

Dokumentacja projektowa

Teoria Szachów

Wojciech Babiński
Paweł Cebula

Sekcja 1

Kierunek: Informatyka

Specjalizacja: Programowanie Aplikacji Mobilnych

Wydział Matematyki Stosowanej

Rok akademicki 2021 / 2022

Temat: *Przetwarzanie notacji FEN na układ bierek na szachownicy. Sprawdzanie poprawności zapisu.*

1 Wstęp

FEN to skrót od Forsyth–Edwards Notation i jest to standardowy zapis szachowy dla określenia pozycji gry szachowej. FEN jest oparty na systemie opracowanym przez szkockiego dziennikarza prasowego Davida Forsytha i stał się popularny w XIX wieku; Steven J. Edwards rozszerzył i zmodyfikował go, aby mogło z niego korzystać oprogramowanie szachowe. Celem FEN jest podanie wszystkich niezbędnych informacji do ponownego rozpoczęcia gry od danej pozycji i w przeciwieństwie do Portable Game Notation oznacza tylko jedną pozycję zamiast ruchów, które do niej prowadzą.

FEN ułatwia przetłumaczenie dowolnej pozycji szachowej na pojedynczy wiersz tekstu. Ułatwia to proces odtwarzania pozycji za pomocą komputera i umożliwia graczom udostępnianie i ponowne uruchamianie gier z dowolnego miejsca. Zastępuje to konieczność wysyłania dużych plików PGN i przyspiesza proces dzielenia się pozycjami, nawet gdy ludzie są daleko od siebie.

2 Opis zagadnienia

Zapis FEN składa się wyłącznie z łańcucha znaków ASCII. Łańcuch ten ma sześć różnych pól oddzielonych od siebie znakiem spacji:

- Pozycja bierek

Pierwsze pole reprezentuje rozmieszczenie bierek na szachownicy. Opisuje zawartość każdego pola, zaczynając od ósmego rzędu, a kończąc na pierwszym.

Małe litery opisują czarne bierki. Podobnie jak w PGN,

- „p” oznacza piona,
- „r” wieżę,
- „n” skoczka,
- „b” gońca,
- „q” hetmana,
- „k” króla,

Te same litery są używane dla białych bierek, ale są one pisane wielkimi literami. Puste pola są oznaczone liczbami od jednego do ośmiu, w zależności od tego, ile pustych pól znajduje się między dwoma bierkami.



Zapis linii: r1b1k1nr

Poszczególne linie oddzielane są znakiem „/”.

- Kolor gracza na ruchu

Drugie pole wskazuje, który gracz znajduje się na ruchu. To pole zawsze jest pisane małymi literami. „w” oznacza, że na ruchu znajdują się białe, podczas gdy „b” oznacza, że czarne grają jako następne.

- Możliwość roszad

Trzecie pole mówi, czy gracze mogą wykonać roszadę i po której stronie. Pierwsze litery wskazują dostępność roszady białych, a kolejno małe litery po stronie czarnych.

Litera „k” oznacza, że roszada krótka jest dostępna, a „q” oznacza, że gracz może wykonać roszadę po stronie hetmana. Symbol „-” oznacza, że żadna ze stron nie może wykonać roszady

- Możliwość bicia w przelocie

Jeśli pionek przemieścił się o dwa pola bezpośrednio przed osiągnięciem pozycji, a zatem jest możliwym celem do bicia w przelocie, łańcuch FEN dodaje współrzędne pola za pionem w notacji algebraicznej w swoim czwartym polu. Jeśli żadne cele nie są dostępne do bicia, używany jest symbol „-”.

- Liczba połówek ruchów

Kolejne pole kodu FEN informuje, ile ruchów wykonali obaj gracze od ostatniego ruchu pionkiem lub zbitia pionka – znane przez programistów szachowych jako liczba półruchów. To pole jest przydatne do wymuszenia zasady remisu 50 ruchów. Gdy licznik osiągnie 100 (co pozwala każdemu graczowi na wykonanie 50 ruchów), gra kończy się remisem.

- Liczba pełnych ruchów

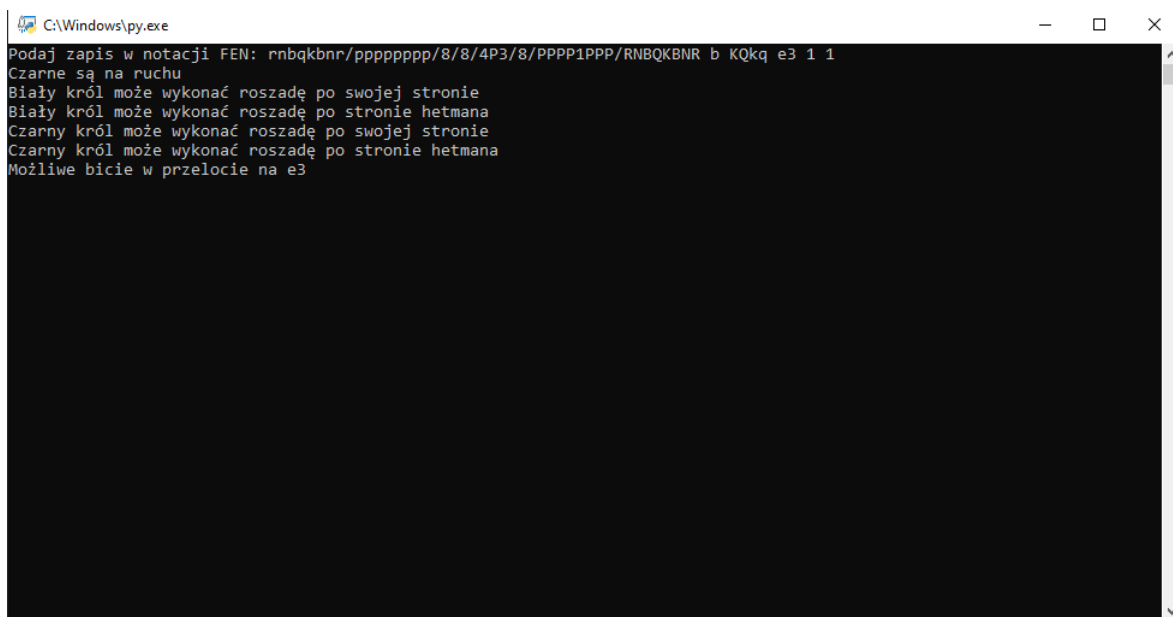
Ostatnie pole kodu FEN pokazuje liczbę ukończonych tur w grze. Ta liczba jest zwiększana o jeden za każdym razem, gdy czarny się porusza. Programiści szachowi nazywają to pełnym ruchem.

3 Opis programu

Program powstał jako projekt zaliczeniowy z przedmiotu Teoria Szchów. Ma za zadanie przetwarzanie notacji FEN na układ bierek na szachownicy. Jednocześnie ma wyłapywać możliwie jak największą liczbę błędów w zadanym zapisie FEN. Do poprawnego działania programu wymagane są:

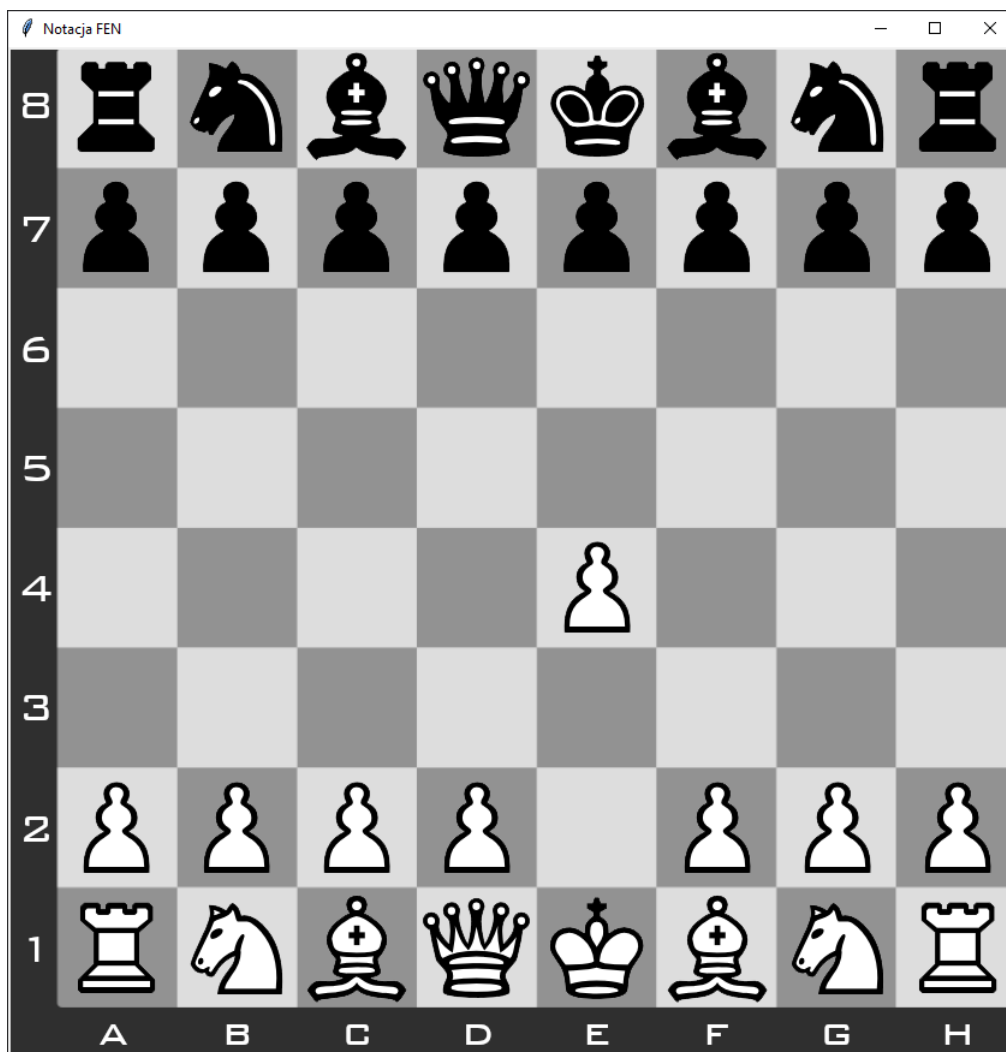
- Python w wersji 3.7
- zainstalowana biblioteka Tkinter

Projekt składa się ze skryptu „fen.py”. Z klawiatury wpisywany jest łańcuch znaków będący zapisem w notacji FEN, który następnie należy zatwierdzić enterem, potwierdzając swój wybór.

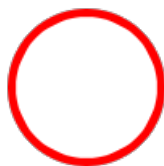


```
C:\Windows\py.exe
Podaj zapis w notacji FEN: rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 1 1
Czarne są na ruchu
Biały król może wykonać roszadę po swojej stronie
Biały król może wykonać roszadę po stronie hetmana
Czarny król może wykonać roszadę po swojej stronie
Czarny król może wykonać roszadę po stronie hetmana
Możliwe bicie w przelocie na e3
```

Następnie, przy pomocy biblioteki tkinter umożliwiającej tworzenie interfejsu graficznego. bierki rozmieszczane są na szachownicy. Sprawdzany jest również szereg warunków w celu zlokalizowania błędów w rozmieszczeniu bierek.



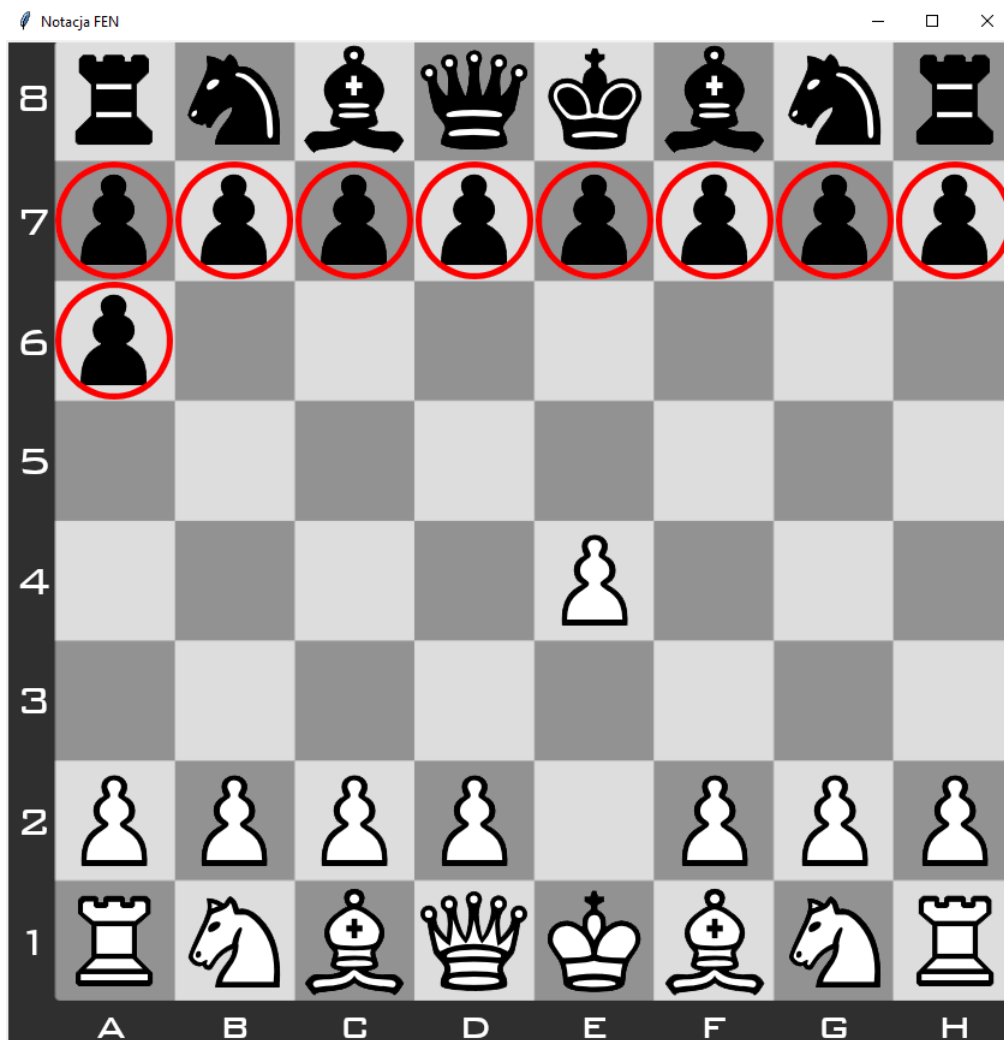
Oznaczenie szachowania w programie



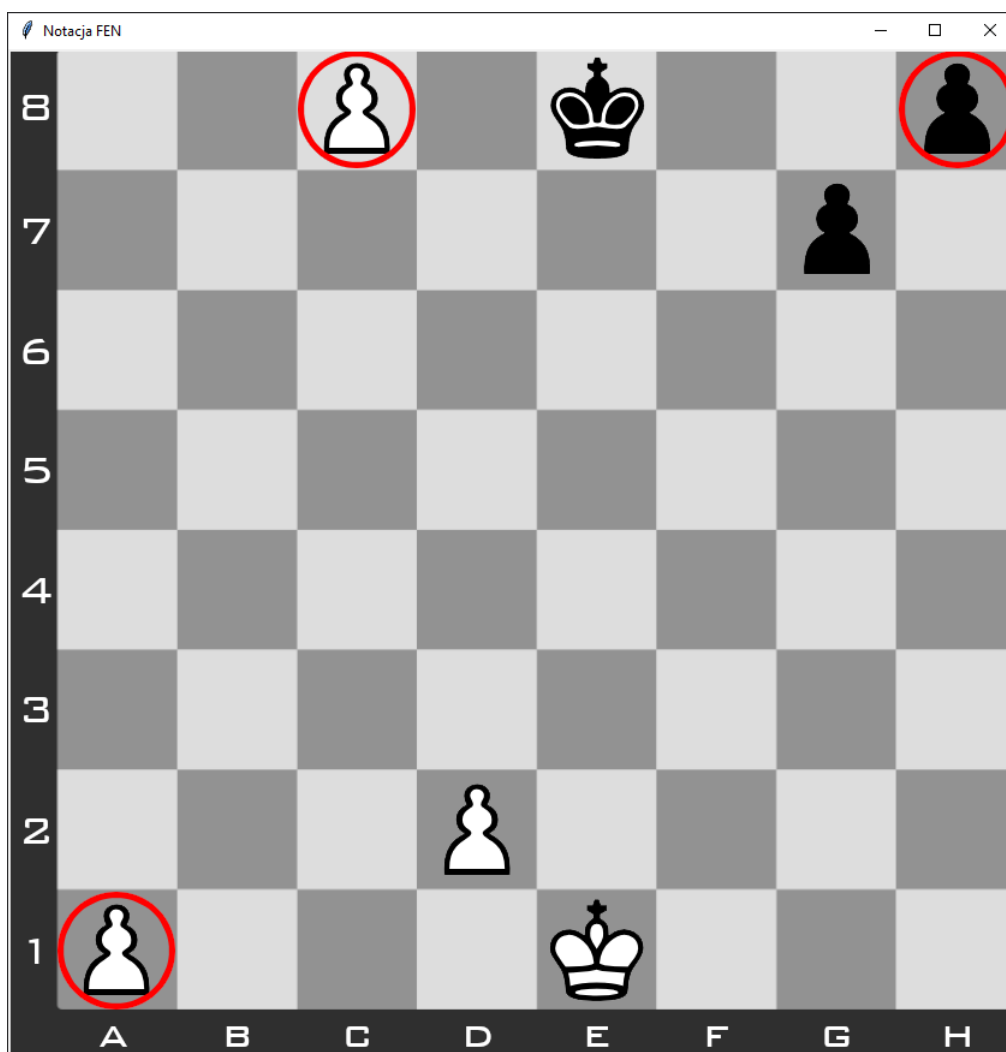
Oznaczenie błędu w programie

4 Opis działania

Program fen.py oczekuje na wpisanie z klawiatury zapisu w notacji FEN, sprawdza czy zapis FEN jest poprawny, a następnie tworzy planszę z podanym ułożeniem bierek. Sprawdzane jest czy rozmiar planszy jest prawidłowy, czy na planszy znajdują się biały oraz czarny król. Następnie liczona jest ilość bierek na szachownicy.

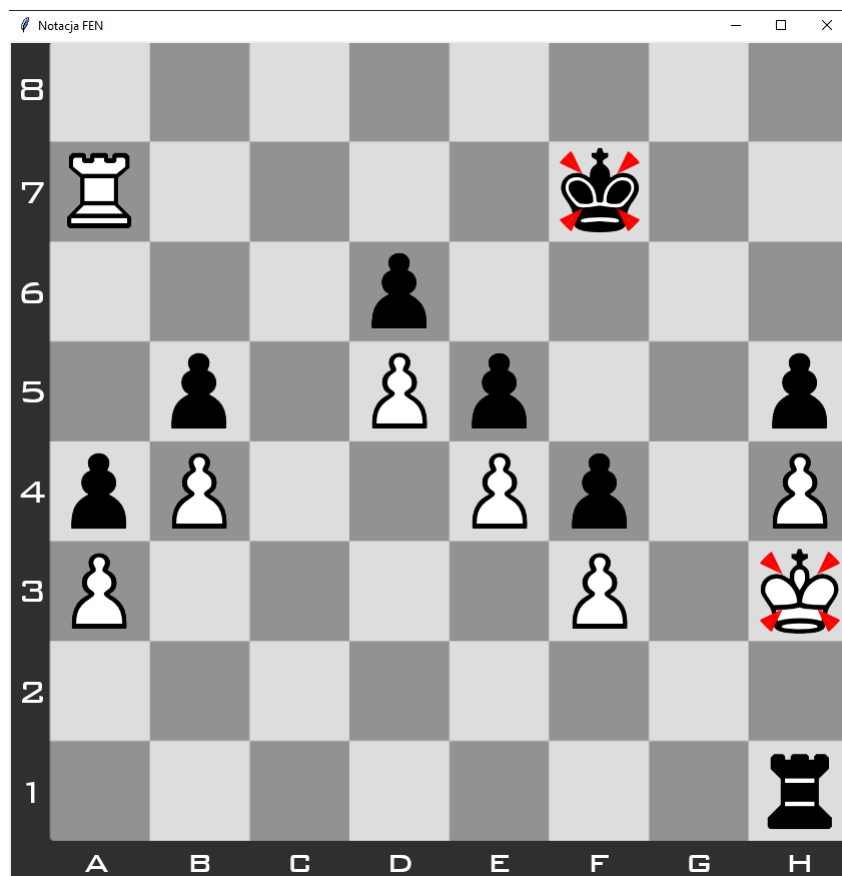


Program sprawdza czy piony nie znajdują się na pierwszej i ostatniej linii.

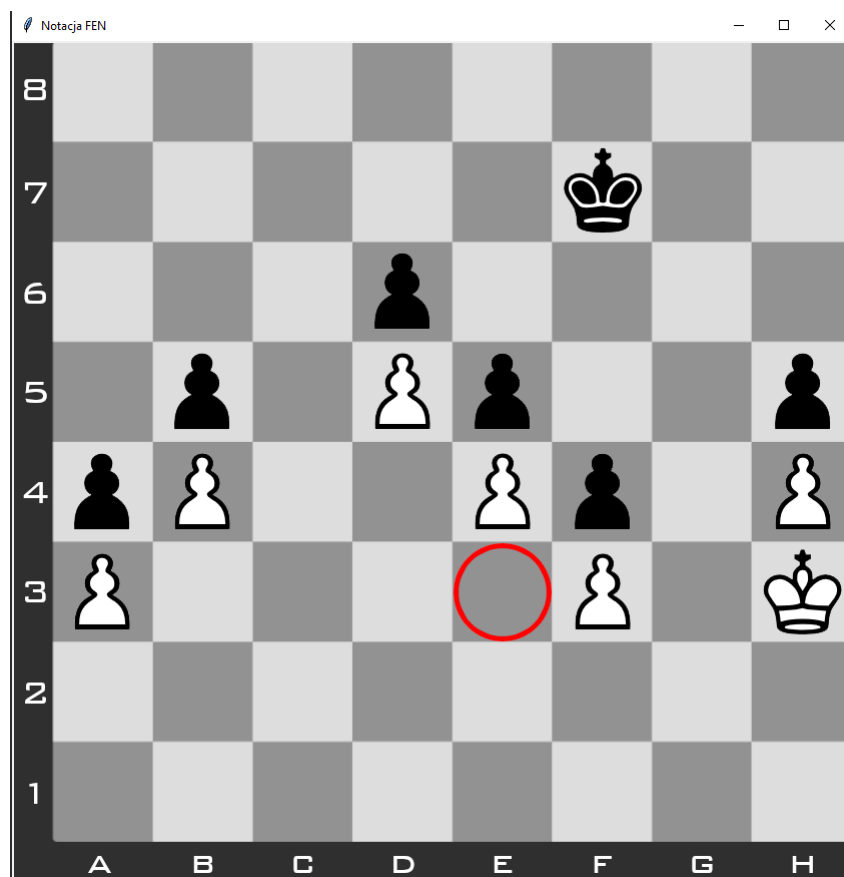
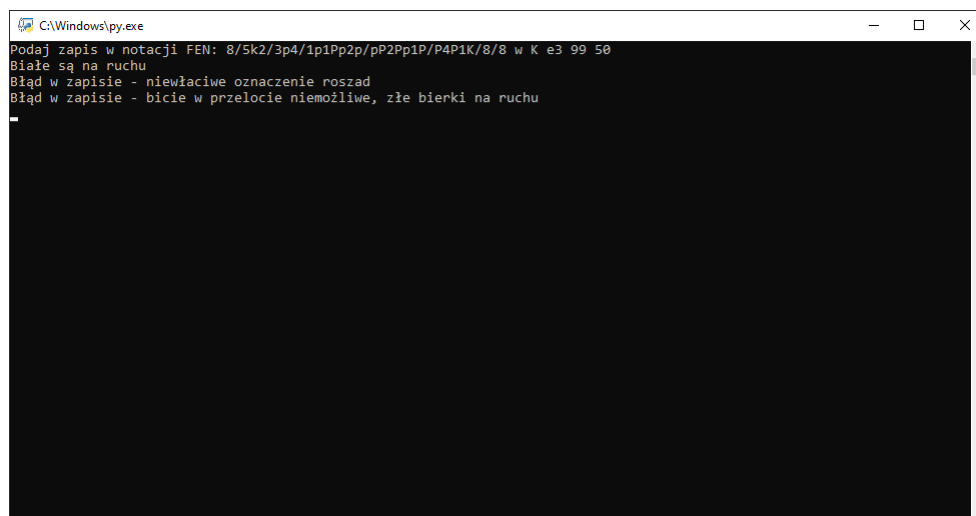


Sprawdza czy króle nie są szachowane w tym samym momencie oraz czy oznaczenie kolejności ruchu w wypadku szacha jest prawidłowe.

```
C:\Windows\py.exe
Podaj zapis w notacji FEN: 8/R4k2/3p4/1p1Pp2p/pP2Pp1P/P4P1K/8/7r b - - 99 50
Błąd w zapisie - obaj króle szachowani
Rozszady nie są możliwe
Brak możliwości bicia w przelocie
```



W dalszej kolejności sprawdza poprawność oznaczenia roszad oraz bicia w przelocie.



Na samym końcu weryfikowana jest ilość pełnych ruchów oraz półruchów.

Jeżeli program wykryje błąd zapisu, poinformuje o tym użytkownika odpowiedni komunikat informujący o błędzie oraz w miarę możliwości zostanie on również oznaczony graficznie na szachownicy.

5 Implementacja

Szachownica reprezentowana jest przez macierz o rozmiarze 8x8 wypełnioną zerami.

$$\mathbf{board} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Bierki w programie posiadają standardowe oznaczenia, gdzie „p” oznacza piona, „r” wieżę, „n” skoczka, „b” goniec, „q” hetmana, a „k” króla dla czarnych bierek oraz dużymi literami dla białych i są podstawiane pod odpowiednie miejsca w macierzy na podstawie pierwszej części łańcucha znaków w notacji FEN. Pozycja startowa wyglądałaby więc następująco.

$$\mathbf{startBoard} = \begin{pmatrix} r & n & b & q & k & b & n & r \\ p & p & p & p & p & p & p & p \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ P & P & P & P & P & P & P & P \\ R & N & B & Q & K & B & N & R \end{pmatrix}$$

Poniżej pseudokody wybranych części algorytmu:

Result: Komunikat o błędzie

```
if halfMoves.isnumeric() and fullMoves.isnumeric() then
    if fullMoves == 1 then
        if fen[1] == 'b' and board == startBoard then
            Komunikat = 'Błąd w zapisie - partię powinny zaczynać białe'
        end
        if correctCastling then
            test = ['K', 'Q', 'k', 'q']
            if test not in castling then
                Komunikat = 'Błąd w zapisie - na początku partii wszystkie opcje
                    roszad powinny być dostępne'
                break;
            end
        end
    end
    else
        if halfMoves > 100 then
            Komunikat = 'Błąd w zapisie - gra już dawno powinna zakończyć się
                remisem'
        end
        if fullMoves < 1 then
            Komunikat = 'Błąd w zapisie - ilość pełnych ruchów powinna wynosić
                minimum 1'
        end
        if (fullPiecesCount == 32) and (halfMoves != 0) then
            Komunikat = 'Błąd w zapisie - brak bicia, liczba pólruchów powinna
                wynosić 0'
        end
        if halfMove / 2 + 1 > fullMoves then
            Komunikat = 'Błąd w zapisie - za dużo pólruchów'
        end
        if (fullPiecesCount != 32) and (halfMoves / 2 + 1 == fullMoves) then
            Komunikat = 'Błąd w zapisie - za dużo pólruchów'
        end
    end
end
else
    Komunikat = 'Błąd w zapisie - zły zapis ilości ruchów'
end
```

Algorithm 1: Sprawdzenie poprawności zapisu pólruchów oraz ruchów

Result: Komunikat o błędzie

```
if len(castling) > 4 or (len(castling) > 1 and '-' in castling) then
    Komunikat = 'Błąd w zapisie - niewłaściwe oznaczenie roszad'
else
    c = Counter(castling)
    if re.match('^kKqQ' * $', castling) and c[max(c)] == 1 then
        if (board[0][4] != 'k' and ('k' in castling or 'q' in castling)) or (board[7][4] !=
            'K' and ('K' in castling or 'Q' in castling)) then
            Komunikat = Błąd w zapisie - niewłaściwe oznaczenie roszad'
        else
            if 'K' in castling then
                if board[7][7] == 'R' then
                    Komunikat = 'Biały król może wykonać roszadę po swojej stronie'
                else
                    Komunikat = 'Błąd w zapisie - niewłaściwe oznaczenie roszad'
                    boardErr[7][7] = 1
                end
            end
            if 'Q' in castling then
                if board[7][0] == 'R' then
                    Komunikat = 'Biały król może wykonać roszadę po stronie hetmana'
                else
                    Komunikat = Błąd w zapisie - niewłaściwe oznaczenie roszad'
                    boardErr[7][0] = 1
                end
            end
            if 'k' in castling then
                if board[0][7] == 'r' then
                    Komunikat = 'Czarny król może wykonać roszadę po swojej stronie'
                else
                    Komunikat = 'Błąd w zapisie - niewłaściwe oznaczenie roszad'
                    boardErr[0][7] = 1
                end
            end
            if 'q' in castling then
                if board[0][0] == 'r' then
                    Komunikat = 'Czarny król może wykonać roszadę po stronie
                        hetmana'
                else
                    Komunikat = 'Błąd w zapisie - niewłaściwe oznaczenie roszad'
                    boardErr[0][0] = 1
                end
            end
        end
    end
    else
        Komunikat = 'Błąd w zapisie - niewłaściwe oznaczenie roszad'
    end
end
```

Algorithm 2: Sprawdzenie poprawności zapisu roszad

6 Kod programu

- fen.py

```
1 import tkinter, re
2 from tkinter import *
3 from collections import Counter
4
5 # -----
6
7 boardImg = 'board.png'
8 pieces = dict()
9 pieces['p'] = 'bp.png'
10 pieces['P'] = 'wp.png'
11 pieces['b'] = 'bb.png'
12 pieces['B'] = 'wb.png'
13 pieces['n'] = 'bn.png'
14 pieces['N'] = 'wn.png'
15 pieces['r'] = 'br.png'
16 pieces['R'] = 'wr.png'
17 pieces['q'] = 'bq.png'
18 pieces['Q'] = 'wq.png'
19 pieces['k'] = 'bk.png'
20 pieces['K'] = 'wk.png'
21 wrong = 'wrong.png'
22 checkImg = 'check.png'
23 correct = TRUE
24 validChars = ['p', 'P', 'b', 'B', 'n', 'N', 'r', 'R', 'q', 'Q', 'k',
                'K']
25 checked = [0, 0]
26 fullPiecesCount = 0
27
28 board = [
29     [0,0,0,0,0,0,0,0],
30     [0,0,0,0,0,0,0,0],
31     [0,0,0,0,0,0,0,0],
32     [0,0,0,0,0,0,0,0],
33     [0,0,0,0,0,0,0,0],
34     [0,0,0,0,0,0,0,0],
35     [0,0,0,0,0,0,0,0],
36     [0,0,0,0,0,0,0,0]
37 ]
38
39 startBoard = [
40     ['r','n','b','q','k','b','n','r'],
41     ['p','p','p','p','p','p','p','p'],
42     [0,0,0,0,0,0,0,0],
43     [0,0,0,0,0,0,0,0],
44     [0,0,0,0,0,0,0,0],
45     [0,0,0,0,0,0,0,0],
46     ['P','P','P','P','P','P','P','P'],
47     ['R','N','B','Q','K','B','N','R']
48 ]
```

```

49
50 boardErr = [
51     [0,0,0,0,0,0,0,0],
52     [0,0,0,0,0,0,0,0],
53     [0,0,0,0,0,0,0,0],
54     [0,0,0,0,0,0,0,0],
55     [0,0,0,0,0,0,0,0],
56     [0,0,0,0,0,0,0,0],
57     [0,0,0,0,0,0,0,0],
58     [0,0,0,0,0,0,0,0]
59 ]
60
61 boardCh = [
62     [0,0,0,0,0,0,0,0],
63     [0,0,0,0,0,0,0,0],
64     [0,0,0,0,0,0,0,0],
65     [0,0,0,0,0,0,0,0],
66     [0,0,0,0,0,0,0,0],
67     [0,0,0,0,0,0,0,0],
68     [0,0,0,0,0,0,0,0],
69     [0,0,0,0,0,0,0,0]
70 ]
71
72 #
-----
73
74 def countPieces():
75     vcs = ['p', 'P', 'bb', 'bw', 'Bb', 'Bw', 'n', 'N', 'r', 'R', 'q',
76           ', 'Q', 'k', 'K']
77     countAll = 0
78     count = dict()
79     count['p'] = 0
80     count['P'] = 0
81     count['bb'] = 0
82     count['bw'] = 0
83     count['Bb'] = 0
84     count['Bw'] = 0
85     count['n'] = 0
86     count['N'] = 0
87     count['r'] = 0
88     count['R'] = 0
89     count['q'] = 0
90     count['Q'] = 0
91     count['k'] = 0
92     count['K'] = 0
93     for i in range(len(board)):
94         for j in range(len(board[i])):
95             if(board[j][i] in validChars):
96                 if board[j][i] in ['b', 'B']:
97                     if (j + i) % 2 == 0:
98                         count[board[j][i] + 'w'] += 1
99                     else:
100                         count[board[j][i] + 'b'] += 1
101             else:

```

```

101         count[board[j][i]] += 1
102         countAll += 1
103     for vc in vcs:
104         #print(vc + ' -> ' + str(count[vc]))
105         if(vc in ['p', 'P'] and count[vc] > 8):
106             for i in range(len(board)):
107                 for j in range(len(board[i])):
108                     if(board[j][i] == vc):
109                         boardErr[j][i] = 1
110         if(vc in ['k', 'K'] and count[vc] != 1):
111             for i in range(len(board)):
112                 for j in range(len(board[i])):
113                     if(board[j][i] == vc):
114                         boardErr[j][i] = 1
115         if(vc in ['n', 'r'] and count[vc] > 2 and count['p'] > 10 -
116             count[vc]):
117             for i in range(len(board)):
118                 for j in range(len(board[i])):
119                     if(board[j][i] == vc):
120                         boardErr[j][i] = 1
121         if(vc in ['N', 'R'] and count[vc] > 2 and count['P'] > 10 -
122             count[vc]):
123             for i in range(len(board)):
124                 for j in range(len(board[i])):
125                     if(board[j][i] == vc):
126                         boardErr[j][i] = 1
127         if(vc in ['q', 'bb', 'bw'] and count[vc] > 1 and count['p']
128             > 9 - count[vc]):
129             for i in range(len(board)):
130                 for j in range(len(board[i])):
131                     if len(vc) == 1 and board[j][i] == vc:
132                         boardErr[j][i] = 1
133                     elif board[j][i] == vc[0] and ((vc[1] == 'w' and
134                         (j + i) % 2 == 0) or (vc[1] == 'b' and (j +
135                         i) % 2 == 1)):
136                         boardErr[j][i] = 1
137         if(vc in ['Q', 'Bb', 'Bw'] and count[vc] > 1 and count['P']
138             > 9 - count[vc]):
139             for i in range(len(board)):
140                 for j in range(len(board[i])):
141                     if len(vc) == 1 and board[j][i] == vc:
142                         boardErr[j][i] = 1
143                     elif board[j][i] == vc[0] and ((vc[1] == 'w' and
144                         (j + i) % 2 == 0) or (vc[1] == 'b' and (j +
145                         i) % 2 == 1)):
146                         boardErr[j][i] = 1
147         if vc in ['k', 'K']:
148             for i in range(len(board)):
149                 for j in range(len(board[i])):
150                     if board[j][i] == vc:
151                         for k in range(i - 1, i + 2):
152                             for l in range(j - 1, j + 2):
153                                 if k in range(0, 7) and l in range
154                                     (0, 7):

```

```

146                                     if (vc == 'k' and board[l][k] ==
                                     'K') or (vc == 'K' and
147                                     board[l][k] == 'k'):
148                                     boardErr[l][k] = 1
149
150     for i in range(len(board[0])):
151         for j in [0, 7]:
152             if board[j][i] in ['p', 'P']:
153                 boardErr[j][i] = 1
154
155     return countAll
156
157 def checkIfChecking(fig, y, x, chk):
158     checkedFig = chk.copy()
159     if fig in ['p', 'n', 'b', 'r', 'q']:
160         target = 'K'
161     elif fig in ['P', 'N', 'B', 'R', 'Q']:
162         target = 'k'
163     if fig in ['p', 'P']:
164         if fig == 'p':
165             xy = [
166                 [x - 1, y + 1],
167                 [x + 1, y + 1]
168             ]
169         else:
170             xy = [
171                 [x - 1, y - 1],
172                 [x + 1, y - 1]
173             ]
174     for ij in xy:
175         j = ij[1]
176         i = ij[0]
177         if(i >= 0 and i <= 7 and j >= 0 and j <= 7):
178             #boardCh[j][i] = 1
179             if board[j][i] == target:
180                 boardCh[j][i] = 1
181                 if target == 'k':
182                     checkedFig[1] = 1
183                 else:
184                     checkedFig[0] = 1
185     elif fig in ['n', 'N']:
186         xy = [
187             [x + 1, y + 2],
188             [x + 2, y + 1],
189             [x + 2, y - 1],
190             [x + 1, y - 2],
191             [x - 1, y - 2],
192             [x - 2, y - 1],
193             [x - 2, y + 1],
194             [x - 1, y + 2]
195         ]
196     for ij in xy:
197         j = ij[1]
198         i = ij[0]
199         if(i >= 0 and i <= 7 and j >= 0 and j <= 7):
200             #boardCh[j][i] = 1
201             if board[j][i] == target:

```



```

199         boardCh[j][i] = 1
200         if target == 'k':
201             checkedFig[1] = 1
202         else:
203             checkedFig[0] = 1
204     elif fig in ['r', 'R']:
205         xy = [
206             [0, 1],
207             [0, -1],
208             [1, 0],
209             [-1, 0]
210         ]
211         for move in xy:
212             j = y + move[1]
213             i = x + move[0]
214             if j in range(8) and i in range(8):
215                 while board[j][i] == 0:
216                     j += move[1]
217                     i += move[0]
218                     if j not in range(8) or i not in range(8):
219                         break
220             if j in range(8) and i in range(8):
221                 if board[j][i] == target:
222                     boardCh[j][i] = 1
223                     if target == 'k':
224                         checkedFig[1] = 1
225                     else:
226                         checkedFig[0] = 1
227     elif fig in ['b', 'B']:
228         xy = [
229             [1, 1],
230             [1, -1],
231             [-1, 1],
232             [-1, -1]
233         ]
234         for move in xy:
235             j = y + move[1]
236             i = x + move[0]
237             if j in range(8) and i in range(8):
238                 while board[j][i] == 0:
239                     j += move[1]
240                     i += move[0]
241                     if j not in range(8) or i not in range(8):
242                         break
243             if j in range(8) and i in range(8):
244                 if board[j][i] == target:
245                     boardCh[j][i] = 1
246                     if target == 'k':
247                         checkedFig[1] = 1
248                     else:
249                         checkedFig[0] = 1
250     elif fig in ['q', 'Q']:
251         xy = [
252             [0, 1],
253             [0, -1],

```

```

254         [1, 0],
255         [-1, 0],
256         [1, 1],
257         [1, -1],
258         [-1, 1],
259         [-1, -1]
260     ]
261     for move in xy:
262         j = y + move[1]
263         i = x + move[0]
264         if j in range(8) and i in range(8):
265             while board[j][i] == 0:
266                 j += move[1]
267                 i += move[0]
268                 if j not in range(8) or i not in range(8):
269                     break
270             if j in range(8) and i in range(8):
271                 if board[j][i] == target:
272                     boardCh[j][i] = 1
273                     if target == 'k':
274                         checkedFig[1] = 1
275                     else:
276                         checkedFig[0] = 1
277     return checkedFig
278
279 #
-----
280
281 fen = input('Podaj zapis w notacji FEN: ').split(' ')
282 if(len(fen) != 6):
283     correct = FALSE
284     print('Blad w zapisie')
285
286 # Ustawienie bierkow
287
288 if(correct):
289     row = fen[0].split('/')
290     if(len(row) != 8):
291         correct = FALSE
292         print('Blad w zapisie - niewlaciwy rozmiar planszy')
293     else:
294         kcmt = 0
295         Kcmt = 0
296         for r in row:
297             if 'k' in r:
298                 kcmt += 1
299             if 'K' in r:
300                 Kcmt += 1
301         if kcmt == 0:
302             correct = FALSE
303             print('Blad w zapisie - brak czarnego krola')
304         if Kcmt == 0:
305             correct = FALSE
306             print('Blad w zapisie - brak bialego krola')

```

```

307         if(correct):
308             for r in range(len(row)):
309                 rowLen = 0
310                 for i in range(len(row[r])):
311                     if(row[r][i] in validChars):
312                         rowLen += 1
313                         board[r][rowLen - 1] = row[r][i]
314                     elif(int(row[r][i]) in [1,2,3,4,5,6,7,8]):
315                         rowLen += int(row[r][i])
316                 if(rowLen != 8):
317                     correct = FALSE
318                     print('Blad w zapisie - niewlaciwy rozmiar
319                           planszy')
320                     break
321     if correct:
322         whitePawns = [
323             [0,0,0,0,0,0,0,0],
324             [0,0,0,0,0,0,0,0],
325             [0,0,0,0,0,0,0,0],
326             [0,0,0,0,0,0,0,0],
327             [0,0,0,0,0,0,0,0],
328             [0,0,0,0,0,0,0,0]
329         ]
330         blackPawns = [
331             [0,0,0,0,0,0,0,0],
332             [0,0,0,0,0,0,0,0],
333             [0,0,0,0,0,0,0,0],
334             [0,0,0,0,0,0,0,0],
335             [0,0,0,0,0,0,0,0],
336             [0,0,0,0,0,0,0,0]
337         ]
338         for i in range(8):
339             for j in range(1, 7):
340                 if board[j][i] == 'P':
341                     whitePawns[6 - j][i] = 1
342         for i in range(8):
343             for j in range(1, 7):
344                 if board[j][i] == 'p':
345                     blackPawns[j - 1][i] = 1
346
347         for i in range(8):
348             for j in range(1, 6):
349                 if whitePawns[j][i] == 1:
350                     start = i - j
351                     if start < 0:
352                         start = 0
353                     stop = j + i
354                     if stop > 7:
355                         stop = 7
356                     test = TRUE
357                     for k in range(start, stop + 1):
358                         if whitePawns[0][k] == 0:
359                             whitePawns[0][k] = str(j) + str(i)
360                             test = FALSE

```

```

361             break
362         if test:
363             boardErr[6 - j][i] = 1
364
365     for i in range(8):
366         for j in range(1, 6):
367             if blackPawns[j][i] == 1:
368                 start = i - j
369                 if start < 0:
370                     start = 0
371                 stop = j + i
372                 if stop > 7:
373                     stop = 7
374                 test = TRUE
375                 for k in range(start, stop + 1):
376                     if blackPawns[0][k] == 0:
377                         blackPawns[0][k] = str(j) + str(i)
378                         test = FALSE
379                         break
380             if test:
381                 boardErr[1 + j][i] = 1
382
383 # Ruch
384
385 if(correct):
386     fullPiecesCount = countPieces()
387     for i in range(len(board)):
388         for j in range(len(board[i])):
389             checked = checkIfChecking(board[j][i], j, i, checked)
390     if checked[0] == checked[1] == 1:
391         print('Blad w zapisie - obaj krole szachowani')
392     elif (fen[1] == 'w' and checked[1] == 1) or (fen[1] == 'b' and
393           checked[0] == 1):
394         print('Blad w zapisie - niewlasciwe oznaczenie kolejnosci
395               ruchu')
396     elif(fen[1] == 'w'):
397         print('Biale sa na ruchu')
398     elif(fen[1] == 'b'):
399         print('Czarne sa na ruchu')
400     else:
401         print('Blad w zapisie - niewlasciwe oznaczenie kolejnosci
402               ruchu')
403
404 # Roszady
405
406 if(correct):
407     castling = fen[2]
408     if(len(castling) > 4 or (len(castling) > 1 and '-' in castling))
409     :
410         print('Blad w zapisie - niewlasciwe oznaczenie roszad')
411     elif(castling == '-'):
412         print('Roszady nie sa mozliwe')
413     else:
414         c = Counter(castling)
415         if(re.match('[kKqQ]*$', castling) and c[max(c)] == 1):

```

```

412         if((board[0][4] != 'k' and ('k' in castling or 'q' in
413             castling)) or (board[7][4] != 'K' and ('K' in
414                 castling or 'Q' in castling))):
415             print('Blad w zapisie - niewlasciwe oznaczenie
416                 roszad')
417         else:
418             if('K' in castling):
419                 if board[7][7] == 'R':
420                     print('Bialy krol moze wykonac roszade po
421                         swojej stronie')
422                 else:
423                     print('Blad w zapisie - niewlasciwe
424                         oznaczenie roszad')
425                     boardErr[7][7] = 1
426             if('Q' in castling):
427                 if board[7][0] == 'R':
428                     print('Bialy krol moze wykonac roszade po
429                         stronie hetmana')
430                 else:
431                     print('Blad w zapisie - niewlasciwe
432                         oznaczenie roszad')
433                     boardErr[7][0] = 1
434             if('k' in castling):
435                 if board[0][7] == 'r':
436                     print('Czarny krol moze wykonac roszade po
437                         swojej stronie')
438                 else:
439                     print('Blad w zapisie - niewlasciwe
440                         oznaczenie roszad')
441                     boardErr[0][7] = 1
442             if('q' in castling):
443                 if board[0][0] == 'r':
444                     print('Czarny krol moze wykonac roszade po
445                         stronie hetmana')
446                 else:
447                     print('Blad w zapisie - niewlasciwe
448                         oznaczenie roszad')
449                     boardErr[0][0] = 1
450         else:
451             print('Blad w zapisie - niewlasciwe oznaczenie roszad')
452
453     # Bicie w przelocie
454
455     def colNum(col):
456         return {
457             'a': 0,
458             'b': 1,
459             'c': 2,
460             'd': 3,
461             'e': 4,
462             'f': 5,
463             'g': 6,
464             'h': 7
465         }[col]

```

```

456 if(correct):
457     enPasTar = fen[3]
458     if(enPasTar in ['a6', 'b6', 'c6', 'd6', 'e6', 'f6', 'g6', 'h6',
459         'a3', 'b3', 'c3', 'd3', 'e3', 'f3', 'g3', 'h3']):
460         if(enPasTar[1] == '6'):
461             eptStart = 1
462             eptItr = 1
463             eptFig = 'p'
464         else:
465             eptStart = 6
466             eptItr = -1
467             eptFig = 'P'
468     if (fen[1] == 'w' and enPasTar[1] == '6') or (fen[1] == 'b'
469         and enPasTar[1] == '3'):
470         if(board[eptStart][colNum(enPasTar[0])] == board[
471             eptStart + eptItr][colNum(enPasTar[0])] == 0 and
472             board[eptStart + eptItr * 2][colNum(enPasTar[0])] ==
473             eptFig):
474             print('Mozliwe bicie w przelocie na ' + enPasTar)
475         else:
476             print('Blad w zapisie - bicie w przelocie niemozliwe
477                 ')
478             boardErr[eptStart + eptItr][colNum(enPasTar[0])] = 1
479         else:
480             print('Blad w zapisie - bicie w przelocie niemozliwe,
481                 zle bierki na ruchu')
482             boardErr[eptStart + eptItr][colNum(enPasTar[0])] = 1
483     elif(enPasTar == '-'):
484         print('Brak mozliwosci bicia w przelocie')
485     else:
486         print('Blad w zapisie - niewlasciwe oznaczenie bicia w
487             przelocie')
488
489 # Poltury i tury
490
491 if correct:
492     halfMoves = fen[4]
493     fullMoves = fen[5]
494
495     if halfMoves.isnumeric() and fullMoves.isnumeric():
496         if int(fullMoves) == 1:
497             if fen[1] == 'b' and board == startBoard:
498                 print('Blad w zapisie - partir powinny zaczynac
499                     biale')
500             if correctCastling:
501                 test = ['K', 'Q', 'k', 'q']
502                 for m in range(4):
503                     if test[m] not in castling:
504                         print('Blad w zapisie - na poczatku partii
505                             wszystkie opcje roszad powinny byc
506                             dostepne')
507                         break
508             else:
509                 if int(halfMoves) > 100:

```

```

499         print('Bład w zapisie - gra juz dawno powinna
              zakonczyc sie remisem')
500     if int(fullMoves) < 1:
501         print('Bład w zapisie - ilosc pelnych ruchow powinna
              wynosic minimum 1')
502     if fullPiecesCount == 32 and int(halfMoves) != 0:
503         print('Bład w zapisie - brak bicia, liczba polruchow
              powinna wynosic 0')
504     elif int(int(halfMoves) / 2) + 1 > int(fullMoves):
505         print('Bład w zapisie - za duzo polruchow')
506     if fullPiecesCount != 32 and (int(halfMoves) / 2) + 1 ==
        int(fullMoves):
507         print('Bład w zapisie - za duzo polruchow')
508
509     else:
510         correct = FALSE
511         print('Bład w zapisie - zly zapis ilosci ruchow')
512
513
514 #

```

```

515
516 if(correct):
517     root = tkinter.Tk()
518     root.geometry("840x840")
519     root.title('Notacja FEN')
520     root.lift()
521     root.wm_attributes("-topmost", True)
522
523     canvas = Canvas(root, width = 840, height = 840)
524     canvas.pack()
525     bg = PhotoImage(file = boardImg)
526     canvas.create_image(420, 420, image = bg)
527     img = []
528     for i in range(len(board)):
529         for j in range(len(board[i])):
530             if(board[j][i] != 0):
531                 img.append(PhotoImage(file = pieces[board[j][i]]))
532                 canvas.create_image(100 * i + 90, 100 * j + 50,
                    image = img[-1])
533
534     for i in range(len(boardErr)):
535         for j in range(len(boardErr[i])):
536             if(boardErr[j][i] == 1):
537                 img.append(PhotoImage(file = wrong))
538                 canvas.create_image(100 * i + 90, 100 * j + 50,
                    image = img[-1])
539     for i in range(len(boardCh)):
540         for j in range(len(boardCh[i])):
541             if(boardCh[j][i] == 1):
542                 img.append(PhotoImage(file = checkImg))
543                 canvas.create_image(100 * i + 90, 100 * j + 50,
                    image = img[-1])
544

```

545 `root.mainloop()`
