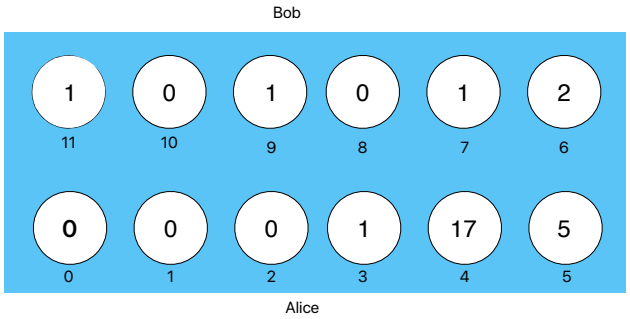
CORRIGÉ DE L'ÉPREUVE ITC CCINP PC 2024

Partie I

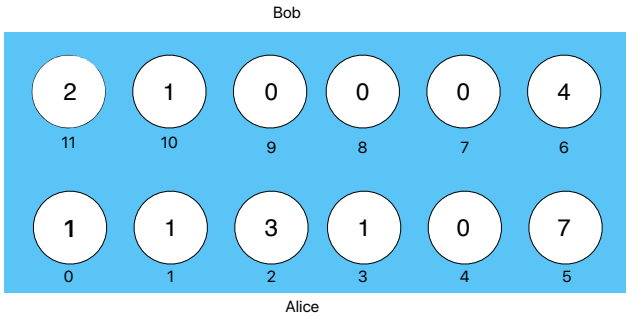
Question 1 – Les cases 0, 1 et 3 sont vides donc Alice ne peut pas les jouer.
Alice peut jouer la case 2 (aucune récolte).
Si Alice joue la case 4, sa dernière semaille s'effectue dans la case 9 et donc Bob ne sera pas affamé (la graine dans la case 11 ne sera pas touchée) ; donc ce coup est valable.
Si Alice joue la case 5, sa dernière semaille s'effectue dans la case 10 et donc Bob ne sera pas affamé (la graine dans la case 11 ne sera pas touchée) ; donc ce coup est valable.

Question 2 –

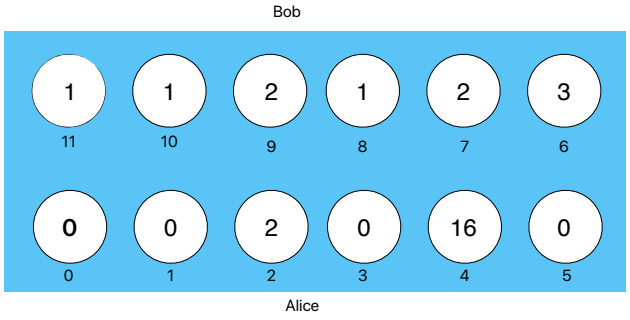
- Si Alice joue la case 2, il n'y a pas de récolte car on n'atteint pas les cases de Bob. Gain : 0.



- Si Alice joue la case 4, la dernière case atteinte est la case 9 et les cases 9, 8 et 7 sont récoltées. Gain : 8.

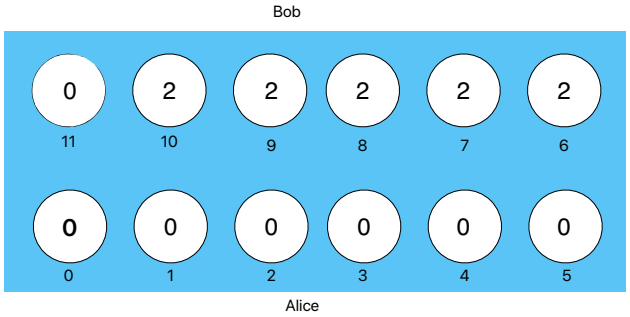


- Si Alice joue la case 5, la dernière case atteinte est la 10, qui ne contient qu'un seul grain, donc il n'y a pas de récolte. Gain : 0.



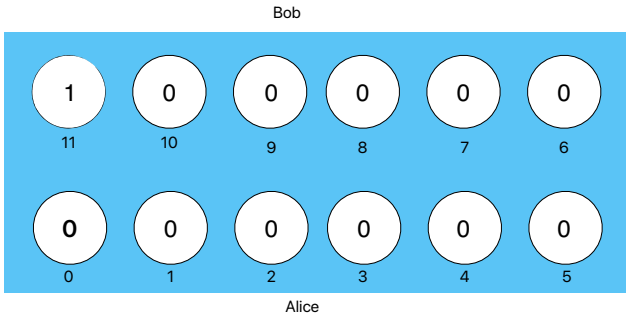
Question 3 –

- Première situation.
  Alice doit jouer la case 5. La dernière graine est semée dans la case 10 et les cases 6 à 10 sont ensuite ramassées, ce qui affamerait son adversaire. La récolte est donc annulée (ce coup peut-il être joué ?).



  Remarque : il y a ici une petite ambiguité dans l'énoncé. Dans une telle situation, est-ce que la partie s'arrête car Alice ne peut pas jouer ou bien est-ce qu'elle peut jouer, mais sans récolter ? Ce point sera clarifié par la suite.

- Deuxième situation.
  Alice doit jouer la case 5. La dernière graine est semée dans la case 10 et les cases 6 à 10 sont ensuite ramassées (gain : 10).



Partie II

Question 4 – `jeu['n']` est pair lorsque c'est au tour du joueur1 de jouer.

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

Question 5 –

```
def tourner_plateau(jeu):
    for case in range(6):
        tmp = jeu['plateau'][case]
        jeu['plateau'][case]  = jeu['plateau'][case+6]
        jeu['plateau'][case+6] = tmp
```

Question 6 – A priori, le nombre maximal de graines dans une case est 48 (Rem : il est probable qu'une étude plus fine du jeu permettre d'affiner ce résultat).
On peut coder les nombres entiers compris entre 0 et 48 sur 6 bits :

$$48 = 32 + 16 = (110000)_2 \ .$$

Question 7 –

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

Question 8 –

```
def deplacer_graines(plateau,case):
        nombre_graines = plateau[case] # nombre de graines restant à semer
        plateau[case] = 0 # on vide case
        numero_case = case # case dans laquelle on va semer
        while nombre_graines > 0:
                numero_case = (numero_case + 1)//12
                if numero_case != case :
                        plateau[numero_case] += 1
                        nombre_graines -= 1
        return numero_case
```

Question 9 –

```
def case_ramassable(plateau,case):
        nb_gr = plateau[case]
        bool1 = (nb_gr == 2) or (nb_gr == 3)
        bool2 = (case > 5) and (case <12)
        return (bool1 and bool2)
```

Question 10 –

```
def ramasser_graines(plateau,case):
        if not case_ramassable(plateau,case):
                return 0
        else :
                nb_graines = plateau[case]
                plateau[case] = 0
                total_graines = nb_graines + ramasser_graines(plateau,case-1)
                return total_graines
```

Question 11 – Remarque : dans cette question, il faut tester si après récolte la partie du plateau de l'adversaire n'est pas vide. Pour cela, on va appliquer les fonctions `deplacer_graines` et `ramasser_graines` au `plateau`. Le problème est que ces fonctions modifient `plateau`. Pour éviter cela, on va se servir d'une copie de `plateau`.

```
def test_famine(plateau,case):
        nouveau_plateau = plateau.copy()
        case_fin = deplacer_graines(nouveau_plateau,case)
        a = ramasser_graines(nouveau_plateau,case_fin)
        for i in range(6,12):
                if nouveau_plateau[i] != 0:
                        return True
        return False
```

Question 12 –

```
def test_case(plateau,case):
        condition3 = test_famine(plateau,case)
        test = condition3 and (plateau[case] != 0) and (case>=0) and (case<6)
        return test
```

Remarque : cette fonction ne modifie pas son entrée `plateau`.

Question 13 –

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

Remarque : comme on a déjà fait attention à ne pas modifier `plateau` dans la fonction `test_famine(plateau,case)` (et donc `test_case(plateau,case)`, cette fonction ne modifie pas `plateau` (et donc pas `jeu`).

Question 14 –

```
def tour_suivant(jeu):
        if jeu['n'] >= 100:
                return False
        a1 , a2 = jeu['score']
        if a1 > 24 or a2 > 24:
                return False
        plateau = jeu['plateau']
        nb_gr = 0
        for case in range(0,12):
                nb_gr += plateau[case]
        if nb_gr <= 3:
                return False
        cases_jouables = cases_possibles(jeu)
        if len(case_jouables) == 0:
                return False
        return True
```

Question 15 –

- instruction 1

```
case_fin = deplacer_graines(plateau,case)
```

- instruction 2

```
graines_gagnees = ramasser_graines(plateau,case_fin)
```

- condition 1

```
tour_joueur1(jeu)
```

- instruction 3

```
jeu['n'] += 1
```

Question 16 –

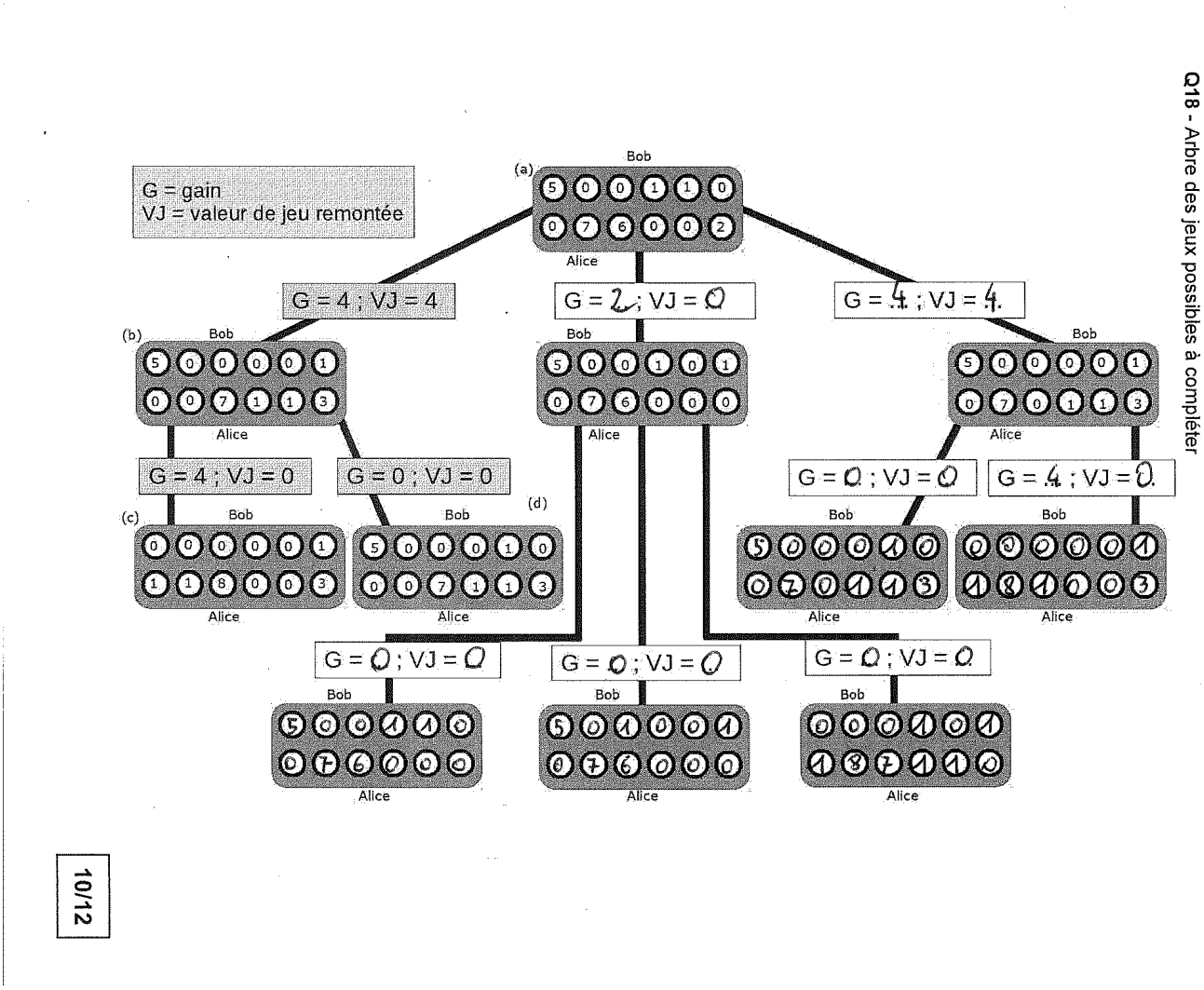
```
def gagnant(jeu):
        plateau = jeu['plateau']
        if tour_joueur1(jeu):
                nb1 = jeu['score'][0]
                for case in range(0,6):
                        nb1 += plateau[case]
                nb2 = jeu['score'][1]
                for case in range(6,12):
                        nb2 += plateau[case]
        else :
                nb2 = jeu['score'][0]
                for case in range(0,6):
                        nb2 += plateau[case]
                nb1 = jeu['score'][1]
                for case in range(6,12):
                        nb1 += plateau[case]
        if nb1 > nb2:
                return jeu['joueur1']
        elif nb1 < nb2:
                return jeu['joueur2']
        else :
                return 'égalité'
```

Partie III

Question 17 –

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

Question 18 –



Pour déterminer le coup que doit jouer Alice, il reste à déterminer la valeur du jeu (VJ) de sa position de départ (en déterminant le max parmi les différences entre son gain et la valeur de jeu pour les fils de sa position) :

$$VJ = max(4-4, 2-0, 4-4) = 2.$$

Et le coup à jouer est donc celui du milieu du schéma (case 5).

Question 19 –

- condition 1

```
not(tour_suivant(jeu))
```

- condition 2

```
profondeur == profondeur_max
```

- instruction 1

```
g , n_jeu    = gain(jeu,case)
```

- instruction 2

```
p = NegaAwale(n_jeu,profondeur_max,profondeur+1)
```

Question 20 –

```
def max_vals(vals_jeu,profondeur):
    case_max , valeur_max = vals_jeu[0]
    for i in range(1,len(vals_jeu)):
        case , valeur = vals_jeu[i]
        if valeur > valeur_max:
            case_max , valeur_max = case  , valeur
    if profondeur == 0:
        return case_max
    else :
        return valeur_max
```

Question 21 – On doit modifier la ligne 6, par exemple de la manière suivante :

```
case_choisie = NegaAwale(jeu,6,0)
```

Question 22 –

```
SELECT id_joueur
FROM Joueur
WHERE niveau > 1900
```

Question 23 – Nombre total de parties ayant commencé par la case 0 :

```
SELECT COUNT(*)
FROM Partie
WHERE jeu LIKE 'a%'
```

Nombre de parties ayant commencé par la case 0 et que le joueur 1 a gégnées :

```
SELECT COUNT(*)
FROM Partie
WHERE jeu LIKE 'a%' AND resultat = 1
```

Pour obtenir le pourcentage :

```
SELECT 100*(SELECT COUNT(*)
       FROM Partie
       WHERE jeu LIKE 'a%' AND resultat = 1 )/(SELECT COUNT(*)
       FROM Partie
       WHERE jeu LIKE 'a%' )
```

Question 24 –

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

Question 25 –

```
SELECT J.nom , J.prenom , COUNT(*)
FROM Joueur AS J JOIN Partie AS P
                 ON J.id_joueur = P.id_joueur1
WHERE P.resultat = 1
GROUP BY J.id_joueur
HAVING COUNT(*) > 100
ORDER BY COUNT(*) DESC
```