Simulation du Naming Game

Initialisation

In [ ]:
```
from random import sample, choice
import matplotlib.pyplot as plt
from itertools import chain
from numpy import mean, array
```

In [ ]:
```
# Ouverture du fichier contenant le dictionnaire français (22735 mots)
with open("liste_francais.txt", "r") as file :
    mots = file.readlines()
mots = [mot[:-1] for mot in mots]

# Inititalisation des variable de la simulation
def init(N):
    inventaire = { i: set() for i in range(N) }
    lexique = []
    return inventaire,lexique
```

Fonctions du modèle

In [ ]:
```
# Pour inventer un mot on choisit un mot aléatoirement dans le dictionnaire
def invente():
    return sample(mots, k=1)[0]

# Modélisation d'une interaction entre un "speaker" et un "listener" conformément
def interaction(inventaire,lexique):
    speaker,listener = sample(inventaire.keys(), k = 2)
    if not inventaire[speaker] : # inventaire vide
        nouveau_mot = invente()
        if nouveau_mot not in lexique:
            lexique.append(nouveau_mot)
        inventaire[speaker].add(nouveau_mot)
    intersection = inventaire[listener] & inventaire[speaker]
    if not intersection: # pas de mot en commun pour decrire l'objet
        inventaire[listener] = inventaire[speaker] | inventaire[listener]
        return 0 # Echec
    else:
        inventaire[speaker] = inventaire[listener] = intersection
        return 1 # Succès

# On compte le nombre de mot distinct dans l'inventaire
def nb_mots_distincs(inventaire):
    return len(set(list(chain(*[list(mot) for mot in list(inventaire.values())]))))
```

Plotting function

In [ ]:
```
def afficher(N,I, **Y): # Afficher des variables pour un N et un I donnés
    x = [x for x in range(I)]
    color={ var:"#"+''.join([choice('0123456789ABCDEF') for i in range(6)]) for var
```



```
    for var,val in Y.items():
```

```
      ax.plot(x, val, color=color[var], label=var)

    ax.set(title= 'Evolution de la variable durant {} interactions entre {} agents
                ylabel= "variable",
                xlabel= 'Interactions')

    plt.legend()
    plt.show()

def afficher2(N, **Y): # Afficher des indicateurs phenoménologique (es, et, Nwtmax)
    x = N
    color={ var:"#"+''.join([choice('0123456789ABCDEF') for i in range(6)]) for var

    fig, ax = plt.subplots()
    for var,val in Y.items():
      ax.plot(x, val, color=color[var], label=var)
      plt.yticks(val)

    ax.set(title= "Evolution de {} en fonction du nombre d'agents ".format("variabl
    ylabel= "variable",
    xlabel= 'N')

    plt.xticks(x)
    plt.grid()
    plt.legend()
    plt.show()
```

Simulation et constructions des indicateurs phénoménologique

In [ ]:

```
# Simulation pour un N et un I donné pour récupérer des variables d'intérêt

def simulation(N,I):

    # Initialisation de la simulation
    inventaire,lexique = init(N)
    es_atteint = False
    es = 0
    succes = []
    mots_inventés = []
    mots_distincts = []

    for i in range(I):

        # On declenche une interaction et on met à jours les variables d'intérêt po
        interaction(inventaire,lexique)
        mots_inventés.append(len(lexique))
        mots_distincts.append(nb_mots_distincs(inventaire))

        # On cherche a detecter l'etat stationnaire
        if not es_atteint and  ( i >= 10 and mots_distincts[-1] == 1 and mots_disti
            es_atteint = True
            es = i

    # indicateur phénoménologique caractéristiques du modèle pour un nombre d'agent
    nmi = mots_inventés[-1] # nombre de mots inventé au total

    # etat dit "transitoire" i.e : quand on atteint l
```

```
    afficher(N,I,Nwt = mots_inventés, Ndt=mots_distincts)
    return array(mots_distincts)

# et des indicateurs phénoménologique

def simulationbis(N,I):

    # Initialisation de la simulation
    inventaire,lexique = init(N)
    es_atteint = False
    es = 0
    mots_inventés = [0]
    mots_distincts = [0]

    for i in range(I-1):

        # On declenche une interaction et on met à jours les variables d'intérêt pa
        interaction(inventaire,lexique)
        mots_inventés.append(len(lexique))
        mots_distincts.append(nb_mots_distincs(inventaire))

        # On cherche a detecter l'etat stationnaire
        if not es_atteint and  ( i >= 10 and mots_distincts[-1] == 1 and mots_disti
            es_atteint = True
            es = i

    # indicateur phénoménologique caractéristiques du modèle pour un nombre d'agent
    nmi = mots_inventés[-1] # nombre de mots inventé au total
    et = mots_inventés.index(nmi) # etat dit "transitoire" i.e : quand on atteint l

    return nmi, es, et


# Simulation pour un N et un I ainsi qu'un paramètre p qui correspond à la mémoire

def simulation2(N,I, p):
    # Initialisation
    inventaire,lexique = init(N)
    succes = [0 for _ in range(p)]
    taux_succes = [0 for _ in range(p)]

    for i in range(p,I):
        succes.append(interaction(inventaire,lexique))
        taux_succes.append(mean(succes[-p:]))



    # Affichage graphique (à enlever pour gagner en compléxité pour effectuer un ma

    # x = list(range(I))
    # fig, ax = plt.subplots()
    # print(len(taux_succes))
    # ax.plot(x, taux_succes)

    # ax.set(title= "Evolution du taux de succès sur les {} dernières interactions
    # ylabel= "taux de succès",
    # xlabel= 'interactions (temps)')

    # plt.xticks([0, et, es, I-1])
    # taux_succes[et] , taux_succes[es], taux_succes[
```

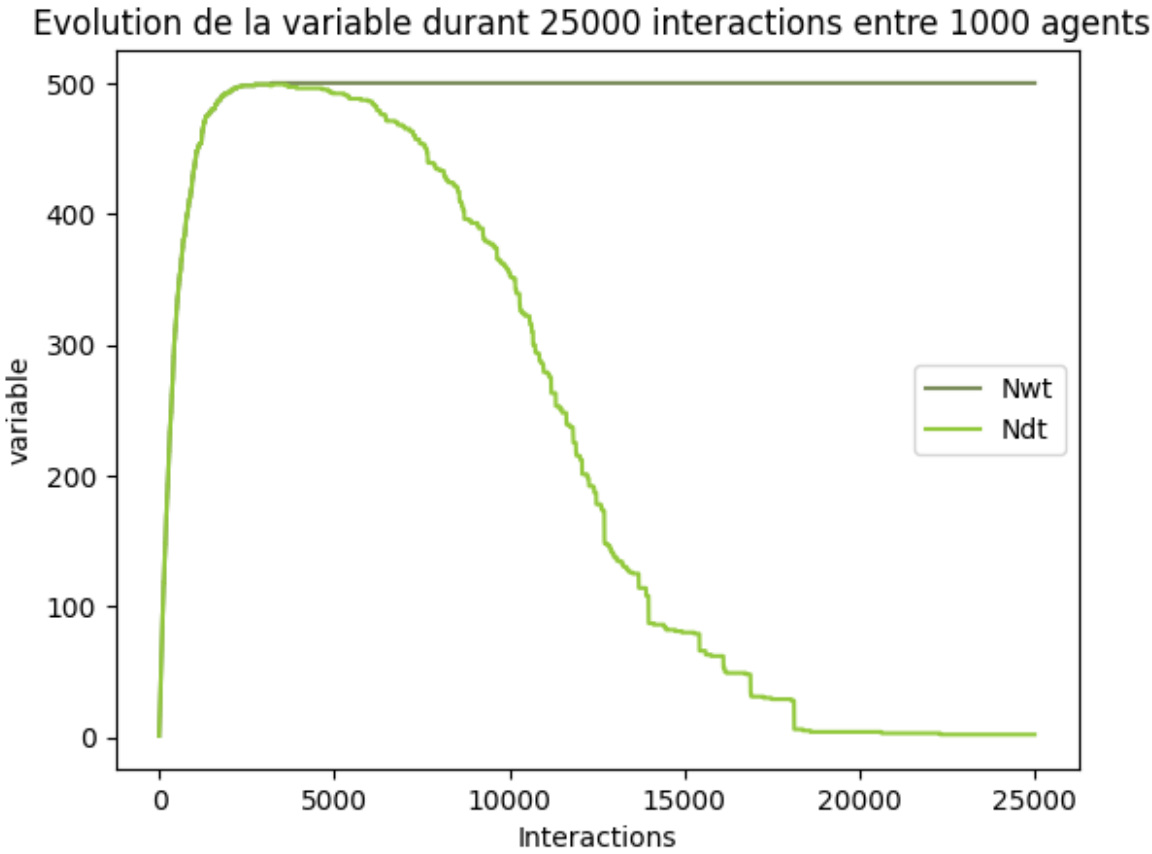
```
    # plt.legend()
    # plt.show()


    return array(taux_succes)

def simulation3(N):
    nb_mots_inventes = []
    etat_stationnaire = []
    etat_transition = []
    taux_succes_final = []
    ecart_et_es = []

    for n in N:
        nmi, es, et = simulation(n)
        nb_mots_inventes.append(nmi)
        etat_stationnaire.append(es)
        etat_transition.append(et)
        ecart_et_es.append(es-et)

    afficher2(N, et = etat_transition, es = etat_stationnaire, ecart = ecart_et_es)
```

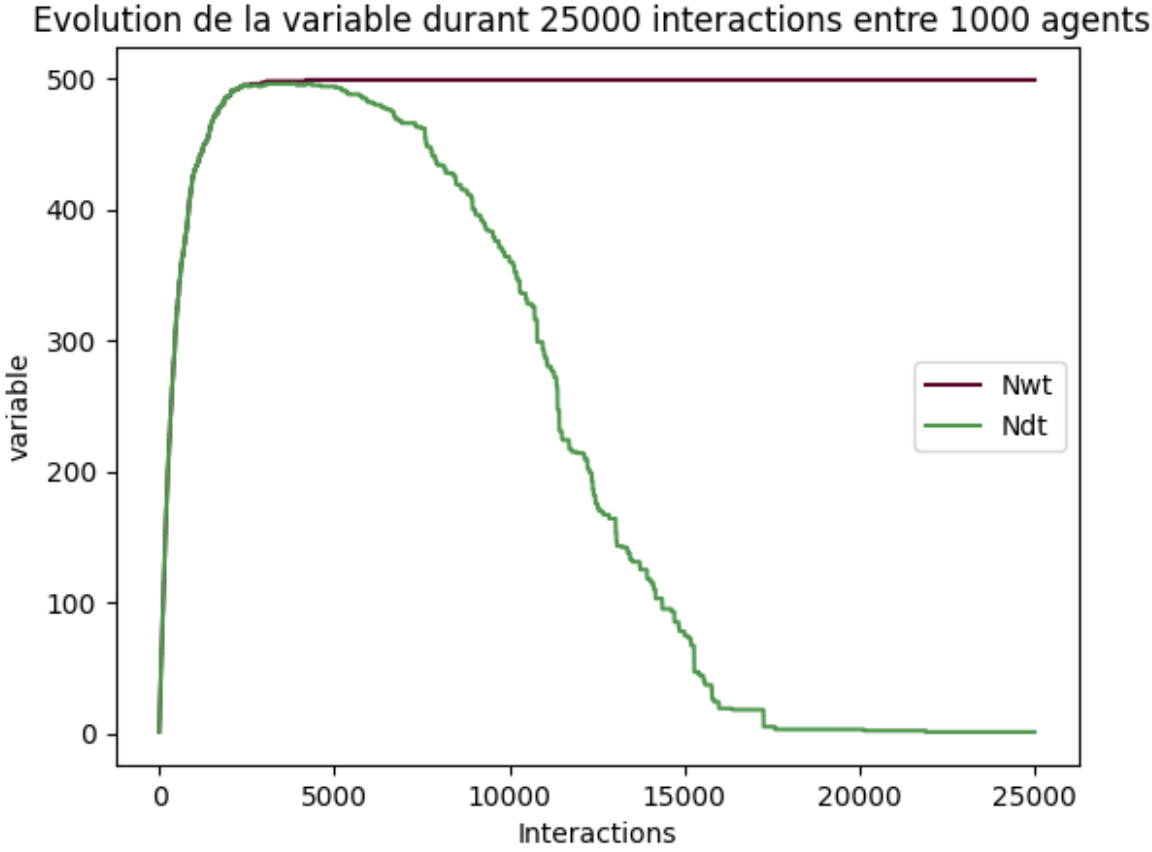
Simulation moyennée sur plusieurs executions

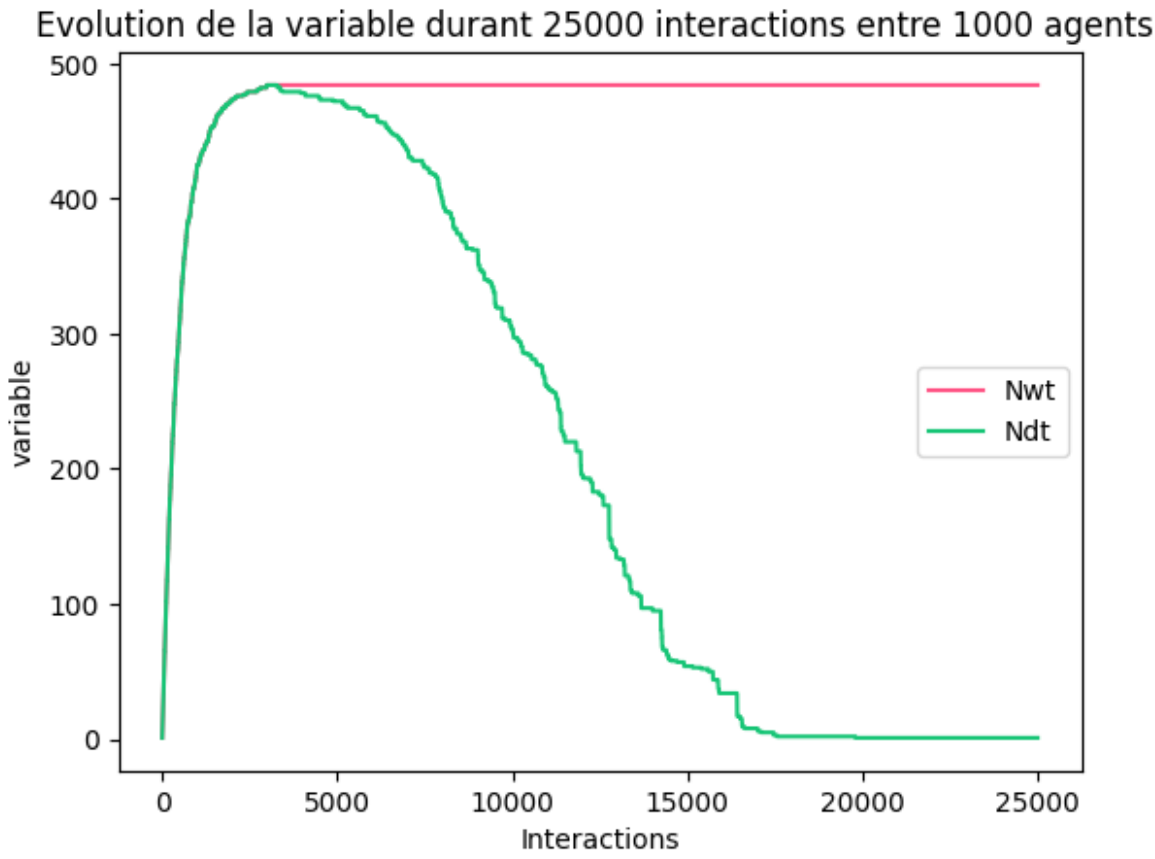
In [ ]:
```
def grande_simulation(p,rep,N):
    I = 25*N
    #ts = array([0.0 for _ in range(I)])
    ts = array([0.0 for _ in range(I)])
    for k in range(rep):
        #ts += simulation2(N, I, p)
        ts += simulation(N, I)

    ts /= rep
    x = list(range(I))
    fig, ax = plt.subplots()

    ax.plot(x, ts)

    ax.set(title= "Evolution du nombre de mots distincts en fonction temps pour {}
    ylabel= "Nombre de mots distincts",
    xlabel= 'interactions (temps)')

    plt.legend()
    plt.show()
```

In [ ]:
```
grande_simulation(1000, rep = 3, N= 1000)
```

```
C:\Users\Bilal\AppData\Local\Temp\ipykernel_9748\1615769403.py:7: DeprecationWarni
ng: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  speaker,listener = sample(inventaire.keys(), k = 2)
```








```
No artists with labels found to put in legend.  Note that artists whose label star
t with an underscore are ignored when legend() is called with no argument.
```