TIPE : Usage de l'Algorithmie Génétique dans la résolution d'un problème de logistique

0. Introduction :

Dans ce TIPE, on commence par établir une épreuve simple de transport de matériaux et on désire créer une intelligence artificielle capable de passer cette épreuve. Pour cela, on utilise un algorithme génétique. L'objectif est donc d'identifier et étudier les stratégies produites par cette méthode et de les comparer avec les stratégies intuitives que peut imaginer un être humain.

Tout sera fait via une simulation informatique où des agents nommés fourmis modéliseront des travailleurs.

1. Les règles du jeu :

On invente un jeu tout simple. Des fourmis sont situées sur une grille où elles ne peuvent se déplacer que d'une case à chaque tour et ce, pendant un nombre donné de tours. Leur objectif est de se déplacer jusqu'à une source de nourriture puis de revenir à leur nid pour y déposer la nourriture. Prendre et poser de la nourriture n'est pas du ressort de la fourmi et a lieu automatiquement quand la fourmi se trouve sur un nid ou sur une source. Chaque fourmi ne transporte qu'une unité de nourriture à la fois et les sources de nourriture ont un nombre limité d'unité à fournir avant de se tarir. Deux fourmis ne peuvent occuper la même case. Une fourmi qui décide d'avancer sur une case déjà occupée par une de ses congénères ne se déplace pas, cependant elle vole l'unité de nourriture de sa cible si elle en possède. [1]

Ces règles tentent de modéliser de façon simplifiée un problème de logistique où un groupe de travailleurs devraient par exemple transporter du matériel de plusieurs sources vers une zone spécifique de manière optimale. Dans une situation comme celle-ci, on veut pouvoir donner des instructions simples aux travailleurs afin de pouvoir faire faire le travail par des robots, pour préparer l'arrivée de l'homme sur Mars par exemple.

2. Les fourmis :

Chaque fourmi possède un code génétique. Le code génétique détermine le comportement de la fourmi. Chaque fourmi est une fonction :

$$\vec{D} = \sum_{i \in I} (a_i * \|\vec{d}_i\|^2 + b_i * \|\vec{d}_i\| + c_i) * \left(\frac{\vec{d}_i}{\|\vec{d}_i\|}\right)$$

Où D est la direction de la fourmi, les d_i sont les vecteurs allant de la fourmi vers l'objet i du terrain et les a,b et c sont les coefficients du code génétique de la fourmi.

Chaque tour, on calcule la direction de la fourmi grâce à cette formule puis on l'a fait avancer dans cette direction.

La seule différence entre deux fourmis est leur code génétique. Ce qui nous intéresse c'est justement ce code génétique. C'est lui que l'on sélectionnera et fera évoluer.

Les codes génétiques sont composés de 60 nombres entre 0 et 500. Ainsi il est facile de créer aléatoirement une fourmi. Il est également facile de faire muter le code d'une fourmi en modifiant un seul nombre dans son code.

3. La sélection :

Au tout début, on crée une population de fourmis aux codes aléatoires. C'est la première génération.

On fait jouer chaque fourmi plusieurs fois et on lui attribue un score pour sa

performance. Apporter de la nourriture au nid rapporte un grand nombre de point et se cogner au mur en enlève un petit nombre par exemple.

On cherche ensuite à constituer la population de la génération suivante. Pour cela, on sélectionne les fourmis ayant le meilleur score. Le but étant d'obtenir des fourmis faisant les scores les plus élevés possibles. On ne garde que les n meilleurs fourmis, on fait k copies de ses fourmis avec de légères modifications et on complète la population avec des fourmis au code entièrement aléatoire. Les paramètres n et p sont choisis avant chaque expérience par nous-même.

On recommence le processus de sélection autant de fois que l'on souhaite. On observe ensuite la dernière génération. On attend des résultats meilleurs dans les dernières générations et c'est effectivement ce que l'on obtient. [2]

4. Première tentative : la compétition

On commence avec des populations de 8 individus à chaque génération. Les 8 fourmis jouent en même temps sur la même grille et sont en compétition les unes avec les autres. Chacune possède son propre nid.

On fait évoluer les fourmis pendant 5000 générations environ. On observe que la stratégie émergente est tout simplement de courir vers la nourriture et de revenir droit sur le nid. Même si cela signifie s'exposer au vol. C'est une stratégie qu'un être humain aurait pu imaginer lui-même.

D'autres stratégies peuvent émerger mais sont vite étouffées. La plus courante consiste à patrouiller autour de son nid et voler les fourmis passant à proximité. Cette stratégie est valable mais pourtant désavantagée par le système de sélection. En effet, quand la stratégie du voleur devient majoritaire, plus aucune fourmi ne va chercher de la nourriture et donc il n'y a plus personne à voler. Le problème étant qu'après avoir été sélectionné, le code génétique du voleur est disséminé dans toute la génération suivante, l'empêchant d'être viable plus d'une génération.

On a affaire à un problème qu'il faudra résoudre dans les prochaines tentatives. Un code génétique très performant dans une situation peut être très mauvais dans une autre.

5. L'importance des paramètres :

On remarque très vite l'importance des paramètres dans la progression des fourmis. Bien entendu, le nombre de générations a une énorme influence sur le résultat final. Mais d'autres paramètres entrent en compte.

L'attribution des points est également d'une grande importance. C'est le seul moyen que l'on possède pour guider l'évolution des fourmis. On souhaite que les fourmis transportent la nourriture des sources vers le nid. Pour cela, on attribue 50 points à chaque fois qu'une fourmi ramasse de la nourriture et 100 points quand elle la dépose dans le nid. Cela suffit à mener les fourmis à transporter la nourriture comme on le souhaite.

On peut cependant accélérer l'évolution en apportant quelques modifications. On constate en effet que beaucoup de fourmis perdent du temps à tenter de sortir des limites du terrain. Même si tenter de sortir est déjà pénalisant pour la fourmi car cela lui fait perdre un tour, on peut décider de lui enlever encore un point. Ainsi chaque collision avec le bord retire un point à la fourmi. Cela peut paraître anodin mais en rendant le critère de notation plus sévère, les fourmis évoluent plus rapidement.

Il faut malgré tout rester prudent. Dans certains cas, les fourmis peuvent rester paralysées dans leur progression. Quand chaque fourmi perd 1000 points par collision, la priorité devient d'échapper au bord à tout prix, au détriment de la recherche de nourriture.

D'autres cas plus surprenants peuvent arriver. Des fourmis peuvent découvrir une faille dans le système de points et l'exploiter. Nous avons ainsi été témoins de fourmis qui exploitaient des bugs du programme qui permettait de dupliquer la nourriture. Même avec ce bug corrigé, certaines fourmis tiraient partie du système de notation d'une manière originale.

Lors de notre deuxième tentative, nous avions attribué 1 points à chaque échange de nourriture entre deux membres d'une même équipe afin de favoriser la coopération et la formation de relais. Il est arrivé qu'une équipe ne se consacre qu'à l'échange de nourriture. Au lieu d'aller chercher les sources plus

lointaines, les fourmis restaient immobiles les unes à côté des autres pour se passer la même unité de nourriture sans fin. En effet, deux fourmis qui donnent et reçoivent la même unité de nourriture rapportent 2 points chaque tour, ce qui est une source sûre et inépuisable de points. Mais c'est aussi une stratégie que l'on ne souhaite pas voir apparaître car elle n'est d'aucune utilité dans la pratique.

6. Le problème de la spécialisation :

En jouant sur les différents paramètres on observe un phénomène problématique. Avec suffisamment de générations, les fourmis finissent par apprendre par cœur le terrain. Elles deviennent donc imbattables dans une situation très spécifique mais complètement inaptes dans toute autre.

Ce résultat est intéressant. On pourrait par exemple résoudre chaque problème en générant assez de générations pour avoir un code génétique parfaitement adapté au problème en question. C'est un processus assez long. Pour chaque problème possible il faudrait refaire plusieurs milliers de générations ce qui n'est pas optimal.

Le résultat que l'on voudrait obtenir est un code génétique polyvalent. Une stratégie qui pourrait répondre à plusieurs situations sans forcément atteindre un score optimal mais en ayant toujours des résultats corrects. C'est ce que l'on est en droit d'attendre d'une intelligence artificielle : l'adaptabilité.

Ainsi, on fait le choix de rendre le terrain aléatoire. Les sources de nourriture sont réparties au hasard avec des quantités tantôt élevées tantôt faibles. On devra donc tester les fourmis sur un grand nombre de terrains afin d'avoir une idée de leur niveau moyen. Par la loi des grands nombres, plus on fait jouer de parties à une fourmi, plus la moyenne de ses scores est proche de son espérance. On espère ainsi obtenir des fourmis plus polyvalentes.

7. Deuxième tentative : la coopération

On décide de former des équipes. Il y aura désormais 16 équipes, chacune comportant 5 fourmis. Chaque équipe jouera seule. Le nid est situé dans un coin de la grille et de la nourriture est disposée aléatoirement sur le reste du terrain. Chaque équipe a affaire à ce terrain aléatoire à tour de rôle. Il n'y a jamais deux équipes qui jouent en même temps.

La grande difficulté ici est la coopération. Quand il y a 5 fourmis sur le terrain avec le même objectif, il est très courant que les fourmis se gênent mutuellement.

Comme le terrain est maintenant aléatoire, les fourmis progressent bien plus lentement. Beaucoup d'équipes sont finalement très douées dans un cas et moyennes dans les autres. Certaines sont perdues en cas d'abondance et d'autres ont du mal à composer avec la pénurie.

Cependant, certaines stratégies sortent du lot. Une première tactique consiste à laisser une des fourmis de l'équipe sur le nid. C'est la technique de l'intendant qui permet d'éviter les embouteillages sur le nid. L'intendant récupère la nourriture de toutes les fourmis passant à côté du nid et les pose directement, servant de relais. Cette technique est souvent abandonnée par les dernières générations parce qu'elle prive l'équipe d'un de ses membres.

La stratégie la plus courante est très intuitive. Il s'agit de mettre en place un roulement. Les fourmis vont en ligne droite vers la nourriture et reviennent vers le nid en contournant le chemin aller afin de ne gêner personne. Cette stratégie est très efficace et donne parmi les meilleurs résultats. Les équipes l'adoptant parviennent souvent à récupérer toute la nourriture disponible et obtenir un score maximal.

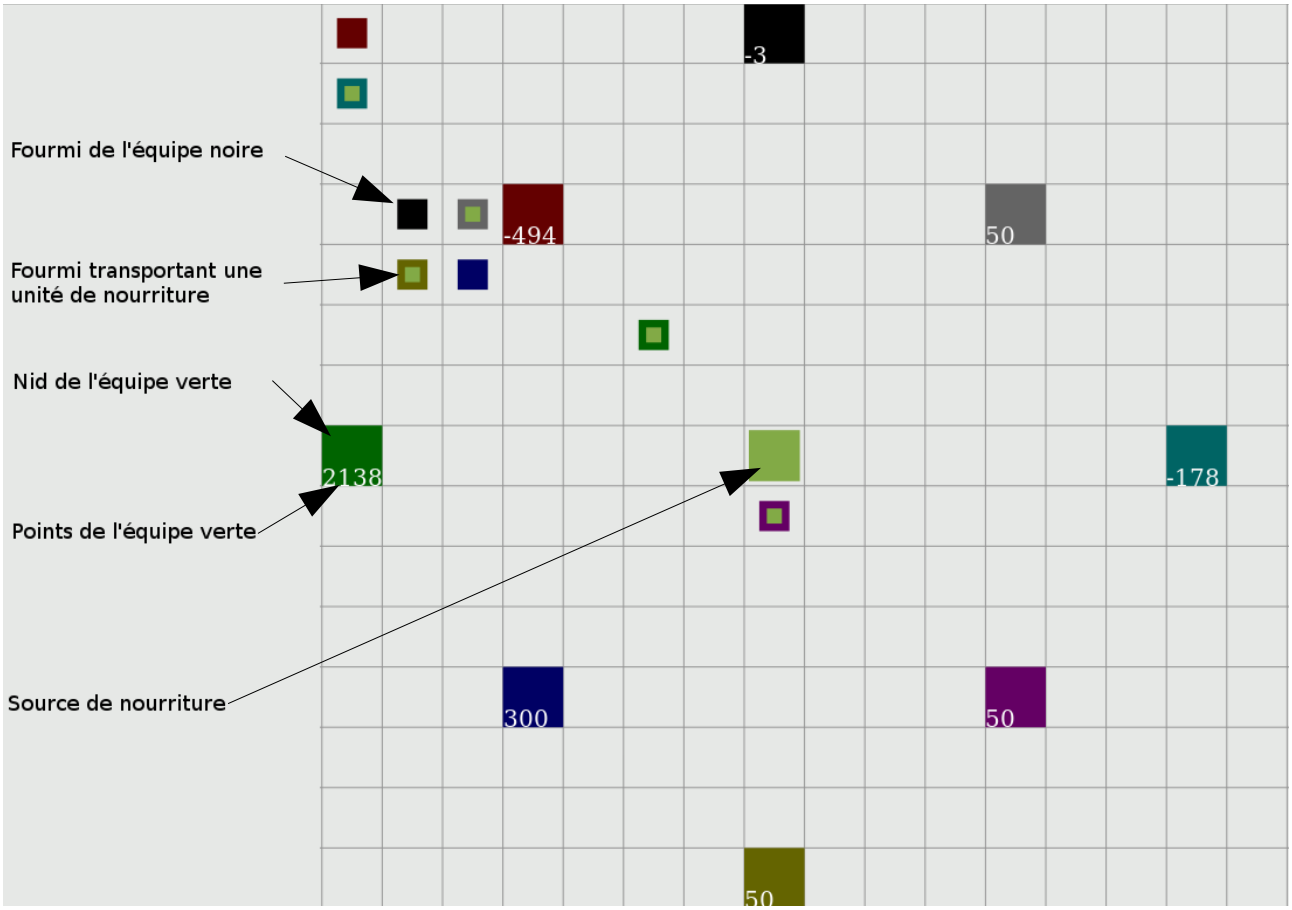
Une stratégie plus originale émerge régulièrement et obtient des résultats similaires dans la plupart des cas. L'équipe de 5 fourmis se sépare en deux groupes. 4 individus restent ensemble et une autre fourmi se tient à l'écart. Le groupe est peu mobile mais peut transporter 4 unités de nourriture et exploite les sources proches jusqu'au bout. La fourmi solitaire rapporte moins de points mais va chercher des sources de nourriture que le groupe n'aurait jamais découvertes. On a là un cas d'équilibre collectif entre les suiveurs et les

explorateurs. Il s’agit d’arriver à un juste milieu entre ceux qui restent sur place pour exploiter les sources déjà découvertes et ceux qui partent en chercher de nouvelles. Ce dilemme est étudié par des chercheurs comme James G. March dans *Exploration and Exploitation in Organizational Learning.* Dans ce cas précis, l’algorithmique génétique nous a appris qu’il fallait 1 explorateur pour 4 suiveurs.

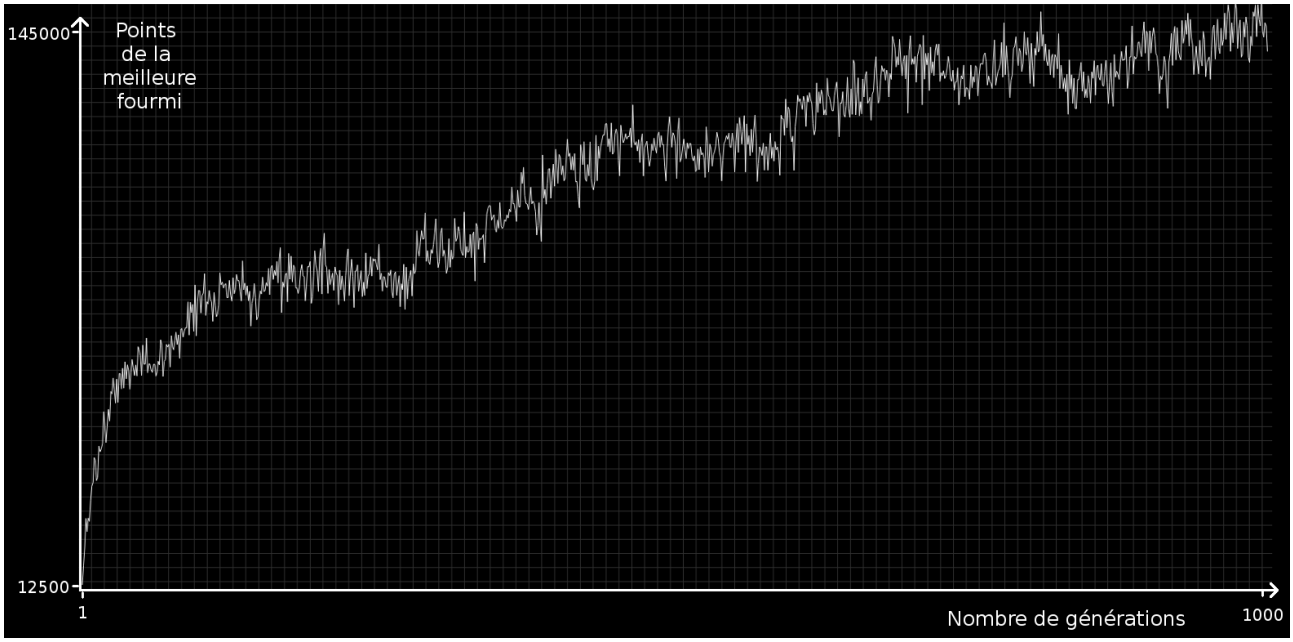
Annexes

Illustrations :

[1] Capture d'écran de la grille de jeu



[2] Courbe du nombre de points par génération. Les fourmis étaient par équipe de 4 et en compétition avec 7 autres équipes.



Code source :

Le code source est un peu long. Il est en intégralité sur github. En annexe se trouve uniquement le fichier worker.js qui s'occupe de l'algorithme génétique à proprement parler.

Les trois autres fichiers sont start.js, draw.js et saveAs.js qui servent à l'interface avec l'utilisateur.

Il y a aussi une version alternative du programme qui est appelée fourmis II et qui est très semblable au niveau du programme. La seule différence est la façon dont les fourmis prennent des décisions, avec un champ de vision réduit et beaucoup plus de coefficients. Mais les résultats n'étaient pas concluants du tout.

worker.js *la partie principale*

```
// Initialisation des variables utiles
let places = [{taille:15,
              fourmis:[[7,0,0],[3,3,0],[0,7,0],[3,11,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0],[7,0,0],[3,3,0],
[0,7,0],[3,11,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0]],
              coor:[[7,0,0],[3,3,0],[0,7,0],[3,11,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0],[7,0,0],[3,3,0],
[0,7,0],[3,11,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0]],
              nourriture:[[6,6,1],[7,6,1],[8,6,1],[6,7,1],[7,7,50],[8,7,1],[6,8,1],[7,8,1],[8,8,1]],
              nourrRaf:[],
              ballon:[7,7,0,0],
              usine:[],
              team:8,
              members:2},
             {taille:15,
              fourmis:[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],        // 1 Cette map
est destinée à des vagues de tailles 1 pour 1 à 5 fourmis par équipe.
              coor:[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
              nourriture:[[14,14,10],[13,14,10],[14,13,10],[13,13,10]],
              nourrRaf:[],
              ballon:[1,8,0,0],
              team:8,
              usine:[[7,7,0],[7,8,0],[8,7,0],[6,7,0],[7,6,0]],
              members:5},
             {taille:41,
              fourmis:[[20,0,0],[0,20,0],[20,40,0],[40,20,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0]],
              coor:[[20,0,0],[0,20,0],[20,40,0],[40,20,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0]],
              nourriture:[[19,19,10],[20,19,10],[21,19,10],[19,20,10],[20,20,50],[21,20,10],[19,21,10],
[20,21,10],[21,21,10]],
              nourrRaf:[],
              ballon:[7,7,0,0],
              team:4,
              usine:[],
              members:10},
             {taille:15,           // Carte avec uniquement le ballon.
              fourmis:[[7,14,0],[11,11,0],[14,7,0],[11,3,0],[7,0,0],[3,3,0],[0,7,0],[3,11,0]],
              coor:[[7,0,0],[3,3,0],[0,7,0],[3,11,0],[7,14,0],[11,11,0],[14,7,0],[11,3,0]],
              nourriture:[],
              nourrRaf:[],
              ballon:[7,7,0,0],
              team:8,
              usine:[],
              members:5},
             {taille:15,
              fourmis:[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],        // 4 Cette map
est destinée à des vagues de tailles 1 pour 1 à 5 fourmis par équipe.
              coor:[[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
              nourriture:[[14,14,10],[13,14,10],[14,13,10],[13,13,10],[0,14,10],[14,0,10]],
              nourrRaf:[],
              ballon:[7,7,0,0],
              team:8,
              usine:[],
              members:5},
             {taille:9,
              fourmis:[[5,0,0],[5,3,0],[0,7,0],[3,8,0],[7,8,0],[8,8,0],[8,7,0],[8,3,0]],
              coor:[[5,0,0],[5,3,0],[0,7,0],[3,8,0],[7,8,0],[8,8,0],[8,7,0],[8,3,0]],
              nourriture:[[6,6,1],[7,6,1],[8,6,1],[6,7,1],[7,7,50],[8,7,1],[6,8,1],[7,8,1],[8,8,1]],
              nourrRaf:[],
              ballon:[5,5,0,0],
              team:8,
              usine:[],
              members:1},
             {taille:9,
              fourmis:[[5,0,0],[5,3,0],[0,7,0],[3,8,0],[7,8,0],[8,8,0],[8,7,0],[8,3,0]], // Ce tableau 6 sert à
tester les fonctions du ballon
              coor:[[5,0,0],[5,3,0],[0,7,0],[3,8,0],[7,8,0],[8,8,0],[8,7,0],[8,3,0]],
              nourriture:[],
              nourrRaf:[],
              ballon:[5,5,0,0],
              team:4,
              usine:[],
              members:1},
```

```
              {taille:15,
               fourmis:[[0,0,0]],        // 7 Carte avec placement aléatoire.
               coor:[[0,0,0]],
               nourriture:[[2,2,1],[15,15,30],3,11],
               nourrRaf:[],
               ballon:[2,2,15,15],
               team:16,
               usine:[[2,2,0],[15,15,0],0,3],
               members:5,
               random:true}
              ];
let placeId = 7;
let waveSize = 1;
let panelTot = 1;
let echantillon = 7;

function copy(a){
    return JSON.parse(JSON.stringify(a));
}

let taillePlateau = places[placeId].taille;
let uselessBorder = 0;
let cases = 10;
let fourmis = copy(places[placeId].fourmis);
let coor = copy(places[placeId].coor);
let nourriture = copy(places[placeId].nourriture);
let nourrRaf = copy(places[placeId].nourrRaf);
let usine = copy(places[placeId].usine);
let ballon = copy(places[placeId].ballon);
let teamN = places[placeId].team;
let membN = places[placeId].members;
let bornI = 0;
let bornS = 0;


function initTerrain(id){
    if (places[id].random){
        // Cette partie est destinée à la création des cartes générées aléatoirement.
        coor = [];
        for(let i = 0; i < places[id].team;i ++){
            coor[i] = copy(places[id].coor[0]);
            places[id].fourmis[i] = copy(places[id].coor[0]);
        }

        let bout_de_nourriture;
        let valuesL = places[id].nourriture;
        nourriture = [];
        for (let i = 0;i < valuesL[2] + rnd(valuesL[3] - valuesL[2]);i++){
            bout_de_nourriture = [];
            for (let j = 0; j < 3; j++){
                bout_de_nourriture.push(valuesL[0][j] + rnd(valuesL[1][j] - valuesL[0][j]));
            }
            nourriture.push(bout_de_nourriture);
        }
        nourrRaf = [];

        valuesL = places[id].usine;
        usine = [];
        for (let i = 0;i < valuesL[2] + rnd(valuesL[3] - valuesL[2]);i++){
            bout_de_nourriture = [];
            for (let j = 0; j < 3; j++){
                bout_de_nourriture.push(valuesL[0][j] + rnd(valuesL[1][j] - valuesL[0][j]));
            }
            usine.push(bout_de_nourriture);
        }
        valuesL = places[id].ballon;
        ballon = [valuesL[0] + rnd(valuesL[2] - valuesL[0]),valuesL[1] + rnd(valuesL[3] - valuesL[1])];
    }
    else{
        // Cette partie remets en place la carte dans l'état enregistré dans places.
        coor = copy(places[id].coor);
        nourriture = copy(places[id].nourriture);
        nourrRaf = copy(places[id].nourrRaf);
        usine = copy(places[id].usine);
        ballon = copy(places[id].ballon);
    }
}

initTerrain(placeId);

let fCode = [];
let nManche = 200; // Ici on décide du nombre de manche par tour.
let actions = [];
let nAction = 0;
let pointsFourmis = [0,0,0,0,0,0,0,0];
let bareme = {"bonk":-1,"catch":20,"deliver":100,"vol":-
100,"takeBonk":2,"dropBonk":0,ballon:0,giveFactory:50,deliverR:500}; // Ici on mets les paramètres de notation.
let nGeneration = 5000; // Ici on décide du nombre de générations qui auront lieu.
let error = 7;  // Ici on décide du nombre du nombre d'erreur lors des copies.

let savedData = [];
```

```
let thingsSaved = ["",""];

onmessage = function(e){ // C'est la fonction qui est appelée quand la page a fini de s'initialiser.
    actions = e.data;    // e.data contient la liste de toutes les actions à effectuer. Elle provient du fichier
start.js.
    whatDoIDoNow();
};

function rnd(n){
    return Math.floor(Math.random()*n);
}

function init(){
    // initialise avant de commencer les tournois, crée des fourmis aux codes aléatoires pour composer la première
génération.
    fourmis = [];
    for (var i = 0; i < teamN; i ++){
        fourmis[i] = new Team();
        fourmis[i].init(membN,i);
        fourmis[i].getData(generateCode() , [coor[i][0],coor[i][1]] , 0);
    }
}

function preselect(initState){
    // Ici le worker se met à jour avec les données que lui passe la fonction start()
    // C'est utile notamment pour charger des fichiers préalablement enregistrés sur l'ordinateur.
    for (var i = 0;i < teamN;i++){
        fourmis[i] = new Team();
        fourmis[i].init(membN,i);
        fourmis[i].getData(initState[0][i], [coor[i][0],coor[i][1]] , 0);
    }
}

function generateCode(){
    // Les codes de fourmis sont sous la forme : element -> fonction
    // type étant dans l'ordre : ennemie nourriture  nid  ballon  nourriture+  usineRepos  usineActive donc on a
besoin de 14 fonctions.
    var result = [];
    for (var memb = 0;memb < membN;memb++){
        result[memb] = [];
        for (var etat = 0;etat < 14;etat++){
            result[memb][etat] = [];
            for (var type = 0; type < 5; type ++){
                result[memb][etat][type] = rnd(100)-50;
            }
        }
    }
    return result;
}

function tournoi(graph,fourmis){ // Si graph == True alors on dessine tout pour le spectateur sinon non !

    for (let groupe = 0;groupe < teamN; groupe += waveSize){
        bornI = groupe;
        bornS = groupe + waveSize;
        for (let id = bornI;id < bornS;id ++){  // On remets les fourmis sur leur nid respectif.
            fourmis[id].goToNest(places[placeId].fourmis);
            fourmis[id].looseCarry();
            fourmis[id].initPts();
        }
        //On n'oublie pas non plus de remettre les stocks de nourriture en place
        initTerrain(placeId);

        if (graph){
            if (groupe != 0) return; // On ne se fatigue pas et on ne s'occupe de dessiner que la première vague
tout simplement.
            var mancheRestante = nManche;
            var slowNumber = 5;
            var slow = slowNumber;
            console.log(fourmis);
            var manche = function(){
                if (slow == 0){
                    // Lors d'une manche, chaque fourmi agit à son tour en selectionnant une direction qui sera
executée si possible (pas d'obstacles) et ramasse/depose de la nourriture automatiquement

                    if (vitesseAffichage){
                        for (let id = bornI;id < bornS;id ++){
                            fourmis[id].yourTurn(bareme,nourriture,coor,fourmis,ballon);
                        }
                        moveBall(); // On n'oublie pas de faire bouger le ballon après toutes les fourmis.
                    }
                    draw(fourmis); // Ici on fait appelle à la fonction contenue dans draw.js qui ne sert qu'à
afficher la partie.
                    if (vitesseAffichage) mancheRestante -= 1;
                    slow = slowNumber;
                }
                else slow -= 1;
                if (mancheRestante > 0) window.requestAnimationFrame(manche);
                else {
                    // On log le code des fourmis qui viennent de jouer.
                    console.log("Voici les fourmis qui viennent de jouer.");
```

```
                    for (let id = bornI;id < bornS;id ++){
                        console.log(waveSize);
                        console.log(fourmis[id].getCode());
                    }
                }
            };
            window.requestAnimationFrame(manche);
        }
        else {
            // Ici c'est la partie sans affichage. Elle est bien plus simple et est utile pour faire des calculs
sans fatiguer le processeur.
            for(var i = 0;i < nManche;i++){
                for (let id = bornI;id < bornS;id ++){
                    fourmis[id].yourTurn(bareme,nourriture,coor,fourmis,ballon);
                }
                moveBall();
            }

        }
    }
}

function moveBall(){
    // Fonction qui s'occupe de faire agir le ballon. Le ballon est un test infructueux. Les fourmis l'ignorent
royalement.
    ballon[4] = ballon[0];
    ballon[5] = ballon[1]; // On stocke en 4 et 5 la position dans laquelle veux aller le ballon.
    if (ballon[2] != 0){
        if (ballon[2] > 0){
            ballon[2] -= 1;
            ballon[4] += 1;
        }
        if (ballon[2] < 0){
            ballon[2] += 1;
            ballon[4] -= 1;
        }
    }
    if (ballon[3] != 0){
        if (ballon[3] > 0){
            ballon[3] -= 1;
            ballon[5] += 1;
        }
        if (ballon[3] < 0){
            ballon[3] += 1;
            ballon[5] -= 1;
        }
    }
    for (let id = bornI;id < bornS;id ++){
        fourmis[id].getListe().forEach(
            function(e){
                if (e.getX() == ballon[4] && e.getY() == ballon[5]){
                    ballon[4] = ballon[0];
                    ballon[5] = ballon[1];
                }
            }
        );
    }
    if (ballon[4] < 0 || ballon[5] < 0 || ballon[4] >= taillePlateau || ballon[5] >= taillePlateau){
        ballon[4] = (ballon[4] + taillePlateau)%taillePlateau;
        ballon[5] = (ballon[5] + taillePlateau)%taillePlateau;
    }
    for (let i = bornI;i < bornS;i ++){
        if (coor[i][0] == ballon[4] && coor[i][1] == ballon[5]){
            fourmis[i].addPoints(bareme.ballon);
            ballon[4] = places[placeId].ballon[0];
            ballon[5] = places[placeId].ballon[1];
        }
    }
    ballon[0] = ballon[4];
    ballon[1] = ballon[5];
}

function whatDoIDoNow(){  // C'est la grosse boucle qui exécute chaque action une par une.
    for(let iAct = 0;iAct<actions.length;iAct++){
        if (actions[iAct] == "init") init();
        else if (actions[iAct] == "takeBackInput") postMessage(["I'm done !"]);
        else if (actions[iAct] == "default") defaultInit();
        else if (actions[iAct] == "changeId") {iAct += 1; placeId += actions[iAct];}
        else if (actions[iAct] == "tournoi") tournoi(false,fourmis);
        //else if (actions[iAct] == "tournoiVisible") tournoi(true);
        else if (actions[iAct] == "selection") selection();
        else if (actions[iAct] == "saveData"){
            iAct += 2;
            let M = 0;
            fourmis.forEach(
                function(e,i){
                    if (e.getPoints() > fourmis[M].getPoints()) M = i;
                }
            );
            if (savedData[actions[iAct]] == undefined) savedData[actions[iAct]] = [];
```

```
            if (actions[iAct-1] == "points"){
                thingsSaved = ["pts",""];
                savedData[actions[iAct]].push(fourmis[M].getPoints());
            }
            else if (actions[iAct-1] == "interetBallon"){
                thingsSaved = ["ballon",""];
                savedData[actions[iAct]].push(fourmis[M].getInteretBall());
            }
        }
        else if (actions[iAct] == "drawGraph"){
            postMessage(["dessine ce truc",JSON.stringify(savedData),thingsSaved[0]]);
        }
        else if (actions[iAct] == "drawMatch"){
            var code = [];
            fourmis.forEach(
                function(e,i){
                    code.push(e.getCode());
                }
            );
            postMessage([code]);
        }
        else if (actions[iAct] == "display"){
            iAct += 1;
            postMessage(["dis cette chose",actions[iAct]]);
        }
        else if (actions[iAct] == "preselected") {
            iAct += 1;
            preselect(actions[iAct]);
        }
        else if (actions[iAct] == "reporting"){
            var code = [];
            fourmis.forEach(
                function(e,i){
                    code.push(e.getCode());
                }
            );
            console.log("RAPPORT EN COURS");
            postMessage(["rapport",code]);
        }
        else if (actions[iAct] == "reportingM"){
            var code = 0;
            fourmis.forEach(
                function(e,i){
                    if (e.getPoints() > fourmis[code].getPoints()) code = i;
                }
            );
            console.log("RAPPORT EN COURS");
            postMessage(["rapportM",fourmis[code].getCode()]);
        }
    }
    //postMessage([code,evolution]);    // J'ai enlevé le message de fin de liste des tâches mais c'est pas
forcément une bonne idée sur le long terme...
}

function selection(){
    let A = 0;
    let B = 1;
    let total = 0;
    fourmis.forEach(
        function(e,i){
            total += e.getPoints();
            if (e.getPoints() >= fourmis[A].getPoints()){
                B = A;
                A = i;
            }
            else if (e.getPoints() >= fourmis[A].getPoints()){
                B = i;
            }
        }
    );
    //console.log(evolution);
    // Les tests semblent indiquer que la selection des meilleurs fourmis fonctionne
    pointsFourmis = [0,0,0,0,0,0,0,0];
    //console.log(JSON.stringify(fCode[A]));
    // On vient de selectionner les deux meilleurs fourmis A et B On va les mettre aux positions 0 et 1 de notre
nouveau groupe de fourmis
    let newFourmis = [fourmis[A].copy(0),fourmis[B].copy(0)];
    let jj = 0;
    for (let i = 2; i < teamN * (7 / 8); i ++){
        if (i%2 == 0) newFourmis[i] = fourmis[A].copy(error);
        else newFourmis[i] = fourmis[B].copy(error);
        jj = i;
    }
    for (let i = jj;i < teamN; i++){
        newFourmis[i] = generateCode(teamN);
    }

    // On a notre nouvelle liste de Fourmis ! Avant on les mélangeait pour qu'elles ne soient pas avantagée par
leur position de départ mais maintenant tout est parfaitement symétrique.
    let liste = [];
    for (let i = 0;i < teamN; i ++){
```

```
        liste[i] = i;
    }

    fourmis.forEach(
        function(e,i){
            let alea = rnd(liste.length);
            e.init(membN,i);
            e.getData(newFourmis[liste[i]],[0,0],0); // On ne prends pas alea en compte.
            e.goToNest(coor);
            //liste.splice(alea,1);
        }
    );
}

const Fourmi = function(){  // C'est la classe Fourmi. Chaque objet Team contient une ou plusieurs Fourmi.
    let position = [0,0];
    let code = [[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,10,10],[0,0,0,0,0],[0,0,0,0,0],
[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,10,10],[0,0,0,0,0],[0,0,0,0,0]];
    let n = 0;
    let carry = 0;
    let points = 0;
};

Fourmi.prototype.fonctionVision = function(n,code){  // Fonction qui renvoie le coefficient d'attraction en
fonction de la distance de l'objet et des coefficients internes.
    var r = code[0] * n * n + code[1] * n + code[2];
    //var r = code[1] * n + code[2];
    r = Math.min(r,code[3]);
    r = Math.max(r,code[4]);
    return r;
};
Fourmi.prototype.calculVect = function(vect,B,code){
    //console.log(D);
    let norme = Math.abs(vect[0]) + Math.abs(vect[1]);  // Fonction qui renvoie le vecteur d'attraction d'un
objet.
    if (norme != 0){
        vect[0] /= norme;
        vect[1] /= norme;
    }
    norme = this.fonctionVision(norme,code);
    B[0] += vect[0] * norme;
    B[1] += vect[1] * norme;
    return B;
};
Fourmi.prototype.getData = function(codeN,positionN,nN,carryN,pointsN){
    // Cette fonction initialise les données internes de la fourmi.
    if (codeN == undefined) codeN = [[50,-2,0,0,0],[0,-2,50,0,0],[0,0,0,10,10],[0,0,0,10,10],[0,0,0,10,10],
[0,0,0,10,10],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,10,10],[0,0,0,0,0],[0,0,0,0,0]];
    this.code = codeN;
    if (positionN == undefined) positionN = [0,0];
    if (nN == undefined) {nN = 0; console.log("On m'a passé un n undefined ! Gros débile !");}
    if (carryN == undefined) carryN = 0;
    if (pointsN == undefined) pointsN = 0;
    this.n = nN;
    this.points = pointsN;
    this.position = positionN;
    this.carry = carryN;
};
Fourmi.prototype.goToNest = function(coor){
    let lulz = coor[this.n];
    this.position[0] = lulz[0];
    this.position[1] = lulz[1];
};
Fourmi.prototype.getX = function(){
    return this.position[0];
};
Fourmi.prototype.getY = function(){
    return this.position[1];
};
Fourmi.prototype.isHere = function(x,y){
    return x == this.position[0] && y == this.position[1];
};
Fourmi.prototype.getCode = function(){
    return this.code;
};
Fourmi.prototype.getPoints = function(){
    return this.points;
};
Fourmi.prototype.getCarry = function(){
    return this.carry;
};
Fourmi.prototype.looseCarry = function(){
    this.carry = 0;
};
Fourmi.prototype.addPoints = function(n){
    this.points += n;
};
Fourmi.prototype.tendance = function(fourmis,nourriture,coor,ball){
    // Cette fonction a besoin de la liste de toutes les fourmis, de la liste des positions de la nourriture, et
les coordonnées des nids.
    // La fourmi n veut faire un choix de direction qui se traduira par un nombre entre 0 et 3  4 si elle ne veut
```

```
aller nulle part. (ce qui est très très rare)
    // D'abord chaque objet va lui envoyer un signal sous la forme d'un vecteur dans sa direction
    // On ajoute ce vecteur au vecteur final de la fourmi qui indiquera au final la direction qu'elle veut le plus
prendre
    let cc = this.carry; // 0 si la fourmi ne transporte rien 1 sinon
    let result = [[0,0],[0,0],[0,0],[0,0]]; // liste resultat
    let vecteurFinal = [0,0];

    for (let j = bornI;j < bornS;j ++){
        let liste = fourmis[j].getListe();
        for (let i = 0;i < liste.length;i++){
            let vecteursA = [liste[i].getX() - this.position[0],liste[i].getY() - this.position[1]];
            vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[0 + cc]);
        }
    }

    for (let i = 0; i < nourriture.length; i ++){
        let vecteursA = [nourriture[i][0] - this.position[0],nourriture[i][1] - this.position[1]];
        vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[2 + cc]);
    }

    for (let i = 0; i < nourrRaf.length; i ++){
        let vecteursA = [nourrRaf[i][0] - this.position[0],nourrRaf[i][1] - this.position[1]];
        vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[8 + cc]);
    }

    // Calcul du vecteur pour le nid

    let vecteursA = [coor[this.n][0] - this.position[0],coor[this.n][1] - this.position[1]];
    vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[4 + cc]);

    // Calcul du vecteur pour le ballon

    vecteursA = [ball[0] - this.position[0],ball[1] - this.position[1]];
    vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[6 + cc]);

    // Calcul du vecteur pour les usines

    for (let i = 0; i < usine.length; i ++){
        let vecteursA = [usine[i][0] - this.position[0],usine[i][1] - this.position[1]];
        let nnn = 12 + cc;
        if (usine[i][2] == 0) nnn = 10 + cc;
        vecteurFinal = this.calculVect(vecteursA,vecteurFinal,this.code[nnn]);
    }

    return vecteurFinal;
};
Fourmi.prototype.chooseDirection = function(fourmis,nourriture,coor,ball){
    // Cette fonction détermine la direction à prendre à partir de la tendance renvoyée par la fonction interne
tendance.

    var final = this.tendance(fourmis,nourriture,coor,ball);

    // Voici la méthode : on prend le vecteur et ça nous donne une probabilité pour les deux directions x et y
    var total = Math.abs(final[0]) + Math.abs(final[1]);
    if (total == 0) return 4;
    var hasard = rnd(total);
    if (hasard < Math.abs(final[0])) {
        if (final[0] > 0) return 1;
        else return 3;
    }
    else {
        if (final[1] > 0) return 2;
        else return 0;
    }

};
Fourmi.prototype.copy = function(erreurs){ // Cette fonction retourne une copie du code de la fourmi avec un
certain nombre d'erreurs !
    var result = [];
    for (var etat = 0;etat < 14;etat++){
        result[etat] = [];
        for (var type = 0; type < 5; type ++){
            result[etat][type] = this.code[etat][type];
        }
    }

    // On effectue des erreurs mineures !
    for (var i = 0; i < erreurs;i++){
        result[rnd(8)][rnd(5)] += rnd(21) - 10;
    }
    return result;
};
Fourmi.prototype.yourTurn = function(that,bareme,nourriture,coor,fourmis,ball){
    // C'est la fonction qui sert à dire à la fourmi : vas-y joue ton tour !
    let dir = this.chooseDirection(fourmis,nourriture,coor,ball); // D'abord la fourmi choisit sa direction.
    that.move(that,dir,bareme,nourriture,coor,ball);              // Et ensuite elle effectue son mouvement.
};
Fourmi.prototype.move = function(that,dir,bareme,nourriture,coor,ball){
    // La fourmi avance dans la direction dir ou ne fait rien si dir == 4
    if (dir == 4) return true;
```

```
    let coors = [];
    coors[0] = this.position[0];
    coors[1] = this.position[1];
    const vecteurs = [[0,-1],[1,0],[0,1],[-1,0]];
    coors[0] += vecteurs[dir][0];
    coors[1] += vecteurs[dir][1];
    if (coors[1] < 0 || coors[0] < 0 || coors[0] >= taillePlateau || coors[1] >= taillePlateau){
        this.points += bareme.bonk;
        coors[0] -= vecteurs[dir][0];
        coors[1] -= vecteurs[dir][1];
    }

    let bonk = 0;
    let canSteal = this.carry == 0;
    //                    Partie vol de nourriture
    let gainFood = 0;
    for (let id = bornI;id < bornS;id ++){
        fourmis[id].getListe().forEach(
            function(e){
                if (e.getX() == coors[0] && e.getY() == coors[1]) {
                    bonk = 1;
                    if (canSteal && e.getCarry() == 1){
                        e.looseCarry();
                        gainFood = e.potential;
                        e.points += bareme.dropBonk;
                    }
                }
            }
        );
    }

    if (gainFood > 0){
        this.carry = 1;  // La fourmis vole la nourriture.
        this.potential = gainFood;
        this.points += bareme.takeBonk;
    }

    // Partie Ballon
    if (ball[0] == coors[0] && ball[1] == coors[1]){
        // La fourmi frappe dans la balle !!!
        bonk = 1; // La fourmi ne peut avancer et subit un malus, frapper dans un ballon n'est une bonne chose que
si l'on sait ce que l'on fait.
        ball[2] += vecteurs[dir][0];
        ball[3] += vecteurs[dir][1]; // Ici on imprime le vecteur mouvement de la fourmi sur la balle. La balle
avance de deux cases en théorie.
    }

    if (bonk == 1) {
        this.points += bareme.bonk; // La fourmi gagne ou perd des points car elle a foncé dans une autre fourmi
ou dans le ballon.
    }
    else {
        this.position[0] = coors[0];
        this.position[1] = coors[1];
    }

    bonk = 0;

    if (that.carry == 0){
        let kill = -1;
        nourriture.forEach(
            function(e,i){
                if (e[0] == coors[0] && e[1] == coors[1]){
                    bonk = 1;
                    e[2] -= 1;
                    if (e[2] <= 0) kill = i;
                }
            }
        );
        if (kill != -1) nourriture.splice(kill,1);


        kill = -1;
        nourrRaf.forEach(
            function(e,i){
                if (e[0] == coors[0] && e[1] == coors[1]){
                    bonk = 2;
                    e[2] -= 1;
                    if (e[2] <= 0) kill = i;
                }
            }
        );
        if (kill != -1) nourrRaf.splice(kill,1);

    }

    if (bonk == 1) {
        this.carry = 1;
        this.potential = bareme.deliver;
        this.points += bareme.catch;
    }
```

```
    else if (bonk == 2){
        this.carry = 1;
        this.potential = bareme.deliverR;
        this.points += bareme.catch;
    }

    if (coor[this.n][0] == coors[0] && coor[this.n][1] == coors[1] && this.carry == 1) {
        this.carry = 0;
        this.points += this.potential;
    }

    // Partie usine !
    for (let i = 0; i < usine.length;i++){
        if (usine[i][0] == coors[0] && usine[i][1] == coors[1]) { // La fourmi est sur une usine.
            if (usine[i][2] == 0 && this.carry == 1 && this.potential != bareme.deliverR){
                this.carry = 0;
                usine[i][2] = 5;
                this.points += bareme.giveFactory;
            }
            else{     // On considère que la fourmi transportant de la nourriture peut travailler à l'usine...
Parce que pourquoi pas.
                if (usine[i][2] == 1) addFoodR(coors[0],coors[1],1);
                usine[i][2] = Math.max(usine[i][2] - 1,0);
            }
        }
    }

};

Fourmi.prototype.getInteretBall = function(){
    let result = 0;
    for (var i = 0; i < taillePlateau; i ++){
        result += this.fonctionVision(i,this.code[6]);
        result += this.fonctionVision(i,this.code[7]);
    }
    result = result/(2*taillePlateau);
    return result;
};
Fourmi.prototype.getCodeResume = function(){
    let norme = [];
    let middle = taillePlateau/2;
    for (let i = 0; i < this.code.length; i++){
        norme[i*2] = 0;
        norme[i*2 + 1] = 0;
        for (let j = 0; j < middle; j ++){
            norme[i*2] += this.fonctionVision(j,this.code[i]) * (middle - j);
        }
        for (let j = Math.floor(middle); j < taillePlateau; j ++){
            norme[i*2+1] += this.fonctionVision(j,this.code[i]) * (j - middle + 1);
        }
    }
    let range = minMax(norme);
    range[1] = range[1] / 3;
    range[3] = range[3] / 3;
    let modif = 1;
    for (let i = 0; i < norme.length; i ++){
        if (norme[i] >= 0) modif = 1;
        else modif = -1;
        norme[i] = Math.floor(Math.abs(norme[i]) / range[1 + 2*Math.floor((i%4)/2)]) * modif;
    }
    return norme;
};

function minMax(liste){
    let result = [0,0,1,0];
    for (let i = 0; i < liste.length;i++){
        if (Math.abs(liste[i]) > result[1 + 2*Math.floor((i%4)/2)]) {
            result[2*Math.floor((i%4)/2)] = i;
            result[1 + 2*Math.floor((i%4)/2)] = Math.abs(liste[i]);
        }
    }
    return result;
}

function addFood(x,y,n){
    // Cette fonction ajoute de la nourriture sur la case de coordonnées x,y
    if (n == undefined) n = 1;
    nourriture.forEach(
        function(e,i){
            if (e[0] == x && e[1] == y){
                nourriture[i][2] += n;
                return;
            }
        }
    );
    nourriture.push([x,y,n]);
}

function addFoodR(x,y,n){
    // Cette fonction ajoute de la nourriture sur la case de coordonnées x,y
    if (n == undefined) n = 1;
```

```
    nourrRaf.forEach(
        function(e,i){
            if (e[0] == x && e[1] == y){
                nourrRaf[i][2] += n;
                return;
            }
        }
    );
    nourrRaf.push([x,y,n]);
}

var Team = function(){
    let members = [];
    let points = 0;
    let n = 0;
};

Team.prototype.init = function(taille,nI){
    this.n = nI;
    this.members = [];
    for (let i = 0;i < taille;i++){
        this.members[i] = new Fourmi();
    }
};
Team.prototype.initPts = function(){
    this.points = 0;
    for (let i = 0; i < this.members.length; i ++){
        this.members[i].addPoints(this.members[i].getPoints() * -1);
    }
};
Team.prototype.getData = function(code,position,points){
    this.points = points;
    for (let i = 0; i < this.members.length; i ++){
        this.members[i].getData(code[i],JSON.parse(JSON.stringify(position)),this.n,0,points);
    }
};
Team.prototype.goToNest = function(position){
    for (let i = 0; i < this.members.length; i ++){
        this.members[i].goToNest(JSON.parse(JSON.stringify(position)));
    }
};
Team.prototype.yourTurn = function(bareme,nourriture,coor,fourmis,ballon){
    for (let i = 0; i < this.members.length; i ++){
        this.members[i].yourTurn(this.members[i],bareme,nourriture,coor,fourmis,ballon);
    }
};
Team.prototype.getCode = function(){
    let result = [];
    for (let i = 0; i < this.members.length; i ++){
        result[i] = this.members[i].getCode();
    }
    return result;
};
Team.prototype.addPoints = function(n){
    this.points += n;
};
Team.prototype.getPoints = function(){
    var result = this.points;
    for (var i = 0; i < this.members.length; i ++){
        result += this.members[i].getPoints();
    }
    return result;
};
Team.prototype.copy = function(error){
    var result = [];
    for (var i = 0; i < this.members.length; i ++){
        result[i] = this.members[i].copy(error);
    }
    return result;
};
Team.prototype.getListe = function(){
    return this.members;
};

Team.prototype.looseCarry = function(){
    for (var i = 0; i < this.members.length; i ++){
        this.members[i].looseCarry();
    }
};

Team.prototype.getInteretBall = function(){
    let result = this.members[0].getInteretBall();
    for (var i = 1; i < this.members.length; i ++){
        let inte = this.members[i].getInteretBall();
        if (inte > result) result = inte;
    }
    return Math.round(result);
};
```

Bibliographie :

- Wikipedia : Problème du sac à dos : https://fr.wikipedia.org/wiki/Problème_du_sac_à_dos

- Raphaël Cerf : Une théorie asymptotique des algorithmes génétiques

- Souqet Amédée Radet Francois-Gérard : ALGORITHMES GENETIQUES

- WebGL genetic algorithms : evolving images : http://math.hws.edu/eck/cs424/graphicsbook2018/source/webgl/image-evolver.html

- Wikipedia : Algorithme génétique : https://fr.wikipedia.org/wiki/Algorithme_génétique

- Guillaume Cloux : Algorithme génétique : Darwin au service de l'intelligence artificielle : https://toiledefond.net/algorithme-genetique-darwin-intelligence-artificielle/

- Mehdi Moussaïd : Imiter ou innover : faut-il faire comme tout le monde ? https://www.youtube.com/watch?v=asHiYmdk9W0