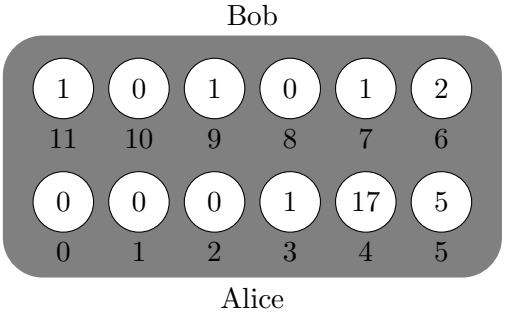
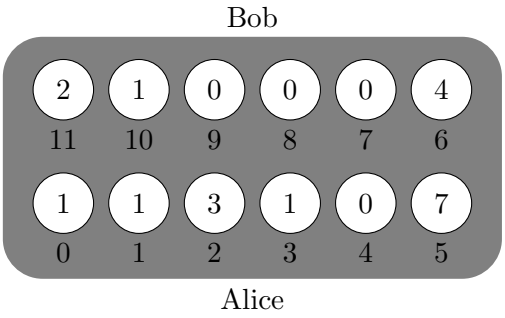
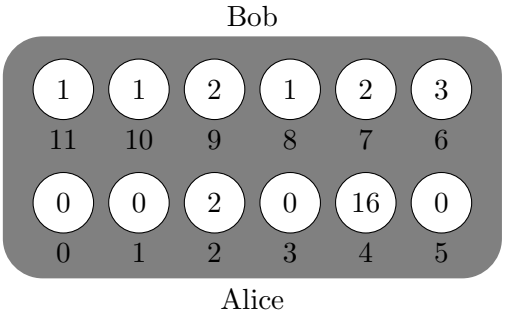
1. Alice peut choisir de jouer toutes les cases non vides de son camp, soit les cases 2, 4 et 5.
2. Dans le premier cas, Alice joue la case 2 et ne peut récolter : son gain est nul. On obtient le plateau suivant.



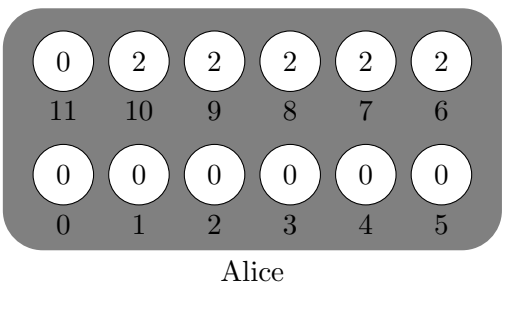
Dans le deuxième cas, Alice joue la case 4 et récolte les cases 7,8,9 : son gain est de 8. On obtient le plateau suivant.



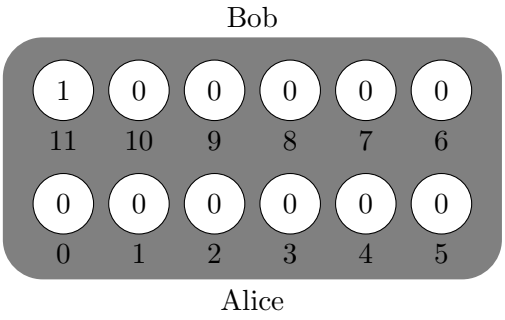
Dans le troisième cas, Alice joue la case 5 et ne peut récolter : son gain est nul. On obtient le plateau suivant.



3. Dans les deux cas, Alice doit jouer la case 5.
   Dans la première situation, si Alice récoltait, elle affamerait son adversaire. Sa phase de récolte est donc annulée (on peut aussi considérer qu'elle ne peut jouer ce coup). On aboutit donc à la situation suivante.



Dans la deuxième situation, Alice récolte les cases 10, 9, 8, 7, 6. On aboutit donc à la situation suivante.



4. Le joueur 1 joue exactement lorsqu'un nombre pair de tours ont déjà été joués, donc lorsque `jeu['n']` est pair.

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

5. La liste $[p_0, \ldots, p_{11}]$ est changée en la liste $[p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_0, p_1, p_2, p_3, p_4, p_5]$. Remarquons que $p_{-1} = p_{11}$,...,$p_{-6} = p_6$. Il suffit donc de soustraire 6 aux coordonnées. On pourrait aussi utiliser deux tranches.

```
3  def tourner_plateau(jeu):
4      jeu['plateau'] = [jeu['plateau'][i-6] for i in range(12)]
```

6. Un case peut contenir au maximum 48 graines. Comme $2^6 = 64$, il suffit de 6 bits pour coder chaque entier représentant le nombre de graines par case.

7.

```
5  def copie(jeu):
6      njeu = initialisation(jeu['joueur1'], jeu['joueur2'])
7      njeu['score'] = jeu['score'].copy()
8      njeu['n'] = jeu['n']
9      njeu['plateau'] = jeu['plateau'].copy()
10     return njeu
```

8. On utilise une variable `caseADeposer` qui donne le numéro de la case dans laquelle on va déposer les graines. Si la graine doit être déposée dans la case de départ, on augmente ce compteur de 1. On effectue une division euclidienne par 12 pour obtenir à chaque fois un numéro de case valide.

```
11 def deplacer_graines(plateau, case) :
12     nbg = plateau[case]
13     plateau[case] = 0
14     caseADeposer = case
15     for i in range(nbg):
16         caseADeposer = (caseADeposer + 1) % 12
17         if caseADeposer == case :
18             caseADeposer = (caseAdeposer + 1) % 12
19         plateau[caseADeposer] +=1
20     return caseADeposer
```

9.

```
21 def case_ramassable(plateau, case):
22     return (6 <= case < 12) and (2 <= plateau[case] <= 3)
```

10.

```
23 def ramasser_graines(plateau, case):
24     if not case_ramassable(plateau, case) :
25         return 0
26     else :
27         nbg, plateau[case] = plateau[case], 0
28         return nbg + ramasser_graines(plateau, case-1)
```

11. Il suffit de copier le plateau, de faire jouer un tour (semaille et récolte), et de vérifier que les cases adverses ne sont pas toutes nulles.

```
29 def test_famine(plateau, case) :
30     nPlateau = plateau.copy()
31     caseFin = deplacer_graines(nPlateau, case)
32     ramasser_graines(nPlateau, caseFin)
33     return nPlateau[6:] != [0]*6
```

12.

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

13.

```
38 def case_possibles(jeu):
39     return [i for i in range(6) if test_case(jeu['plateau'], i)]
```

14. On vérifie une à une les conditions données. On réalise la somme manuellement, la fonction `sum` étant proscrite.

```
40 def tour_suivant(jeu):
41     if jeu['score'][0] >= 25 or jeu['score'][1] >= 25 :
42         return False
43     elif  jeu['n'][0] >= 100:
44         return False
45     elif len(cases_posibles(jeu)) == 0 :
46         return False
47     else :
48         nbg = 0
49         for g in jeu['plateau'] :
50             nbg += g
51         return nbg >= 4
```

15.

```
52 def tour_jeu(jeu, case) :
53     plateau = jeu['plateau']
54     if test_case(plateau, case) :
55         caseFinale = deplacer_graines(plateau, case)
56         graines_gagnees = ramasser_graines(plateau, caseFinale)
57         if tour_joueur1(jeu):
58             jeu['score'][0] +=  graines_gagnees
59         else :
60             jeu['score'][1] +=  graines_gagnees
```

```
61          jeu['n'] += 1
62          tourner_plateau(jeu)
63          return tour_suivant(jeu)
64      else :
65          print("La case choisie n'est pas valable")
66          return True
```

16. Il faut faire attention à vérifier quel est le joueur actif afin de décider quel joueur rammase chaque camp.

```
67  def gagnant(jeu):
68      if tour_joueur1(jeu):
69          for i in range(6):
70              jeu['score'][0] += jeu['plateau'][i]
71              jeu['score'][1] += jeu['plateau'][6+i]
72      else :
73          for i in range(6):
74              jeu['score'][1] += jeu['plateau'][i]
75              jeu['score'][0] += jeu['plateau'][6+i]
76      if jeu['score'][0] > jeu['score'][1]:
77          return jeu['joueur1']
78      elif jeu['score'][0] < jeu['score'][1]:
79          return jeu['joueur2']
80      else :
81          return "egalite"
```

17.

```
82  def gain(jeu, case):
83      jeuSuivant = copie(jeu)
84      plateau = jeuSuivant['plateau']
85      caseFinale = deplacer_graines(plateau, case)
86      graines_gagnees = ramasser_graines(plateau, caseFinale)
87      if tour_joueur1(jeuSuivant):
88          jeuSuivant['score'][0] +=  graines_gagnees
89      else :
90          jeuSuivant['score'][1] += graines_gagnees
91      jeuSuivant['n'] += 1
92      tourner_plateau(jeuSuivant)
93      return graines_gagnees, jeuSuivant
```

18. Alice va jouer la case 5 (voir arbre en fin de corrigé).

19. — Condition 1.

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

— Condition 2.

```
95 profondeur >= profondeur_max
```

— Instruction 1

```
96 g, jeuSuivant = gain(jeu, case)
```

— Instruction 2

```
97 p = NegaAwale(jeuSuivant, profondeur_max, profondeur+1)
```

20. On réalise au début un calcul classique de maximum. On décide en fin de fonction si l'on renvoie la valeur de la case ou de $g - p$, en fonction de la profondeur.

```
98  def max_vals(vals_jeu, profondeur) :
99      imax, n = 0, len(vals_jeu)
100     for i in range(1,n):
101         if vals_jeu[i][1] > vals_jeu[imax][1] :
102             i = imax
103     if profondeur == 0 :
104         return vals_jeu[imax][0]
105     else :
106         return vals_jeu[imax][1]
```

21. On remplace la ligne 6 par la ligne suivante.

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

22.

```
108 SELECT id_Joueur
109 FROM Joueur
110 WHERE niveau > 1900 ;
```

23. On calcule dans une sous-requête le nombre total de victoires du joueur 1, quand il joue dans la case 0.

```
111 SELECT 100.0 * nbVic / COUNT(*)
112 FROM Partie, (SELECT COUNT(*) AS nbVic
113               FROM Partie
114               WHERE jeu LIKE 'a%' AND resultat = 1)
115 WHERE jeu LIKE 'a%' ;
```

24.

```
116 SELECT nom, prenom
117 FROM Joueur
118 ORDER BY niveau DESC
119 LIMIT 3 ;
```

25.

```
120 SELECT nom, prenom, COUNT(*) AS nbVic
121 FROM Joueur
122 JOIN Partie ON id_Joueur = id_joueur1
123 WHERE resultat = 1.0
124 GROUP BY id_Joueur
125 HAVING nbVic > 100
126 ORDER BY nbVic DESC ;
```

NE RIEN ÉCRIRE DANS CE CADRE

Q18 - Arbre des jeux possibles à compléter