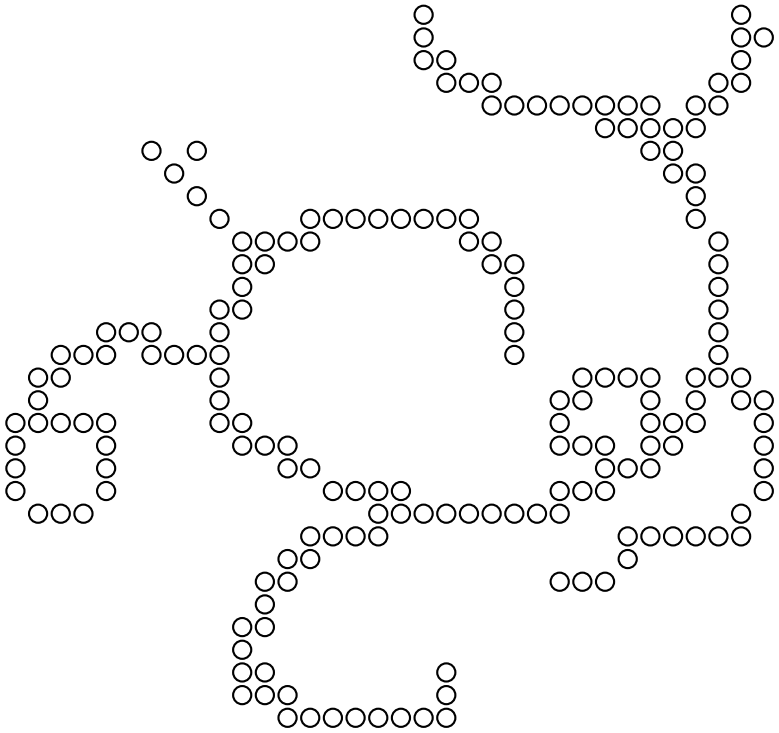
UFR Sciences
Département Informatique et Interactions
Site de Luminy
TER Mai 2014

Laboratoire d'Informatique Fondamentale de Marseille
Unité Mixte de Recherche 7279
CNRS - Aix-Marseille Université

Rapport de Travail Encadré de Recherche

Rassemblement d'agents mobiles

Eloi Perdereau



Tuteur : Shantanu Das
Responsable TER : Omar Boucelma

Remerciements

Je tiens à remercier mon tuteur, Mr Shantanu Das qui m'a été d'une aide précieuse tout au long du projet, en m'accordant son temps et en m'apportant ses connaissances et son expérience du monde de la recherche.

Table des matières

1 Introduction

Dans le cadre de ma première année de Master informatique à Aix-Marseille Univsersité (site de Luminy), je suis amené à faire un projet (TER) encadré par un chercheur d'AMU ou un stage en entreprise. Ayant plus une vocation liée à la recherche, je me suis orienté vers un projet mêlant programmation et théorie fondamentale.

Le projet rentre dans le domaine de l'algorithmique distribuée ; branche de l'informatique théorique dont le but est de développer des algorithmes pour des systèmes distribués impliquant plusieurs processus interconnectés par un réseau. Ces processus indépendants interagissent et coopèrent entre eux en vue de réaliser une tâche donnée. L'idée principal du calcul distribué est que les processus communiquent entre eux par l'envoi de messages. Néanmoins, nous nous intéressons ici à un paradigme légèrement différent : les *agents mobiles*. Ce sont des programmes qui peuvent se déplacer de nœuds en nœuds à l'intérieur du réseau de manière autonome. Bien qu'ils peuvent être implémentés via le modèle précédent (et font donc parti du calcul distribué), ils fournissent une abstraction plutôt naturelle pour le développement d'algorithmes tels que la détection d'intrus, l'exploration d'un réseau inconnu, ou encore la formation d'un motif par des robots (*robot pattern formation problem*). Ce dernier problème consiste à arranger les robots dans le plan pour qu'ils forment et conservent un motif donné.

L'objectif ici est de développer et d'implémenter une solution pour un cas particulier de ce problème : le rassemblement d'agents mobiles. Aussi appelé le Gathering Problem, beaucoup de contributions y ont déjà été apporté, impliquant souvent un plan continu ou une visibilité infinie. Nous nous restreignons ici à un espace discret et une visibilité constante des robots. De plus, le système est synchrone, c'est à dire qu'il existe une horloge globale partagée par tous les processus. Autrement dit, les algorithmes fonctionneront par *rondes* (ou *étapes*.) Pour notre cas, cela signifie que tous les robots décident et se déplacent en même temps.

Le travail à réaliser dans ce projet est donc la recherche des cas de voisinage et de déplacement des robots pour le Gathering Problem dans notre contexte. Il faut également développer une application permettant la visualisation des robots et de l'algorithme distribué. Cette partie sera faite en utilisant le langage Python et la librairie TkInter pour l'interface graphique.

Dans un premier temps je vais vous présenter plus en détail le sujet et les outils utilisés ; puis je vous exposerais le travail que j'ai réalisé développant les différentes étapes de recherche d'un algorithme correct ainsi que les techniques d'implémentation utilisés. Enfin, je conclurai par un bilan personnel et professionnel.

2 Cadre du projet

2.1 Présentation du sujet

Bien que le problème ne soit théorique, nous modélisons un système plus ou moins réaliste qui pourrait émerger d'applications pratiques. Néanmoins, nous allons nous concentrer sur le développement d'un algorithme correct pour résoudre le problème posé sans se soucier des cas pratiques.

Comme énoncé dans l'introduction, nous nous plaçons dans le cadre d'un univers discret. C'est à dire que le plan peut être représenté par une grille infinie à deux dimension où chaque cellule possède 8 cellules adjacentes différentiables par leur coordonnées relatives. Par convention, on définit l'axe x *vers la droite* et l'axe y *vers le bas*. Les entités mobiles (ou robots) sont modélisés comme des unités de calcul ayant une mémoire locale et sont capables d'effectuer des calculs locaux. Les robots sont placés dans le plan et sont représentés par des cellules "pleines" ayant un couple de coordonnées (x, y) dans $\mathbb{N}^2$. Ils sont capables de se déplacer librement dans l'espace de manière synchrone et sont dotés de capacités sensorielles leur permettant d'observer la position de leur voisin à un instant t. Plus formellement, les robots effectuent en permanence une succession de trois opérations :

(i) *Observer*. Le robot observe son voisinage en activant ses capteurs qui lui renvoient un instantané des positions des robots à l'intérieur de son champ de visibilité.

(ii) *Calculer*. Le robot effectue un calcul local donné par l'algorithme. Le résultat de ce calcul est un point de destination (à distance au plus 1 du robot concerné). Si ce point est la position du robot, celui-ci ne se déplace pas.

(iii) *Se_déplacer*. Le robot se déplace à la position renvoyée par le calcul.

La séquence *Observer-Calculer-Se_déplacer* constitue un *cycle de calcul*, aussi appelé *ronde* ou *étape*. On définit plusieurs contraintes et suppositions sur les robots dont il faut nous soumettre.

Synchronicité Le temps est divisé en étapes (discrétisation). À chaque étape de temps, les robots effectuent une ronde, i.e. les trois opérations cités ci-dessus.

Pas de communication directe entre les robots Certains modèles permettent l'envoi de messages entre les robots en plus de leur déplacement. Cela est en effet assez réaliste si les robots sont considérés comme des machines électroniques avec des capacités de communications et un protocole défini. Ils communiquent néanmoins indirectement via leur mouvements et leur positions relatives.

Visibilité limitée Les robots ne sont capables de recevoir de leur capteurs uniquement des informations sur leur entourage restreint. Ils n'ont donc pas de connaissance du nombre total de robots où de la présence de robots en dehors de leur champ de visibilité (*visibility radius.*)

Homogénéité Tous les robots sont pré-programmés avec un même algorithme et commencent à la même étape. Le système étant synchrone, ils peuvent garder localement un compteur de cycle qu'ils effectuent, et ce compteur aura la même valeur pour tous les robots à chaque instant de l'algorithme.

Anonymat Il n'existe aucun paramètre global permettant de dissocier les robots. Leur capteurs leur renvoie donc que des informations sur la position des robots alentours, et nullement des particularités propres à chaque robot voisin.

Déterminisme Les voisinages déterminent le mouvement : pour un même voisinage, les robots réagissent de la même façon. L'algorithme développé sera donc déterministe. Une solution au Gathering Problem a déjà été apporté dans un contexte non-déterministe.

Points denses Plusieurs robots peuvent se retrouver dans une même position a un instant t. Il n'est pas nécessaires pour un robot de connaitre le nombre de robots sur un point particulier, car une fois qu'ils sont à la même position, tous recevrons les mêmes informations de leur capteurs, et prendrons donc la même décision du fait de leur homogénéité. On peut donc les considérer comme un seul robot.

Mémoire constante Chaque robot ne peut retenir en mémoire qu'un nombre limité d'informations. Par exemple, son dernier mouvement ou son entourage précédent.

Déplacement local À chaque étape, les robots ne peuvent se déplacer que sur une cellule adjacente.

Désorientation En plus d'avoir un système de coordonnée différente, les robots ont également leur propre orientation. Nous ne pouvons donc pas nous fier à l'orientation des instantanés reçues sur les entourages et devons les considérer comme s'ils avaient subies aléatoirement des rotations successives des 90°.

Autonomie Le mécanisme de coordination utilisé par les robots pour se rassembler doit être totalement décentralisé, i.e. aucun contrôle central n'est utilisé.

Terminaison Du fait de l'autonomie des robots : à la fin de chaque étape, chaque robot doit savoir s'il est dans un état terminal ou non. Il n'y a donc pas de système global permettant la désactivation des robots à distance. Quand tous les robots sont dans un état terminal, ils s'arrêtent et l'algorithme se termine. Quand aucun robot ne se déplace lors d'une étape, le système n'évoluera plus ; on dit qu'on a atteint un état *quescient*.

Le graphe défini comme sous graphe de l'espace ne contenant que les nœuds occupés par des robots est connexe au départ, et doit le rester après chaque cycle. C'est à dire que chaque robot est accessible par tous les autres en ne passant que par des robots à distance 1. Cette restriction est nécessaire car, intuitivement, il est très difficile, voire impossible pour un robot qui n'a pas de voisin de se rassembler avec les autres robots de l'espace dû à sa visibilité limitée. On peut dire qu'on a un point de rassemblement par composante connexe.

Concernant ce point de rassemblement, on ne peut pas garantir son unicité à cause de la désorientation des robots. Par exemple, s'il ne reste que deux robots adjacents, ils ont le même voisinage (à une rotation près), et donc à cause de leur déterminisme, il n'y a pas moyen de les dissocier pour choisir un point unique pour le rassemblement. Nous considérons donc que s'il y a 1, 2 ou 4 robots adjacents, l'algorithme peut se terminer et les robots sont rassemblés. Plus

précisément, les cas de terminaison sont exposés en figure 1.

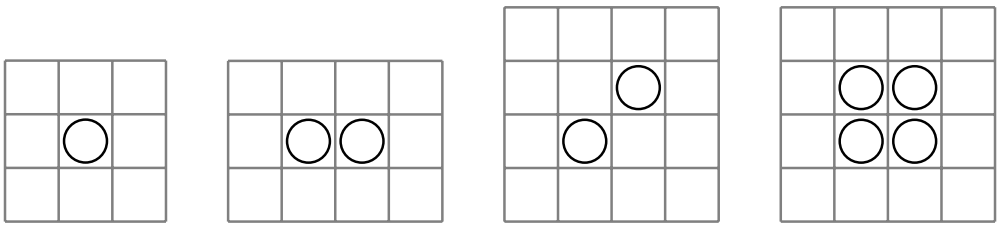


FIGURE 1 – Cas terminaux globaux

Pour résumer, on a un ensemble de robots sur le plan discret disposé de manière connexe qui doivent se rassembler. Ils se déplacent de manière synchrone, sont autonomes, homogènes, anonymes et désorientés. L'objectif du sujet est de trouver un algorithme déterministe à exécuter par tous les robots, qui parvient à les rassembler en un nombre de rondes fini. Je me suis d'abord attaché à déterminer les cas de voisinage à distance 1, tout en implémentant l'interface graphique permettant une visualisation et la mise en œuvre de l'algorithme. Puis il a fallu modifier, sophistiquer et prouver la correction de l'algorithme.

2.2 Déroulement du projet

Le TER a commencé le 10 mars, étant réalisé seul, et la programmation assez légère, je n'ai pas eu à utiliser de méthodes de développement très développées. J'ai néanmoins utilisé un système de gestion de version tout au long du projet. Cela m'a permis d'avoir les codes sources et toute l'historique de mon travail où que je sois (car hébergé sur internet). De plus, cela rendais la communication avec mon tuteur plus simple, il pouvais voir le travail que je réalisais à tout moment.

Une fois la base de code écrite, elle faisait l'objet de beaucoup de petites modifications temporaires pour tester les avancement de recherche effectués à côté. La preuve de l'algorithme a également fait appel à la programmation, mais cela était toujours temporaire et n'affectais pas l'application graphique, d'où la nécessité de versionner les sources.

Enfin, la soutenance s'est tenue le 2 juin devant un jury composé de membres du laboratoire d'informatique fondamentale (LIF).

2.3 Outils

Je vais vous présenter ici les outils utilisés durant le projet. La majorité sont des logiciels libres, car je pense que les privilégier est un devoir moral de tout utilisateur.

Environnement de développement *Vim* et *tmux* dans un système *GNU/Linux*. Vim (Vi iMproved) est un éditeur de texte modal très personnalisable en mode texte. Il est disponible par défaut sur beaucoup de distributions GNU/Linux. Tmux est un multiplexeur de terminaux en mode texte qui permet de manipuler et d'utiliser plusieurs terminaux virtuels dans un seul *terminal* en tant que processus système.

Rapport *LaTeX*. L'édition en mode texte simple (plaintext) du rapport me permet de le versionner et d'utiliser mon environnement de développement habituel. Je peux également générer et intégrer facilement au rapport des schémas à partir du code Python en utilisant le package *tikz*.

Code *Python* et la bibliothèque *TkInter*. Python est un langage interprété et interactif de haut niveau doté d'une grande variété de modules intégrés par défaut. Parmi ceux-ci, TkInter : rapide à prendre en main, il permet la création d'interfaces graphiques simples.

Versionning *Git* : logiciel de gestion de versions décentralisé permettant une gestion très fine du code source ou de n'importe quel contenu texte. Le dépôt du projet est hébergé sur *GitHub* (`http://github.com/perelo/RobotGathering`), ainsi que la configuration de mon environnement de développement.

Tous ces outils me permettent de travailler confortablement sur des machines minimalement équipées, où même à travers une session ssh distante.

3 Travail réalisé

Cette section retrace le travail que j'ai réalisé durant ce projet, les difficultés que j'ai rencontré ainsi que les solutions apportées.

3.1 Développement et preuve de l'algorithme

Le processus de recherche a nécessité plusieurs étapes, et est encore en cours. Dans un premier temps, il a fallu déterminer les cas de voisinage (instantanés) dans lesquels les robots effectuent un mouvement. On considère d'abord que les robots n'ont qu'une visibilité de 1, c'est à dire qu'ils ne voient que les robots qui sont dans des cellules adjacentes. La figure 10 en annexe montre tous les cas de voisinage à distance 1 qu'un robot peut rencontrer. Il n'est pas nécessaire de traiter les cas analogues correspondant à des rotations de 90°, 180° et 270° de chacun de ces cas. En effet, lorsqu'un robot reçoit un instantané qui ne fait pas parti des cas exposés, il effectue la rotation nécessaire, et applique l'algorithme selon le cas ayant subi la rotation.

L'idée de base pour le rassemblement est d'*arrondir les angles*. La figure 2 montre les cas sujet à mouvements. Les autres cas de voisinage n'impliquent aucun mouvement de la part des robots.

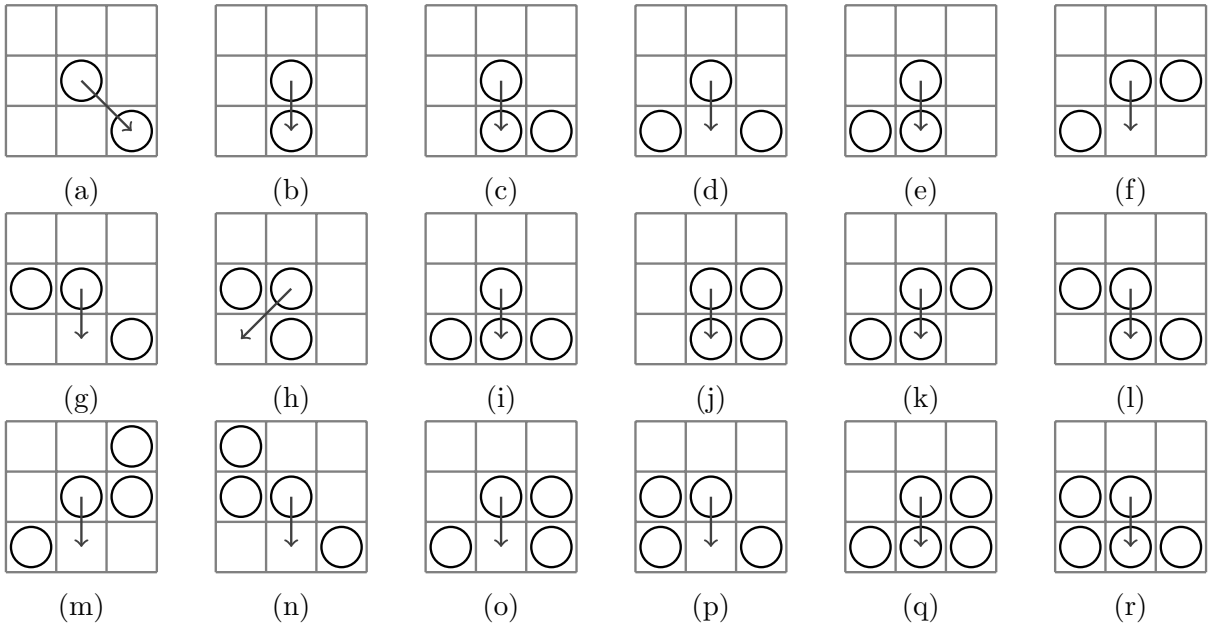


FIGURE 2 – Cas de voisinnage sujet à mouvement

Ces cas semblent assez naturels, cependant ce n'est pas aussi simple. Par exemple, examinons de plus près les cas (f) et (g). Si les robots se déplacent dans ces cas, il existe une infinité d'arrangements des robots qui mènent à un graphe non connexe. Or, si ils ne se déplacent pas, il existe une infinité d'arrangements quescient alors que les robots ne sont pas du tout rassemblés. Il en est de même pour les cas (m) et (n). Les figures 3 et 4 montrent des exemples de cas symétriques où les robots sont respectivement déconnectés et quescient si les cas (f) et (g) ne sont respectivement utilisés et non utilisés. Même s'il n'était pas nécessaire de le prouver, cela montre que le GATHERING PROBLEM ne pourra se résoudre avec des robots sans mémoire (*oblivious.*)

Pour résoudre ce problème, il faut réussir à détecter la déconnexion, puis utiliser la mémoire des robots pour revenir à la position précédente. Cela nécessite deux rondes :

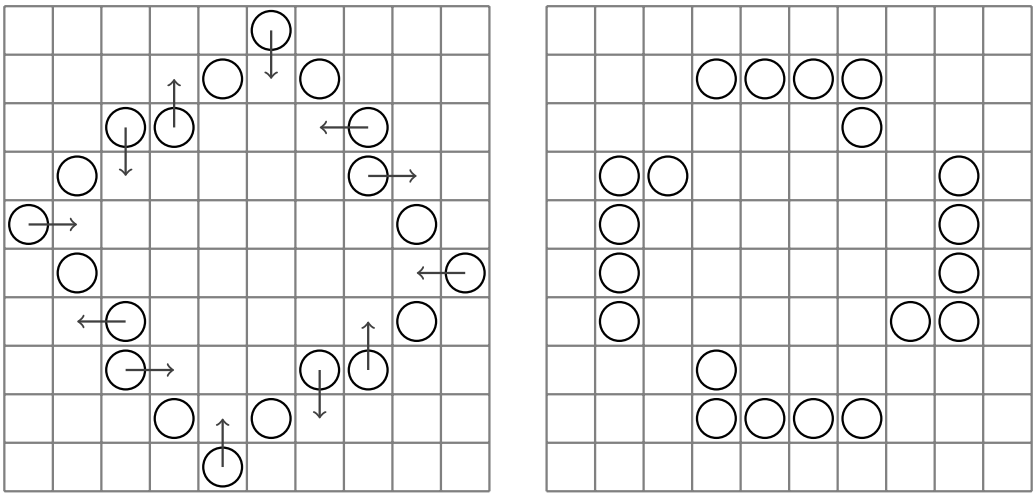


FIGURE 3 – Déconnexion

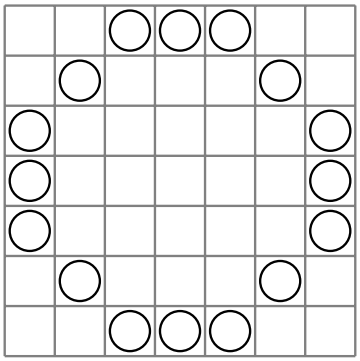


FIGURE 4 – Quescient

1 Regarder son entourage et se déplacer en utilisant les cas (f), (g), (m) ou (n) s'il le faut, et enregistrer le mouvement.

2 Détecter la déconnexion si on était dans un des cas (f), (g), (m) ou (n) et se déplacer à la position précédente si nécessaire.

Il faut que tous les robots exécutent ces deux rondes, et qu'ils ne se déplacent pas à la deuxième ronde s'ils ne sont pas déconnectés. Cela peut se réaliser facilement avec un compteur de rondes : Comme les robots commencent tous en même temps, on peut définir que les rondes paires correspondent à la première étape, et les rondes impaires à la deuxième. Ainsi, cela double le nombre de rondes sans affecter la complexité de l'algorithme.

Si un robot en position (i,j) est dans le cas (f) ou (m) à la première étape, la déconnexion est détectée dans la deuxième étape par l'absence totale de robots aux positions $(i+1,j)$, $(i+1,j-1)$ et $(i,j-1)$. Pour les cas (g) et (n) ce sont les positions $(i-1,j)$, $(i-1,j-1)$ et $(i,j-1)$.

Ceci étant réglé, un autre problème se pose : il existe des paires d'arrangements circulaires de robots qui alternent entre eux. Autrement dit, les robots ne sont pas quescient mais ne se rassemblent jamais. La figure 5 donne un tel exemple d'oscillation.

Nous avons modifié l'algorithme de beaucoup de manières pour essayer de résoudre ce problème. Nous essayions d'identifier les cas d'oscillation (locales et globales), et sommes arrivés à une solution qui semble marcher dans tous les cas. C'est de nouveau les instantanés (f), (g), (m) et (n) qui

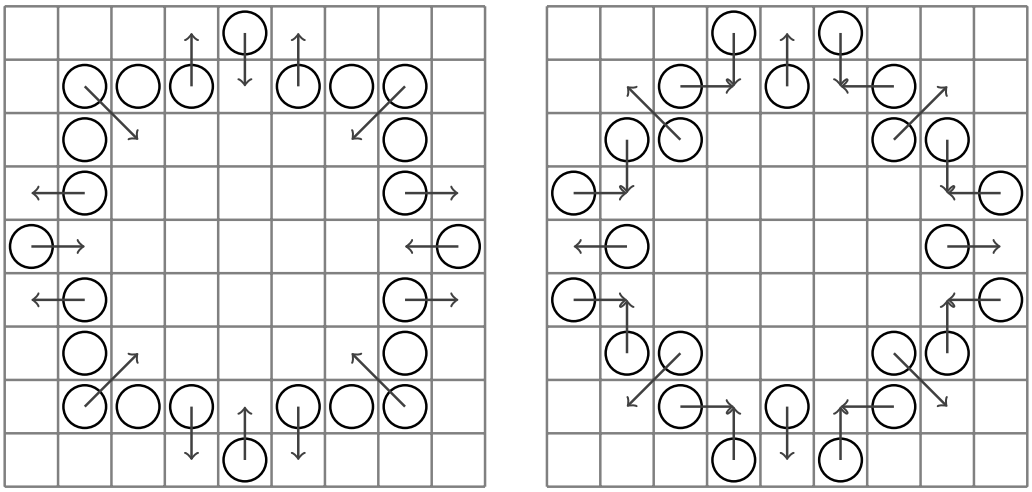


FIGURE 5 – Oscillation

posent problème : dans les cas (f) et (m), on n'effectue le mouvement que si les cellules $(i-2,j)$ et $(i-2,j+1)$ sont toutes les deux soit vides soit pleines. Pour les cas (g) et (n), ce sont les cellules $(i+2,j)$ et $(i+2,j+1)$. Notez que ces cellules ne sont pas à portée directe des robots (elles sont à distance 2.) Concernant la mise en œuvre de cette modification, nous avons d'abord essayer d'étendre la solution du problème précédent en ajoutant deux nouvelles rondes : l'une pour se déplacer à portée des cellules concernés ; l'autre pour revenir avec l'information. Mais cela est tout simplement faux car lors de cette première ronde, *tous* les robots qui se trouvent dans les cas (f), (g), (m) ou (n) vont se déplacer, et probablement remplir des cellules à vérifier par d'autres robots. Or la présence ou non des robots dans ces cellules concerne l'état précédent tout déplacement de robots. La solution proposée ne peut donc se réaliser qu'avec des robots ayant un rayon de visibilité supérieur ou égal à 2. Nous continuons tout de même a utiliser les instantanés des positions des voisins à distance 1 (figure 2), mais lorsqu'un robot se trouve dans le cas (f), (g), (m) ou (n), il regarde en plus les deux positions concernés sans se soucier des autres robots à distance 2. Même si cela est frustrant de devoir augmenter les capacités des robots pour si peu, leur visibilité est toujours limitée, et nous restons cohérent avec les contraintes que nous nous sommes imposés. Les cas (f), (g), (m) et (n) modifiés sont donc présentés en figure 6.

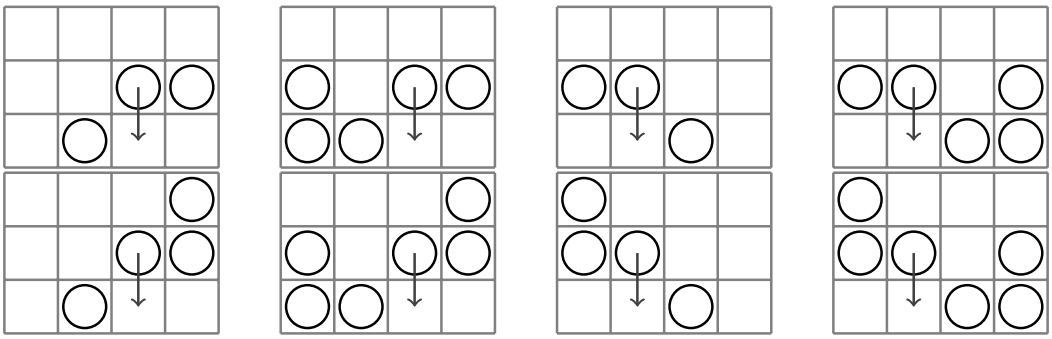


FIGURE 6 – cas (f), (g), (m) et (n) modifiés

Au final, nous avons un algorithme qui fonctionne dans tous les cas testés. Nous définissons deux fonctions pour simplifier l'écriture formelle de l'algorithme :

- *prendre_instantane*() qui prend un instantané des voisins accessibles et effectue la rotation nécessaire pour qu'il soit dans un des cas de voisinage de la figure 10 ;
- *rotation*180(N_k) qui effectue une rotation de 180° de l'instantané passé en paramètre.

La figure 1 donne le pseudo-code de l'algorithme final.

Algorithme 1 : Algorithme de rassemblement d'agents mobiles

```
fini ← Faux;
k ← 0;
tant que non fini faire
    N_k ← prendre_instantane();
    si k%2 = 0 alors
        si N_k ne contient aucun voisin ou
        N_k est le cas (a), (b) ou (j) et N_{k-2} = rotation180(N_k) alors
            fini ← Vrai;
        sinon si N_k est le cas (f) ou (m) et (i − 2,j) et (i − 2,j + 1) sont pleins ou vides ou
        N_k est le cas (g) ou (n) et (i + 2,j) et (i + 2,j + 1) sont pleins ou vides alors
            se dépacer selon N_k;
        fin
    sinon
        si N_{k−1} est le cas (f) ou (g) et (i + 1,j), (i + 1,j − 1) et (i,j − 1) sont tous vides ou
        N_{k−1} est le cas (g) ou (n) et (i − 1,j), (i − 1,j − 1) et (i,j − 1) sont tous vides alors
            se déplacer en (i, j − 1);
        fin
    fin
    k ← k + 1;
fin
```

Au cours du développement de l'algorithme, j'ai effectué une étude sur le temps de convergence des robots lorsqu'ils sont agencés en rectangles ou en blocs (rectangle rempli.) J'ai extrait des motifs qui se répétaient durant le processus, me permettant de calculer le temps qu'il fallait pour les robots pour se rassembler de manière très précise (à la ronde près.) Mais l'algorithme à été modifié depuis, et les robots ne respectent plus les motifs identifiés précédemment, rendant l'étude obsolète. Je ne l'exposerai donc pas dans ce rapport.

Une fois qu'un algorithme pour le Gathering Problem a été identifié, nous nous sommes attaché à prouver sa correction. C'est à dire, apporter une preuve mathématique qui montre que le rassemblement s'effectue bien en un nombre fini de rondes si on utilise cet algorithme. En réalité, ce processus n'est pas si linéaire. Dès qu'un premier algorithme semblait fonctionner, nous avons commencé à réfléchir à une preuve. Cela nous amenai à voir que l'algorithme était faux, nous obligeant à le modifier, et ainsi de suite. L'algorithme de la figure 1 est donc la dernière version et a été testé sur de nombreux cas aléatoires et connu comme posant problème (arrangement en cercles principalement.) La preuve de cet algorithme est à ce jour bien avancée mais pas totalement terminée. Si nous succédons à prouver la correction de l'algorithme, nous pourrons publier un article (en cours de rédaction) exposant ces résultats. Je vais vous présenter ici l'avancement de la preuve, des éléments techniques, ainsi que ce qu'il manque pour la terminer.

L'idée principale de la preuve est de considérer le plus petit rectangle englobant tout les robots à un instant t, noté $BB(t)$ (*Bounding Box*), et de montrer que ce rectangle diminue de taille après un nombre fini de rondes. Cela se réduit à considérer la ligne la plus haute de ce rectangle (*topmost row*) et de montrer qu'elle *descend* en un nombre fini de rondes. Pour cela, nous montrons dans un premier temps que s'il n'y a qu'un seul robot sur cette ligne, celle-ci descend au bout d'un nombre fini d'étapes. Puis, dans un deuxième temps, on considère le robot le plus à gauche se situant sur cette ligne (*leftmost robot*) à un instant t, et on montre qu'au bout d'un nombre fini de rondes, le robot le plus à gauche s'est déplacé vers la droite et ne se déplacera plus sur la gauche. Le robot considéré n'est pas forcément le même, c'est le robot le plus à gauche sur la ligne la plus haute à un instant t. Analogiquement, on réitère la preuve avec le robot le plus à droite qui se déplace sur la gauche, et l'ensemble prouve bien que la ligne la plus haute descend au bout d'un nombre fini de rondes.

Soit $r(t)$ le robot (seul) sur la ligne la plus haute de $BB(t)$ à l'étape t. S'il y a plus d'un robot sur cette ligne, $r(t)$ n'existe pas. Les coordonnées de $r(t)$ sont notés $X(t)$ et $Y(t)$. Nous assumerons en général que $Y(t) = 0$ sauf mention contraire. L'arrangement global des robots à l'étape t est notée $C(t)$. Si $C(t)$ est un cas terminal de l'algorithme, alors c'est appelé un arrangement ou une configuration GATHERED (rassemblée.)

Proposition 1. *Si $r(t)$ exixte et est en $(i, 0)$, alors il y avait un robot en $(i-1, 0)$, $(i, 0)$ ou en $(i+1, 0)$ à l'étape $t-1$.*

Proposition 2. *À l'étape t, si un robot w n'est pas dans le voisinage visible de $r(t)$, alors w ne pourra pas être dans la ligne la plus haute à l'étape $t+1$.*

Ces deux propositions implique qu'il suffit d'étudier le voisinage de $r(t)$ pour des nouveaux robots sur la ligne la plus haute pour déterminer si celle-ci est descendu.

Lemma 3. *Si $r(t)$ existe et $C(t)$ n'est pas* GATHERED *alors il existe une constante c telle qu'après c rondes, soit $BB(t+c) \subset BB(t)$ soit on est arrivé dans une configuration* GATHERED.

On prouve d'abord la première partie : lorsque il n'y a qu'un seul robot sur la ligne la plus haute.

Démonstration. On définit le graphe $G_{single}(V_{single}, E_{single})$ commt suit :

- V_{single} : instantanés de rayon 1 de $r(t)$
- $(u, v) \in E_{single}$ si u est le voisinage de $r(t)$ et v est le voisinage de $r(t+1)$ tel que $Y(t) = Y(t+1)$ et $C(t+1)$ n'est pas GATHERED

Autrement dit, on créé un graphe qui représente les mouvements du robot sur la ligne la plus haute. Ce graphe a été généré par programmation en énumérant tous les cas possibles de voisinage du robot après une étape. Pour cela, on énumère tous les arrangement possibles de l'espace sur un rectangle suffisamment grand tel que le robot au milieu est soit sur la ligne la plus haute. Puis on lance l'algorithme sur une étape, et on récupère l'entourage du robot. L'image du graphe est générée en utilisant le programme *dot* et est présentée en figure 7 en annexe. Les instantanés sur les nœuds sont représenté par une matrice où 0 et 1 représentent respectivement l'absence et la présence d'un robot.

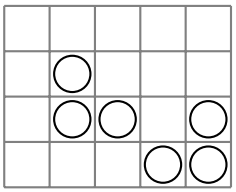
Si un nœud n'a pas d'arête sortante, cela signifie que lorsque $r(t)$ est dans le cas représenté par ce nœud, à $t+1$, soit la ligne est descendue, soit on a atteint une configuration GATHERED. On

remarque que le graphe comporte des cycles ; si un chemin cyclique est suivi par l'algorithme, alors la ligne la plus haute peut ne jamais descendre. Néanmoins, les arêtes du graphe ne représentent qu'une seule étape de l'algorithme. Nous allons donc montrer qu'en réalité l'algorithme ne peut suivre aucun cycles. Il nous suffit pour cela d'étudier seulement trois chemins, et le lemme sera prouvé.

Dans ce rapport, je me contenterai de montrer l'impossibilité d'un seul chemin. Les autres suivent le même raisonnement, je suis à votre disposition si vous êtes intéressé par la preuve complète. Toutes les affirmations cités ici ont été vérifiées par programmation par énumération.

Notons de A à G les 7 nœuds de G_{single} du haut vers le bas puis de la gauche vers la droite.

Considérons le chemin $(B \to E \to D)$. Le seul voisinage possible pour que l'algorithme suive l'arête $B \to E$ est le suivant (c'est un déplacement vers la droite) :



Selon la rêgle (n), le robot en $(X(t)+1, Y(t)+1)$ va descendre et occuper la cellule $(X(t+1), Y(t+1)+2)$. Or, pour que l'algorithme suive l'arête $E \to D$ à $t+1$, la cellule $(X(t+1), Y(t+1)+2)$ doit être vide. Par conséquent, le chemin $(B \to E \to D)$ n'existe dans aucune exécution de l'algorithme. Analogiquement, le chemin $D \to E \to B$ n'existe pas non plus ; ainsi que les chemins $(B \to E \to B)$ et $(D \to E \to D)$ du fait de la symétrie des nœuds B et D.

Nous montrons de manière similaire l'impossibilité des chemins $(B \to C \to D)$ et $(B \to D \to B)$, et par analogie et symétrie, tous les cycles de G_{single}.

□

Donc nous avons bien prouvé une première partie du lemme qui implique que s'il y a qu'un seul robot sur la ligne la plus haute, celle-ci descend après un nombre fini de rondes.

Nous abordons la deuxième partie de la preuve de manière similaire. Mais elle en demeure plus compliquée. Notons $g(t)$ le robot le plus à gauche à l'étape t, et redéfinissons $X(t)$ et $Y(t)$ comme coordonnées de $g(t)$.

Proposition 4. *Si $C(t)$ n'est pas* GATHERED *alors il existe une constante c telle qu'après c rondes, soit $C(t)$ est* GATHERED*, soit $g(t+c)$ est plus à droite que $g(t)$, i.e. $X(g(t+c)) > X(g(t))$.*

Démonstration. Nous définissons de manière similaire $G_{leftmost\_mid}$ le graphe des mouvement de $g(t)$ en ajoutant une contrainte : les arêtes représentent des déplacement *sur place*. C'est à dire que le robot au milieu du cas au nœud origine a les mêmes coordonnées que celui au milieu du cas au nœud destination. Le graphe est présenté en figure 8 en annexe. Pour prouver le lemme, il faut démontrer l'impossibilité des chemins cycliques dans ce graphe, puis étudier les cas de déplacement vers la gauche. À ce jour, seul la première partie a été démontrée. Je vais vous présenter ici comment nous avons traité les cycles n'impliquant qu'un seul nœud, avec celui dessiné en haut à gauche pour exemple. Les autres sont traités comme pour G_{single}.

On va regarder la ligne d'en dessous. Pour que ce cycle s'effectue, il ne faut pas que le robot $(X(t), Y(t))$ ne se déplace. Et comme $Y(t)$ représente la coordonnée de la ligne la plus haute,

il n'y a aucun robot au dessus, donc (dû au cas (f) modifié) y a obligatoirement un robot en $(X(t)-2, Y(t)+1)$. Par conséquent, ce dernier n'a pas de voisin à distance 1 au dessus de lui. On va donc reprendre la preuve avec pour $g(t)$ le robot le plus à gauche de la ligne d'en dessous. Ce processus a forcément une fin, car le nouveau robot considéré est plus à gauche que le précédent, donc à un moment donné, la boucle ne sera pas effectuée car intuitivement, on arrivera sur le bord gauche de la figure. Donc si on démontre la proposition avec $G_{leftmost\_mid}$ privé de la boucle considérée ici, elle sera vrai pour $G_{leftmost\_mid}$ complet, et donc le lemme sera également vrai.

□

Le graphe représentant les mouvement à gauche de $g(t)$ est aussi présenté en annexe sur la figure 9. Celui des mouvement de droite est très complexe et ne présente pas grand intérêt pour ce rapport, il sera donc omis.

Un dernier point concerne la complexité de l'algorithme. Si on arrive à compter le nombre de rondes nécessaire pour la ligne la plus haute à descendre, on aura la complexité totale du rassemblement assez facilement. Notre intuition nous dirige vers une complexité linéaire en la taille des robots (il y en a au plus $l * h$ si l et h sont les dimensions de $BB(0)$.)

Cela conclue donc la partie recherche théorique du TER. Bien que l'unité d'enseignement ne sois terminée, je continue à prouver la correction de l'algorithme en partenariat avec mon tuteur. Je vais maintenant vous parler de l'aspect programmation du projet.

3.2 Programmation

J'ai donc utilisé le langage Python et la bibliothèque TkInter, en me basant sur la charte de codage conseillée par Python : PEP8 ; le tout en anglais, permettant une plus grande maintenabilité. L'objectif était de créer une interface graphique permettant de visualiser les algorithmes distribués développés sur papier.

L'interface graphique nécessaire est très sommaire : un canevas contenant une grille fixe, et deux boutons :

- *next* pour avancer d'une étape dans l'algorithme
- *prev* pour reculer

Les clics de la souris sur le canevas permettent d'ajouter des robots à la volée sur l'espace où d'en enlever. Un autre bouton *clear* a été ajouté pour supprimer tous les robots. Avec ceci, il est très facile d'effectuer des tests sur des exemples d'arrangements de robots qui pourrait poser problème. La figure 11 en annexe montre une capture d'écran de l'application avec plusieurs arrangements de robots de tests en cours de rassemblement. Les robots colorés en rouge sont ceux qui se sont détectés comme étant dans un état quescient.

L'ensemble des robots est stocké en mémoire dans un conteneur Python appelé `set`. Il sert à stocker des ensembles, c'est à dire une liste d'objet uniques non ordonnée. Cela est suffisant car on a dit que les robots sur une même position peuvent être vus comme un seul et même robot. Pour éviter de calculer plusieurs fois une étape (par exemple lors de clics successifs sur *next* et *prev*), on stocke une liste d'ensemble de robots indexés par le numéro de la ronde. C'est la classe `Space` qui contient cette liste, et elle fourni les méthodes d'avancement des robots ainsi que des méthodes pour le remplissage de l'espace.

Des configurations de robots peuvent être écrites dans un fichier texte sous forme binaire, qui sera lu et automatiquement stocké en mémoire par le programme. Il suffit alors de choisir quels arrangements utiliser dans l'espace avant de lancer le programme. Il a donc fallu écrire une fonction qui traduit les coordonnées binaires des robots (décalages) en coordonnées cartésiennes (x et y). Voici l'exemple de la figure 5 :

```
spiky-square
0 0 0 0 1 0 0 0 0
0 1 1 1 0 1 1 1 0
0 1 0 0 0 0 0 1 0
0 1 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 1 0
0 1 0 0 0 0 0 1 0
0 1 1 1 0 1 1 1 0
0 0 0 0 1 0 0 0 0
```

En effet, tout les calculs sur les arrangements de robots sont en binaire. Ce n'est que lorsqu'on entre en contact avec l'interface graphique que l'espace est converti. Il a notamment fallu écrire une fonction de génération de configurations aléatoires. L'idée est de commencer par un espace empli de robots (une suite de 1), puis de les enlever (mettre un bit à 0) au fur et à mesure jusqu'à ce qu'on en ai le nombre désiré. Et à chaque fois qu'on met un bit à 0, on test si l'espace est encore connexe, si oui on continue, sinon, on remet bit à 1. Pour tester la connexité, on utilise un parcours en largeur des robots avec une pile en commençant par un le bit à 1 de poids le plus faible.

Concernant l'implémentation de l'algorithme, tout d'abord, les cas de mouvement sont stockés dans un fichier à part comme dans la figure 2. Les rotations sont effectués automatiquement et le tout est stocké dans un dictionnaire Python permettant d'avoir directement le mouvement à effectuer à partir du voisinage. Le format est le suivant : Les trois premières lignes correspondent à la version binaire de l'instantané, et la quatrième contient les coordonnées relatives au mouvement à effectuer. Les instantanés sont séparés par des lignes vides. Voici l'exemple du cas (d) :

```
0 0 0
0 1 0
1 0 1
1, 0
```

Cela permet une grande flexibilité par rapport aux instantanés utilisés pour l'algorithme.

Une première différence avec l'algorithme théorique est dû à la parallélisation des calculs effectués par les robots. Notre programme est séquentiel, il faut donc parcourir tous les robots deux fois au moins : une pour prévoir la prochaine position des robots, une deuxième pour les déplacer. Lorsqu'un clic est effectué sur le bouton *next*, on récupère l'entourage de chaque robot sous forme binaire ainsi que leur mouvement associé. Puis on change les coordonnées des robots en gardant à part ceux qui étaient dans les cas (f), (g), (m) et (n). Enfin, pour ces derniers, on test s'ils sont déconnectés avec une opération binaire entre un masque et l'entourage récupéré, et on re-change leur coordonnées si nécessaire.

Nous avons beaucoup utilisé la programmation pour la preuve de l'algorithme, notamment pour générer les graphes et vérifier les contraintes d'entourages pour traverser certaines arêtes (voir preuve.) Il a fallu écrire une fonction assez complexe manipulant des bits : à partir d'un champs de bit interprété comme une matrice de 0 et de 1, récupérer une sous matrice dont les dimensions sont données. C'est peut-être la fonction qui m'a donné le plus de mal mais qui est très courte et fonctionne au final très bien.

Le fait que nous ne manipulions que des champs de bits rend la génération de configurations très facile : il suffit de compter en décimal, puis de convertir en binaire. Néanmoins, pour générer les graphes, il faut générer tous les arrangements sur un rectangle $7 * 7$, et à chaque fois, tester la connexité, avancer d'une étape, puis récupérer le nouveau voisinage ; et ce pour chaque instantané sujet à mouvement. Pour ce travail, il a fallu optimiser un maximum toutes ces opérations, en changeant les structures de données, en minimisant les opérations de vérification (de terminaison par exemple), etc... La génération de $G_{leftmost\_mid}$ a durée environ 45 minutes. C'est pour cela que nous n'avons pas généré des graphes pour deux étapes de l'algorithme, ça aurait multiplier le temps de calcul de façon exponentielle, car il aurait fallu générer les arrangements sur des plus gros rectangles.

Pour terminer, j'ai ajouter temporairement un système de *dump* qui affiche sur la sortie standard les positions des robots présents sur l'espace. Le tout formaté pour *tikz*. Je pouvais ainsi très facilement dessiner sur le rapport des configurations de robots venant directement du programme.

Conclusion

todo

Références

[1] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on*, 15(5) :818–828, 1999.

[2] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In *Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, pages 125–140, 2008.

[3] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing for mobile robots : Gathering. *SIAM Journal on Computing*, 41(4) :829–879, 2012.

[4] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of robots with limited visibility. *Theoretical Computer Science*, 337(1-3) :147–168, 2005.

[5] N. Gordon, Y. Elor, and A. M. Bruckstein. Gathering multiple robotic agents with crude distance sensing capabilities. In *6th International Conference on Ant Colony Optimizatin and Swarm Intelligence*, LNCS 5217, pages 72–83, 2008.

[6] Noam Gordon, Israel A. Wagner, and Alfred M. Bruckstein. Gathering multiple robotic a(ge)nts with limited sensing capabilities. In *Proc. 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS)*, pages 142–153, 2004.

[7] Tien-Ruey Hsiang, Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, and Joseph S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 77–94, 2002.

A Graphes de mouvement

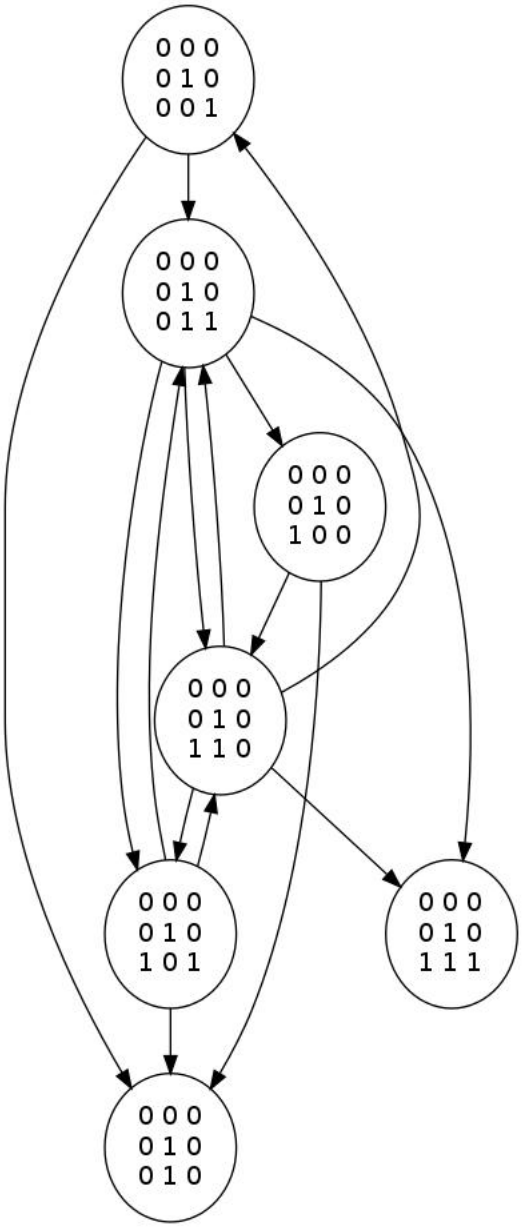


FIGURE 7 – G_{single}

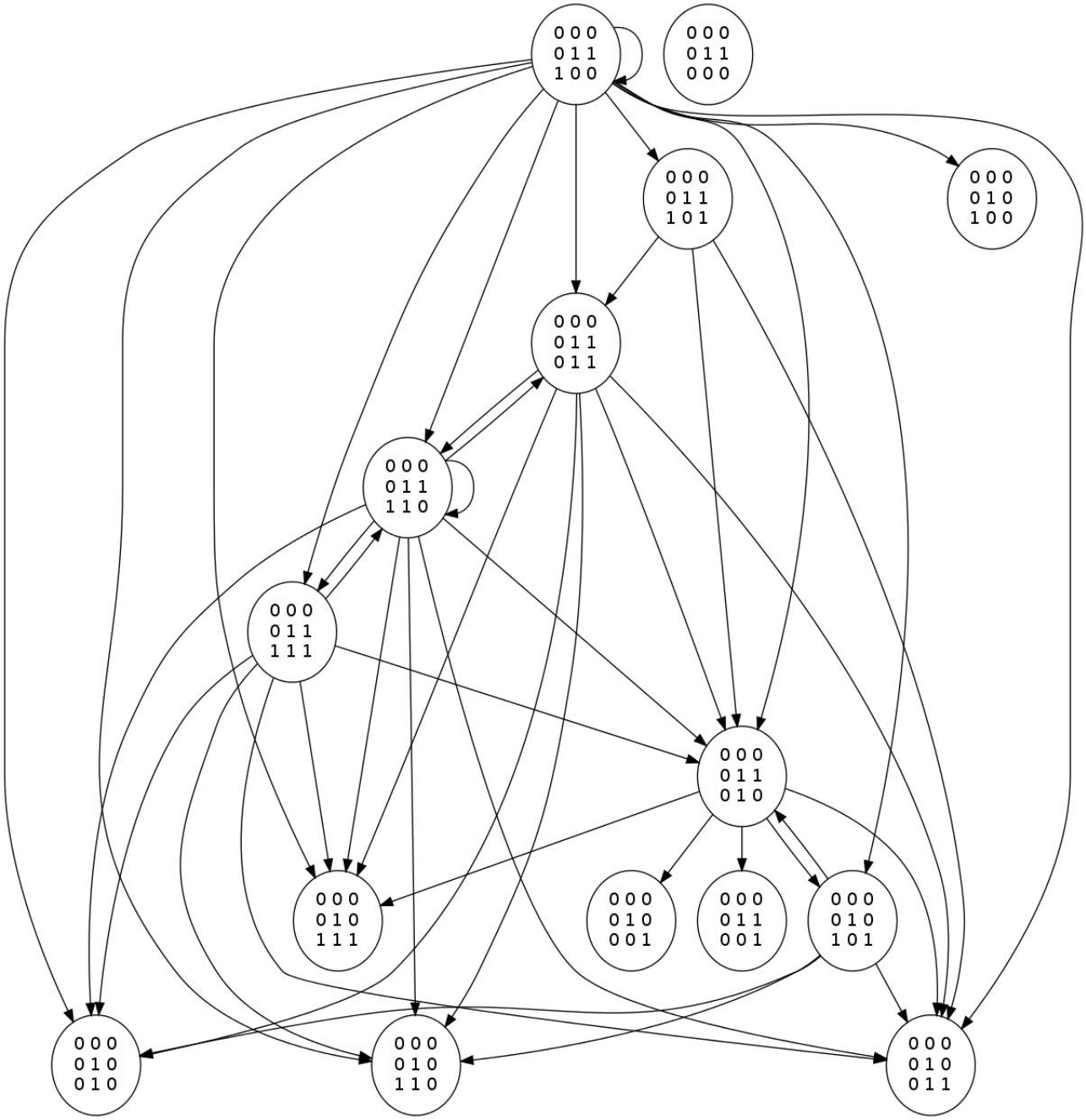


FIGURE 8 – $G_{leftmost\_mid}$

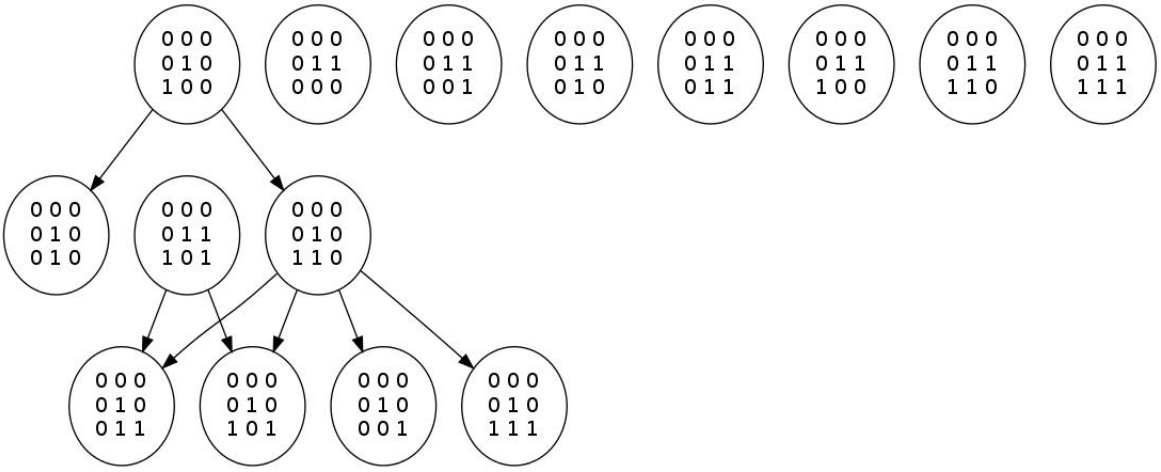


FIGURE 9 – $G_{leftmost\_left}$

B Cas de voisinage

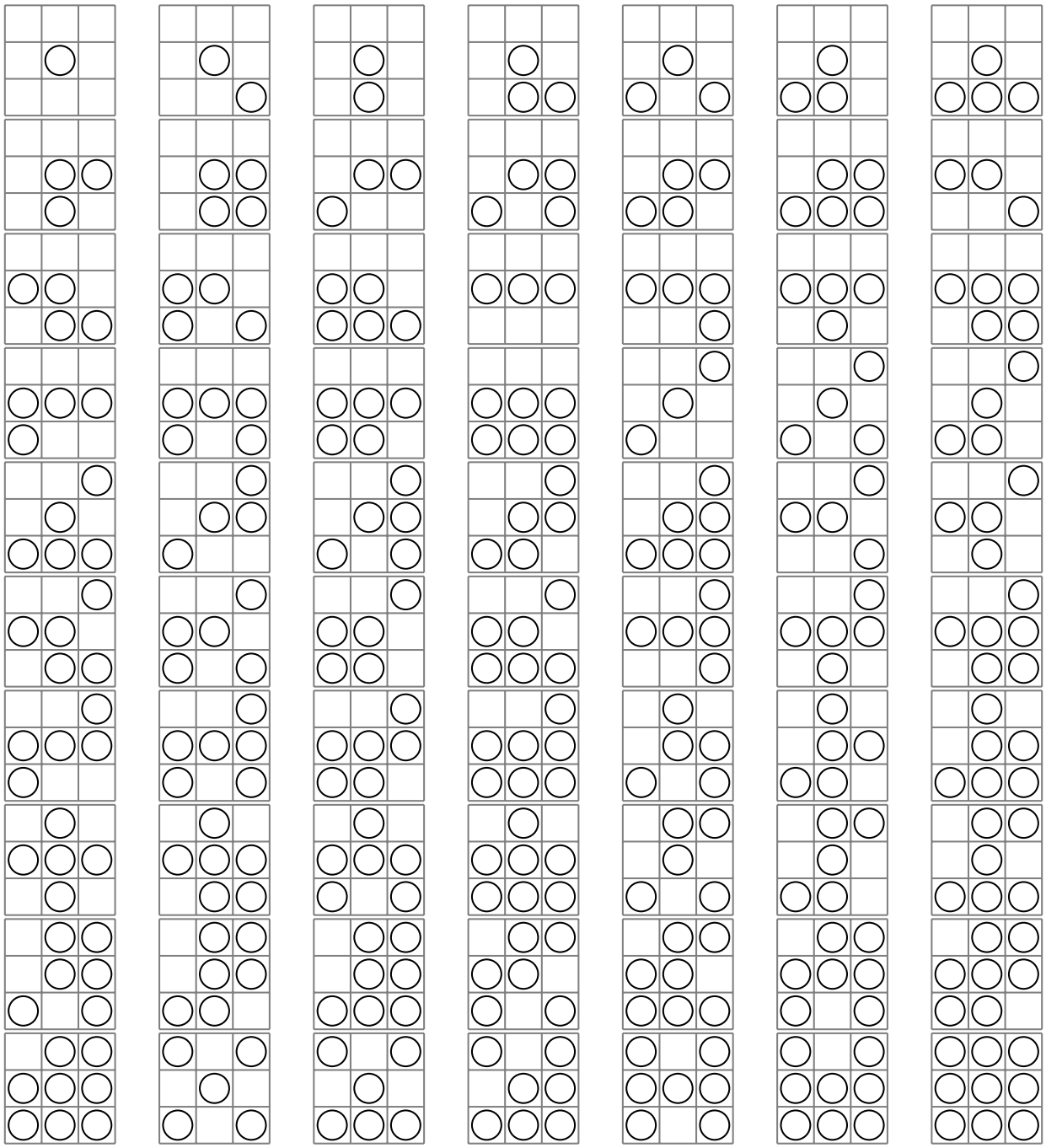


FIGURE 10 – Tous les cas de voisinnage

C Capture d'écran

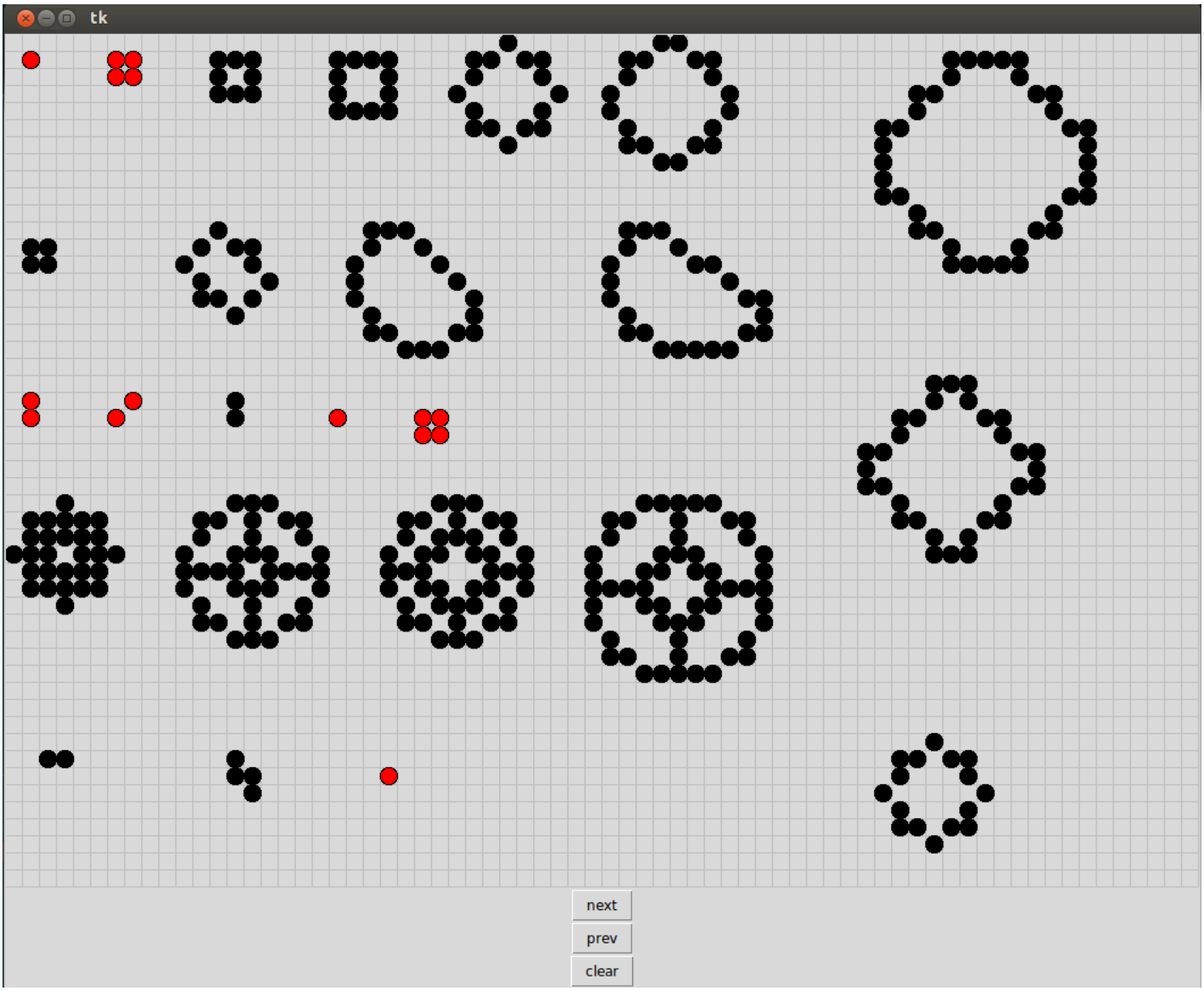


FIGURE 11 – Capture d'écran de l'application