocatedNode` qui elle-même hérite de la classe `WaypointNode` [2] et donc ces dernières possèdent un champ `speed` indiquant leur vitesse de déplacement. Dans notre code, si ce champ est fixé à une valeur inférieur strictement à 23, les fourmis vont se retrouver bloquées à une certaine profondeur, ne pouvant ni casser un nouveau bloc, ni se déplacer.
Plus généralement, si une fourmi possède une `speed` n alors elle peut se déplacer librement de la profondeur 1 à n et se retrouvera bloquée sur la profondeur $n+1$ exactement. Par exemple, avec une `speed` intialisée à 7, nous obtenons le résultat ci-dessous [3] :

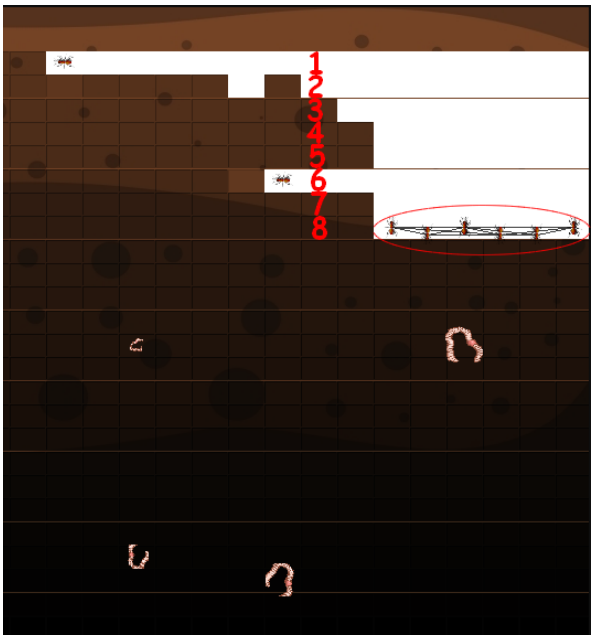

FIGURE 2 – speed à 7

Nous avons dû faire face à un autre problème celui de la détection de la nourriture présente au milieu par les fourmis se trouvant dans les tunnels. Pour empêcher que les fourmis d'une même ligne prennent en permanence la nourriture qu'elles venaient de poser, nous avons dû mettre en place un deuxième système de phéromone : *des phéromones queen*. Ces phéromones sont posées dans la colonne du milieu en même temps que la nourriture et empêche les fourmis dans les tunnels de prendre la nourriture posée dessus.

2. Dans notre implémentation
3. à noter que cette image a été prise au début de notre développement

FIGURE 3 – Nourriture posée sur une phéromone queen

Dernier problème et non des moindres, la détection de nourriture proche à côté de la phéromone tunnel. Lorsqu'une fourmi détectait de la nourriture depuis la phéromone tunnel celle-ci se bloquait dans la colonne du milieu à cause de notre fonction pour faire déposer la nourriture au milieu. Afin de corriger ce problème, nous avons dû empêcher la détection de nourriture par les fourmis ayant un tunnel lorsqu'elles sont sur la phéromone *tunnel*. Ainsi si de la nourriture apparaît à cet endroit, celle-ci va être récupérée non pas directement par la fourmi mais après avoir fait un aller-retour dans la galerie. Cette solution n'est bien sûr pas optimale mais a permis de fixer un problème majeur.

2.3 Quelques résultats

Nous avons testé notre algorithme suivant différents paramètres tels que la durée de vie des fourmis, des phéromones, le coût des blocs ainsi que le nombre initiale de nourriture à 25. Dans un premier temps, nous avons choisi d'utiliser des coûts de blocs variant suivant la profondeur car cela nous paraît plus réel. Mais ce choix n'affecte en rien notre algorithme et augmente juste la difficulté de la colonie à survivre. Contrairement à la durée de vie des phéromones qui dans notre algorithme doit être supérieure à celle des fourmis car sinon celles-ci pourraient ressortir des tunnels et causer des bugs dans notre algorithme. Malgré tout, nous avons décidé de fixer une valeur inférieur [4] afin de, comme on le verra plus tard, optimiser la fourmilière à long terme. Dans la suite, nous allons évoquer des résultats obtenus avec une durée de vie des fourmis allant de 1000 à 2000 tour d'horloge. Nous avons pu observer 2 cas possibles.

— Soit la configuration n'est pas assez favorable au début pour que la colonie survive et celle-ci meurt rapidement.

4. Dans notre implémentation, les phéromones décrémentent de 0.1 tous les 500 tours d'horloge alors qu'il faudrait mettre 2000 afin d'obtenir une version stable et sans bug

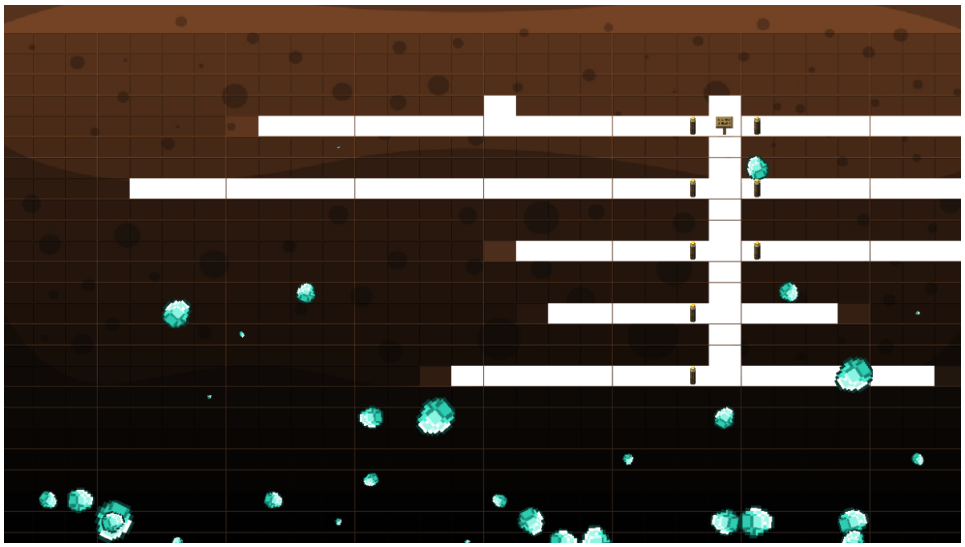

FIGURE 4 – La colonie meurt au début

— Soit la colonie survie au début, commence à creuser quelques galeries en profondeur et survie longtemps voir indéfiniment. En effet après 2h, temps auquel nous avons stoppé l'expérience, la colonie était toujours vivante

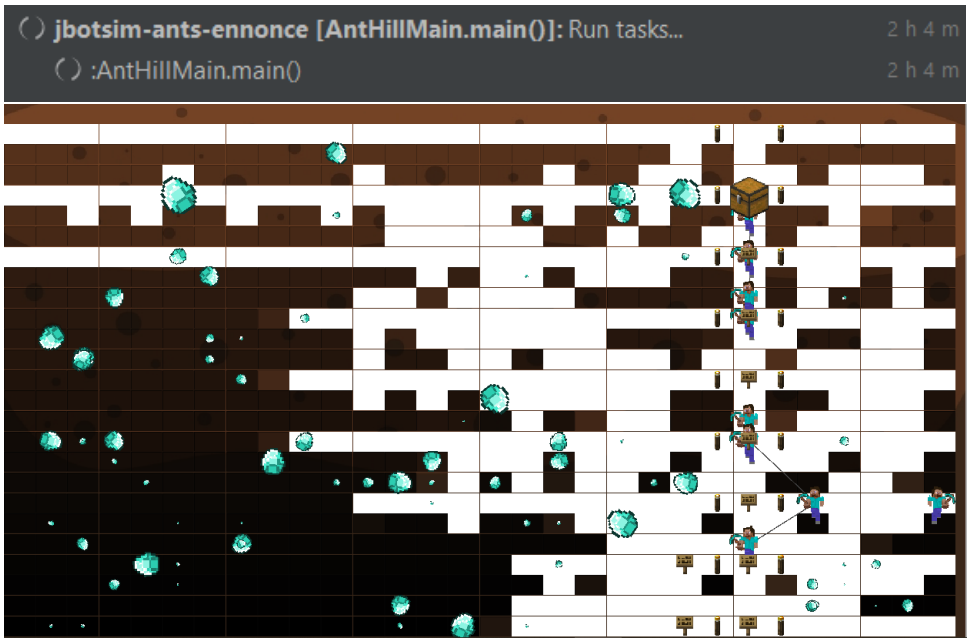

FIGURE 5 – La colonie continue de survivre

3 Pour aller plus loin..

Ici, nous allons présenter les idées nous semblant pertinentes mais que nous n'avons pas réalisé.

3.1 Rentrer dans des tunnels déjà occupés

Pour améliorer le fonctionnement de la fourmilière quand celle-ci a déjà commencé tous les débuts de galerie, nous avons cherché à réduire le temps de vie d'une phéromone tunnel. Mais nous avons pu observer qu'une valeur égale à la moitié de la vie d'une fourmi pouvait entraîner des bugs de notre algorithme contrairement aux cas où la valeur est plus élevée. Dans la première phrase de notre algorithme notamment quand les tunnels doivent se creuser et que la fourmi met du temps à revenir à sa phéromone. Mais dans la deuxième partie, cette valeur se révèle plus intéressante car les fourmis deviennent naturellement plus créatives et vont plus rapidement dans les tunnels laissés à l'abandon.
Afin de pallier à ces bugs, nous pouvons imaginer garder un temps de vie élevé pour les phéromones mais ajouter un système afin que les fourmis puissent rentrer dans le tunnel même si celle-ci contient une phéromone à son entrée. En effet après un certain temps, quand les fourmis commencent à mourir dans leurs galeries, les tunnels mettent un temps assez long avant de redevenir accessibles par les autres fourmis. Ce temps pouvant aller jusqu'à 10 fois l'espérance maximale d'une fourmi, il est nécessaire de trouver une solution afin d'améliorer le fonctionnement de la fourmilière en général. Une solution possible serait de déterminer lors d'un aller-retour dans la colonne du milieu, la phéromone ayant la plus petite valeur. Ainsi si la fourmi fait un aller-retour cela signifie que toutes les galeries ont déjà commencé à être creusées. Donc il n'est pas nécessaire d'attendre en faisant des allers-retours dans la colonne principale et perdre de l'espérance de vie sans n'avoir rien fait. Le fait d'envoyer une fourmi dans un tunnel ayant encore une phéromone pourrait peut-être permettre d'aider une fourmi à creuser le tunnel et récupérer de la nourriture mais surtout ne pas faire perdre du temps à la fourmilière.

3.2 Communication inter-fourmis

Dans notre implémentation, la seule forme de communication entre les fourmis résident dans les phéromones. En effet, avec les `Node` il est possible de créer un réseau et d'y faire circuler des messages. Nous pourrions alors imaginer que les fourmis s'échangent des des messages afin de creuser plus rapidement des galeries ou encore récolter toute la nourriture d'un spot. Mais cela complexifierait encore plus notre stratégie puisqu'il s'agirait de mettre d'autres priorités pour prendre en compte les messages reçus selon leur type.

3.3 Battle Royale entre fourmilières

Une dernière idée, plus originale, serait de faire apparaître non pas une mais deux reines fourmi avec comme seul objectif : décimer la reine adverse tout en essayant de survivre. Émanerait alors de nouvelles stratégies beaucoup plus avancées que celle du minage optimisé, il faudrait créer des groupes de fourmis. En effet, la mise en place de petit groupe permettrait d'effectuer diverses actions

sur la carte : récolter de la nourriture ou bien essayer de trouver la reine adverse. Bien évidemment si deux fourmis se croisent en étant ennemies, elle peuvent se tuer.