UTC

IA01

TP2 : *JEU DES ALLUMETTES*

OUEDRAOGO Taoufiq

BRENA-LABEL Yannis

Compte-rendu

SOMMAIRE

Q1. Liste des états possibles pour 11 allumettes

La liste (x J) signifie qu'il reste x allumette et c'est au joueur J de jouer.

Chaque joueur possède 3 possibilités d'actions (cad retirer soit 1, 2 ou 3 allumettes) et pour un nombre d'allumettes donné il existe toujours 2 joueurs qui peuvent agir selon les cas. D'où les 24 états:

((11 J2) (11 J1)

(10 J2) (10 J1)

(9 J2) (9 J1)

(8 J2) (8 J1)

(7 J2) (7 J1)

(6 J2) (6 J1)

(5 J2) (5 J1)

(4 J2) (4 J1)

(3 J2) (3 J1)

(2 J2) (2 J1)

(1 J2) (1 J1)

(0 J2) (0 J1)))

Ici, il n'est pas nécessaire de représenter les 2 états pour lesquels le nombre d'allumettes = 0. Car dans notre cas on pourra considérer que le jeu s'arrête dès que le nombre d'allumettes= 1, 2 ou 3 et donc les états

(0 J1) et (0 J2) ne seront jamais atteints.

Q2. Liste des opérateurs possibles

Un état **(J x)** -> **joueur J retire x allumette**

Pour un état courant, chaque joueur peut enlever jusqu'à 3 allumettes. Les opérateurs possibles sont :

- **(J1 3) -> J1 retire 3 allumettes**
- **(J1 2) -> J1 retire 2 allumettes**
- **(J1 1) -> J1 retire 1 allumettes**
- **(J2 3) -> J2 retire 3 allumettes**
- **(J2 2) -> J2 retire 2 allumettes**
- **(J2 1) -> J2 retire 1 allumettes**

Par exemple, pour l'état **(2 J1),** l'opérateur **(J1 3)** est impossible car il ne reste que 2 allumettes.

-Pour classer ces opérateurs on se servira juste des règles du jeu donc :

Selon la pioche : le joueur ne pourra pas retirer un nombre d'allumettes > nombre d'allumettes sur le plateau. Sachant qu'il est également contraint d'en retirer soit 1,2 ou 3.

Selon le joueur qui a la main : un joueur ne peut jouer que s'il a la main. Donc, le joueur doit être le même pour les différents opérateurs possibles pour un état donné ainsi que pour l'état en question.

Par exemple, pour l'état **(2 J1),** les opérateurs **(J2 3), (J2 2)** et **(J2 1)** sont impossibles car c'est **J1** qui a la main.

Q3. Sachant que J1 commence :

<u>Etat Initial</u> = **(11 J1)** car J1 a la main et il y'a initialement 11 allumettes.

<u>Etats Finaux</u> = **(1 J1)**, **(1 J2), (2 J1), (2 J1), (3 J1)** ou **(3 J1)** en fonction des pioches des 2 joueurs durant la partie. Et sachant que le joueur qui gagne est celui qui retirera la/les dernière/s allumette/s, puisqu'on peut en retirer jusqu'à 3. Et donc celui qui aura la main avec un nombre d'allumettes = 1, 2 ou 3 gagne la partie.

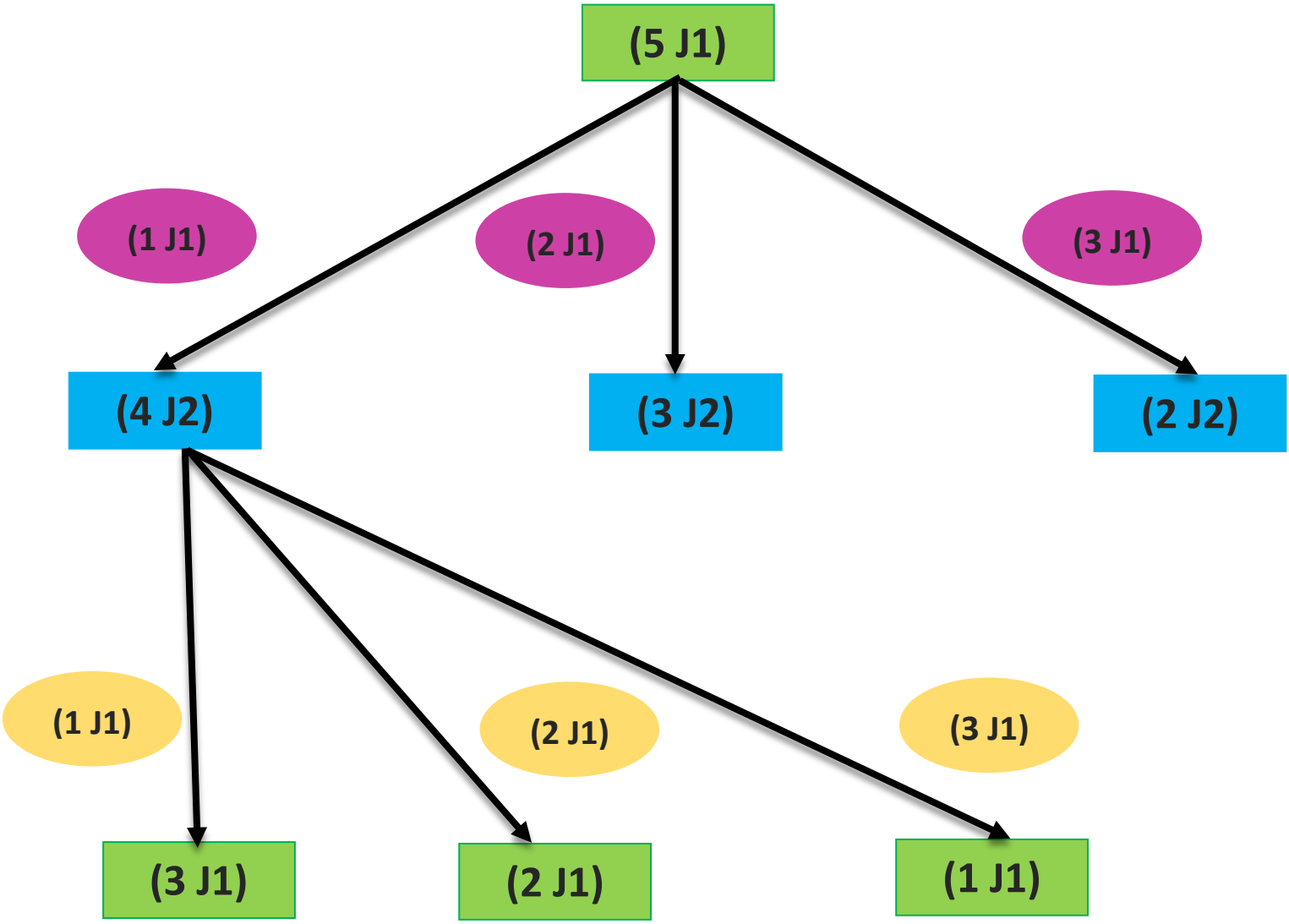
Q4. Arbre de recherche pour que J1 gagne

Arbre de recherche pour dimension initiale=5. On constate ici qu'il y a 3 chemins possibles :

((1 J1) (4 J2) (5 J1))

((2 J1) (4 J2) (5 J1))

((3 J1) (4 J2) (5 J1))



Q5. Fonctions définies

On définit les fonctions :

- **creer_etat(dim J)** retourne la liste de l'état composé de dim et J si dim > 0. Car dans notre modélisation la dimension 0 n'est jamais atteinte.

- **creer_operateur(dim J)** retourne la liste de l'opérateur composé de dim et J.

- **jouer(etat) et dimension(etat)** renvoyant successivement le joueur et la dimension concernée dans etat. (on aurait pu s'en passer …)

- **fin?(etat j)** qui renvoie t j gagne cad etat= (3 J1), (2 J1), (1 J1). Les états où **j** aura la main avec 1, 2 ou 3 allumettes.

- **successeurs(etat)** renvoie la liste des successeurs de etat cad les états possibles après chaque opérateur si la dimension de etat >3 car dans notre modélisation on considère qu'il y a victoire dès qu'il ne reste plus que 3, 2 ou 1 allumette .

- **successeurs_valides(etat etats_visites)** retourne la liste des successeurs valides de etat en considérant les états déjà visités. Sachant que les états valides ont leur dimension < dimension des états parcourus.

- **coup_effectue(etat1 etat2)** qui affiche l'opérateur utilisé pour passer de etat1 -> etat2. Et renvoie un message d'erreur si le nombre d'allumettes retirés est invalide cad > 3 ou < 1 ou si les joueurs sont identiques dans etat1 et etat2.

Q5. Parcours en profondeur

Pour le parcours en profondeur, on développe les états et on ne revient en arrière que s'il y a une impasse.

- explore_profondeur(etat j chemin)

renvoie tous les chemins parcourus menant directement à la victoire de j

On affiche l'état courant

On ajoute l'état courant dans chemin puis on l'affiche

Si à la fin on a trouvé la solution cad un état avec 3, 2 ou 1 allumettes au tour de j, on renvoie le chemin

Sinon

suivants = successeurs valides

Pour chaque successeur valide

On affiche l'opérateur qui amène au successeur

On rappelle la fonction explore_profondeur

Q5. Parcours en largeur

Pour le parcours en largeur, on développe les états d'un même niveau jusqu'à trouver la solution.

- **explore_largeur** renvoie tous les chemins parcourus menant directement à la sortie

```
explore_largeur (etats j chemin)
    pour chaque x dans etats
        ajouter x à chemin
        afficher successeurs de x
        pour chaque successeur y de x
        si y est un etat final pour lequel j gagne
            on ajoute y dans chemin
```

La recherche en largeur étant assez complexe à établir, notre retour n'est pas tout à fait parfait.

Exemple : pour **(explore_largeur '((4 j2)) 'j1 '())** on aura comme solution :

; ///// J1 gagne | Chemin possible = **((3 J1) (4 J2))** /////

; ///// J1 gagne | Chemin possible = **((2 J1) (3 J1) (4 J2))** /////

; ///// J1 gagne | Chemin possible = **((1 J1) (2 J1) (3 J1) (4 J2))** /////

Dans l'exemple, la deuxième solution n'aurait pas dû intégrer **(3 J1)**

Dans l'exemple, la troisième solution n'aurait pas dû intégrer **(3 J1)** et **(2 J1)**