Théorie des jeux

Objectifs

Algorithme min-max avec heuristique et calcul des attracteurs. (sujet d'après L. Lang)

Introduction

Jeu de Nim (ou « jeu des allumettes ») : on dispose de $n \in N^\star$ allumettes disposées sur une table. Deux joueurs J_0 et J_1 jouent à tour de rôle, et J_0 commence. Chaque joueur, quand c'est son tour, prend entre 1 et 3 allumettes. Le joueur qui prend la dernière allumette a perdu.

Nous allons écrire un programme qui permet à un humain de jouer contre l'ordinateur. Nous utiliserons dans un premier temps l'algorithme min-max avec heuristique. Dans un deuxième temps, nous fabriquerons le graphe associé au jeu, et procéderons au calcul des attracteurs pour déterminer une stratégie gagnante.

Modélisation du jeu

Une configuration du jeu est un couple $c = (k, i)$ où $k \in [0, n]$ représente le nombre d'allumettes présentes sur la table et $i \in [0, 1]$ est l'indice du joueur dont c'est le tour de jouer.

1. Écrire une fonction `affiche_configuration(c)` qui prend en argument une configuration et affiche cette configuration à l'écran. Par exemple, l'annexe 1 présente ce qu'on pourra afficher si $c = (7, 0)$.

2. Écrire une fonction `liste_coups_suivants(c)` qui renvoie la liste des configurations possibles au coup suivant. Si c'est une configuration finale, on renverra une liste vide. Par exemple `liste_coups_suivants((3,0))` renverra la liste `[(2, 1), (1, 1), (0, 1)]`. L'ordre des éléments dans la liste renvoyée n'a pas d'importance.

Algorithme min-max avec heuristique

3. Écrire une fonction `h(c)` qui renvoie une évaluation de la configuration `c` calculée comme suit : si `c` est une configuration finale, `h(c)` vaut `1` si la configuration finale est gagnante pour J_0 et `-1` si elle est gagnante pour J_1 ; dans les autres cas, `h(c)` vaut `0`.

4. Écrire une fonction `minmax(c,h,p)` qui renvoie une évaluation du score de la configuration `c` par algorithme min-max avec profondeur `p`, en utilisant l'heuristique `h`.

5. Nous souhaitons maintenant écrire une fonction qui, étant donné une configuration, calculera le meilleur coup à jouer.

5.1. Écrire une fonction `XminimisantY(X,Y)` qui prend en arguments deux listes non vides `X` et `Y` de même longueur, et renvoie un élément `X[i]` tel que `Y[i]` soit un plus petit élément de `Y`, c'est-à-dire tel que $Y[i] = \min(Y[j] j \in [0, len(Y) - 1])$.

5.2. Écrire de même une fonction `XmaximisantY(X,Y)` qui prend en arguments deux listes non vides `X` et `Y` de même longueur, et renvoie un élément `X[i]` tel que `Y[i]` soit un plus grand élément de `Y`.

5.3. Écrire enfin une fonction `coup_suivant_par_minmax(c,h,p)` qui, pour une configuration non finale `c` donnée, renvoie le meilleur coup suivant au regard du score min-max (estimé avec l'heuristique `h` à une profondeur `p`).

6. Écrire une fonction `joue(n,h,p)` qui prend en argument un entier naturel non nul `n` représentant le nombre initial d'allumettes, une heuristique `h` et une profondeur `p`, et fait jouer l'humain contre l'ordinateur, l'humain commençant et l'ordinateur jouant en utilisant le meilleur coup calculé par algorithme min-max avec profondeur `p` et l'heuristique `h`. La fonction informera, à chaque tour, du nombre d'allumettes restantes, et interrogera tous les deux tours l'humain sur le nombre d'allumettes qu'il veut prendre. On pourra aussi indiquer, quand c'est à l'humain de jouer, quel est le meilleur coup qu'il ait à jouer. À titre d'exemple, on donne en annexe 2 ce à quoi pourra ressembler une partie.

7. Faire quelques parties contre l'ordinateur pour $n = 21$, en modifiant la valeur de `p`, afin d'évaluer la qualité du jeu de l'ordinateur. Quelle valeur faut-il donner à `p` pour que l'ordinateur soit imbattable ? Note : pour la valeur de `n` proposée, l'ordinateur a une stratégie gagnante (c'est-à-dire que s'il calcule ses coups avec une profondeur maximale, il gagne nécessairement). Tester pour $n = 9$ et tracer l'arbre correspondant. Quelles sont les positions gagnantes ?

Remarque : on peut démontrer que si $n \equiv 1[4]$, le second joueur a une stratégie gagnante et si cette condition n'est pas vérifiée, c'est le premier joueur qui a une stratégie gagnante. Pouvez-vous deviner quelle est la stratégie gagnante ?

Graphe du jeu et calcul des attracteurs

Nous représenterons le graphe du jeu des allumettes sous forme d'un dictionnaire dont les clés sont les configurations du jeu, et la valeur associée à une configuration est la liste des configurations atteignables au coup suivant. Par exemple le dictionnaire associé au jeu avec $n = 4$ allumettes est donné en annexe 3.

Pour fabriquer ce dictionnaire, une possibilité est de procéder comme pour un parcours en largeur d'un graphe : on met dans une file la configuration initiale `(n,0)`, puis tant que la file n'est pas vide, on défile son plus vieil élément, on le met dans le dictionnaire (s'il n'y est pas déjà) en lui associant tous les coups suivants possibles, et on enfile ces mêmes coups suivants dans la file.

8. Écrire une fonction `GA(n)` qui renvoie le dictionnaire du graphe associé au jeu des allumettes avec `n` allumettes initiales. Vous pourrez tester votre fonction avec l'exemple mentionné ci-dessus.

9. Écrire une fonction `attracteur(G,S0,W0)` qui prend en arguments un graphe représenté par son dictionnaire `G`, la liste `S0` des sommets contrôlés par J_0 et la liste `W0` des sommets finaux gagnants pour J_0, et qui renvoie la liste des sommets du graphe qui sont dans l'attracteur pour le joueur J_0. Vérifier pour $n = 8$ que l'attracteur pour J_0 vaut la liste donnée en annexe 4 (l'ordre des éléments n'a pas d'importance).

10. En modifiant la fonction `attracteur`, écrire une fonction `strategie_gagnante(G,S0,W0)` qui renvoie un dictionnaire indiquant, pour chaque configuration non finale contrôlée par J_0, le prochain coup à jouer qui amène en une position gagnante. On vérifiera que pour $n = 8$, les stratégies gagnantes pour J_0 et J_1 sont données respectivement par l'annexe 5.

Annexes

Annexe 1 - 1. - Exemple c=(7,0)

```
>>> C'est au joueur J0 de jouer, il y a 7 allumette(s) sur la table.
```

Annexe 2 - 6. - Exemple de partie

```
>>> Vous êtes le joueur J0.
>>> C'est au joueur J0 de jouer, il y a 8 allumette(s) sur la table.
>>> Votre meilleur coup à jouer est de prendre 3 allumette(s).
>>> Combien d'allumette(s) souhaitez-vous enlever ? 3
>>> C'est au joueur J1 de jouer, il y a 5 allumette(s) sur la table.
>>> Le joueur J1 enlève 1 allumette(s).
>>> C'est au joueur J0 de jouer, il y a 4 allumette(s) sur la table.
>>> Votre meilleur coup à jouer est de prendre 3 allumette(s).
>>> Combien d'allumette(s) souhaitez-vous enlever ? 3
>>> C'est au joueur J1 de jouer, il y a 1 allumette(s) sur la table.
>>> Le joueur J1 enlève 1 allumette(s).
>>> Le joueur J0 a gagné.
```

Annexe 3 - Exemple n=4

```
{(4,0):[(3,1),(2,1),(1,1)], (3,1):[(2,0),(1,0),(0,0)], (2,1):[(1,0),(0,0)],
 ↪  (1,1):[(0,0)], (2,0):[(1,1),(0,1)], (1,0):[(0,1)], (0,0):[], (0,1):[]}
```

Annexe 4 - 9. - Vérification attracteur J0 n=8

```
>>> [(0,0),(1,1),(2,0),(3,0),(4,0),(5,1),(6,0),(8,0)]
```

Annexe 5 - 10. - Vérification stratégies gagnantes J0 et J1

```
>>> (4, 0): (1, 1), (3, 0): (1, 1), (8, 0): (5, 1), (6, 0): (5, 1), (2, 0):
 ↪  (1, 1) # strategie gagnante J0
>>> (4, 1): (1, 0), (3, 1): (1, 0), (7, 1): (5, 0), (6, 1): (5, 0), (2, 1):
 ↪  (1, 0) # strategie gagnante J1
```

Bon courage et bon travail ! ☺