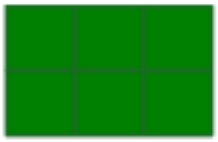
TP n=°6 : Introduction à la théorie des jeux

Informatique tronc commun 2ère année

1 Jeu du chocolat empoisonné



Dans ce jeu à deux joueurs, on joue avec un autre joueur à l'aide d'une tablette de chocolat en carré. La tablette de départ est rectangulaire, possède une largeur de l et une hauteur de h. À tour de rôle, les joueurs choisissent un carré de chocolat encore présent sur la tablette, et mangent tous les carrés qui sont encore présents au-dessus et à droite de ce carré. On va noter appelé ce jeu chocolat h*l.

À droite est représentée une situation de départ dans le cas où la largeur est 3 et la hauteur est 2. Le premier joueur a décidé de manger le carré sur la ligne du haut dans la deuxième colonne, et a donc mangé le carré situé dans la ligne du haut troisième colonne.

Le joueur 2 a ensuite répliqué en mangeant le carré situé sur la ligne du bas deuxième colonne.

1.1 Exemple

1) Dessiner le graphe du jeu 3*2 en faisant bien attention de séparer les sommets où le joueur 1 joue et les sommets où le joueur 2 joue. (Mettre les sommets où le joueur 1 joue sur une colonne et les sommets où le joueur 2 joue sur une deuxième colonne.

2) Annoter d'une croix la/les position(s) finale(s) gagnante(s) pour le joueur 1 et d'un rond la/les position(s) finale(s) gagnante(s) pour le joueur 2.

Voici l'algorithme de calcul des positions gagnantes vu en cours :

Entrées : Un jeu $(S, A), J_1, J_2, F_1, F_2, F_0$, et un sommet $s \in S$
Sorties : True si s est une position gagnante pour le joueur 1 False sinon
Début :
- Si $s \in F1$ Renvoyer True, si $s \in F_2 \cup F_0$ Renvoyer False
- Sinon Si $s \in J_1$, Pour tous a tel que $(s, a) \in A$, Faire :
  - Regarder récursivement si a est gagnant, si oui Renvoyer True

  A la fin de la boucle, renvoyer False
- Si $s \in J_2$, Pour tous a tel que $(s, a) \in A$, Faire :
  - Regarder récursivement si a est gagnant, si non Renvoyer False

  A la fin de la boucle, renvoyer True

3) À l'aide de l'algorithme ci-dessus, déterminer les positions gagnantes et perdantes pour le joueur 1.

1.2 Programmation informatique

On va coder ce jeu informatique. Pour cela, on va noter une position de la tablette par un tuple d'entier. L'entier en position i du tuple contiendra le nombre de carré qui se trouve encore sur la tablette sur la ligne i. Par exemple, la position de départ donnée en exemple peut se noté par (3,3), la position suivante par (3,1) et celle d'après par (2,1).

4) Dessiner les états de tablettes représentés par les tuples (5,2,1) et (4,5,2,1). Laquelle de ces positions n'est une position possible de notre jeu? Pourquoi? Est-ce que le tuple (2,-1) est un état de notre jeu?

5) Écrire une fonction **etat_possible(t : tuple) -> bool** qui prend en entrée un tuple représentant l'état d'une tablette et renvoie True si l'état de cette tablette est atteignable et False sinon.

On va représenter les coups par un couple donnant les coordonnées du coup joué.

6) Écrire une fonction **coup_jouer(t,coup) -> tuple** qui prend en entrée un tuple représentant l'état d'une tablette et un tuple `coup` ayant les coordonnées d'un coup joué.
La fonction renvoie un tuple représentant l'état de la tablette après coup. Si tablette = (3,3,2) et coup = (2,1), la fonction renvoie (3,3,1). Si une ligne est vide, elle sera supprimée du tuple renvoyé.
On pourra utiliser tuple(l) et list(t) pour transformer une liste en tuple et vice-versa.

On souhaite pouvoir renvoyer tous les coups possibles de notre jeu.

7) Écrire une fonction **coup_possible(t : tuple) -> list** qui prend en entrée une position et renvoie une liste de tuple de tous les coups jouables. On ne considère pas (0,0) comme un coup jouable.

Vous pouvez vérifier votre fonction en écrivant ce script :

```
for c in coup_possible((3,3)) :
    print(coup_jouer((3,3),c))
```

8) Écrire une fonction **est_fini(t : tuple) -> bool** qui teste si l'état de la partie représenté est finie.

9) Compléter la fonction **construire_graphe(h; int,l; int) -> dict** qui construit le graphe du jeu du chocolat de hauteur h et de largeur l.

10) Écrire la fonction **est_gagnant(sommet : tuple,g : dict) -> bool** qui prend en entrée un sommet du graphe et un graphe et renvoie True si la position est gagnante pour le joueur 1.

11) Écrire une fonction basée sur la fonction précédente et la mémoisation.

12) Tester plusieurs positions de départ et indiquer s'il faut mieux laisser la main ou prendre la main au jeu du chocolat.

2 Grundy

Le jeu du mathématicien britannique Patrick Mickael Grundy (1917-1959) est un jeu de jetons dont les règles sont les suivantes :
— Au tout début, il n'y a qu'une pile de jetons
— À son tour de jeu, on doit diviser un tas de jetons en deux tas de jetons inégaux.
— S'il n'y a pas de tas à diviser, alors on a perdu.

On remarquera donc qu'une pile d'un ou deux jetons ne peut plus être divisée. À la fin d'une partie, il ne reste que des piles de un ou deux jetons.

13) Montrez qu'il n'existe qu'une partie commençant par un tas de 4 jetons. Comment gagner dans ce cas là ?

On va représenter une partie par une liste $(tas_1, tas_2, ..., tas_k)$., et un booléen PJoueur indiquant si le tour en cours est celui du premier joueur.

14) Quelle est la valeur de cette liste au début d'une partie avec un tas de n jetons ? Lors d'un jeu, quelle est la valeur de la somme éléments de cette liste à chaque instant ? Quelle est la valeur de Pjoueur au premier tour ?

15) Déterminez toutes les parties possibles lorsque le tas de départ contient 5 jetons. En déduire une stratégie gagnante pour le joueur de départ.
En fait, l'état où il y a un tas de taille 5 est appelée une **position gagnante**, car il existe depuis cette position une stratégie gagnante pour le joueur qui joue.

16) Dessinez le graphe du jeu pour 6 jetons. Est-il acyclique ? À quelle propriété « dans le monde réel » peut-on relier l'acyclicité du graphe décrivant un jeu ?
Ouvrez le fichier **Grundy.py**. Pour pouvoir jouer au Grundy, on va créer quelques fonctions permettant de jouer, notamment initialiser une partie et l'afficher sur la console.

17) Écrivez la fonction **initialisation_grundy(n)** qui prend un entier n et renvoie une liste et un booléen représentant la partie de grundy qui commence avec un tas de taille n par le tour du premier joueur.

18) Écrivez une fonction **afficher_partie(partie,joueur)** qui prend en entrée une liste représentant une partie et joueur le booléen indiquant si c'est le tour du premier joueur et affiche dans la console l'état de la partie.

On s'attend au résultat suivant pour afficher_partie([2,4,3,5],True) :

```
Le premier joueur doit jouer
La ligne 0  contient ce nombre de jetons :  2
La ligne 1  contient ce nombre de jetons :  4
La ligne 2  contient ce nombre de jetons :  3
La ligne 3  contient ce nombre de jetons :  5
```

Pour que les joueurs puissent indiquer quel coup ils veulent jouer, on va modéliser les coups par deux valeurs nb_tas et taille_tas1. nb_tas va indiquer l'indice du tas que l'on veut diviser, et taille_tas1 va indiquer la taille du premier tas. La taille du deuxième tas est obtenue par soustraction.

19) Écrivez une fonction **verif_valid(partie,nb_tas,taille_tas1)** qui vérifie si on peut diviser la pile nb_tas en une pile de taille taille_tas1 et une pile partie[nb_tas] - taille_tas1. Si ce n'est pas possible, on renvoie False, sinon, on renvoie True.

Par exemple, verif_valid([2,1,4],2,2) renvoie False car on essaye de diviser le tas de 4 jetons en deux tas de taille 2 et 4 - 2 = 2, donc de taille égale, et verif_valid([3,2,5],2,3) renvoie True.

20) Écrivez une fonction **suivant(partie,joueur,nb_tas,taille_tas)** qui renvoie une liste et un booléen représentant l'état de la partie après que le coup (nb_tas,taile_tas1) soit jouer, sans modifier la liste partie en entrée. On pourra rajouter les deux nouveaux tas à la fin de la liste.

Par exemple, suivant([2,4,3,5],True, 1,3) renvoie [2,3,5,3,1], False et suivant([2,4,3,5],False,3,2) renvoie [2,4,3,2,3], True

Une partie est fini s'il n'y a plus de tas divisible. Donc s'il n'y a plus de tas dont le nombre de jetons est différent de 1 ou 2.

21) Écrivez une fonction **fini(partie)** qui renvoie True s'il n'y a plus de coup possible, et False sinon.

Pour pouvoir jouer, on va mettre en place une intelligence artificielle. Les coups que joueront l'ordinateur seront aléatoire parmi tous les coups possibles.

22) Écrivez une fonction **coup_possible(partie)** qui prend en entrée une liste représentant les tailles des tas, et renvoie une liste des coups possibles jouables sur ces tas.

coup_possible([2,3,4,5]) doit renvoyer [(1, 1), (1, 2), (2, 1), (2, 3), (3, 1), (3, 2), (3, 3), (3, 4)]

Grâce à la fonction coup_possible, on va pouvoir mettre en place notre intelligence artificielle.

23) Écrivez une fonction **intel_art(partie)** qui calcule les coups possible, et en renvoie un au hasard. On pourra utiliser la fonction randrange du module random.

Grâce à la fonction **faire_graphe(n)**, on peut obtenir le graphe du jeu commençant par un tas de taille n, stocké sous forme de listes d'adjacence.

24) Combien y a t il de positions possibles lorsqu'on commence avec n = 10.

25) Dans le cas où la position P que l'on joue est une position finale, comment savoir si la position est gagnante pour le joueur 1 ?

26) Soit une position P, et soit A_p toutes les positions atteignables par P en un coup. Expliquez pourquoi si le joueur 1 doit jouer sur P, alors P est gagnante pour lui s'il existe une position dans A_p et si le joueur 2 doit jouer sur P, alors P est gagnante pour le joueur 1 s'il n'existe aucune position dans A_p perdante pour le joueur 1.

27) Définissez une fonction récursive **pos_gagnante(partie,joueur)** qui détermine si la position est gagnante pour le premier joueur. Trouvez une condition sur la victoire du premier joueur.

La fonction codée précédemment est basé sur l'algorithme min-max.