```python
# TP_minimax.py

## TP algorithme bassin d'attraction


def Graphe_Nim(n):
    """
    def Graphe_Nim(n:int)->
Tuple[List[int],List[List[int]],List[str]]:

    prend un entier n en entrée et retourne un tuple contenant
3 listes :
    La première liste est une liste d'entiers qui représente
les sommets du graphe.
    La seconde liste est une liste de listes d'entiers qui
représente les arêtes du graphe ou les fils de chaque noeud de
l'arbre.
    La troisième liste est une liste de chaînes de caractères
qui représente les étiquettes des sommets ou des noeuds.

    La fonction permet de créer un graphe pour le jeu de Nim,
où chaque état est défini par un couple (i,j) où i est le nombre de
bâtons restants et j est le joueur qui doit jouer ("0" ou "1").
    """
    S=[0]
    A=[]
    E=[str(n)+";0"]
    i=0
    while i < len(S):
        s=E[i]
        batons,joueur=s.split(";")
        batons=int(batons)
        joueur_suivant = "1" if joueur=="0" else "0"
        for k in [1,2,3]:
            if batons-k >=1:
                etiq=str(batons-k)+";"+joueur_suivant
                if etiq in E:
                    j=E.index(etiq)
                    A.append([i,j])
                else :
                    S.append(len(S))
                    E.append(etiq)
                    A.append([i,len(S)-1])
        i=i+1
    return S,A,E



def Arbre_Nim(n):
    """
    def Arbre_Nim(n:int)->
Tuple[List[int],List[List[int]],List[str]]:

    prend un entier n en entrée et retourne un tuple contenant
```

```
3 listes :
045|     La première liste est une liste d'entiers qui représente
les sommets de l'arbre.
046|     La seconde liste est une liste de listes d'entiers qui
représente les arêtes du graphe ou les fils de chaque noeud de
l'arbre.
047|     La troisième liste est une liste de chaînes de caractères
qui représente les étiquettes des sommets ou des noeuds.
048|
049|     La fonction permet de créer un arbre pour le jeu de Nim,
où chaque état est défini par un couple (i,j) où i est le nombre de
bâtons restants et j est le joueur qui doit jouer ("0" ou "1").
050|     """
051|     S=[0]
052|     A=[]
053|     E=[str(n)+";0"]
054|     i=0
055|     while i < len(S):
056|         s=E[i]
057|         batons,joueur=s.split(";")
058|         batons=int(batons)
059|         joueur_suivant = "1" if joueur=="0" else "0"
060|         for k in [1,2,3]:
061|             if batons-k >=1:
062|                 etiq=str(batons-k)+";"+joueur_suivant
063|                 S.append(len(S))
064|                 E.append(etiq)
065|                 A.append([i,len(S)-1])
066|         i=i+1
067|     return S,A,E
068|
069|
070| def predsuccesseur(S,A):
071|     """
072|     La fonction prend en paramètres :
073|     S: une liste d'entiers, qui correspond aux sommets d'un
graphe
074|     A: une liste de tuples d'entiers (i,j), qui correspondent
aux arêtes d'un graphe
075|
076|     La fonction renvoie un tuple contenant 4 listes:
077|     suc: une liste de listes, où chaque sous-liste contient
les successeurs d'un sommet de S
078|     nbsuc: une liste contenant le nombre de successeurs pour
chaque sommet de S
079|     pred: une liste de listes, où chaque sous-liste contient
les prédécesseurs d'un sommet de S
080|     nbpred: une liste contenant le nombre de prédécesseurs
pour chaque sommet de S.
081|     En utilisant les informations fournies par S et A, cette
fonction permet de construire les listes des successeurs et
prédécesseurs pour chaque sommet du graphe, ainsi que le nombre de
successeurs et prédécesseurs pour chaque           sommet.
082|     """
083|     suc=[[] for i in S]
```

```
084|     nbsuc=[0 for i in S]
085|     pred=[[] for i in S]
086|     nbpred=[0 for i in S]
087|     for (i,j) in A:
088|         suc[i].append(j)
089|         pred[j].append(i)
090|         nbpred[j]+=1
091|         nbsuc[i]+=1
092|     return suc,nbsuc,pred,nbpred
093|
094|
095|
096|
097|
098| def J0_Nim(S,A,E):
099|     S0=[]
100|     F0=[]
101|     for k in range(len(S)):
102|         batons,joueur=E[k].split(";")
103|         if joueur=="0":
104|             S0.append(k)
105|         elif batons=="1":
106|             F0.append(k)
107|     return S0,F0
108|
109|
110| def m_et_p(s,attr,compt,pred,S0):
111|     """
112|     Sous-fonction permettant de marquer et propager le bassin
d'attraction
113|     s : un entier qui représente un sommet dans le bassin
d'attraction
114|     attr : une liste d'entiers qui contient les sommets déjà
dans le bassin d'attraction
115|     compt : une liste d'entiers qui contient le nombre de
prédécesseurs non dans le bassin d'attraction pour chaque sommet
116|     pred : une liste de listes qui contient les prédécesseurs
pour chaque sommet
117|     S0 : une liste d'entiers qui contient les sommets contrôlé
par J0
118|
119|     Elle renvoie Attr et compt mis à jour
120|     """
121|     if s not in attr:
122|         attr.append(s)
123|     for u in pred[s]:
124|         compt[u]=compt[u]-1
125|         if u in S0 or compt[u]==0:
126|             attr,compt=m_et_p(u,attr,compt,pred,S0)
127|     return attr,compt
128|
129| def bassin_attractions(S,A,S0,F0):
130|     """
131|     La fonction prend en paramètres :
132|
```

```
133|     S : une liste de n'importe quel type de données, qui
correspond aux sommets d'un graphe
134|     A : une liste de tuples d'entiers (i,j), qui correspondent
aux arêtes d'un graphe
135|     S0: une liste d'entiers qui contient les sommets contrôlés
par J0
136|     F0: une liste d'entiers qui contient les sommets gagnants
pour J0
137|     La fonction renvoie une liste d'entiers qui contient les
sommets dans le bassin d'attraction. Cette fonction utilise la
fonction "predsuccesseur" pour construire les listes des
successeurs et prédécesseurs pour chaque sommet    du graphe, ainsi
que le nombre de successeurs et prédécesseurs pour chaque sommet.
Elle utilise ensuite la fonction "m_et_p" pour calculer le bassin
d'attraction en parcourant récursivement les prédécesseurs de
chaque sommet de F0. Pour chaque sommet visité, la fonction vérifie
s'il est déjà dans le bassin d'attraction ou non, et décrémente le
compteur de ses prédécesseurs. Si un prédécesseur est dans S0 ou si
son compteur est égal à zéro, alors il est également ajouté au
bassin d'attraction. Le processus se poursuit jusqu'à ce que tous
les prédécesseurs de F0 aient été visités et ajoutés au bassin
d'attraction.
138|
139|     """
140|     suc,nbsuc,pred,nbpred=predsuccesseur(S,A)
141|     Attr=[]
142|     compteur=nbsuc
143|     for x in F0:
144|         Attr,compteur=m_et_p(x,Attr,compteur,pred,S0)
145|     return Attr
146|
147|
148| import random as rd
149|
150| def strat(S,A,E,p):
151|     suc,nbsuc,pred,nbpred=predsuccesseur(S,A)
152|     S0,F0=J0_Nim(S,A,E)
153|     Attr=bassin_attractions(S,A,S0,F0)
154|     L=[]
155|     for k in suc[p]:
156|         if k in Attr:
157|             L.append(k)
158|     if L !=[]:
159|         return rd.choice(L)
160|     return rd.choice(suc[p])
161|
162|
163|
164|
165|
166|
167|
168|
169|
170|
```

```
171|
```