Théorie des jeux

Informatique tronc commun

Sylvain Pelletier

PSI - LMSC

Les buts de ce chapitre sont :

- Revoir le vocabulaire des graphes et des arbres.
  Le vocabulaire sera rappelé sur un sujet d'écrit mais il est toujours intéressant de se familiariser avec le formalisme.
- Revoir le travail fait en TP sur le jeu de Nims et le formaliser. (1ère partie)
- Étudier l'algorithme de min-max. (2ème partie)

Les buts de ce chapitre sont :

- Revoir le vocabulaire des graphes et des arbres.
  Le vocabulaire sera rappelé sur un sujet d'écrit mais il est toujours intéressant de se familiariser avec le formalisme.
- Revoir le travail fait en TP sur le jeu de Nims et le formaliser. (1ère partie)
- Étudier l'algorithme de min-max. (2ème partie)

Les buts de ce chapitre sont :

- Revoir le vocabulaire des graphes et des arbres.
  Le vocabulaire sera rappelé sur un sujet d'écrit mais il est toujours intéressant de se familiariser avec le formalisme.
- Revoir le travail fait en TP sur le jeu de Nims et le formaliser. (1ère partie)
- Étudier l'algorithme de min-max. (2ème partie)

Vocabulaire de la théorie des jeux

- On considère un jeu à deux joueurs (noté J_0 et J_1), pour lequel :
  - l'issue du jeu peut être la victoire de J_0, la victoire de J_1 ou la partie nulle.
  - Chaque joueur joue à son tour en modifiant L'ÉTAT DU JEU.
  - Il n'y a pas de hasard.
  - Les joueurs ont accès à la même information (pas de cartes cachées).
- On représente alors le jeu sous la forme d'un ARBRE DE JEU.
- La racine est la situation initiale. Une branche est une séquence de coups dans une partie. Les noeuds de l'arbre sont les positions dans le jeu, les arcs représentent les mouvements.

Vocabulaire de la théorie des jeux

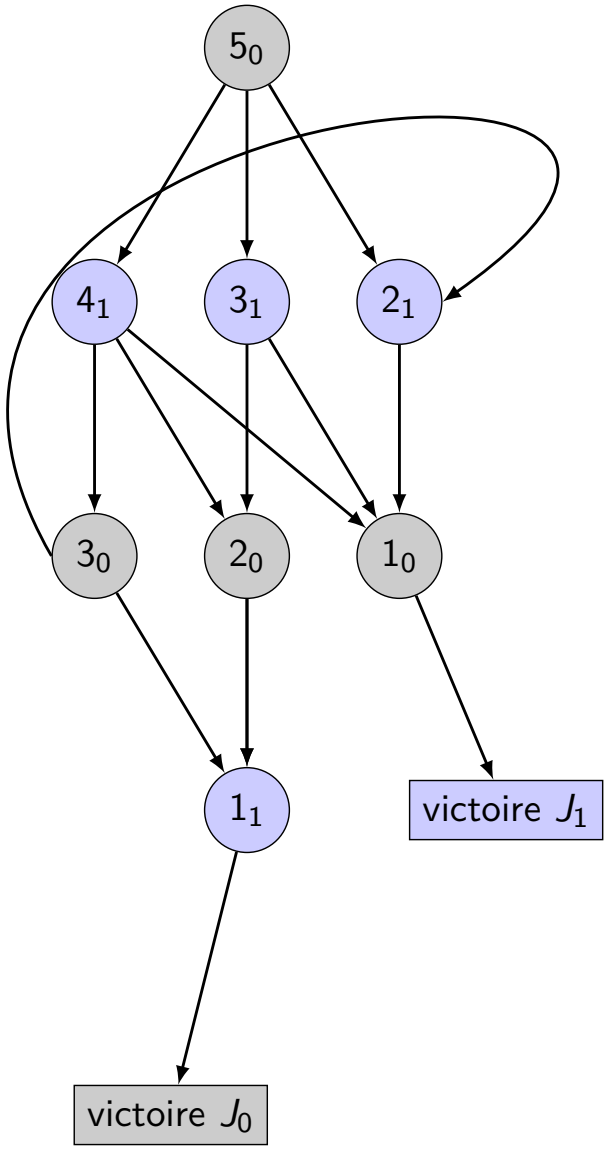
- On considère un jeu à deux joueurs (noté J_0 et J_1), pour lequel :
  - l'issue du jeu peut être la victoire de J_0, la victoire de J_1 ou la partie nulle.
  - Chaque joueur joue à son tour en modifiant L'ÉTAT DU JEU.
  - Il n'y a pas de hasard.
  - Les joueurs ont accès à la même information (pas de cartes cachées).
- On représente alors le jeu sous la forme d'un ARBRE DE JEU.
- La racine est la situation initiale. Une branche est une séquence de coups dans une partie. Les noeuds de l'arbre sont les positions dans le jeu, les arcs représentent les mouvements.

Vocabulaire de la théorie des jeux

- On considère un jeu à deux joueurs (noté J_0 et J_1), pour lequel :
  - l'issue du jeu peut être la victoire de J_0, la victoire de J_1 ou la partie nulle.
  - Chaque joueur joue à son tour en modifiant L'ÉTAT DU JEU.
  - Il n'y a pas de hasard.
  - Les joueurs ont accès à la même information (pas de cartes cachées).
- On représente alors le jeu sous la forme d'un ARBRE DE JEU.
- La racine est la situation initiale. Une branche est une séquence de coups dans une partie. Les noeuds de l'arbre sont les positions dans le jeu, les arcs représentent les mouvements.

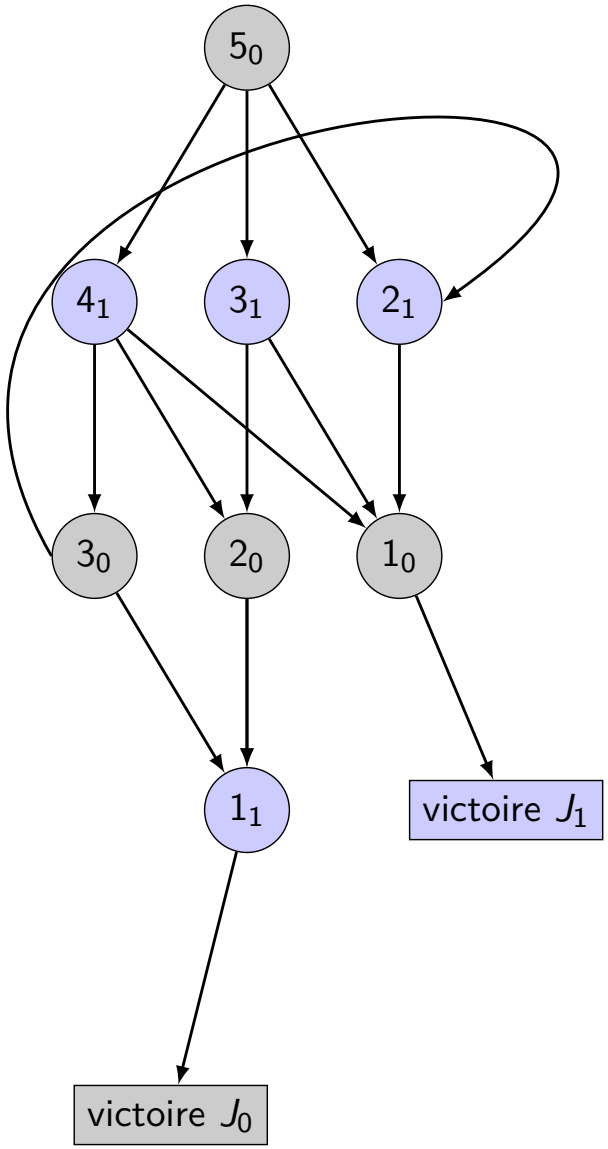
Arbre de jeu

- jeu de Nims : tas de N allumettes et chaque joueur peut prendre une, deux ou trois allumettes. Celui qui prend la dernière a perdu
- L'état du jeu est donc le couple : nombre d'allumettes, parité du numéro du tour (joueur 0 ou joueur 1).
- On parle de graphes bipartis puisque chaque état est contrôlé par le joueur 0 ou le joueur 1, c'est-à-dire que c'est au joueur 0 ou au joueur 1 de jouer.



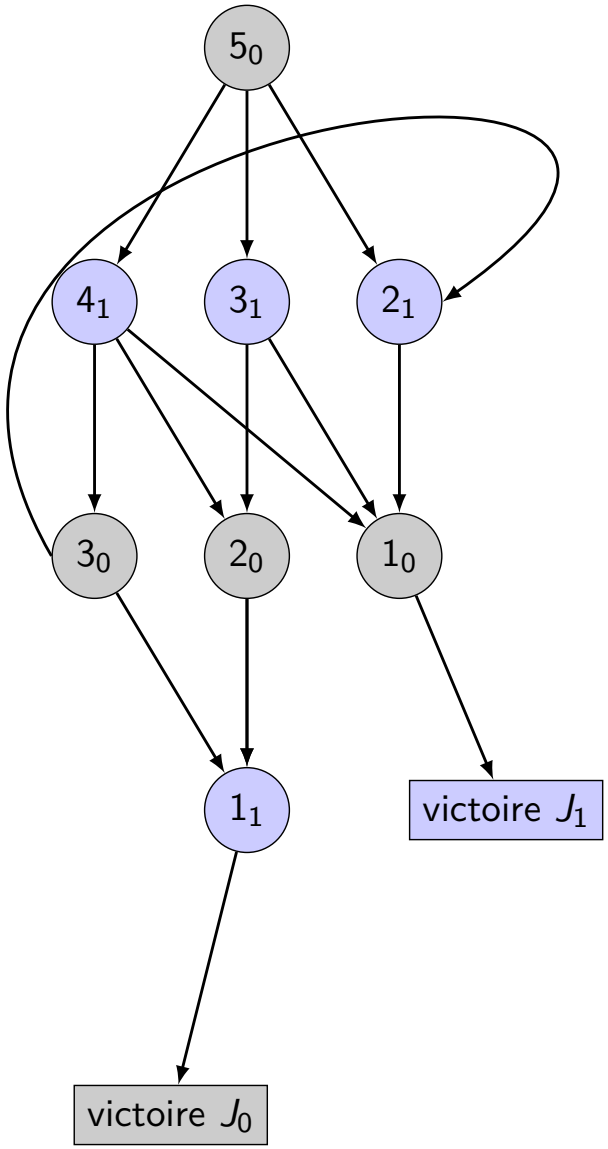
Arbre de jeu

- jeu de Nims : tas de N allumettes et chaque joueur peut prendre une, deux ou trois allumettes. Celui qui prend la dernière a perdu
- L'état du jeu est donc le couple : nombre d'allumettes, parité du numéro du tour (joueur 0 ou joueur 1).
- On parle de graphes bipartis puisque chaque état est contrôlé par le joueur 0 ou le joueur 1, c'est-à-dire que c'est au joueur 0 ou au joueur 1 de jouer.



Arbre de jeu

- jeu de Nims : tas de N allumettes et chaque joueur peut prendre une, deux ou trois allumettes. Celui qui prend la dernière a perdu
- L'état du jeu est donc le couple : nombre d'allumettes, parité du numéro du tour (joueur 0 ou joueur 1).
- On parle de graphes bipartis puisque chaque état est contrôlé par le joueur 0 ou le joueur 1, c'est-à-dire que c'est au joueur 0 ou au joueur 1 de jouer.



Définitions sur les graphes

- Un graphe orienté G est la donnée d'un ensemble fini S (ensemble des sommets) et d'une partie de S^2 (l'ensemble des arc). On note $G = (S, A)$.
- Pour un couple $(x, y) \in A$, x est l'origine de l'arc, et y est l'extrémité de l'arc.
- Le graphe n'est pas orienté si :
$$\forall (x, y) \in S^2, (y, x) \in A \iff (x, y) \in A$$
(autrement dit si les arc sont toutes à double sens). Dans ce cas, on parle plutôt d'arêtes que d'arc.
Dans le cas contraire, on dit donc que le graphe est orienté.
- Pour tout $s \in S$, les successeurs de s sont les extrémités des arcs dont s est l'origine.
Les prédécesseurs de s sont les origines des arcs dont s est l'extrémité.
- Un sommet est terminal si il n'a pas de successeur.

Définitions sur les graphes

- Un graphe orienté G est la donnée d'un ensemble fini S (ensemble des sommets) et d'une partie de S^2 (l'ensemble des arc). On note $G = (S, A)$.
- Pour un couple $(x, y) \in A$, x est l'origine de l'arc, et y est l'extrémité de l'arc.
- Le graphe n'est pas orienté si :
$$\forall (x, y) \in S^2, (y, x) \in A \iff (x, y) \in A$$
(autrement dit si les arc sont toutes à double sens). Dans ce cas, on parle plutôt d'arêtes que d'arc.
Dans le cas contraire, on dit donc que le graphe est orienté.
- Pour tout $s \in S$, les successeurs de s sont les extrémités des arcs dont s est l'origine.
Les prédécesseurs de s sont les origines des arcs dont s est l'extrémité.
- Un sommet est terminal si il n'a pas de successeur.

Définitions sur les graphes

- Un graphe orienté G est la donnée d'un ensemble fini S (ensemble des sommets) et d'une partie de S^2 (l'ensemble des arc). On note $G = (S, A)$.
- Pour un couple $(x, y) \in A$, x est l'origine de l'arc, et y est l'extrémité de l'arc.
- Le graphe n'est pas orienté si :
$$\forall (x, y) \in S^2, (y, x) \in A \iff (x, y) \in A$$
(autrement dit si les arc sont toutes à double sens). Dans ce cas, on parle plutôt d'arêtes que d'arc.
Dans le cas contraire, on dit donc que le graphe est orienté.
- Pour tout $s \in S$, les successeurs de s sont les extrémités des arcs dont s est l'origine.
Les prédécesseurs de s sont les origines des arcs dont s est l'extrémité.
- Un sommet est terminal si il n'a pas de successeur.

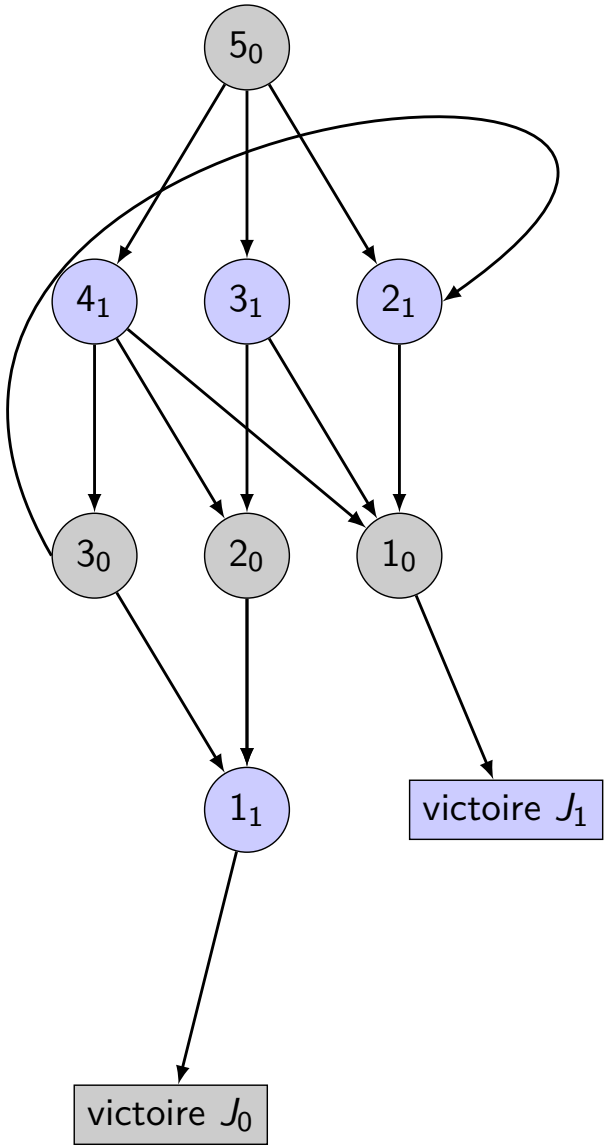
Définitions sur les graphes

- Un graphe orienté G est la donnée d'un ensemble fini S (ensemble des sommets) et d'une partie de S^2 (l'ensemble des arc). On note $G = (S, A)$.
- Pour un couple $(x, y) \in A$, x est l'origine de l'arc, et y est l'extrémité de l'arc.
- Le graphe n'est pas orienté si :
$$\forall (x, y) \in S^2, (y, x) \in A \iff (x, y) \in A$$
(autrement dit si les arc sont toutes à double sens). Dans ce cas, on parle plutôt d'arêtes que d'arc.
Dans le cas contraire, on dit donc que le graphe est orienté.
- Pour tout $s \in S$, les successeurs de s sont les extrémités des arcs dont s est l'origine.
Les prédécesseurs de s sont les origines des arcs dont s est l'extrémité.
- Un sommet est terminal si il n'a pas de successeur.

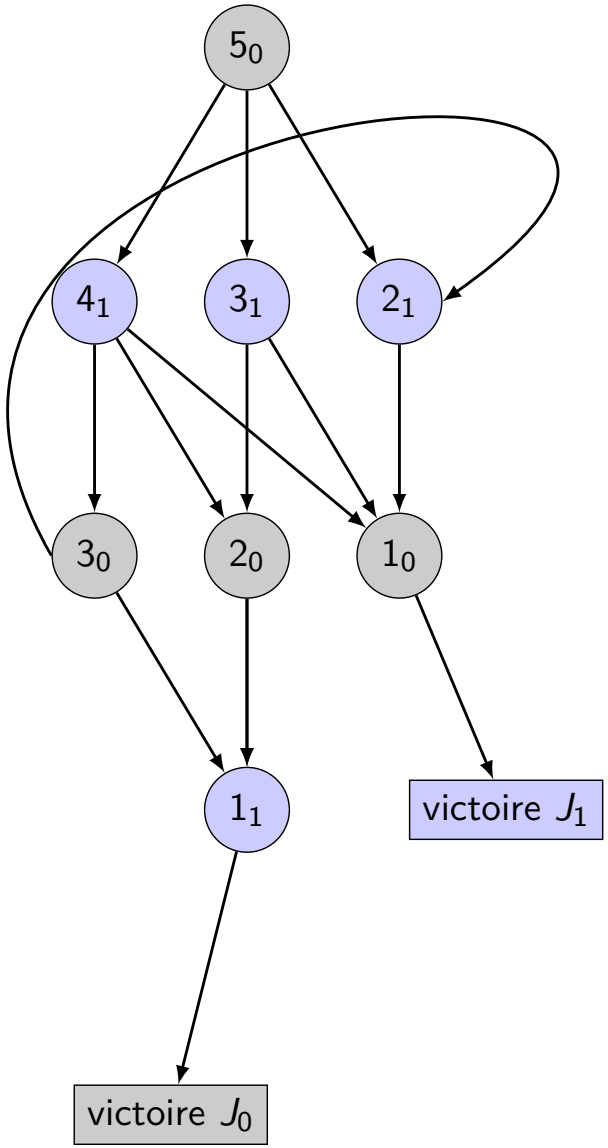
Définitions sur les graphes

- Un graphe orienté G est la donnée d'un ensemble fini S (ensemble des sommets) et d'une partie de S^2 (l'ensemble des arc). On note $G = (S, A)$.
- Pour un couple $(x, y) \in A$, x est l'origine de l'arc, et y est l'extrémité de l'arc.
- Le graphe n'est pas orienté si :
$$\forall (x, y) \in S^2, (y, x) \in A \iff (x, y) \in A$$
(autrement dit si les arc sont toutes à double sens). Dans ce cas, on parle plutôt d'arêtes que d'arc.
Dans le cas contraire, on dit donc que le graphe est orienté.
- Pour tout $s \in S$, les successeurs de s sont les extrémités des arcs dont s est l'origine.
Les prédécesseurs de s sont les origines des arcs dont s est l'extrémité.
- Un sommet est terminal si il n'a pas de successeur.

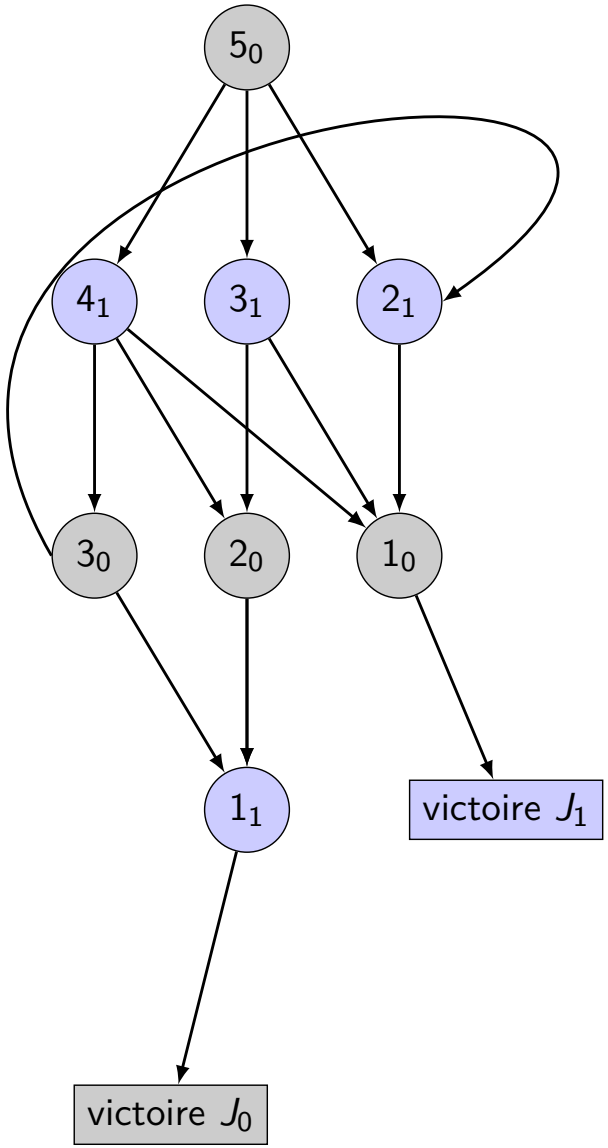
- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0,4_1,3_1,2_1,\ 3_0,2_0,1_0,2_1,\ 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



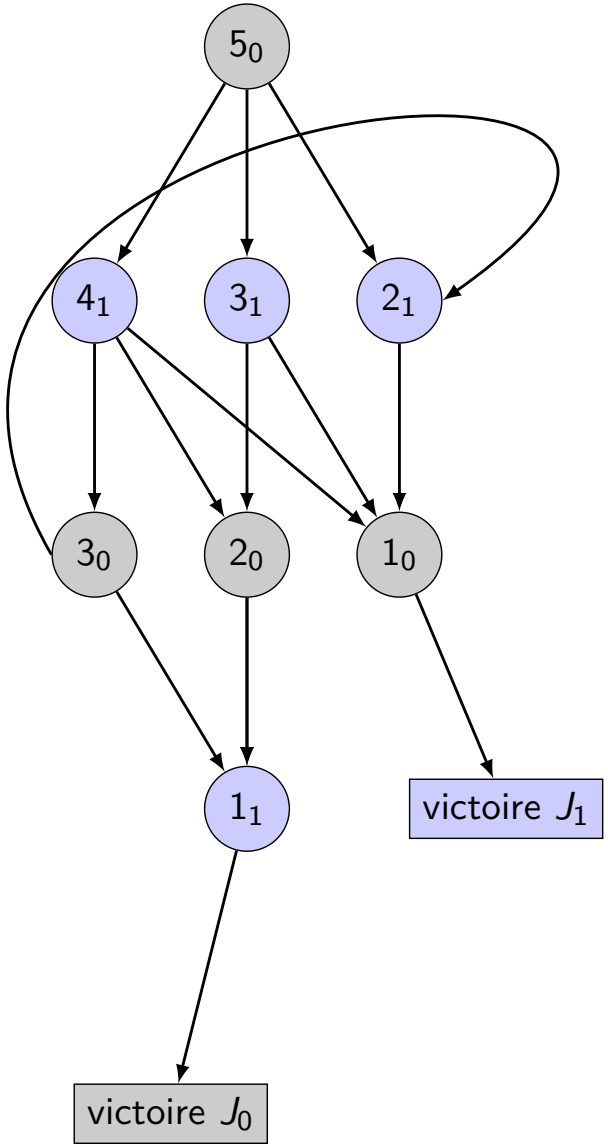
- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0,4_1,3_1,2_1,\ 3_0,2_0,1_0,2_1,\ 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



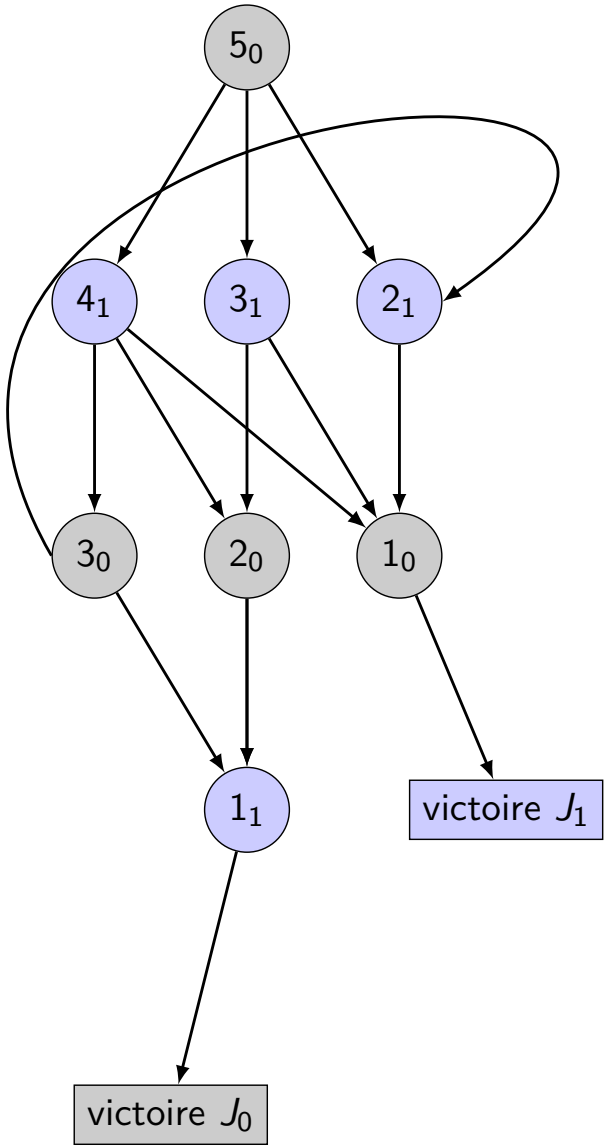
- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0, 4_1, 3_1, 2_1, 3_0, 2_0, 1_0, 2_1, 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



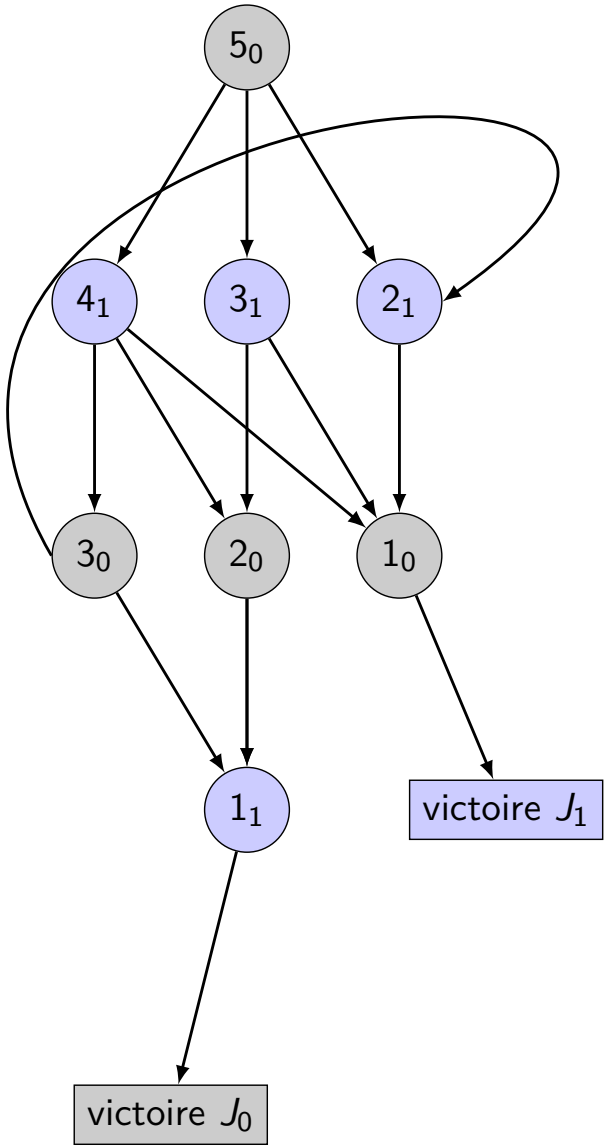
- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0, 4_1, 3_1, 2_1, 3_0, 2_0, 1_0, 2_1, 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0,4_1,3_1,2_1,\ 3_0,2_0,1_0,2_1,\ 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



- Il n'y pas de cycle, ni de noeud isolé (le graphe est connexe).
- $(5_0,4_1,3_1,2_1,\ 3_0,2_0,1_0,2_1,\ 1_1)$, Victoire J_1, Victoire J_0 est l'ensemble des sommets.
- 5_0 est l'origine,
- $(5_0, 4_1)$ est un arc.
- 4_1 est un successeurs de 5_0. 2_1 est un prédécesseur de 1_0.
- Victoire de J_0 est un sommet terminal.



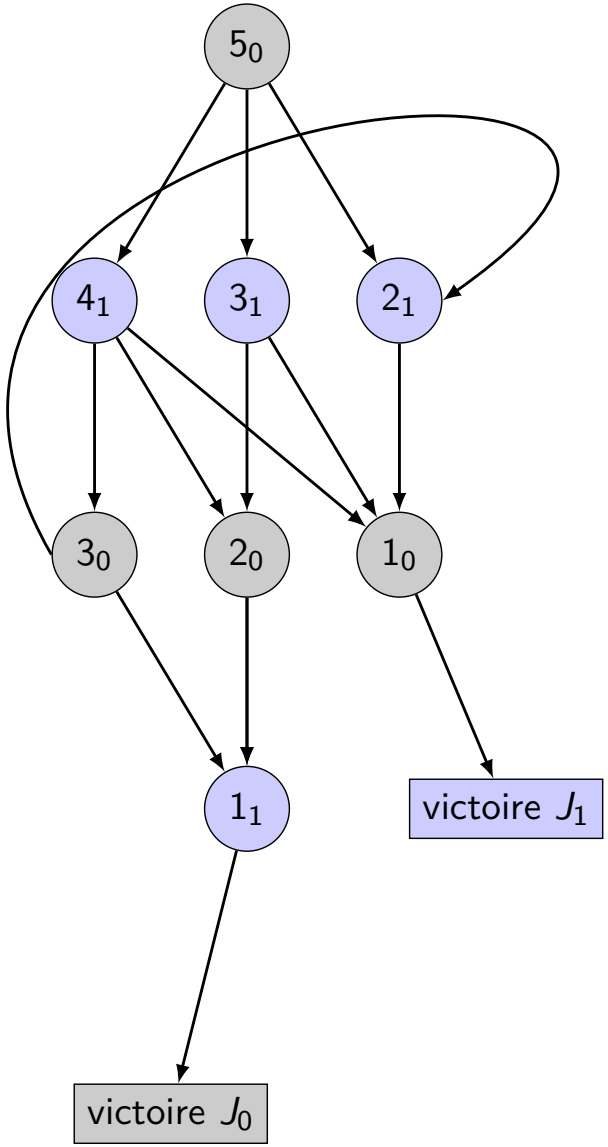
- Le graphe $G = (S, A)$ est **biparti** si il existe deux sous-ensembles de sommets S_0 et S_1, formant une partition des sommets S, tel que pour toute arrête $(x, y) \in A$, l'origine x et l'extrémité y de l'arc ne sont pas dans la même partie (ie $x \in S_0 \iff y \in S_1$).
- On parle de graphe de jeu à deux joueurs ou arène pour désigner un graphe fini et biparti.
- On définit aussi les conditions de gains pour le joueur J_0 : c'est un sous-ensembles des sommets terminaux où J_0 a gagné, et idem pour le joueur J_1.
- Lorsqu'un joueur joue, le jeu est à un des sommets s du graphe (contrôlé par J_0 ou J_1 selon qui a le trait).
  Il y a deux possibilités :
  - si s est terminal alors le joueur ne peut plus jouer, la partie est finie.
  - il choisit un arc (s, s') d'origine s dans la graphe. s' est contrôlé par l'autre joueur.

- Le graphe $G = (S, A)$ est **biparti** si il existe deux sous-ensembles de sommets S_0 et S_1, formant une partition des sommets S, tel que pour toute arrête $(x, y) \in A$, l'origine x et l'extrémité y de l'arc ne sont pas dans la même partie (ie $x \in S_0 \iff y \in S_1$).
- On parle de graphe de jeu à deux joueurs ou arène pour désigner un graphe fini et biparti.
- On définit aussi les conditions de gains pour le joueur J_0 : c'est un sous-ensembles des sommets terminaux où J_0 a gagné, et idem pour le joueur J_1.
- Lorsqu'un joueur joue, le jeu est à un des sommets s du graphe (contrôlé par J_0 ou J_1 selon qui a le trait).
  Il y a deux possibilités :
  - si s est terminal alors le joueur ne peut plus jouer, la partie est finie.
  - il choisit un arc (s, s') d'origine s dans la graphe. s' est contrôlé par l'autre joueur.

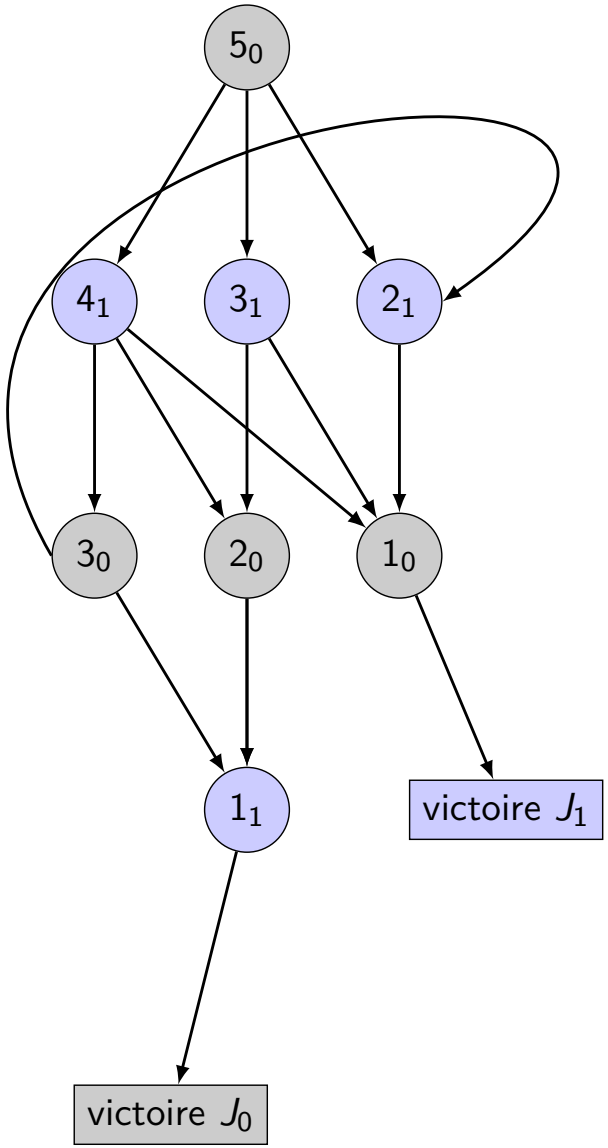
- Le graphe $G = (S, A)$ est **biparti** si il existe deux sous-ensembles de sommets S_0 et S_1, formant une partition des sommets S, tel que pour toute arrête $(x, y) \in A$, l'origine x et l'extrémité y de l'arc ne sont pas dans la même partie (ie $x \in S_0 \iff y \in S_1$).
- On parle de graphe de jeu à deux joueurs ou arène pour désigner un graphe fini et biparti.
- On définit aussi les conditions de gains pour le joueur J_0 : c'est un sous-ensembles des sommets terminaux où J_0 a gagné, et idem pour le joueur J_1.
- Lorsqu'un joueur joue, le jeu est à un des sommets s du graphe (contrôlé par J_0 ou J_1 selon qui a le trait).
  Il y a deux possibilités :
  - si s est terminal alors le joueur ne peut plus jouer, la partie est finie.
  - il choisit un arc (s, s') d'origine s dans la graphe. s' est contrôlé par l'autre joueur.

- Le graphe $G = (S, A)$ est **biparti** si il existe deux sous-ensembles de sommets S_0 et S_1, formant une partition des sommets S, tel que pour toute arrête $(x, y) \in A$, l'origine x et l'extrémité y de l'arc ne sont pas dans la même partie (ie $x \in S_0 \iff y \in S_1$).
- On parle de graphe de jeu à deux joueurs ou arène pour désigner un graphe fini et biparti.
- On définit aussi les conditions de gains pour le joueur J_0 : c'est un sous-ensembles des sommets terminaux où J_0 a gagné, et idem pour le joueur J_1.
- Lorsqu'un joueur joue, le jeu est à un des sommets s du graphe (contrôlé par J_0 ou J_1 selon qui a le trait).
  Il y a deux possibilités :
  - si s est terminal alors le joueur ne peut plus jouer, la partie est finie.
  - il choisit un arc (s, s') d'origine s dans la graphe. s' est contrôlé par l'autre joueur.

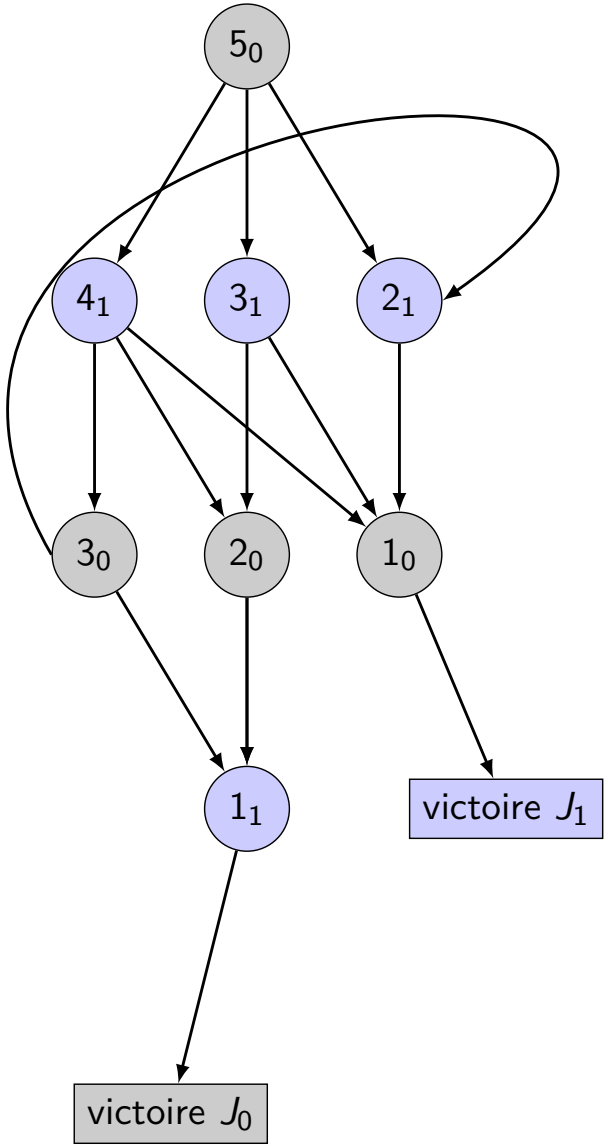
- Un jeu à deux joueurs = graphe biparti.
- Les conditions de gains sont les sommets terminaux : "victoire J_1", "victoire J_0".
- Le fait de considérer les conditions de gains comme des sommets terminaux signifie que le fait de gagner ne dépend que de la position, non de la séquence de coups précédents.



- Un jeu à deux joueurs = graphe biparti.
- Les conditions de gains sont les sommets terminaux : "victoire J_1", "victoire J_0".
- Le fait de considérer les conditions de gains comme des sommets terminaux signifie que le fait de gagner ne dépend que de la position, non de la séquence de coups précédents.



- Un jeu à deux joueurs = graphe biparti.
- Les conditions de gains sont les sommets terminaux : "victoire J_1", "victoire J_0".
- `Le fait de considérer les conditions de gains comme des sommets terminaux signifie que le fait de gagner ne dépend que de la position, non de la séquence de coups précédents.`



Parties et stratégies

- Un chemin dans la graphe orienté G est une suite (finie ou non) de sommets $(s_i)_{i \in I}$ reliés par des arcs, ie :
$$\forall i \in I,\ (s_i, s_{i+1}) \in A$$
- Un chemin fini est maximal lorsque l'extrémité de son dernier arc est terminale.
- Dans un graphe de jeu, une partie finie débutant en s_0 (position de départ) est un chemin fini maximal de G dont le premier sommet est s_0.
- Une partie partielle débutant en s_0 est un chemin fini dont le premier sommet est s_0.

Parties et stratégies

- Un chemin dans la graphe orienté G est une suite (finie ou non) de sommets $(s_i)_{i \in I}$ reliés par des arcs, ie :
$$\forall i \in I,\ (s_i, s_{i+1}) \in A$$
- Un chemin fini est maximal lorsque l'extrémité de son dernier arc est terminale.
- Dans un graphe de jeu, une partie finie débutant en s_0 (position de départ) est un chemin fini maximal de G dont le premier sommet est s_0.
- Une partie partielle débutant en s_0 est un chemin fini dont le premier sommet est s_0.

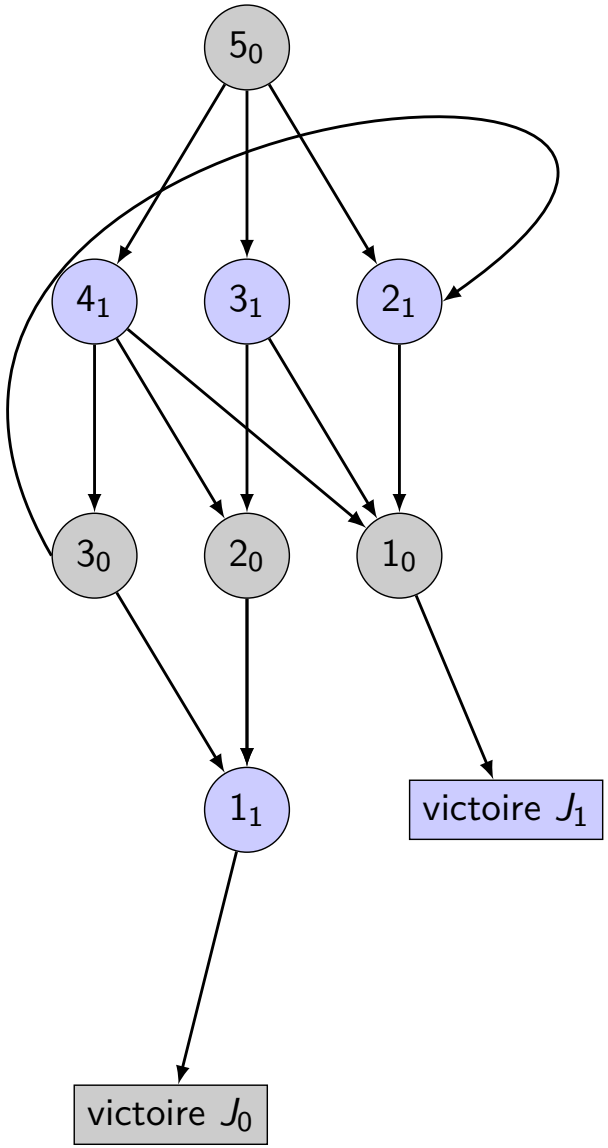
Parties et stratégies

- Un chemin dans la graphe orienté G est une suite (finie ou non) de sommets $(s_i)_{i \in I}$ reliés par des arcs, ie :
$$\forall i \in I,\ (s_i, s_{i+1}) \in A$$
- Un chemin fini est maximal lorsque l'extrémité de son dernier arc est terminale.
- Dans un graphe de jeu, une partie finie débutant en s_0 (position de départ) est un chemin fini maximal de G dont le premier sommet est s_0.
- Une partie partielle débutant en s_0 est un chemin fini dont le premier sommet est s_0.

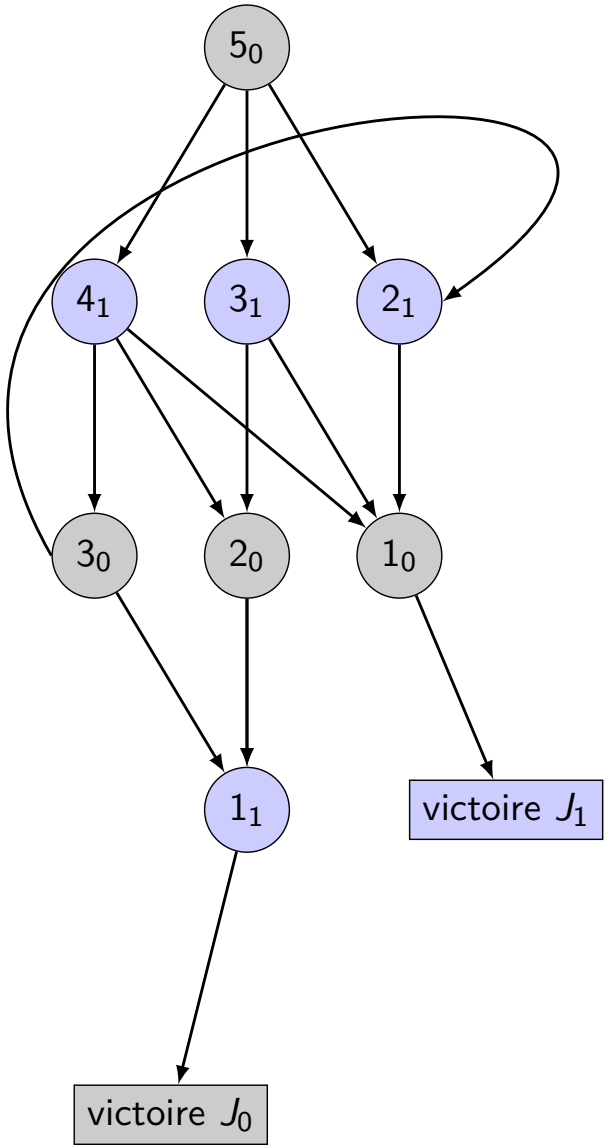
Parties et stratégies

- Un chemin dans la graphe orienté G est une suite (finie ou non) de sommets $(s_i)_{i \in I}$ reliés par des arcs, ie :
$$\forall i \in I,\ (s_i, s_{i+1}) \in A$$
- Un chemin fini est maximal lorsque l'extrémité de son dernier arc est terminale.
- Dans un graphe de jeu, une partie finie débutant en s_0 (position de départ) est un chemin fini maximal de G dont le premier sommet est s_0.
- Une partie partielle débutant en s_0 est un chemin fini dont le premier sommet est s_0.

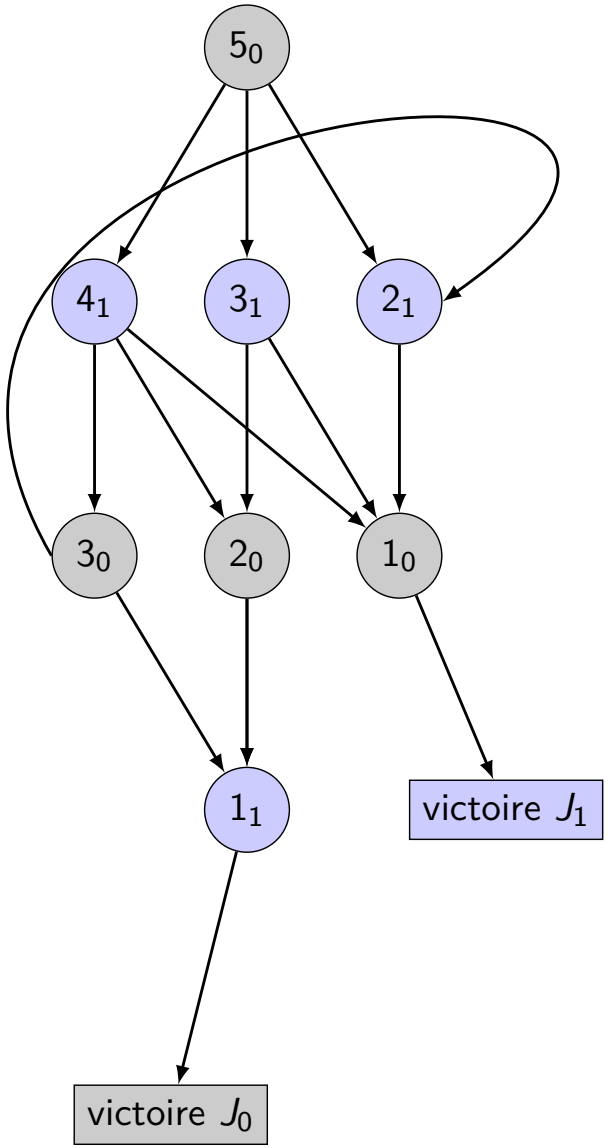
- $(3_1, 2_0, 1_1)$ est un chemin,
- $(3_1, 2_0, 1_1, \text{Victoire } J_0)$ est un chemin maximal = finit par un sommet terminal.
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie finie.
- $(5_0, 3_1, 2_0)$ est une partie partielle = commence par la position initiale.



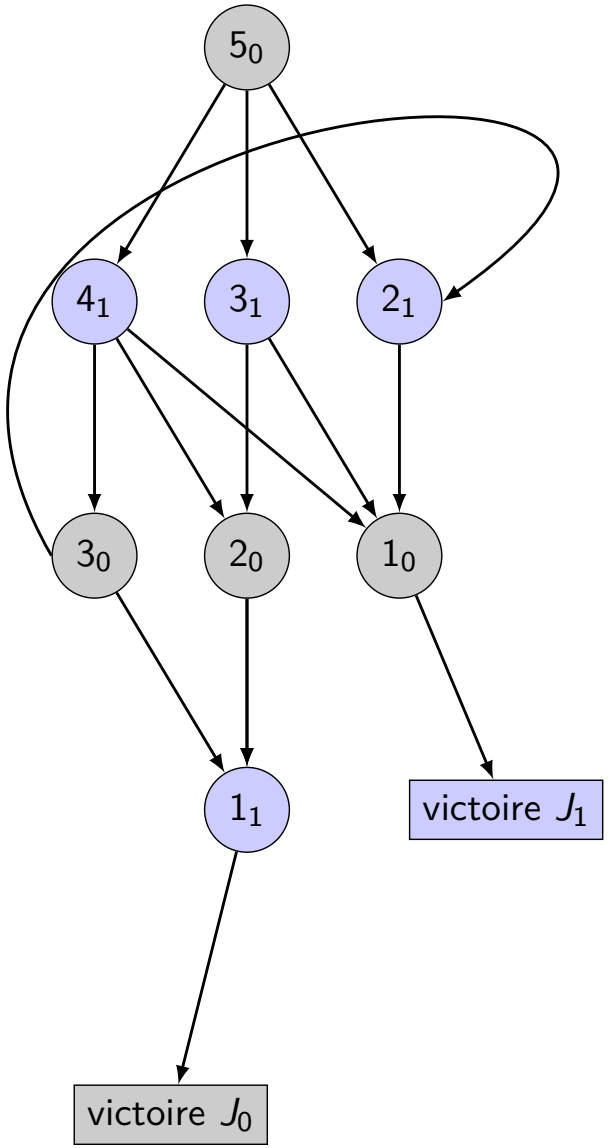
- $(3_1, 2_0, 1_1)$ est un chemin,
- $(3_1, 2_0, 1_1, \text{Victoire } J_0)$ est un chemin maximal = finit par un sommet terminal.
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie finie.
- $(5_0, 3_1, 2_0)$ est une partie partielle = commence par la position initiale.



- $(3_1, 2_0, 1_1)$ est un chemin,
- $(3_1, 2_0, 1_1, \text{Victoire } J_0)$ est un chemin maximal = finit par un sommet terminal.
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie finie.
- $(5_0, 3_1, 2_0)$ est une partie partielle = commence par la position initiale.



- $(3_1, 2_0, 1_1)$ est un chemin,
- $(3_1, 2_0, 1_1, \text{Victoire } J_0)$ est un chemin maximal = finit par un sommet terminal.
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie finie.
- $(5_0, 3_1, 2_0)$ est une partie partielle = commence par la position initiale.



Stratégie sans mémoire

Dans un graphe de jeu $((S, A), S_0, S_1)$:

- Une **stratégie (sans mémoire)** pour le joueur J_0 est une application :

$$\sigma : \begin{cases} S_0 & \to & S_1 \\ x & \longmapsto & y \end{cases}$$

qui à une position x du jeu où le joueur 0 doit jouer, associe un sommet y (le choix d'un coup), avec $(x, y) \in A$ (ie le coup est jouable).

- Une stratégie permet ainsi de savoir quel coup jouer (de manière unique car σ est une application) étant donné une situation de jeu.
- Une partie $(s_0, \ldots, s_{2t})$ est **conforme à une stratégie** pour le joueur 0 si :

$$s_1 = \sigma(s_0),\ s_3 = \sigma(s_2),\ \ldots$$

(autrement dit, le joueur J_0 a toujours utilisé l'application σ pour choisir son coup).

Stratégie sans mémoire

Dans un graphe de jeu $((S, A), S_0, S_1)$:

- Une stratégie (sans mémoire) pour le joueur J_0 est une application :

$$\sigma : \begin{cases} S_0 & \to & S_1 \\ x & \longmapsto & y \end{cases}$$

qui à une position x du jeu où le joueur 0 doit jouer, associe un sommet y (le choix d'un coup), avec $(x, y) \in A$ (ie le coup est jouable).

- Une stratégie permet ainsi de savoir quel coup jouer (de manière unique car σ est une application) étant donné une situation de jeu.
- Une partie $(s_0, \ldots, s_{2t})$ est conforme à une stratégie pour le joueur 0 si :

$$s_1 = \sigma(s_0),\ s_3 = \sigma(s_2),\ \ldots$$

(autrement dit, le joueur J_0 a toujours utilisé l'application σ pour choisir son coup).

Stratégie sans mémoire

Dans un graphe de jeu $((S, A), S_0, S_1)$:

- Une stratégie (sans mémoire) pour le joueur J_0 est une application :

$$\sigma : \begin{cases} S_0 & \to & S_1 \\ x & \longmapsto & y \end{cases}$$

qui à une position x du jeu où le joueur 0 doit jouer, associe un sommet y (le choix d'un coup), avec $(x, y) \in A$ (ie le coup est jouable).

- Une stratégie permet ainsi de savoir quel coup jouer (de manière unique car σ est une application) étant donné une situation de jeu.
- Une partie $(s_0, \ldots, s_{2t})$ est conforme à une stratégie pour le joueur 0 si :

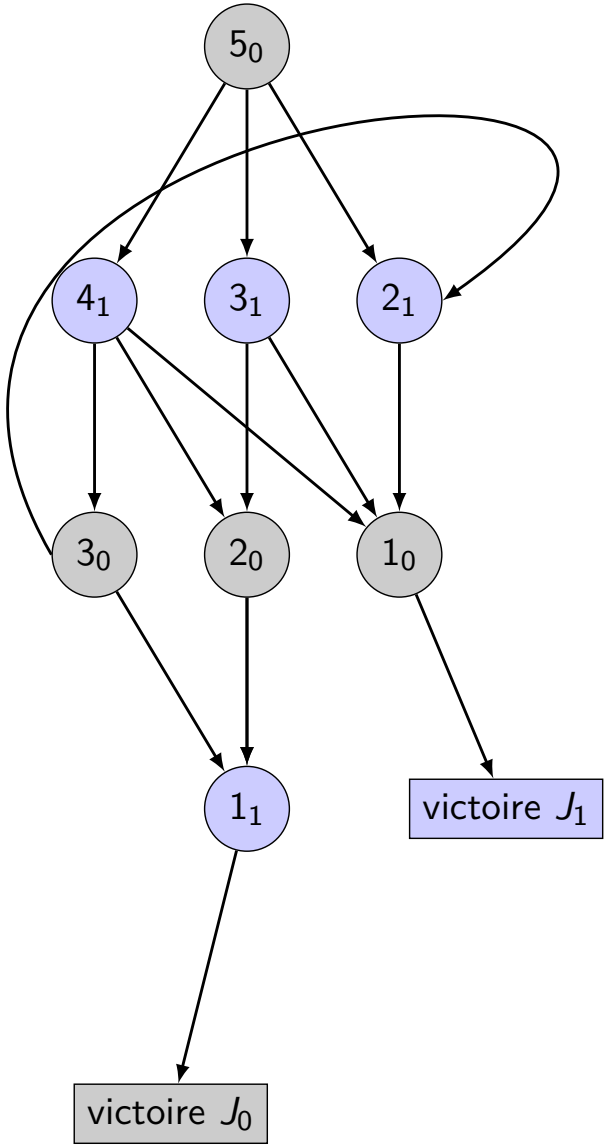
$$s_1 = \sigma(s_0),\ s_3 = \sigma(s_2),\ \ldots$$

(autrement dit, le joueur J_0 a toujours utilisé l'application σ pour choisir son coup).

- Une stratégie pour le joueur 0 est :

$$5_0 \mapsto 3_1,\ 3_0 \mapsto 1_1,\ 2_0 \mapsto 1_1,$$
$$1_0 \mapsto \text{Victoire } J_1,$$

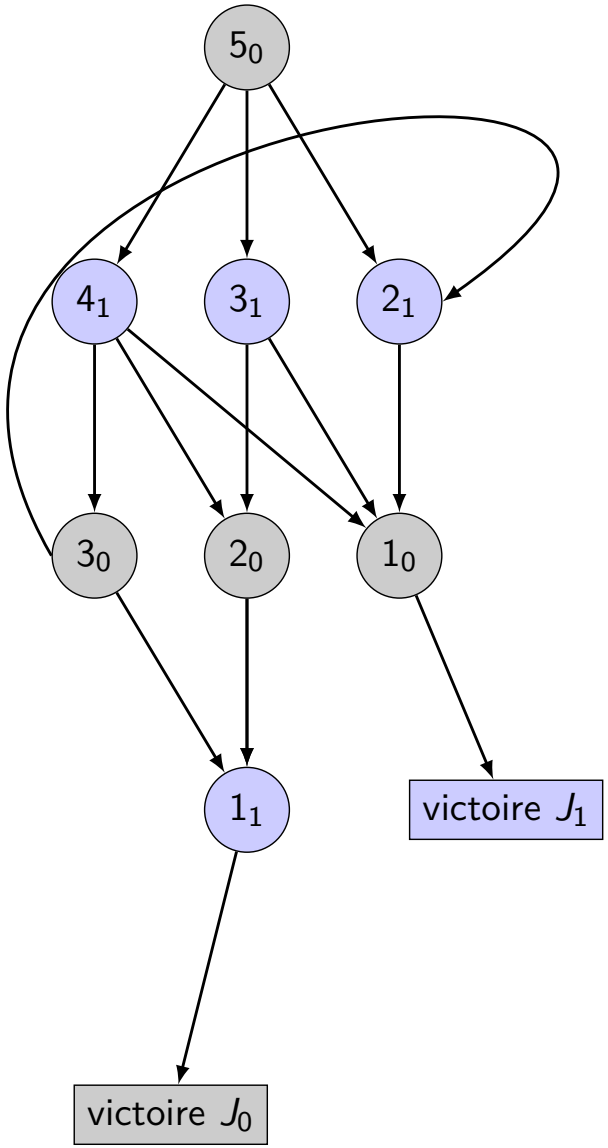
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie conforme à cette stratégie.
- Le but de l'IA est de proposer des stratégies.



- Une stratégie pour le joueur 0 est :

$$5_0 \mapsto 3_1,\ 3_0 \mapsto 1_1,\ 2_0 \mapsto 1_1,$$
$$1_0 \mapsto \text{Victoire } J_1,$$

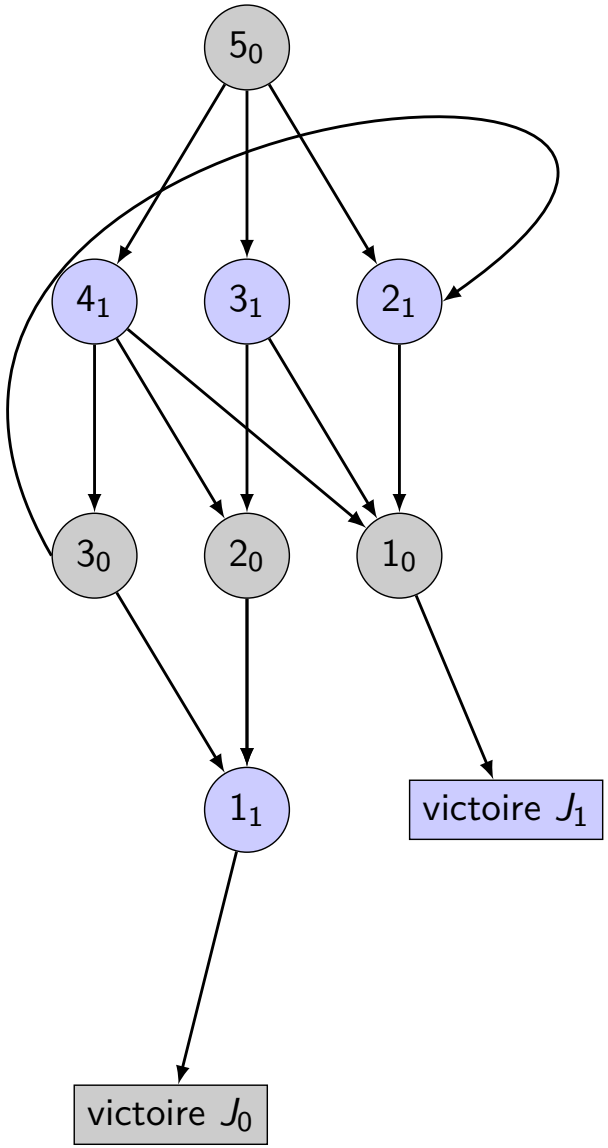
- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie conforme à cette stratégie.
- Le but de l'IA est de proposer des stratégies.



- Une stratégie pour le joueur 0 est :

$$5_0 \mapsto 3_1,\ 3_0 \mapsto 1_1,\ 2_0 \mapsto 1_1,$$
$$1_0 \mapsto \text{Victoire } J_1,$$

- $(5_0, 3_1, 2_0, 1_1, \text{Victoire } J_0)$ est une partie conforme à cette stratégie.
- Le but de l'IA est de proposer des stratégies.



Stratégie sans mémoire

- On définit de même la notion de stratégie sans mémoire pour le joueur J_1 et la notion de partie conforme à une stratégie pour le joueur 1.
- Une fois choisir une stratégie pour le joueur 0, une position $d \in S$ est gagnante pour le joueur J_0 lorsque toute partie conforme à la stratégie et qui débute en d est gagné par J_0 (ie finit par passer par un sommet qui vérifie la condition de gain pour le joueur 0).
- Une stratégie est gagnante pour le joueur J_0 si toute partie conforme à la stratégie qui commence par la position de départ est gagnante pour J_0

`On se restreint aux stratégies sans mémoire` : il suffit d'observer la position sur le jeu pour décider quel coup jouer (sans avoir besoin de l'historique des coups précédents).

Stratégie sans mémoire

- On définit de même la notion de stratégie sans mémoire pour le joueur J_1 et la notion de partie conforme à une stratégie pour le joueur 1.
- Une fois choisir une stratégie pour le joueur 0, une position $d \in S$ est gagnante pour le joueur J_0 lorsque toute partie conforme à la stratégie et qui débute en d est gagné par J_0 (ie finit par passer par un sommet qui vérifie la condition de gain pour le joueur 0).
- Une stratégie est gagnante pour le joueur J_0 si toute partie conforme à la stratégie qui commence par la position de départ est gagnante pour J_0

`On se restreint aux stratégies sans mémoire` : il suffit d'observer la position sur le jeu pour décider quel coup jouer (sans avoir besoin de l'historique des coups précédents).

Stratégie sans mémoire

- On définit de même la notion de stratégie sans mémoire pour le joueur J_1 et la notion de partie conforme à une stratégie pour le joueur 1.
- Une fois choisir une stratégie pour le joueur 0, une position $d \in S$ est gagnante pour le joueur J_0 lorsque toute partie conforme à la stratégie et qui débute en d est gagné par J_0 (ie finit par passer par un sommet qui vérifie la condition de gain pour le joueur 0).
- Une stratégie est gagnante pour le joueur J_0 si toute partie conforme à la stratégie qui commence par la position de départ est gagnante pour J_0

`On se restreint aux stratégies sans mémoire` : il suffit d'observer la position sur le jeu pour décider quel coup jouer (sans avoir besoin de l'historique des coups précédents).

Stratégie sans mémoire

- On définit de même la notion de stratégie sans mémoire pour le joueur J_1 et la notion de partie conforme à une stratégie pour le joueur 1.
- Une fois choisir une stratégie pour le joueur 0, une position $d \in S$ est gagnante pour le joueur J_0 lorsque toute partie conforme à la stratégie et qui débute en d est gagné par J_0 (ie finit par passer par un sommet qui vérifie la condition de gain pour le joueur 0).
- Une stratégie est gagnante pour le joueur J_0 si toute partie conforme à la stratégie qui commence par la position de départ est gagnante pour J_0

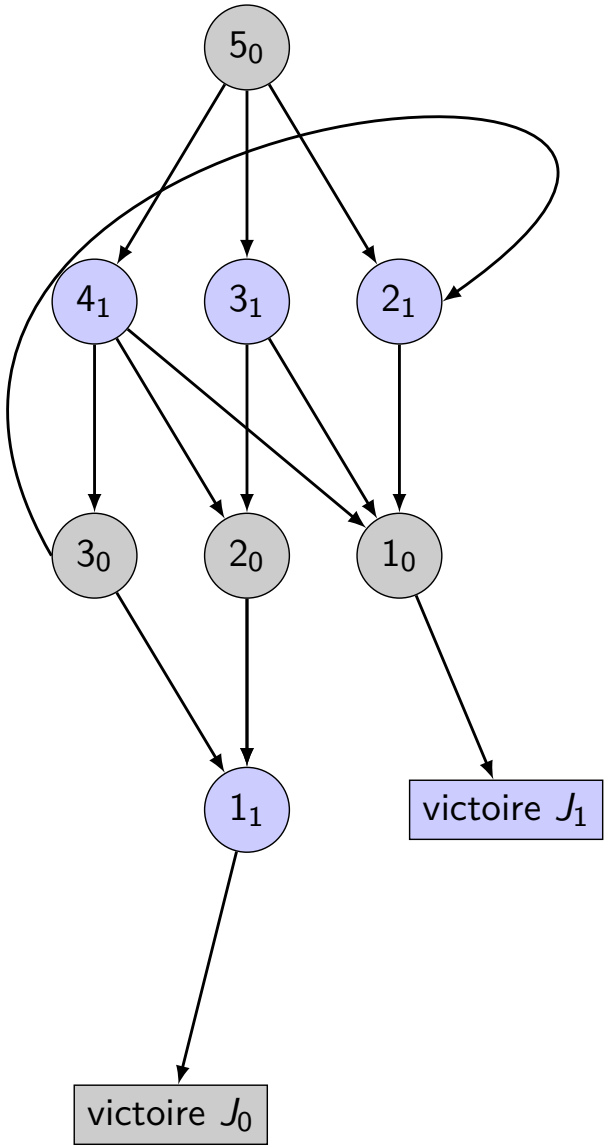
`On se restreint aux stratégies sans mémoire` : il suffit d'observer la position sur le jeu pour décider quel coup jouer (sans avoir besoin de l'historique des coups précédents).

- La stratégie pour le joueur 0 :

$$5_0 \mapsto 3_1,\ 3_0 \mapsto 1_1,\ 2_0 \mapsto 1_1,$$
$$1_0 \mapsto \text{Victoire } J_1,$$

est gagnante à partir de la position 3_0.

- Il n'existe pas de stratégie gagnante pour le joueur 0 depuis la position de départ.

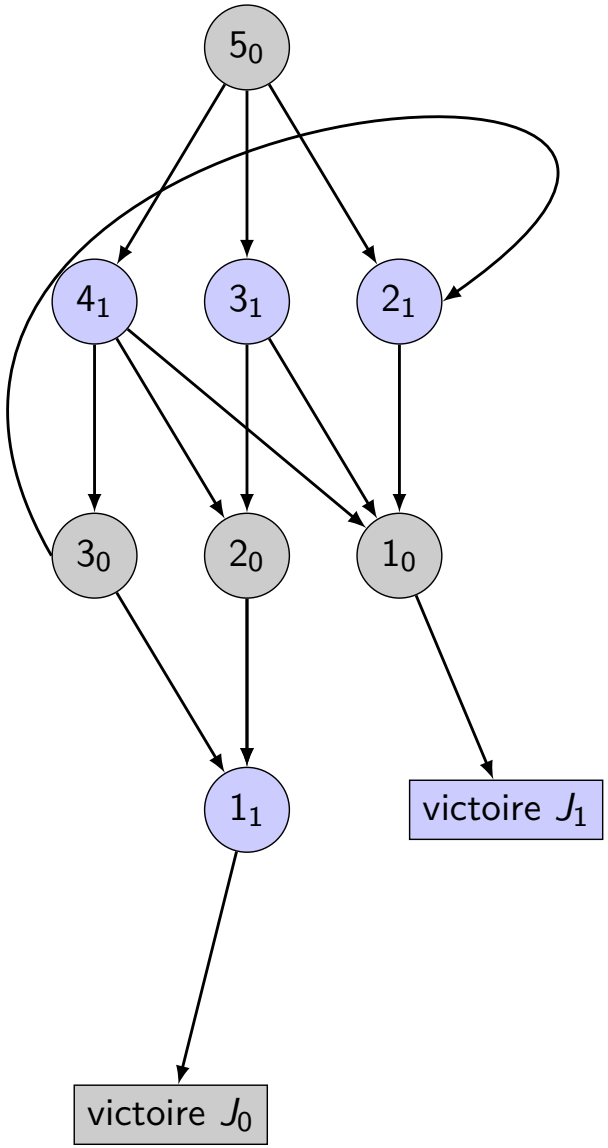


- La stratégie pour le joueur 0 :

$$5_0 \mapsto 3_1,\ 3_0 \mapsto 1_1,\ 2_0 \mapsto 1_1,$$
$$1_0 \mapsto \text{Victoire } J_1,$$

est gagnante à partir de la position 3_0.

- Il n'existe pas de stratégie gagnante pour le joueur 0 depuis la position de départ.



Calcul des attracteurs dans les jeux d'accessibilité

- On note F_0 l'ensemble des sommets terminaux où le joueur 0 gagne, et F_1 l'ensemble des sommets terminaux où le joueur 1 gagne et on suppose qu'il n'y a pas d'autres sommets terminaux (pas de partie nulle).
- Lorsque c'est au joueur J_0 de jouer il est sûr de gagner lorsque :
  - Il est au trait et il peut choisir un coup qui le mène à une position gagnante.
  - L'adversaire (J_1) est au trait et tous ses coups mènent à une position gagnante pour le joueur 0.
- On cherche à calculer une partie des sommets notée $\mathcal{G}$ tels que si la position du jeu fait partie de $\mathcal{G}$, le joueur 0 est sûr de gagner.

Calcul des attracteurs dans les jeux d'accessibilité

- On note F_0 l'ensemble des sommets terminaux où le joueur 0 gagne, et F_1 l'ensemble des sommets terminaux où le joueur 1 gagne et on suppose qu'il n'y a pas d'autres sommets terminaux (pas de partie nulle).
- Lorsque c'est au joueur J_0 de jouer il est sûr de gagner lorsque :
  - Il est au trait et il peut choisir un coup qui le mène à une position gagnante.
  - L'adversaire (J_1) est au trait et tous ses coups mènent à une position gagnante pour le joueur 0.
- On cherche à calculer une partie des sommets notée $\mathcal{G}$ tels que si la position du jeu fait partie de $\mathcal{G}$, le joueur 0 est sûr de gagner.

Calcul des attracteurs dans les jeux d'accessibilité

- On note F_0 l'ensemble des sommets terminaux où le joueur 0 gagne, et F_1 l'ensemble des sommets terminaux où le joueur 1 gagne et on suppose qu'il n'y a pas d'autres sommets terminaux (pas de partie nulle).
- Lorsque c'est au joueur J_0 de jouer il est sûr de gagner lorsque :
  - Il est au trait et il peut choisir un coup qui le mène à une position gagnante.
  - L'adversaire (J_1) est au trait et tous ses coups mènent à une position gagnante pour le joueur 0.
- On cherche à calculer une partie des sommets notée $\mathcal{G}$ tels que si la position du jeu fait partie de $\mathcal{G}$, le joueur 0 est sûr de gagner.

- Supposons que l'on dispose d'une partie des sommets notée $\mathcal{G}_n$ telle que tous les sommets de $\mathcal{G}_n$ soit gagnant pour le joueur 0 en moins de n coups (ie le joueur 0 joue au plus n fois).
- On peut alors définir les ensembles suivants :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 0 qui permettent au joueur 0 d'amener le jeu en une position gagnante (ie de $\mathcal{G}_n$).

- Autrement dit qui permettent de forcer le joueur 1 à recevoir une position perdante pour lui.
- On a aussi :

$$\left\{ s \in S_1 \middle| \forall (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 1 où il ne peut qu'amener le jeu en une position de $\mathcal{G}_n$.

- Supposons que l'on dispose d'une partie des sommets notée $\mathcal{G}_n$ telle que tous les sommets de $\mathcal{G}_n$ soit gagnant pour le joueur 0 en moins de n coups (ie le joueur 0 joue au plus n fois).
- On peut alors définir les ensembles suivants :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 0 qui permettent au joueur 0 d'amener le jeu en une position gagnante (ie de $\mathcal{G}_n$).

- Autrement dit qui permettent de forcer le joueur 1 à recevoir une position perdante pour lui.
- On a aussi :

$$\left\{ s \in S_1 \middle| \forall (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 1 où il ne peut qu'amener le jeu en une position de $\mathcal{G}_n$.

- Supposons que l'on dispose d'une partie des sommets notée $\mathcal{G}_n$ telle que tous les sommets de $\mathcal{G}_n$ soit gagnant pour le joueur 0 en moins de n coups (ie le joueur 0 joue au plus n fois).
- On peut alors définir les ensembles suivants :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 0 qui permettent au joueur 0 d'amener le jeu en une position gagnante (ie de $\mathcal{G}_n$).

- Autrement dit qui permettent de forcer le joueur 1 à recevoir une position perdante pour lui.
- On a aussi :

$$\left\{ s \in S_1 \middle| \forall (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$

c'est l'ensemble des positions (sommets) contrôlés par le joueur 1 où il ne peut qu'amener le jeu en une position de $\mathcal{G}_n$.

- Supposons que l'on dispose d'une partie des sommets notée $\mathcal{G}_n$ telle que tous les sommets de $\mathcal{G}_n$ soit gagnant pour le joueur 0 en moins de n coups (ie le joueur 0 joue au plus n fois).
- On peut alors définir les ensembles suivants :
$$\left\{ s \in S_0 \middle| \exists (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$
c'est l'ensemble des positions (sommets) contrôlés par le joueur 0 qui permettent au joueur 0 d'amener le jeu en une position gagnante (ie de $\mathcal{G}_n$).
- Autrement dit qui permettent de forcer le joueur 1 à recevoir une position perdante pour lui.
- On a aussi :
$$\left\{ s \in S_1 \middle| \forall (s, s') \in A,\ s' \in \mathcal{G}_n \right\}$$
c'est l'ensemble des positions (sommets) contrôlés par le joueur 1 où il ne peut qu'amener le jeu en une position de $\mathcal{G}_n$.

- On a construit deux ensembles :

$$\left\{s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n\right\} \qquad \left\{s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n\right\}$$

- On voit que cela permet de construire $\mathcal{G}_{n+1}$ en ajoutant à $\mathcal{G}_n$ les deux ensembles précédents.
- Puis il faudra prendre la réunion de tous les $\mathcal{G}_n$ (ensemble des états gagnants en n coups) pour obtenir $\mathcal{G}$ (ensemble des états gagnants en un nombre quelconque de coups).
- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n\right\}$$

$$\cup \left\{s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n\right\}$$

- On a construit deux ensembles :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\} \qquad \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- On voit que cela permet de construire $\mathcal{G}_{n+1}$ en ajoutant à $\mathcal{G}_n$ les deux ensembles précédents.
- Puis il faudra prendre la réunion de tous les $\mathcal{G}_n$ (ensemble des états gagnants en n coups) pour obtenir $\mathcal{G}$ (ensemble des états gagnants en un nombre quelconque de coups).
- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

$$\cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- On a construit deux ensembles :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\} \qquad \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- On voit que cela permet de construire $\mathcal{G}_{n+1}$ en ajoutant à $\mathcal{G}_n$ les deux ensembles précédents.
- Puis il faudra prendre la réunion de tous les $\mathcal{G}_n$ (ensemble des états gagnants en n coups) pour obtenir $\mathcal{G}$ (ensemble des états gagnants en un nombre quelconque de coups).
- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

$$\cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- On a construit deux ensembles :

$$\left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\} \qquad \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- On voit que cela permet de construire $\mathcal{G}_{n+1}$ en ajoutant à $\mathcal{G}_n$ les deux ensembles précédents.
- Puis il faudra prendre la réunion de tous les $\mathcal{G}_n$ (ensemble des états gagnants en n coups) pour obtenir $\mathcal{G}$ (ensemble des états gagnants en un nombre quelconque de coups).
- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\} \cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

Calcul des attracteurs

- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\}$$
$$\cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- La suite $\mathcal{G}_n$ est croissante pour l'inclusion, or il y a un nombre fini de sommets. Donc il existe n_0 tel que :

$$\forall n \geqslant n_0, \ \mathcal{G}_n = \mathcal{G}_{n_0}$$

- On pose alors $\mathcal{G} = \mathcal{G}_{n_0}$, c'est l'ensemble des positions gagnantes pour le joueur 0, On parle d'attracteurs. Précisément c'est les attracteurs de F_0 pour le joueur J_0.

Calcul des attracteurs

- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\}$$
$$\cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- La suite $\mathcal{G}_n$ est croissante pour l'inclusion, or il y a un nombre fini de sommets. Donc il existe n_0 tel que :

$$\forall n \geqslant n_0,\ \mathcal{G}_n = \mathcal{G}_{n_0}$$

- On pose alors $\mathcal{G} = \mathcal{G}_{n_0}$, c'est l'ensemble des positions gagnantes pour le joueur 0, On parle d'attracteurs. Précisément c'est les attracteurs de F_0 pour le joueur J_0.

Calcul des attracteurs

- On pose donc :

$$\mathcal{G}_0 = F_0 \text{ ensembles des sommets terminaux victoire de} J_0$$

$$\mathcal{G}_{n+1} = \mathcal{G}_n \cup \left\{ s \in S_0 \middle| \exists (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

$$\cup \left\{ s \in S_1 \middle| \forall (s, s') \in A, s' \in \mathcal{G}_n \right\}$$

- La suite $\mathcal{G}_n$ est croissante pour l'inclusion, or il y a un nombre fini de sommets. Donc il existe n_0 tel que :

$$\forall n \geqslant n_0, \ \mathcal{G}_n = \mathcal{G}_{n_0}$$

- On pose alors $\mathcal{G} = \mathcal{G}_{n_0}$, c'est l'ensemble des positions gagnantes pour le joueur 0, On parle d'attracteurs. Précisément c'est les attracteurs de F_0 pour le joueur J_0.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

Algorithme

- On pose $\mathcal{G} = F_0$.
- Tant qu'il y a du changement dans $\mathcal{G}$.
  - on balaie les sommets $s \in S$.
  - si s est dans S_0, on regarde si il existe un voisin s' de s tel que $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
  - si s est dans S_1, on regarde si pour tout voisin s' de s, on a $s' \in \mathcal{G}$. Dans ce cas s est ajouté à $\mathcal{G}$.
- Une fois que l'on a $\mathcal{G}$ on a la stratégie gagnante.

```
def ilExiste(s, G):
    """
    algorithme classique de recherche
    """
    for sprime in sommets:
        if [s,sprime] in arretes and sprime in G:
            return True
    return False
def quelqueSoit(s, G):
    """
    algorithme classique de recherche
    une difficulté: il faut vérifier que le sommet
    n'est pas terminal
    """
    estTerminal = True
    for sprime in sommets:
        if [s,sprime] in arretes:
            estTerminal = False
            if sprime not in G:
                return False
    if estTerminal :
        return False
    else:
        return True
```

```
G = ["V_0"]
chgt = True
while chgt:
  chgt = False
  for s in sommets:
    if s not in G:
      controle = s[-1]
      if (  (controle == "0" and ilExiste(s, G))
         or (controle == "1" and quelqueSoit(s, G)) ):
          G.append(s)
          chgt = True
```

```
def strategie(s, G):
    """
    algorithme classique de recherche
    on renvoie le bon voisin
    """
    if s== "V_0":
        return "on a gagné!"
    for sprime in sommets:
        if [s,sprime] in arretes and sprime in G:
            return sprime
    return "jouer un coup au hasard"
```

1 Vocabulaire des graphes et calcul des attracteurs

2 Algorithme de Min-Max

- On considère donc une position donnée dans un jeu et l'on cherche toujours à répondre à la question suivante : quel est le meilleur coup à jouer ? ie, on cherche des stratégies sans mémoire.
- Le calcul des attracteurs de la section précédente n'est pas toujours possible, il est souvent trop long.
- On se déplace plutôt dans le cadre où l'on va regarder l'ensemble des parties comme un arbre et chercher la meilleure branche en calculant un nombre donné de coups à l'avance.

- On considère donc une position donnée dans un jeu et l'on cherche toujours à répondre à la question suivante : quel est le meilleur coup à jouer ? ie, on cherche des stratégies sans mémoire.
- Le calcul des attracteurs de la section précédente n'est pas toujours possible, il est souvent trop long.
- On se déplace plutôt dans le cadre où l'on va regarder l'ensemble des parties comme un arbre et chercher la meilleure branche en calculant un nombre donné de coups à l'avance.

- On considère donc une position donnée dans un jeu et l'on cherche toujours à répondre à la question suivante : quel est le meilleur coup à jouer ? ie, on cherche des stratégies sans mémoire.
- Le calcul des attracteurs de la section précédente n'est pas toujours possible, il est souvent trop long.
- On se déplace plutôt dans le cadre où l'on va regarder l'ensemble des parties comme un arbre et chercher la meilleure branche en calculant un nombre donné de coups à l'avance.

Retour sur le vocabulaire des arbres

- Un arbre est un ensemble $\mathcal{A}$ non vide dont les éléments sont appelés noeuds et sur lequel est défini une relation $\mathcal{P}$ avec : $x\mathcal{P}y$ signifie x est père de y tel que :
  - il existe un noeud r et un seul appelé racine n'ayant pas de père.
  - tous les autres éléments admettent un père et un seul.
- La propriété fondamentale des arbres est : pour tout $x \in \mathcal{A} \setminus \{r\}$, il existe une suite de noeud $(x_0, \ldots, x_p)$ qui relie r à x (ie $x_0 = r$ et $x_p = x$) avec $x_i \mathcal{P} x_{i+1}$.
- on a le vocabulaire suivant :
  - les noeuds sans descendants sont les feuilles de l'arbre,
  - le degré d'un noeud est le nombre de descendants,
  - la hauteur d'un arbre est la longueur maximal des chemins qui relient un noeud à la racine.
  - Un arbre est dit étiqueté lorsqu'à chaque noeud on associe une information appelée étiquette.

Retour sur le vocabulaire des arbres

- Un arbre est un ensemble $\mathcal{A}$ non vide dont les éléments sont appelés noeuds et sur lequel est défini une relation $\mathcal{P}$ avec : $x\mathcal{P}y$ signifie x est père de y tel que :
  - il existe un noeud r et un seul appelé racine n'ayant pas de père.
  - tous les autres éléments admettent un père et un seul.
- La propriété fondamentale des arbres est : pour tout $x \in \mathcal{A} \setminus \{r\}$, il existe une suite de noeud $(x_0, \ldots, x_p)$ qui relie r à x (ie $x_0 = r$ et $x_p = x$) avec $x_i \mathcal{P} x_{i+1}$.
- on a le vocabulaire suivant :
  - les noeuds sans descendants sont les feuilles de l'arbre,
  - le degré d'un noeud est le nombre de descendants,
  - la hauteur d'un arbre est la longueur maximal des chemins qui relient un noeud à la racine.
  - Un arbre est dit étiqueté lorsqu'à chaque noeud on associe une information appelée étiquette.

Retour sur le vocabulaire des arbres

- Un arbre est un ensemble $\mathcal{A}$ non vide dont les éléments sont appelés noeuds et sur lequel est défini une relation $\mathcal{P}$ avec : $x\mathcal{P}y$ signifie x est père de y tel que :
  - il existe un noeud r et un seul appelé racine n'ayant pas de père.
  - tous les autres éléments admettent un père et un seul.
- La propriété fondamentale des arbres est : pour tout $x \in \mathcal{A} \setminus \{r\}$, il existe une suite de noeud $(x_0, \ldots, x_p)$ qui relie r à x (ie $x_0 = r$ et $x_p = x$) avec $x_i \mathcal{P} x_{i+1}$.
- on a le vocabulaire suivant :
  - les noeuds sans descendants sont les feuilles de l'arbre,
  - le degré d'un noeud est le nombre de descendants,
  - la hauteur d'un arbre est la longueur maximal des chemins qui relient un noeud à la racine.
  - Un arbre est dit étiqueté lorsqu'à chaque noeud on associe une information appelée étiquette.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

Dans le cas qui nous intéresse :

- la situation de départ est donc la racine de notre arbre.
- chaque branche correspond à un coup possible qui aboutit donc à d'autres noeuds étiquetés par une autre position.
- Les feuilles de l'arbre sont les fins de parties possibles (avec donc victoire d'un joueur ou de l'autre).

- On considère que l'on dispose d'une fonction U qui évalue une position du point de vue du joueur 0.
- Cette fonction est positive si la position est favorable au joueur 0, négative sinon.
- On parle de fonction d'évaluation ou fonction d'utilité.

- Par exemple une fonction qui renvoie 1 si la position est gagnante, -1 si elle est perdante 0 sinon.

```python
def utilite(listeJeu):
    """
    entrée: listeJeu = liste de 7 de str "X" ou "O"
            contenu de chaque  colonne
    sortie 0 si pas de victoire
           1 (resp -1) si victoire "X" (resp "O")
    évalue le jeu pour X
    """
    jeu = list2array(listeJeu)

    assert jeu.shape == (6,7)
    horizontal = [ ([i,j], [i+1,j], [i+2,j], [i+3,j])
                        for i in range(3) for j in range(7) ]
    vertical = [( [i,j], [i,j+1], [i,j+2], [i,j+3])
                        for i in range(6) for j in range(4) ]
    diagonaleNE = [ ([i,j], [i+1, j-1], [i+2, j-2], [i+3, j-3])
                        for i in range(3) for j in range(3,7)]
    diagonaleSE = [ ( [i,j], [i-1, j-1], [i-2, j-2], [i-3, j-3])
                        for i in range(3, 6)  for j in range(3,7) ]
    for extrait in horizontal + vertical +diagonaleNE + diagonaleSE:
        liste4 = [jeu[i,j] for i,j in extrait]
        assert len(liste4) == 4
        if liste4.count("X") == 4:
            #print(extrait, liste4)
            return 1
        elif liste4.count("O") == 4:
            #print(extrait, liste4)
            return -1
    return 0
```

- Pour le joueur 0, une stratégie simple consiste à regarder tous les coups possibles et à choisir celui qui maximise cette fonction d'utilité.
- Étant donné une position s, le meilleur coup t à choisir est celui qui réalise :

$$\max_{t \in succ(s)} U(t)$$

(il n'y bien sûr a pas unicité d'un tel coup). dans cette notation $succ(s)$ désigne les successeurs de s cad les noeuds reliés à s.

- Pour le joueur 0, une stratégie simple consiste à regarder tous les coups possibles et à choisir celui qui maximise cette fonction d'utilité.
- Étant donné une position s, le meilleur coup t à choisir est celui qui réalise :

$$\max_{t \in succ(s)} U(t)$$

(il n'y bien sûr a pas unicité d'un tel coup). dans cette notation $succ(s)$ désigne les successeurs de s cad les noeuds reliés à s.

```
def minmax1(listeJeu):
    """
    entrée: listeJeu = liste de 7 de str "X" ou "O"
        contenu de chaque colonne
    sortie: meilleur coup en choisissant 1 coup à l'avance
    """
    listeCoupEval = {} # dictionnaire de la forme
                       coupjoué / évaluation.
    for i in range(7):
        if len(listeJeu[i]) <6 :
            copieJeu = listeJeu.copy()
            copieJeu[i] += 'X'
            listeCoupEval[i] = utilite(copieJeu)

    # On cherche alors le max du jeu
    valmax = -2
    indmax = 0
    for i in listeCoupEval:
        if listeCoupEval[i] > valmax:
            valmax = listeCoupEval[i]
            indmax = i

    return indmax
```

```python
def minmax1(listeJeu):
    """
    entrée: listeJeu =
      liste de 7 de str "X" ou "O"
      contenu de chaque  colonne
    sortie: meilleur coup en
       choisissant 1 coup à l'avance
    """
    listeCoupEval = {} # dictionnaire
  #de la forme coupjoué / évaluation.

    for i in range(7):
        if len(listeJeu[i]) <6 :
            copieJeu = listeJeu.copy()
            copieJeu[i] += 'X'
            listeCoupEval[i] = utilite(copieJeu)

    # On cherche alors le max du jeu
    valmax = -2
    indmax = 0
    for i in listeCoupEval:
        if listeCoupEval[i] > valmax:
            valmax = listeCoupEval[i]
            indmax = i

    return indmax
```

```
------ situation de jeu ------
 0 1 2 3 4 5 6
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |X|X|  |  |
|  |  |  |O|O|X|O|
|  |  |  |X|X|O|X|
 0 1 2 3 4 5 6

------Test minmax niveau 1 ------
------ minmax niveau 1 ------
si X joue en  0 eval =  0
si X joue en  1 eval =  0
si X joue en  2 eval =  0
si X joue en  3 eval =  1
si X joue en  4 eval =  0
si X joue en  5 eval =  0
si X joue en  6 eval =  0
meilleur coup à jouer (1 coup à
l'avance): 3
```

- En calculant uniquement un coup à l'avance, on choisit t tel que :

$$\max_{t \in succ(s)} U(t)$$

- Si on va un peu plus loin, une stratégie un peu plus avancé consiste à choisir le coup qui réalise la plus grand utilité quelque soit la réponse de l'adversaire.
- Il faut alors trouver un coup t qui réalise :

$$\max_{t \in succ(s)} \begin{cases} \min_{u \in succ(t)} U(t) & \text{si } succ(t) \neq \emptyset \\ U(t) & \text{si } succ(t) = \emptyset \end{cases}$$

- En prévoyant 3 coups en avance, cela donnerait (en négligeant les cas terminaux) :

$$\max_{t \in succ(s)} \min_{u \in succ(t)} \max_{v \in succ(u)} U(v)$$

- Cette construction se généralise et porte le nom d'algorithme de min-max.

- En calculant uniquement un coup à l'avance, on choisit t tel que :

$$\max_{t \in succ(s)} U(t)$$

- Si on va un peu plus loin, une stratégie un peu plus avancé consiste à choisir le coup qui réalise la plus grand utilité quelque soit la réponse de l'adversaire.
- Il faut alors trouver un coup t qui réalise :

$$\max_{t \in succ(s)} \begin{cases} \min_{u \in succ(t)} U(t) & \text{si } succ(t) \neq \emptyset \\ U(t) & \text{si } succ(t) = \emptyset \end{cases}$$

- En prévoyant 3 coups en avance, cela donnerait (en négligeant les cas terminaux) :

$$\max_{t \in succ(s)} \min_{u \in succ(t)} \max_{v \in succ(u)} U(v)$$

- Cette construction se généralise et porte le nom d'algorithme de min-max.

- En calculant uniquement un coup à l'avance, on choisit t tel que :

$$\max_{t \in succ(s)} U(t)$$

- Si on va un peu plus loin, une stratégie un peu plus avancé consiste à choisir le coup qui réalise la plus grand utilité quelque soit la réponse de l'adversaire.
- Il faut alors trouver un coup t qui réalise :

$$\max_{t \in succ(s)} \begin{cases} \min_{u \in succ(t)} U(t) & \text{si } succ(t) \neq \emptyset \\ U(t) & \text{si } succ(t) = \emptyset \end{cases}$$

- En prévoyant 3 coups en avance, cela donnerait (en négligeant les cas terminaux) :

$$\max_{t \in succ(s)} \min_{u \in succ(t)} \max_{v \in succ(u)} U(v)$$

- Cette construction se généralise et porte le nom d'algorithme de min-max.

- En calculant uniquement un coup à l'avance, on choisit t tel que :

$$\max_{t \in succ(s)} U(t)$$

- Si on va un peu plus loin, une stratégie un peu plus avancé consiste à choisir le coup qui réalise la plus grand utilité quelque soit la réponse de l'adversaire.
- Il faut alors trouver un coup t qui réalise :

$$\max_{t \in succ(s)} \begin{cases} \min_{u \in succ(t)} U(t) & \text{si } succ(t) \neq \emptyset \\ U(t) & \text{si } succ(t) = \emptyset \end{cases}$$

- En prévoyant 3 coups en avance, cela donnerait (en négligeant les cas terminaux) :

$$\max_{t \in succ(s)} \min_{u \in succ(t)} \max_{v \in succ(u)} U(v)$$

- Cette construction se généralise et porte le nom d'algorithme de min-max.

- En calculant uniquement un coup à l'avance, on choisit t tel que :

$$\max_{t \in succ(s)} U(t)$$

- Si on va un peu plus loin, une stratégie un peu plus avancé consiste à choisir le coup qui réalise la plus grand utilité quelque soit la réponse de l'adversaire.
- Il faut alors trouver un coup t qui réalise :

$$\max_{t \in succ(s)} \begin{cases} \min_{u \in succ(t)} U(t) & \text{si } succ(t) \neq \emptyset \\ U(t) & \text{si } succ(t) = \emptyset \end{cases}$$

- En prévoyant 3 coups en avance, cela donnerait (en négligeant les cas terminaux) :

$$\max_{t \in succ(s)} \min_{u \in succ(t)} \max_{v \in succ(u)} U(v)$$

- Cette construction se généralise et porte le nom d'algorithme de min-max.

```
def minmax2(listeJeu):
    """
    entrée: listeJeu = liste de 7 de str "X" ou "O"
        contenu de chaque  colonne
    sortie: meilleur coup en choisissant 2 coup à l'avance
    """

    listeCoupEval = {} # dictionnaire de la forme [i,j] / évaluation.
    for i in range(7):
        if len(listeJeu[i]) <6 :
            copieJeu = listeJeu.copy()
            copieJeu[i] += 'X'
            for j in range(7):
                if len(copieJeu[j]) <6 :
                    copieJeu2 = copieJeu.copy()
                    copieJeu2[j] += 'O'
                    listeCoupEval[(i,j)] = utilite(copieJeu2)

    listMin = {} # donne pour chaque i jouable le min listeCoupEval[i,j]
    for i,j in  listeCoupEval:
        if i not in listMin or listeCoupEval[i,j] < listMin[i] :
            listMin[i] = listeCoupEval[i,j]
*
    # On cherche alors le max de listMin
    valmax = -2
    indmax = 0
    for i in listMin:
        if listMin[i] > valmax:
            valmax = listMin[i]
            indmax = i

    return indmax
```

```
  ——— situation de jeu ———
 0 1 2 3 4 5 6
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |O|O|  |  |
|  |  |  |X|X|O|X|
|  |  |  |O|O|X|O|
 0 1 2 3 4 5 6
——— minmax niveau 2 ———
si X joue en  0 puis que O joue en  0 eval =  0
si X joue en  0 puis que O joue en  1 eval =  0
...
si X joue en  0 puis que O joue en  3 eval =  -1
si X joue en  1 puis que O joue en  3 eval =  -1
si X joue en  1 puis que O joue en  4 eval =  0
évaliuation en 2 coup si X joue en  0 min de l'éval =  -1
évaliuation en 2 coup si X joue en  1 min de l'éval =  -1
évaliuation en 2 coup si X joue en  2 min de l'éval =  -1
évaliuation en 2 coup si X joue en  3 min de l'éval =  0
évaliuation en 2 coup si X joue en  4 min de l'éval =  -1
évaliuation en 2 coup si X joue en  5 min de l'éval =  -1
évaliuation en 2 coup si X joue en  6 min de l'éval =  -1
meilleur coup à jouer (2 coups à l'avance): 3
```

- Soit U une fonction d'utilité pour le joueur 0, c'est-à-dire une fonction qui évalue les sommets (ie les positions) du point de vue du joueur 0.
- On note S_0 l'ensemble des sommets où c'est le joueur 0 qui a le trait, et S_1 l'ensemble des sommets où c'est le joueur 1 qui a le trait.
- On définit par récurrence pour $n \geqslant 0$ les utilités maximales à n coups noté $(U^r)_{r \in [\![0,n]\!]}$ par :

$$\forall s \in S,\ U^0(s) = U(s)$$

$$\forall n \geqslant 1,\ \forall s \in S,\ U^n(s) = \begin{cases} U^{n-1}(s) & \text{si } succ(s) = \emptyset \\ \max_{t \in succ(s)} U^{n-1}(t) & \text{si } s \in S_0 \\ \min_{t \in succ(s)} U^{n-1}(t) & \text{si } s \in S_1 \end{cases}$$

- Une stratégie pour le joueur 0 qui réalise l'utilité maximale en p coups consiste à choisir pour tout sommet $s \in S_0$ non terminal :

$$f(s) \in \operatorname{argmax} U^p(t)$$

autrement dit qui consiste à choisir pour le joueur 0 une valeur qui réalise le maximum de l'utilité.

- Soit U une fonction d'utilité pour le joueur 0, c'est-à-dire une fonction qui évalue les sommets (ie les positions) du point de vue du joueur 0.
- On note S_0 l'ensemble des sommets où c'est le joueur 0 qui a le trait, et S_1 l'ensemble des sommets où c'est le joueur 1 qui a le trait.
- On définit par récurrence pour $n \geqslant 0$ les utilités maximales à n coups noté $(U^r)_{r \in [\![0,n]\!]}$ par :

$$\forall s \in S,\ U^0(s) = U(s)$$

$$\forall n \geqslant 1,\ \forall s \in S,\ U^n(s) = \begin{cases} U^{n-1}(s) & \text{si } succ(s) = \emptyset \\ \max_{t \in succ(s)} U^{n-1}(t) & \text{si } s \in S_0 \\ \min_{t \in succ(s)} U^{n-1}(t) & \text{si } s \in S_1 \end{cases}$$

- Une stratégie pour le joueur 0 qui réalise l'utilité maximale en p coups consiste à choisir pour tout sommet $s \in S_0$ non terminal :

$$f(s) \in \operatorname{argmax} U^p(t)$$

autrement dit qui consiste à choisir pour le joueur 0 une valeur qui réalise le maximum de l'utilité.

- Soit U une fonction d'utilité pour le joueur 0, c'est-à-dire une fonction qui évalue les sommets (ie les positions) du point de vue du joueur 0.
- On note S_0 l'ensemble des sommets où c'est le joueur 0 qui a le trait, et S_1 l'ensemble des sommets où c'est le joueur 1 qui a le trait.
- On définit par récurrence pour $n \geqslant 0$ les utilités maximales à n coups noté $(U^r)_{r\in[\![0,n]\!]}$ par :

$$\forall s \in S,\ U^0(s) = U(s)$$

$$\forall n \geqslant 1,\ \forall s \in S,\ U^n(s) = \begin{cases} U^{n-1}(s) & \text{si } succ(s) = \emptyset \\ \max_{t\in succ(s)} U^{n-1}(t) & \text{si } s \in S_0 \\ \min_{t\in succ(s)} U^{n-1}(t) & \text{si } s \in S_1 \end{cases}$$

- Une stratégie pour le joueur 0 qui réalise l'utilité maximale en p coups consiste à choisir pour tout sommet $s \in S_0$ non terminal :

$$f(s) \in \operatorname{argmax} U^p(t)$$

autrement dit qui consiste à choisir pour le joueur 0 une valeur qui réalise le maximum de l'utilité.

- Soit U une fonction d'utilité pour le joueur 0, c'est-à-dire une fonction qui évalue les sommets (ie les positions) du point de vue du joueur 0.
- On note S_0 l'ensemble des sommets où c'est le joueur 0 qui a le trait, et S_1 l'ensemble des sommets où c'est le joueur 1 qui a le trait.
- On définit par récurrence pour $n \geqslant 0$ les utilités maximales à n coups noté $(U^r)_{r\in[\![0,n]\!]}$ par :

$$\forall s \in S,\ U^0(s) = U(s)$$

$$\forall n \geqslant 1,\ \forall s \in S,\ U^n(s) = \begin{cases} U^{n-1}(s) & \text{si } succ(s) = \emptyset \\ \max_{t\in succ(s)} U^{n-1}(t) & \text{si } s \in S_0 \\ \min_{t\in succ(s)} U^{n-1}(t) & \text{si } s \in S_1 \end{cases}$$

- Une stratégie pour le joueur 0 qui réalise l'utilité maximale en p coups consiste à choisir pour tout sommet $s \in S_0$ non terminal :

$$f(s) \in \text{argmax} U^p(t)$$

autrement dit qui consiste à choisir pour le joueur 0 une valeur qui réalise le maximum de l'utilité.

```python
def utiliteMaximale(listeJeu, n, tour):
    """
    entrée: listeJeu = liste de 7 de str "X" ou "O" contenu de chaque  colonne
        n = int = niveau       niveau 0 = fct utilite
        niveau 1 = max de fct utilite       niveau 2 = minmax de fct utilite, etc.
      tour = "X" ou "O": joueur qui doit jouer
    sortie: évaluation de listeJeu à n niveau
    """
    if n == 0 :
      return utilite(listeJeu)

    symInv = "X" if tour == "O" else "O"

    evalSucc = {}
    for i in range(7):
      if len(listeJeu[i]) <6 :
        copieJeu = listeJeu.copy()
        copieJeu[i] += tour
        evalSucc[i] = utiliteMaximale(copieJeu, n-1, symInv)

    if len(evalSucc) == 0 :
      # pas de coup possible
      return utiliteMaximale(listeJeu, n-1, symInv)

    valmax = -2
    valmin = 2
    for i in evalSucc :
      if evalSucc[i] > valmax:
        valmax = evalSucc[i]
      if evalSucc[i] < valmin:
        valmin = evalSucc[i]

    if tour == "X":
      return valmax
    else :
      return valmin
```

```
def strategie(listeJeu, n):
    """
    entrée: listeJeu = liste de 7 de str "X" ou "O"
      contenu de chaque  colonne
        n = int = niveau
        niveau 0 = fct utilite
        niveau 1 = max de fct utilite
        niveau 2 = minmax de fct utilite, etc.
    sortie: meilleur coup en choisissant n coups à l'avance
    """
    listeCoupEval = {} # dictionnaire de la forme
                       # coupjoué / évaluation.
    for i in range(7):
        if len(listeJeu[i]) <6 :
            copieJeu = listeJeu.copy()
            copieJeu[i] += "X"
            listeCoupEval[i] = utiliteMaximale(copieJeu, n, "O")


    # On cherche alors le max
    valmax = -1
    indmax = 0
    for i in listeCoupEval:
        if listeCoupEval[i] > valmax:
            valmax = listeCoupEval[i]
            indmax = i
    print("évaluation finale de la position", valmax)
    return indmax
```

```
------ situation de jeu ------
 0 1 2 3 4 5 6
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |O|O|  |  |
 0 1 2 3 4 5 6

------ minmax niveau  2 ------
si X joue en  0 éval après  2 coups:  0
si X joue en  1 éval après  2 coups:  0
si X joue en  2 éval après  2 coups:  0
si X joue en  3 éval après  2 coups:  0
si X joue en  4 éval après  2 coups:  0
si X joue en  5 éval après  2 coups:  0
si X joue en  6 éval après  2 coups:  0
évaluation finale de la position 0
meilleur coup à jouer (3 coups à l'avance): 0
-------
------ minmax niveau  3 ------
si X joue en  0 éval après  3 coups:  -1
si X joue en  1 éval après  3 coups:  -1
si X joue en  2 éval après  3 coups:  0
si X joue en  3 éval après  3 coups:  -1
si X joue en  4 éval après  3 coups:  -1
si X joue en  5 éval après  3 coups:  0
si X joue en  6 éval après  3 coups:  -1
évaluation finale de la position 0
meilleur coup à jouer (4 coups à l'avance): 2
-------
```

- L'algorithme de min max est assez simple à mettre en oeuvre, le problème principal est dans la fonction d'utilité.
- On parle méthodes d'heuristiques pour désigner les méthodes qui fournissent des solutions acceptables sans que l'on ne puisse prouver leur optimalité.
- Dans le contexte de l'algorithme de min max, on désigne par heuristique les fonctions d'évaluation que l'on utilise sans pouvoir prouver qu'elles permettent de gagner à coup sûr.

- L'algorithme de min max est assez simple à mettre en oeuvre, le problème principal est dans la fonction d'utilité.
- On parle méthodes d'heuristiques pour désigner les méthodes qui fournissent des solutions acceptables sans que l'on ne puisse prouver leur optimalité.
- Dans le contexte de l'algorithme de min max, on désigne par heuristique les fonctions d'évaluation que l'on utilise sans pouvoir prouver qu'elles permettent de gagner à coup sûr.

- L'algorithme de min max est assez simple à mettre en oeuvre, le problème principal est dans la fonction d'utilité.
- On parle méthodes d'heuristiques pour désigner les méthodes qui fournissent des solutions acceptables sans que l'on ne puisse prouver leur optimalité.
- Dans le contexte de l'algorithme de min max, on désigne par heuristique les fonctions d'évaluation que l'on utilise sans pouvoir prouver qu'elles permettent de gagner à coup sûr.