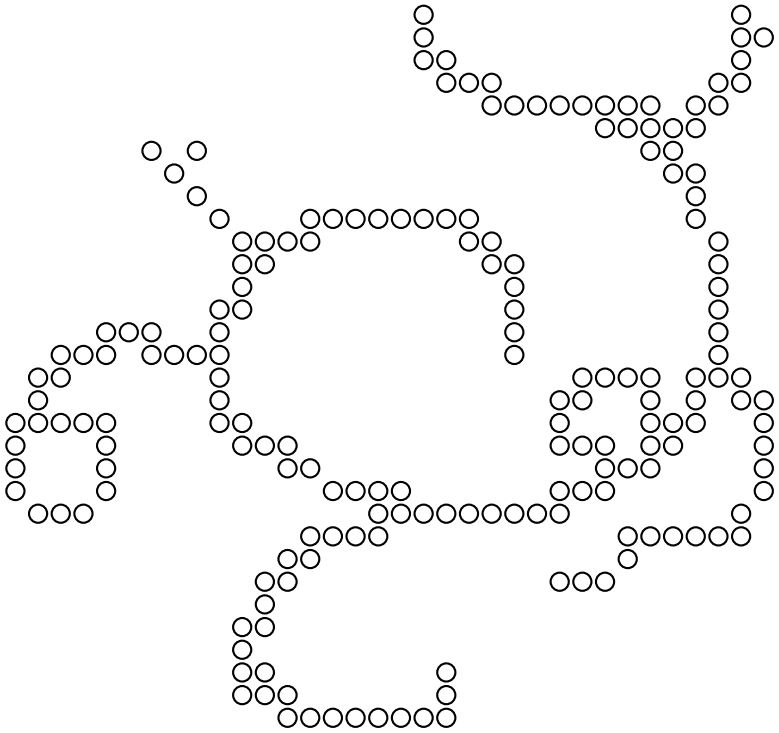
UFR Sciences
Département Informatique et Interactions
Site de Luminy
TER Mai 2014

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.

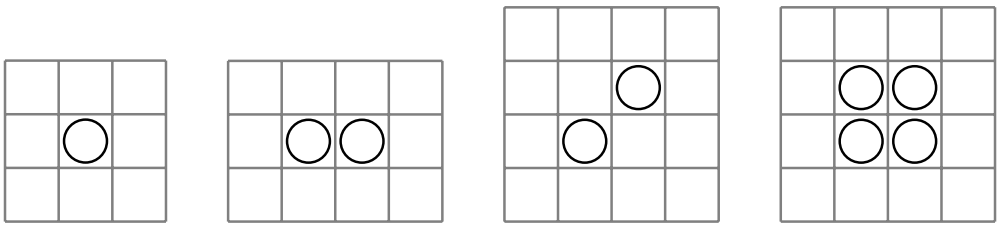


FIGURE 1 – Cas terminaux globaux

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 Recherche fondamentale

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 6 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.

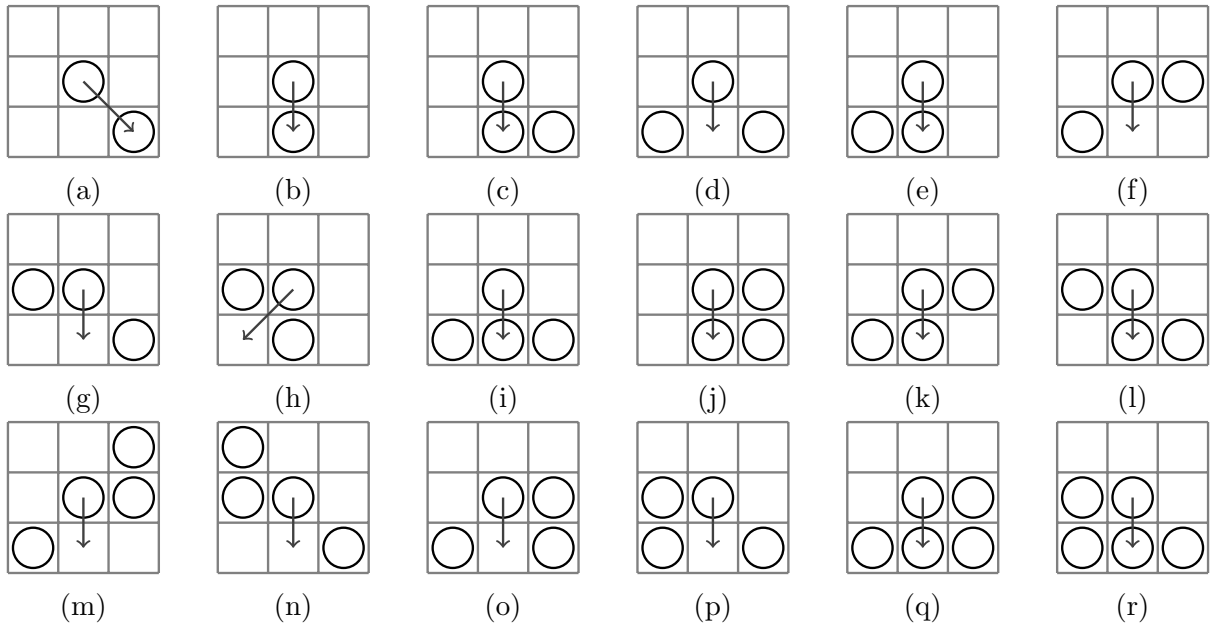


FIGURE 2 – Cas de voisinnage sujet à mouvement

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 :

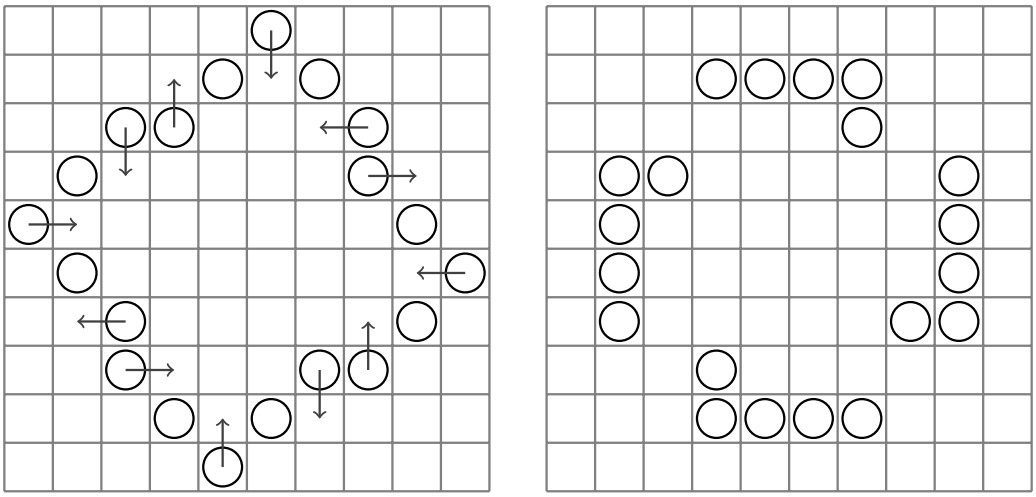


FIGURE 3 – Déconnexion

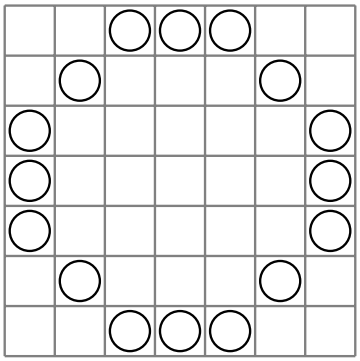


FIGURE 4 – Quescient

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 que les robots 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

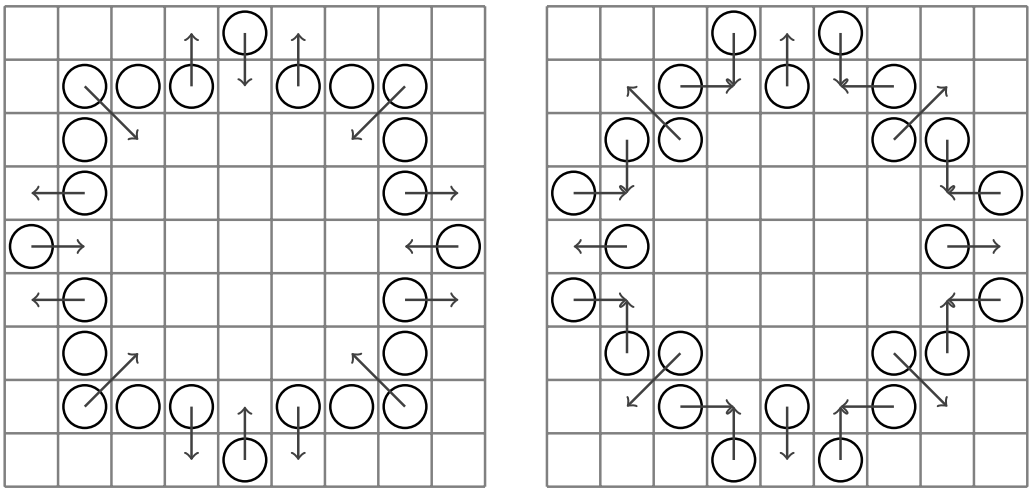


FIGURE 5 – Oscillation

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 par d'autres robots. Or la présence ou non des robots dans ces cellules concerne l'état précédent tout déplacement de robot. 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.

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 6 ;
- $rotation180(N_k)$ qui effectue une rotation de 180° de l'instantané passé en paramètre.

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

TODO : proof + étude de rassemblement des carrés ?

3.2 Programmation

Dans un algorithme séquentiel, pour l'implémentation, il faut 2 parcours de tous les robots : prévoir la prochaine position des robots, puis les déplacer.

Algorithme 1 : Algorithme de rassemblement d'agents mobiles

$fin \leftarrow Faux$;
$k \leftarrow 0$;
tant que *non fini* **faire**
  $N_k \leftarrow prendre\_instantane()$;
  si $k\%2 = 0$ **alors**
    si N_k *ne contient aucun voisin ou*
    N_k *est le cas* (a), (b) *ou* (j) *et* $N_{k-2} = rotation(N_k)$ **alors**
      $fini \leftarrow Vrai$;
    sinon si N_k *est le cas* (f) *ou* (m) *et* $(i-2,j)$ *et* $(i-2,j+1)$ *sont pleins soit vides ou*
    N_k *est le cas* (g) *ou* (n) *et* $(i+2,j)$ *et* $(i+2,j+1)$ *sont pleins soit 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
  $k \leftarrow k+1$;
fin

1 screenshot de l'appli

3.3 Programmation

IG pep8

Conclusion

todo

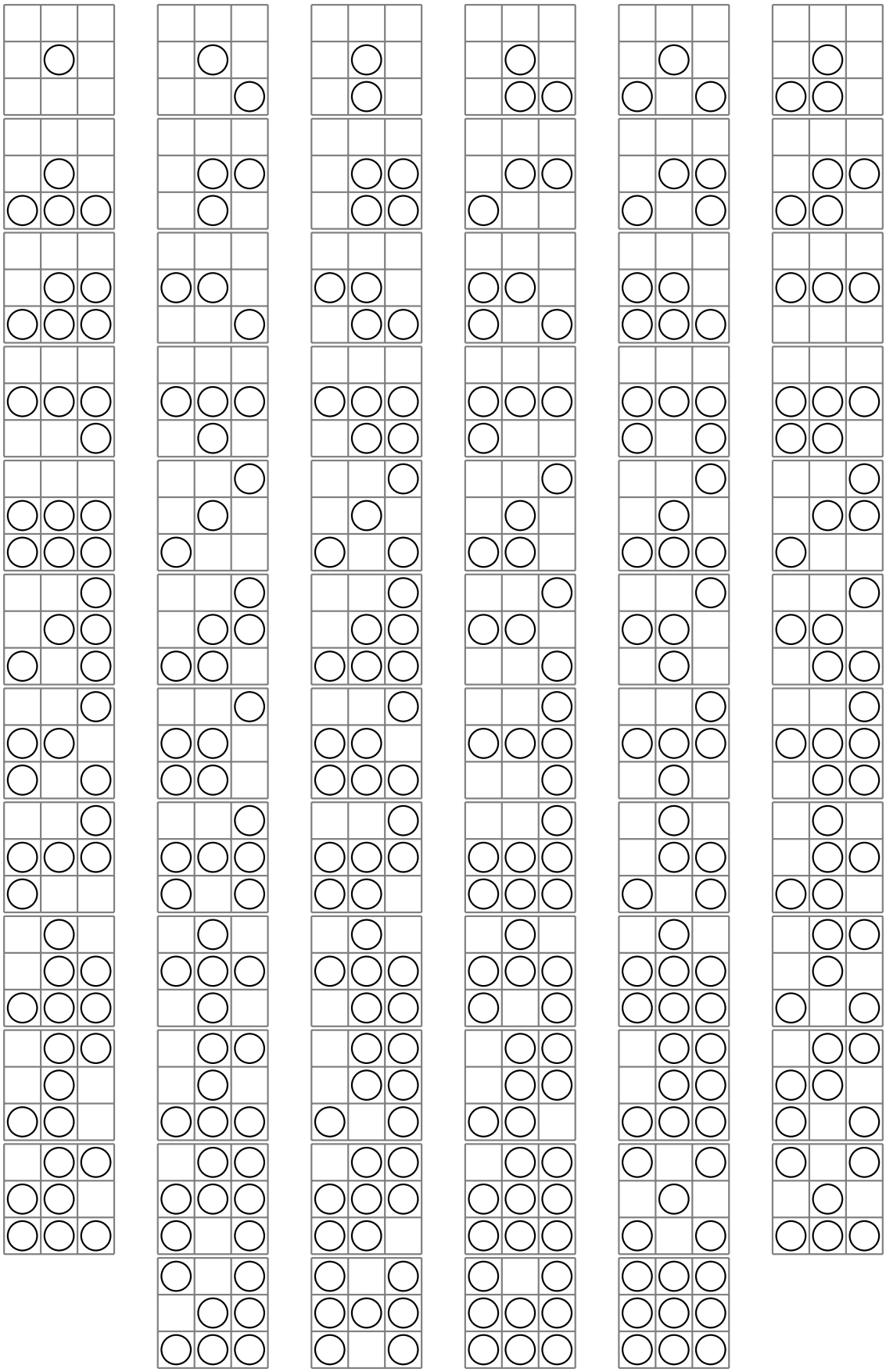


FIGURE 6 – Tous les cas de voisinnage