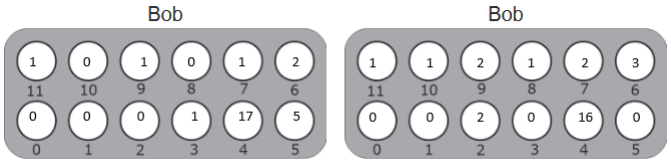
Corrigé CCINP 2024

Informatique tronc commun 2ème année

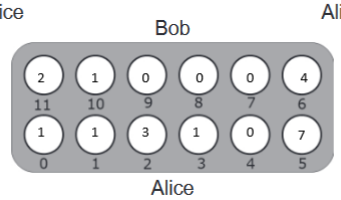
1 Awalé

1) Elle peut jouer dans toutes les cases non-vides.
C'est à dire en 2, 4 et 5.

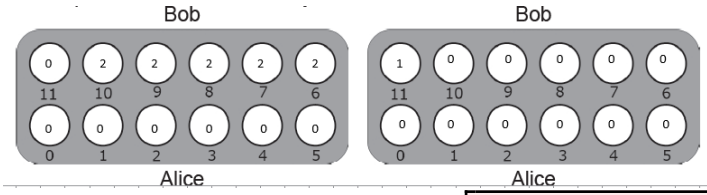
2)



Dans la dernière solution, on a joué la case 4 où il y avait un 16, et on a pu récolter les cases 9, 8 et 7 pour un gain de 3 + 2 + 3 = 8 graines.



3)



Dans la première situation, il n'y a pas de récolte, sinon, Bob n'aurait pas de coup à jouer.

4) Lorsque le joueur 1 doit jouer, n est pair.

```
def tour_joueur1(jeu) :
    return jeu['n']%2 == 0
```

5)

```
def tourner_plateau(jeu) :
    cop = jeu['plateau'][:]
    for i in range (6) :
        cop[i] = jeu['plateau'][i + 6]
    for i in range (6,12) :
        cop[i] = jeu['plateau'][i - 6]
    jeu['plateau'] = cop
```

Faire attention de bien copier la liste du plateau réponse

6) Même si la situation doit être assez rare, on va supposer que la maximum de graine que peut contenir une case est 48 (nombre de graines initiales). On a donc besoin de 6 bits car on peut aller jusqu'à la valeur 63 ($2^6 - 1$) avec 6 bits.

Remarque : Il est sûrement probable de pouvoir coder avec 5 bits par cases ce jeu.

7)

```
def copie(jeu) :
    rep = {}
    rep['joueur1'] = jeu['joueur1']
    rep['joueur2'] = jeu['joueur1']
    rep['score'] = jeu['score'][:]
    rep['n'] = jeu['n']
    rep['plateau'] = jeu['plateau'][:]
    return rep
```

8)
```
def deplacer_graines(plateau,case) :
    enmain, pos = plateau[case], case
    while enmain != 0 :
        pos = (pos + 1)%12
        if pos == case :
            pos = (pos + 1)%12
        plateau[pos] += 1
        enmain = enmain - 1
    plateau[case] = 0
    return pos
```

9)
```
def case_ramassable(plateau,case) :
    condition_en_face = case > 5
    condition_nombre = plateau[case] == 2 or plateau[case] == 3
    return condition_en_face and condition_nombre
```

10)
```
def ramasser_graines(plateau,case) :
    if case == 5 :
        return 0
    else :
        if case_ramassable(plateau,case) :
            rep = plateau[case]
            plateau[case] = 0
            return rep + ramasser_graines(plateau,case-1)
        else :
            return 0
```

11)
```
def test_famine(plateau,case) :
    plateau_test = plateau[:]
    case_r = deplacer_graines(plateau_test,case)
    if case_ramassable(plateau,case_r) :
        ramasser_graines(plateau_test,case_r)
        for j in range (6,12) :
            if plateau_test[j] != 0 :
                return True
        return False
    else :
        return True
```

12)
```
test = case <= 5 and \\
plateau[case] != 0 and \\
condition3
```

13)
```
def cases_possibles(jeu) :
    rep = []
    for i in range (0,6) :
        if test_case(jeu['plateau'],i) :
            rep.append(i)
    return rep
```

14)
```
def tour_suivant(jeu) :
    for s in jeu['score'] :
        if i >= 25 :
            return False
    if jeu['n'] >= 100 :
        return False
    s = 0
    for g in jeu['plateau'] :
        s += g
    if s <=3 :
        return False
    pos = cases_possibles(jeu)
    if len(pos) == 0 :
        return False
    return True
```

15)
```
casef = deplacer_graines(plateau,case) # Instruction 1
graines_gagnees = ramasser_graines(plateau,casef) # Instruction 2

tour_joueur1(jeu) # Condition 1

jeu['n'] = jeu['n'] + 1 # Instruction 3
```

16)
```
def gagnant(jeu:dict) :
    plateau = jeu['plateau']
    for i in range (6) :
        jeu['score'][0+jeu['n']%2] += plateau[i]
    for i in range (6,12) :
        jeu['score'][1+jeu['n']%2] += plateau[i]
    if jeu['score'][0] > jeu['score'][1] :
        return jeu['joueur1']
    elif jeu['score'][0] < jeu['score'][1] :
        return jeu['joueur2']
    else :
            return "égalité"
```

17)
```
def gain(jeu, case) :
    jeu_cop = copie(jeu)
    plateau = jeu_cop['plateau']
    casef = deplacer_graines(plateau,case)
    graines_gagnees = ramasser_graines(plateau,casef)
    if tour_joueur1(jeu_cop) :
        jeu_cop['score'][0] += graines_gagnees
    else :
        jeu_cop['score'][1] += graines_gagnees
    jeu['n'] = jeu['n'] + 1
    tourner_plateau(jeu_cop)
    return graines_gagnees, jeu
```

18)

On va d'abord remplir les gains :
- 1ère ligne, G = 2 et G = 4
- 2ème ligne, G = 4 et G = 0
- 3ligne, G = 0, G = 0 et G = 0

On va maintenant remplir les valeurs de jeu :
- 3ère ligne, VJ = 0, VJ = 0 et VJ = 0 (profondeur 2 atteinte
- 2ème ligne, VJ = 0 et VJ = 0 (profondeur 2 atteinte)
- 1ligne, VJ = 0 et VJ = 4

19)
```
not tour_suivant(jeu) # Condition 1

profondeur == profondeur_max # Condition 2

G, jeu_case = gain(jeu,case) # Instruction 1
NegaAwale(jeu_case,profondeur_max,profondeur + 1) # Instruction 2
```

20)
```
def max_vals(vals_jeu, profondeur) :
    case_max, val_max = vals_jeu[0]
    for case, val in vals_jeu :
        if val_max < val :
            val_max = val
            case_max = case
    if profondeur = 1 :
        return case_max
    else :
        return val_max
```

21) Les lignes devant être modifiées sont les lignes de choix de case, on peut donc supprimer la ligne 6 (la ligne 5 n'est plus nécessaire mais peut être intéressante pour le suivi de la partie).

```
NegaAwale(jeu,6,0) # Ligne 6
```

2 SQL

22)
```
SELECT id_Joueur
FROM Joueur
WHERE niveau > 1900
```

23)
```
SELECT SUM(resultat) * 100 / (SELECT COUNT(*) FROM Partie WHERE jeu LIKE 'a%')
FROM Partie
WHERE resultat <> 0.5 AND jeu LIKE 'a%'
```

24)
```
SELECT nom, prenom
FROM Joueur
ORDER BY niveau DESC LIMIT 3
```

25)
```
SELECT nom, prenom, COUNT(*) as victoire
FROM Partie JOIN
Joueur on id_joueur1 = id_Joueur
WHERE resultat = 1
GROUP BY id_Joueur
HAVING victoire > 100
ORDER BY victoire DESC
```