11 - Théorie des jeux

Nous nous intéressons à la modélisation de certains jeux à 2 joueurs : Tic-Tac-Toe, jeu de Nim, échecs, Othello, Puissance 4, etc...

Ces jeux devront posséder les propriétés suivantes :

- être à information complète (pas d'information cachée pour les joueurs, ce qui exclut la plupart des jeux de cartes) ;
- être déterministe (aucune intervention du hasard, pas de jeu de dés par exemple) ;
- les deux joueurs jouent à tout de rôle ;
- le jeu se termine par le gain d'un des 2 joueurs ou éventuellement par un match nul.

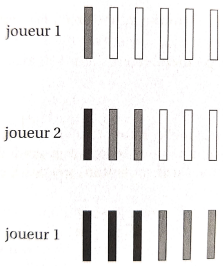
Pour les jeux simples dont l'ensemble des configurations possibles est de taille relativement petite, nous pourrons faire une étude complète (en particulier étudier le **caractère gagnant ou perdant** de chaque configuration).

Pour les jeux plus complexes (comme les échecs), ceci n'est pas possible en raison du nombre bien trop important de possibilités, mais nous verrons comment construire des programmes d'un niveau de jeu intéressant grâce à l'utilisation conjointe d'une **heuristique** et d'un algorithme appelé **minmax**.

1 Un exemple de jeu simple : Le jeu de Nim-Fibonnacci

Le jeu de Nim-Fibonacci se joue, à deux joueurs, avec un ensemble de n allumettes disposées sur une table. Les joueurs jouent tour à tour et enlèvent des allumettes de la table. Le joueur qui enlève la dernière allumette disponible a gagné.
Les règles sont les suivantes :
- au premier tour, le premier joueur doit enlever au moins une allumette et laisser au moins une allumette ;
- durant les tours suivants, le joueur doit enlever au moins une allumette et au plus le double d'allumettes que celles enlevées au tour précédent.

Dans la partie ci-contre, 6 allumettes sont disposées sur la table. Le joueur 1 prend une allumette, le joueur 2 en prend deux et enfin le joueur 1 en prend trois et gagne la partie :



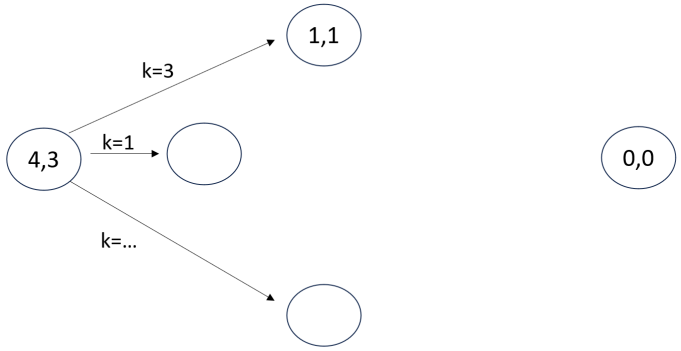
Ce jeu peut être modélisé par un graphe où les sommets sont étiquetés par des couples (i, j) tels que i est le nombre d'allumettes restantes sur la table et j est le nombre maximal d'allumettes qui peuvent être enlevées à ce tour.

Étant donné un joueur qui se retrouve en configuration (i, j), il a le droit d'enlever un nombre d'allumettes k compris entre 1 et j. Ainsi, le joueur suivant se retrouvera avec $i-k$ allumettes sur la table et il pourra théoriquement enlever au plus $2k$ allumettes. Cependant, comme il n'y a que $i-k$ allumettes sur la table, le joueur suivant pourra donc enlever au plus $min(2k, i-k)$ allumettes.

Ainsi, partant d'une position (i, j), on ne peut accéder qu'aux positions $(i-k, \min\{2k, i-k\})$ pour $1 \leq k \leq i$.

On construit ainsi des arêtes orientées de (i, j) vers les autres positions accessibles, i.e. les couples $(i-k, \min\{2k, i-k\})$ pour tout $k \in \{1, \ldots, i\}$
Au départ, le jeu commence avec le couple $(n, n-1)$. Le joueur qui se retrouvera avec le couple$(0, 0)$ aura perdu.

QUESTION : Compléter le graphe ci-dessous, correspondant à une partie où 4 allumettes sont disposées sur la table au départ :



1.1 Situations perdantes, situations gagnantes

Définition 1 - *Fin du jeu*
Un sommet d'où ne part aucune arête est appelé **sommet terminal**. La partie s'arrête.
Ce sommet peut correspondre :

- à une situation gagnante : le joueur qui se retrouve dans cette position a gagné le jeu (par exemple 4 pions de la même couleur alignés au puissance 4) ;
- à une situation perdante : le joueur qui se retrouve dans cette position a perdu le jeu (par exemple 0 allumettes sur la table au jeu de Nim) ;
- à une situation de match nul (par exemple au Tic-Tac-Toe, les 9 cases sont complétées mais personne n'a aligné 3 symboles identiques).

Définition 2 - *Situation perdante, situation gagnante*
Considérons une situation qui ne soit pas une situation terminale.

- On dira que cette position est ***gagnante*** si la personne qui joue à partir de cette position peut parvenir à gagner en fin de partie quels que soient les coups de son adversaire (on parle de **stratégie gagnante**). Le premier coup de la séquence permettant d'y parvenir est appelé *coup gagnant*.
- La position est dite *gagnante* si quel que soit le coup joué, la position suivante sera gagnante pour l'adversaire.

Proposition 1
Dans un jeu où le match nul n'est pas possible, **toute position est soit perdante, soit gagnante**.

Faisons la preuve dans le cas du jeu de Nim, par récurrence forte sur le nombre d'allumettes que comprend la position :

- *Initialisation : le position contenant 0 allumettes est perdante.*
- *Supposons que pour tout k entre 0 et n, les situations à k allumettes sont soit perdantes soit gagnantes. Considérons une situation à $n+1$ allumettes. Tout coup joué à partir de là amène à une situation comprenant un nombre d'allumettes inférieur ou égal à n. D'arpès l'hypothèse de récurrence, chacune de ces situations est soit perdante soit gagnante. Il y a alors deux possibilités :*
  - *Soit tous les coups joués amènent l'adversaire dans une situation gagnante, donc cette situation est perdante*
  - *Soit il existe un coup qui amène l'adversaire dans une situation perdante, donc cette situation est gagnante.*

  L'hérédité est donc montrée !

QUESTION : Sur le graphe ci-dessus, étiqueter chaque position (i.e. chaque sommet du graphe) comme "gagnante" ou "perdante". (on pourra remonter depuis la position $(0,0)$ qui est est perdante).
La position de départ sur ce jeu avec 4 allumettes est-elle donc perdante ou gagnante ?

Programmation

Pour déterminer si une position est gagnante on va mettre en place un algorithme récursif : une position (i, j) est gagnante pour le joueur 1 si et seulement si, parmi les positions (k, ℓ) accessibles au coup suivant, il en existe une où le joueur 2 ne peut pas gagner.

1. Écrire une fonction récursive `gagnant_rec(i, j)` qui renvoie `True` si la position (i, j) est gagnante (i.e si le joueur peut gagner avec i allumettes sur la table, en pouvant piocher au plus j allumettes)et `False` sinon.
   Le principe est le suivant : la position $(0,0)$ est perdante (test d'arrêt, et pour une position (i, j), on examine toutes les positions suivantes possibles et on leur applique la fonction récursive : soit on rencontre une position perdante, soit on n'en rencontre aucune...
2. En déduire une fonction `gagnant(n)` qui permet de savoir au joueur qui commence s'il peut gagner avec n allumettes sur la table. Tester votre fonction pour quelques valeurs de n.
3. Si ce n'est pas déjà fait, améliorer la fonction `gagnant_rec(i, j)` en utilisant le principe de mémoïsation : vous stockez dans un dictionnaire le statut de chaque position : les clés sont des couples (i,j), et les valeurs sont `True` (position gagnante) ou `False` (position perdante).
4. Tester `gagnant(n)` pour des valeurs entières successives de n et émettez une conjecture sur les positions de départ qui sont gagnantes ou perdantes.

1.2 Stratégies gagnantes

Ce que nous avons vu ci-dessus nous permet de savoir si une situation est perdante ou gagnante, mais nous n'avons pas les coups à jouer pour gagner à coup sûr lorsque l'on est dans une situation gagnante.

Pour répertorier les coups gagnants, nous allons utiliser la notion de **graphe biparti**, et la notion d'**attracteur**.

1.2.1 Graphe biparti

L'inconvénient du graphe effectué sur la page précédente est qu'il ne permet pas de distinguer les coups réalisés par le premier joueur J_1 et ceux réalisés par le second J_2.

Pour pallier ce problème, on ne va pas seulement représenter les positions possibles mais aussi différencier les sommets selon si c'est au premier ou au deuxième joueur de jouer. Pour ce faire, on va en quelque sorte «dédoubler» ce graphe (sauf la position de départ qui ne peut être jouée que par le premier joueur).

Voici donc le graphe biparti correspondant au jeu avec 4 allumettes au départ :

Définition 3
- Un graphe G est **biparti** si l'ensemble S de ses sommets peut être divisé en deux sous-ensembles disjoints S_1 et S_2 tels que chaque arête de E (ensemble des arêtes) possède une extrémité dans S_1 et une autre dans S_2.
- Une **arène** (ou graphe du jeu) est un triplet (G, S_1, S_2) où G est un graphe orienté biparti.
  En pratique, S_1 représente les sommets contrôlés par le joueur J_1 (c'est-à-dire ceux pour lesquels c'est à lui de jouer) et S_2 ceux contrôlés par J_2. On prendra pour convention que J_1 contrôle le sommet s_0 représentant la position de départ, i.e. $s_0 \in S_1$.
- Une **partie** sur un tel graphe est un chemin $s_0, s_1, s_2, \ldots, s_n$ sur ce graphe. A chaque fois, si c'est possible, le joueur contrôlant le sommet choisit un arc vers un autre sommet. Si ce n'est pas possible, la partie s'arrête.

QUESTION : Donner un exemple de partie avec le graphe du jeu de Nim. Est-elle perdante ou gagnante pour J_1 ?

Remarque sur le nombre de parties possibles :

• Au Tic-Tac-Toe, en ne comptant qu'une fois toutes les parties qui sont identiques à rotation ou symétrie près, il existe 26830 parties possibles.

•Aux échecs, le nombre de parties possibles est de l'ordre de 10^{120}. (À titre de comparaison, la physique actuelle donne une estimation du nombre d'atomes dans l'univers observable de l'ordre de 10^{80})

1.2.2 Calcul des attracteurs

Définition 4
Dans un graphe biparti (G, S_1, S_2), **l'attracteur de J_1 est l'ensemble des positions gagnantes pour J_1.**

Pour déterminer cet attracteur, on va partir de l'ensemble des sommets terminaux sur lesquels J_1 gagne la partie, de "remonter le temps" pour savoir comment être sûr d'y arriver. (on va le faire pour J_1, on pourrait le faire de manière analogue pour J_2)
Plus précisément :

Proposition 2 - *Comment déterminer l'attracteur de J1*
Soit W_1 l'ensemble des positions finales gagnantes de J_1. On définit par récurrence la suite d'ensembles suivante :
- $A_0 = W_1$;
- $\forall j \in \mathbb{N}, \quad A_{j+1} = A_j \cup \{s \in S_1, \exists s' \in A_j, (s,s') \in E\} \cup \{s \in S_2, \forall s' \in S, (s,s') \in E \Rightarrow s' \in A_j\}$

L'attracteur A du joueur J_1 est alors défini comme $A = \bigcup_{j=0}^{\infty} A_j$

Ce qu'il faut comprendre : il s'agit d'une suite d'ensembles croissante pour l'inclusion (autrement dit, chaque ensemble contient le précédent), stationnaire à partir d'un certain rang puisque le nombre de sommets est fini, et pour laquelle chaque ensemble intermédiaire A_{j+1} contient les sommets de l'ensemble précédent A_j, auquel on ajoute d'une part les sommets de S_1 (donc contrôlés par J_1) pour lesquels il existe au moins un arc permettant de rejoindre un sommet de A_j, et d'autre part les sommets de S_2 (donc contrôlés par l'autre joueur) dont tous les arcs aboutissent à des sommets de A_j.

QUESTION : Sur le graphe biparti du jeu de Nim, colorier l'attracteur de J_1 et les arêtes qui définissent sa stratégie gagnante.

2 Un exemple de jeu plus complexe : Le jeu de Puissance 4

Dans ce cas le calcul des positions gagnantes n'est plus possible car il a une complexité beaucoup trop grande. Il devient donc nécessaire de s'appuyer non plus sur une évaluation exacte de chaque position (on ne va pas déterminer pour chaque position si elle est gagnante ou perdante) , mais sur **une estimation de la valeur de la position atteinte** : c'est la notion d'*heuristique*.

2.1 Heuristique associée à un jeu

Définition 5 - *Heuristique*
Nous allons attribuer à chaque position p de jeu un nombre noté $h(p)$, de sorte que **plus ce nombre est grand plus le joueur J_1 est susceptible de gagner**, et **plus le nombre est petit plus J_2 est susceptible de gagner**.
Par convention, si on sait la position p est gagnante pour J_1, on note $h(p) = +\infty$, et si la position p est gagnante pour J_2, on note $h(p) = -\infty$

Exemple 1 : Aux échecs, on peut attribuer à chacune des pièces un nombre de points (classiquement 1 par pion, 3 par fou et cavalier, 5 par tour, 9 par dame et 0 pour le roi), et faire la différence entre les points de J_1 et les points de J_2.
Par conséquent, si le jeu est dans une position p, plus h(p) est élevé et plus J_1 a de matériel que son adversaire ,et donc plus il est susceptible de gagner.

Dans toute la suite, nous allons travailler avec l'exemple du Puissance 4 : le but du jeu est d'aligner (horizontalement, verticalement ou en diagonale) une suite de quatre pions de même couleur sur une grille comptant six rangées et sept colonnes. Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, et le pion coulisse alors jusqu'à la position la plus basse possible. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé d'alignement, alors la partie est déclarée nulle.

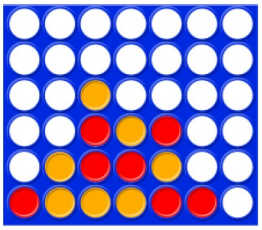


FIGURE 1 – Exemple de position de Puissance 4

Nous allons alors construire une heuristique de la façon suivante :

1. On attribue tout d'abord une valeur à chaque case, correspondant au nombre d'alignements potentiels de 4 pions lorsqu'on place un pion à cet emplacement. Voici donc les nombres attribués à chaque case :

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

FIGURE 2 – Nombres associés à chaque case

2. Pour calculer la valeur de l'heuristique $h(p)$ correspondant à une position de jeu p, on somme les nombres attribués à chaque case occupée par J_1, puis on soustrait les nombres attribués à chaque case occupée par J_2 ;

   Par exemple, pour la position ci-dessus, si on convient que les pions jaunes sont ceux du joueur 1, la valeur de l'heuristique est égale à $4+5+7+6+8+13+11-3-5-4-8-10-11-11=2$.

Évidemment, ces exemples de fonctions ne reflètent que très partiellement la réalité. On peut gagner une partie d'échecs avec moins de pièces que son adversaire si celles-ci sont bien placées. De même, au jeu de Puissance 4 , on peut très bien gagner avec un jeton dans le coin tandis que son adversaire qui a joué au milieu perd. Nous ne cherchons à décrire que partiellement la réalité.

2.2 L'algorithme Min-Max

2.2.1 Principe général

Au moment où l'un des deux protagonistes doit jouer, plusieurs possibilités s'offrent à lui. Une solution simple pour choisir le coup à jouer consiste à calculer l'heuristique correspondant à chacune des configurations atteignables, et à jouer celle d'heuristique maximale (pour J_1) ou minimale (pour J_2).

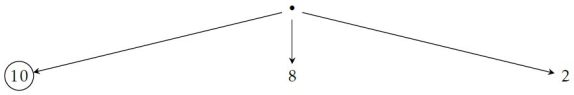


FIGURE 3 – Trois coups possibles pour J_1

Ainsi, dans l'exemple de la figure 3, J_1 choisira le coup le plus à gauche, correspondant à une évaluation égale à 10. Mais on peut aussi tenir compte du coup que va jouer J_2 ensuite, et donc calculer l'heuristique de chacune des positions que J_2 pourra atteindre. Si on observe la figure 4, on constate qu'il vaut mieux pour J_1 jouer le coup central, en partant du principe que J_2 joue au mieux son coup.

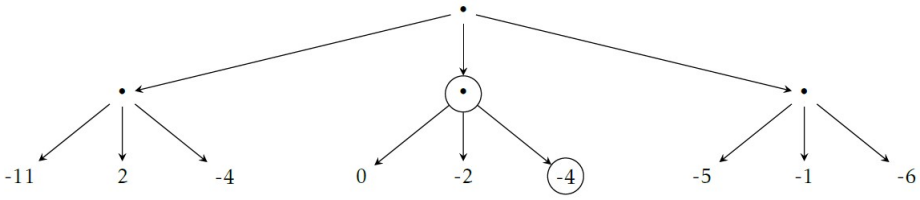


FIGURE 4 – C'est à J_1 de jouer, en tenant compte du coup que jouera J_2

Bien évidemment, on peut réitérer ce raisonnement et tenir compte du coup suivant, joué cette fois par J_1.
La figure 5 montre qu'en tenant compte des deux coups suivant, J_1 a en fait intérêt à jouer le coup le plus à droite :

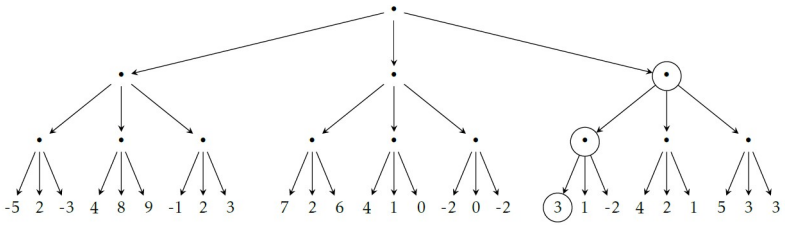
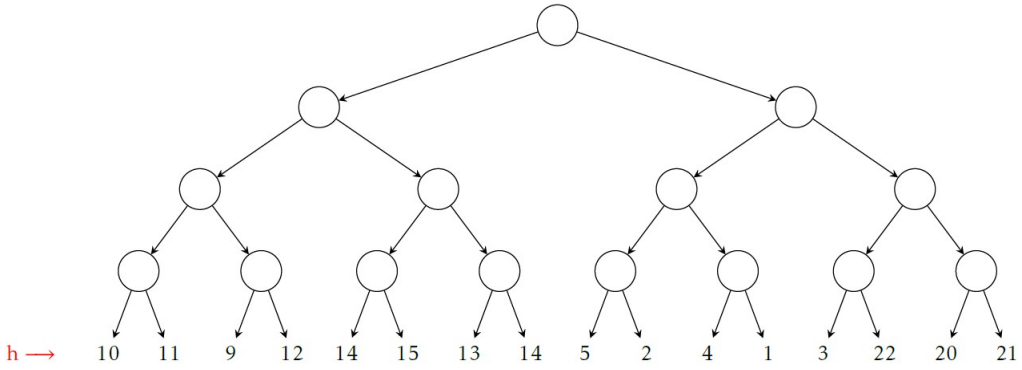


FIGURE 5 – C'est à J_1 de jouer, en anticipant sur les 3 coups suivants

QUESTION : C'est J_1 qui commence à jouer. Déterminer le chemin qu'il doit prendre dans l'arbre ci-dessous, en appliquant l'algorithme min-max avec la profondeur 4.



On peut répéter ce raisonnement, mais le nombre de configurations à examiner ayant tendance à croitre exponentiellement, il est nécessaire de limiter la profondeur de la recherche.

2.2.2 Implémentation de l'algorithme de min-max.

Notons n la profondeur de recherche (ce qui correspond à une anticipation de n coups).

Nous allons écrire deux fonctions :

- `maxiJ1(p,n)` (destinée au joueur 1) va chercher à maximiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux ;
- `miniJ2(p,n)` (destinée au joueur 2) va chercher à minimiser l'heuristique après n coups en partant de la position p, en supposant que son adversaire joue au mieux.

Pour mettre en place ces deux fonctions, nous allons supposer dans un premier temps que nous disposons des fonctions suivantes :

1. Une fonction `gagne`, de paramètres une position p et un numéro de joueur i, qui renvoie `True` si cette position possède un alignement de 4 pions du joueur i, `False` sinon.
2. Une fonction h, qui à toute position p, associe la valeur $h(p)$ de l'heuristique de cette position.
3. Une fonction `coups_possibles`, qui pour toute position p,renvoie la liste des coups possibles, sous la forme d'une liste de numéro de colonne où l'on peut placer un pion.
4. Une fonction `jouer`, de paramètres une position p, un numéro de colonne c (que l'on suppose valide pour pouvoir jouer) et un numéro de joueur i, qui renvoie la nouvelle position de jeu si le joueur i joue le coup c.

Programmation

Ecrire le script des deux fonctions `maxiJ1(p,n)` et `miniJ2(p,n)`, en suivant les principes suivants :

- Ces deux fonctions sont **mutuellement récursives** : pour calculer `maxiJ1(p,n)` on calcule pour chaque position $p1,...,pk$ atteignable à partir de p la valeur de `miniJ2(pi,n-1)` et on prend le maximum de ces valeurs.
  De manière symétrique, pour calculer `miniJ2(p,n)` on calcule pour chaque position $p1,...,pk$ atteignable à partir de p la valeur de `maxiJ1(pi,n-1)` et on prend le minimum de ces valeurs.
- Chacune de ces fonctions doit retourner un couple :
  - Le premier élément est la valeur associée à la position p une fois que l'algorithme aura tourné. (c'est cette valeur dont on va se servir pour bâtir la récursivité)
  - Le deuxième élément est le coup qu'il faut alors jouer pour prendre cette direction (ce coup correspond à un numéro de colonne, donc c'est un entier entre 0 et 6).
- Conditions d'arrêt :
  - Lorsque la profondeur vaut 0, alors on évalue la valeur de la position p grâce à l'heuristique $h(p)$, le deuxième élément renvoyé est None
  - Lorsque la position est gagnante pour J_1, `maxiJ1(p,n)` doit renvoyer le couple `(`$+\infty$`,None)`
  - Lorsque la position est gagnante pour J_2, `miniJ2(p,n)` doit renvoyer le couple `(`$-\infty$`,None)`
  - Lorsqu'aucun coup n'est possible à partir de la position p (grille pleine), la fonction doit renvoyer le couple `(h(p),None)`

2.3 Programmer le jeu Puissance 4

Une position sera représentée par une liste de liste : les places vides contiendront un 0, les pions de J_1 seront représentés par un 1, et ceux de J_2 par un 2. A l'affichage on convertira cette liste de liste en tableau numpy, ainsi une position de jeu pourra être par exemple :

```
[[ 0,  0,  0,  0,  0,  0,  0],
 [ 0,  0,  0,  0,  0,  0,  0],
 [ 0,  0,  0,  2,  0,  1,  0],
 [ 0,  0,  1,  1,  2,  1,  0],
 [ 0,  0,  2,  2,  1,  2,  0],
 [ 0,  1,  2,  1,  2,  1,  2]]
```

Programmation

- Créer une fonction `vide` sans arguments qui renvoie une grille vide (position de jeu au départ)
- Créer une fonction `pleine`, d'arguments une position et un numéro de colonne entre 0 et 6, qui renvoie un booléen indiquant si la colonne en question est pleine ou non.
- Créer la fonction `coups_possibles` décrite dans le paragraphe précédent.
- Créer la fonction `jouer` décrite dans le paragraphe précédent.
- (plus difficile) Créer la fonction `gagne` décrite dans le paragraphe précédent.
- Créer la fonction `h` décrite dans le paragraphe précédent.

On peut alors programmer une simulation de partie, par exemple en faisant jouer les deux joueurs au hasard, où en faisant jouer un joueur au hasard et l'autre qui utilise l'algorithme Min-Max pour optimiser ses coups.

2.4 Remarque sur la notion d'heuristique

Lors du cours sur les graphes, nous avons étudié l'algorithme de Dijkstra, permettant de calculer la plus courte distance d'un sommet à un autre (en calculant en fait la distance du sommet de départ à **tous** les autres sommets du graphe). On peut améliorer cet algorithme (pour le rendre plus rapide) en utilisant une heuristique (par exemple si les sommets ont des positions spatiales, on peut utiliser la "distance à vol d'oiseau" entre deux sommets) : on force alors l'algorithme à aller explorer certains sommets a priori plus intéressants que d'autres. Ce type d'algorithme s'appelle **A*** (lire *"A star"*).

3 Pour votre culture générale...

- Le jeu du Puissance 4 a été commercialisé pour la première fois en 1974, et a été entièrement résolu en 1988 : on connaît une stratégie gagnante pour le premier joueur, le premier coup de cette stratégie consistant à jouer dans la colonne du milieu.
- C'est grâce à une heuristique qu'un ordinateur a été capable de battre un humain aux échecs : en 1996, l'ordinateur IBM Deep Blue a battu le champion du monde Gary Kasparov.
- Il a fallu attendre 2015 et les réseaux de neurones de l'intelligence artificielle pour qu'un ordinateur batte un joueur humain professionnel au jeu de Go.