TP 8 et 9 : projet

1. Le jeu Awalé

Le jeu Awalé se présente sous la forme d'un tablier creusé de deux rangées de six trous, chaque trou contenant initialement quatre graines. Un coup consiste à prendre le contenu d'une case de son camp (la rangée placée de son côté) et à tourner dans le sens trigonométrique le long des deux rangées en distribuant les graines à raison d'une graine par case. Il y a prise quand un coup se termine dans une case du camp adverse dont le contenu est porté à 2 ou 3 graines. Dans ce cas, on récolte le contenu de cette case, ainsi que celui des cases précédentes du camp adverse qui contiennent également 2 ou 3 graines. La partie se termine quand un joueur n'a plus de coup à jouer, ou qu'il n'est plus possible de faire de prises. Le gagnant est celui qui a récolté le plus de graines. Les cases sont numérotées de la gauche du joueur vers sa droite, de 1 à 6 comme ci-dessous.

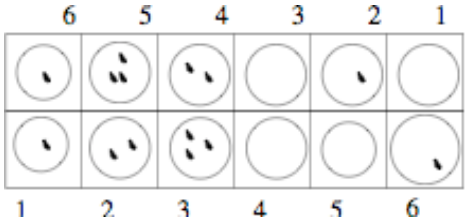

Figure 5 : le jeu Awalé

1. Écrire une fonction `distribue` qui répartit x graines dans des cases données sous la forme d'une liste. Le résultat est un couple comportant le nombre de graines restantes, et la nouvelle liste de cases.

```
(distribue 5 '(2 3 1 5 5 2)) -> (0 (3 4 2 6 6 2))
(distribue 5 '(2 3 1)) -> (2 (3 4 2))
```

2. Écrire la fonction `vide` qui vide une case spécifiée par sa position (numérotée à partir de 1) en distribuant son contenu dans les cases suivantes. La liste des cases est également donnée en paramètre de la fonction. Le résultat est un couple donnant le nombre de graines restantes, et la nouvelle liste de cases. Cette fonction utilisera la fonction `distribue`.

```
(vide 4 '(2 3 1 5 5 2)) -> (3 (2 3 1 0 6 3))
(vide 6 '(2 3 1 5 5 2)) -> (2 (2 3 1 5 5 0))
```

3. Écrire la fonction `prise` qui vide les cases consécutives ayant 2 ou 3 graines. La prise s'arrête à la première case rencontrée dont le nombre de graines est différent de 2 ou 3. Le résultat est un couple donnant le nombre de graines accumulées, et la nouvelle liste de cases.

```
(prise '(2 3 1 5)) -> (5 (0 0 1 5))
(prise '()) -> (0 ())
```

4. Nous souhaitons désormais écrire la fonction `distribue-avec-prise`, analogue à la fonction `distribue`, c'est-à-dire qui distribue x graines dans les cases données, mais qui en plus, retourne en arrière si la dernière case visitée comporte 2 ou 3 graines et qui continue la prise si les cases précédentes ont également 2 ou 3 graines.

Le résultat de cette fonction est un triplet donnant le nombre de graines restantes, les graines accumulées en cas de prise, et la nouvelle liste de cases.

```
(distribue-avec-prise 3 '(2 3 1 5 5 2)) -> (0 2 (3 4 0 5 5 2))
(distribue-avec-prise 3 '(1 1 1 5 5 2)) -> (0 6 (0 0 0 5 5 2))
(distribue-avec-prise 8 '(2 3 1 5 5 2)) -> (2 0 (3 4 2 6 6 3))
(distribue-avec-prise 8 '(1 2 2 1 1 1)) -> (2 0 (2 3 3 2 2 2))
```

Pour écrire cette fonction, écrivez la fonction intermédiaire `distribue-avec-prise-aux` possédant un paramètre supplémentaire qui va servir à construire la nouvelle liste des cases. Cette fonction utilisera la fonction `prise`. Au final, la fonction `distribue-avec-prise` sera définie de la façon suivante :

```
(define distribue-avec-prise ; …
      (lambda (x l) ; …
            (distribue-avec-prise-aux x l '())))
```

REMARQUE : vous aurez peut-être besoin de la fonction Scheme `reverse` qui permet de renverser une liste : `(reverse '(3 2 1))` renvoie la liste (1 2 3).

5. Soient l1 et l2 les rangées de six cases des deux joueurs, et g le gain actuel du joueur courant. Nous voulons désormais écrire la fonction `joue` qui vide la case en position n de l1, distribue les graines dans les cases suivantes de l1, puis dans celles de l2 (avec arrêt et prise s'il y a lieu), puis de nouveau dans celles de l1 s'il reste des graines, etc. Le résultat est un triplet donnant les nouvelles listes l1 et l2 et le nouveau gain g qui devra tenir compte, s'il y a lieu, de la prise effectuée chez l'adversaire.

```
(joue 2 '(4 4 4 4 4 4) '(4 4 4 4 4 4) 0) -> ((4 0 5 5 5 5) (4 4 4 4 4 4) 0)
(joue 6 '(1 3 4 3 2 4) '(4 1 2 2 4 4) 0) -> ((1 3 4 3 2 0) (5 0 0 0 4 4) 8)
(joue 6 '(1 1 4 3 2 8) '(4 1 2 2 4 4) 0) -> ((2 2 4 3 2 0) (5 2 3 3 5 5) 0)
```

La fonction `joue` va ainsi commencer par vider la case n de l1, puis distribuer les graines obtenues dans l1 et ensuite utiliser une autre fonction nommée `tourne`, que vous devez également écrire, pour pouvoir continuer le jeu dans la ligne l2 de l'adversaire, et ainsi de suite jusqu'à ne plus avoir de graines. Ces fonctions utiliseront les fonctions `vide`, `distribue` et `distribue-avec-prise`.

6. Écrire la fonction affiche-jeu qui affiche le jeu comme dans l'exemple ci-dessous.

```
(affiche-jeu "Joueur1" "Joueur2" '(4 4 4 4 4 4) '(4 4 4 4 4 4) 0 0) ->
                        Joueur1=0
                        4 4 4 4 4 4
                        4 4 4 4 4 4
                        Joueur2=0
```

Notez que la fonction `newline` permet de revenir à la ligne, et la fonction `display` permet d'afficher du texte à l'écran (ex : `(display "Joueur1")`, où les guillemets permettent de définir une chaîne de caractères). De plus l'instruction `begin` permet de spécifier une expression comme une suite d'instructions : `(begin instruction-1 … instruction-n)`.

7. Ajouter la fonction `saisie` suivante qui permet de récupérer le coup joué par un des deux joueurs :

```
(define saisie ; recupere et renvoie la valeur saisie
  (lambda (j) ; j chaine de caracteres
    (begin
    (newline)(display "position pour ")(display j)(display " ? ")
    (read))))
```

Cette fonction peut ensuite être utilisée de la façon suivante :

```
(let ((position1 (saisie "Joueur1")) …)
```

8. Écrire la fonction `affiche-gagnant` qui permet d'afficher le joueur gagnant.

```
(affiche-gagnant "Joueur1" "Joueur2" 3 1) -> le gagnant est : Joueur1
(affiche-gagnant "Joueur1" "Joueur2" 1 3) -> le gagnant est : Joueur2
(affiche-gagnant "Joueur1" "Joueur2" 1 1) -> le gagnant est : exaequo
```

9. Écrire la fonction `boucle` qui affiche le jeu, récupère la position jouée par le joueur 1, effectue le coup, puis recommence pour le coup suivant du joueur 2. Cette fonction prendra notamment en paramètres les gains g1 et g2 des deux joueurs. Notez que la touche 'q' doit permettre d'arrêter le jeu et d'afficher le résultat.

```
(boucle "Joueur1" "Joueur2" '(4 4 4 4 4 4) '(4 4 4 4 4 4) 0 0) -> …
```

10. Ajoutez la fonction `awale` suivante qui lance le jeu avec les bons paramètres de départ.

```
(define awale
   (lambda ()
         (boucle "Joueur1" "Joueur2" '(4 4 4 4 4 4) '(4 4 4 4 4 4) 0 0)))
```

Puis testez votre jeu pour vérifier que toutes les fonctions sont bien implantées.

11. BONUS :

a. Analysez les fonctions nécessaires pour que vous puissiez jouer contre l'ordinateur.

b. Implantez ces fonctions.

2. Évaluation du projet

Lors de la dernière séance de TP (semaine du 21 mai), vous devrez faire une démonstration de votre programme à votre encadrant. Il vous demandera notamment d'expliquer les fonctions que vous avez implantées.

Vous lui enverrez également par mail le code source qui doit être commenté, et qui doit comporter des appels pertinents aux différentes fonctions afin de le tester.