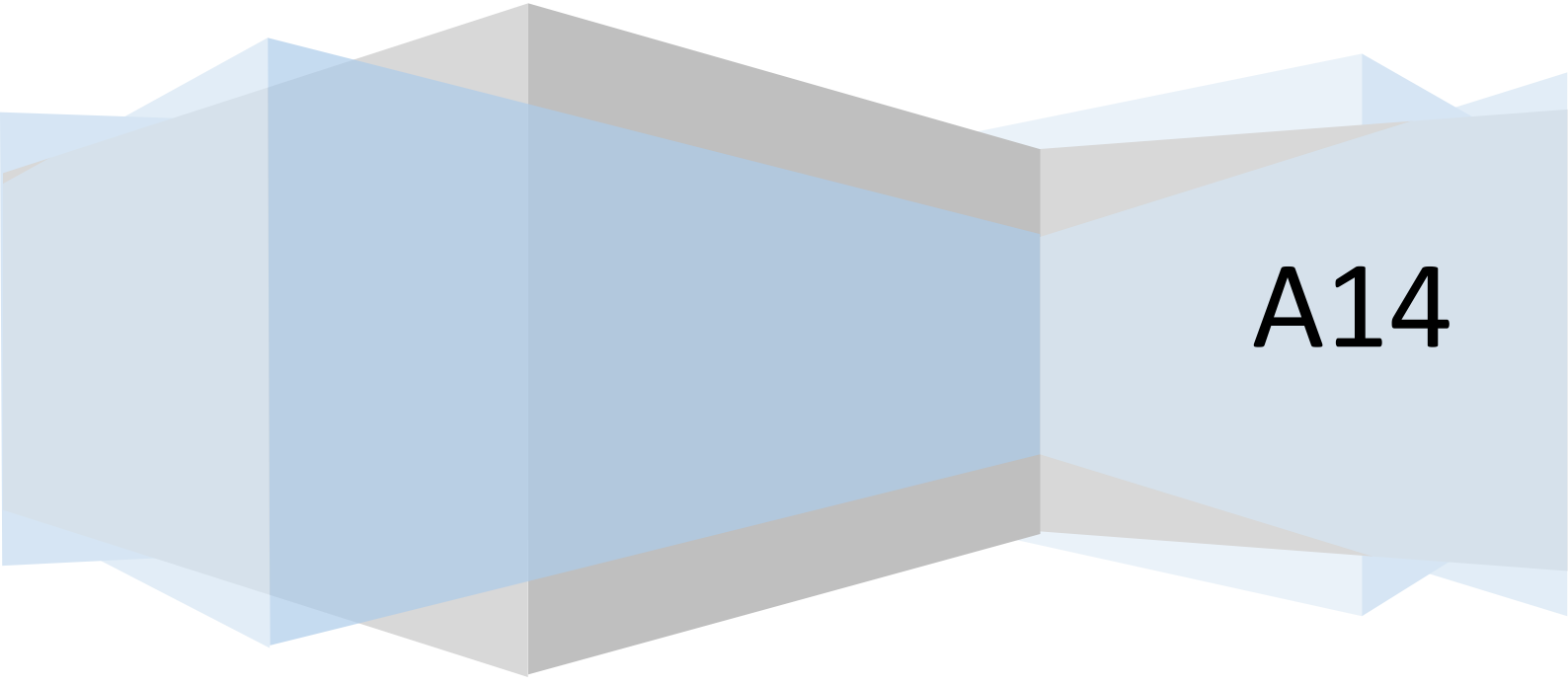
IA01

TP2 : Jeu des allumettes

Alexis Schad - Matthieu Zins



A14

SOMMAIRE

Sommaire 2
1. Liste des états possibles pour une partie de 5 allumettes 3
2. Liste des opérateurs en fonction de l'état courant 3
3. Etat initial et états finaux 3
4. Arbre de recherche 3
5. Résolution du problème 4
   A. Représentation 4
   B. Fonctions de service 4
   C. Parcours en profondeur 5
   D. Parcours en largeur d'abord 8
6. Conclusion 9

1. LISTE DES ETATS POSSIBLES POUR UNE PARTIE DE 5 ALLUMETTES

On représente un état par deux variables, une qui représente le joueur qui a la main (1 ou 2) et une autre qui représente le nombre d'allumettes restantes. Les états possibles sont donc :

(1, 5)	(1, 4)	(1, 3)	(1, 2)	(1, 1)	(1, 0)
(2, 5)	(2, 4)	(2, 3)	(2, 2)	(2, 1)	(2, 0)

2. LISTE DES OPERATEURS EN FONCTION DE L'ETAT COURANT

On représente un opérateur par deux variables. La première représente le joueur qui joue et la seconde représente le nombre d'allumettes qu'il retire.

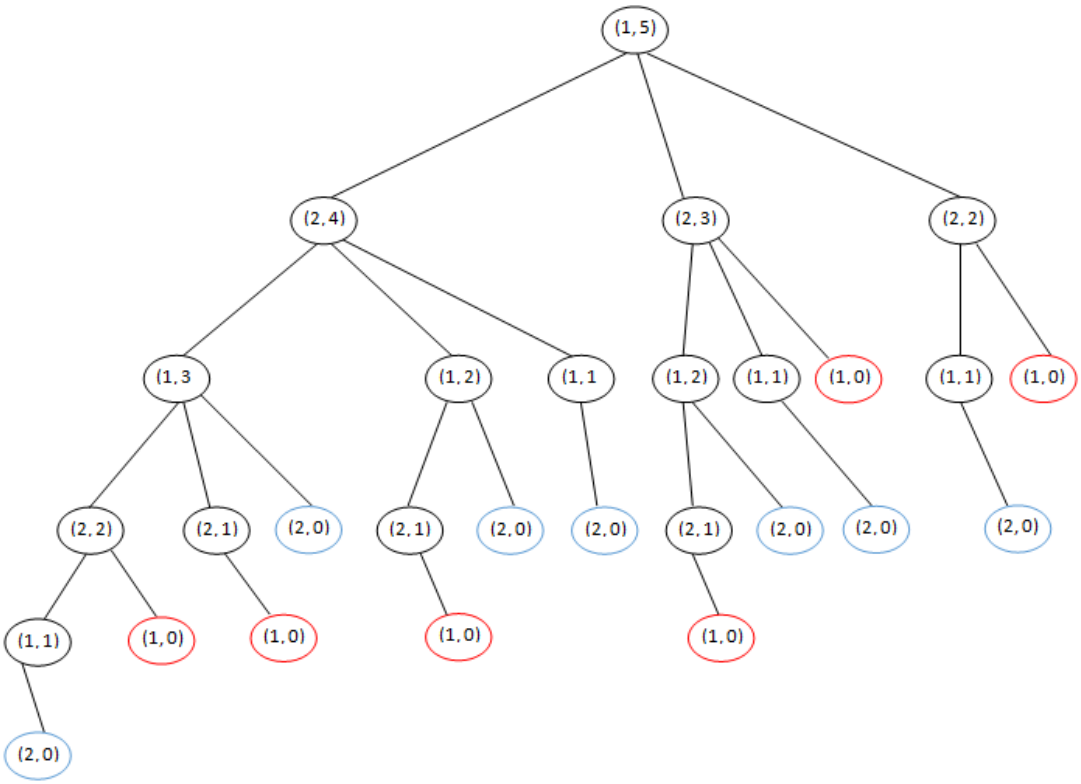
Etat courant	Opérateurs possibles
(1, 5) ou (1, 4) ou (1, 3)	(1, 1) ou (1, 2) ou (1, 3)
(2, 5) ou (2, 4) ou (2, 3)	(2, 1) ou (2, 2) ou (2, 3)
(1, 2)	(1, 1) ou (1, 2)
(2, 2)	(2, 1) ou (2, 2)
(1, 1)	(1, 1)
(2, 1)	(2, 1)
(1, 0) ou (2, 0)	aucun

3. ETAT INITIAL ET ETATS FINAUX

L'état initial est **(1, 5)**.
Les états finaux sont **(1, 0)** et **(2, 0)** en fonction du joueur qui doit gagner.

4. ARBRE DE RECHERCHE

En rouge : les états finaux qui correspondent à une victoire du joueur 2
En bleu : ceux pour une victoire du joueur 1.



5. RESOLUTION DU PROBLEME

A. REPRESENTATION

Nous représenterons un état sous la forme d'une liste :

(joueur_qui_a_la_main allumettes_restantes)

EXEMPLE :

```
(setq etat_intitial '(1 5))
```

Les opérateurs seront également représentés par des listes :

(joueur allumettes_retirées)

Nous avons enregistré l'ensemble des coups possibles dans une liste (liste_coups).

```
(setq liste_coups '((1 1)
                    (1 2)
                    (1 3)
                    (2 1)
                    (2 2)
                    (2 3))
      )
```

B. FONCTIONS DE SERVICE

coups_possibles : Cette fonction renvoie la liste des coups possibles quand on est dans un certain état. Pour cela, on parcourt la liste de tous les coups et lorsqu'un coup est compatible on l'ajoute à "`liste`". Un coup est compatible à un état quand le joueur qui a la main est celui qui fait le coup et que le nombre d'allumettes retirées est inférieur ou égal au nombre d'allumettes restantes.

```
(defun coups_possibles (etat liste_coups)
  (let (liste)
    (dolist (x liste_coups liste)
      (when (and (equal (car x) (car etat)) (>= (cadr etat) (cadr x)))
          (setq liste (append liste (list x)))
        )
      )
    )
  )
```

EXEMPLE : On cherche les coups possibles quand le joueur 1 a la main et qu'il reste 5 allumettes.

```
>(coups_possibles '(1 5) liste_coups)
((1 1) (1 2) (1 3))
```

suivant : Cette fonction renvoie l'état suivant en fonction de l'action effectuée. Si le joueur 1 joue, le joueur 2 aura la main dans l'état suivant et le nombre d'allumettes restantes sera mis à jour.

```
(defun suivant (etat choix)
  (when (and (= (car etat) (car choix)) (<= (cadr choix) 3))
    (let (suiv joueur)
      (if (= (car etat) 1) (setq joueur 2) (setq joueur 1))
      (setq suiv (list joueur (- (cadr etat) (cadr choix))))
      )
    )
  )
```

EXEMPLE : Il reste 5 allumettes, le joueur 1 a la main il en enlève 2 et on veut calculer l'état suivant.

```
>(suivant '(1 5) '(1 2))
(2 3)
```

calc_nb_coups : Cette fonction servira dans l'affichage final pour calculer le nombre de coups effectués par le joueur. Elle prend en paramètre une liste d'actions qui correspondent au déroulement d'une partie et le numéro du joueur dont on veut compter les coups. On parcourt les actions et on incrémente une variable lorsque ce joueur joue.

```
(defun calc_nb_coups (actions joueur)
  (let ((n 0))
    (dolist (x actions n)
      (when (equal joueur (car x))
        (setq n (+ 1 n))
        )
      )
    )
  )
```

EXEMPLE : On veut calculer le nombre de fois que le joueur 1 a joué lors de cette partie

```
>(calc_nb_coups '((1 1) (2 1) (1 1) (2 1) (1 1)) 1)
3
```

afficher : Cette fonction prend en paramètre un chemin (une liste d'états) qui correspond aux états successifs du déroulement d'une partie et des actions (une liste d'opérateur) qui correspondent aux coups joués lors de la partie. On utilise `mapcar` pour afficher le déroulement d'une partie. A la fin, on affiche le joueur qui gagne. Celui-ci correspond au dernier joueur qui a joué, donc au `car` du dernier coup que l'on obtient avec `(car (last actions))`.

```
(defun afficher (chemin actions)
  (mapcar #'(lambda (c a)
              (format t "Le joueur ~S a la main. Il reste ~S allumette(s). ~&" (car c) (cadr c))
              (format t "Le joueur ~S enlève ~S allumette(s). ~&" (car a) (cadr a))
              )
    chemin actions)
  (format t "Le joueur ~S gagne !!~%~%" (caar (last actions)))
  )
```

C. PARCOURS EN PROFONDEUR

explore_profondeur : Cette fonction prend en paramètre l'état, la liste des coups, le joueur que l'on veux faire gagner et deux paramètres qui serviront à enregistrer les chemins et les actions.
Pour ce parcours, il faut rappeler la fonction récursivement. Lors d'un appel, s'il ne reste plus d'allumettes et que le joueur "vainqueur" a la main, on renvoie `nil` puisqu'il aura perdu. Sinon, on enregistre l'état dans `chemin` et on teste si le joueur "vainqueur" a gagné, c'est-à-dire s'il ne reste plus d'allumettes et que l'adversaire a la main. Si c'est le cas, on pourrait afficher le résultat directement en utilisant la fonction `afficher`, mais nous avons plutôt choisi de renvoyer, à la fin la fonction, une liste de toutes les solutions possibles et que l'affichage se fasse dans une autre fonction pour pouvoir compter les solutions et trouver une solution optimale. Il faut donc renvoyer la solution trouvée. Une solution est une liste composée de la liste des états successifs (`chemin`) et de la liste des actions (`actions`). On renvoie `(list (list chemin actions))` parce que l'on utilisera append pour l'ajouter aux autre solutions.
Sinon, on parcourt les coups possibles dans notre état. Pour chaque coup on rappelle la fonction avec en paramètre l'état suivant, la liste de tous les coups, le vainqueur, le chemin et les actions auxquelles on aura ajouté le coup considéré.
Lorsque la fonction renvoie autre chose que nil, on ajoute la ou les solutions trouvées (`temp`) à `res`. Une fois que tous les coups possibles ont été parcourus, res contiendra l'ensemble des solutions trouvées à partir de l'état où l'on se trouve et sera renvoyé.

```
(defun explore_profondeur (etat liste_coups vainqueur chemin actions)
 (let ((res) (temp))
  (if (and (equal (car etat) vainqueur) (= (cadr etat) 0))
      nil
    (progn
       (setq chemin (append chemin (list etat)))
      (if (and (not (equal (car etat) vainqueur)) (= (cadr etat) 0))
          (list (list chemin actions)) ;; on pourrait aussi afficher directement les solutions ici
        (dolist (x (coups_possibles etat liste_coups) res)
          (setq temp (explore_profondeur (suivant etat x) liste_coups vainqueur chemin (append actions
(list x))))
          (when temp
            (setq res (append res temp)
              )
            )
          )
        )
      )
    )
   )
  )
```

EXEMPLE : On obtient une liste de solutions pour une partie à 5 allumettes avec le joueur 1 qui commence. Chaque solution comporte une première liste des états successifs et une seconde des coups qui ont été joués.

```
>(explore_profondeur '(1 5) liste_coups 1 () ())
((((1 5) (2 4) (1 3) (2 2) (1 1) (2 0)) ((1 1) (2 1) (1 1) (2 1) (1 1)))
 (((1 5) (2 4) (1 3) (2 0)) ((1 1) (2 1) (1 3)))
 (((1 5) (2 4) (1 2) (2 0)) ((1 1) (2 2) (1 2)))
 (((1 5) (2 4) (1 1) (2 0)) ((1 1) (2 3) (1 1)))
 (((1 5) (2 3) (1 2) (2 0)) ((1 2) (2 1) (1 2)))
 (((1 5) (2 3) (1 1) (2 0)) ((1 2) (2 2) (1 1)))
 (((1 5) (2 2) (1 1) (2 0)) ((1 3) (2 1) (1 1))))
```

afficher_solutions_profondeur : Cette fonction va utiliser `explore_profondeur` et gérer l'affichage des solutions. Elle prend l'état initial, la liste de tous les coups et le numéro du joueur que l'on veut faire gagner en paramètres. Elle va ensuite appeler `explore_profondeur` avec les mêmes paramètres plus deux listes vides et stocker le résultat dans `sol`. Puis pour chaque solution on l'affiche grâce à notre fonction de service `afficher`. Les paramètres passés à `afficher` sont le chemin `(car x)` et les actions `(cadr x)`. On affiche également le numéro de la solution grâce à `"i"` que l'on incrémente à chaque fois. Cela permettra de savoir combien de déroulements différents permettent de faire gagner le joueur. On affiche également le nombre de coups joués par le joueur que l'on calcule grâce à la fonction `calc_nb_coups`. Ensuite on cherche une solution optimale en parcourant les solutions et cherchant celle dont la taille de la liste des actions est minimale. On l'affiche à la fin.

```
(defun afficher_solutions_profondeur (etat liste_coups vainqueur)
  (let ((i 1)(min)(optimal)(sol))
    (setq sol (explore_profondeur etat liste_coups vainqueur () ()))
    (dolist (x sol)
      (format t "~%Solution ~S en ~S coups : ~&" i (calc_nb_coups (cadr x) vainqueur))
      (afficher (car x) (cadr x))
      (setq i (+ i 1))
      )
    (setq min (length (cadar sol)))
    (setq optimal (car sol))
    (dolist (x sol)
      (when (< (length (cadr x)) min)
        (setq min (length (cadr x)))
        (setq optimal x)
        )
      )
    (format t "~%~%Solution optimale : en ~S coups : ~&" (calc_nb_coups (cadr optimal) vainqueur))
    (afficher (car optimal) (cadr optimal))
    )
  )
```

EXEMPLE : On veut afficher les solutions pour faire gagner le joueur 1 dans une partie à 5 allumettes où celui-ci commence.

```
> (afficher_solutions_profondeur '(1 5) liste_coups 1)
Solution 1 en 3 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 3 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 2 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 2 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 3 allumette(s).
Le joueur 1 enlève 3 allumette(s).
Le joueur 1 gagne !!

Solution 3 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 2 allumette(s).
Le joueur 1 a la main. Il reste 2 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 1 gagne !!

Solution 4 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 3 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 5 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 2 a la main. Il reste 3 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 2 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 1 gagne !!

Solution 6 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 2 a la main. Il reste 3 allumette(s).
Le joueur 2 enlève 2 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 7 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 3 allumette(s).
Le joueur 2 a la main. Il reste 2 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!


Solution optimale : en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 3 allumette(s).
Le joueur 1 enlève 3 allumette(s).
Le joueur 1 gagne !!
```

D. PARCOURS EN LARGEUR D'ABORD

explore_largeur : Cette fonction cherche les solutions et réalise également l'affichage. Pour parcourir l'arbre en largeur d'abord, la fonction n'est pas récursive mais il faut utiliser un file. On utilisera une liste pour la réaliser. Dans cette liste on enfile à la fin grâce à `append` et on défile le premier élément grâce à `pop`.
On utilisera trois files, la première (`file`) pour enregistrer les états atteints, la deuxième (`file_chemins`) pour les chemins parcourus et la troisième (`file_actions`) pour les coups joués.
Au début, on initialise `file` avec une liste contenant l'état initial, et file_chemins avec une liste contenant une liste de l'état initial. Ensuite on boucle jusqu'à ce que la file principale soit vide. A chaque itération on défile les files et on enregistre les éléments défilés dans `temp`, `temp_chemins` et `temp_actions`.
Lorsque l'élément défilé (`temp`) est un état solution (c'est-à-dire si l'adversaire a la main et qu'il ne reste plus d'allumettes) on affiche la solution. Pour l'affichage on réutilise notre fonction de service. Avec ce parcours la première solution trouvée est forcément une solution optimale puisque la recherche se fait par niveau.
Si ce n'est pas un état solution, on enfile tous ses états successeurs dans la file. Pour cela, on parcourt chaque coups possibles, on calcule l'état suivant et on l'enfile. On enfile en parallèle le chemin et la liste des actions qui mènent à cet état. Pour cela on ajoute l'état suivant (`x`) à l'ancien chemin (`temp_chemins`) et on ajoute le coup joué (`a`) à l'ancienne liste des actions (`temp_actions`) et on enfile ces deux nouvelles listes. Avec cet enfilage en parallèle, chaque état de la file a son chemin et la liste des actions dans les files `file_chemins` et `file_actions` au même indice.

Exemple de la gestion des trois files :

	Etat initial	Dans la boucle	Après une itération
`file`	`((1 5))`	On défile (1 5) qui n'est pas un état final, donc on enfile tous ses successeurs.	`((2 4) (2 3)(2 2))`

`file_chemins`	`(((1 5)))`	On défile ((1 5)) et à chaque successeur on enfile le chemin qui a permis de l'atteindre	`(((1 5) (2 4))`
			`((1 5) (2 3))`
			`((1 5) (2 2)))`

`file_actions`	`()`	A chaque successeur on enfile les listes des actions qui ont permis de l'atteindre	`(((1 1)) ((1 2)` `((1 3)))`

```
(defun explore_largeur (etat liste_coups vainqueur)
  (let ((temp)(file)(file_chemins)(temp_chemins)(file_actions)(temp_actions)(i 1)(x))
    (setq file (list etat))
    (setq file_chemins (list (list etat)))
    (while file
      (setq temp (pop file))
      (setq temp_chemins (pop file_chemins))
      (setq temp_actions (pop file_actions))
      (if (and (not (equal vainqueur (car temp))) (= 0 (cadr temp)))
          (progn
            (if (= i 1)
                (format t "~%Solution optimale en ~S coups : ~&" (calc_nb_coups temp_actions
vainqueur))
              (format t "~%Solution ~S en ~S coups : ~&" i (calc_nb_coups temp_actions vainqueur))
              )
            (afficher temp_chemins temp_actions)
            (setq i (+ 1 i))
            )
        (dolist (a (coups_possibles temp liste_coups))
          (setq x (suivant temp a))
          (setq file (append file (list x)))
          (setq file_chemins (append file_chemins (list (append temp_chemins (list x))))))
          (setq file_actions (append file_actions (list (append temp_actions (list a)))))
        )
      )
    )
  )
```

EXEMPLE : On veut afficher les solutions pour faire gagner le joueur 1 dans une partie à 5 allumettes où celui-ci commence.

```
>(explore_largeur '(1 5) liste_coups 1)
Solution 1 (optimale) en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 3 allumette(s).
Le joueur 1 enlève 3 allumette(s).
Le joueur 1 gagne !!

Solution 2 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 2 allumette(s).
Le joueur 1 a la main. Il reste 2 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 1 gagne !!

Solution 3 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 3 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 4 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 2 a la main. Il reste 3 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 2 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 1 gagne !!

Solution 5 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 2 allumette(s).
Le joueur 2 a la main. Il reste 3 allumette(s).
Le joueur 2 enlève 2 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 6 en 2 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 3 allumette(s).
Le joueur 2 a la main. Il reste 2 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!

Solution 7 en 3 coups :
Le joueur 1 a la main. Il reste 5 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 4 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 3 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 2 a la main. Il reste 2 allumette(s).
Le joueur 2 enlève 1 allumette(s).
Le joueur 1 a la main. Il reste 1 allumette(s).
Le joueur 1 enlève 1 allumette(s).
Le joueur 1 gagne !!
```

6. CONCLUSION

Ce TP a permis d'illustrer la représentation d'un problème comme un espace d'états et sa résolution comme une recherche dans cet espace pour trouver des états solutions. Nous avons également pu voir les deux méthodes de parcours (profondeur ou largeur) d'un arbre de recherche.