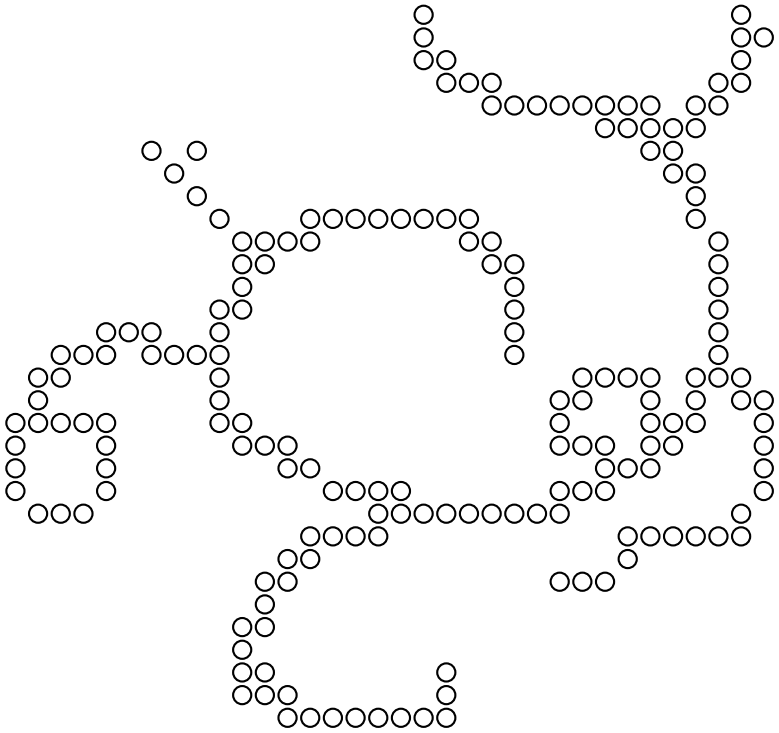
UFR Sciences
Département Informatique et Interactions
Site de Luminy
TER Mai 2014

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

Résumé

Le problème considéré est le Gathering Problem : soit n robots placés dans le plan, tous avec des positions différentes, les faire se rassembler en un point en un nombre fini de cycles. Le point de rassemblement n'est pas fixé à l'avance. Cette tâche constitue l'une des primitives de base pour le contrôle et la coordination de robots mobiles autonomes.

Table des matières

Introduction **3**

1 Cadre du projet **4**
1.1 Présentation du sujet 4
1.2 Déroulement du projet 6

2 Outils utilisés **7**

3 Travail réalisé **8**

Conclusion **9**

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 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. Concernant la programmation, j'ai utilisé le langage Python avec 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.

1 Cadre du projet

1.1 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 un graphe simple non-orienté où chaque sommet possède 8 voisins différentiables par leur coordonnées relatives au sommet considéré. 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 points de $\mathbb{N}^2$. 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. On définit plusieurs contraintes et suppositions sur les robots dont il faut nous soumettre.

Synchronicité Le temps passé pour *Observer*, *Calculer* et *Se_déplacer* est le même pour tous les robots. De plus, ils commencent à *Observer* en même temps.

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 sont des machines électroniques avec des capacités de communications et un protocole défini. Néanmoins, nous n'autorisons ici qu'une communication indirecte avec 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.

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 1 À chaque étape, les robots ne peuvent se déplacer que sur un nœud voisin direct.

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 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.

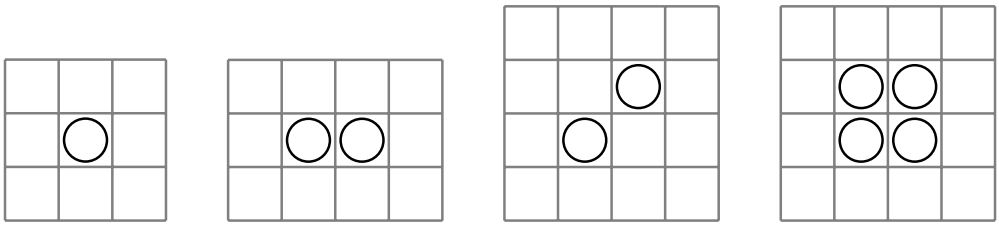


FIGURE 1 – Cas terminaux globaux

1.2 Déroulement du projet

TODO

Le projet est fait seul, et j'utilise un système de gestion de version (Git) hébergé par Github. Cela me permet d'avoir les sources et toute l'historique de mon travail où que je sois, et cela rend la communication avec mon tuteur plus simple, il peut voir le travail que je réalise à tout moment. LaTeX-¿ plaintext -¿ suivi des modifications des documents. python tkinter

2 Outils utilisés

- Vim + Tmux
- Git
- Python 2.7 + TKinter
- LaTeX

3 Travail réalisé

item choix d'implémentation
étapes pour arriver à l'algo final
étapes de la preuve

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

étude du rassemblement de carrés ?

On définit les 3 fonctions de rotation

1 screenshot de l'appli

Conclusion

todo

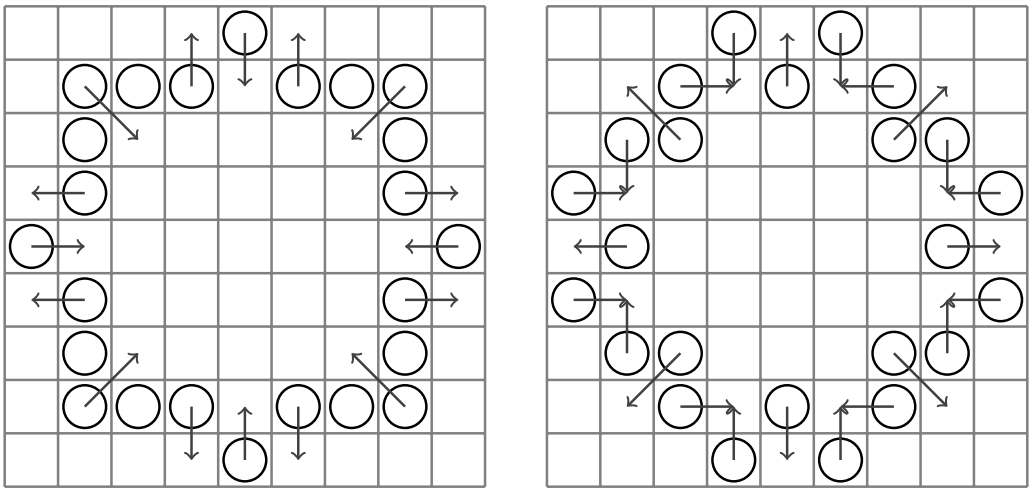


FIGURE 2 – Oscillation

Algorithme 1 :

$finish \leftarrow False$;
$k \leftarrow 0$;
while *not* $finish$ **do**
  $N_k \leftarrow get\_neighbors()$;
  if $k\%2 = 0$ **then**
    if N_k *doesn't contain any neighbor or*
    (N_k *is case* (a) *or* (b) *and* $N_{k-2} = rotate180(N_k)$*) or*
    (N_k *is case* 1.3.2 *and* $N_{k-2} = rotate90(N_k)$*)* **then**
      $finish \leftarrow True$;
    else if $(i, j-2)$ *and* $(i+1, j-2)$ *are both empty or both full* **then**
      move according to N_k;
    end
  else
    if *(*N_{k-1} *is case* (f), (g), (m) *or* (n)*) and (*$(i, j+1)$, $(i-1, j)$, $(i-1, j+1)$ *are all empty)* **then**
      move to $(i-1, j)$;
    end
  end
  $k \leftarrow k + 1$;
end

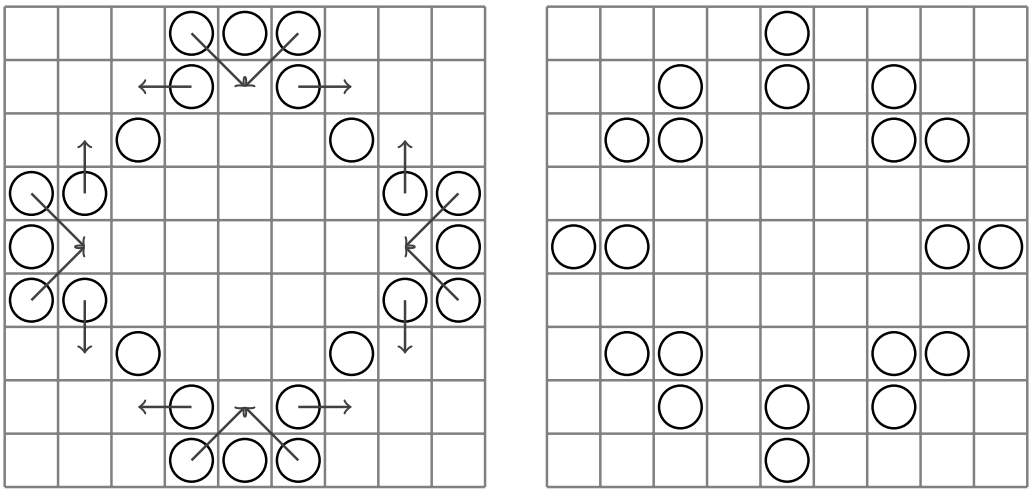


FIGURE 3 – Déconnexion

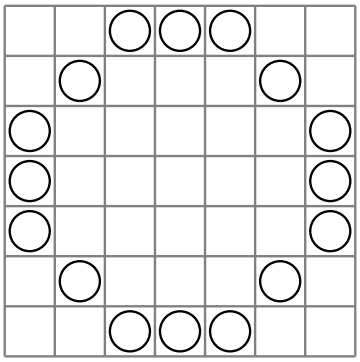


FIGURE 4 – Quescient

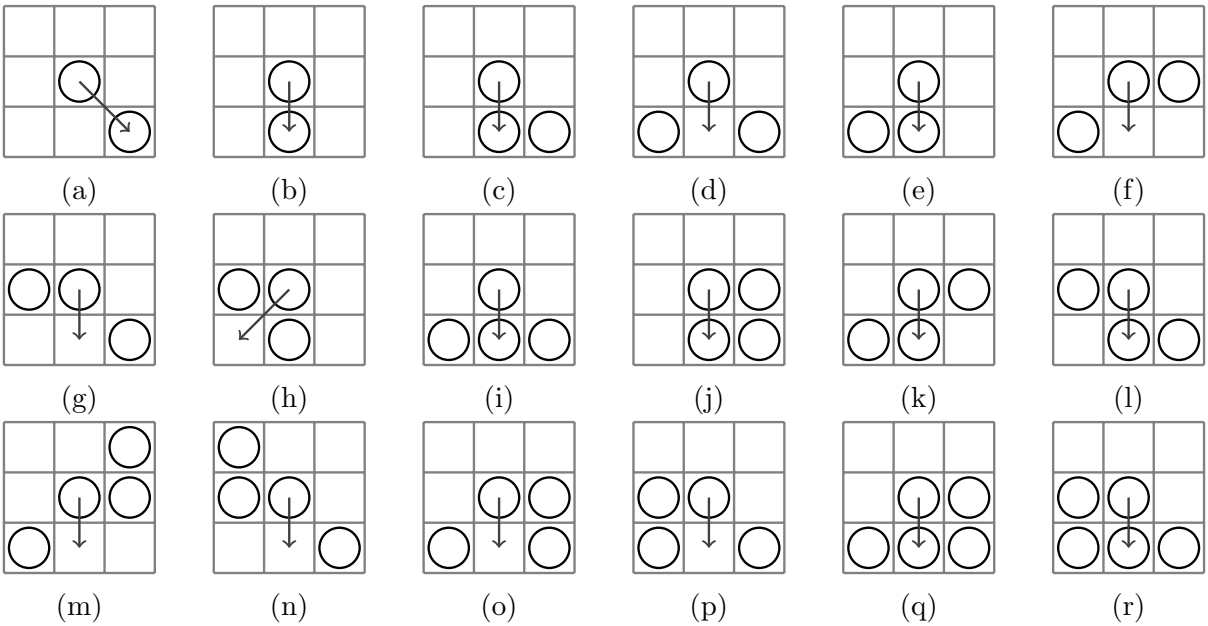


FIGURE 5 – Cas de voisinnage sujet à mouvement

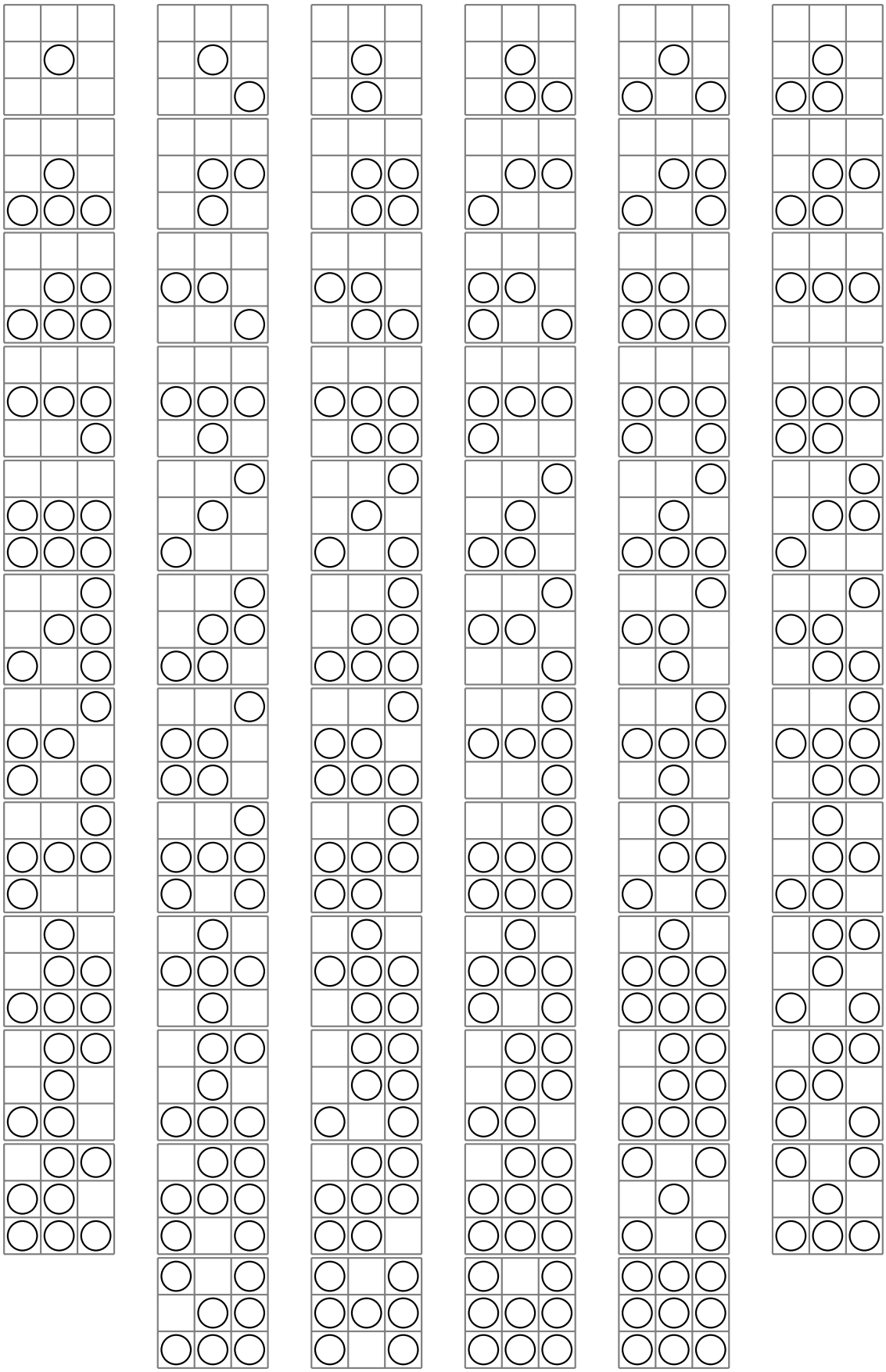


FIGURE 6 – Tous les cas de voisinnage