Chapitre 7

Elements de théorie des jeux

Plan

7.1	**Introduction**	80
7.1.1	Quelques exemples de jeux réels	80
7.1.2	Vocabulaire associé aux jeux du chapitre	81
7.2	**L'étude des jeux partiaux**	82
7.2.1	Représentation par graphes bipartis pour les jeux partiaux	82
7.2.2	Etats et stratégies gagnants pour Adam dans un graphe biparti	83
7.3	**L'étude des jeux partiaux sans cycles**	84
7.3.1	Représentation arborescente	85
7.3.2	Détermination des stratégies et états gagnants	86
7.4	**Les jeux impartiaux**	88
7.4.1	Le jeu de Chomp	88
7.4.2	Arène d'un jeu impartial	89
7.4.3	Détermination de stratégies et états gagnants à partir de l'arène du jeu	90

L'objectif de ce chapitre est d'étudier des jeux. Nous nous intéresserons principalement aux jeux à deux joueurs, même si la plupart des résultats peuvent être étendus. Toute personne qui a déjà joué à un jeu sait qu'il existe des stratégies de jeu qui sont meilleures que d'autres et que, dans certaines configuration de jeu, il est impossible pour l'un des joueurs de l'emporter. Ce sont ces notions que nous allons formaliser dans les paragraphes et sections suivants.

7.1 Introduction

Dans ce chapitre nous allons nous intéresser à l'étude théorique et algorithmique de jeux à deux joueurs opposés jouant au tour par tour. Chacun de ces joueurs sont appelés traditionnellement Adam et Ève.

Les jeux auxquels nous allons nous intéresser sont des jeux :

— **à espace d'états finis** : il existe un nombre fini (potentiellement gigantesque) de configurations possible,
— **sans hasard** : aucun lancer de dé ou autre expérience aléatoire n'a d'impact sur le déroulement d'une partie de jeu,
— **au tour par tout** : les joueurs jouent alternativement et la durée de réflexion n'a pas d'influence sur le jeu,
— **déterministes** : l'effet d'une action par un joueur dans une configuration donnée conduit toujours à la même configuration.

7.1.1 Quelques exemples de jeux réels

Nous allons décrire ici trois jeux qui vont nous servir d'exemples tout au long de ce chapitre.

Le jeu de reversi.

Reversi est un jeu de société pour deux joueurs. Le but du jeu est de finir avec le plus de pions de sa couleur sur le plateau. Le plateau de jeu est un damier 8x8. Les joueurs ont chacun des pions de deux couleurs différentes, généralement noirs et blancs.

Au début de la partie, il y a deux pions noirs et deux pions blancs disposés en diagonale au centre du plateau, formant un carré. Les joueurs jouent à tour de rôle, en posant un pion de leur couleur sur une case vide. Lorsqu'un joueur place un pion, il doit "retourner" les pions de l'adversaire qui se trouvent sur une ligne droite (horizontale, verticale ou diagonale) entre le nouveau pion et un pion de sa propre couleur déjà présent sur le plateau. Les pions retournés prennent alors la couleur du joueur qui les a retournés.

Le jeu se termine lorsqu'il n'y a plus de case vide sur le plateau ou lorsqu'aucun des joueurs ne peut plus poser de pion. Le joueur qui a le plus de pions de sa couleur sur le plateau à la fin de la partie est déclaré vainqueur.

Il y a des règles spécifiques pour les situations de fin de parties, comme le fait que si un joueur ne peut pas jouer, il doit passer son tour. Il existe des variantes pour les règles de jeu de Reversi, elles peuvent varier en fonction des versions ou des variantes du jeu

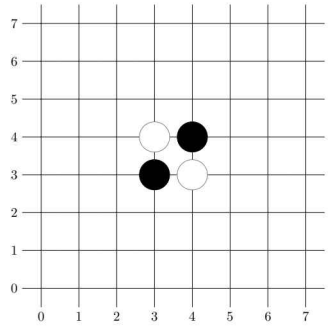

FIGURE 7.1 – Configuration initiale du jeu de Reversi.

Le jeu d'échecs.

Le jeu d'échec est un jeu de stratégie à deux joueurs. Les joueurs déplacent des pièces sur un échiquier (de taille 8 par 8), chacun avec l'objectif de capturer la pièce principale de l'adversaire, appelée le roi. Chaque pièce a des possibilités de mouvements et des fonctions qui lui sont propres. Par exemple, le roi ne peut se déplacer que d'une case à la fois, la reine peut se déplacer en diagonale, en ligne droite ou en diagonale, et les pions qui ne peuvent avancer que vers l'avant. Les joueurs doivent constamment anticiper les mouvements de l'adversaire et trouver des moyens de protéger leur propre roi tout en tentant de capturer celui de l'adversaire.

Le jeu de Chomp.

Le jeu de Chomp est surnommé la *roulette russe des amateurs de chocolat.* Il se déroule de la manière suivante :

1. On part d'une tablette de chocolat entière dont le carré supérieur gauche est supposé empoisonné,
2. A tour de rôle, chaque joueur choisit un carré à manger et consomme tous les carrés situés à gauche et en dessous du carré choisi,
3. Le joueur qui perd est celui qui est obligé de manger le dernier carré !

La figure 7.3 illustre le déroulement d'un coup d'un joueur.

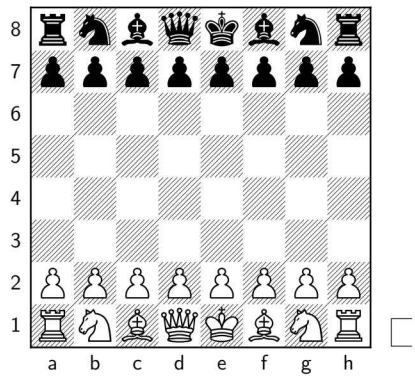

FIGURE 7.2 – Echiquier en début de partie.

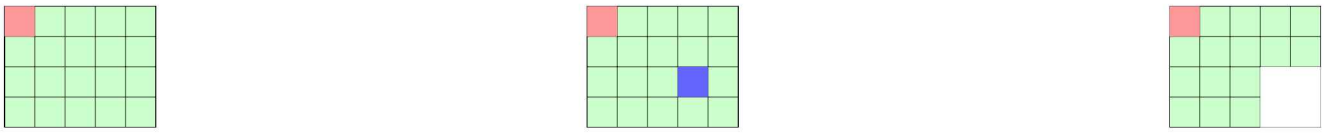

FIGURE 7.3 – Déroulement d'une partie de Chomp. En haut à gauche de la tablette se trouve le carré interdit. A chaque tour, un joueur choisit un carré et consomme ce qui est situé à gauche et en dessous du carré. Comme la tablette a quatre lignes et cinq colonnes, on parlera de jeu de Chomp pour une tablette $(4,5)$.

7.1.2 Vocabulaire associé aux jeux du chapitre

Intuitivement, un jeu est entièrement déterminé par la configuration initiale et séquence des actions jouées par les joueurs. Nous allons formaliser cela mathématiquement.

Nous considérons que deux joueurs, Adam et Eve s'affrontent au tour par tour selon les règles d'un jeu.

Définition 7.1.1 — Définitions. On appelle :
- **plateau** de jeu : l'espace sur lequel les deux joueurs s'affrontent,
- **configuration** de jeu : une disposition du plateau de jeu,
- **état** de jeu : la donnée d'une configuration **et** d'un joueur qui doit jouer, on note S, l'ensemble des état nous supposons que tous les états sont accessibles par une certaine succession de coups de l'un ou l'autre des joueurs,
- **état d'Adam** : un état où Adam doit jouer, on note S_a cet ensemble,
- **état d'Eve** : un état où Eve doit jouer. On note S_e cet ensemble,
- **état gagnant d'Adam** : un état de jeu dans lequel on convient que Adam a gagné la partie et que le jeu s'arrête. On note Ω_a cet ensemble,
- **état gagnant d'Eve** : un état de jeu dans lequel on convient que Adam a gagné la partie et que le jeu s'arrête. On note Ω_e cet ensemble,
- **état de partie nulle** : un état de jeu dans lequel on décide que la partie s'arrête sans vainqueur,
- **état final** : un état de jeu nul ou gagnant pour Adam ou Eve.

Avec ce vocabulaire, on peut parler de *jeu d'accessibilité* : c'est à dire que chaque joueur souhaite atteindre un de ses états gagnants.

Au delà de ce vocabulaire général, on peut introduire d'autres notions. Pour le moment nous avons décrit le plateau de jeu mais nous avons encore à définir les leviers d'action sur le jeu.

Définition 7.1.2 On définit :
- une **action d'Adam** est une transition d'un état d'Adam vers un état d'Eve,
- une **action d'Eve** est une transition d'un état d'Eve vers un état d'Adam,
- l'ensemble des actions d'Adam sera noté $\mathscr{A}_a$ et l'ensemble des actions d'Eve $\mathscr{A}_e$,
- une **stratégie** d'Adam est une application f_A de $\mathscr{S}_a$ dans $\mathscr{A}_a$,
- une **stratégie gagnante d'Adam** est une stratégie f_A qui conduit à un état final gagnant pour Adam, quelle que soit la stratégie de Eve,
- une **stratégie** de Eve est une application f_E de $\mathscr{S}_e$ dans $\mathscr{A}_e$,
- une **stratégie gagnante de Eve** est une stratégie f_E qui conduit à un état final gagnant pour Adam, quelle que soit la stratégie d'Adam,
- une **état gagnant** pour Adam est un état à partir duquel Adam dispose d'une stratégie gagnante,
- une **partie** est une succession d'actions d'Adam et de Eve,

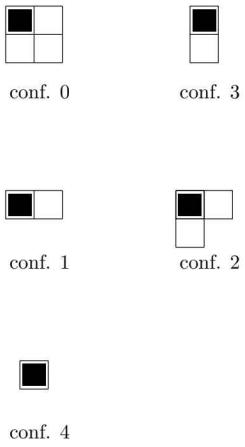

FIGURE 7.4 – Les différentes configuration possible du jeu de chomp 2 par 2.

— on dit que des parties sont **jouées** selon les stratégies f_A et f_E lorsque les deux joueurs appliquent les stratégies en question.

Essayons également d'abstraire quelques caractéristiques importantes des jeux à deux joueurs :
— un jeu est **impartial** si les actions possibles du point de vue d'Adam et de Eve sont les mêmes,
— un jeu est **acyclique** si, à partir d'un état donné, il n'est pas possible de retourner dans cet état au cours d'une partie.

▪ **Exemple 7.1** Illustrons ces notions sur des jeux bien connus :
— le jeu de Nim (les allumettes de Fort Boyard) est un jeu impartial sans cycle,
— le jeu du morpion est un jeu partial sans cycle,
— le jeu de Puissance 4 est un jeu partial sans cycle,
— le jeu du Reversi (Othello) est partial sans cycle,
— le jeu des échecs est partial avec cycle.

7.2 L'étude des jeux partiaux

Nous allons nous intéresser à la théorie générale des jeux partiaux (potentiellement cycliques). Nos deux joueurs, Adam et Eve s'affrontent à un certain jeu

7.2.1 Représentation par graphes bipartis pour les jeux partiaux

Pour modéliser les jeux énoncés dans le chapeau de ce paragraphe, nous allons utiliser le **graphe biparti** associé au jeu à deux joueurs (cf figure 7.5 pour le cas du jeu de Chomp 2×2).

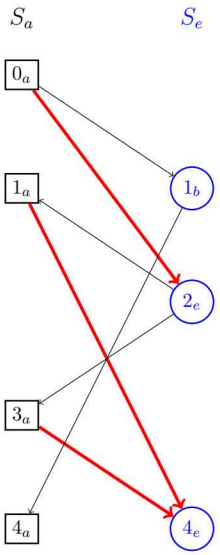

FIGURE 7.5 – Graphe biparti associé au jeu de Chomp $(2,2)$. Chaque état indiqué est une combinaison du numéro de la configuration de jeu (cf figure 7.13) et du nom du joueur qui joue.

Tout jeu partial peut se représenter par un tel graphe.

Théorème 7.2.1 — Jeu sur un graphe biparti. Soit un jeu partial à deux joueurs, Adam et Eve, potentiellement cyclique. Alors le jeu peut se modéliser par un graphe $\mathscr{G} = (S, \mathscr{A})$ qui vérifie les propriétés suivantes :

1. chaque sommet du graphe correspond à un et un seul état du jeu,
2. le graphe est biparti. C'est à dire qu'il existe deux sous ensembles disjoints de S, notés S_a et S_e tels que $S = S_a \cup S_e$ et telle que toute arrête $(u,v) \in \mathscr{A}$ vérifie **a)** soit $u \in S_a$ et $v \in S_e$, soit **b)** $u \in S_e$ et $v \in S_a$.
3. il existe un ensemble $\Omega_A \subset S_A$ de sommets gagnants pour Adam et un ensemble $\Omega_E \subset S_E$ de sommets gagnants pour Eve.

Démonstration. L'existence du graphe du jeu est assuré par le caractère déterministe et fini du jeu. En effet, étant donné un état du jeu et une action, le déterminisme garantit que cette action conduira toujours au même état.

L'existence de sommets gagnants et perdants vient de la définition même du jeu. ■

Remarquons que l'existence de ce graphe est principalement une commodité théorique. En raison du grand nombre d'états possible, il est très difficile de représenter concrètement les graphes correspondant aux jeux usuels (échecs, go, ou même morpion 3×3). Donnons désormais deux exemples de graphes de jeu :

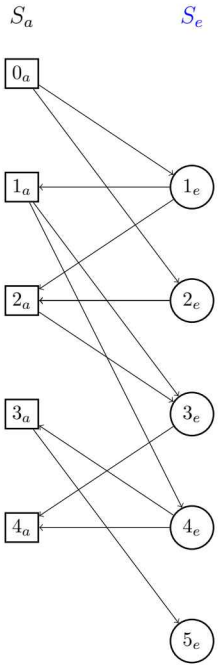

FIGURE 7.6 – Dans cet exemple, le graphe biparti est acyclique.

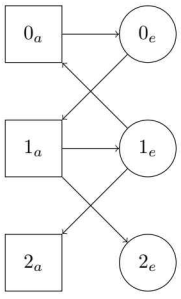

FIGURE 7.7 – Dans cet exemple il existe un cycle.

Sur les exemples représentés par les figures 7.6 et 7.7, il saute aux yeux que la propriété d'acyclicité permet d'éviter les parties infinies. Nous reviendrons sur ce point ultérieurement.

7.2.2 Etats et stratégies gagnants pour Adam dans un graphe biparti

Calcul des états gagnants pour Adam.

Les états permettant à Adam de l'emporter sont de deux natures :

1. soit ce sont des états d'Adam (où c'est à lui de jouer) qui sont gagnantes,
2. soit ce sont des états où Eve n'a pas d'autre choix que de jouer en envoyant sur un état gagnant d'Adam.

Cela suggère de définir la notion d'**attracteur** :

Définition 7.2.1 — Attracteur d'Adam au sens du graphe biparti. Soit $\mathscr{G} = (S, \mathscr{A})$ le graphe biparti d'un jeu. On appelle **attracteur d'Adam** l'ensemble défini par la procédure algorithmique [a] suivante :

- On initialise l'attracteur comme étant l'ensemble des états gagnants pour Adam. Autrement dit $\mathscr{A}tt_a^0 \leftarrow \Omega_a$
- On ajoute à l'attracteur les noeuds étiquetés :
  - soit par des états d'Eve dont **tous** les successeurs sont dans l'attracteur d'Adam (états pièges de Eve),
  - soit par des états d'Adam dont **au moins un** successeur se trouve dans l'attracteur d'Adam.

  Autrement dit : on met à jour $\mathscr{A}tt_a^{i+1} \leftarrow \mathscr{A}tt_a^i \cup \left\{s \in S_a, \exists v \in \mathscr{A}tt_a^i, (s,v) \in \mathscr{A}\right\} \cup \left\{s \in S_e, \forall v \in S, (s,v) \in \mathscr{A} \Rightarrow v \in S_a\right\}$.
- On s'arrête dès que $\mathscr{A}tt_a^i = \mathscr{A}tt_a^{i+1}$.

a. Une telle définition, par récurrence, se nomme **définition inductive** en informatique.

Démonstration. Il faut vérifier que la définition donnée plus haut en est bien une et que la procédure ainsi définie termine toujours. Raisonnons par l'absurde et supposons que la procédure ne termine jamais. Cela signifie qu'il est toujours possible de rajouter dans l'attracteur un sommet qui n'a pas été ajouté auparavant. Cela conduirait donc à ajouter une infinité de sommets. C'est contradictoire avec le caractère fini des graphes que nous étudions. ■

La définition précédente donne un algorithme récursif simple pour calculer l'attracteur pour Adam (et donc a fortiori les états gagnants pour lui).

Pour cela, étant donné le graphe biparti du jeu, il suffit :

1. D'initialiser l'attracteur avec les états gagnants pour Adam.
2. Tant qu'il est possible d'ajouter des sommets, on ajoute à l'attracteur :
   - les états d'Adam à l'origine d'un arc pointant vers un état déjà dans l'attracteur,
   - les états d'Eve dont tous les arcs pointent vers l'attracteur.

R Les états de Eve qui se trouvent dans l'attracteur s'appellent le **piège de Eve**.

Etudions deux exemples

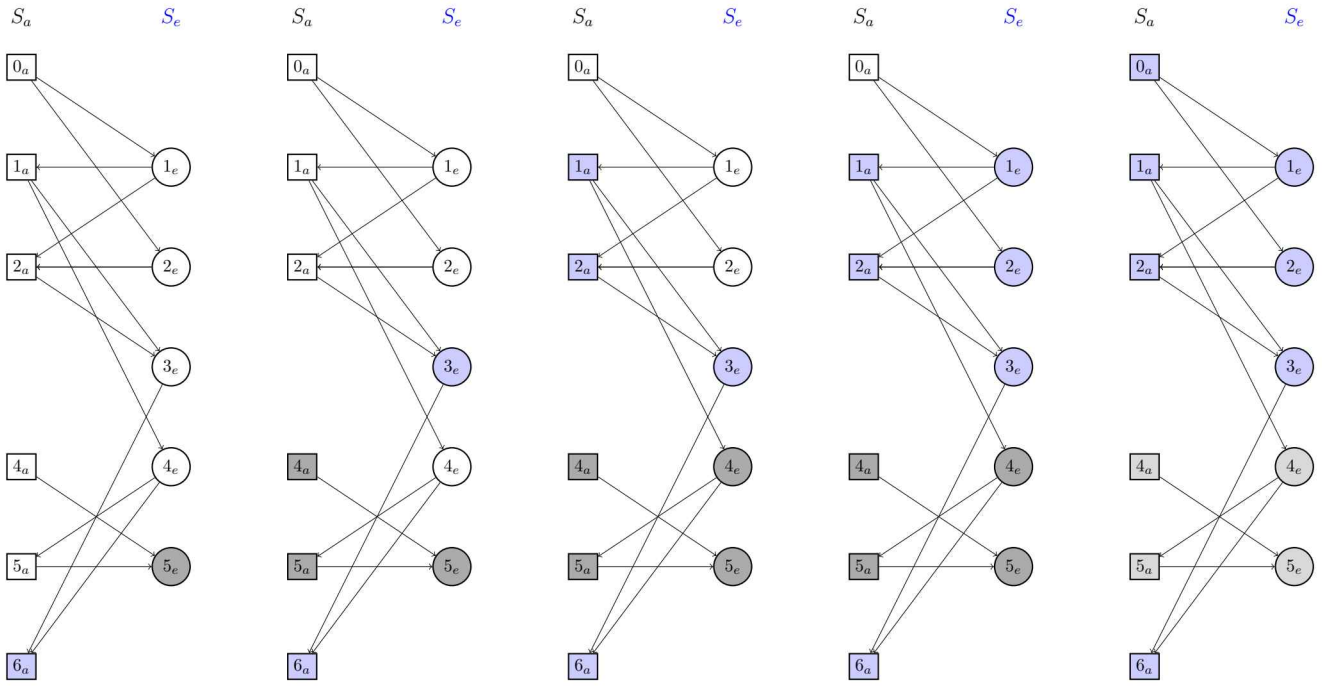

FIGURE 7.8 – Illustration du calcul des attracteurs de Adam (bleu) et de Eve (gris) sur un exemple. On suppose qu'Adam ne possède qu'un seul état gagnant (6_a) puis on ajoute d'abord l'état 3_e (mais pas le 4_e). Eve ne dispose également que d'un état gagnant : le 5_e.

Sur l'exemple de la figure 7.8, on voit que l'ensemble des états sont coloriés à la fin de la procédure ; alors que dans la figure 7.9, certains états ne sont pas coloriés en fin de calcul des attracteurs.

Intérêt des attracteurs : on sent intuitivement que l'attracteur est une notion qui permet de savoir s'il existe une stratégie gagnante pour un joueur. En effet, si Adam arrive à attirer Eve dans son attracteur, elle n'aura d'autre choix que de jouer des coups qui laisseront la possibilité à Adam de rester dans son attracteur ; jusqu'à finalement gagner le jeu.

7.3 L'étude des jeux partiaux sans cycles

Dans cette partie, nous allons nous focaliser sur les jeux partiaux sans cycles. Nous allons démontrer que le calcul des attracteurs permet, pour ces jeux, d'obtenir une stratégie gagnante pour les joueurs.

FIGURE 7.9 – Sur cet exemple, le calcul se termine très vite, en raison du cycle.

Des cas typiques de tels jeux sont le jeu de Reversi, le jeu de Morpion ou encore le jeu de Puissance 4.

Dans ces jeux en effet, le nombre de cases libres diminue de une unité à chaque coup joué ; ce qui garantit que la partie se terminera.

Néanmoins le cas des jeux cycliques n'est pas si restrictif que cela, en effet on peut décréter qu'on n'a pas le droit de repasser une infinité de fois par le même état de jeu (imaginez si une telle situation se passait dans une partie réelle de jeu d'échec).

Pour pouvoir faire l'étude d'un jeu sans cycle, comme celui de la figure 7.6, il est naturel d'introduire une autre représentation ; moins économique au sens où elle va faire apparaître plusieurs fois la même information mais qui va avoir l'avantage de s'analyser facilement sur le plan mathématique.

7.3.1 Représentation arborescente

Comme illustré dans la figure 7.10, il est assez naturel de doter un tel jeu d'une *représentation arborescente*. Il s'agit d'un graphe orienté dans lequel on distingue les états d'Adam et de Eve. Les arêtes du graphe correspondent aux actions (transitions) possible en jouant au jeu.

On peut faire plusieurs remarques concernant les propriétés de ce graphe :

— chaque noeud du graphe est étiqueté par un état de jeu,
— un même état de jeu peut apparaître plusieurs fois, s'il est accessible de plusieurs façons,
— un état, l'état initial de la partie a un statut particulier, on dit alors que c'est la **racine** de l'arbre,
— le nombre d'arêtes entre la racine et un noeud (nombre de coups joués) s'appelle la **profondeur** du noeud,
— la profondeur maximale d'un noeud s'appelle la **hauteur** de l'arbre,
— les états sans successeurs sont exactement les états finals ; on les appelle les **feuilles** de l'arbre,
— une branche de l'arbre correspond exactement à une partie possible.

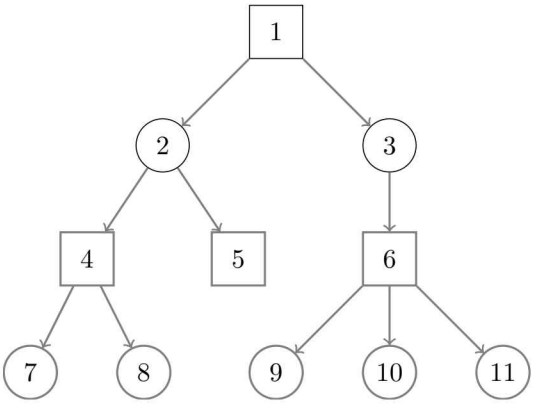

FIGURE 7.10 – Un exemple de représentation arborescente pour un jeu. Les états en rectangle correspondent aux états S_a d'Adam et ceux en cercle aux états S_e de Eve.

Attention cependant à ne pas croire qu'il y a une bijection entre les états et les noeuds. En effet, plusieurs séquences d'actions pouvant mener au même état, des noeuds distincts peuvent être étiquetés par un même état. Notons cependant que, dans ce cas, les sous arbres extraits à partir des noeuds étiquetés par le même état sont identiques.

Il faut plutôt voir l'arbre du jeu comme la description exhaustive de toutes les parties possibles.

Cette situation est représentée par l'arbre de la figure 7.11.

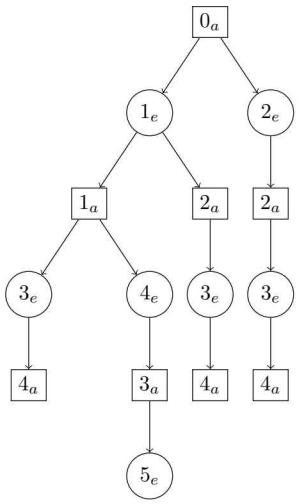

FIGURE 7.11 – Dans l'arbre du jeu un même état peut revenir plusieurs fois **mais** lorsqu'il revient les sous arbres associés sont toujours les mêmes (par exemple quand on regarde les branches issues de 2_a dans ce graphe, elles sont toutes identiques).

7.3.2 Détermination des stratégies et états gagnants

Attracteurs.

Comme nous l'avons dit lorsque nous nous intéressions à la représentation par graphe biparti, certains états de jeux sont plus favorables que d'autres.

Bien évidemment, les états gagnants sont particulièrement favorables mais au delà de ceux-ci on peut remarquer que, du point de vue d'Adam, les noeuds les plus intéressants dans l'arbre du jeu sont :

— les noeuds d'Adam dont au moins un des noeuds successeur est gagnant (il suffit alors à Adam de jouer le coup gagnant),
— les noeuds de Eve dont tous les successeurs sont gagnants pour Adam (du point de vue de Eve, on dira que ce noeud est un *piège*),
— de manière récursive, nous avons ainsi pu définir les noeuds les plus favorables à Adam, ce sont ceux ci que nous allons appeler l'*attracteur d'Adam* au sens de la représentation arborescente du jeu.

On doit noter, dans le contexte de l'arbre du jeu ; il n'est pas encore clair qu'on puisse définir l'attracteur simplement à partir de l'arbre. En effet ; chaque état pouvant être l'étiquette de plusieurs noeuds de l'arbre et il faudra s'assurer que tous les noeuds étiquetés par un même état se trouvent en même temps dans l'attracteur d'Adam.

Commençons donc par définir une notion d'attracteur dans le contexte de la représentation arborescente du jeu.

Définition 7.3.1 — Attracteur d'Adam au sens de l'arbre du jeu. On appelle **attracteur d'Adam** l'ensemble défini par la procédure algorithmique [a] suivante :

— On initialise $\mathscr{A}tt_a$ à Ω_a l'ensemble des noeuds étiquetés par des états gagnants pour Adam.
— On met à jour récursivement à jour l'ensemble $\mathscr{A}tt_a$ en ajoutant
  1. les noeuds d'Adam dont **au moins un** successeur se trouve dans $\mathscr{A}tt_a$,
  2. les noeuds de Eve dont **tous** les successeurs se trouvent dans $\mathscr{A}tt_a$ (états *pièges* pour Eve).
— On s'arrête dès que l'attracteur d'Adam ne grossit plus.

a. Une telle définition, par récurrence, se nomme **définition inductive** en informatique.

R Le nombre total d'état étant fini, la procédure précédente s'arrête forcément.

On peut bien entendu définir l'attracteur d'Eve de manière symétrique.

Justifions que les deux notions d'attracteurs sont congruentes.

Proposition 7.3.1 Un état est dans l'attracteur d'Adam au sens du graphe biparti si, et seulement si, tous les noeuds étiquetés par cet état dans la représentation arborescente sont dans l'attracteur d'Adam au sens de l'arbre du jeu.

Démonstration. Cela découle de la remarque faite précédemment sur les sous arbres extraits :

tous les noeuds étiquetés par un certain état s ont le même arbre descendant. La définition inductive de l'attracteur en commençant par les feuilles garantit alors que chaque noeud étiquetés par l'état s seront ajoutés au même moment. ■

Exemple de calcul des attracteurs.

Le calcul des attracteurs d'Adam et de Eve peut se faire à partir de l'arbre du jeu par un algorithme de **propagation de couleur**.

— On commence par colorier les feuilles correspondant aux états finals gagnants pour l'un des deux joueurs (en bleu pour Adam et gris pour Eve),

— on colorie en bleu :
  1. les états d'Adam dont **au moins un** successeur est bleu,
  2. les états de Eve dont **tous** les successeurs sont bleus.
— on colorie en gris :
  1. les états d'Eve dont **au moins un** successeur est gris,
  2. les états de Adam dont **tous** les successeurs sont gris,
— on s'arrête lorsqu'on ne peut plus appliquer cette procédure. L'attracteur d'Adam est alors colorié en bleu et celui de Eve en gris.

On va calculer les attracteurs des deux joueurs dans le cas de l'exemple de la figure 7.10.

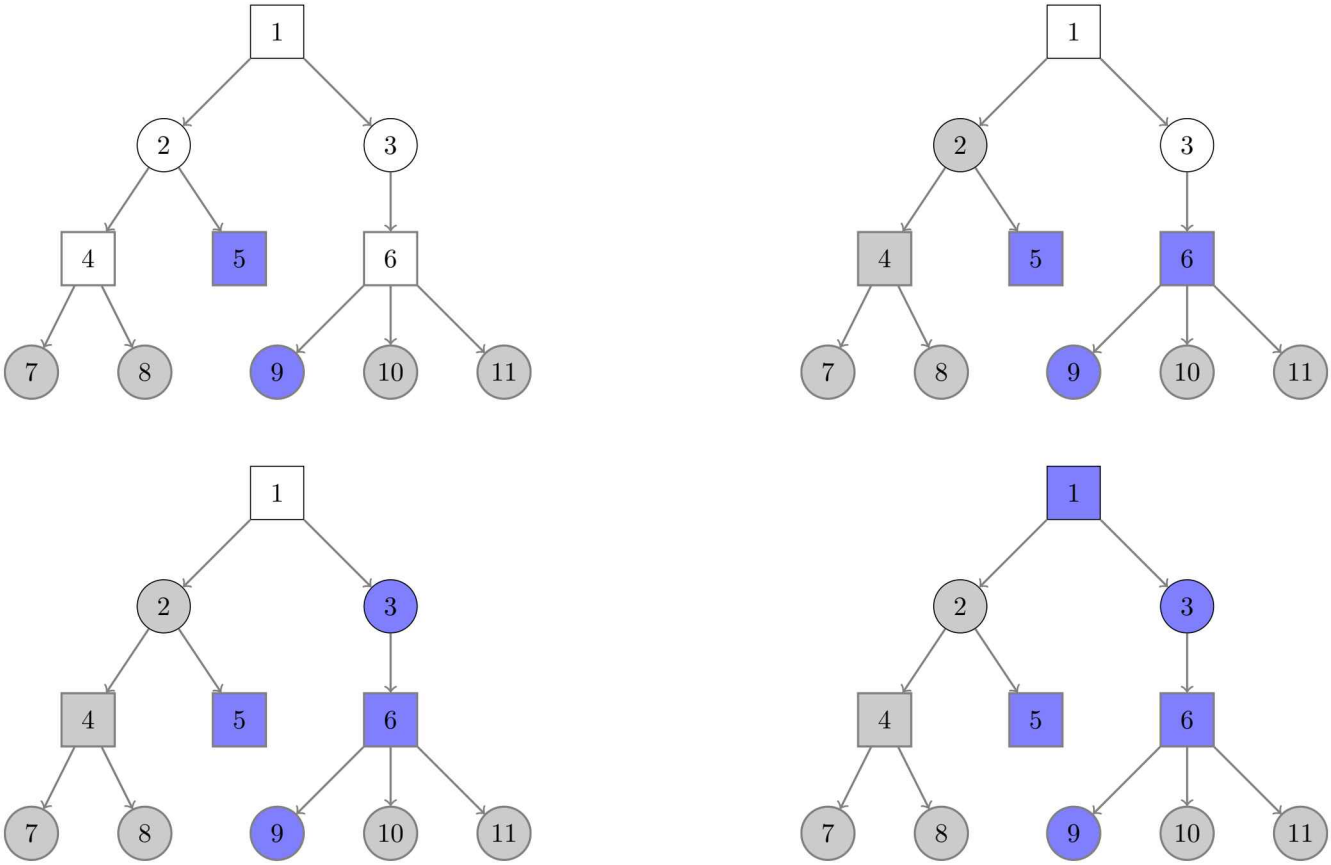

FIGURE 7.12 – Algorithme de "propagation de couleur".

Proposition 7.3.2 — Interprétation du coloriage d'un arbre associé à un jeu sans partie nulle. Les noeuds coloriés en gris sont **exactement** les noeuds correspondant à un état de l'attracteur de Eve. Les noeuds coloriés en bleu sont **exactement** les noeuds correspondant à un état de l'attracteur de Adam.

Théorème de Zermelo.

On peut se convaincre intuitivement que la procédure de propagation de couleur va finir par colorier l'intégralité de l'arbre.

Théorème 7.3.1 — Coloriage d'un arbre associé à un jeu sans partie nulle. L'algorithme de coloriage colorie l'intégralité de l'arbre du jeu.

Démonstration. On peut procéder par l'absurde et supposer qu'il existe un état de jeu qui ne soit ni dans l'attracteur d'Adam ni dans celui de Eve. Soit N un noeud non colorié de profondeur maximale alors :
— N n'est pas une feuille (qui sont coloriées à la première étape),
— si N est étiqueté par un état d'Adam alors ses successeurs (qui sont de profondeur supérieure à celle de N et qui sont donc coloriés par hypothèse d'absurdité) sont tous étiquetés par des des états de l'attracteur de Eve, mais alors par définition de l'attracteur de Eve, N est dans l'attracteur de Eve. Donc N devrait être colorié,
— de manière analogue, N ne peut pas être un état d'Adam.

Ainsi la procédure de coloriage colorie tous les noeuds de l'arbre. ■

On peut alors démontrer le théorème suivant, dû à Zermelo :

Théorème 7.3.2 — Theoreme de Zermelo. Soit un jeu à deux joueurs au tour par tour, acyclique, partial et sans partie nulle. Un des deux joueurs dispose d'une stratégie gagnante.

Démonstration. C'est une conséquence du théorème de coloriage.

— On considère un arbre de hauteur h. Par le théorème 7.3.1, la racine a été coloriée. Pour fixer les idées, nous allons supposer qu'elle a été coloriée en bleu et qu'elle correspond donc à un état dans l'attracteur d'Adam.

— On remarque que chaque noeud étiqueté par un état de l'attracteur d'Adam (bleu) dispose d'au moins un successeur qui est également étiqueté dans l'attracteur d'Adam. On peut alors définir, pour chaque état d'Adam qui est dans l'attracteur d'Adam une stratégie conduisant à jouer un coup envoyant sur un noeud étiqueté par un état d'Eve et colorié en bleu (donc également dans l'attracteur d'Adam).

— Vérifions que cette stratégie est gagnante : le premier coup d'Adam envoie sur un état de Eve située dans $\mathscr{A}tt_a$ et à profondeur 1. Comme ce coup est un coup de Eve, il ne peut être dans l'attracteur que si, quel que soit le coup joué par Eve, cela conduise sur un état d'Adam colorié en bleu à profondeur 2, puis le coup d'Adam conduit à un état bleu de Eve à profondeur 3 etc.

— L'arbre étant de hauteur finie, cette procédure finit par terminer et donc tombe sur une feuille dans l'attracteur d'Adam, autrement dit un état gagnant pour Adam.

■

7.4 Les jeux impartiaux

Définition 7.4.1 — Jeu impartial. Un jeu est dit *impartial* si un observateur extérieur observant deux situations de jeu successives ne peut pas déterminer si le coup a été joué par Adam ou Eve.

Nous allons maintenant nous intéresser aux jeux impartiaux ; c'est à dire les jeux où les deux joueurs ont les mêmes levier d'action sur le cours du jeu étant donné une configuration. Commençons par présenter un des plus fameux jeux de cette catégorie.

A priori, et contrairement à ce que la terminologie pourrait laisser entendre, un jeu impartial n'est qu'un cas particulier de jeu partial. Néanmoins nous allons voir que les propriétés inhérentes de ces jeux permettent d'interpréter ces jeux comme étant le déplacement d'un pion sur un graphe et que l'on peut en déduire des propriétés très intéressantes et utiles pour l'analyse théorique de ces jeux.

7.4.1 Le jeu de Chomp

Revenons sur le jeu de Chomp. On va considérer une partie de Chomp avec une tablette de taille 2×2. Sur la figure 7.13 on représente les diverses configurations possibles.

Chacune de ces configurations est accessible à l'un ou l'autre joueur ; les états sont donc simplement des paires composées d'une configuration et d'un identifiant (a ou e) indiquant si c'est à Adam ou Eve de jouer.

On peut, à l'aide de la théorie des jeux partiaux, construire le graphe biparti suivant :

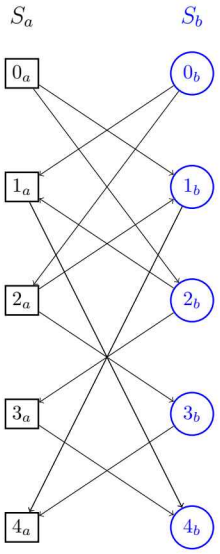

FIGURE 7.13 – Graphe biparti associé au jeu de chomp 2 par 2 dans lequel Adam commence systématiquement.

Une propriété saute alors aux yeux : ce graphe possède un axe de symétrie. C'est logique car, Adam et Eve ont les mêmes leviers d'action sur le plateau de jeu et sont indiscernables du point de vue d'un observateur extérieur.

Cette propriété, observée dans le cas du jeu de Chomp peut donc être érigée au rang de proposition :

Proposition 7.4.1 — Graphe biparti associé à un jeu impartial. Le graphe biparti d'un jeu impartial :

— comporte autant d'état d'Adam et de Eve,

— est symétrique c'est à dire que s'il existe une arête entre l'état numéro i d'Adam et le numéro j de Eve, alors il existe une arête entre l'état numéro i de Eve et le j d'Adam.

7.4.2 Arène d'un jeu impartial

La propriété de symétrie du graphe précédent indique donc que les états ne dépendent que de l'ordre dans lequel les joueurs jouent.

Pour un jeu de ce type, plutôt que de considérer les états qui sont des paires formées d'une configuration de jeu **ET** d'un joueur dont "c'est le tour de jouer", on peut se borner à considérer les *configurations de jeu*. Le graphe biparti d'un jeu impartial étant symétrique, on peut construire de manière univoque un graphe orienté (S,A) appelé arène.

Chaque noeud du graphe représente une configuration possible du jeu. Un unique noeud n'a pas de prédecesseur, c'est la configuration initiale (dans le cas de chomp, personne n'a encore touché à la tablette !). Les arêtes du graphe représentent les différents coups possibles pour les joueurs, en reliant deux configurations entre elles. Par exemple, dans le jeu de Chomp, la figure 7.14 montre comment est représenté le graphe associé à une tablette de taille (2,2). Les joueurs se déplacent tour à tour sur les arêtes du graphe, en partant de la configuration courante et en se déplaçant vers une configuration suivante. Le but du jeu est de suivre un chemin sur ce graphe à partir d'un point de départ donné. Ce jeu est appelé "jeu d'accessibilité" car il n'y a pas de cycle dans le graphe, ce qui garantit que la partie est finie. La fin de la partie est déterminée par les configurations qui n'ont pas de successeurs. Dans le jeu "Chomp", la seule configuration qui entraîne la fin de la partie est celle du carré empoisonné, atteindre cette configuration signifie la victoire/défaite pour celui qui y parvient.

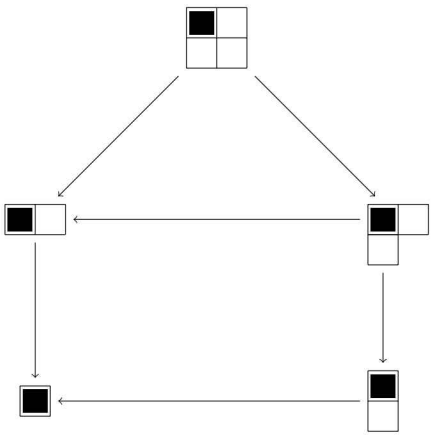

FIGURE 7.14 – Arène associée au jeu de Chomp $(2,2)$.

Remarque : en règle générale, on préfèrera représenter l'arène en donnant un nom "logique" à chacun des noeuds du graphe, cf figure 7.15.

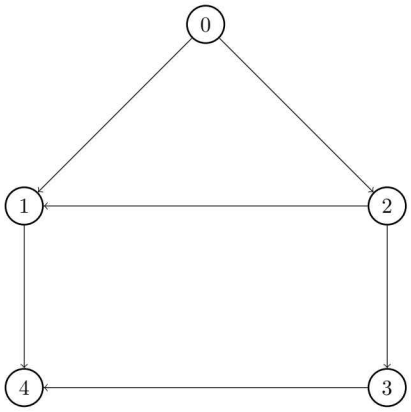

FIGURE 7.15 – Arène logique associée au jeu de Chomp $(2,2)$.

On peut voir une **partie** du jeu comme une suite de sommets reliés par des arêtes dans le graphe biparti et qui part de l'état initial pour arriver dans un des états finals. Mais on peut également voir une partie comme le déplacement d'un pion sur l'arène du jeu ; Adam et Eve déplaçant successivement le pion le long d'une arête jusqu'à atteindre un état gagnant.

7.4.3 Détermination de stratégies et états gagnants à partir de l'arène du jeu

Fonction de Sprague-Grundy (nimber) associée à l'arène d'un jeu impartial acyclique

Hypothèse : nous supposons que le jeu est acyclique. Par conséquent son arène sera acyclique. Un exemple de telle arène est représentée dans la figure 7.16

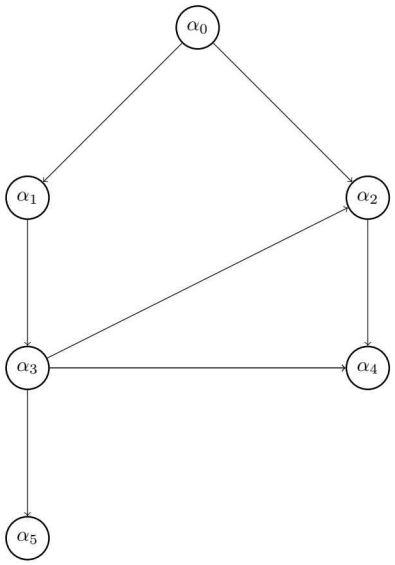

FIGURE 7.16 – Un graphe connexe acyclique. Les sommets $1,2$ sont les successeurs du sommet 0.

Quelques rappels de théorie des graphes pourraient nous servir :

Proposition 7.4.2 — Propriétés des graphes orientés acyclique. Soit $\mathscr{G} = (S,A)$ un graphe orienté acyclique alors :

1. il existe un sommet sans prédecesseur,
2. il existe au moins un sommet sans successeur,
3. si l'on retire un sommet sans successeur et toutes les arêtes pointant vers ce sommet, le graphe résultant est acyclique.

Dans les années 1930, les mathématiciens Sprague et Grundy ont développé l'étude spécifique des jeux impartiaux. Ils ont, pour cela, introduit la fonction suivante, dite de Sprague-Grundy (également appelée *nimber*).

Définition 7.4.2 — Fonction de Sprague-Grundy. Soit $\mathscr{G} = (S,A)$ un graphe orienté acyclique. La fonction de Sprague-Grundy de chaque sommet peut être définie de manière récursive en commençant par les feuilles. Pour cela, on peut suivre les étapes suivantes :

1. On définit la fonction de Sprague-Grundy des sommets terminaux (c'est-à-dire ceux qui n'ont pas de successeur) comme étant $\mathrm{SG}(s) = 0$.
2. On examine chaque sommet s : si la fonction de Sprague Grundy de s n'est pas connue mais que celle des successeurs de s est connue, on calcule la fonction de Sprague-Grundy du sommet s comme étant le "mex" (minimum exclu) de l'ensemble S des successeurs de s : $S = \{s', \exists (s,s') \in A\}$, c'est-à-dire le plus petit entier naturel (c'est à dire positif ou nul) non présent dans l'ensemble S. Autrement dit :

$$\mathrm{SG}(s) = mex\{\mathrm{SG}(s') | (s,s') \in A\}.$$

Démonstration. Il n'est pas si clair que la fonction de Sprague-Grundy ainsi définie permet d'attribuer un nombre à chaque sommet du graphe.

Pour le démontrer, on peut procéder par récurrence sur le nombre n de sommets du graphe. Considérons la propriété $\mathscr{P}(n)$ suivante :

$\mathscr{P}(n)$: «Pour un graphe à n sommet, la fonction de Sprague-Grundy définie par la définition récursive précédente attribue une valeur à chaque sommet.»

— **Initialisation :** pour un graphe à 1 seul sommet, on attribue la valeur 0 à cet unique sommet et c'est terminé.

— **Hérédité :** supposons la propriété vraie pour les graphes à n noeuds. Soit un graphe à $n+1$ noeuds. Il existe donc un sommet s_0 sans prédécesseur. Retirons le provisoirement du graphe. On obtient alors un graphe à n noeuds. Par **hypothèse de récurrence** on peut donc attribuer une valeur de Sprague-Grundy a chacun de ses sommets. On peut ensuite reconnecter le sommet s_0 en lui attribuant pour valeur le mex de ses successeurs, qui a bien été défini.

— La propriété est donc vraie en vertu du principe de récurrence.

■

▪ **Exemple 7.2** Calculons la fonction de Sprague Grundy du graphe de la figure 7.16.

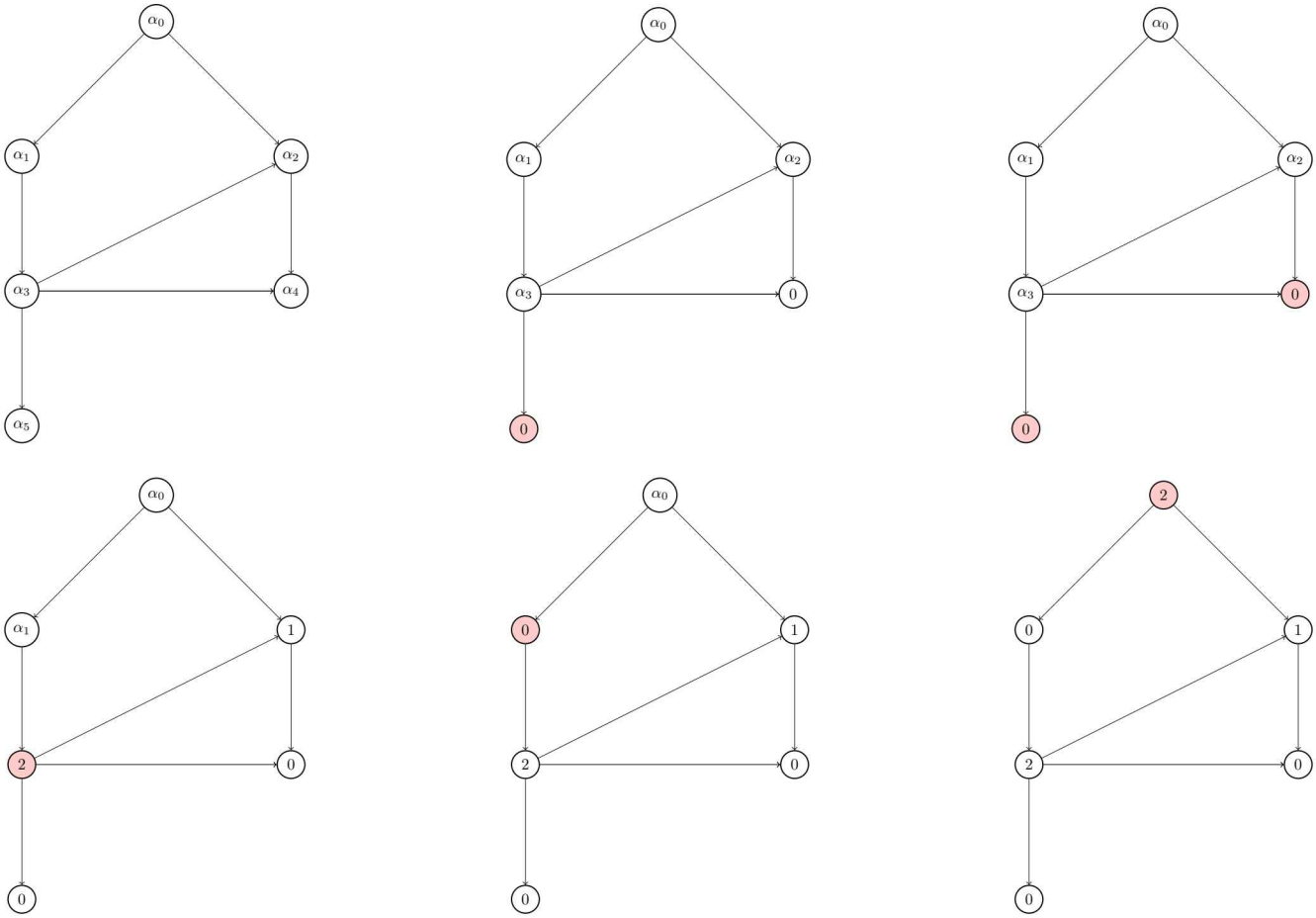

FIGURE 7.17 – Exemple de calcul de la fonction de Sprague-Grundy sur le graphe de la figure 7.16.

Détermination des configurations gagnantes

L'enjeu est désormais de déterminer le lien entre configuration gagnante et fonction de Sprague-Grundy.

Hypothèse : on suppose que le jeu ne comporte pas de partie nulle.

Remarque : dans l'arène, les noeuds sans successeurs correspondent aux états finals. Le joueur qui se retrouve dans un noeud sans successeur et qui doit jouer a donc **perdu**. Par exemple, au jeu de chomp, si Adam se retrouve dans la configuration où il ne reste que la case empoisonnée, il a donc perdu.

Définition 7.4.3 — états de type N et P. Soit $\mathscr{G} = (S,A)$ l'arène d'un jeu impartial sans cycle.
1. On appelle état de type N toute état dont la valeur de Sprague-Grundy est nulle.
2. On appelle état de type P toute état dont la valeur de Sprague-Grundy est strictement positive.

Proposition 7.4.3 Soit s un sommet, alors :
— si s est de type N, tous ses éventuels successeurs sont de type P,
— si s est de type P, il existe un successeur de s qui est de type N.

Démonstration. Si s est de type N, alors ses successeurs ont tous un nombre de Sprague-Grundy différent de 0 et donc strictement positif.

Si s est de type P, au moins un de ses successeurs a un nombre de Sprague-Grundy nul (sinon s aurait du se voir attribuer la valeur 0).

■

Définition 7.4.4 — Configuration gagnante. On dit qu'une configuration est gagnante pour le joueur dont c'est le tour de jouer si, il existe une stratégie gagnante à partir de cette configuration.

Théorème 7.4.1 — Theorème de Sprague-Grundy. On considère un jeu à deux joueurs sans partie nulle, représenté par une arène acyclique. Une configuration est gagnante pour le joueur qui joue si, et seulement si, elle est de type P. Pour le joueur qui joue, il suffit alors de systématiquement choisir un coup qui mène à un état de type N.

Démonstration. On peut le faire par récurrence sur n le nombre de noeuds dans l'arène.

— **Initialisation :** pour un graphe à 1 seul sommet, c'est une unique feuille, de valeur de Sprague-Grundy 0 et le théorème est vrai puisque le joueur qui joue a perdu.
— **Hérédité :** supposons la propriété vraie pour les graphes à n noeuds. Soit un graphe à $n+1$ noeuds. Considérons s_0 le sommet initial, qui n'a donc pas de prédecesseur :
  — si c'est un sommet de type P, supposons que le joueur qui joue choisit une configuration s_1 de type N.
  — si c'est un sommet de type N, alors le joueur qui joue choisit une configuration s_1 de type P. Comme le graphe est acyclique, on ne peut pas revenir au sommet initial. Retirons le ainsi que toutes les arêtes qui en partent, et considérons que le jeu commence en configuration s_1. En appliquant la propriété au nouveau graphe (de n noeuds), on obtient le résultat voulu.
— La propriété est donc vraie en vertu du principe de récurrence.

■

Chapitre 8

Modèles arborescents pour l'étude des jeux

Plan

8.1 Réinterprétation de l'algorithme de propagation de couleur 94
8.2 Notion d'heuristique 95
8.2.1 Définition d'une heuristique . . . 95
8.2.2 L'exemple du Reversi . . . 95
8.3 Algorithme min-max 95
8.3.1 Choix du coup à l'aide de l'heuristique . . . 95
8.3.2 Un exemple au Reversi. . . . 95
8.3.3 Principe de l'algorithme min-max . . . 96

Le chapitre précédent nous a donné une méthode puissante de résolution de nombreux jeux usuels par le calcul des attracteurs. Néanmoins, nous n'avons utilisé ces techniques que sur des jeux relativement simples (nim, chomp etc.) en disant que la méthode se généralisait sans peine aux jeux à information complète et sans hasard. C'était quelque peu hâtif.

L'algorithme de calcul des attracteurs que nous avons décrit a une complexité dans le pire des cas en $O(n^3)$, où n est le nombre de sommets du graphe, autrement dit le nombre de configurations que l'on peut rencontrer lors d'une partie. Cependant, cet entier n est souvent extrêmement grand : il est estimé de l'ordre de 10^{32} pour les dames, entre 10^{43} et 10^{50} pour le jeu d'échecs, de l'ordre de 10^{20} pour le reversi sur un plateau 8×8, de l'ordre de 10^{100} pour le jeu de go. Il apparaît donc impossible, en termes de calcul, de calculer les attracteurs pour ces jeux.

Nous allons donc délaisser la représentation d'un jeu sous forme d'arène ou de graphe biparti pour n'utiliser que la représentation arborescente du chapitre précédent.

Nous allons partir de l'idée suivante, très simple, appelée la force brute : étant donné l'état courant du jeu, le joueur qui doit jouer va choisir le coup qui maximise ses chances de gain. Pour déterminer un tel coup, il lui suffit a priori d'appliquer l'algorithme de coloriage de l'arbre ; dont la complexité est rédhibitoire pour les jeux issus de la vie réelle.

Néanmoins, aussi inapplicable qu'elle soit, cette approche porte en elle une idée intéressante. A partir de la configuration courante, on ne va dresser que **le début** de l'arbre du jeu et on choisira celle des branches qui a l'air le plus favorable.

Bien évidemment, comme le but est de ne pas explorer l'arbre en entier, nous allons devoir être capable de trouver une fonction nous permettant de comparer des configurations du jeu entre elles. En d'autres termes nous allons avoir besoin d'une fonction affectant à chaque configuration une certaine valeur dont le but sera de la maximiser (du point de vue d'Adam) ou de la minimiser (du point de vue de Eve). L'enjeu est de donner un sens quantifiable à "avoir l'air favorable".

Pour cela, nous allons devoir avoir recours à une notion, déjà étudiée dans le cas de la recherche de plus court chemin dans un graphe : la notion d'**heuristique**.

En pratique, c'est en se fondant sur une telle approche, que les intelligences artificielles battant les meilleurs joueurs d'échec ont pu être mises en oeuvres avant d'être adaptées pour le jeu de Go ou le jeu de stratégie StarCraft.

8.1 Réinterprétation de l'algorithme de propagation de couleur

Reprenons l'exemple du chapitre précédent, quand nous avions calculé une stratégie gagnante en coloriant l'arbre du jeu.

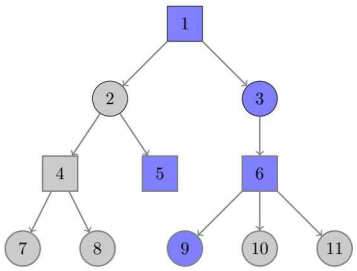

FIGURE 8.1 – En bleu, les états gagnants d'Adam et en bleu ceux d'Eve.

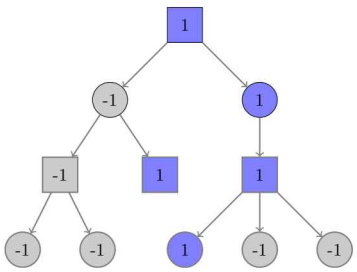

FIGURE 8.2 – On remplace les états gagnants d'Adam par 1 et ceux de Eve par -1.

L'algorithme de propagation de couleur depuis les feuilles peut donc se voir comme la remontée d'une information depuis les feuilles de l'arbre vers la racine suivant la règle suivante (qui sera ce que nous allons appeler la règle du **min-max**) :

— les états d'Adam propagent le maximum de la valeur des noeuds successeurs,

— les états de Eve, propagent le minimum des valeurs des noeuds successeurs.

Dans la figure 8.2, on fait remonter la valeur 1 ou -1 car on a été capable de déterminer précisément si un noeud permettait d'accéder à un état final gagnant ou pas. Néanmoins, cette information numérique pourrait être différente...

C'est la clef de voûte pour la suite. En effet, nous allons reprendre l'idée de la propagation de couleur, mais cette fois, au lieu de propager une information binaire correspondant à l'existence d'une stratégie gagnante, nous allons propager des nombres correspondant à des *estimations* de la valeur d'une configuration.

C'est ce que l'on appellera dans la suite une **heuristique**. L'idée sera, partant d'une configuration courante, de bâtir le début de l'arbre (sur quelques étages, de sorte que tout cela reste informatiquement calculable en temps raisonnable), puis de

calculer les valeurs des configurations atteintes au niveau des feuilles de cet arbre partiellement déroulé (les configurations ne seront pas des états finals), et enfin de faire propager les valeurs vers la racine de l'arbre en supposant que les deux joueurs jouent de manière rationnelle ; c'est à dire que :

— Adam sera supposé chercher à **maximiser** la valeur de la position qu'il peut atteindre,
— Eve sera supposée chercher à **minimiser** la valeur de la position qu'elle peut atteindre,

8.2 Notion d'heuristique

8.2.1 Définition d'une heuristique

Définition 8.2.1 — Heuristique. Soit un jeu à deux joueurs, Adam et Eve, où l'ensemble des configurations licites est noté $\mathscr{P}$. On dit qu'une fonction $h : \mathscr{P} \to \mathbb{R}$ est une **heuristique** pour Adam si elle répond aux critères suivants :

1. elle est facile à calculer par l'examen de la configuration,
2. si la valeur de la fonction h pour une configuration p est élevée, cela signifie que la configuration est favorable à Adam,
3. si la valeur de la fonction h pour une configuration p est faible, cela signifie que la configuration est favorable à Eve,
4. elle est maximale pour une configuration gagnante pour Adam et minimale pour une configuration perdante.

▪ **Exemple 8.1** Voici quelques exemples d'heuristiques pour des jeux usuels :

— Pour le jeu d'échecs, une heuristique pourrait être la différence de nombre de pièces entre les deux joueurs. Plus la différence est grande en faveur d'Adam, plus l'heuristique sera élevée.
— Pour le jeu de dames, une heuristique pourrait être le nombre de pions et de dames possédés par chaque joueur. Plus Adam a de pièces et de dames par rapport à Eve, plus l'heuristique sera élevée.
— Pour le jeu de Go, une heuristique pourrait être le nombre de points de contrôle sur le plateau de jeu. Plus Adam a de points de contrôle par rapport à Eve, plus l'heuristique sera élevée.

8.2.2 L'exemple du Reversi

Pour illustrer concrètement cette section, nous allons utiliser l'exemple du jeu de Reversi.

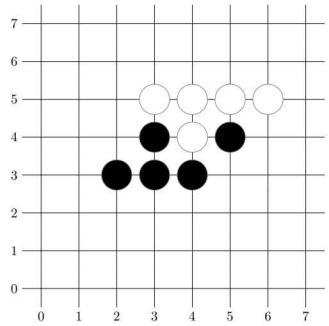

FIGURE 8.3 – Une configuration licite de reversi.

Une heuristique simple consiste à compter le nombre de pions noirs sur le plateau. Le joueur noir va chercher à maximiser ce nombre. Dans l'exemple de la Figure 8.3, si le joueur blanc souhaite maximiser son heuristique, il va jouer en $(1,2)$ ou $(3,2)$.

8.3 Algorithme min-max

8.3.1 Choix du coup à l'aide de l'heuristique

Lorsque l'un des joueurs doit jouer, il a plusieurs options possibles (dépendant de la configuration courante au Reversi). Une méthode simple pour décider du coup à jouer consiste à évaluer chaque configuration accessible en utilisant une heuristique, puis à jouer celle qui a la valeur d'heuristique la plus élevée (pour Adam) ou la plus faible (pour Eve).

Remarque : dans la terminologie usuelle de l'algorithme min-max, on dit qu'Adam est le JOUEUR_MAX et Eve le JOUEUR_MIN .

8.3.2 Un exemple au Reversi.

On s'intéresse à une partie de Reversi représentée en figure 8.4.

Il a deux possibilités, représentées dans les figures 8.5 et 8.6.

La prise en compte de la maximisation de l'heuristique «nombre de pions noirs» au rang 1 conduit au graphe 8.7.

Au passage on se constate que l'heuristique invite simplement à suivre une approche gloutonne. Le joueur max va donc avoir tendance à jouer la première possibilité.

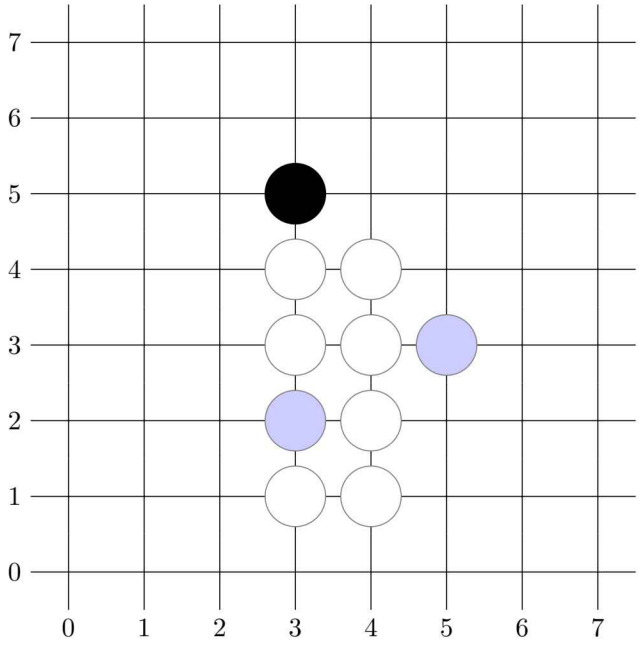

FIGURE 8.4 – Configuration initiale. Adam, le «joueur max» doit placer un pion noir pour se tirer de cette fâcheuse situation.

La prise en compte de la maximisation de l'heuristique «nombre de pions noirs» au rang 1 conduit au graphe 8.8. La situation est cette fois radicalement différente, puisque jouer la première possibilité conduit à une valeur nulle de l'heuristique et donc à la défaite du joueur.

Le joueur max va donc avoir tendance à jouer la deuxième possibilité. On voit donc bien sur cette exemple que la prise en compte de l'heuristique n'amène pas avec certitude à la victoire.

Remarquons que l'on peut répéter ce raisonnement en tenant compte des noeuds de niveaux 3, 4 etc. Mais le nombre de configurations à examiner ayant tendance à croître exponentiellement, il est nécessaire de limiter la profondeur de la recherche. C'est l'intérêt de l'algorithme min-max.

8.3.3 Principe de l'algorithme min-max

Supposons que l'on dispose d'une heuristique et que c'est à Adam de jouer. On se fixe une profondeur maximale d'exploration n pour l'heuristique.

On commence par dresser l'arbre du jeu à partir de la configuration courante en listant toutes les possibilités de parties en n coup. Ensuite, pour déterminer le meilleur coup pour Adam, il est nécessaire de commencer par calculer la valeur heuristique de toutes les configurations atteignables en n coups (qui forment les feuilles de l'arbre que nous venons de partiellement dérouler). Si n est pair, cela signifie qu'Eve a joué le dernier coup, donc pour chaque feuille, le parent aura comme valeur la valeur minimale de ses fils. Inversement, si n est impair, cela signifie qu'Adam a joué le dernier coup, donc pour chaque feuille, le parent aura comme valeur la valeur maximale de ses fils. Ainsi, progressivement, chaque configuration de l'arbre aura une valeur attribuée (voir illustration dans la figure 8.10 et suivantes).

Un pseudo-code est donné dans la figure 8.9.

Si l'on préfère un «pseudo-Python», on peut écrire :

```
import math

def minMax(situation, profondeurMax, joueurActuel):
    if profondeurMax == 0 or situationEstTerminale(situation):
        return heuristique(situation)
    elif joueurActuel == JOUEUR_MAX:
        meilleurScore = -math.inf
        for coup in coupsPossibles(situation):
            score = minMax(situationApresCoup(situation, coup), profondeurMax-1, JOUEUR_MIN)
            meilleurScore = max(meilleurScore, score)
        return meilleurScore
    else:
        meilleurScore = math.inf
        for coup in coupsPossibles(situation):
            score = minMax(situationApresCoup(situation, coup), profondeurMax-1, JOUEUR_MAX)
            meilleurScore = min(meilleurScore, score)
```

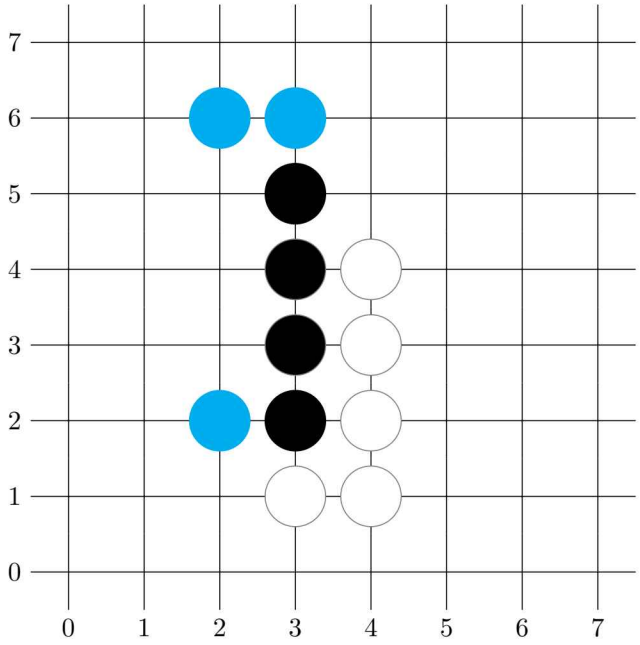

FIGURE 8.5 – Première possibilité pour Adam.

```
        return meilleurScore


#####################################################################
###################### SCRIPT D'EXECUTION ###########################
#####################################################################


situationInitiale = ...
profondeurMax = ...
joueurInitial = ...
coupOptimal = minMax(situationInitiale, profondeurMax, joueurInitial)
```

Un autre exemple.

Supposons que nous disposions d'une heuristique pour jouer à un certain jeu. Dans ce jeu, chaque joueur a trois possibilités à chaque étape. On suppose pouvoir calculer l'heuristique d'une configuration.

Décrivons le résultat de l'algorithme du paragraphe précédent où l'on calcule l'heuristique à profondeur 3. On suppose de plus qu'Adam est le JOUEUR_MAX et que Eve est le JOUEUR_MIN :

Etape 1 : (phase de descente des appels récursifs) on commence par calculer les valeurs de l'heuristique à profondeur 2 de la configuration courante. Cette situation est représentée dans la figure 8.10.

On remonte les appels récursifs désormais.

Etape 2 : on remonte à profondeur 2, comme il s'agit d'un étage d'Adam (JOUEUR_MAX), on va noter dans les noeuds les valeurs maximales des valeurs à profondeur 3. Cette situation est représentée dans la figure 8.11.

Etape 3 : on remonte à profondeur 1, comme il s'agit d'un étage de Eve (JOUEUR_MIN), on va noter dans les noeuds les valeurs minimales des valeurs à profondeur 2. Cette situation est représentée dans la figure 8.12.

Etape 4 : Adam choisit son coup en maximisant la valeur de la configuration suivante.

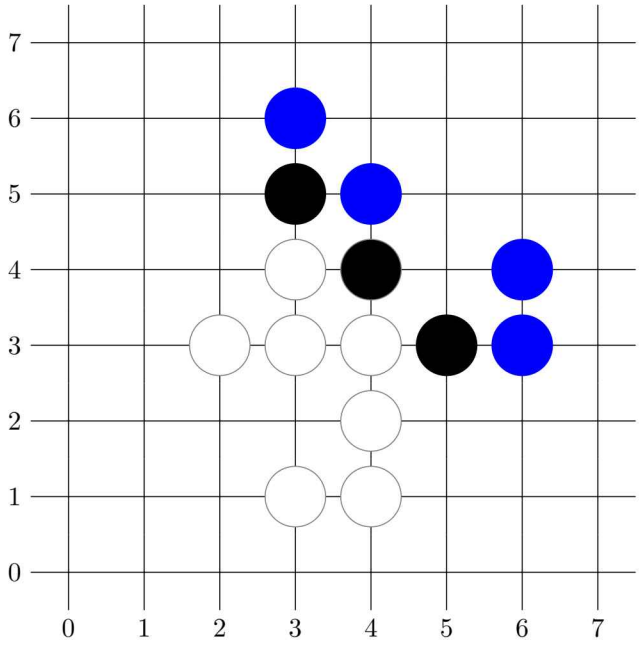

FIGURE 8.6 – Deuxième possibilité pour Adam.

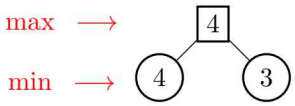

FIGURE 8.7 – Calcul de l'heuristique du premier niveau.

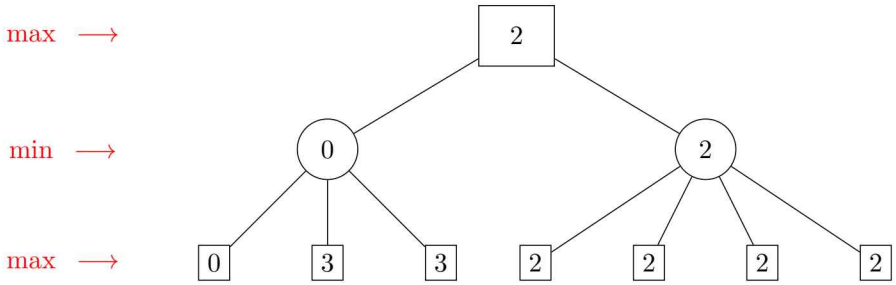

FIGURE 8.8 – Calcul de l'heuristique du deuxième niveau.

Algorithme 1 : Algorithme Min-Max

Entrée *Une configuration initiale configuration, une profondeur maximale profondeurMax et un joueur actuel joueurActuel*;
Sortie *Le coup optimal pour la configuration initiale*;

meilleurScore ← copie de la valeur initiale en fonction de *joueurActuel*;
Si *profondeurMax = 0* **Alors**
  Renvoyer la valeur de l'heuristique correspondant à la configuration
Sinon
  Pour *chaque coup possible dans cette configuration* **Faire**
    Si *joueurActuel* = $JOUEUR\_MAX$ **Alors**
      score ← minMax(configuration après *coup*, $profondeurMax - 1$, $JOUEUR\_MIN$)
    Finsi
    sinon si *joueurActuel* = $JOUEUR\_MIN$ **Alors**
      score ← minMax(configuration après *coup*, $profondeurMax - 1$, $JOUEUR\_MAX$)
    Finsi
    ;
    meilleurScore ← min ou max en fonction de *joueurActuel*(*meilleurScore*, *score*);
  FinPour
  Renvoyer *coup associé à meilleurScore*;
Finsi

FIGURE 8.9 – Un pseudo-code pour l'algorithme min-max.

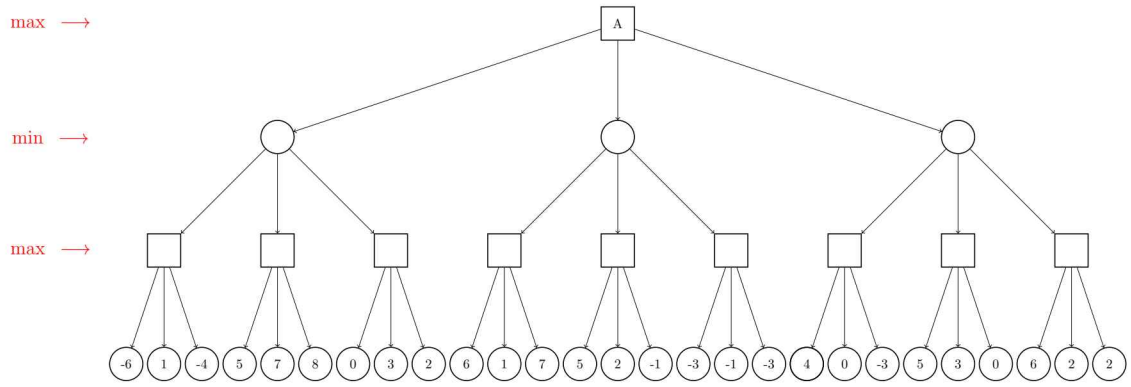

FIGURE 8.10 – Exemple de l'exécution de l'algorithme Min-Max sur un exemple générique. On développe l'arbre du jeu jusqu'à la profondeur 3.

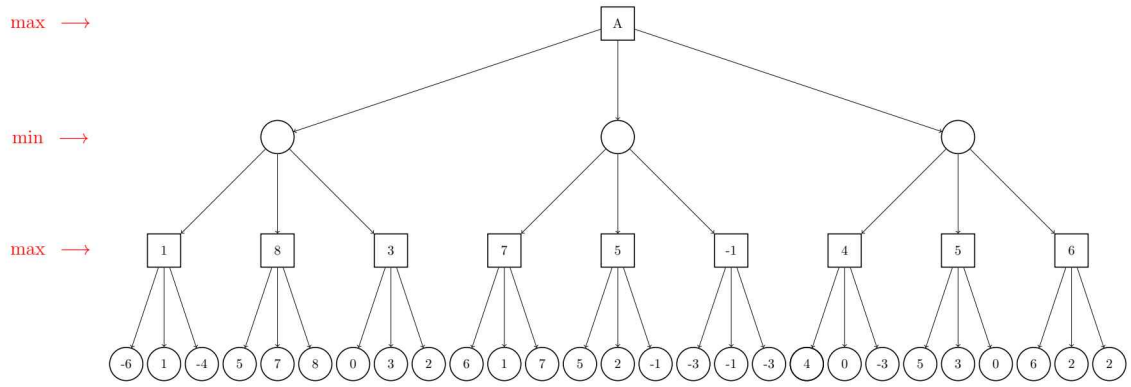

FIGURE 8.11 – Remontée à profondeur 2 des informations lors de l'exécution de l'algorithme Min-Max sur un exemple générique.

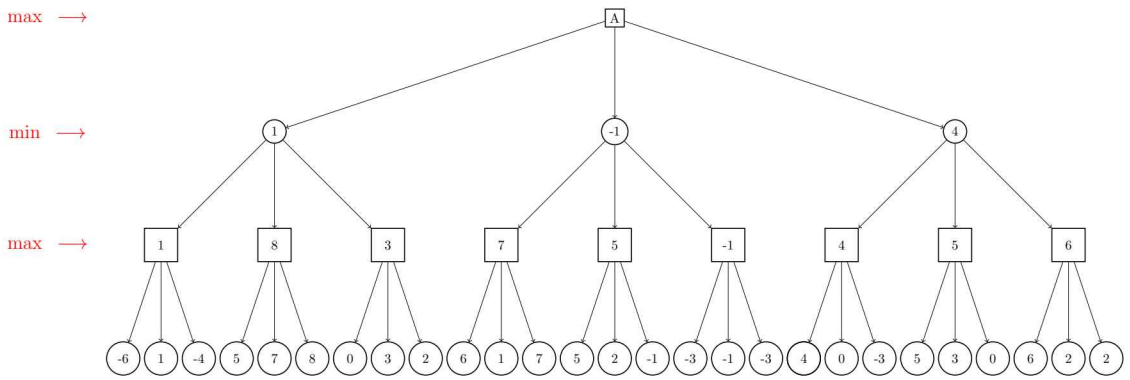

FIGURE 8.12 – Remontée à profondeur 1 des informations lors de l'exécution de l'algorithme Min-Max sur un exemple générique.