

## Jeux

### 1 Jeux à deux joueurs

#### 1.1 Graphe et stratégie

On modélise les jeux par un graphe.

**Définition 1.1.** Un jeu à deux joueurs 0 et 1 est un graphe orienté biparti  $G = (S_0 \cup S_1, A)$  appelé arène. Un sommet  $v$  représente un état du jeu qui est contrôlé par le joueur 0 si  $v \in S_0$  et par 1 sinon.

Un coup possible depuis l'état  $v \in S_p$  contrôlé par le joueur  $p$  est un arc  $v \rightarrow w$  de  $A$ .

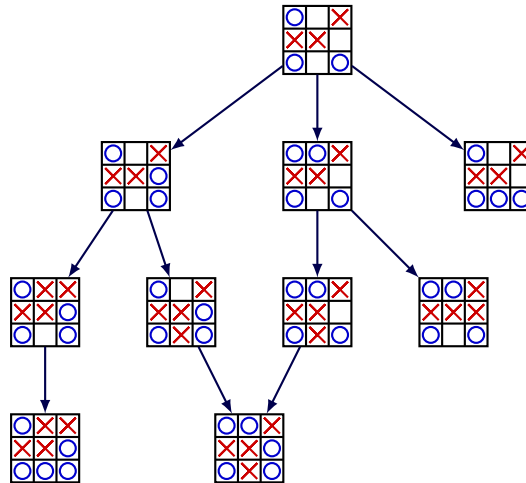
Un jeu possède un état initial  $s_d \in S$ .

Les états sans arc sortant, appelés terminaux, sont divisés en trois sous-ensembles  $T_0, T_1, T_{\text{nul}}$  qui représentent les états gagnants pour 0, pour 1 et les états de match nul. On note  $T = T_0 \cup T_1 \cup T_{\text{nul}}$ .

*Remarque 1.1.* Les états contrôlés par le joueur 0 correspondent aux états pour lesquels c'est au joueur 0 de jouer.

*Remarque 1.2.* Le nombre de configurations est toujours fini dans cette modélisation.

**Exemple 1.1.** Le jeu du tic-tac-toe ou morpion. Dans une grille  $3 \times 3$ , deux joueurs inscrivent, à leur tour, leur symbole (X ou O). Le premier qui en aligne trois a gagné.



**Exercice 1.1.** Dans le jeu Chomp, deux joueurs ont une tablette de chocolat rectangulaire. Chacun leur tour, les joueurs choisissent un carré et mangent tous les carrés situés plus en bas et plus à droite. Le carré en haut à gauche est empoisonné. Celui qui le mange a perdu.

1. Représenter le jeu de Chomp  $3 \times 2$  par un graphe non nécessairement biparti.
2. Comment ce jeu se représente-t-il avec le formalisme ci-dessus ?

**Définition 1.2.** Dans un jeu, une partie (finie) est un chemin de l'état initial à un état final. Une partie infinie est un chemin infini depuis l'état initial.


Une stratégie pour le joueur  $i$  est une application  $f$  de  $S_i \setminus T$  noté  $S'_i$  dans  $S_{1-i}$  telle que pour tout  $v \in S'_i$ ,  $(v, f(v)) \in A$ .

Une partie est jouée selon la stratégie  $f$  par le joueur  $i$ , si pour tout état  $v$  visité pendant la partie, si  $v \in S'_i$ , alors  $i$  a joué le coup menant à  $f(i)$ .

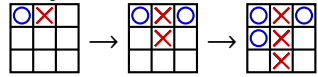
*Remarque 1.3.*

- Le jeu est à information parfaite : les joueurs disposent de toute l'information sur l'état pour choisir leur coup.
- Les stratégies sont sans mémoire, elles ne dépendent que de l'état actuel.
- Si les deux joueurs suivent tous les deux une stratégie, alors il y a une unique partie possible, potentiellement infinie.

**Exemple 1.2.** Une stratégie possible pour le morpion est de toujours jouer dans la case la plus petite pour l'ordre lexicographique sur les lignes et les colonnes.

Si les deux joueurs suivent cette stratégie, la configuration finale est : . Elle est gagnante pour le joueur 0.

**Définition 1.3.** Une stratégie est gagnante pour le joueur  $i$  si toute partie jouée selon cette stratégie pour  $i$  se termine sur un état de  $T_i$  (victoire de  $i$ ).

**Exemple 1.3.** La stratégie ci-dessus n'est gagnante ni pour le joueur 0 ni pour le joueur 1. Voici un exemple de partie non gagnante pour le joueur 0 alors qu'il joue cette stratégie : 

**Exemple 1.4.** Pour le jeu de Chomp  $1 \times 1$ , le joueur 1 a une stratégie gagnante. En fait, n'importe quelle stratégie convient.

Pour le jeu de Chomp  $1 \times n$ ,  $n \geq 2$ , le joueur 1 a une stratégie qui consiste à manger tous les carrés sauf celui qui est empoisonné.

**Exercice 1.2.** On considère le jeu de Cram :

On dispose d'une grille  $n \times p$ . Chacun leur tour (en commençant par le joueur 0), les joueurs placent un domino  $2 \times 1$  sur deux cases adjacentes libres de la grille. Le premier joueur qui ne peut plus jouer a perdu. Montrer que si  $n$  ou  $p$  est pair, un des joueurs a une stratégie gagnante.

**Proposition 1.1.** *Au plus un joueur a une stratégie gagnante.*

## 1.2 Positions gagnantes

**Définition 1.4.** Un sommet  $s$  est une position gagnante pour le joueur  $i$  si ce joueur possède une stratégie gagnante en prenant  $s$  comme sommet initial du jeu.

L'ensemble des positions gagnantes pour un joueur s'appelle un attracteur.

**Exemple 1.5.** Au morpion, la position suivante est gagnante pour le joueur  $\circ$  :



On va maintenant calculer ces attracteurs. On remarque que c'est facile pour les états terminaux.

De plus, si c'est son tour, un joueur a une stratégie gagnante si il a un coup qui l'amène dans une position gagnante.

Si ce n'est pas son tour, il faut que quoi que fasse son adversaire, on arrive dans une position gagnante pour le joueur.

**Définition 1.5.** On définit la suite  $(\mathcal{A}_n(i))_{i \in \{0,1\}}$  par :

- $\mathcal{A}_0(i) = T_i$
- $\mathcal{A}_{n+1}(i) = \mathcal{A}_n(i) \cup \{u \in S'_i \mid \exists v \in \mathcal{A}_n(i), (u, v) \in A\} \cup \{u \in S'_{1-i} \mid \forall (u, v) \in A, v \in \mathcal{A}_n(i)\}$


**Exercice 1.3.** Calculer la suite  $\mathcal{A}_n(i)$  pour chacun des joueurs dans la portion d'arbre de l'exemple 1.1.

**Définition 1.6.** Soit  $S' \subset S$  avec  $s_0 \in S'$ . Une stratégie gagnante sur les sommets de  $S'$  est une fonction qui à chaque état de  $S' \cap S'_i$  associe un coup telle que toute partie jouée selon  $f$  reste dans les états de  $S'$  et est gagnante pour le joueur  $i$ .

**Proposition 1.2.**

- (i) La suite  $\mathcal{A}_n(i)$  est stationnaire à partir d'un certain rang.
- (ii) L'ensemble  $\mathcal{A}(i) = \bigcup_{n \in \mathbb{N}} \mathcal{A}_n(i)$  est alors l'attracteur du joueur  $i$ .

**Exercice 1.4.** Calculer l'arbre des parties possibles et représenter les attracteurs sur cet arbre pour le jeu

de Cram depuis la grille initiale suivante : 

Remarque 1.4. À partir de la suite  $\mathcal{A}_n(i)$ , on peut retrouver la stratégie gagnante du joueur  $i$ .

## 2 Algorithmique des jeux

### 2.1 Min-max

On reformule le calcul des attracteurs sous une forme un peu plus algorithmique.

**Définition 2.1.** On définit inductivement la fonction partielle  $v$  qui à un état  $s$  associe une valeur dans  $\{-1, 0, 1\}$  par :

- si l'état  $s$  est terminal  $v(s) = \begin{cases} 1 & \text{si } s \in T_0 \\ -1 & \text{si } s \in T_1 \\ 0 & \text{si } s \in T_{\text{nul}} \end{cases}$
- sinon si  $v$  est définie sur tous les successeurs de  $s$ , alors :
  - si  $s \in S'_0$ , alors  $v(s) = \max_{(s,t) \in A} v(t)$ .
  - si  $s \in S'_1$ , alors  $v(s) = \min_{(s,t) \in A} v(t)$ .

**Proposition 2.1.** Si l'arène est acyclique, alors  $v$  est bien définie pour tout sommet  $s$ .

**Proposition 2.2.** Si l'arène est acyclique alors, pour un état  $s$ ,  $v(s) = 1$  si et seulement si  $s \in \mathcal{A}(0)$ .

De même,  $v(s) = -1$  si et seulement si  $s \in \mathcal{A}(1)$ .

**Théorème 2.3.** Si l'arène est acyclique et que tous les états terminaux sont gagnants pour l'un des deux joueurs, alors, pour tout état, un des deux joueurs a une stratégie gagnante.

```
(* Les joueurs sont le joueur 0 et le joueur 1 *)
type etat_ttt = char array array
(* On utilise les caractères ' ' '0' et 'X', respectivement pour *)
(* la case vide, le joueur 0 et le joueur 1 *)

(* On dispose d'une fonction coup qui renvoie la liste des coups possibles *)
(* On dispose d'une fonction gagnant prenant un état et renvoie le joueur
dont c'est un état terminal et -1 sinon *)

let rec min_max jr etat =
  match coups jr etat with
  | [] -> let g = gagnant etat in
    if g = 1 then -1 else g+1
  | l -> let f, deb = if jr = 0 then max, ref (-1) else min, ref 1 in
    let h successeur =
      let v = min_max (1-jr) successeur in
      deb := f !deb v
    in
    List.iter h l;
    !deb

let etat_initial = Array.make_matrix 3 3 ' '
let etat_intermediaire_0 = [| [| ' ' ; ' ' ; '0' |];
                             [| ' ' ; 'X' ; ' ' |];
                             [| '0' ; ' ' ; 'X' |] |]

let etat_intermediaire_1 = [| [| ' ' ; ' ' ; ' ' |];
                             [| '0' ; ' ' ; '0' |];
                             [| 'X' ; ' ' ; ' ' |] |]

let etat_final = [| [| ' ' ; '0' ; 'X' |];
                    [| '0' ; 'X' ; '0' |];
                    [| 'X' ; ' ' ; ' ' |] |]
```

**Exercice 2.1.** On voudrait ne pas appeler plusieurs fois la fonction sur une même grille.

1. Quelle technique va-t-on utiliser?
2. Compléter le code à trous sur la feuille annexe en utilisant cette technique.

## 2.2 Heuristiques et élagages

Le nombre d'états de la plupart des jeux est beaucoup trop élevé pour une exploration exhaustive.

On souhaite donc explorer seulement une partie de l'arbre, mais pour ça il faut pouvoir évaluer des états non terminaux.

**Définition 2.2.** Une heuristique est une fonction associant à chaque état  $s$  une valeur dans un intervalle  $[-v_{\max}, v_{\max}]$  telle que :

- si la valeur est  $-v_{\max}$ , alors  $s \in T_1$  ;
- si la valeur est  $v_{\max}$ , alors  $s \in T_0$  ;
- sinon, la valeur est dans  $] -v_{\max}, v_{\max}[$ , plus elle est grande, plus la position est a priori avantageuse pour le joueur 1.

En général, un état nul a la valeur 0.

*Remarque 2.1.* On pourrait prendre  $v_{\max} = 1$  comme précédemment mais pour avoir une heuristique entière, on prendra  $v_{\max} = 100$  dans la suite.

**Exemple 2.1.** Aux échecs, on peut attribuer une valeur à chaque pièce ( $\text{♙}$  : 1,  $\text{♘}$ /  $\text{♞}$  : 3,  $\text{♚}$  : 5,  $\text{♔}$  : 9) et prendre comme heuristique la somme des valeurs des pièces du joueur 0 moins celles du joueur 1.

L'algorithme consiste alors à effectuer le min-max précédent mais on s'arrête à la profondeur  $d$  et on renvoie l'heuristique.

**Exercice 2.2.** Compléter le code sur la feuille en annexe.

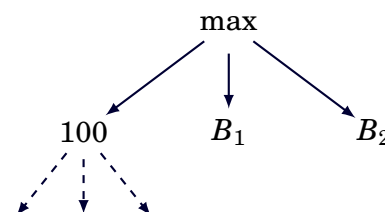
*Remarque 2.2.* La mémorisation peut encore une fois s'avérer utile si on regarde à une profondeur importante.

On aimerait pouvoir élaguer l'arbre de recherche : c'est à dire ne pas explorer les branches qui ne changeront pas le résultat final de l'algorithme.

**Exemple 2.2.** On a obtenu le début d'arbre suivant :

Faut-il explorer les branches  $B_1$  et  $B_2$  ?

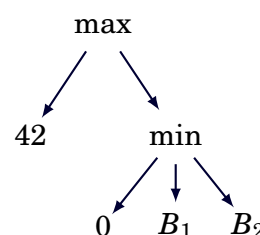
Non, car le résultat sera de toutes façons 100.



On veut utiliser cet élagage avec l'heuristique donc on va généraliser.

**Exemple 2.3.** Pas besoin d'explorer les branches  $B_1$  et  $B_2$  car le résultat du min sera inférieur à 0 et donc sera ignoré par le max.

Cela se généralise à n'importe quelle profondeur.



**Proposition 2.4** (admise). Supposons qu'on ait un nœud max dont une des branches vaut  $\alpha$ . Supposons aussi que dans une des autres branches de ce nœud max, à une profondeur quelconque, on ait un nœud min avec une valeur  $x$  telle que  $x \leq \alpha$ . Alors, la valeur des autres branches de ce nœud min ne change rien à la valeur de retour du nœud max.

On a une propriété similaire pour les nœuds min dont on connaît déjà une valeur  $\beta$ .

L'élagage  $\alpha$ - $\beta$  consiste alors à réaliser cet élagage avec le plus petit  $\alpha$  parmi ses ancêtres pour les nœuds min et le plus grand  $\beta$  parmi ses ancêtres pour les nœuds max.

**Exercice 2.3.** Calculer la valeur du nœud racine (et le coup à jouer) en considérant les enfants dans l'ordre (de gauche à droite) et en essayant de limiter au maximum le nombre de nœuds évalués.

*Remarque 2.3.* Appliquer la mémorisation avec l'élagage  $\alpha$  -  $\beta$  n'a pas l'air intéressant, a priori.