```python
import chess
import chess.engine
import chess.pgn
import time
import math
from tqdm import tqdm
from time import sleep
import json
import pprint
import numpy as np
import matplotlib.pyplot as plt


pgn_file = open("/Users/victor/Desktop/TIPE/parties/fichiertest.pgn")
game_count = 0
game = chess.pgn.read_game(pgn_file)
cpt = 0
engine = chess.engine.SimpleEngine.popen_uci("/opt/homebrew/bin/stockfish")

#première analyse de partie
def analyse (game,temps):
    board = game.board()
    cpt_erreur = 0
    header = game.headers
    list = [0,0,0]
    id=0
    nb_coup = 0
    moy_elo = 0
    if (header.get("WhiteElo") != None) and (header.get("BlackElo") != None) :          #calcul de l'élo moyen de la partie
        moy_elo = (int(header.get("WhiteElo")) + int(header.get("BlackElo")))/2
    if header.get("Event")[0:15]== "Amber-blindfold":          #identification des parties à l'aveugle et des parties rapides (id=1 -> aveugle | id=0 -> rapide)
        id = 1
    instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
    for move in tqdm(game.mainline_moves(), desc="Analyzing moves",total=game.end().ply() ):          #analyse de chaque coup de la partie
        board.push(move)
        instant2 = engine.analyse(board, chess.engine.Limit(time= temps))
        if instant2["score"].relative.score() == None or abs(instant2["score"].relative.score()) >300:
            break
        else :
            nb_coup += 1
            if (abs(instant2["score"].relative.score()) + instant1["score"].relative.score()) > 150 :
                cpt_erreur += 1
                if nb_coup < 20 :
                    list[0] += 1
                elif nb_coup < 80 :
                    list[1] += 1
                else :
                    list[2] += 1
        instant1 = instant2
    return cpt_erreur,list, moy_elo, id, nb_coup

#analyse de partie avec un temps de calcul plus court pour les premiers coups (on saute les premiers coups car ils sont connus par tous les joueurs)

def analyse_bis (game,temps):
    board = game.board()
    cpt_erreur = 0
    header = game.headers
    list = [0,0,0]
    id=0
    nb_coup = 0
    moy_elo = 0
    if (header.get("WhiteElo") != None) and (header.get("BlackElo") != None) : #calcul de l'élo moyen de la partie
        moy_elo = (int(header.get("WhiteElo")) + int(header.get("BlackElo")))/2
    if header.get("Event")[0:15]== "Amber-blindfold": #identification des parties à l'aveugle et des parties rapides (id=1 -> aveugle | id=0 -> rapide)
        id = 1
    instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
    for move in tqdm(game.mainline_moves(), desc="Analyzing moves",total=game.end().ply() ):          #analyse de chaque coup de la partie
        board.push(move)
        nb_coup += 1
        if nb_coup > 5 :
            instant2 = engine.analyse(board, chess.engine.Limit(time= temps))
            if instant2["score"].relative.score() == None or abs(instant2["score"].relative.score()) >300:
                break
            else :
                if (abs(instant2["score"].relative.score()) + instant1["score"].relative.score()) > 150 :
                    cpt_erreur += 1
                    if nb_coup < 20 :
                        list[0] += 1
                    elif nb_coup < 80 :
                        list[1] += 1
                    else :
                        list[2] += 1
            instant1 = instant2
    return cpt_erreur,list, moy_elo, id, nb_coup

#analyse des parties faites sur internet, cela implique qu'un seul joueur joue à l'aveugle donc on ne prends pas en compte les erreurs de l'adversaire
def analyse_partie_internet (game,temps):
    board = game.board()
    cpt_erreur = 0
    header = game.headers
    list = [0,0,0]
    id=1
    nb_coup = 0
    moy_elo = 0
    if (header.get("WhiteElo") != None) and (header.get("BlackElo") != None) :      #calcul de l'élo moyen de la partie
        moy_elo = (int(header.get("WhiteElo")) + int(header.get("BlackElo")))/2
    instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
    for move in tqdm(game.mainline_moves(), desc="Analyzing moves",total=game.end().ply() ):        #analyse de chaque coup de la partie
        board.push(move)
        nb_coup += 1
        if nb_coup > 5 :
            instant2 = engine.analyse(board, chess.engine.Limit(time= temps))
            if instant2["score"].relative.score() == None or abs(instant2["score"].relative.score()) >300:
                break
            else :
                if (abs(instant2["score"].relative.score()) + instant1["score"].relative.score()) > 150 :
                    cpt_erreur += 1
                    if nb_coup < 20 :
                        list[0] += 1
                    elif nb_coup < 80 :
                        list[1] += 1
                    else :
                        list[2] += 1
            instant1 = instant2
    return cpt_erreur,list, moy_elo, id, nb_coup

#permet de lancer l'analyse de plusieurs parties en ne commençant pas par la première
def compte (game_count, pgn_file,dict_parties,current_game,temps):
    for i in tqdm(range(game_count), desc = "nb moves", total = game_count):
        game = chess.pgn.read_game(pgn_file)
        if game is None:
            break
        cpt_erreur,list, moy_elo, id, nb_coup = analyse_partie_internet(game,temps)
        dict_parties[current_game+i] = (cpt_erreur,list, moy_elo, id, nb_coup)

#permet de lancer l'analyse de plusieurs parties en commençant par la première et de les stocker dans un fichier json
def traitement_donnees (game_count):
    pgn_file = open("/Users/victor/Desktop/TIPE/parties/moi.pgn")
    write_file = "/Users/victor/Desktop/TIPE/parties/moi.json"
    dict_parties = {}
    compte(game_count, pgn_file, dict_parties,0,0.1)
    with open(write_file, "w") as write_file:
        json.dump(dict_parties, write_file)
    pgn_file.close()
    read_file = "/Users/victor/Desktop/TIPE/parties/moi.json"
    with open(read_file, "r") as read_file:
```

```
    data = json.load(read_file)
  pprint.pprint(data)



# Fonction servant à récupérer les données stockées dans un fichier json et les analyser
def recup ():
  read_file = "/Users/victor/Desktop/TIPE/parties/chesscom.json"
  with open(read_file, "r") as read_file:
    data = json.load(read_file)
  compte_coups = 0
  cpt_erreur = 0
  pprint.pprint(data)
  for key in data:
    compte_coups = compte_coups + data[key][4]
    cpt_erreur += data[key][1][0] + data[key][1][1]
  return compte_coups, cpt_erreur, cpt_erreur/compte_coups*100

#Fonction permettant de traiter les données stockées dans un fichier json
def traitement_etude_principale ():
  read_file = "/Users/victor/Desktop/TIPE/parties/Parties.json"
  with open(read_file, "r") as read_file:
    data = json.load(read_file)
  nb_blindfold = 0
  nb_rapide = 0
  for key in data:
    if data[key][3] == 1:
      nb_blindfold += 1
    else :
      nb_rapide += 1
  coups_r = np.zeros(nb_rapide)
  erreur_debut_r = np.zeros(nb_rapide)
  erreur_milieu_r = np.zeros(nb_rapide)
  erreur_fin_r = np.zeros(nb_rapide)
  coups_b = np.zeros(nb_blindfold)
  erreur_debut_b = np.zeros(nb_blindfold)
  erreur_milieu_b = np.zeros(nb_blindfold)
  erreur_fin_b = np.zeros(nb_blindfold)
  erreur_par_match_r = np.zeros(nb_rapide)
  erreur_par_match_b = np.zeros(nb_blindfold)
  m_elo_b = np.zeros(nb_blindfold)
  nb_coups_r = 0
  nb_coups_b = 0
  nb_erreur_r = 0
  nb_erreur_b = 0
  i,j = 0,0
  for key in data:
    if data[key][3] == 1:
      coups_b[j] = data[key][4]
      erreur_debut_b[j] = data[key][1][0]
      erreur_milieu_b[j] = data[key][1][1]
      erreur_fin_b[j] = data[key][1][2]
      nb_coups_b += data[key][4]
      nb_erreur_b += data[key][1][0] + data[key][1][1]
      erreur_par_match_b[j] = data[key][0]
      m_elo_b[j] = data[key][2]
      j += 1

    else :
      coups_r[i] = data[key][4]
      erreur_debut_r[i] = data[key][1][0]
      erreur_milieu_r[i] = data[key][1][1]
      erreur_fin_r[i] = data[key][1][2]
      nb_coups_r += data[key][4]
      nb_erreur_r += data[key][1][0] + data[key][1][1]
      erreur_par_match_r[i] = data[key][0]
      i += 1

  coups_r[0] = 12


  #calcul des moyennes pour les parties rapides
  moyen_coup_r = np.mean(coups_r)
  moyen_erreur_debut_r = np.mean(erreur_debut_r)
  moyen_erreur_milieu_r = np.mean(erreur_milieu_r)
  moyen_erreur_fin_r = np.mean(erreur_fin_r)


  #calcul des moyennes pour les parties à l'aveugle

  moyen_coup_b = np.mean(coups_b)
  moyen_erreur_debut_b = np.mean(erreur_debut_b)
  moyen_erreur_milieu_b = np.mean(erreur_milieu_b)
  moyen_erreur_fin_b = np.mean(erreur_fin_b)
  moyen_elo = np.mean(m_elo_b)

  #Calcul des moyennes globales

  moy_err_cp_r = np.mean(erreur_par_match_r/coups_r)
  moy_err_par_match_r = np.mean(erreur_par_match_r)
  moy_err_cp_b = np.mean(erreur_par_match_b/coups_b)
  moy_err_par_match_b = np.mean(erreur_par_match_b)
  print (coups_r)
  print (erreur_par_match_r/coups_r)
  mediane_r = np.median(coups_r)
  mediane_b = np.median(coups_b)
  ecart_type_nb_coup_r = np.std(coups_r)
  ecart_type_nb_coup_b = np.std(coups_b)
  et_erreur_par_match_r = np.std(erreur_par_match_r)
  et_erreur_par_match_b = np.std(erreur_par_match_b)
  et_erreur_par_coup_r = np.std(erreur_debut_r + erreur_milieu_r)
  et_erreur_par_coup_b = np.std(erreur_debut_b + erreur_milieu_b)



  print ("\n \n       Nb coups | Erreur début | Erreur milieu | Erreur fin | Erreur par match | Erreur par coups \n")
  print ("RAPIDE : ", moyen_coup_r, "+-", ecart_type_nb_coup_r, moyen_erreur_debut_r, moyen_erreur_milieu_r, moyen_erreur_fin_r, moy_err_par_match_r,"+-",et_erreur_par_match_r, moy_err_cp_r,"+-",et_erreur_par_coup_r,"\n" )
  print ("BLINDFOLD : ",moyen_coup_b, "+-", ecart_type_nb_coup_b, moyen_erreur_debut_b, moyen_erreur_milieu_b, moyen_erreur_fin_b, moy_err_par_match_b,"+-",et_erreur_par_match_b, moy_err_cp_b,"+-",et_erreur_par_coup_b,"\n" )
  print ("ELO : ", moyen_elo)



def traitement_donnees_parties_internet() :
  read_file = "/Users/victor/Desktop/TIPE/parties/chesscom.json"
  with open(read_file, "r") as read_file:
    data = json.load(read_file)
  nb_blindfold = 0
  for key in data:
    if data[key][3] == 1:
      nb_blindfold += 1
  coups_b = np.zeros(nb_blindfold)
  erreur_debut_b = np.zeros(nb_blindfold)
  erreur_milieu_b = np.zeros(nb_blindfold)
  erreur_fin_b = np.zeros(nb_blindfold)
  erreur_par_match_b = np.zeros(nb_blindfold)
  nb_coups_b = 0
  nb_erreur_b = 0
  m_elo = np.zeros(nb_blindfold)
  j = 0
  for key in data:
    if data[key][3] == 1:
      coups_b[j] = data[key][4]
      erreur_debut_b[j] = data[key][1][0]
      erreur_milieu_b[j] = data[key][1][1]
      erreur_fin_b[j] = data[key][1][2]
      nb_coups_b += data[key][4]
      nb_erreur_b += data[key][1][0] + data[key][1][1]
      erreur_par_match_b[j] = data[key][0]
      m_elo[j] = data[key][2]
      j += 1
  moyen_coups_b = np.mean(coups_b)
  moyen_erreur_debut_b = np.mean(erreur_debut_b)
  moyen_erreur_milieu_b = np.mean(erreur_milieu_b)
  moyen_erreur_fin_b = np.mean(erreur_fin_b)
  moyen_erreur_par_match_b = np.mean(erreur_par_match_b)
  moy_err_cp_b = np.mean(erreur_par_match_b/coups_b)
  et_erreur_par_match_b = np.std(erreur_par_match_b)
  et_erreur_par_coups_b = np.std(erreur_debut_b + erreur_milieu_b)
```

```
  et_coups_b = np.std(coups_b)
  moyen_elo = np.mean(m_elo)

  print ("\n \n       Nb coups | Erreur début | Erreur milieu | Erreur
fin | Erreur par match | Erreur par coups \n")
  print ("BLINDFOLD : ",moyen_coups_b, moyen_erreur_debut_b,
moyen_erreur_milieu_b, moyen_erreur_fin_b, moy_err_cp_b, "\n" )
  print(moyen_erreur_par_match_b)
  print(moyen_elo)
  print ("+-", et_coups_b, et_erreur_par_match_b, et_erreur_par_coups_b)
  print (nb_coups_b, nb_erreur_b, nb_erreur_b/nb_coups_b*100)


def test ():
  read_file1= "/Users/victor/Desktop/TIPE/parties/test.json"
  read_file2= "/Users/victor/Desktop/TIPE/parties/Parties.json"
  with open(read_file1, "r") as read_file1:
    data1 = json.load(read_file1)
  with open(read_file2, "r") as read_file2:
    data2 = json.load(read_file2)
  cpterreur1=0
  cpterreur2=0
  cpt1, cpt2 = 0,0
  for key in data1:
    cpterreur1 += data1[key][1][0] + data1[key][1][1]
    cpt1 += data1[key][4]
  for i in range(100):
    cpterreur2 += data2[str(i)][1][0] + data2[str(i)][1][1]
    cpt2 += data2[str(i)][4]
  pourcentage1 = cpterreur1/cpt1
  pourcentage2 = cpterreur2/cpt2
  return cpterreur1, cpt1, cpterreur2, cpt2, pourcentage1, pourcentage2

def main ():
  game_count= 1000
  traitement_donnees(game_count)
  traitement_etude_principale()
  traitement_donnees_parties_internet()
  test()


main()

engine.quit()
```