Jeu de Nim. Corrigé

1) a)

```
def arene(n) :
    G = {}
    for k in range(n) :
        G[(k,1)] = [] ; G[(k,2)] = []
    for k in range(n) :
        for q in [1,2] :
            if k-q >= 0 :  G[(k,1)].append((k-q,2))
        for q in [1,3] :
            if k-q >= 0 :  G[(k,2)].append((k-q,1))
    return G
```

b)

```
def sommetsJ1(n) :
    S1 = {}
    for k in range(n) :  S1[(k,1)] = True ; S1[(k,2)] = False
    return S1
```

c)

```
def finalJ1(n) :  return {(0,1):True}
```

2)

```
def transpose(G) :
    Gt = {} # initialisation du graphe de la transposée
    for s in G : # pour chaque sommet de G
        Gt[s] = [] # on crée la clé correspondante
    for s in G : # pour chaque sommet de G
        for t in G[s] :  Gt[t].append(s)
   return Gt
```

3) a)

```
def degres_sortants(G) :
    d = {} # dictionnaire des degrés sortants de G
    for s in G : d[s] = len(G[s])
    return d
```

b)

```
def attracteur(G,S1,F1):
    n = len (G) ; compteur = degres_sortants(G) ; Gt = transpose(G)
    A = {}      # attracteur : dictionnaire des positions gagnantes déjà déterminées
    pile = F1.keys() # initialisation de la pile
    while pile
        s = pile.pop()
         if s not in A :
              A[s] = True      # on met le sommet dans l 'attracteur
              for t in Gt[s]:       # pour les prédécesseurs de s dans G
                   compteur[t] = compteur[t]-1
                   if S1[t] or compteur == 0 :
                        pile.append(t)
    return A
```

c) $(0,1)$ est gagnante pour J_1. Donc $(1,2)$ aussi. Donc $(2,1)$ et $(3,1)$ aussi.

Et $(3,2)$ aussi, car ses successeurs sont $(0,1)$ et $(2,1)$. Donc $(4,1)$ aussi.

Les autres parties $(0,2)$, $(1,1)$, $(2,2)$ et $(4,2)$ sont gagnantes pour J_2.

4) a)

```
def minmax(G,S1,V1,s) :
    if G[s] == [] :
         if (s in finalJ1) :  return 1
         else :  return -1
    if S1[s] :
         return max([minmax(G,S1,V1,t) for t in G[s]]
    else :
         return min([minmax(G,S1,V1,t) for t in G[s]]
```

b) On montre inductivement que la valeur d'une position s vaut $+1$ ssi elle est gagnante pour J_1.

c) La complexité vérifie $c(s) = 1 + \sum_{v \in G[s]} c(v)$, ce qui conduit à une complexité exponentielle en le nombre de sommets.

Pour obtenir une complexité linéaire, on utiliser la mémoïsation afin de ne calculer la valeur qu'une seule fois pour chaque sommet (qui est un descendant du sommet initial) ; on utilise donc ici un dictionnaire pour y stocker les valeurs concernées :

```
def minmax(G,S1,V1,s) :
    dico = {}
    def aux(s) :     ### procédure récursive qui définit dico[s] s'il ne l'est déjà
         if (s in dico) :  return dico[s]
```

```
        if G[s] == [] :
            if s in finalJ1 :  dico[s] = 1
            else :  dico[s] = -1
        if S1[s] :
            for t in G[s] :  aux(t)
            dico[s] = max([dico[t] for t in G[s]]
        else :
            for t in G[s] :  aux(t)
            dico[s] = min([dico[t] for t in G[s]]
    return dico[s]
```

d)

```
def minmaxNim(s) :
    n = s[0] ; G = arene(n)
    dico[(0,1)] = 1 ; dico[(0,2)] = -1
    for k in range(1,n+1) :
        s = (k,1)
        dico[s] = max([dico[t] for t in G[s]]
        s = (k,2)
        dico[t] = min([dico[t] for t in G[s]]
   return dico[s]
```

5) a) Soit $s \in A$ une position gagnante pour le joueur J_1. On a $m(s) = 0$ si $s \in F_1$, et sinon,

$$\begin{cases} \text{si } s \in S_1,\ m(s) = \min\{1 + m(t),\ t \in G[s] \cap A\} \\ \text{si } s \in S_2,\ m(s) = \max\{1 + m(t),\ t \in G[s]\} \end{cases}$$

Remarque : Lorsque $s \in S_2$, tous les successeurs appartiennent à A.

b) Le plus simple est d'utiliser l'attracteur construit à la question 3).

```
def nombres(G,S1,F1) :
    A = attracteur(G,S1,F1) ; M = {}
    if s in A and not(s in F1) :
        if S1[s] :
           L = []
           for t in G[s] :
               if (t in A) : L.append(t)
                   M[s] = min([1+M[t] for t in L])
```

```
        else
            M[s] = max([1+M[t] for t in G[s]]
    return M
```

c)

```
def strategieGagnante(G,S1,V1) :
    A = attracteurBis(G,S1,F1) ; Sg = {(0,1):(0,1)}
    for s in A :
        if S1[s] :
            for t in G[s] :
                if A[s] = 1+A[t] :  Sg[s] = t
    return Sg
```

d) Les valeurs successives de A sont

clé	(0,1)	(1,2)	(2,1)	(3,1)	(3,2)	(4,1)
valeur	0					
valeur	0	1				
valeur	0	1	2	2		
valeur	0	1	2	2	3	4

Donc `Sg[(2,1):(1,2),(2,1):(1,2),(3,1):(1,2),(4,1):(3,2)]`