

Devoir surveillé d'Informatique n° 1
Lundi 18 Novembre
Durée 2 heures

Le soin apporté à la rédaction, la clarté des raisonnements, l'orthographe et la présentation seront pris en compte dans la notation.
Il est vivement recommandé d'encadrer les résultats et de signaler toute question admise.
Documents et calculatrices sont interdits.

Parties I et II de l'épreuve d'Informatique CCINP 2024 (qui en comporte 4)

Le jeu de l'awalé

Les fonctions seront définies avec leur signature dans le sujet :

`ma_fonction(arg1:type1, arg2:type2) → type3.`

Cette notation permet de définir une fonction qui se nomme `ma_fonction` qui prend deux arguments en entrée `arg1` de type `type1` et `arg2` de type `type2`. Cette fonction renvoie une valeur de type `type3`.

Il ne faut pas recopier les signatures des fonctions, il faut écrire directement :

```
def ma_fonction(arg1, arg2):  
    # liste d'instructions
```

Partie I - Présentation et règles

Le sujet porte sur l'étude de l'awalé, un jeu de stratégie très ancien qui fait partie des jeux de semailles. En effet, à son origine, il était pratiqué à l'aide de graines qui étaient semées dans deux rangées de 6 trous creusés dans un plateau en bois ou à même le sol. Ce jeu est très répandu en Afrique et à partir du XVIIe siècle des indices de sa pratique sont également trouvés en Amérique du Sud et en Asie.



Figure 1 - Plateau d'awalé en bois

Les règles de ce jeu sont particulièrement simples et s'apprennent rapidement. Il en existe plusieurs variantes mais ce sujet n'en exposera qu'une seule. L'awalé se joue à deux. À tour de rôle, les joueurs prennent les graines situées dans un trou de leur rangée pour ensuite les déplacer dans les autres trous. Des graines peuvent ensuite être récoltées pour que les joueurs se constituent une réserve personnelle.

I. 1 - But du jeu

L'objectif est de récolter plus de graines que son adversaire. Au départ, le plateau est composé de 2 rangées de 6 trous contenant chacun 4 graines comme le montre la figure 2. Le jeu s'arrête si l'un des joueurs obtient dans sa réserve personnelle à côté du plateau plus de la moitié des graines en jeu, c'est-à-dire au moins 25 graines ou jusqu'à une situation empêchant le gain de nouvelles graines.

I. 2 - Déroulement de la partie

Les 2 participants jouent à tour de rôle. Les joueurs sont appelés par la suite Alice et Bob ; Alice joue en premier. Les joueurs sont face à face. La rangée de 6 trous située juste devant le joueur est appelée son camp.

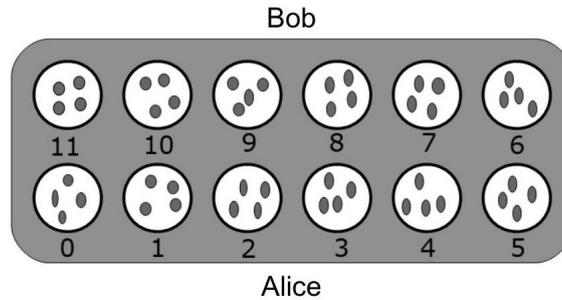


Figure 2 - Situation de départ

Chaque coup consiste à choisir une case non vide de son camp, à prendre toutes les graines de cette case en main et à les semer à raison d'une graine par case en suivant le sens direct, c'est-à-dire le sens inverse des aiguilles d'une montre (figure 3). On ne sème jamais de graine dans la case d'origine choisie. En effet, si la case de départ choisie contient plus de 11 graines, le joueur va semer sur un tour de plateau complet et revenir à la case d'origine des graines ; il faut alors sauter cette case pour continuer de semer les graines dans les autres cases. À la fin de son tour de jeu, la case de départ choisie par le joueur est nécessairement vide.

Bob

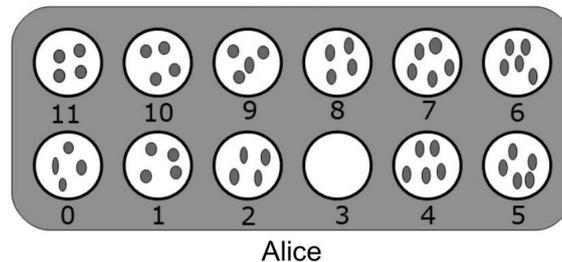


Figure 3 - Situation obtenue après qu'Alice ait semé la case d'indice 3

Une règle fondamentale de l'awalé est l'interdiction d'affamer son adversaire (**règle de la famine**). Lorsqu'un joueur n'a plus de graines dans son camp, son adversaire est obligé de jouer un coup qui lui en apporte au moins une.

Par ailleurs, il est interdit de jouer un coup qui ôte, après récolte, toutes les graines du camp adverse.

Une fois qu'un joueur a terminé de semer, il peut récolter les graines du plateau de jeu (en respectant la règle précédente).

La récolte consiste à retirer les graines du plateau pour les stocker dans sa réserve personnelle sur la table à côté du plateau de jeu. Le joueur récolte les graines disponibles après son tour de semence, en commençant par la dernière case dans laquelle il a semé et sous les conditions suivantes :

- la case appartient au camp adverse (condition 1) ;
- cette case contient exactement 2 ou 3 graines (condition 2) ;
- s'il vient de ramasser les graines de la case, le joueur doit continuer la récolte dans le sens inverse de la semence, si la case respecte les deux premières conditions ;
- il est interdit d'affamer son adversaire, on ne peut donc pas prendre toutes les graines du camp adverse. Si la phase de récolte se termine ainsi, alors la récolte est annulée (condition 3 liée à la règle de la famine).

I. 3 - Conditions de fin

La partie s'arrête sous certaines conditions :

- un joueur obtient au moins 25 graines dans sa réserve ;
- 3 graines ou moins restent sur le plateau ;
- un joueur est dans l'incapacité de jouer car aucun coup ne permet de respecter les différentes règles.

À la fin de la partie, chaque joueur ramasse les graines de son camp pour les transférer dans sa réserve personnelle. Le décompte des points peut alors se faire. Le joueur ayant obtenu au moins 25 graines est alors déclaré vainqueur. Une situation de nullité est possible si les deux joueurs obtiennent autant de graines chacun à la fin de la partie.

I. 4 - Compréhension des règles

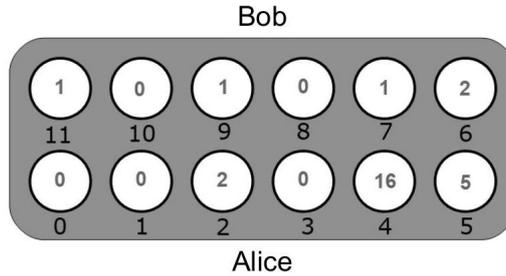


Figure 4 - Exemple de situation - Alice doit jouer

On considère la situation de jeu indiquée par la figure 4. C'est au tour d'Alice de jouer.

Q1. Indiquer, en justifiant, les indices des cases qu'Alice peut choisir de jouer.

Q2. Pour chacun des choix de cases possibles, renseigner la situation possible du plateau de jeu après le coup d'Alice (c'est-à-dire après avoir semé et récolté les graines). Indiquer également le gain éventuel pour chaque coup possible de la figure 4.

On considère les deux situations de jeu indiquées par la figure 5. C'est au tour d'Alice de jouer.

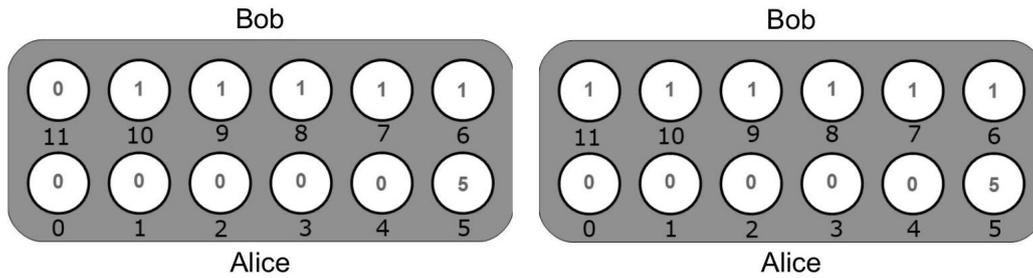


Figure 5 - Autres exemples de situation - Alice doit jouer

Q3. Pour chacune des deux situations de jeu de la figure 5, donner et justifier la situation du plateau après le tour de jeu d'Alice.

Partie II - Programmation de la structure de jeu

Cette partie aborde la modélisation et la structure du jeu dans le langage Python.

II. 1 - Représentation du jeu

Le choix d'un dictionnaire a été fait pour stocker l'ensemble des données du jeu. De plus, l'appel de la fonction `initialisation(nom_joueur1:str,nom_joueur2:str) → dict` permet de créer la structure du jeu dans les conditions de départ.

```
def initialisation(nom_joueur1, nom_joueur2) :
    jeu = {}
    jeu['joueur1'] = nom_joueur1           # Nom du premier joueur
    jeu['joueur2'] = nom_joueur2           # Nom du second joueur
    jeu['score'] = [0,0]                    # Réserve du joueur1, puis du joueur2
    jeu['n'] = 0                             # Nombre de tours déjà effectués
    jeu['plateau'] = [4]*12                  # Plateau de jeu initial
    return jeu
```

Le compteur de tours détermine le tour de jeu des joueurs et commence donc à zéro. Le joueur1 commence la partie. Le plateau de jeu est séparé en deux parties égales. Les six premières cases correspondent à celles du joueur dont c'est le tour ; les six dernières cases sont celles de l'adversaire. À la fin d'un tour de jeu, il faut échanger les deux ensembles de six cases. Ainsi les cases d'indices 0 à 5 correspondent toujours à celles du joueur dont c'est le tour (joueur actif) et les cases d'indices 6 à 11 à celles de son adversaire.

L'argument `jeu:dict` fait référence à un dictionnaire représentant le jeu de structure identique à celui renvoyé par `initialisation`.

Q4. Donner la parité de `jeu['n']` lorsque c'est au tour du joueur1 de jouer. Écrire une fonction `tour_joueur1(jeu:dict) → bool` qui renvoie `True` si c'est le tour de jeu du joueur1 et `False` sinon.

Q5. Écrire une fonction `tourner_plateau(jeu:dict) → None` qui modifie l'entrée "plateau" du dictionnaire jeu en échangeant les cases d'indices 0 à 5 avec celles de 6 à 11 (figure 6).

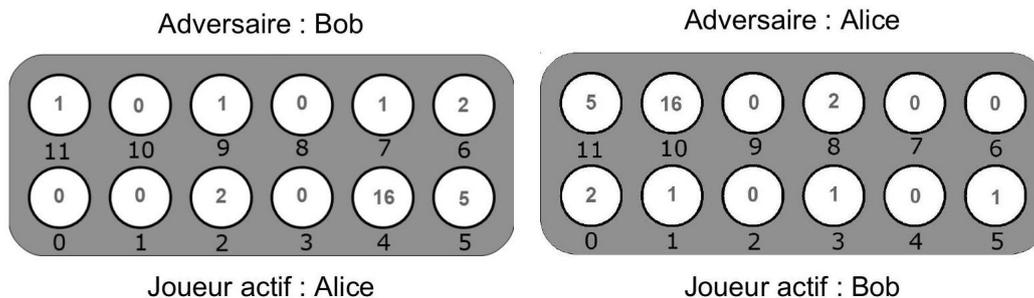


Figure 6 - Exemple d'inversion de plateau à la fin d'un tour entre Alice et Bob

Q6. Donner le maximum de graines que peut contenir une case. Déterminer alors le nombre de bits nécessaires pour coder les entiers représentant le nombre de graines par case.

Dans la suite du sujet, nous aurons besoin d'une fonction permettant de copier le jeu (dictionnaire) en entier dont certains éléments sont des listes.

Q7. Écrire une fonction `copie(jeu:dict) → dict` qui renvoie une copie profonde de la structure de dictionnaire retenue. Il est interdit d'utiliser la fonction `deepcopy` du module `copy`. L'opérateur `copy` des listes est autorisé.

Le déroulement d'un jeu d'awalé entre deux joueurs "humains" a la structure suivante :

```
1 def awale_jcj(nom_joueur1, nom_joueur2):
2   jeu = initialisation(nom_joueur1, nom_joueur2)
3   jeu_continue = True
4   while jeu_continue :
5       affiche(jeu['plateau'])
6       case_choisie = int(input("Choisir une case : "))
7       jeu_continue = tour_jeu(jeu, case_choisie)
8   return gagnant(jeu)
```

Remarque : la fonction **input** permet de récupérer une chaîne de caractères fournie par le joueur dans la console.

La fonction **affiche** permet d'afficher le plateau de jeu à l'écran.

Dans le code proposé, nous considérons que le joueur ne commet pas d'erreur de frappe et rentre toujours dans la console l'indice de la case choisie.

II. 2 - Programmation d'un tour de jeu

Une fois la case de début de tour choisie par le joueur, un tour de jeu a la structure suivante :

1. tester si la case où l'on prend les graines est valide ou non (les conditions de validité sont explicitées ensuite) ;
2. si le choix est valide, semer les graines puis récolter des graines. On incrémente alors le nombre de tours, on ajoute au score du joueur le nombre de graines récoltées et on échange les deux parties du plateau ;
3. tester si la partie est finie ou non (les conditions sont décrites dans la première partie).

Les pré-conditions des fonctions de cette partie et de la partie suivante sont répertoriées ci-dessous :

- l'argument **jeu:dict** fait référence à un dictionnaire représentant le jeu comme présenté au début de cette partie ;
- l'argument **plateau:[int]** fait référence à une structure de données similaire à ce que contient **jeu['plateau']** ;
- l'argument **case:int** fait référence à un entier compris entre 0 inclus et 12 exclu.

On s'intéresse tout d'abord à l'étape 2 de manière à écrire deux fonctions :

- **deplacer_graines(plateau:[int], case:int) → int** ;
- **ramasser_graines(plateau:[int], case:int) → int**.

La fonction **deplacer_graines(plateau:[int], case:int) → int** prend comme argument le plateau de jeu ayant la structure choisie précédemment et la case non vide où le joueur actuel prend les graines. Cette fonction réalise la prise des graines de la case choisie et les sème une par une dans le sens direct (sens inverse des aiguilles d'une montre). Elle renvoie l'indice de la case où la dernière graine a été semée.

Q8. Proposer une fonction **deplacer_graines(plateau:[int], case:int) → int** modifiant "en place" l'argument **plateau** (c'est-à-dire que les modifications se feront directement sur la liste **plateau** sans créer d'autre liste). On rappelle que l'on ne sème pas de graine dans la case choisie en début du tour.

Q9. Écrire une fonction auxiliaire **case_ramassable(plateau:[int], case:int) → bool** qui renverra **True** si le joueur dont c'est le tour a le droit de ramasser les graines de la case proposée et **False** sinon. On ne testera pas la question de la famine pour simplifier le problème. Pour rappel, le joueur peut ramasser le contenu de la case si :

- la case appartient au camp de l'adversaire, soit toujours dans la deuxième moitié du plateau ;
- la case contient 2 ou 3 graines.

La fonction **ramasser_graines(plateau:[int], case:int) → int** prend comme argument le plateau de jeu après déplacement des graines et l'indice de la case où la dernière graine a été semée. Cette fonction procède au ramassage des graines.

Si le joueur peut ramasser le contenu de la case, alors il ramasse le contenu et passe à la case précédente puis ramasse les graines de cette case si ces mêmes conditions sont respectées et ainsi de suite. Pour simplifier, la fonction **ramasser_graines** ne testera pas la condition de famine citée précédemment. Elle renverra le nombre de graines récoltées (c'est-à-dire le nombre de points gagnés).

Q10. Écrire la fonction **ramasser_graines(plateau:[int], case:int) → int**. On demande que la fonction **ramasser_graines** soit une fonction récursive. Cette fonction utilisera la fonction précédente **case_ramassable** et devra modifier le plateau en place et renvoyer le résultat de la récolte des graines.

Il faut vérifier à chaque tour de jeu si le choix d'une case est autorisé ou non. Si c'est un joueur humain qui joue, son choix peut se porter sur une case "interdite", c'est-à-dire dont il ne peut pas prendre les graines. Si c'est un joueur virtuel, celui-ci doit pouvoir faire la liste des cases "acceptables". Une case est "acceptable" si :

- condition 1 : elle est du côté du joueur dont c'est le tour ;
- condition 2 : elle est non vide ;
- condition 3 : à la fin du tour de jeu, les cases de l'adversaire ne sont pas complètement vides (condition de famine).

Q11. Écrire une fonction `test_famine(plateau:[int], case:int) → bool` qui vérifie que la case choisie vérifie la condition 3 (renvoie `True` si la condition 3 est vérifiée et `False` sinon). Il n'est pas nécessaire qu'elle vérifie les deux conditions 1 et 2. Il est possible d'utiliser les fonctions demandées précédemment quelle que soit leur implémentation. *Remarque : les arguments d'entrée ne doivent pas être modifiés par la fonction.*

Q12. On propose ci-dessous une fonction qui vérifie que la case choisie vérifie bien les trois conditions précédentes, connaissant l'état actuel du jeu (le plateau). Compléter la condition de valeur `test` afin de déterminer si la case est acceptable ou non.

```
def test_case(plateau, case):
    """ Vérifie si la case choisie par le joueur est acceptable
    renvoie True si la case est acceptable, False sinon """
    condition3 = test_famine(plateau, case)
    # Case acceptable
    test = # à compléter
    return test
```

Q13. Écrire la fonction `cases_possibles(jeu:dict) → [int]` qui renvoie la liste des indices de toutes les cases jouables par le joueur actif. Le dictionnaire `jeu` ainsi que sa clé `plateau` ne devront pas être modifiés.

Après un tour, il faut vérifier si le jeu est terminé ou si les joueurs peuvent continuer à jouer.

Le jeu se termine si l'une des conditions suivantes est vérifiée :

- un des joueurs possède plus de la moitié des graines (soit au moins 25) ;
- le nombre de tours joués est supérieur ou égal à 100 (pour éviter un jeu infini lorsqu'il y a peu de graines) ;
- il reste 3 graines ou moins sur le plateau ;
- le joueur qui va jouer ne possède plus de case jouable. On suppose que le plateau a été échangé avant de faire les tests, donc le joueur qui doit jouer a ses graines dans les cases d'indices 0 à 5.

Q14. Écrire une fonction `tour_suivant(jeu:dict) → bool`. Cette fonction renvoie `True` si le jeu peut continuer et `False` sinon.

L'ébauche de la fonction `tour_jeu(jeu:dict, case:int) → bool` est donnée ci-dessous. Si le test de la case n'est pas valide, la fonction affiche un message et renvoie `True` pour permettre au joueur de proposer une nouvelle case. Sinon, elle renvoie `True` si le jeu peut continuer et `False` si le jeu ne peut pas continuer.

Q15. Donner l'instruction 1, l'instruction 2, la condition 1 et l'instruction 3 permettant à la fonction de répondre à la description précédente.

```
def tour_jeu(jeu, case):
    plateau = jeu['plateau']
    if test_case(plateau, case): # La case jouée est acceptable
        # Instruction 1 : déplacer les graines
        # Instruction 2 : ramasser les graines
        """ "Pour augmenter le score, il faut savoir qui joue grâce à la
        parité du nombre de tours"""
        if # Condition 1:
            jeu['score'][0] = jeu['score'][0] + graines_gagnees
        else:
            jeu['score'][1] = jeu['score'][1] + graines_gagnees
        # Instruction 3 # On incrémente le nombre de tours
        tourner_plateau(jeu) # Echanger les plateaux
        return tour_suivant(jeu)
    else:
        print("La case choisie n'est pas valable")
        return True
```

Q16. Écrire la fonction `gagnant(jeu:dict) → str` prenant comme argument le dictionnaire `jeu` contenant l'état actuel du jeu. La fonction doit procéder au ramassage des graines de chaque côté du plateau et les affecter au score, puis renvoyer le nom du joueur qui a gagné, c'est-à-dire qui a le plus de graines à la fin du jeu. S'il y a match nul, la fonction devra renvoyer la chaîne de caractère "égalité".

FIN