I. Minimax

On dispose d'un ensemble de billes disposées sur un tas. À tour de rôle, chaque joueur (J_0 et J_1) prend 1, 2 ou 4 billes sur le tas (dans la limite du nombre de billes encore disponibles).

On fixe les règles suivantes :

• S'il reste une bille sur la table, quel que soit le joueur dont c'est le tour de jouer, la partie est nulle (et s'arrête) ;

• Si J_0 prend les deux dernières billes (ou plus), alors il a gagné (et la partie s'arrête) ;

• Si J_1 prend les deux dernières billes (ou plus), alors il a gagné (et la partie s'arrête).

Exemples de parties :

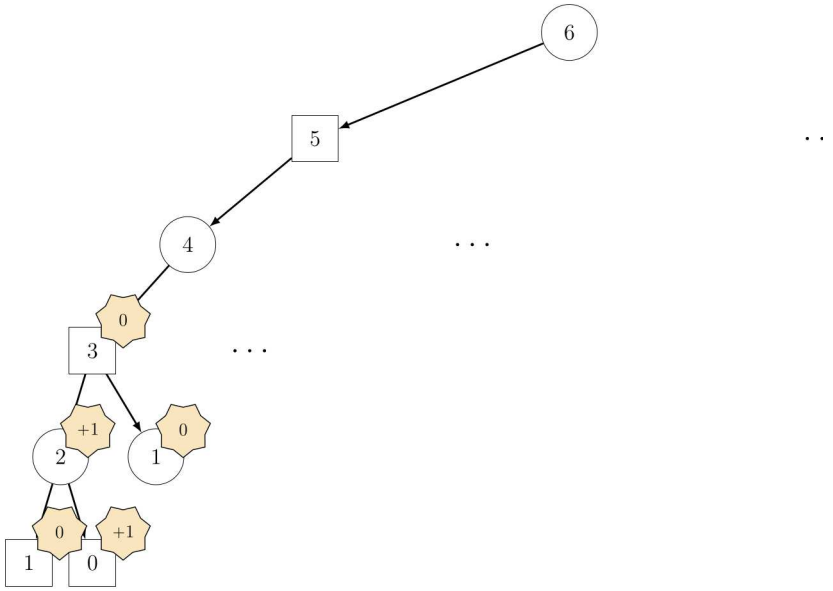
• J_0 prend 4, restent 2 billes ; J_1 prend 2, restent 0 billes, donc J_1 a gagné !

• J_0 prend 1, restent 5 billes ; J_1 prend 4, restent 1 bille, donc partie nulle !

• J_0 prend 2, restent 4 billes ; J_1 prend 2, restent 2 billes ; J_0 prend 2, restent 0 billes, donc J_0 a gagné !

On désire connaître l'issue de la partie s'il y a initialement 6 billes sur la table, que J_0 commence à jouer, et que les deux joueurs jouent parfaitement. Pour répondre à la question posée, nous allons utiliser l'algorithme minimax. Nous voulons attribuer à chaque sommet une valeur de jeu : ‘1 si le sommet est gagnant pour J_0 ; ´1 si le sommet est gagnant pour J_1 ; 0 si le sommet est un sommet de partie nulle. Ainsi, vu que pour un joueur sa victoire est préférable à une partie nulle, qui est elle-même préférable à la victoire de l'autre joueur, J_0 cherchera à placer le jeu sur un état avec une évaluation la plus élevée possible, tandis que J_1 cherchera à placer le jeu sur un état avec une évaluation la plus faible possible. Nous représentons le déroulement de toutes les parties possibles à l'aide d'un arbre : les sommets appartenant au joueur J_0 (quand c'est à J_0 de jouer) sont représentés par des cercles ; les sommets appartenant au joueur J_1 sont représentés par des carrés. À l'intérieur d'un sommet figure le nombre de billes restantes. Par exemple : un carré avec un 5 dedans signifie que c'est à J_1 de jouer et qu'il reste 5 billes.

1. Représentez (sans justification), tel que décrit ci-dessus, les états finals-gagnants pour J_0 , finals-gagnants pour J_1 , et les états finals de partie nulle.
2. Compléter (proprement, sur votre feuille) l'arbre de jeu généré par l'algorithme, et ébauché à la figure.



En regard de chaque noeud, vous placerez son évaluation calculé avec l'algorithme minimax. Expliquez brièvement comment vous avez fait.

3. Concluez sur la question posée : quelle est l'issue du jeu proposé (6 billes initialement, J_0 commence) ?

II. Algorithmes d'IA

Calculs pour les k moyennes

Etant donné un nuage de N points de $\mathbb{R}^2$, sous la forme $N = \{x_1; x_2; \ldots; x_n\}$, on cherche à partitionner en k ensembles $S = \{S_1; S_2; \ldots; S_k\}$ en minimisant la distance entre les points à l'intérieur de chaque partition.

Les points x_i seront assimilés à des tableaux unidimensionnels de la forme $[x_{i,1}, x_{i,2}]$ de type `np.array` où $x_{i,1}$ et $x_{i,2}$ sont de type `float`, à l'aide du package `import numpy as np`.

On définit la fonction f (dite fonction de coût) qui va de l'ensemble S des partitions de vers $\mathbb{R}^+$ par

$$f(S) = \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|_2^2$$

pour toute partition $S = \{S_1; S_2; \ldots; S_k\}$ où μ_i est l'isobarycentre des points dans S_i.

On cherche donc une partition S pour que $f(S)$ soit minimale.

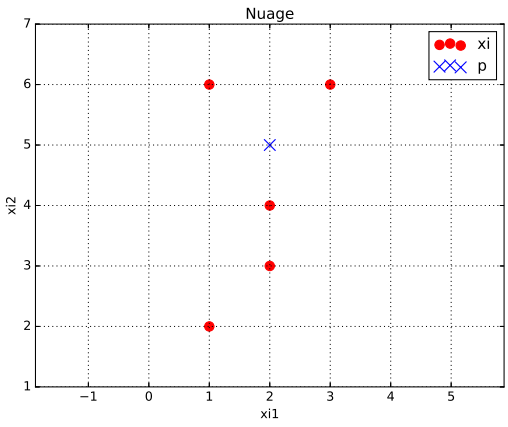
4. Ecrire la fonction `moyenne(E)` qui retourne l'isobarycentre $\mu = \dfrac{1}{\text{Card(E)}} \displaystyle\sum_{x \in E} x$ des points de E ; E étant une liste de points du plan représentés par des tableaux `np.array` à deux éléments.
5. Ecrire une fonction `d2(t1,t2)` qui prend en paramètres deux vecteurs et retourne le carré de la distance euclidienne entre les points qu'ils représentent.
6. Ecrire une fonction `dpe(p,E)` qui prend en paramètre un point p représenté par un tableau unidimensionnel et une liste E représentant un ensemble de points. La fonction retourne la somme des carrés des distances de p aux points de E.
7. En déduire une fonction `cout(S)` qui calcule le coût d'une partition S de N, où S est une liste de listes de sommets.

Calculs pour les k plus proches voisins

8. Que fait la fonction suivante ?

```
def proche(x,L):
  m=min([d2(x,e)for e in L])
  for i in range(len(L):
      if d2(x,L[i])==m:
         return(i)
```

9. Pour $p = [2, 5]$ et $L = [[1, 2], [2, 3], [2, 4], [1, 6], [3, 6]]$ que doit renvoyer `proche(p,L)` ?



10. On rappelle que la commande `L.sort` permet de trier la liste `L`. Quelle est la complexité $C(N)$ en nombre de comparaisons de cette commande de tri pour une liste de longueur N ?

11. La commande `sorted(liste, key=f)` permet de trier les éléments de la liste `liste` par valeurs croissantes de $f : \mathbb{R}^2 \to \mathbb{R}$. Elle peut aussi s'écrire `sorted(liste, key=lambda x: min(f(i) for i in x))`.

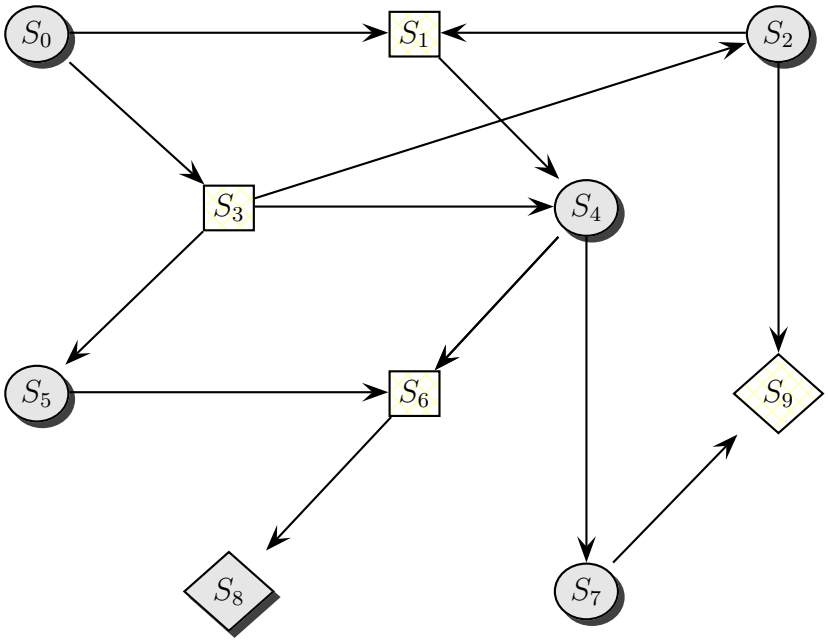
    Compléter le code de la fonction `kproches(x,L,k)` qui renvoie la liste des k plus proches voisins de x dans la liste L.

```
def kproches(table, cible, k) :
    """Revoie la liste des k plus proches voisins de la cible"""
    def distance_cible(donnee) :
        """ renvoie la distance entre la donnée et la cible,
        on choisit la distance euclidienne"""
        distance = d2(donnee,cible)
        return distance

    table_triee = sorted(table, key = distance_cible)
    # table_triee contient la liste des points de ......................
    # triée par valeurs..................... de ..................................


    proches_voisins = []
        for i in range(k) :
        proches_voisins.append(......)
    return .....
```

III. Calcul de positions gagnantes sur un graphe d'accessibilité à deux joueurs.

On considère un jeu d'accessibilité, où le premier joueur J_1 joue les noeuds ronds, le second joueur J_2 joue les noeuds carrés. Les noeuds représentés par un losange correspondent à la victoire de l'un des joueurs.

Initialement le joueur J_1 démarre la partie sur le sommet S_0.



Légende :

état j contrôlé par J_1 

état i contrôlé par J_2 

victoire du joueur J_1 

victoire du joueur J_2 

On note $G = (V, E)$ où V est l'ensemble des sommets, E l'ensemble des arêtes.

12. Justifiez que l'arène du jeu est bipartie.
13. Justifier que l'attracteur pour la victoire du joueur J_1 doit être initialisé à $A_0 = \{S_8\}$.
14. Calculer A_1, A_2, A_3.

    on rappelle que A_{n+1} s'obtient en ajoutant à A_n les noeuds appartenant à J_1 qui permettent de rejoindre un noeud de A_n ou les noeuds appartenant à J_2 qui conduisent obligatoirement à rejoindre un noeud de A_n
15. En déduire les positions gagnantes du joueur 1.
16. En notant $V_1^+ = \{S_0, S_2, S_4, S_5, S_7\}$ l'ensemble des sommets de degré sortant non nul contrôlés par le joueur J_1, Proposer une stratégie gagnante pour le joueur 1, notée $\varphi : V_1^+ \to V$, telle que $\forall s \in V_1^+$, $(s, \varphi(s)) \in E$.

DOCUMENT RÉPONSE

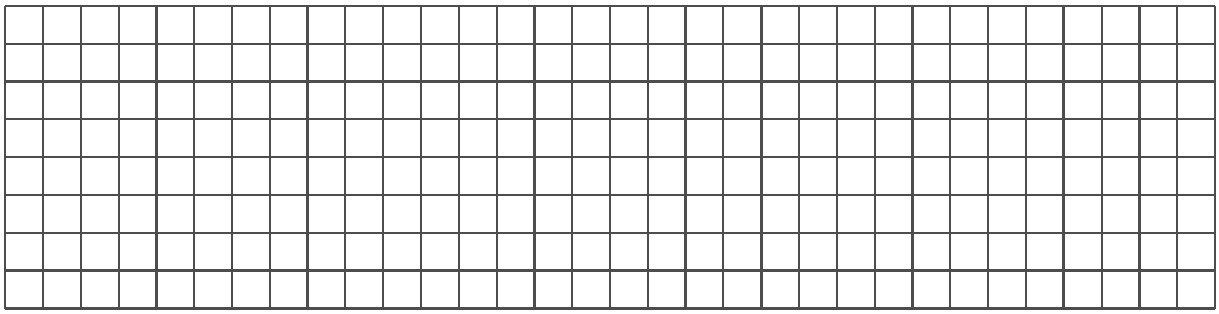
Ce Document Réponse doit être rendu dans son intégralité.

NOM : **Prénom :**

Q1- Etats finaux et gagnants pour J_0, pour J_1, Etats finaux nuls

Q2- Arbre du jeu

Q3- Conclusion : issue du jeu



Q4- Compléter le code de la fonction moyenne(E :list)->np.array

```
def  moyenne(E):
   N=len(E)
   x=0
   y=0
   for i in range(......):
      x+=E[i][0]
      y+=.........
   return([......,......])
```

Q5- Compléter le code de la fonction d2(t1 :np.array,t2 :np.array)->float

```
def  d2(t1,t2):
   x=.........
   y=.........
   return(x**2+y**2)
```

Q6- Compléter le code de la fonction dpe(p :np.array,E :list)->float

```
def  dpe(p,E):
   S=0
   for ......:
      S+=......
   return(S)
```

Q7- Compléter le code de la fonction cout(S :list)->float

```
def  cout(S):
   k=len(S)
   C=......
   for j in range(k):
      L=S[j]
      C+=dpe(............,............)
   return(C)
```

Q8- Que fait la fonction `proche` ?

Q9- Que renvoie `proche(p,L)` ?

Q10- On rappelle que la commande `L.sort` permet de trier la liste `L`. Quelle est la complexité $C(N)$ en nombre de comparaisons de cette commande de tri pour une liste de longueur N ?

Q11- Compléter le code de

```
def kproches(table, cible, k) :
    """Revoie la liste des k plus proches voisins de la cible"""
    def distance_cible(donnee) :
        """ renvoie la distance entre la donnée et la cible,
        on choisit la distance euclidienne"""
        distance = d2(donnee,cible)
        return distance

    table_triee = sorted(table, key = distance_cible)
    # table_triee contient la liste des points de ......................
    # triée par valeurs..................... de ..................................

    proches_voisins = []
        for i in range(k) :
        proches_voisins.append(.................................)
    return .....
```

Q12- Justifier que l'arène est bipartie.

Q13- Justifier l'initialisation $A_0 = \{S_8\}$ pour l'attracteur de J_1.

Q14- Calculer A_1, A_2, A_3

Q15- Positions gagnantes de J_1.

Q16- Stratégie gagnante φ pour J_1.