Szachy c++

Opis kodu

Spis treści:

1. Funkcja main().

W pliku main.cpp znajduję się funkcja main(). Na początku funkcji tworzony jest obiekt QApplication z biblioteki QApplication. Następnie tworzę obiekt klasy Szachy.h i wywołujemy na nim metodę show().

```cpp
#include "szachy.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Szachy w;
    w.show();
    return a.exec();
}
```

Rys 1 Funkcja main()

2. Klasa akcje

```
#ifndef AKCJE_H
#define AKCJE_H

#include "szachy.h"

class akcje: public QMainWindow
{
public:
    akcje();

    bool szach_bialy(std::vector<int> wektor);
    bool szach_czarny(std::vector<int> wektor);

    static void uzupelnij_biale(void);
    static void uzupelnij_czarne(void);


    static void pionek(bool strona, int wybor);
    static void wieza(bool strona, int wybor);
    static void kon(bool strona, int wybor);
    static void laufer(bool strona, int wybor);
    static void dama(bool strona, int wybor);
    static void krol(bool strona, int wybor);

    //   bool szach = false;
    static std::vector<int> atak_bialy;
    static std::vector<int> atak_czarny;
};

#endif // AKCJE_H
```

Rys 2.1 Zawartość pliku akcje.h

W klasie akcje deklarujemy w części publicznej metody:

- **szach_bialy() -** metoda ta przyjmuje jako argument wektor liczb całkowitych, zwraca natomiast liczbę typu boolowskiego.
Dla danego wektora opisującego położenie figur na planszy, sprawdza czy białemu królowi w danej chwili grozi szach.

```
bool akcje::szach_bialy(std::vector<int> wektor)
{
    int indeks=-1;
    for(int i=63; i>=0; i--){
        if(Szachy::tablica[i]==6){
            indeks=i;
            break;
        }
    }


    bool szach=0;
    //VVV
    for(int i=indeks+8; i<64; i=i+8){
        if(wektor[i]==8 || wektor[i]==11)
            szach=1;
        else if(wektor[i]!=0)
            break;
    }

    //^^^
    for(int i=indeks-8; i>=0; i-=8){
        if(wektor[i]==8 || wektor[i]==11)
            szach=1;
        else if(wektor[i]!=0)
            break;
    }

    //<<<
    if(indeks%8!=0)
    for(int i=indeks-1; i%8!=7; i--){
        if(wektor[i]==8 || wektor[i]==11)
            szach=1;
        else if(wektor[i]!=0)
            break;
    }
```

Rys 2.2 Fragment metody szach_bialy().

- szach_czarny() - metoda ta przyjmuje jako argument wektor liczb całkowitych, zwraca natomiast liczbę typu boolowskiego.
Dla danego wektora opisującego położenie figur na planszy, sprawdza czy czarnemu królowi w danej chwili grozi szach.

```
132 bool akcje::szach_czarny(std::vector<int> wektor)
133 {
134     int indeks=-1;
135     for(int i=63; i>=0; i--){
136         if(Szachy::tablica[i]==12){
137             indeks=i;
138             break;
139         }
140     }
141
142
143     bool szach=0;
144     //VVV
145     for(int i=indeks+8; i<64; i=i+8){
146         if(wektor[i]==2 || wektor[i]==5)
147             szach=1;
148         else if(wektor[i]!=0)
149             break;
150     }
151
152     //^^^
153     for(int i=indeks-8; i>=0; i-=8){
154         if(wektor[i]==2 || wektor[i]==5)
155             szach=1;
156         else if(wektor[i]!=0)
157             break;
158     }
159
160     //<<<
161     if(indeks%8!=0)
162         for(int i=indeks-1; i%8!=7; i--){
163             if(wektor[i]==2 || wektor[i]==5)
164                 szach=1;
165             else if(wektor[i]!=0)
166                 break;
167         }
```

Rys 2.3 Fragment metody szach_czarny().

- **uzupelnij_biale() –** statyczna metoda bezparametrowa, która na początku zeruje wartości wektora atak_bialy, następnie dla każdej białej figury wywołuje odpowiednią metodę ustawiające pola wektora atak_bialy.

```
257 void akcje::uzupelnij_biale(void)
258 {
259     for(int i=0; i<64; i++)
260     {
261         atak_bialy[i]=0;
262     }
263
264     for(int i=0; i<64; i++){
265         if(Szachy::tablica[i]!=0 && Szachy::tablica[i]<7){
266             switch(Szachy::tablica[i]){
267             case 1:
268                 pionek(0, i);
269                 break;
270             case 2:
271                 wieza(0, i);
272                 break;
273             case 3:
274                 kon(0, i);
275                 break;
276             case 4:
277                 laufer(0, i);
278                 break;
279             case 5:
280                 dama(0, i);
281                 break;
282             case 6:
283                 krol(0, i);
284                 break;
285             }
286         }
287     }
288 }
```

Rys 2.4 Fragment metody uzupelnij_biale().

- uzupelnij_czarne() - statyczna metoda bezparametrowa, która na początku zeruje wartości wektora atak_czarny, następnie dla każdej czarnej figury wywołuje odpowiednią metodę ustawiające pola wektora atak_czarny.

```
290 void akcje::uzupelnij_czarne(void)
291 {
292     for(int i=0; i<64; i++){
293         atak_czarny[i]=0;
294     }
295 
296     for(int i=0; i<64; i++){
297         if(Szachy::tablica[i]!=0 && Szachy::tablica[i]>6){
298             switch(Szachy::tablica[i]){
299             case 7:
300                 pionek(1, i);
301                 break;
302             case 8:
303                 wieza(1, i);
304                 break;
305             case 9:
306                 kon(1, i);
307                 break;
308             case 10:
309                 laufer(1, i);
310                 break;
311             case 11:
312                 dama(1, i);
313                 break;
314             case 12:
315                 krol(1, i);
316                 break;
317             }
318         }
319     }
320 }
```

Rys 2.5 Fragment metody uzupelnij_czarne().

- **pionek()** – statyczna metoda przyjmująca dwa parametry: boolowski strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce pionka na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez pionki rywala.

```
322 void akcje::pionek(bool strona, int wybor)
323 {
324     if(!strona){
325         if(wybor%8!=0 && Szachy::tablica[wybor-9]<7)
326             atak_bialy[wybor-9]=2;
327 
328         if(wybor%8!=7 && Szachy::tablica[wybor-7]<7)
329             atak_bialy[wybor-7]=2;
330 
331     }
332     else{
333         if(wybor%8!=7 && (Szachy::tablica[wybor+9]==0 || Szachy::tablica[wybor+9]>6))
334                 atak_czarny[wybor+9]=2;
335 
336         if(wybor%8!=0 && (Szachy::tablica[wybor+7]==0 || Szachy::tablica[wybor+7]>6))
337                 atak_czarny[wybor+7]=2;
338         }
339 
340 }
```

Rys 2.6 Fragment metody pionek().

- **wieza()** - statyczna metoda przyjmująca dwa parametry: boolowski strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce wieży na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez wieże rywala.

```
342 void akcje::wieza(bool strona, int wybor)
343 {
344     if(!strona){
345         //^^^
346         if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
347             if(Szachy::tablica[wybor-8]==0)
348                 atak_bialy[wybor-8]=1;
349             else
350                 atak_bialy[wybor-8]=2;
351 
352             if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
353                 if(Szachy::tablica[wybor-16]==0)
354                     atak_bialy[wybor-16]=1;
355                 else
356                     atak_bialy[wybor-16]=2;
357 
358                 if(wybor>23 && (Szachy::tablica[wybor-24]==0 || Szachy::tablica[wybor-24]>6) && Szachy::tablica[wybor-16]==0){
359                     if(Szachy::tablica[wybor-24]==0)
360                         atak_bialy[wybor-24]=1;
361                     else
362                         atak_bialy[wybor-24]=2;
363 
364                     if(wybor>31 && (Szachy::tablica[wybor-32]==0 || Szachy::tablica[wybor-32]>6) && Szachy::tablica[wybor-24]==0){
365                         if(Szachy::tablica[wybor-32]==0)
366                             atak_bialy[wybor-32]=1;
367                         else
368                             atak_bialy[wybor-32]=2;
```

Rys 2.7 Fragment metody wieza().

- **kon()** – statyczna metoda przyjmująca dwa parametry: boolowski strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce skoczka na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez skoczka rywala.

```
void akcje::kon(bool strona, int wybor)
{
    if(!strona){
        if(wybor>15){
            if(wybor%8!=0 &&  (Szachy::tablica[wybor-17]==0 || Szachy::tablica[wybor-17]>6)){
                if(Szachy::tablica[wybor-17]==0)
                    atak_bialy[wybor-17]=1;
                else if(Szachy::tablica[wybor-17]>6)
                    atak_bialy[wybor-17]=2;
            }

            if(wybor%8!=7 &&  (Szachy::tablica[wybor-15]==0 || Szachy::tablica[wybor-15]>6)){
                if(Szachy::tablica[wybor-15]==0)
                    atak_bialy[wybor-15]=1;
                else if(Szachy::tablica[wybor-15]>6)
                    atak_bialy[wybor-15]=2;
            }
        }

        if(wybor<48){
            if(wybor%8!=0 &&  (Szachy::tablica[wybor+15]==0 || Szachy::tablica[wybor+15]>6)){
                if(Szachy::tablica[wybor+15]==0)
                    atak_bialy[wybor+15]=1;
                else if(Szachy::tablica[wybor+15]>6)
                    atak_bialy[wybor+15]=2;
            }

            if(wybor%8!=7 &&  (Szachy::tablica[wybor+17]==0 || Szachy::tablica[wybor+17]>6)){
                if(Szachy::tablica[wybor+17]==0)
                    atak_bialy[wybor+17]=1;
                else if(Szachy::tablica[wybor+17]>6)
                    atak_bialy[wybor+17]=2;
            }
        }
```

Rys 2.8 Fragment metody kon().

- **laufer()** - statyczna metoda przyjmująca dwa parametry: boolowski strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce laufra na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez laufra rywala.

```
void akcje::laufer(bool strona, int wybor)
{
    if(!strona){
        //^^^<<<
        if(wybor>7 && wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6)){
            if(Szachy::tablica[wybor-9]==0)
                atak_bialy[wybor-9]=1;
            else if(Szachy::tablica[wybor-9]>6)
                atak_bialy[wybor-9]=2;

            if(wybor>15 && wybor%8!=1 && (Szachy::tablica[wybor-18]==0 || Szachy::tablica[wybor-18]>6) && Szachy::tablica[wybor-9]==0){
                if(Szachy::tablica[wybor-18]==0)
                    atak_bialy[wybor-18]=1;
                else if(Szachy::tablica[wybor-18]>6)
                    atak_bialy[wybor-18]=2;

                if(wybor>23 && wybor%8!=2 && (Szachy::tablica[wybor-27]==0 || Szachy::tablica[wybor-27]>6)&& Szachy::tablica[wybor-18]==0){
                    if(Szachy::tablica[wybor-27]==0)
                        atak_bialy[wybor-27]=1;
                    else if(Szachy::tablica[wybor-27]>6)
                        atak_bialy[wybor-27]=2;
```

Rys 2.9 Fragment metody laufer().

- **dama()** - statyczna metoda przyjmująca dwa parametry: boolowski

strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce królowej na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez królową rywala.

```
1355 void akcje::dama(bool strona, int wybor)
1356 {
1357     if(!strona){
1358         //^^^
1359         if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
1360             if(Szachy::tablica[wybor-8]==0)
1361                 atak_bialy[wybor-8]=1;
1362             else
1363                 atak_bialy[wybor-8]=2;
1364 
1365             if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
1366                 if(Szachy::tablica[wybor-16]==0)
1367                     atak_bialy[wybor-16]=1;
1368                 else
1369                     atak_bialy[wybor-16]=2;
1370 
1371                 if(wybor>23 && (Szachy::tablica[wybor-24]==0 || Szachy::tablica[wybor-24]>6) && Szachy::tablica[wybor-16]==0){
1372                     if(Szachy::tablica[wybor-24]==0)
1373                         atak_bialy[wybor-24]=1;
1374                     else
1375                         atak_bialy[wybor-24]=2;
```

Rys 2.10 Fragment metody dama().

- **krol()** - statyczna metoda przyjmująca dwa parametry: boolowski strona określający dla którego gracza ma wykonać tę funkcje oraz liczbę całkowitą wybor określającą miejsce króla na planszy. Metoda ta wyznacza pola, na których figury gracza będącego przy głosie będą zagrożone zbiciem przez króla rywala.

```
2220 void akcje::krol(bool strona, int wybor)
2221 {
2222     if(!strona){
2223         if(wybor>7){
2224             if(Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6){
2225                 if(Szachy::tablica[wybor-8]==0)
2226                     atak_bialy[wybor-8]=1;
2227                 else if(Szachy::tablica[wybor-8]>6)
2228                     atak_bialy[wybor-8]=2;
2229             }
2230 
2231             if(wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6)){
2232                 if(Szachy::tablica[wybor-9]==0)
2233                     atak_bialy[wybor-9]=1;
2234                 else if(Szachy::tablica[wybor-9]>6)
2235                     atak_bialy[wybor-9]=2;
2236             }
2237 
2238             if(wybor%8!=7 && (Szachy::tablica[wybor-7]==0 || Szachy::tablica[wybor-7]>6)){
2239                 if(Szachy::tablica[wybor-7]==0)
2240                     atak_bialy[wybor-7]=1;
2241                 else if(Szachy::tablica[wybor-7]>6)
2242                     atak_bialy[wybor-7]=2;
2243             }
2244 
2245         }
```

Rys 2.11 Fragment metody krol().

Poza wyżej wymienionymi metodami deklarujemy dwa wektory liczb całkowitych:
- **atak_bialy** – statyczny wektor liczb całkowitych (std::vector<int>) zawierający 64 pola odpowiadających polom planszy, początkowo są ustawione na 0. Metody zmieniają wartości konkretnych 0 określając, że pionki białe w następnej kolejce będą mogły w tym miejscu potencjalnie skuć figurę przeciwnika.
- **atak_czarny** - statyczny wektor liczb całkowitych (std::vector<int>) zawierający 64 pola odpowiadających polom planszy, początkowo są ustawione na 0. Metody zmieniają wartości konkretnych 0 określając, że pionki czarne w następnej kolejce będą mogły w tym miejscu potencjalnie skuć figurę przeciwnika.

```
std::vector<int> akcje::atak_bialy(64, 0);
std::vector<int> akcje::atak_czarny(64, 0);
```

Rys 2.12 Definicja wektorów atak_bialy i atak_czarny w pliku akcje.cpp.

3. Klasa figury.

```
#ifndef FIGURY_H
#define FIGURY_H

#include <QMainWindow>
#include <QPainter>
#include <QPainterPath>
#include <vector>
#include "szachy.h"

class Figury: public QMainWindow
{
public:
    Figury();

    void wieza(bool kolei, int wybor);
    void kon(bool kolei, int wybor);
    void laufer(bool kolei, int wybor);
    void dama(bool kolei, int wybor);
    void krol(bool kolei, int wybor);
    void pionek(bool kolei, int wybor);

    void wyzeruj();
    void coijak();

    static std::vector<int> atak;
};

#endif // FIGURY_H
```

Rys 3.1 Zawartość pliku figury.h.

W klasie figury definiujemy następujące metody prywatne:
- **wieza()** – metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się wieża. Metoda ta wyznacza możliwe pola, na które może się przemieścić wieża w danym momencie.

```
void Figury::wieza(bool kolei, int wybor)
{
    if(!kolei){
        //^^^
        if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
            if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
                atak[wybor-8]=1;
            else if(zasady::sprawdz_biale(wybor, wybor-8))
                atak[wybor-8]=2;

            if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
                if(Szachy::tablica[wybor-16]==0 && zasady::sprawdz_biale(wybor, wybor-16))
                    atak[wybor-16]=1;
                else if(zasady::sprawdz_biale(wybor, wybor-16))
                    atak[wybor-16]=2;

                if(wybor>23 && (Szachy::tablica[wybor-24]==0 || Szachy::tablica[wybor-24]>6) && Szachy::tablica[wybor-16]==0){
                    if(Szachy::tablica[wybor-24]==0 && zasady::sprawdz_biale(wybor, wybor-24))
                        atak[wybor-24]=1;
                    else if(zasady::sprawdz_biale(wybor, wybor-24))
                        atak[wybor-24]=2;
```

Rys 3.2 Fragment metody wieza() klasy figury.

- **kon()** – metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się skoczek. Metoda ta wyznacza możliwe pola, na które może się przemieścić skoczek w danym momencie.

```
void Figury::kon(bool kolei, int wybor)
{
    if(!kolei){
        if(wybor>15){
            if(wybor%8!=0 &&  (Szachy::tablica[wybor-17]==0 || Szachy::tablica[wybor-17]>6) && zasady::sprawdz_biale(wybor, wybor-17)){
                if(Szachy::tablica[wybor-17]==0)
                    atak[wybor-17]=1;
                else if(Szachy::tablica[wybor-17]>6)
                    atak[wybor-17]=2;
            }

            if(wybor%8!=7 &&  (Szachy::tablica[wybor-15]==0 || Szachy::tablica[wybor-15]>6) && zasady::sprawdz_biale(wybor, wybor-15)){
                if(Szachy::tablica[wybor-15]==0)
                    atak[wybor-15]=1;
                else if(Szachy::tablica[wybor-15]>6)
                    atak[wybor-15]=2;
            }
        }
```

Rys 3.3 Fragment metody kon() klasy figury.

- **laufer()** – metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się laufer. Metoda ta wyznacza możliwe pola, na które może się przemieścić laufer w danym momencie.

```
558 void Figury::laufer(bool kolei, int wybor)
559 {
560     if(!kolei){
561         //^^^<<<
562         if(wybor>7 && wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6)){
563             if(Szachy::tablica[wybor-9]==0 && zasady::sprawdz_biale(wybor, wybor-9))
564                 atak[wybor-9]=1;
565             else if(Szachy::tablica[wybor-9]>6 && zasady::sprawdz_biale(wybor, wybor-9))
566                 atak[wybor-9]=2;
567 
568             if(wybor>15 && wybor%8!=1 && (Szachy::tablica[wybor-18]==0 || Szachy::tablica[wybor-18]>6) && Szachy::tablica[wybor-9]==0){
569                 if(Szachy::tablica[wybor-18]==0 && zasady::sprawdz_biale(wybor, wybor-18))
570                     atak[wybor-18]=1;
571                 else if(Szachy::tablica[wybor-18]>6 && zasady::sprawdz_biale(wybor, wybor-18))
572                     atak[wybor-18]=2;
573 
574                 if(wybor>23 && wybor%8!=2 && (Szachy::tablica[wybor-27]==0 || Szachy::tablica[wybor-27]>6)&& Szachy::tablica[wybor-18]==0){
575                     if(Szachy::tablica[wybor-27]==0 && zasady::sprawdz_biale(wybor, wybor-27))
576                         atak[wybor-27]=1;
577                     else if(Szachy::tablica[wybor-27]>6 && zasady::sprawdz_biale(wybor, wybor-27))
578                         atak[wybor-27]=2;
```

Rys 3.4 Fragment metody laufer() klasy figury.

- **dama()** – metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się królowa. Metoda ta wyznacza możliwe pola, na które może się przemieścić królówka w danym momencie.

```
1017 void Figury::dama(bool kolei, int wybor)
1018 {
1019     if(!kolei){
1020         //^^^
1021         if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
1022             if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
1023                 atak[wybor-8]=1;
1024             else if(zasady::sprawdz_biale(wybor, wybor-8))
1025                 atak[wybor-8]=2;
1026 
1027             if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
1028                 if(Szachy::tablica[wybor-16]==0 && zasady::sprawdz_biale(wybor, wybor-16))
1029                     atak[wybor-16]=1;
1030                 else if(zasady::sprawdz_biale(wybor, wybor-16))
1031                     atak[wybor-16]=2;
```

Rys 3.5 Fragment metody dama() klasy figury.

- **krol()** - metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się krol. Metoda ta wyznacza możliwe pola, na które może się przemieścić krol w danym momencie.

```
1881 void Figury::krol(bool kolei, int wybor)
1882 {
1883     if(!kolei){
1884 
1885         if(wybor>7){
1886             if((Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6) && zasady::sprawdz_biale(wybor, wybor-8)){
1887                 if(Szachy::tablica[wybor-8]==0)
1888                     atak[wybor-8]=1;
1889                 else if(Szachy::tablica[wybor-8]>6)
1890                     atak[wybor-8]=2;
1891             }
1892 
1893             if(wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6) && zasady::sprawdz_biale(wybor, wybor-9)){
1894                 if(Szachy::tablica[wybor-9]==0)
1895                     atak[wybor-9]=1;
1896                 else if(Szachy::tablica[wybor-9]>6)
1897                     atak[wybor-9]=2;
1898             }
1899 
1900             if(wybor%8!=7 && (Szachy::tablica[wybor-7]==0 || Szachy::tablica[wybor-7]>6) && zasady::sprawdz_biale(wybor, wybor-7)){
1901                 if(Szachy::tablica[wybor-7]==0)
1902                     atak[wybor-7]=1;
1903                 else if(Szachy::tablica[wybor-7]>6)
1904                     atak[wybor-7]=2;
1905             }
1906 
1907         }
```

Rys 3.6 Fragment metody krol() klasy figury.

- **pionek()** – metoda niezwracająca wartości, przyjmująca dwa parametry: boolowski kolei informujący, który z graczy korzysta z metody oraz wybor określający, na którym polu znajduję się pionek. Metoda ta wyznacza możliwe pola, na które może się przemieścić pionek w danym momencie.

```
2022 void Figury::pionek(bool kolei, int wybor)
2023 {
2024     if(!kolei){
2025         if(wybor>47 && wybor<56)
2026         {
2027             if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
2028                 atak[wybor-8]=1;
2029             if(Szachy::tablica[wybor-8]==0 && Szachy::tablica[wybor-16]==0&& zasady::sprawdz_biale(wybor, wybor-16))
2030                 atak[wybor-16]=1;
2031 
2032 
2033             if(wybor%8!=0 && Szachy::tablica[wybor-9]>6 && Szachy::tablica[wybor-9]<13 && zasady::sprawdz_biale(wybor, wybor-9))
2034                 atak[wybor-9]=2;
2035 
2036             if(wybor%8!=7 && Szachy::tablica[wybor-7]>6 && Szachy::tablica[wybor-7]<13&& zasady::sprawdz_biale(wybor, wybor-7))
2037                 atak[wybor-7]=2;
2038 
2039         }
```

Rys 3.7 Fragment metody pionek() klasy figury

- **wyzeruj()** – metoda niezwracająca wartości i bezparametrowa. Zeruje wartości wektora liczb całkowitych atak.

```
2079 void Figury::wyzeruj(){
2080     for(int i=0; i<64; i++)
2081         atak[i]=0;
2082 }
```

Rys 3.8 Fragment metody wyzeruj() klasy figury.

- **coijak()** – metoda niezwracająca wartości i bezparametrowa. Ustawia odpowiednią metodę dla wybranej za pomocą myszki figury na planszy.

```
2084  void Figury::coijak()
2085  {
2086      switch(Szachy::tablica[Szachy::wybor])
2087      {
2088      case 1:
2089          pionek(0, Szachy::wybor);
2090          break;
2091      case 2:
2092          wieza(0, Szachy::wybor);
2093          break;
2094      case 3:
2095          kon(0, Szachy::wybor);
2096          break;
2097      case 4:
2098          laufer(0, Szachy::wybor);
2099          break;
2100      case 5:
2101          dama(0, Szachy::wybor);
2102          break;
2103      case 6:
2104          krol(0, Szachy::wybor);
2105          break;
2106      case 7:
2107          pionek(1, Szachy::wybor);
2108          break;
2109      case 8:
2110          wieza(1, Szachy::wybor);
2111          break;
2112      case 9:
2113          kon(1, Szachy::wybor);
2114          break;
2115      case 10:
2116          laufer(1, Szachy::wybor);
2117          break;
2118      case 11:
2119          dama(1, Szachy::wybor);
2120          break;
```

Rys 3.9 Fragment metody coijak() klasy figury.

W klasie figury poza metodami definiujemy również statyczny wektor liczb całkowitych atak, wypełniony początkowo 64 zerami. Metody

zmieniają wartości pól wektora, które odpowiadają polom planszy określając możliwości poruszania się figurami.

```
2128    std::vector<int> Figury::atak(64, 0);
```

Rys 3.10 Definicja wektora atak w klasie figury.

4. Klasa Szachy

```
18  class Szachy : public QMainWindow
19  {
20      Q_OBJECT
21
22  public:
23      Szachy(QWidget *parent = nullptr);
24      void paintEvent(QPaintEvent *event);
25      void rysujPlansze(void);
26      void rysujFigury(void);
27      void rysujZaznacz(void);
28
29      void mousePressEvent(QMouseEvent *event);
30
31
32      void wieza(bool kolor, int Posx, int Posy);
33      void kon(bool kolor, int Posx, int Posy);
34      void laufer(bool kolor, int Posx, int Posy);
35      void dama(bool kolor, int Posx, int Posy);
36      void krol(bool kolor, int Posx, int Posy);
37      void pionek(bool kolor, int Posx, int Posy);
38
39      void lista(void);
40      void pionek_dotarl(void);
41      void ustaw_za_pionek(int co);
42
43      void rezultat(void);
44
45      static std::vector<int> tablica;
46      static std::vector<int> zbite;
47      static bool kolei;
48      static bool szach;
49      static bool koniec;
50      static int wybor;
51      bool wlacz_liste=false;
52      int wybrana_figure=0;
53
54      ~Szachy();
```

Rys 4.1 Zawartość pliku Szachy.h.

W części publicznej definiujemy metody:

- **paintEvent()** – metoda przyjmująca wskaźnik do eventu QPaintEvent, która jest odpowiedzialna za wyrysowanie planszy i figur na ekranie okna.

```
16  void Szachy::paintEvent(QPaintEvent *event)
17  {
18
19      rysujPlansze();
20      rysujZaznacz();
21      rysujFigury();
22  }
```

Rys 4.2 Metoda paintEvent() w klasie Szachy.

- **rysujPlansze() –** metoda bezparametrowa i niezwracająca wartości, odpowiedzialna za wyrysowanie planszy na ekranie.

```
24  void Szachy::rysujPlansze(void)
25  {
26      QPainter myline(this);
27
28      //Pola legendy
29      QPainter painter(this);
30      QBrush brush(Qt::gray);
31      QRect rect1(280,25, 640, 65);
32      painter.fillRect(rect1, brush);
33      QRect rect2(280, 610, 640, 65);
34      painter.fillRect(rect2, brush);
35      QRect rect3(200, 90, 80, 520);
36      painter.fillRect(rect3, brush);
37      QRect rect4(920, 90, 80, 520);
38      painter.fillRect(rect4, brush);
39
40      //Pola na zbite pionki
41      QBrush brush2(Qt::darkGreen);
42      QRect rect5(40, 123, 80, 454);
43      painter.fillRect(rect5, brush2);
44      QRect rect6(120, 90, 80, 520);
45      painter.fillRect(rect6, brush2);
46      QRect rect7(1080, 123, 80, 454);
47      painter.fillRect(rect7, brush2);
48      QRect rect8(1000, 90, 80, 520);
49      painter.fillRect(rect8, brush2);
```

Rys 4.3 Fragment metody rysujPlansze() w klasie Szachy.

- **rysujFigury() –** metoda bezparametrowa i niezwracająca wartości odpowiedzialna za wyrysowanie figur na polach planszy.

```
374 void Szachy::rysujFigury(void)
375 {
376     int indeks=0;
377 
378     for(int i=0; i<8; i++)
379     {
380         for(int j=0; j<8; j++)
381         {
382             switch(tablica[indeks]){
383             case 0:
384                 break;
385             case 1:
386                 pionek(0, 280+ 80*j, 90+65*i);
387                 break;
388             case 2:
389                 wieza(0, 280+ 80*j, 90+65*i);
390                 break;
391             case 3:
392                 kon(0, 280+ 80*j, 90+65*i);
393                 break;
394             case 4:
395                 laufer(0, 280+ 80*j, 90+65*i);
396                 break;
397             case 5:
398                 dama(0, 280+ 80*j, 90+65*i);
399                 break;
400             case 6:
401                 krol(0, 280+ 80*j, 90+65*i);
402                 break;
403             case 7:
404                 pionek(1, 280+ 80*j, 90+65*i);
405                 break;
406             case 8:
407                 wieza(1, 280+ 80*j, 90+65*i);
408                 break;
```

Rys 4.4 Fragment metody rysujFigury() w klasie Szachy.

- **rysujZaznacz() –** metoda bezparametrowa niezwracająca wartości, odpowiedziana za wyrysowania podkolorowanych pól szachownicy określające możliwości przemieszczania wybranej przez użytkownika figury.

```
1120 void Szachy::rysujZaznacz(void)
1121 {
1122     if(wybor>=0 && wybor<=63)
1123     {
1124         QPainter painter(this);
1125         QBrush brush(Qt::green);
1126         QRect rect1(280+(80*(wybor%8)),90+(65*(wybor/8)), 80, 65);
1127         painter.fillRect(rect1, brush);
1128 
1129         Figury ruchy;
1130         ruchy.wyzeruj();
1131         ruchy.coijak();
1132 
1133         for(int i=0; i<64; i++)
1134         {
1135             if(Figury::atak[i]>0)
1136             {
1137                 QRect rect2(280+(80*(i%8)),90+(65*(i/8)), 80, 65);
1138                 if(Figury::atak[i]==1 && ((akcje::atak_bialy[i]==0 && kolei) || (akcje::atak_czarny[i]==0 && !kolei))){
1139                     QBrush brush1(Qt::blue);
1140                     painter.fillRect(rect2, brush1);
1141                 }
1142                 else if(Figury::atak[i]==2){
1143                     QBrush brush2(Qt::red);
1144                     painter.fillRect(rect2, brush2);
1145                 }
```

Rys 4.5 Fragment metody RysujZaznacz() w klasie Szachy.

- **mousePressEvent() –** metoda odpowiedzialna za odbieranie komunikatów nadawanych przez użytkownika za pomocą myszki.

```
1041 void Szachy::mousePressEvent(QMouseEvent *event)
1042 {
1043     emit Mouse_Pressed();
1044     int indeks, x, y;
1045     QPointF point = event->position();//localPos();
1046
1047     if(!wlacz_liste){
1048
1049     if(point.x()>280 && point.x()<920 &&
1050        point.y()>90 && point.y()<610)
1051     {
1052         x= (point.x()-280)/80;
1053         y= (point.y()-90)/65;
1054         indeks=y*8+x;
1055
1056         if(!kolei && tablica[indeks]>0 && tablica[indeks]<7)
1057         {
1058             wybor=indeks;
1059             update();
1060         }
1061         else if(kolei && tablica[indeks]>6 && tablica[indeks]<13)
1062         {
1063             wybor=indeks;
1064             update();
1065         }
1066
1067         else if(wybor>=0 && Figury::atak[indeks]==1){
1068             tablica[indeks]=tablica[wybor];
1069             tablica[wybor]=0;
1070             if(kolei)
1071                 kolei=0;
1072             else
1073                 kolei=1;
1074             wybor=-1;
1075             zasady::pat_mat(!kolei);
1076             update();
1077         }
```

Rys 4.6 Fragment metody mousePressEvent() klasy Szachy.

- **wieza() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy wieża ma być biała czy czarna oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować wieże na wybranym polu szachownicy.

```
545 void Szachy::wieza(bool kolor, int Posx, int Posy)
546 {
547     QPainter painter(this);
548     QPolygon poly;
549     poly <<QPoint(Posx+15, Posy+55)<<
550 
551         QPoint(Posx+65, Posy+55)<<
552         QPoint(Posx+66, Posy+53)<<
553         QPoint(Posx+65, Posy+50)<<
554         QPoint(Posx+55, Posy+50)<<
555 
556         QPoint(Posx+55, Posy+35)<<
557         QPoint(Posx+50, Posy+32)<<
558         QPoint(Posx+51, Posy+30)<<
559         QPoint(Posx+55, Posy+28)<<
560 
561         QPoint(Posx+55, Posy+20)<<
562         QPoint(Posx+60, Posy+20)<<
563         QPoint(Posx+60, Posy+10)<<
564 
565
```

Rys 4.7 Fragment metody wieza() klasy Szachy.

- **kon() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy skoczek ma być biały czy czarny oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować skoczka na wybranym polu szachownicy.

```
613 void Szachy::kon(bool kolor, int Posx, int Posy)
614 {
615     QPainter painter(this);
616     QPolygon poly;
617     poly <<QPoint(Posx+20, Posy+55)<<
618         QPoint(Posx+40, Posy+55)<<
619         QPoint(Posx+45, Posy+50)<<
620         QPoint(Posx+39, Posy+47)<<
621         QPoint(Posx+36, Posy+45)<<
622         QPoint(Posx+30, Posy+40)<<
623         QPoint(Posx+35, Posy+43)<<
624         QPoint(Posx+38, Posy+41)<<
625         QPoint(Posx+42, Posy+38)<<
626         QPoint(Posx+47, Posy+34)<<
627         QPoint(Posx+52, Posy+30)<<
628         QPoint(Posx+57, Posy+29)<<
629         QPoint(Posx+63, Posy+31)<<
630         QPoint(Posx+67, Posy+32)<<
631         QPoint(Posx+70, Posy+32)<<
632         QPoint(Posx+69, Posy+30)<<
```

Rys 4.8 Fragment metody kon() klasy Szachy.

- **laufer() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy laufer ma być biały czy czarny oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować laufra na wybranym polu szachownicy.

```
693 void Szachy::laufer(bool kolor, int Posx, int Posy)
694 {
695     QPainter painter(this);
696     QPolygon poly;
697     poly <<QPoint(Posx+12, Posy+55)<<
698         QPoint(Posx+68, Posy+55)<<
699         QPoint(Posx+70, Posy+53)<<
700         QPoint(Posx+68, Posy+51)<<
701         QPoint(Posx+45, Posy+50)<<
702
703         QPoint(Posx+50, Posy+40)<<
704         QPoint(Posx+58, Posy+44)<<
705         QPoint(Posx+61, Posy+40)<<
706         QPoint(Posx+58, Posy+35)<<
707         QPoint(Posx+50, Posy+31)<<
708         QPoint(Posx+45, Posy+28)<<
```

Rys 4.9 Fragment metody laufer() klasy Szachy.

- **dama() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy królowa ma być biała czy czarna oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować królową na wybranym polu szachownicy.

```
775 void Szachy::dama(bool kolor, int Posx, int Posy)
776 {
777     QPainter painter(this);
778     QPolygon poly;
779     poly <<QPoint(Posx+25, Posy+55)<<
780         QPoint(Posx+55, Posy+55)<<
781         QPoint(Posx+60, Posy+35)<<
782         QPoint(Posx+69, Posy+15)<<
783         QPoint(Posx+70, Posy+12)<<
784         QPoint(Posx+68, Posy+15)<<
785         QPoint(Posx+55, Posy+38)<<
786
787         QPoint(Posx+51, Posy+35)<<
788 //          QPoint(Posx+50, Posy+28)<<
789         QPoint(Posx+51, Posy+15)<<
790         QPoint(Posx+50, Posy+12)<<
791         QPoint(Posx+49, Posy+10)<<
792         QPoint(Posx+48, Posy+12)<<
793         QPoint(Posx+47, Posy+15)<<
794         QPoint(Posx+45, Posy+35)<<
```

Rys 4.10 Fragment metody dama() klasy Szachy.

- **krol() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy król ma być biały czy czarny oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować króla na wybranym polu szachownicy.

```
849 void Szachy::krol(bool kolor, int Posx, int Posy)
850 {
851     QPainter painter(this);
852     QPolygon poly;
853     poly <<QPoint(Posx+15, Posy+60)<<
854         QPoint(Posx+65, Posy+60)<<
855         QPoint(Posx+63, Posy+56)<<
856         QPoint(Posx+61, Posy+53)<<
857         QPoint(Posx+56, Posy+48)<<
858         QPoint(Posx+52, Posy+46)<<
859         QPoint(Posx+48, Posy+45)<<
860         QPoint(Posx+45, Posy+42)<<
861
862         QPoint(Posx+46, Posy+40)<<
863         QPoint(Posx+47, Posy+39)<<
864         QPoint(Posx+48, Posy+37)<<
865         QPoint(Posx+49, Posy+40)<<
866         QPoint(Posx+48, Posy+37)<<
867         QPoint(Posx+46, Posy+36)<<
868         QPoint(Posx+45, Posy+35)<<
```

Rys 4.11 Fragment metody krol() klasy Szachy.

- **pionek() –** metoda niezwracająca wartości, która przyjmuję 3 parametry: boolowski kolor określający czy pionek ma być biały czy czarny oraz dwie zmienne całkowite określające pozycje na planszy. Dzięki tej metodzie można narysować pionek na wybranym polu szachownicy.

```
973 void Szachy::pionek(bool kolor, int Posx, int Posy)
974 {
975     QPainter painter(this);
976     QPolygon poly;
977
978     QPainter oko(this);
979
980     if(kolor){
981         oko.setBrush(Qt::black);
982         oko.drawEllipse(Posx+33, Posy+17, 15, 15);
983
984 }
985
986     poly <<QPoint(Posx+20, Posy+55)<<
987         QPoint(Posx+60, Posy+55)<<
988         QPoint(Posx+59, Posy+52)<<
989         QPoint(Posx+57, Posy+51)<<
990         QPoint(Posx+56, Posy+50)<<
991         QPoint(Posx+55, Posy+50)<<
992
993         QPoint(Posx+52, Posy+50)<<
994         QPoint(Posx+48, Posy+42)<<
995         QPoint(Posx+44, Posy+34)<<
996         QPoint(Posx+42, Posy+26)<<
997         QPoint(Posx+41, Posy+18)<<
998
```

Rys 4.12 Fragment metody pionek() klasy Szachy.

- **lista() –** metoda bezparametrowa i niezwracająca wartości, która ma za zadanie wyświetlić listę z wyborami zamiany pionka na figurę.

```
1174 void Szachy::lista(void)
1175 {
1176     QPainter painter(this);
1177     QBrush brush(Qt::darkBlue);
1178     QBrush brush2(Qt::darkCyan);
1179     QBrush brush3(Qt::darkYellow);
1180
1181     QRect tlo(1000,155, 180, 390);
1182     painter.fillRect(tlo, brush3);
1183
1184     QRect rect1(1000,170, 180, 360);
1185     painter.fillRect(rect1, brush);
1186
1187     QRect rect2(1000, 260, 180, 90);
1188     QRect rect3(1000, 440, 180, 90);
1189     painter.fillRect(rect2, brush2);
1190     painter.fillRect(rect3, brush2);
1191
```

Rys 4.13 Fragment metody lista klasy Szachy.

- **pionek_dotarl() –** metoda bezparametrowa i niezwracająca wartości. W metodzie tej sprawdzane jest czy pionki dotarły do końca planszy i można je zamienić na lepsze figury.

```
1156 void Szachy::pionek_dotarl(void)
1157 {
1158     for(int i=0; i<8; i++){
1159         if(tablica[i]==1){
1160             wlacz_liste=true;
1161             wybrana_figure = i;
1162         }
1163     }
1164
1165     for(int i=56; i<64; i++){
1166         if(tablica[i]==7){
1167             wlacz_liste=true;
1168             wybrana_figure = i;
1169         }
1170     }
1171
1172 }
```

Rys 4.14 Metoda pionek_dotarl klasy Szachy.

- **ustaw_za_pionek() –** metoda bezparametrowa i niezwracająca wartości. W metodzie tej pionek na końcu planszy jest zamieniona na wybraną przez użytkownika figurę.

```
1201 void Szachy::ustaw_za_pionek(int co)
1202 {
1203     if(wybrana_figure<8){
1204         switch(co){
1205         case 1:
1206             tablica[wybrana_figure] = 5;
1207             wybrana_figure = -1;
1208             wlacz_liste = false;
1209             update();
1210             break;
1211         case 2:
1212             tablica[wybrana_figure] = 2;
1213             wybrana_figure = -1;
1214             wlacz_liste = false;
1215             update();
1216             break;
1217         case 3:
1218             tablica[wybrana_figure] = 4;
1219             wybrana_figure = -1;
1220             wlacz_liste = false;
1221             update();
1222             break;
1223         case 4:
1224             tablica[wybrana_figure] = 3;
1225             wybrana_figure = -1;
1226             wlacz_liste = false;
1227             update();
1228             break;
1229         }
1230     }
```

Rys 4.15 Fragment metody ustaw_za_pionek() klasy Szachy.

- **rezultat() -** metoda bezparametrowa i niezwracająca wartości. Dzięki metodzie tej zostaje narysowana na ekranie plansza końcowa z werdyktem na temat rozgrywki.

```
1261  void Szachy::rezultat(void)
1262  {
1263      QPainter painter(this);
1264      QBrush brush(Qt::darkRed);
1265      QRect rect1(200, 610, 800, 100);
1266      painter.fillRect(rect1, brush);
1267
1268      QBrush brush2(Qt::white);
1269      QRect rect2(250, 635, 700, 50);
1270      painter.fillRect(rect2, brush2);
1271
1272      QPainter gracze(this);
1273      gracze.setPen(Qt::black);
1274      gracze.setFont(QFont("Times", 20));
1275
1276      if(zasady::koniec == 1)
1277          gracze.drawText(QRect(250, 635, 700, 50), Qt::AlignCenter, "Szach-mat. Gracz 2 Wygrywa.");
1278      else if(zasady::koniec == 0)
1279          gracze.drawText(QRect(250, 635, 700, 50), Qt::AlignCenter, "Sytuacja patowa. Remis.");
1280      else if(zasady::koniec == 2)
1281          gracze.drawText(QRect(250, 635, 700, 50), Qt::AlignCenter, "Szach-mat. Gracz 1 Wygrywa.");
1282  }
```

Rys 4.16 Metoda rezultat() klasy Szachy.

Poza metodami w części publicznej definiujemy również pola:

- **tablica** – statyczny wektor liczb całkowitych przechowujący 64 liczby. Wektor określa rozłożenie figur na planszy.

- **zbite** – statyczny wektor liczb całkowitych przechowujący informacje o zbitych figurach.

- **kolei** – statyczna zmienna boolowska określająca, który gracz obecnie może wykonać ruch.

- **koniec** – statyczna zmienna boolowska określająca czy rozgrywka się zakończyła.

- **wybor** – statyczna zmienna liczba całkowita określająca wybrane pole przez użytkownika.

- **wlacz_liste** – zmienna boolowska określająca czy lista z wyborem zamiany pionka na figurę ma być wyświetlona.

- **wybrana_figure** – zmienna liczba całkowita określająca jaką figurę użytkownik wybrał by zamienić pionka, kiedy dotrze na koniec planszy.

```
1285  bool Szachy::kolei = false;
1286
1287  std::vector<int> Szachy::tablica =
1288  {8, 9, 10, 11, 12, 10, 9, 8,
1289   7, 7, 7, 7, 7, 7, 7, 7,
1290   0, 0, 0, 0, 0, 0, 0, 0,
1291   0, 0, 0, 0, 0, 0, 0, 0,
1292   0, 0, 0, 0, 0, 0, 0, 0,
1293   0, 0, 0, 0, 0, 0, 0, 0,
1294   1, 1, 1, 1, 1, 1, 1, 1,
1295   2, 3, 4, 5, 6, 4, 3, 2};
1296
1297  std::vector<int> Szachy::zbite={};
1298
1299  int Szachy::wybor = -1;
1300
1301  //bool Szachy::szach = false;
1302
1303  bool Szachy::koniec = false;
1304
```

Rys 4.17 Fragment pliku Szachy.cpp.

5. Klasa zasady

```
#ifndef ZASADY_H
#define ZASADY_H

#include "szachy.h"
#include "akcje.h"

class zasady: public QMainWindow
{
public:
    zasady();
    static bool sprawdz_biale(int indeks, int gdzie);
    static bool sprawdz_czarne(int indeks, int gdzie);

    static bool szach_b(std::vector<int> wektor);
    static bool szach_c(std::vector<int> wektor);

    static void pat_mat(bool strona);

    static void czy_biale(void);
    static void czy_czarne(void);

    static void pionek2(bool kolei, int wybor);
    static void wieza2(bool kolei, int wybor);
    static void kon2(bool kolei, int wybor);
    static void laufer2(bool kolei, int wybor);
    static void dama2(bool kolei, int wybor);
    static void krol2(bool kolei, int wybor);

    static std::vector<int> wektor;

    static int licznik;
    static int koniec;

};

#endif // ZASADY_H
```

5.1 Zawartość pliku zasady.h.

W klasie zasady definiujemy publiczne pola:

- **sprawdz_biale()** – statyczna metoda zwracająca zmienną boolowską, posiada dwa parametry będące liczbami całkowitymi: indeks odpowiadający za określenie pola wybranej figury oraz gdzie liczbę określającej miejsce docelowe na szachownicy. Metoda przemieszcza wybraną figurę w wybrane

miejsce i sprawdza czy po modyfikacji następuję zagrożenie dla białego króla.

```
6  bool zasady::sprawdz_biale(int indeks, int gdzie)
7  {
8      for(int i=0; i<64; i++){
9          wektor[i] = Szachy::tablica[i];
10     }
11     int pom = wektor[indeks];
12     wektor[indeks] = 0;
13     wektor[gdzie] = pom;
14
15     return !szach_b(wektor);
16 }
```

Rys 5.2 Metoda sprawdz_biale() klasy zasady.

- **sprawdz_czarne()** – statyczna metoda zwracająca zmienną boolowską, posiada dwa parametry będące liczbami całkowitymi: indeks odpowiadający za określenie pola wybranej figury oraz gdzie liczbę określającej miejsce docelowe na szachownicy. Metoda przemieszcza wybraną figurę w wybrane miejsce i sprawdza czy po modyfikacji następuję zagrożenie dla czarnego króla.

```
18 bool zasady::sprawdz_czarne(int indeks, int gdzie)
19 {
20     for(int i=0; i<64; i++){
21         wektor[i] = Szachy::tablica[i];
22     }
23     int pom = wektor[indeks];
24     wektor[indeks] = 0;
25     wektor[gdzie] = pom;
26
27     return (!szach_c(wektor));
28 }
```

Rys 5.3 Metoda sprawdz_czarne() klasy zasady.

- **szach_b()** – statyczna metoda zwracająca zmienną boolowską, posiada parametr wektor będący wektorem liczb całkowitych. Metoda sprawdza czy dla danych z wektora następuje szach dla białego króla.

```
31  bool zasady::szach_b(std::vector<int> wektor)
32  {
33      int indeks=-1;
34      for(int i=63; i>=0; i--){
35          if(wektor[i]==6){
36              indeks=i;
37              break;
38          }
39      }
40
41      bool szach=0;
42      //VVV
43      for(int i=indeks+8; i<64; i=i+8){
44          if(wektor[i]==8 || wektor[i]==11)
45              szach=1;
46          else if(wektor[i]!=0)
47              break;
48      }
49
50      //^^^
51      for(int i=indeks-8; i>=0; i-=8){
52          if(wektor[i]==8 || wektor[i]==11)
53              szach=1;
54          else if(wektor[i]!=0)
55              break;
56      }
```

Rys 5.4 Metoda szach_b() klasy zasady.

- **szach_c()** – statyczna metoda zwracająca zmienną boolowską, posiada parametr wektor będący wektorem liczb całkowitych. Metoda sprawdza czy dla danych z wektora następuje szach dla czarnego króla.

```
156 bool zasady::szach_c(std::vector<int> wektor)
157 {
158     int indeks=-1;
159     for(int i=63; i>=0; i--){
160         if(wektor[i]==12){
161             indeks=i;
162             break;
163         }
164     }
165
166
167     bool szach=0;
168     //VVV
169     for(int i=indeks+8; i<64; i=i+8){
170         if(wektor[i]==2 || wektor[i]==5)
171             szach=1;
172         else if(wektor[i]!=0)
173             break;
174     }
175
176     //^^^
177     for(int i=indeks-8; i>=0; i-=8){
178         if(wektor[i]==2 || wektor[i]==5)
179             szach=1;
180         else if(wektor[i]!=0)
181             break;
182     }
183
```

Rys 5.5 Metoda szach_c() klasy zasady.

- **pat_mat()** – statyczna metoda niezwracająca wartości, posiada parametr będący zmienną boolowską określający dla którego gracza wykonuje się dana funkcja. Metoda sprawdza czy nie ma pata albo szach-mata.

```
281 void zasady::pat_mat(bool strona)
282 {
283     if(strona)
284     {
285         licznik=0;
286         czy_biale();
287 
288         if(licznik>0)
289             return;
290 
291         int ind=-1;
292         for(int i=0; i<64; i++)
293             if(Szachy::tablica[i]==6)
294                 ind=i;
295 
296         if(sprawdz_czarne(ind, ind))
297             koniec=2;
298 
299         Szachy::koniec=true;
300 
301     }
```

Rys 5.6 Metoda pat_mat() klasy zasady.

- **czy_biale()** – statyczna metoda bezparametrowa i niezwracająca wartości. Metoda sprawdza czy białe figury mają przynajmniej jedną możliwość ruchu.

```
323 void zasady::czy_biale(void)
324 {
325     for(int i=0; i<64; i++){
326         if(licznik>0)
327             break;
328         if(Szachy::tablica[i]!=0 && Szachy::tablica[i]<7){
329             switch(Szachy::tablica[i]){
330             case 1:
331                 pionek2(0, i);
332                 break;
333             case 2:
334                 wieza2(0, i);
335                 break;
336             case 3:
337                 kon2(0, i);
338                 break;
339             case 4:
340                 laufer2(0, i);
341                 break;
342             case 5:
343                 dama2(0, i);
344                 break;
345             case 6:
346                 krol2(0, i);
347                 break;
348             }
```

Rys 5.7 Metoda czy_biale() klasy zasady.

- **czy_czarne()** – statyczna metoda bezparametrowa i niezwracająca wartości. Metoda sprawdza czy czarne figury mają przynajmniej jedną możliwość ruchu.

```
354 void zasady::czy_czarne(void)
355 {
356     for(int i=0; i<64; i++){
357         if(licznik>0)
358             break;
359         if(Szachy::tablica[i]!=0 && Szachy::tablica[i]>6){
360             switch(Szachy::tablica[i]){
361             case 7:
362                 pionek2(1, i);
363                 break;
364             case 8:
365                 wieza2(1, i);
366                 break;
367             case 9:
368                 kon2(1, i);
369                 break;
370             case 10:
371                 laufer2(1, i);
372                 break;
373             case 11:
374                 dama2(1, i);
375                 break;
376             case 12:
377                 krol2(1, i);
378                 break;
379             }
```

Rys 5.8 Metoda czy_czarne() klasy zasady.

- **pionek2()** – statyczna metoda niezwracająca wartości przyjmująca dwa parametry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje pionek. Metoda sprawdza czy pionek na podanym polu może wykonać co najmniej jeden ruch.

```
2406 void zasady::pionek2(bool kolei, int wybor)
2407 {
2408     if(!kolei){
2409         if(wybor>47 && wybor<56)
2410         {
2411             if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
2412                 licznik++;
2413             if(Szachy::tablica[wybor-8]==0 && Szachy::tablica[wybor-16]==0&& zasady::sprawdz_biale(wybor, wybor-16))
2414                 licznik++;
2415
2416
2417             if(wybor%8!=0 && Szachy::tablica[wybor-9]>6 && Szachy::tablica[wybor-9]<13 && zasady::sprawdz_biale(wybor, wybor-9))
2418                 licznik++;
2419
2420             if(wybor%8!=7 && Szachy::tablica[wybor-7]>6 && Szachy::tablica[wybor-7]<13&& zasady::sprawdz_biale(wybor, wybor-7))
2421                 licznik++;
2422
2423         }
```

Rys 5.9 Metoda pionek2() klasy zasady.

- **wieza2()** – statyczna metoda niezwracająca wartości przyjmująca dwa parametry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje wieża. Metoda sprawdza czy wieża na podanym polu może wykonać co najmniej jeden ruch.

```
386 void zasady::wieza2(bool kolei, int wybor)
387 {
388     if(!kolei){
389         //^^^
390         if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
391             if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
392                 licznik++;
393             else if(zasady::sprawdz_biale(wybor, wybor-8))
394                 licznik++;
395 
396             if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
397                 if(Szachy::tablica[wybor-16]==0 && zasady::sprawdz_biale(wybor, wybor-16))
398                     licznik++;
399                 else if(zasady::sprawdz_biale(wybor, wybor-16))
400                     licznik++;
```

Rys 5.10 Metoda wieza2 () klasy zasady.

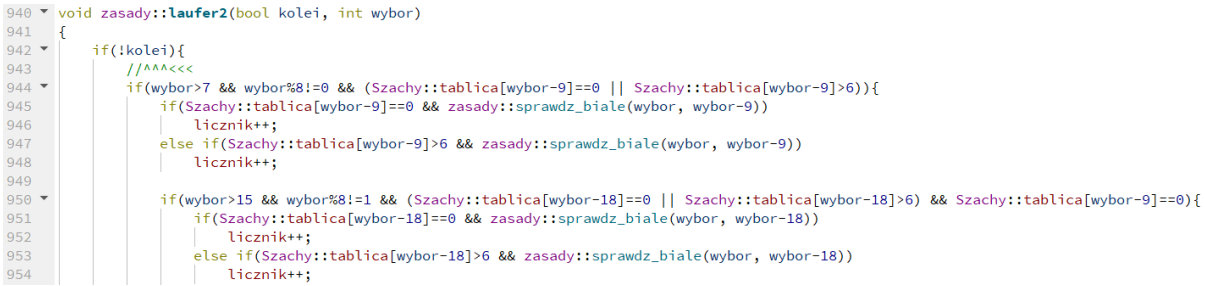
- **kon2()** – statyczna metoda niezwracająca wartości przyjmująca dwa parametry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje skoczek. Metoda sprawdza czy skoczek na podanym polu może wykonać co najmniej jeden ruch.

```
804 void zasady::kon2(bool kolei, int wybor)
805 {
806     if(!kolei){
807         if(wybor>15){
808             if(wybor%8!=0 &&  (Szachy::tablica[wybor-17]==0 || Szachy::tablica[wybor-17]>6) && zasady::sprawdz_biale(wybor, wybor-17)){
809                 if(Szachy::tablica[wybor-17]==0)
810                     licznik++;
811                 else if(Szachy::tablica[wybor-17]>6)
812                     licznik++;
813             }
814 
815             if(wybor%8!=7 &&  (Szachy::tablica[wybor-15]==0 || Szachy::tablica[wybor-15]>6) && zasady::sprawdz_biale(wybor, wybor-15)){
816                 if(Szachy::tablica[wybor-15]==0)
817                     licznik++;
818                 else if(Szachy::tablica[wybor-15]>6)
819                     licznik++;
820             }
821         }
```

Rys 5.11 Metoda kon2() klasy zasady.

- **laufer2()** – statyczna metoda niezwracająca wartości przyjmująca dwa parametry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje

laufer. Metoda sprawdza czy laufer na podanym polu może wykonać co najmniej jeden ruch.

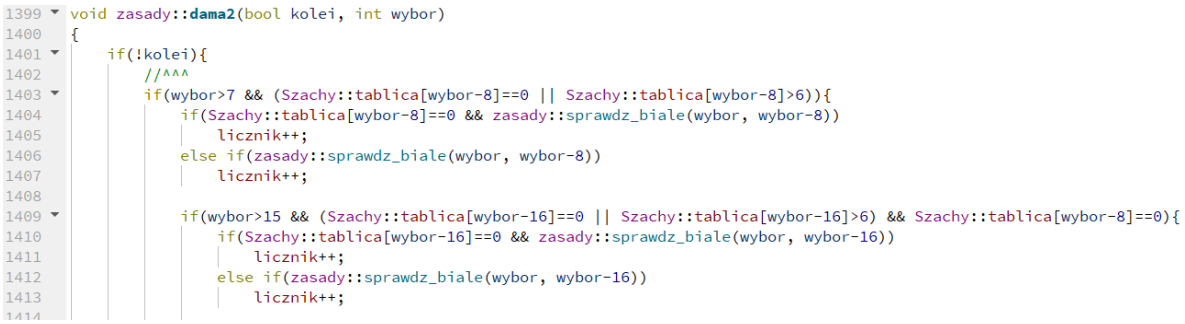


```
void zasady::laufer2(bool kolei, int wybor)
{
    if(!kolei){
        //^^^<<<
        if(wybor>7 && wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6)){
            if(Szachy::tablica[wybor-9]==0 && zasady::sprawdz_biale(wybor, wybor-9))
                licznik++;
            else if(Szachy::tablica[wybor-9]>6 && zasady::sprawdz_biale(wybor, wybor-9))
                licznik++;

            if(wybor>15 && wybor%8!=1 && (Szachy::tablica[wybor-18]==0 || Szachy::tablica[wybor-18]>6) && Szachy::tablica[wybor-9]==0){
                if(Szachy::tablica[wybor-18]==0 && zasady::sprawdz_biale(wybor, wybor-18))
                    licznik++;
                else if(Szachy::tablica[wybor-18]>6 && zasady::sprawdz_biale(wybor, wybor-18))
                    licznik++;
```

Rys 5.12 Metoda laufer2() klasy zasady.

- **dama2()** – statyczna metoda niezwracająca wartości przyjmująca dwa parametry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje królowa. Metoda sprawdza czy królowa na podanym polu może wykonać co najmniej jeden ruch.



```
void zasady::dama2(bool kolei, int wybor)
{
    if(!kolei){
        //^^^
        if(wybor>7 && (Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6)){
            if(Szachy::tablica[wybor-8]==0 && zasady::sprawdz_biale(wybor, wybor-8))
                licznik++;
            else if(zasady::sprawdz_biale(wybor, wybor-8))
                licznik++;

            if(wybor>15 && (Szachy::tablica[wybor-16]==0 || Szachy::tablica[wybor-16]>6) && Szachy::tablica[wybor-8]==0){
                if(Szachy::tablica[wybor-16]==0 && zasady::sprawdz_biale(wybor, wybor-16))
                    licznik++;
                else if(zasady::sprawdz_biale(wybor, wybor-16))
                    licznik++;
```

Rys 5.13 Metoda dama2() klasy zasady.

- **krol2()** – statyczna metoda niezwracająca wartości przyjmująca dwa paramatry: jeden boolowski kolei określający dla jakiego gracza wykonywana jest metoda, drugi będący liczbą całkowitą wybor określający pole na jakim występuje król. Metoda sprawdza czy król na podanym polu może wykonać co najmniej jeden ruch.

```
2265 void zasady::krol2(bool kolei, int wybor)
2266 {
2267     if(!kolei){
2268 
2269         if(wybor>7){
2270             if((Szachy::tablica[wybor-8]==0 || Szachy::tablica[wybor-8]>6) && zasady::sprawdz_biale(wybor, wybor-8)){
2271                 if(Szachy::tablica[wybor-8]==0)
2272                     licznik++;
2273                 else if(Szachy::tablica[wybor-8]>6)
2274                     licznik++;
2275             }
2276 
2277             if(wybor%8!=0 && (Szachy::tablica[wybor-9]==0 || Szachy::tablica[wybor-9]>6) && zasady::sprawdz_biale(wybor, wybor-9)){
2278                 if(Szachy::tablica[wybor-9]==0)
2279                     licznik++;
2280                 else if(Szachy::tablica[wybor-9]>6)
2281                     licznik++;
2282             }
2283 
2284             if(wybor%8!=7 && (Szachy::tablica[wybor-7]==0 || Szachy::tablica[wybor-7]>6) && zasady::sprawdz_biale(wybor, wybor-7)){
2285                 if(Szachy::tablica[wybor-7]==0)
2286                     licznik++;
2287                 else if(Szachy::tablica[wybor-7]>6)
2288                     licznik++;
2289             }
```

Rys 5.14 Metoda krol2() klasy zasady.

Poza metodami deklarujemy zmienne:

- **wektor** – statyczny wektor liczb całkowitych przechowujący informacje o 64 polach szachownicy pełni on role tymczasowego przechowywania danych symulowanych planszy.

- **licznik** – statyczna liczba całkowita odpowiadająca za zliczanie możliwości gracza.

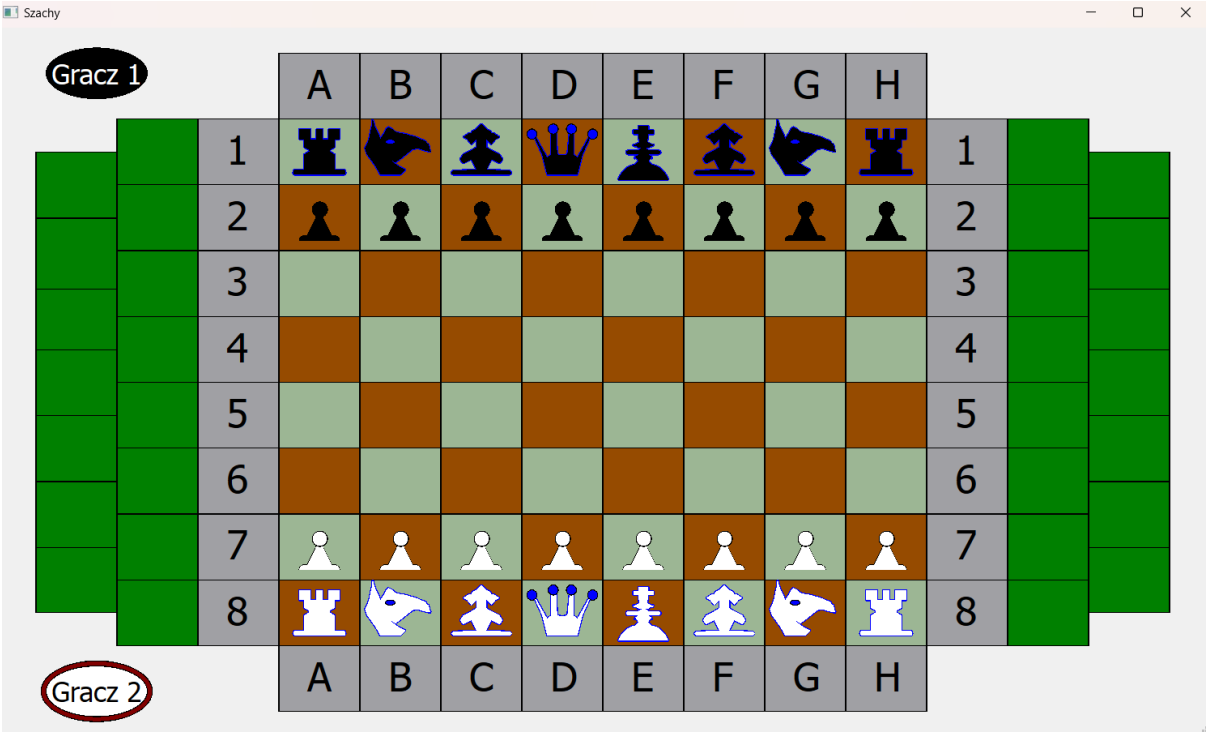
- **koniec** – statyczna zmienna boolowska odpowiadająca za określenie czy nastąpił koniec rozgrywki.

```
2465 std::vector<int> zasady::wektor(64, 0);
2466 
2467 int zasady::licznik=0;
2468 
2469 int zasady::koniec = 0;
```

Rys 5.15 Pola statyczne klasy zasady.

6. Wymagania odnośnie ocen

Są 3 kryteria oceny pod jaki podlega ten program umożliwiający rozegranie partii szachów dla dwóch graczy. Każde kryterium określa wymagania na daną ocenę.

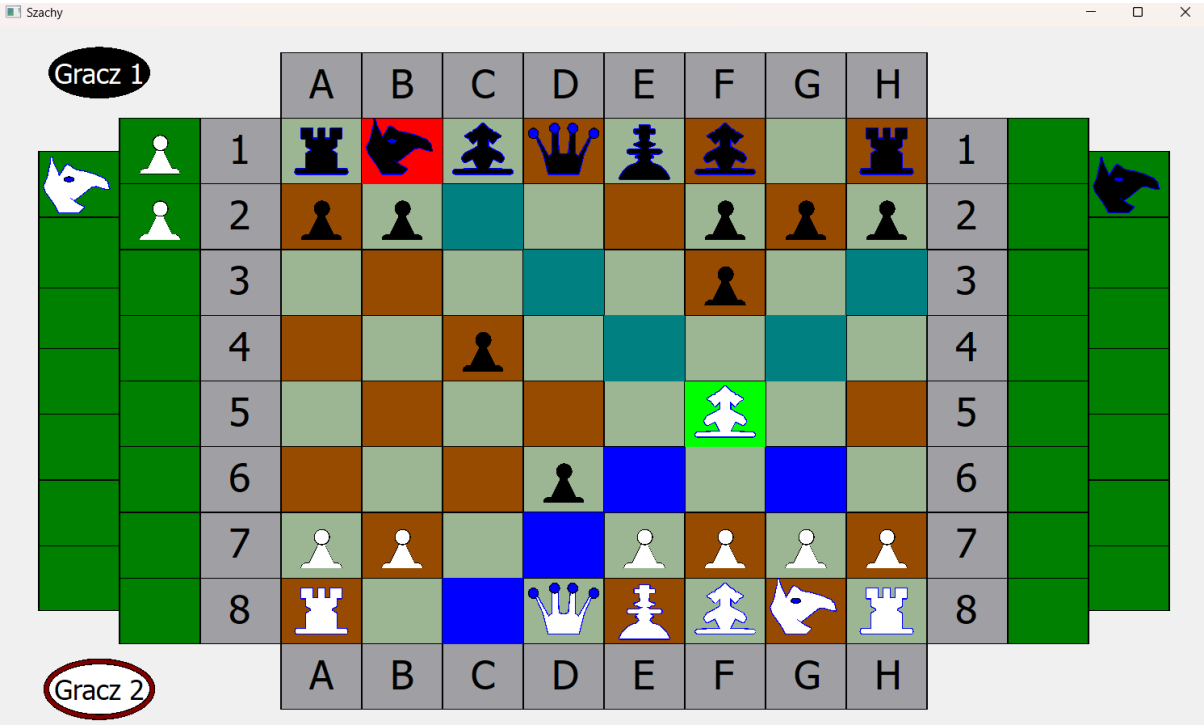


Rys 6.1 Plansza na początku rozgrywki.

6.1 Wymagania na ocenę 3.0

Na ocenę 3.0 trzeba stworzyć program, który zasymuluje rozgrywkę szachów dla dwóch osób. Figury powinny się ruszać odpowiednio względem swojego rodzaju. Powinno uwzględnić się bicie pionków i możliwości poruszania się ograniczane przez inne figury.

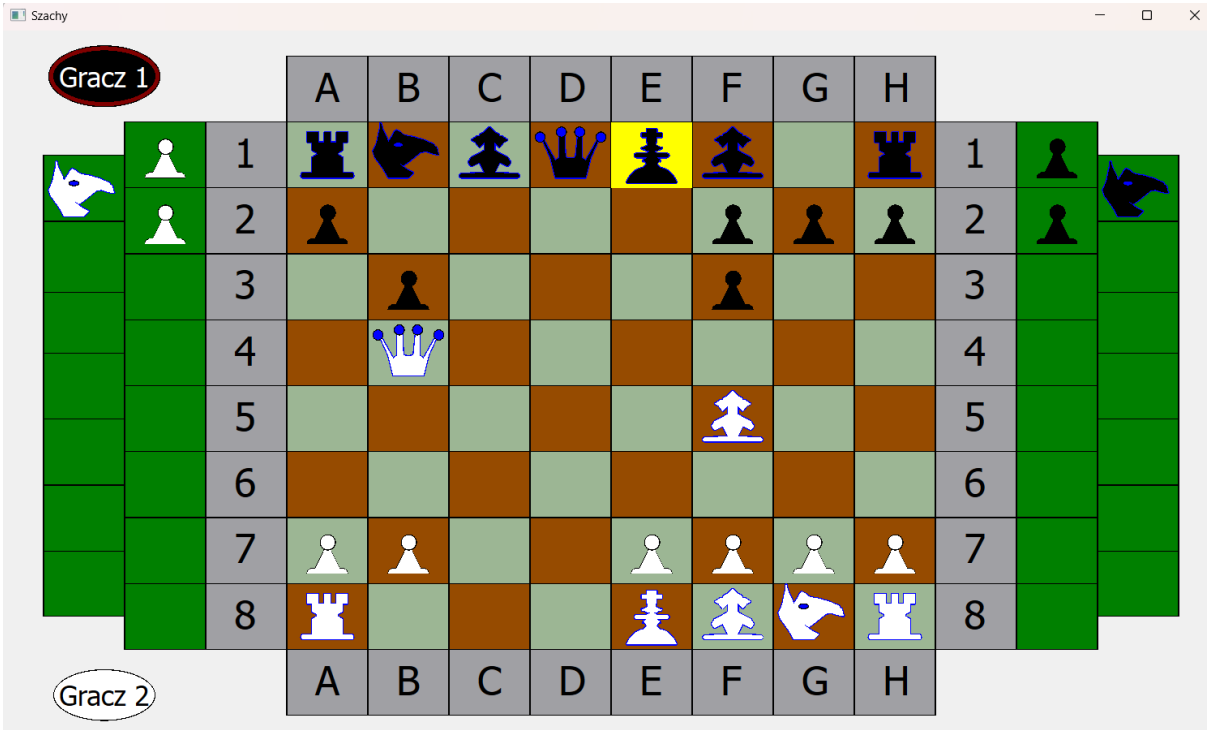
Rys 6.2 Przykładowe możliwości ruchu skoczka.



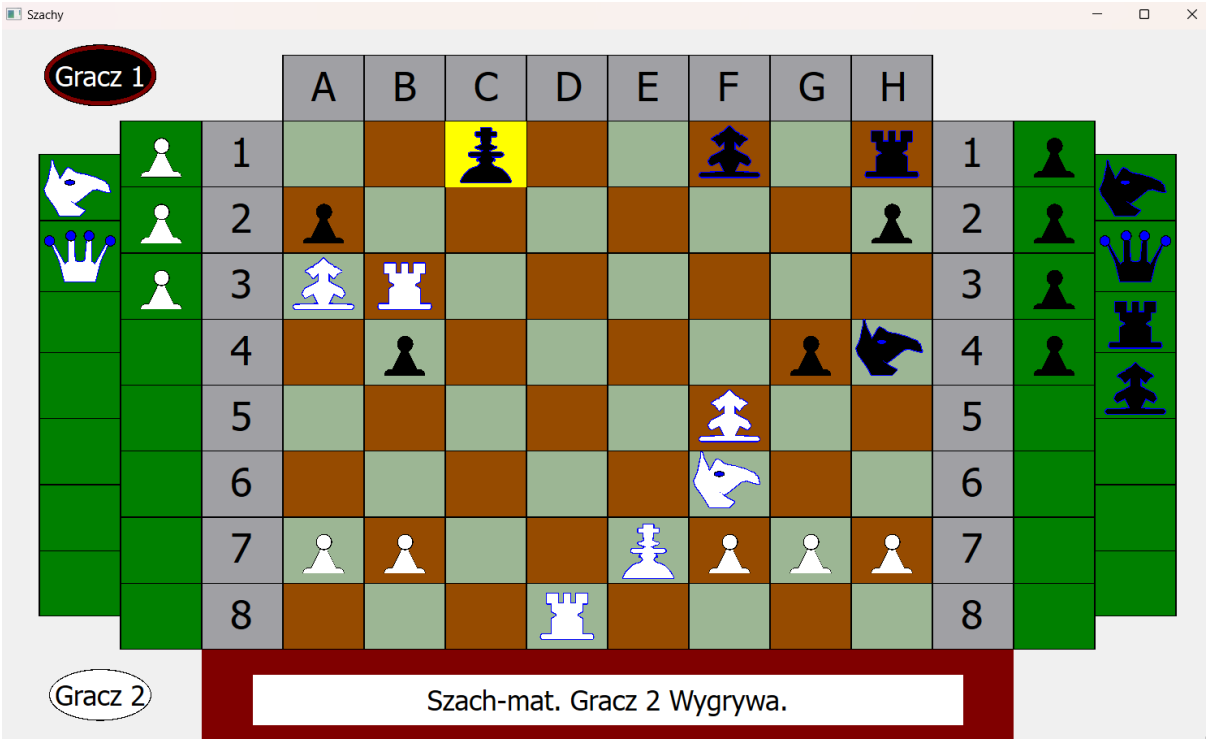
Rys 6.3 Przykładowe możliwości ruchu laufra.

6.2 Wymagania na ocenę 4.0

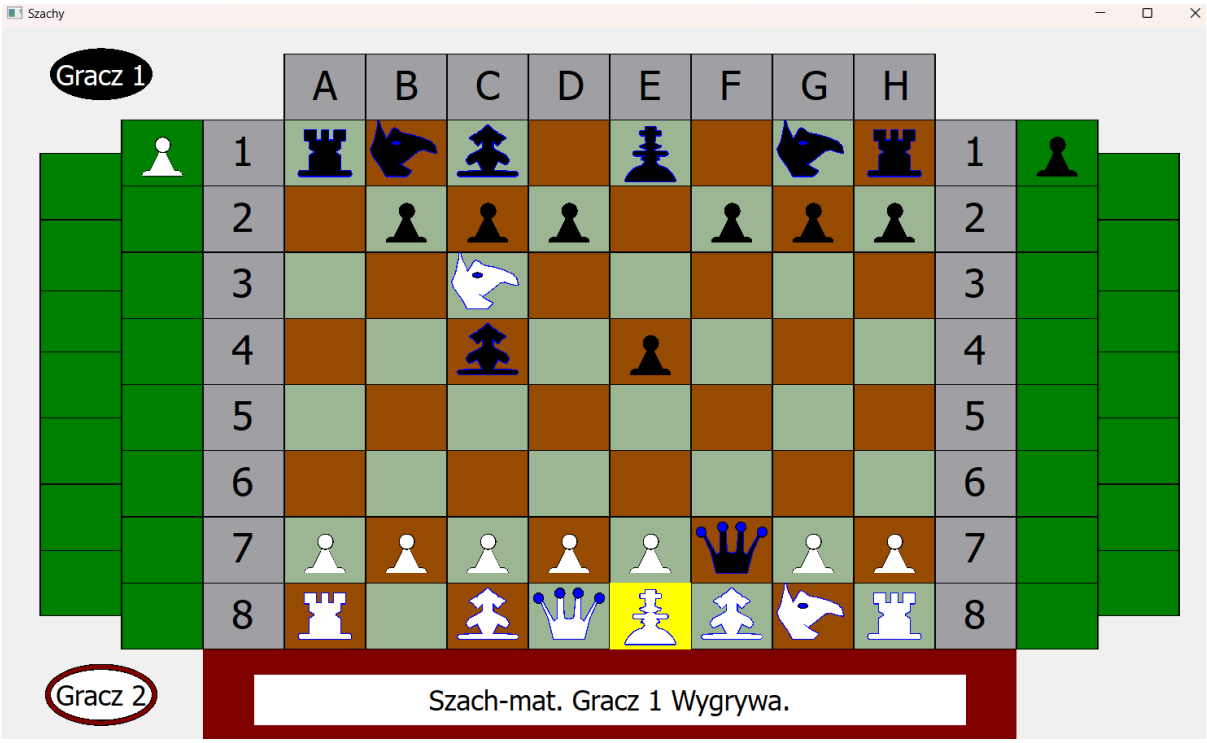
Na ocenę 4.0 trzeba stworzyć dodatkowe zależności, dzięki którym rozgrywka będzie zgodna z zasadami szach klasycznych. Trzeba uwzględnić zjawisko szachu, pata i szacha-mata. Gdy wykryte zostanie zjawisko szacha, nie będzie możliwe wykonanie ruchu, po którym nie zostanie rozwiązana sytuacja szachu. Jeżeli wystąpi szach-mat lub pat to zostanie wyświetlony odpowiedni komunikat i nie będzie możliwości ruchu żadną figurą.



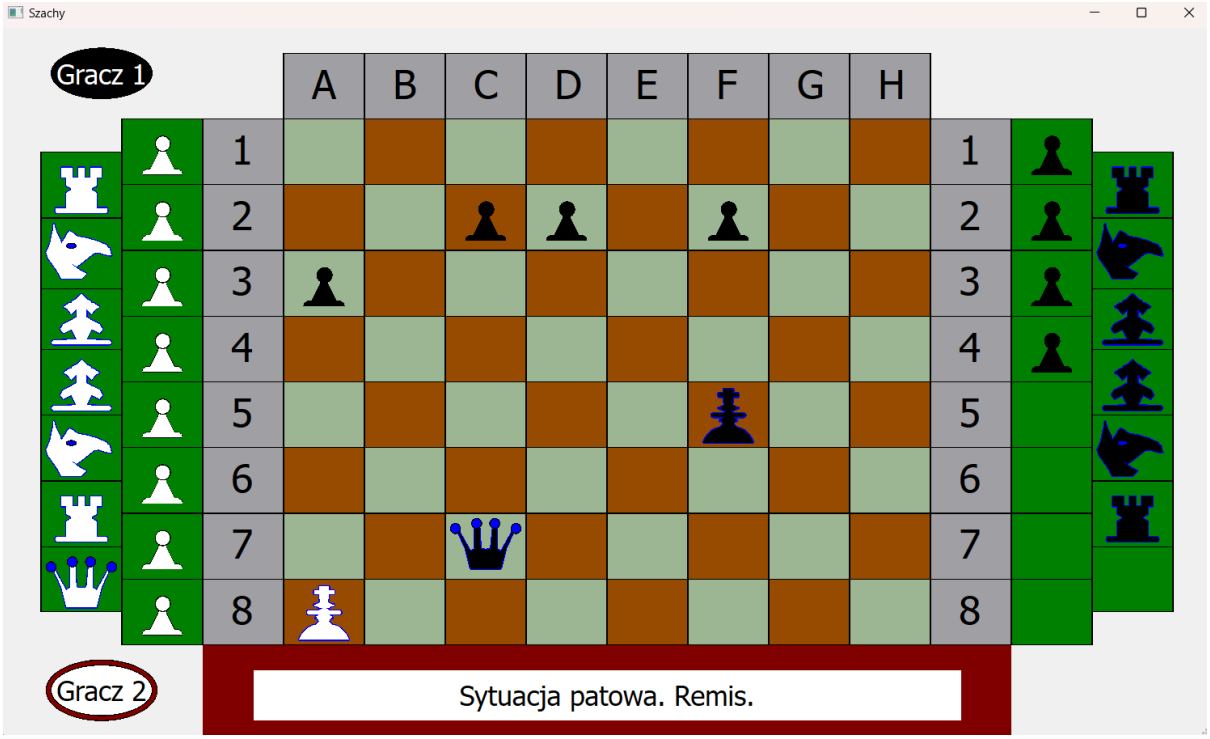
Rys 6.4 Przykładowa sytuacja wystąpienia szacha.



Rys 6.5 Przykład wystąpienia szach-mata stworzonego przez gracza grającego białymi figurami.



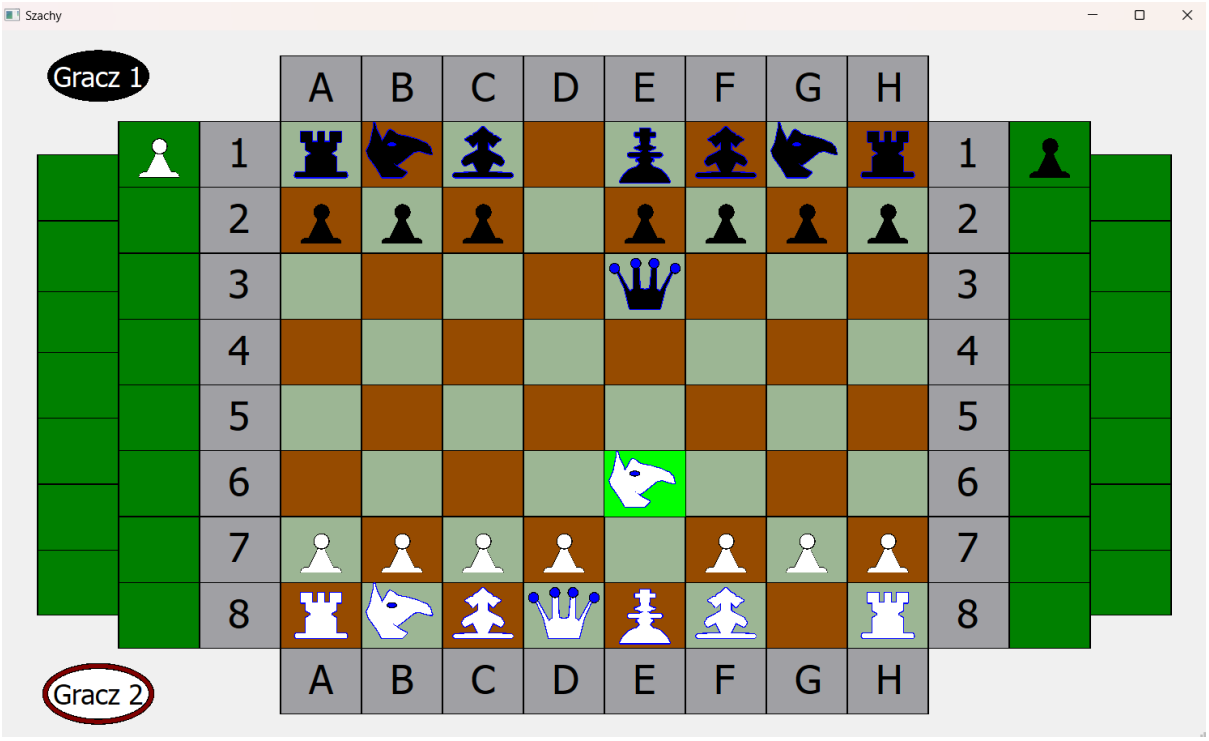
Rys 6.6 Przykład wystąpienia szach-mata stworzonego przez gracza używającego czarnych figur.



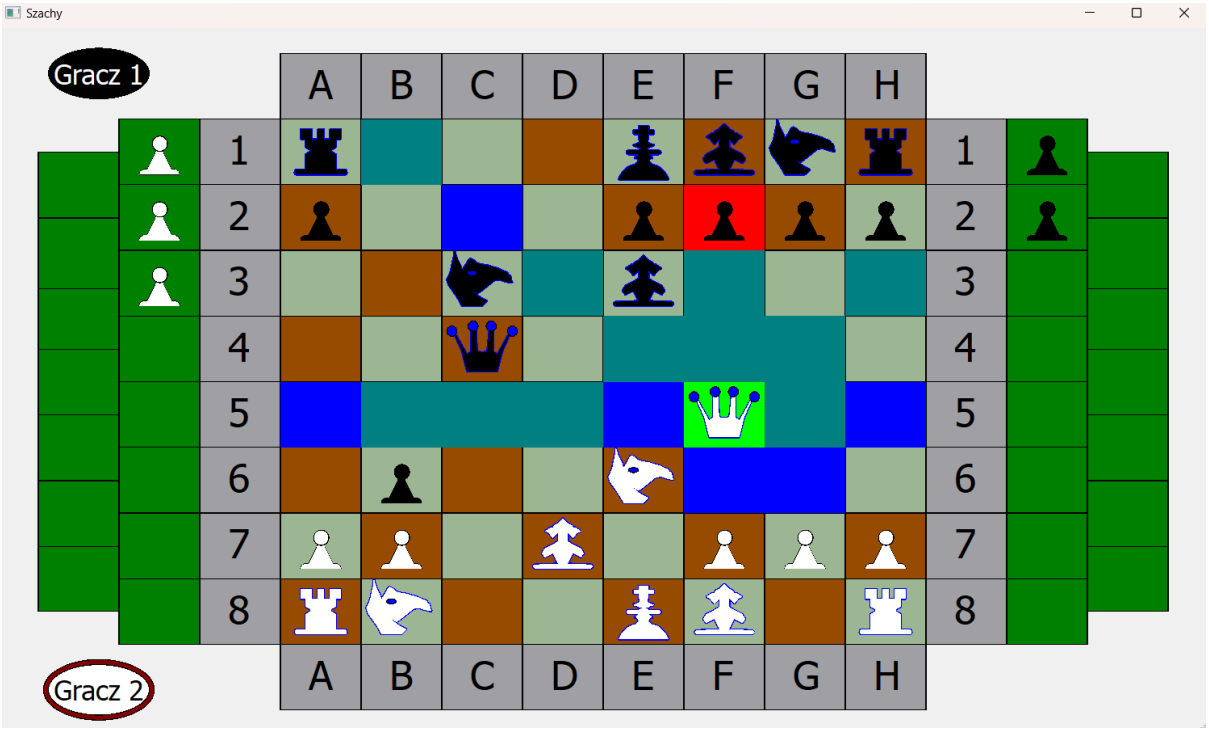
Rys 6.7 Przykład Pata, sytuacji bez wyjścia.

6.3 Wymagania na ocenę 5.0

Na ocenę 5.0 konieczne jest dodanie funkcji dzięki, której przy określaniu możliwości ruchu na polach, które są w zasięgu figur przeciwnika, pojawi się inny kolor ostrzegający, że po wykonaniu ruchu na dane pole figura może zostać zbita.
Ponadto trzeba sprawdzać czy figura nie jest sprzężona z królem, czyli czy po wykonaniu ruchu figurą nie wystąpi szach.



Rys 6.8 Przykład sprzężenia, nie można ruszyć się skoczkiem gdyż skutkowało by to szachem.



Rys 6.9 Przykład algorytmu przewidywania zbicia w kolejnym ruchu. Jasnoniebieskim kolorem oznaczono pola zagrożone przez figury przeciwnika.