teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun INP

Année : 2024

Filière : PSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles).

L'équipe UPSTI

Le jeu de l'awalé

Corrigé UPSTI

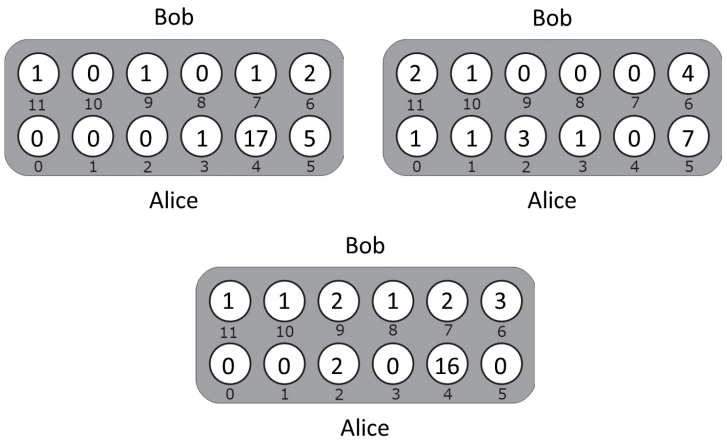
1 Présentation et règles

Question 1

Alice peut jouer les cases 2, 4 et 5.

- Ces cases appartiennent à son camp.
- Ces cases sont non vides.
- Jouer une de ses cases n'entraine pas de famine.

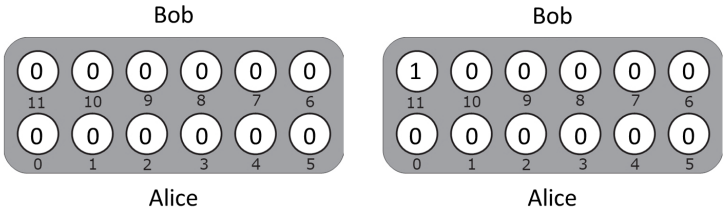
Question 2



Gains éventuels :

- case 2 : aucun gain ;
- case 5 : aucun gain ;
- case 4 : la dernière case semée 9 contient 3 graines ; la case 8 contient 2 graines ; la case 7 contient 3 graines donc les graines des cases 8 et 7 sont ramassées et le gain est de 8.

Question 3



- Situation de gauche : le coup d'Alice affame Bob donc la récolte est annulée ; il n'y a plus de coup possible pour Alice donc la partie se termine.
- Situation de droite : il reste une graine dans le camp de Bob donc la récolte a lieu ; il ne reste qu'une graine sur la plateau donc la partie se termine.

2 Programmation de la structure de jeu

Question 4

Lorsque c'est au joueur1 de jouer, jeu['n'] est pair.

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

Question 5

```
def tourner_plateau(jeu) :
    for i in range(6) :
        jeu["plateau"][i], jeu["plateau"][i+6] = jeu["plateau"][i+6],
                                                 jeu["plateau"][i]
```

Question 6

Au maximum, il y a 48 graines dans une case. 6 bits sont donc nécessaires.

Question 7

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

Question 8

```
def deplacer_graines(plateau,case) :
    nb = plateau[case]
    plateau[case] = 0
    i = 1
    j = 0
    while j < nb :
        if case+i%12 != case :
            plateau[(case+i)%12] += 1
            j += 1
        i += 1
    return((case+i-1)%12)
```

Question 9

```
def case_ramassable(plateau,case) :
    return (case>5 and plateau[case] in [2,3])
```

Question 10

```
def ramasser_graines(plateau,case) :
    if case_ramassable(plateau,case) == False : # cas terminal
        return 0
    R = plateau[case] # cas general
    plateau[case] = 0
    return R + ramasser_graines(plateau,case-1)
```

Question 11

```
def test_famine(plateau, case) :
    if plateau[case] == 0 :
        return False
    plateau_test = plateau.copy()
    derniere_case = deplacer_graines(plateau_test,case)
    ramasser_graines(plateau_test,derniere_case)
    return sum(plateau_test[6:]) != 0
```

Question 12

```
def test_case(plateau,case):
    condition3 = test_famine(plateau,case)
    test = condition3 and case <= 5 and plateau[case] != 0 and case >= 0
    return test
```

Question 13

```
def cases_possibles(jeu) :
    S=[]
    for i in range(6) :
        if test_case(jeu["plateau"],i) :
            S.append(i)
    return S
```

Question 14

```
def tour_suivant(jeu) :
    if jeu["score"][0] >= 25 or jeu["score"][1] >= 25:
        return False
    if jeu["n"] >= 100:
        return False
    Somme = 0
    for elt in jeu["plateau"]:
        Somme += elt
    if Somme <= 3:
        return False
    if len(cases_possibles(jeu)) == 0:
        return False
    return True
```

Question 15

```
def tour_jeu(jeu, case) :
    plateau = jeu["plateau"]
    if test_case(plateau,case) :
        derniere_case = deplacer_graines(plateau,case)
        graines_gagnees = ramasser_graines(plateau,derniere_case)
        if tour_joueur1(jeu) :
            jeu["score"][0] = jeu["score"][0] + graines_gagnees
        else :
            jeu["score"][1] = jeu["score"][1] + graines_gagnees
        jeu["n"] += 1
        tourner_plateau(jeu)
        return tour_suivant(jeu)
    else :
        print("La case choisie n'est pas valable")
        return True
```

Question 16

```
def gagnant(jeu) :
    if tour_joueur1(jeu) :
        jeu["score"][0] += sum(jeu["plateau"][:6])
        jeu["score"][1] += sum(jeu["plateau"][6:])
    else :
        jeu["score"][1] += sum(jeu["plateau"][:6])
        jeu["score"][0] += sum(jeu["plateau"][6:])
    if jeu["score"][0] > jeu["score"][1] :
        return jeu["joueur1"]
    elif jeu["score"][0] < jeu["score"][1] :
        return jeu["joueur2"]
    else :
        return "egalite"
```

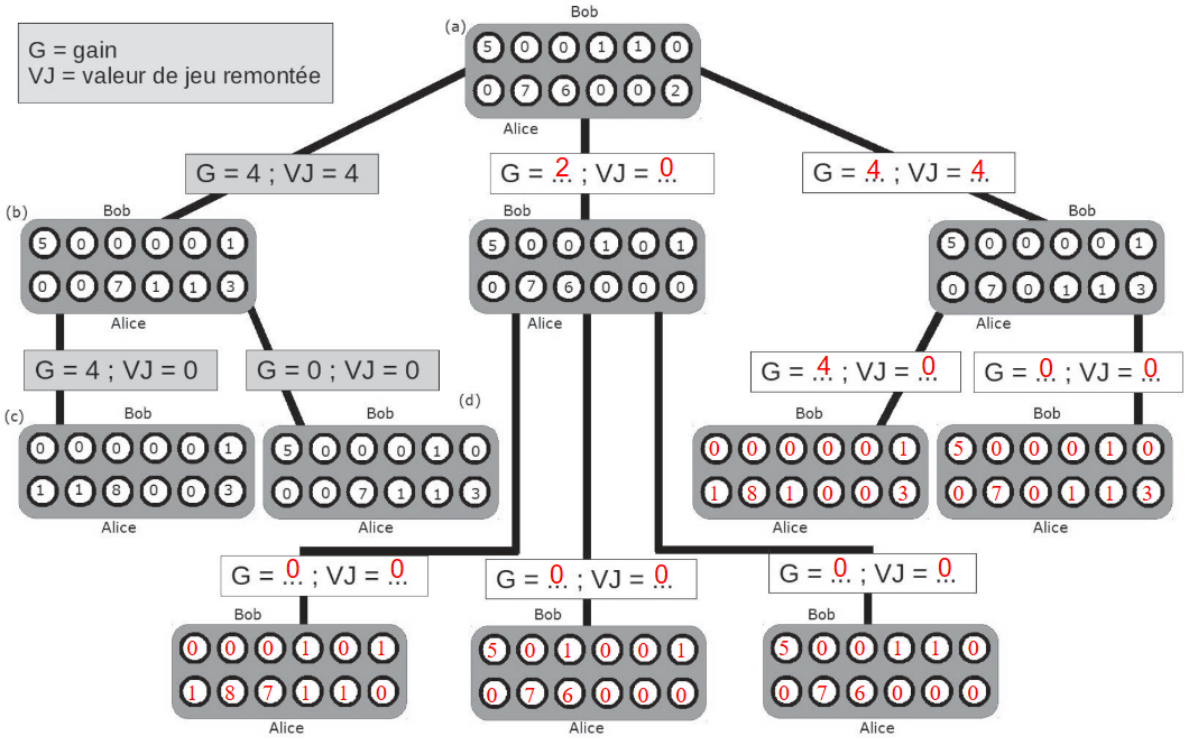
3 Programmation de l'Intelligence Artificielle

Question 17

```
def gain(jeu,case):
    jeu_bis = copie(jeu)
    derniere_case = deplacer_graines(jeu_bis["plateau"],case)
    graines_gagnees = ramasser_graines(jeu_bis["plateau"],derniere_case)
    if tour_joueur1(jeu_bis) :
        jeu_bis['score'][0] = jeu_bis['score'][0]+graines_gagnees
    else :
        jeu_bis['score'][1] = jeu_bis['score'][1]+graines_gagnees

    #Mise a jour du plateau
    jeu_bis["n"] += 1
    tourner_plateau(jeu_bis)
    return graines_gagnees, jeu_bis
```

Question 18



Avec cette profondeur de recherche, Alice maximise son gain en jouant la case 5.

Question 19

Condition 1 :

```
tour_suivant(jeu) == False
```

Condition 2 :

```
profondeur == profondeur_max
```

Instruction 1 :

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

Instruction 2 :

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

Question 20

```
def max_vals(vals_jeu, profondeur) :
    if profondeur==0:
        indice_maxi=0
        maxi=vals_jeu[0][1]
        for i in range(1, len(vals_jeu)):
            if vals_jeu[i][1] > maxi:
                indice_maxi=i
                maxi=vals_jeu[i][1]
        return vals_jeu[indice_maxi][0]
```

```
    else:
        maxi=vals_jeu[0][1]
        for i in range(1, len(vals_jeu)):
            if vals_jeu[i][1] > maxi:
                maxi=vals_jeu[i][1]
        return maxi
```

Question 21

```
def awale_jcj_IA(nom_joueur1, nom_joueur2) :
    jeu=initialisation(nom_joueur1, nom_joueur2)
    jeu_continue=True
    while jeu_continue:
        affiche(jeu["plateau"])
        case_choisie=NegaAwale(jeu, 5, 0)
        jeu_continue=tour_jeu(jeu, case_choisie)
    return gagnant(jeu)
```

Question 22

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

Question 23

```
SELECT COUNT(resultat)*100 / (SELECT COUNT(resultat) FROM Partie WHERE jeu LIKE "a%")
AS Pourcentage FROM Partie WHERE resultat = 1 AND jeu LIKE "a%" ;
```

Question 24

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

Question 25

```
SELECT nom, prenom, COUNT(resultat) from Partie JOIN Joueur ON id_Joueur = id_joueur1
WHERE resultat = 1 GROUP BY id_Joueur1 HAVING COUNT(resultat) >= 2
ORDER BY COUNT(resultat) ASC;
```