

```

1  import chess
2  import chess.engine
3  import chess.pgn
4  import time
5  import math
6  from tqdm import tqdm
7  from time import sleep
8  import json
9  import pprint
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 pgn_file = open("/Users/victor/Desktop/TIPE/parties/fichieretest.pgn")
14 game_count = 0
15 game = chess.pgn.read_game(pgn_file)
16 cpt = 0
17 engine =
chess.engine.SimpleEngine.popen_uci("/opt/homebrew/bin/stockfish")
18
19 #première analyse de partie
20 def analyse (game,temps):
21     board = game.board()
22     cpt_erreur = 0
23     header = game.headers
24     list = [0,0,0]
25     id=0
26     nb_coup = 0
27     moy_elo = 0
28     if (header.get("WhiteElo") != None) and (header.get("BlackElo") !=
None) :         #calcul de l'élo moyen de la partie
29         moy_elo = (int(header.get("WhiteElo")) +
int(header.get("BlackElo")))/2
30     if header.get("Event")[0:15]== "Amber-blindfold":
#identification des parties à l'aveugle et des parties rapides (id=1 ->
aveugle | id=0 -> rapide)
31         id = 1
32     instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
33     for move in tqdm(game.mainline_moves(), desc="Analyzing
moves",total=game.end().ply() ):         #analyse de chaque coup de la partie
34         board.push(move)
35         instant2 = engine.analyse(board, chess.engine.Limit(time= temps))
36         if instant2["score"].relative.score() == None or
abs(instant2["score"].relative.score()) >300:
37             break
38         else :
39             nb_coup += 1
40             if (abs(instant2["score"].relative.score()) +
instant1["score"].relative.score()) > 150 :
41                 cpt_erreur += 1
42             if nb_coup < 20 :
43                 list[0] += 1
44             elif nb_coup < 80 :
45                 list[1] += 1
46             else :
47                 list[2] += 1

```

```

48         instant1 = instant2
49     return cpt_erreur,list, moy_elo, id, nb_coup
50
51 #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)
52
53 def analyse_bis (game,temps):
54     board = game.board()
55     cpt_erreur = 0
56     header = game.headers
57     list = [0,0,0]
58     id=0
59     nb_coup = 0
60     moy_elo = 0
61     if (header.get("WhiteElo") != None) and (header.get("BlackElo") != None) :
#calcul de l'élo moyen de la partie
62         moy_elo = (int(header.get("WhiteElo")) + int(header.get("BlackElo")))/2
63     if header.get("Event")[0:15]== "Amber-blindfold":
#identification des parties à l'aveugle et des parties rapides (id=1 -> aveugle |
id=0 -> rapide)
64         id = 1
65     instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
66     for move in tqdm(game.mainline_moves(), desc="Analyzing
moves",total=game.end().ply() ):         #analyse de chaque coup de la partie
67         board.push(move)
68         nb_coup += 1
69         if nb_coup > 5 :
70             instant2 = engine.analyse(board, chess.engine.Limit(time= temps))
71             if instant2["score"].relative.score() == None or
abs(instant2["score"].relative.score()) >300:
72                 break
73             else :
74                 if (abs(instant2["score"].relative.score()) +
instant1["score"].relative.score()) > 150 :
75                     cpt_erreur += 1
76                     if nb_coup < 20 :
77                         list[0] += 1
78                     elif nb_coup < 80 :
79                         list[1] += 1
80                     else :
81                         list[2] += 1
82                     instant1 = instant2
83     return cpt_erreur,list, moy_elo, id, nb_coup
84
85 #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
86 def analyse_partie_internet (game,temps):
87     board = game.board()
88     cpt_erreur = 0
89     header = game.headers
90     list = [0,0,0]
91     id=1
92     nb_coup = 0

```

```

93     moy_elo = 0
94     if (header.get("WhiteElo") != None) and (header.get("BlackElo") !=
None) :         #calcul de l'élo moyen de la partie
95         moy_elo = (int(header.get("WhiteElo")) +
int(header.get("BlackElo")))/2
96     instant1 = engine.analyse(board, chess.engine.Limit(time= 0.1))
97     for move in tqdm(game.mainline_moves(), desc="Analyzing
moves",total=game.end().ply() ):         #analyse de chaque coup de la partie
98         board.push(move)
99         nb_coup += 1
100         if nb_coup > 5 :
101             instant2 = engine.analyse(board, chess.engine.Limit(time=
temps))
102             if instant2["score"].relative.score() == None or
abs(instant2["score"].relative.score()) >300:
103                 break
104             else :
105                 if (abs(instant2["score"].relative.score()) +
instant1["score"].relative.score()) > 150 :
106                     cpt_erreur += 1
107                     if nb_coup < 20 :
108                         list[0] += 1
109                     elif nb_coup < 80 :
110                         list[1] += 1
111                     else :
112                         list[2] += 1
113                 instant1 = instant2
114     return cpt_erreur,list, moy_elo, id, nb_coup
115
116 #permet de lancer l'analyse de plusieurs parties en ne commençant pas
par la première
117 def compte (game_count, pgn_file,dict_parties,current_game,temps):
118     for i in tqdm(range(game_count), desc = "nb moves", total =
game_count):
119         game = chess.pgn.read_game(pgn_file)
120         if game is None:
121             break
122         cpt_erreur,list, moy_elo, id, nb_coup =
analyse_partie_internet(game,temps)
123         dict_parties[current_game+i] = (cpt_erreur,list, moy_elo, id,
nb_coup)
124
125 #permet de lancer l'analyse de plusieurs parties en commençant par la
première et de les stocker dans un fichier json
126 def traitement_donnees (game_count):
127     pgn_file = open("/Users/victor/Desktop/TIPE/parties/moi.pgn")
128     write_file = "/Users/victor/Desktop/TIPE/parties/moi.json"
129     dict_parties = {}
130     compte(game_count, pgn_file, dict_parties,0,0.1)
131     with open(write_file, "w") as write_file:
132         json.dump(dict_parties, write_file)
133     pgn_file.close()
134     read_file = "/Users/victor/Desktop/TIPE/parties/moi.json"
135     with open(read_file, "r") as read_file:

```

```

136     data = json.load(read_file)
137     pprint.pprint(data)
138
139 # Fonction servant à récupérer les données stockées dans un fichier
140 json et les analyser
141 def recup ():
142     read_file = "/Users/victor/Desktop/TIPE/parties/chesscom.json"
143     with open(read_file, "r") as read_file:
144         data = json.load(read_file)
145     compte_coups = 0
146     cpt_erreur = 0
147     pprint.pprint(data)
148     for key in data:
149         compte_coups = compte_coups + data[key][4]
150         cpt_erreur += data[key][1][0] + data[key][1][1]
151     return compte_coups, cpt_erreur, cpt_erreur/compte_coups*100
152 #Fonction permettant de traiter les données stockées dans un fichier
153 json
154 def traitement_etude_principale ():
155     read_file = "/Users/victor/Desktop/TIPE/parties/Parties.json"
156     with open(read_file, "r") as read_file:
157         data = json.load(read_file)
158     nb_blindfold = 0
159     nb_rapide = 0
160     for key in data:
161         if data[key][3] == 1:
162             nb_blindfold += 1
163         else :
164             nb_rapide += 1
165     coups_r = np.zeros(nb_rapide)
166     erreur_debut_r = np.zeros(nb_rapide)
167     erreur_milieu_r = np.zeros(nb_rapide)
168     erreur_fin_r = np.zeros(nb_rapide)
169     coups_b = np.zeros(nb_blindfold)
170     erreur_debut_b = np.zeros(nb_blindfold)
171     erreur_milieu_b = np.zeros(nb_blindfold)
172     erreur_fin_b = np.zeros(nb_blindfold)
173     erreur_par_match_r = np.zeros(nb_rapide)
174     erreur_par_match_b = np.zeros(nb_blindfold)
175     m_elo_b = np.zeros(nb_blindfold)
176     nb_coups_r = 0
177     nb_coups_b = 0
178     nb_erreur_r = 0
179     nb_erreur_b = 0
180     i,j = 0,0
181     for key in data:
182         if data[key][3] == 1:
183             coups_b[j] = data[key][4]
184             erreur_debut_b[j] = data[key][1][0]

```

```

185             erreur_fin_b[j] = data[key][1][2]
186             nb_coups_b += data[key][4]
187             nb_erreur_b += data[key][1][0] + data[key][1][1]
188             erreur_par_match_b[j] = data[key][0]
189             m_elo_b[j] = data[key][2]
190             j += 1
191
192         else :
193             coups_r[i] = data[key][4]
194             erreur_debut_r[i] = data[key][1][0]
195             erreur_milieu_r[i] = data[key][1][1]
196             erreur_fin_r[i] = data[key][1][2]
197             nb_coups_r += data[key][4]
198             nb_erreur_r += data[key][1][0] + data[key][1][1]
199             erreur_par_match_r[i] = data[key][0]
200             i += 1
201
202     coups_r[0] = 12
203
204
205     #calcul des moyennes pour les parties rapides
206     moyen_coup_r = np.mean(coups_r)
207     moyen_erreur_debut_r = np.mean(erreur_debut_r)
208     moyen_erreur_milieu_r = np.mean(erreur_milieu_r)
209     moyen_erreur_fin_r = np.mean(erreur_fin_r)
210
211     #calcul des moyennes pour les parties à l'aveugle
212
213     moyen_coup_b = np.mean(coups_b)
214     moyen_erreur_debut_b = np.mean(erreur_debut_b)
215     moyen_erreur_milieu_b = np.mean(erreur_milieu_b)
216     moyen_erreur_fin_b = np.mean(erreur_fin_b)
217     moyen_elo = np.mean(m_elo_b)
218
219     #Calcul des moyennes globales
220
221     moy_err_cp_r = np.mean(erreur_par_match_r/coups_r)
222     moy_err_par_match_r = np.mean(erreur_par_match_r)
223     moy_err_cp_b = np.mean(erreur_par_match_b/coups_b)
224     moy_err_par_match_b = np.mean(erreur_par_match_b)
225     print (coups_r)
226     print (erreur_par_match_r/coups_r)
227     mediane_r = np.median(coups_r)
228     mediane_b = np.median(coups_b)
229     ecart_type_nb_coup_r = np.std(coups_r)
230     ecart_type_nb_coup_b = np.std(coups_b)
231     et_erreur_par_match_r = np.std(erreur_par_match_r)
232     et_erreur_par_match_b = np.std(erreur_par_match_b)
233     et_erreur_par_coup_r = np.std(erreur_debut_r + erreur_milieu_r)
234     et_erreur_par_coup_b = np.std(erreur_debut_b + erreur_milieu_b)

```

```

235
236
237
238     print ("\n \n          Nb coups | Erreur début | Erreur milieu | Erreur
239 fin | Erreur par match | Erreur par coups \n")
240     print ("RAPIDE : ", moyen_coup_r, "+-", ecart_type_nb_coup_r,
241 moyen_erreur_debut_r, moyen_erreur_milieu_r, moyen_erreur_fin_r,
242 moy_err_par_match_r,"+-",et_erreur_par_match_r,
243 moy_err_cp_r,"+-",et_erreur_par_coup_r,"\n" )
244     print ("BLINDFOLD : ",moyen_coup_b, "+-", ecart_type_nb_coup_b,
245 moyen_erreur_debut_b, moyen_erreur_milieu_b, moyen_erreur_fin_b,
246 moy_err_par_match_b,"+-",et_erreur_par_match_b,
247 moy_err_cp_b,"+-",et_erreur_par_coup_b,"\n" )
248     print ("ELO : ", moyen_elo)
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
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)

```

```

280     et_coups_b = np.std(coups_b)
281     moyen_elo = np.mean(m_elo)
282
283     print ("\n \n          Nb coups | Erreur début | Erreur milieu | Erreur
fin | Erreur par match | Erreur par coups \n")
284     print ("BLINDFOLD : ",moyen_coups_b, moyen_erreur_debut_b,
moyen_erreur_milieu_b, moyen_erreur_fin_b, moy_err_cp_b, "\n" )
285     print(moyen_erreur_par_match_b)
286     print(moyen_elo)
287     print ("+-", et_coups_b, et_erreur_par_match_b, et_erreur_par_coups_b)
288     print (nb_coups_b, nb_erreur_b, nb_erreur_b/nb_coups_b*100)
289
290 def test ():
291     read_file1= "/Users/victor/Desktop/TIPE/parties/test.json"
292     read_file2= "/Users/victor/Desktop/TIPE/parties/Parties.json"
293     with open(read_file1, "r") as read_file1:
294         data1 = json.load(read_file1)
295     with open(read_file2, "r") as read_file2:
296         data2 = json.load(read_file2)
297     cpterreur1=0
298     cpterreur2=0
299     cpt1, cpt2 = 0,0
300     for key in data1:
301         cpterreur1 += data1[key][1][0] + data1[key][1][1]
302         cpt1 += data1[key][4]
303     for i in range(100):
304         cpterreur2 += data2[str(i)][1][0] + data2[str(i)][1][1]
305         cpt2 += data2[str(i)][4]
306     pourcentage1 = cpterreur1/cpt1
307     pourcentage2 = cpterreur2/cpt2
308     return cpterreur1, cpt1, cpterreur2, cpt2, pourcentage1, pourcentage2
309
310 def main ():
311     game_count= 1000
312     traitement_donnees(game_count)
313     traitement_etude_principale()
314     traitement_donnees_parties_internet()
315     test()
316
317
318
319 engine.quit()
320

```