Politechnika Lubelska
Wydział Podstaw Techniki

Dokumentacja projektu zaliczeniowego z przedmiotu: „Projekt z zakresu programowania” SUDOKU

Autorzy:
Alicja Bilińska
Marcin Woźniak

Matematyka
I rok

Treść zadania

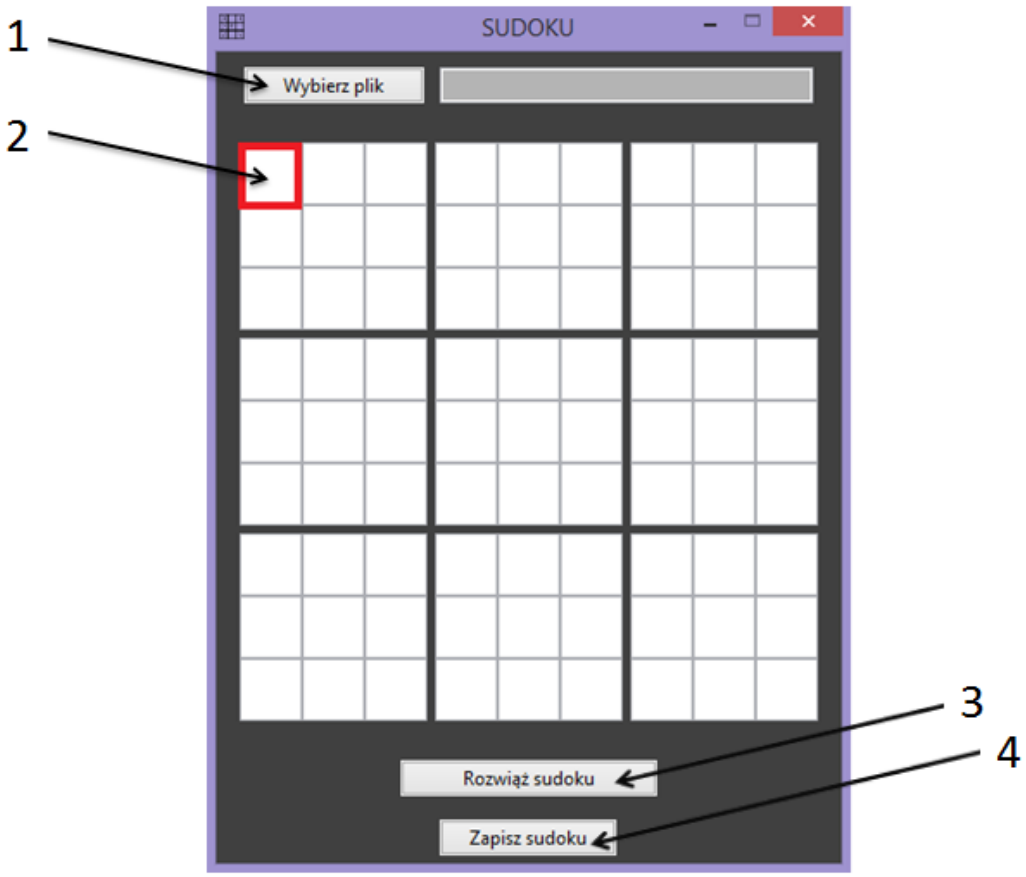
Sudoku

Utworzony w ramach projektu program:

- rozwiązuje klasyczne sudoku,
- sprawdza jednoznaczność rozwiązania,
- umożliwia wygodne wpisywanie za pomocą GUI konkretnych przykładów,
- umożliwia zapisywanie i odczytywanie przykładów z plików dyskowych.

1 Interfejs sudoku

Interfejs utworzony został w środowisku Code::Blocks przy użyciu wxWidgets.



Rysunek 1: Interfejs aplikacji

1. Przycisk pozwalający użytkownikowi wybrać planszę sudoku z dysku zapisaną w pliku typu .txt

2. Pole do wpisywania cyfr

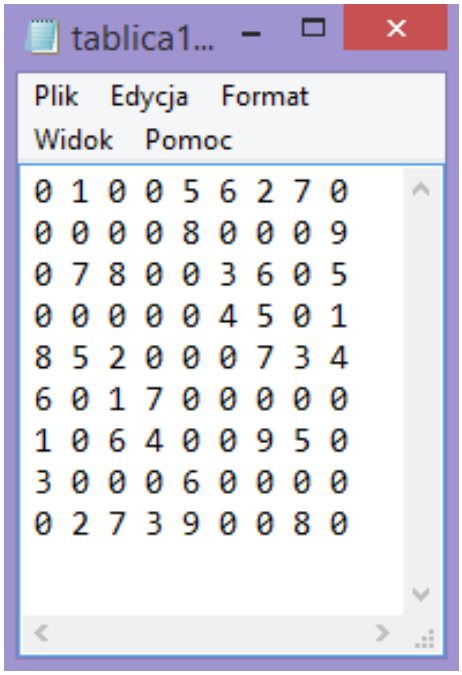
3. Przycisk aktywujący algorytm rozwiązujący sudoku

4. Przycisk pozwalający użytkownikowi zapisać do pliku typu .txt wczytaną tablicę sudoku w istniejącym już pliku lub stworzyć nowy

2 Klasy

2.1 Wczytywanie tablicy

Funkcja uruchamiana po kliknięciu przycisku „wybierz sudoku". Odczuje ona dane zawarte tablicy 9x9 i zapisuje je w sposób uporządkowany (Rysunek 1) w pliku .txt.



Rysunek 2: Przykładowa tablica do wczytania sudoku

```
#include "wczytywanie.h"
#include<fstream>
#include<string>
using namespace std;

wczytywanie::wczytywanie(string nazwa_pliku, int tablica
    [9][9])
{
        ifstream f(nazwa_pliku);
        string kropka=".";
        for(int i=0;i<9;i++){
                for(int j=0;j<9;j++){
                        string tmp;
                        f>>tmp;
```

```
                        if (tmp==kropka){
                                tablica[i][j]=0;
                        }
                        else{
                                tablica[i][j]=stoi(tmp);
                        }
                }
        }
}
```

2.2 Zapisywanie tablicy

Funkcja uruchamiana po kliknięciu przycisku „zapisz sudoku". Jej uruchomienie wywołuje okno wyboru pliku txt w którym ma zostać zapisana tablica sudoku. Po wyborze przez użytkownika konkretnego pliku txt, program wczytuje dane z komórek tekstowych na ekranie do tablicy int 9x9 oraz tworzy zmienną w klasie zapisywanie. Następuje pobranie danych z tablicy, a następnie zapisanie ich w sposób uporządkowany w pliku txt.

```
#include "zapisywanie.h"
#include<fstream>
#include<string>
using namespace std;

zapisywanie::zapisywanie(string nazwa_pliku, int tablica
    [9][9])
{
        ofstream plik(nazwa_pliku);
        for(int i=0;i<9;i++){
                for(int j=0;j<9;j++){
                        plik<<tablica[i][j]<<" ";
                }
                plik<<endl;
        }
}
```

2.3 Algorytm rozwiązujący sudoku

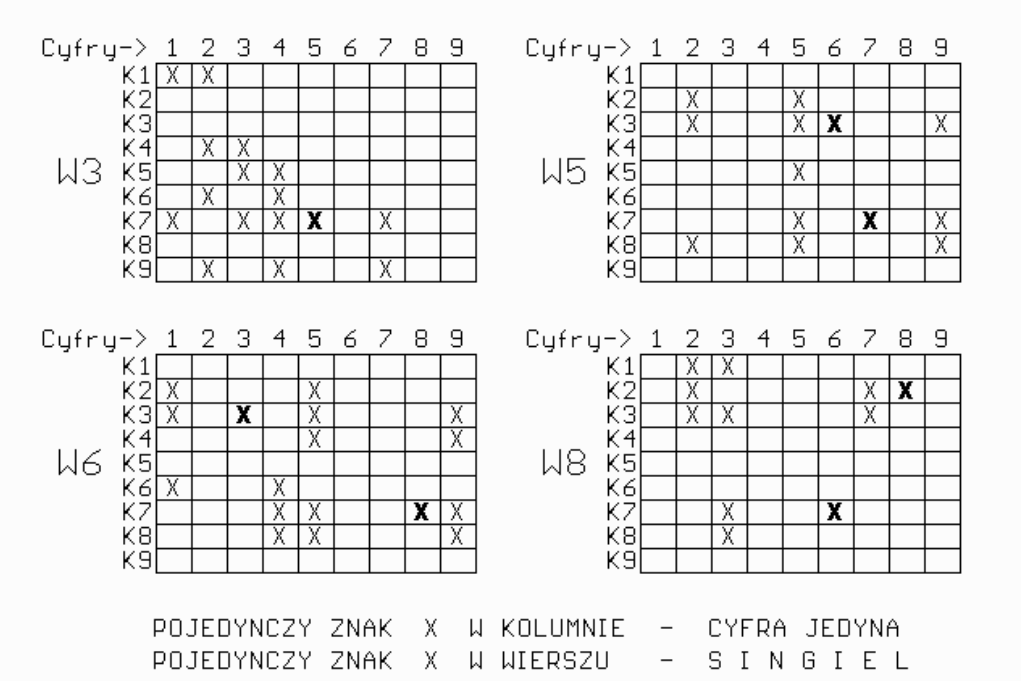
Zastosowaliśmy tzw. algorytm standardowy, który wyszukuje dwa typy rozwiązań w poszczególnych obiektach (wierszach, kolumnach, sekcjach - tablicach 3x3).

Kluczową rolę pełni funkcja `mozliwe`, która dla każdej komórki tworzy zbiór rozwiązań możliwych tzn takich, które nie występują w wierszu, kolumnie i sekcji, do których należy dana komórka. Funkcja tworzy zbiór liczb naturalnych od 1 do 9 (w pojemniku typu `set`), a następnie przegląda kolejno wiersz, kolumnę i sekcję usuwając przy tym z pojemnika cyfry, które się w tych obiektach pojawiły:

```
void mozliwe(int i, int j, int tablica[9][9], int table[9][9],
    int pomoc){
        set<int>rozwiazania;
        if(tablica[i][j]==0)
        {
```

```
5                   for(int k=1;k<10;k++) rozwiazania.insert(k);
6                   for(int n=0;n<9;n++)
7                   //petla analizujaca i-ty wiersz i j-kolumne
8                   {
9            rozwiazania.erase(tablica[i][n]);
10           //usuniecie ze zbioru elementow i-tego wiersza
11           rozwiazania.erase(tablica[n][j]);
12           //usuniecie ze zbioru elementow j-tej kolumny
13                  }
14
15          int x=i-i%3;
16          int xo=x+3;
17          for(x;x<xo;x++)
18           //petla analizujaca odpowiednia sekcje (kwadrat 3x3)
19          {
20                          int y=j-j%3;
21                          int yo=y+3;
22                          for(y;y<yo;y++)
23                          {
24                          rozwiazania.erase(tablica[x][y]);
25                          }
26                  }
27
28                  for( int p=1;p<10;p++)
29                  {
30                          if(rozwiazania.count(p)==1){
31                                  table[pomoc][p-1]=1;
32
33                          }
34                          else ;
35                  }
36          }
37          else for( int p=1;p<10;p++) ;
38  }
```

Następnymi ważnymi funkcjami są `pomocnicza...`, które tworzą pomocnicze tablice zbiorów rozwiązań możliwych, dzięki których będą znajdywane dwa typy rozwiązań. Pierwszy z nich, to rozwiązania typu SINGIEL - występują one wtedy, gdy zbiór rozwiązań możliwych dla danej komórki jest jednoelementowy. Drugi typ - JEDYNA - to taka cyfra, która występuje tylko raz w zestawieniu zbiorów rozwiązań możliwych dla danego obiektu (wiersza, kolumny, sekcji). Jeżeli takie rozwiązania zostają znalezione, są one od razu wpisywane do odpowiedniej komórki w tablicy sudoku:



Rysunek 3: Przykładowe tablice pomocnicze

```
void pomocnicza_w(int i,int tablica[9][9],int table[9][9]){

        int pomoc=0;
        for(int j=0;j<9;j++)
        {
                mozliwe(i,j,tablica,table,pomoc);
                pomoc++;
        }
        for(int n=0;n<9;n++){
                if(suma_poz(n,table)==1)
                {
                for(int m=0;m<9;m++){
                        if(table[n][m]==1) tablica[i][n]=m+1;
                        }
                }
        }

        for(int n=0;n<9;n++){
                if(suma_pion(n,table)==1)
                {

                for(int m=0;m<9;m++){
                        if(table[m][n]==1) tablica[i][m]=n+1;
                        }
                }
        }

}
```

Po przedstawieniu tych najważniejszych funkcji możemy przejść do opisu działania algorytmu.

Na początku tworzone są trzy zmienne: `bool ok=false`, `string kom="Ta tablica nie ma rozwiązania"`, `bool niejednoznaczna=false`, których wartości będą zmieniane w zależności od rozwiązania, jakie uzyska algorytm.

Alogrytm 100-krotnie (stworzyliśmy, pętlę while, która obraca się 100 razy) wyszukuje dla każdego obiektu (wiersza, kolumny, sekcji) podane wcześniej dwa typy rozwiązań. Jeżeli po wykonaniu tych działań sudoku nadal nie jest rozwiązane (czyli gdzieś w tablicy nadal występuje 0), program znajduje pierwszą nierozwiązaną komórkę i wstawia do niej jedno z jej możliwych rozwiązań. Zmienna `niejednoznaczna` zmienia wartość na `true` i program ponownie rozwiązuje sudoku do momentu zapełnienia wszystkich komórek.

Na koniec wywoływana jest fukcja, która sprawdza poprawność podanego rozwiązania. Jeżeli sudoku jest rozwiązane poprawnie, to zmienna `ok` przyjmuje wartość `true`, a zmienna `kom` zmienia zawartość w zależności od tego, czy rozwiązanie to było jednoznaczne, czy też nie. Jeżeli natomiast rozwiązanie jest niepoprawne, to zmienna `ok` przyjmuje wartość `false`, a zmienna `kom` otrzymuje komentarz o braku rozwiązań dla danej tablicy.

pozostałe funkcje to `bool algorytm::OK()`, która zwraca wartość zmiennej `ok`, `string algorytm::komunikat()`, która zwraca komunikat uzyskany ze zmiennej `kom` oraz pomocnicze funkcje `suma_pion` i `suma_poz`.

Cała zawartość pliku `alogrytm.cpp`:

```
#include "algorytm.h"
#include<iostream>
#include<set>
#include<fstream>
#include<string>
using namespace std;

bool ok=false;
string kom="Ta tablica nie ma rozwiazania";
bool znajdz_zero(int tablica[9][9]){
        for(int n=0;n<9;n++){
                for(int m=0;m<9;m++){
                        if(tablica[n][m]==0)
                        {
                                return true;
                        }
                }
        }
        return false;
}

int suma_pion(int cyfra,int table[9][9]){
        int suma=0;
        for(int n=0;n<9;n++) suma+=table[n][cyfra];
        return suma;
}

int suma_poz(int grid,int table[9][9]){
        int suma=0;
```

```
30          for(int n=0;n<9;n++) suma+=table[grid][n];
31          return suma;
32  }
33
34  void mozliwe(int i, int j, int tablica[9][9], int table[9][9],
        int pomoc){
35          set<int>rozwiazania;
36          if(tablica[i][j]==0)
37          {
38                  for(int k=1;k<10;k++) rozwiazania.insert(k);
39
40          for(int n=0;n<9;n++)
41           //petla analizujaca i-ty wiersz i j-kolumne
42                  {
43          rozwiazania.erase(tablica[i][n]);
44          //usuniecie ze zbioru elementow i-tego wiersza
45          rozwiazania.erase(tablica[n][j]);
46          //usuniecie ze zbioru elementow j-tej kolumny
47                  }
48
49          int x=i-i%3;
50          int xo=x+3;
51          for(x;x<xo;x++)
52          //petla analizujaca odpowiednia sekcje (kwadrat 3x3)
53                  {
54                          int y=j-j%3;
55                          int yo=y+3;
56                          for(y;y<yo;y++)
57                          {
58                          rozwiazania.erase(tablica[x][y]);
59                          }
60                  }
61
62                  for( int p=1;p<10;p++)
63                  {
64                          if(rozwiazania.count(p)==1){
65                                  table[pomoc][p-1]=1;
66
67                          }
68                          else ;
69                  }
70          }
71          else for( int p=1;p<10;p++) ;
72  }
73
74  void pomocnicza_w(int i,int tablica[9][9],int table[9][9]){
75
76          int pomoc=0;
77          for(int j=0;j<9;j++)
78          {
79                  mozliwe(i,j,tablica,table,pomoc);
80                  pomoc++;
81          }
```

```
82          for(int n=0;n<9;n++){
83                  if(suma_poz(n,table)==1)
84                  {
85                  for(int m=0;m<9;m++){
86                          if(table[n][m]==1) tablica[i][n]=m+1;
87                          }
88                  }
89          }
90
91          for(int n=0;n<9;n++){
92                  if(suma_pion(n,table)==1)
93                  {
94                  for(int m=0;m<9;m++){
95                          if(table[m][n]==1) tablica[i][m]=n+1;
96
97                          }
98                  }
99          }
100
101 }
102
103 void pomocnicza_k(int j,int tablica[9][9],int table[9][9]){
104
105         int pomoc=0;
106         for(int i=0;i<9;i++)
107         {
108                 mozliwe(i,j,tablica,table,pomoc);
109                 pomoc++;
110         }
111         for(int n=0;n<9;n++){
112                 if(suma_poz(n,table)==1)
113                 {
114                 for(int m=0;m<9;m++){
115                         if(table[n][m]==1) tablica[n][j]=m+1;
116                         }
117                 }
118         }
119
120         for(int n=0;n<9;n++){
121                 if(suma_pion(n,table)==1)
122                 {
123                 for(int m=0;m<9;m++){
124                         if(table[m][n]==1) tablica[m][j]=n+1;
125
126                         }
127                 }
128         }
129
130 }
131
132 void pomocnicza_sekcja_komorki( int sekcja[2], int tablica
    [9][9],int table[9][9]){
133
```

```
134         int pomoc=0;
135         int i=sekcja[0];
136         int x=i-i%3;
137         int xo=x+3;
138         for(x;x<xo;x++)
139         {
140                 int j=sekcja[1];
141
142                 int y=j-j%3;
143                 int yo=y+3;
144                 for(y;y<yo;y++)
145                 {
146
147                         mozliwe(x,y,tablica,table,pomoc);
148                         pomoc++;
149                 }
150         }
151
152
153         for(int n=0;n<9;n++){
154   //sprawdzam, czy w n kolumnie jest rozwiazanie typu JEDYNA
155                 if(suma_pion(n,table)==1)
156                 {
157                 int i=sekcja[0];
158                 int x=i-i%3;
159                 int xo=x+3;
160                 int l=0;
161                 for(x;x<xo;x++)
162                 {
163                         int j=sekcja[1];
164                         int y=j-j%3;
165                         int yo=y+3;
166                         for(y;y<yo;y++)
167                         {
168                                 if(table[l][n]==1)
169                                 {
170                                 tablica[x][y]=n+1;
171                                         }
172                                         l++;
173                                 }
174                         }
175                 }
176         }
177
178
179 }
180
181 bool sprawdz_sudoku(int tablica[9][9]){
182         for(int i=0;i<9;i++){
183                 set<int>wiersz;
184                 for(int j=0;j<9;j++){
185                         if(wiersz.count(tablica[i][j])==1)
186                         {
```

```
187                                     return false;
188                             }
189                             else wiersz.insert(tablica[i][j]);
190                             if(wiersz.count(0)==1) {
191                                     return false;
192                             }
193                     }
194
195             }
196
197             for(int i=0;i<9;i++){
198                     set<int>kolumna;
199                     for(int j=0;j<9;j++){
200                             if(kolumna.count(tablica[j][i])==1)
201                             {
202
203                                     return false;
204                             }
205
206                             else kolumna.insert(tablica[j][i]);
207                             if(kolumna.count(0)==1) {
208
209                                     return false;
210                             }
211                     }
212             }
213
214             for(int i=0;i<1;i++){
215                     for(int j=0;j<9;j++){
216                             set<int>sekcja;
217                             int x=i-i%3;
218                             int xo=x+3;
219                             for(x;x<xo;x++)
                                    //petla analizujaca odpowiednia
                                    sekcje (kwadrat 3x3)
220                             {
221                             int y=j-j%3;
222                             int yo=y+3;
223                             for(y;y<yo;y++)
224                             {
225                             if(sekcja.count(tablica[x][y])==1)
226                                     {
227                                     return false;
228                                     }
229
230                     else sekcja.insert(tablica[x][y]);
231                             if(sekcja.count(0)==1) {
232                                     return false;
233                                             }
234                                     }
235
236                             }
237                     }
```

```
238         }
239
240         for(int i=0;i<9;i++)
241         {
242                 for(int j=0;j<9;j++)
243                 {
244                         if(tablica[i][j]>9) return false;
245                         if(tablica[i][j]<1) return false;
246                 }
247         }
248
249         return true;
250 }
251
252 bool rozwiazywanie_probne(int tab[9][9]){
253
254 bool niejednoznaczna=false;
255
256         int sekcja1[2]={0,0};
257         int sekcja2[2]={0,3};
258         int sekcja3[2]={0,6};
259         int sekcja4[2]={3,0};
260         int sekcja5[2]={3,3};
261         int sekcja6[2]={3,6};
262         int sekcja7[2]={6,0};
263         int sekcja8[2]={6,3};
264         int sekcja9[2]={6,6};
265         int z=0;
266         while(z<100){
267                 z++;
268                 for(int n=0;n<9;n++){
269                         int table[9][9];
270                         for(int i=0;i<9;i++){
271
272                                 for(int j=0;j<9;j++)
273                                 {
274                                         table[i][j]=0;
275                                 }
276                         }
277                         pomocnicza_w(n,tab,table);
278                 }
279                 for(int n=0;n<9;n++){
280                         int table[9][9];
281                         for(int i=0;i<9;i++){
282
283                                 for(int j=0;j<9;j++)
284                                 {
285                                         table[i][j]=0;
286                                 }
287                         }
288                         pomocnicza_k(n,tab,table);
289                 }
290
```

```
291                     int table[9][9];
292                     for(int i=0;i<9;i++){
293
294                             for(int j=0;j<9;j++)
295                             {
296                                     table[i][j]=0;
297                             }
298                     }
299                     pomocnicza_sekcja_komorki(sekcja1,tab,table);
300
301                     for(int i=0;i<9;i++){
302
303                             for(int j=0;j<9;j++)
304                             {
305                                     table[i][j]=0;
306                             }
307                     }
308                     pomocnicza_sekcja_komorki(sekcja2,tab,table);
309
310                     for(int i=0;i<9;i++){
311
312                             for(int j=0;j<9;j++)
313                             {
314                                     table[i][j]=0;
315                             }
316                     }
317                     pomocnicza_sekcja_komorki(sekcja3,tab,table);
318
319                     for(int i=0;i<9;i++){
320
321                             for(int j=0;j<9;j++)
322                             {
323                                     table[i][j]=0;
324                             }
325                     }
326                     pomocnicza_sekcja_komorki(sekcja4,tab,table);
327
328                     for(int i=0;i<9;i++){
329
330                             for(int j=0;j<9;j++)
331                             {
332                                     table[i][j]=0;
333                             }
334                     }
335                     pomocnicza_sekcja_komorki(sekcja5,tab,table);
336
337                     for(int i=0;i<9;i++){
338
339                             for(int j=0;j<9;j++)
340                             {
341                                     table[i][j]=0;
342                             }
343                     }
```

```
344                     pomocnicza_sekcja_komorki(sekcja6,tab,table);
345
346                     for(int i=0;i<9;i++){
347
348                             for(int j=0;j<9;j++)
349                             {
350                                     table[i][j]=0;
351                             }
352                     }
353                     pomocnicza_sekcja_komorki(sekcja7,tab,table);
354
355                     for(int i=0;i<9;i++){
356
357                             for(int j=0;j<9;j++)
358                             {
359                                     table[i][j]=0;
360                             }
361                     }
362                     pomocnicza_sekcja_komorki(sekcja8,tab,table);
363
364                     for(int i=0;i<9;i++){
365
366                             for(int j=0;j<9;j++)
367                             {
368                                     table[i][j]=0;
369                             }
370                     }
371                     pomocnicza_sekcja_komorki(sekcja9,tab,table);}
372
373             if(!znajdz_zero(tab));
374             else{
375                     for(int n=0;n<9;n++){
376                     for(int m=0;m<9;m++){
377                     if(tab[n][m]==0)
378                             {
379                             int table[9][9];
380                             for(int i=0;i<9;i++){
381                             for(int j=0;j<9;j++)
382                                     {
383                                     table[i][j]=0;
384                                     }
385                             }
386                             mozliwe(n,m,tab,table,0);
387                             int tmp=0;
388                             int w=0;
389                             while(tmp==0){
390                                     w++;
391                                     tmp=table[0][w-1];
392                                             }
393                             tab[n][m]=w;
394                             rozwiazywanie_probne(tab);
395                                     }
396                             }
```

```
397                     }
398                     niejednoznaczna=true;
399             }
400 if(sprawdz_sudoku(tab)) {
401 ok=true;
402 if(niejednoznaczna)     kom="Ta tablica wiecej niz jedno
      rozwiazanie";
403 else      kom="Ta tablica ma tylko jedno rozwiazanie";
404             }
405             else{
406                     ok=false;
407                     kom="Ta tablica nie ma rozwiazan";
408             }
409             return niejednoznaczna;
410 }
411
412 algorytm::algorytm(int tablica[9][9],bool niejednoznaczna)
413 {
414             rozwiazywanie_probne(tablica);
415 }
416
417 bool algorytm::OK(){
418             return ok;
419 }
420
421 string algorytm::komunikat(){
422             return kom;
423
424 }
```

3 Funkcja główna (interfejsMain.cpp)

W części głównej programu utworzona została tablica 81 wskaźników odpowiadających zmiennym TextCtrl, które reprezentowane są jako komórki na ekranie:

```
1  komorki[0][0]=TextCtrl1;
2  komorki[0][1]=TextCtrl2;
3  komorki[0][2]=TextCtrl3;
4  komorki[0][3]=TextCtrl4;
5  komorki[0][4]=TextCtrl5;
6  komorki[0][5]=TextCtrl6;
7  komorki[0][6]=TextCtrl7;
8  komorki[0][7]=TextCtrl8;
9  komorki[0][8]=TextCtrl9;
10 ...
```

3.1 Funkcja wczytująca tablicę sudoku przypisana do przycisku ToggleButton „Wybierz plik”

Jej uruchomienie wywołuje okno wyboru pliku typu txt z tablicą sudoku. Po wybraniu pliku przez użytkownika, w klasie wczytywanie tworzona zostaje zmienna - następuje pobranie danych z pliku txt do tablicy int 9x9. Program wypisuje zawartość tablicy w komórkach tekstowych, jednocześnie resetowana jest zawartość komórek pozostała po poprzednium wczytywaniu. Zera w tablicy reprezentowane są jako puste komórki na ekranie. W komórkach, w których pojawiają się cyfry, blokowana jest możliwość ich zmiany przez użytkownika (poprzez funkcję `Enable`):

```
1  void interfejsDialog::OnToggleButton2Toggle1(wxCommandEvent&
       event)      //Akcja przycisku "Wybierz plik"
2  {
3          wxFileDialog*dlg_wczytaj_dane=new wxFileDialog(this,_(
               "Wskaz plik z danymi"),_(""),_(""),_("*.txt"),
               wxFD_OPEN|wxFD_CHANGE_DIR|wxFD_FILE_MUST_EXIST);
4          if(dlg_wczytaj_dane->ShowModal()!=wxID_CANCEL)
5          {
6          TextCtrl82->SetValue(dlg_wczytaj_dane->GetPath());
7          }
8          delete dlg_wczytaj_dane;
9          string sciezka=TextCtrl82->GetValue().ToStdString();
10
11         if(sciezka.length()!=0)
12         {
13                 wczytywanie w(sciezka, tablica);
14
15                 for(int i=0;i<9;i++){
16                         for(int j=0;j<9;j++){
17                                 if (tablica[i][j]==0)
18                 {
19                 komorki[i][j]->Clear();
20                 komorki[i][j]->SetBackgroundColour(
                       wxSystemSettings::GetColour(
                       wxSYS_COLOUR_WINDOW));
21                 komorki[i][j]->Enable(true);
22                                 }
23                                 else
24                                 {
25                 komorki[i][j]->SetValue(s2w(to_string(tablica[
                       i][j])));
26                 komorki[i][j]->SetBackgroundColour(
                       wxSystemSettings::GetColour(
                       wxSYS_COLOUR_INACTIVECAPTION));
27                 komorki[i][j]->Enable(false);
28                                 }
29                         }
30                 }
31         }
32 }
```

3.2 Funkcja rozwiązująca sudoku przypisana do przycisku ToggleButton „Rozwiąż sudoku”

Następuje pobranie danych z komórek tekstowych na ekranie - dzięki temu uwzględnione również zostają cyfry wpisane przez użytkownika. W konsekwencji tworzona jest zmienna w klasie algorytm oraz wyświetlany zostaje odpowiedni komunikat, dopasowany do ilości możliwych rozwiązań wybranej planszy sudoku:

```
1  void interfejsDialog::OnToggleButton1Toggle(wxCommandEvent&
       event)
2      //Akcja przycisku "Rozwiaz sudoku"
3  {
4          for(int i=0;i<9;i++){
5                  for(int j=0;j<9;j++){
6  if((komorki[i][j]->GetValue().ToStdString()).length()==0)
       tablica[i][j]=0;
7          else
8          tablica[i][j]=stoi(komorki[i][j]->GetValue().
               ToStdString());
9                  }
10         }
11 algorytm r(tablica);
12 if(r.OK())
13 {
14 for(int i=0;i<9;i++){
15 for(int j=0;j<9;j++){
16 komorki[i][j]->SetValue(s2w(to_string(tablica[i][j])));
17                 }
18         }
19 wxMessageBox(s2w(r.komunikat()), _("Sukces!"),
       wxICON_INFORMATION);
20         }
21         else
22 wxMessageBox(s2w(r.komunikat()), _("Wystapil problem!"),
       wxICON_ERROR);
23 }
```

3.3 Funkcja przypisana do przycisku „zapisz sudoku”

Jej uruchomienie wywołuje okno wyboru pliku txt w którym ma zostać zapisana tablica sudoku. Po wyborze przez użytkownika konkretnego pliku txt, program wczytuje dane z komórek tekstowych na ekranie do tablicy int 9x9 oraz tworzy zmienną w klasie zapisywanie. Następuje pobranie danych z tablicy, a następnie zapisanie ich w sposób uporządkowany w pliku txt:

```
1  void interfejsDialog::OnToggleButton3Toggle(wxCommandEvent&
       event)       //Akcja przycisku "Zapisz sudoku"
2  {
3          wxFileDialog*dlg_zapisz_dane=new wxFileDialog(this,_("
               Wskaz plik, w ktorym zapiszesz sudoku lub utworz
               nowy"),_(""),_(""),_("*.txt"),wxFD_CHANGE_DIR);
```

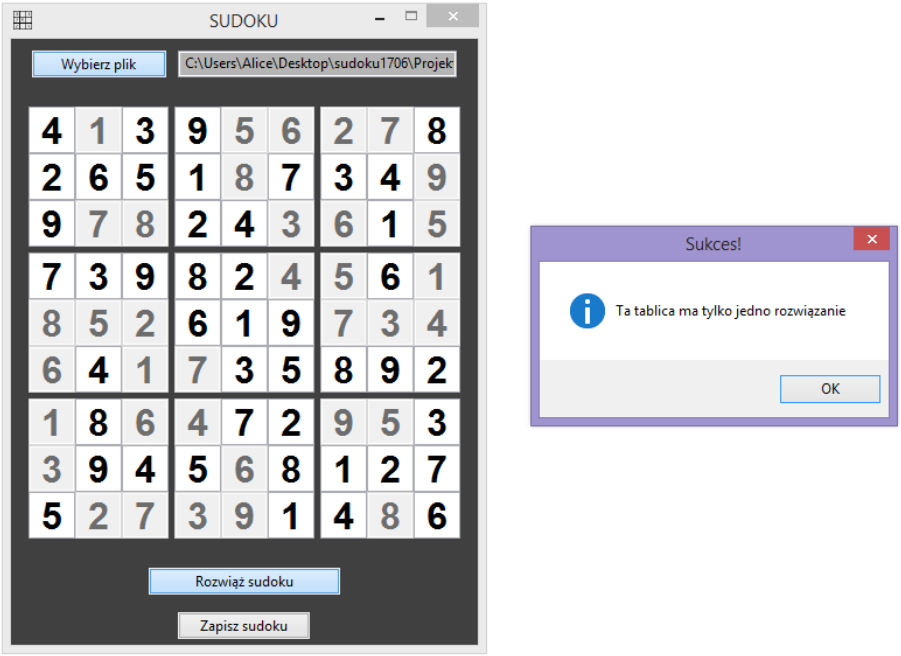
```
4           if(dlg_zapisz_dane->ShowModal()!=wxID_CANCEL)
5           {
6           TextCtrl82->SetValue(dlg_zapisz_dane->GetPath());
7           }
8           delete dlg_zapisz_dane;
9           string sciezka=TextCtrl82->GetValue().ToStdString();
10
11          if(sciezka.length()!=0)
12          {
13                  for(int i=0;i<9;i++){
14                          for(int j=0;j<9;j++){
15          if((komorki[i][j]->GetValue().ToStdString()).length()
               ==0) tablica[i][j]=0;
16          else
17          tablica[i][j]=stoi(komorki[i][j]->GetValue().
               ToStdString());
18                          }
19                  }
20                  zapisywanie z(sciezka, tablica);
21          }
22  }
```

Dodatkowo do aplikacji ustawiona została ikona.

4 Przykład działania programu

4.1 Rozwiązanie jednoznaczne

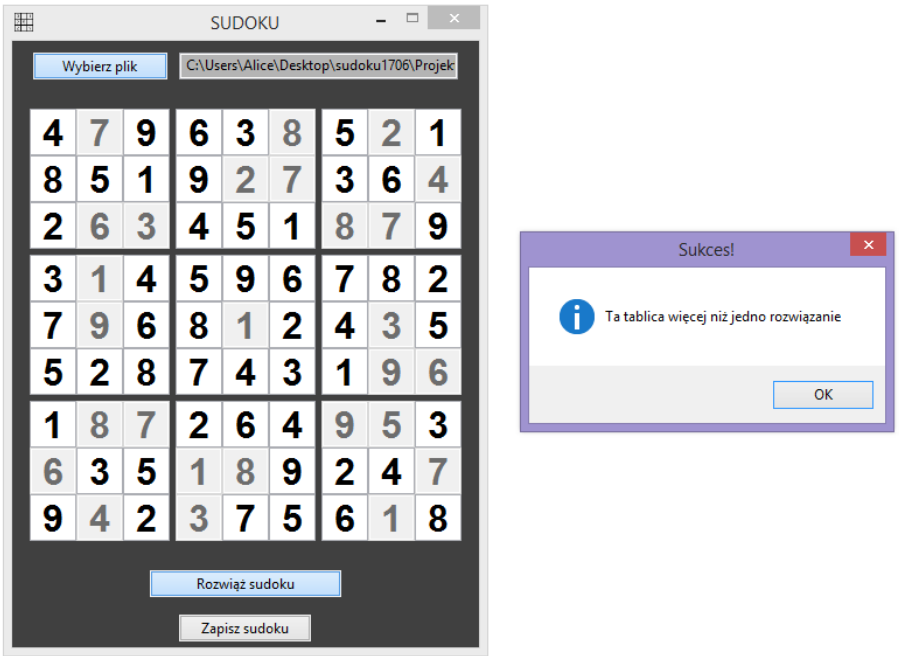
Przykład działania algorytmu w przypadku planszy sudoku, która ma tylko jedno możliwe rozwiązanie.



Rysunek 4: Tablica sudoku mająca tylko 1 rozwiązanie

4.2 Rozwiązanie niejednoznaczne

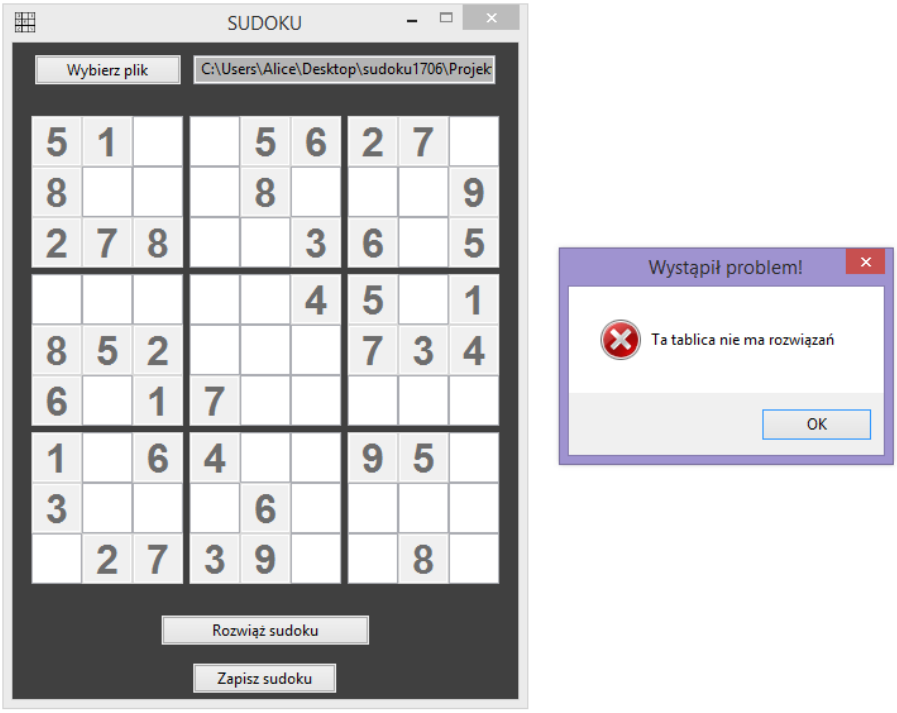
Przykład działania algorytmu w przypadku planszy sudoku, która ma więcej niż jedno rozwiazanie.



Rysunek 5: Tablica sudoku mająca więcej niż 1 możliwe rozwiązanie

4.3 Brak rozwiązań

Przykład działania algorytmu w przypadku błędnej planszy sudoku, niemożliwej do poprawnego rozwiązania.



Rysunek 6: Tablica błędna

5 Podsumowanie

Wykonanie:

- Plan projektu - Alicja Bilińska 40%, Marcin Woźniak 60%
- GUI - Alicja Bilińska
- Funkcja główna - Alicja Bilińska 50%, Marcin Woźniak 50%
- Algorytm - Marcin Woźniak
- Wczytywanie - Marcin Woźniak
- Zapisywanie - Marcin Woźniak
- Dokumentacja - wstępny zarys - Alicja Blińska; poprawki - Alicja Bilińska 60%, Marcin Woźniak 40%

Algorym został stworzony na podstawie opisu ze strony internetowej:
http://taborsudoku.pl/algorytm-standardowy/podstawy/

Rysunek 3 pochodzi ze strony internetowej:
http://taborsudoku.pl/algorytm-standardowy/podstawy/

Spis treści