**POLITECHNIKA LUBELSKA**
**WYDZIAŁ MATEMATYKI I INFORMATYKI TECHNICZNEJ**
*Kierunek: Inżynieria i analiza danych*


Projekt z zakresu programowania

Projekt zaliczeniowy

*Praca wykonana pod kierunkiem:*
**dr —— —-**

*Autorzy:*
——— ————
**nr. albumu: ——**
**Patrycja Mazur**
**nr. albumu: ——**

## Spis treści


Rysunek 1: Ikona aplikacji

## 1. Wprowadzenie

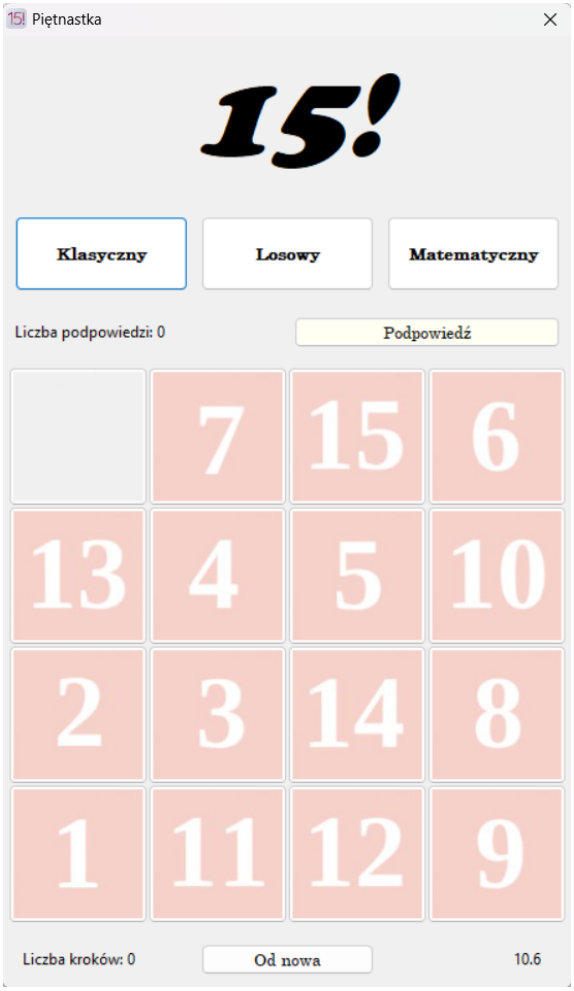
W Code::Blocks w języku programowania C++ razem stworzyłyśmy popularną łamigłówkę logiczną „Piętnastka”, w której gracz próbuje ułożyć liczby od 1 do 15 w odpowiedniej kolejności poprzez przesuwanie kafelków(klocków) po planszy.

Ikona naszej aplikacji przedstawiona jest na ilustracji 1.

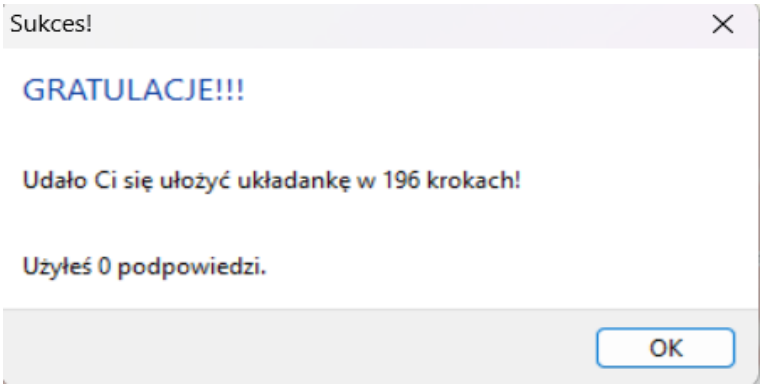
Wygląd naszej aplikacji przedstawiony jest na ilustracji 2.

Natomiast wygląd okienka wyświetlanego po ułożeniu całej układanki przedstawiony jest na ilustracji 3.

Gra ta została wynaleziona w 1878 roku przez Samuela Loyda. Jej popularność wzrosła na przestrzeni lat, stając się jedną z najbardziej znanych łamigłówek na świecie.




Rysunek 2: Ekran startowy aplikacji




Rysunek 3: Okienko wyświetlane po wygranej

## 2. Cel Gry

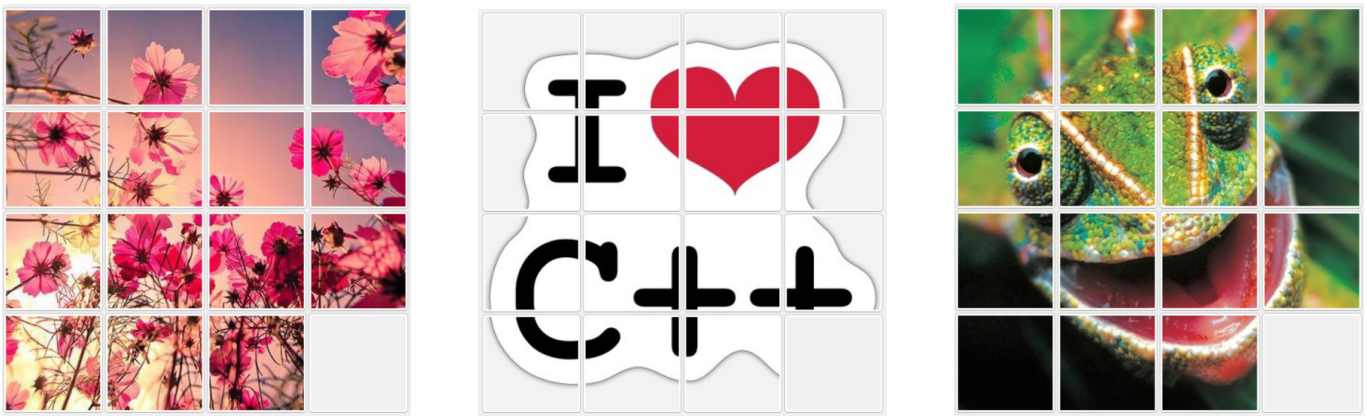
Celem gry jest przemieszczanie kafelów w taki sposób, aby ułożyć je w odpowiedniej kolejności. Do naszej gry dodane są trzy poziomy do wyboru:

— **Klasyczny** - układanie klocków od 1-15.




Rysunek 4: Poziom klasyczny

— **Losowy** - układanie wylosowanego obrazka. Przykładowe obrazki:



Rysunek 5: Poziom losowy

— **Matematyczny** - układanie klocków od 1-15, z tym że trzeba najpierw obliczyć działanie aby dowiedzieć się co jest jakim numerem.



Rysunek 6: Poziom matematyczny

Początkowy stan planszy jest wymieszany, a gra informuje gracza o zwycięstwie, gdy wszystkie kafelki stoją już na odpowiednim miejscu.

**Dodatkowe informacje**:
— Plansza składa się z kwadratowej siatki, o wymiarach 4x4.
— Na planszy znajdują się kafelki z numerami od 1 do 15 oraz jedno puste pole, aby można było przesunąć na nie sąsiedni kafelek.
— Ruchy są ograniczone do przesuwania kafelka w puste miejsce.
— Domyślnie aplikacja daje do ułożenia poziom klasyczny.

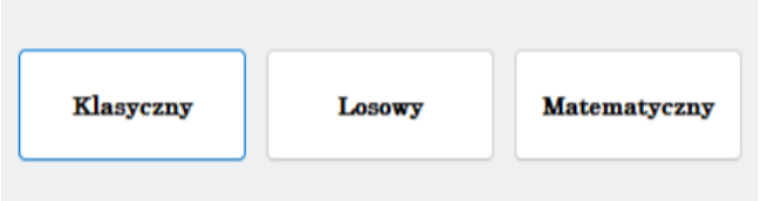
## 3. Opis interfejsu

Poniżej przedstawiamy kluczowe elementy interfejsu:

— **15! :** Umieszczona na samej górze interfejsu, nawiązuje do nazwy gry „Piętnastka”.

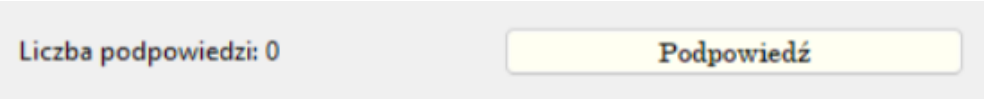

Rysunek 7: Górny wygląd interfejsu

— **Trzy przyciski z poziomami:** Bezpośrednio poniżej nazwy wyświetlane są przyciski do wyboru poziomu, który gracz zdecyduje się ułożyć. Poziom można zmienić w dowolnej chwili.



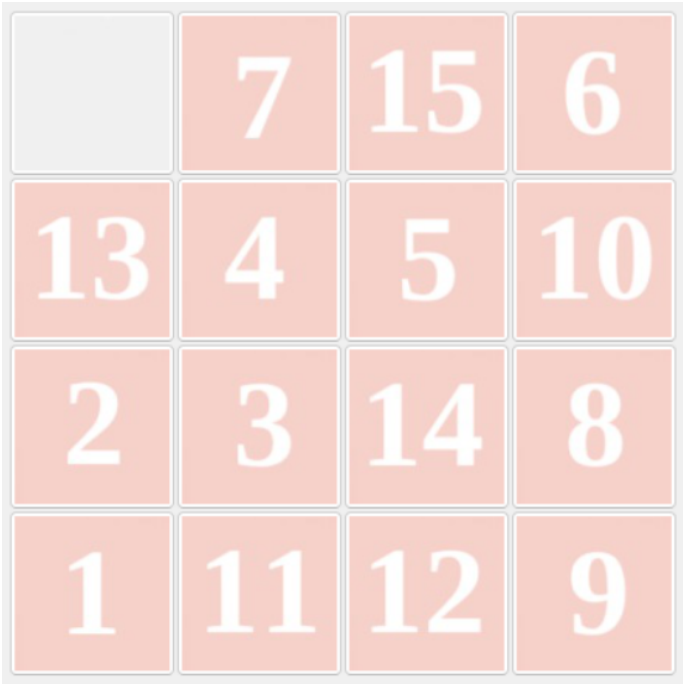

Rysunek 8: Poziomy

— **Podpowiedź:** Po prawej stronie umieszczony jest przycisk podpowiedzi. Gdy gracz go użyje to wyświetli się na 3s poprawnie ułożona układanka, a następnie wróci do swojej pomieszanej wersji. Natomiast po lewej stronie znajduje się informacja o ilości użytych podpowiedzi.



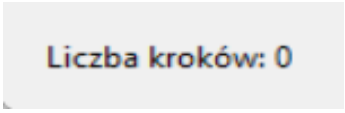

Rysunek 9: Licznik podpowiedzi oraz przycisk „Podpowiedź”

— **16 kafelków:** Główna część aplikacji, gdzie się układa wybrany obrazek, bądź liczby w odpowiedniej kolejności.




Rysunek 10: Plansza

— **Liczba kroków:** Po lewej stronie, poniżej głównej części gry ustawiona jest informacja ile kroków wykonał już gracz.

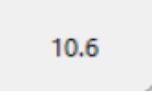



Rysunek 11: Licznik kroków

— **Nowa gra:** Przycisk znajdujący się po środku, na samym dole okienka. Poprzez jego kliknięcie, gracz dostaje nową grę na tym samym poziomie, który przed chwilą układał.


Rysunek 12: Przycisk „Od nowa”

— **Czas:** Czas gry wyświetlany jest po prawej stronie, ukazuje ile gracz poświęcił już czasu na ułożenie łamigłówki.




Rysunek 13: Licznik czasu

# 4. Architektura Systemu

Przybliżymy kilka kluczowych funkcji naszej aplikacji, opisując krótko ich działania:

## 4.1. Funkcja Pomieszaj

— **void pomieszaj();**

Funkcja **pomieszaj()** realizuje mieszanie układanki poprzez losowe przemieszczanie elementów.
Funkcja rozpoczyna od ustawienia permutacji na sekwencję liczb od 1 do 15, reprezentujących ułożoną układankę, oraz 0 dla pustego pola.
Ustawia początkową pozycję pustego pola **poz_0** na ostatni element permutacji.
Następnie, w pętli (**for (int i = 0; i < 5000; i++)**), wybiera losowe kierunki przemieszczenia pustego pola (góra, dół, lewo, prawo) i zamienia jego pozycję z sąsiednim elementem.
Mieszanie odbywa się przez zamianę miejscami pustego pola z sąsiadującym polem (liczbą od 1 do 15) w losowy sposób.
Na koniec, dla każdego przycisku w układance, ustawia odpowiedni obrazek zgodnie z uzyskaną permutacją.

```
1       void ukladankaDialog::pomieszaj() {
2        for (int i=1; i<16; i++)
3        permutacja[i-1] = i;
4        permutacja[15]=0;
5        poz_0=15;
6        for (int i = 0; i<5000; i++) {
7         int w0 = poz_0 / 4;
8         int k0 = poz_0 % 4;
9         int k,w;
10        do {
11         if (rand() % 2 == 0) {
12          k = k0 + (rand()%2)*2-1;
13          w = w0;
14         } else {
15          w = w0 + (rand()%2)*2-1;
16          k = k0;
17         }
18        } while ((k<0) || (k>3) || (w<0)|| (w>3));
19
20
21
22        int poz = w*4+k;
23        swap(permutacja[poz],permutacja[poz_0]);
24        poz_0 = poz;
25
26       }
27
28
29
30       for (int i =0; i<16; i++) {
31        pola[i]->SetBitmap(rysunki[ permutacja[i]]);
32       }
33
34      }
```

### 4.2. Funkcja CzyUlozona

— **bool CzyUlozona();**

Funkcja **CzyUlozona()** sprawdza, czy układanka jest ułożona. Przechodzi przez permutację klocków w układance i porównuje każdy element. Jeśli znajdziemy liczbę, która nie jest na swoim miejscu, funkcja zwraca **false**, w przeciwnym razie, gdy układanka jest ułożona, zwraca **true**.
Funkcja nie uwzględnia ostatniego elementu (pustego pola), ponieważ ostatni element będzie zawsze na swoim miejscu, gdy inne będą również na swoich miejscach, a więc porównanie jest tylko dla liczb od 1 do 15.

```
1       bool ukladankaDialog::CzyUlozona() {
2        for (int i = 0; i < 15; ++i) {
3         if (permutacja[i] != i + 1) {
4          return false;
5         }
6        }
7
8        return permutacja[15] == 0;
9       }
10     }
```

### 4.3. Przycisk „Od nowa”

— **void OnButton4Click1(wxCommandEvent& event);** - Przycisk „Od nowa” (ilustracja nr 12).

Po kliknięciu na przycisk **„Od nowa”** następuje wywołanie wcześniej opisywanej funkcji **pomieszaj()** 4.1, która losuje namelementy w układance. Następnie liczba kroków oraz podpowiedzi są zerowane. Licznik czasu jest ustawiany na zero oraz jest ponownie uruchamiany. Ten fragment kodu przygotowuje i resetuje różne zmienne, aby przygotować aplikację do nowej rundy gry.

```
1     void ukladankaDialog::OnButton4Click1(wxCommandEvent& event)
2     {
3      pomieszaj();
4
5      liczbaKrokow = 0;
6      wxString krokiString;
7      krokiString.Printf(_("  Liczba krokow: %d"), liczbaKrokow);
8      StaticText2->SetLabel(krokiString);
9
10     liczbaPodpowiedzi=0;
11     wxString podpString;
12     podpString.Printf(_("  Liczba podpowiedzi: %d"), liczbaPodpowiedzi);
13     StaticText4->SetLabel(podpString);
14
15     licznik = 0;
16     Timer2.Start(Timer2.GetInterval());
17
18    }
```

### 4.4. Przycisk „Podpowiedź”

— **void OnButton5Click1(wxCommandEvent& event);** - Przycisk „Podpowiedź” (ilustracja nr 9).

Po kliknięciu na przycisk **„Podpowiedź”** elementy naszej układancki są na moment (a dokładniej na 3 sekundy) pokazywane w poprawnym ułożeniu.
W pętli ustawiane są obrazki w przyciskach **pola** na kolejne obrazy z tablicy **rysunki**.
**zabronione=true;** sprawia, że ruchy gracza są zablokowane. To zapobiega interakcji z układanką podczas trwania podpowiedzi.
Następnie zwiększamy liczbę podpowiedzi o jeden i wracamy do stanu układanki przed podpowiedzią.

```
1     void ukladankaDialog::OnButton5Click1(wxCommandEvent& event)
2     {
3      for (int i =0; i<16; i++) {
4       pola[i]->SetBitmap(rysunki[(i+1)%16]);
5      }
6
7      zabronione=true;
8
9      Timer1.Start(3000,true);
10
11
12     liczbaPodpowiedzi++;
13
14     wxString podpString;
15     podpString.Printf(_("  Liczba podpowiedzi: %d"), liczbaPodpowiedzi);
16     StaticText4->SetLabel(podpString);
17
18    }
```

## 5. Napotkane problemy podczas tworzenia

Podczas tworzenia naszej aplikacji napotkałyśmy kilka problemów, większość z nich udało nam się rozwiązać. Poniżej opiszemy kilka z nich:

### 5.1. Problem z funkcją pomieszaj()

Zastanawiałyśmy się jak pozamieniać elementy na naszej planszy w odpowiedni sposób, początkowo nasza funkcja do przemieszania elementów wyglądała następująco:

```
1    void ukladankaDialog::pomieszaj() {
2     random_shuffle(permutacja, permutacja + 16);
3     for (int i=0; i< 16; i++)
4     if (permutacja[i] == 0) {
5      poz_0 = i;
6      break;
7     }
8     for (int i =0; i<16; i++) {
9      pola[i]->SetBitmap(rysunki[ permutacja[i]]);
10    }
11   }
```

Jednak po kilku próbach rozwiązywania naszej łamigłówki, okazało się, że nie wszystkie losowania dają się „ułożyć". Zazwyczaj zostawały dwa elementy na ukos, które blokowały rowiązanie zagadki. Funkcja losowo przyporządkowywała elementy, przez co mogła zdarzyć się właśnie taka sytuacja i nasza łamigłowka nie działała poprawnie.

Rozwiązaniem problemu było napisanie inaczej działającej funkcji pomieszaj() 4.1, która odbywa się przez zamianę pustego pola tylko z sąsiednim polem, co gwarantuje, że nasza układanka będzie dała się ułożyć.

### 5.2. Niewłaściwe liczenie czasu

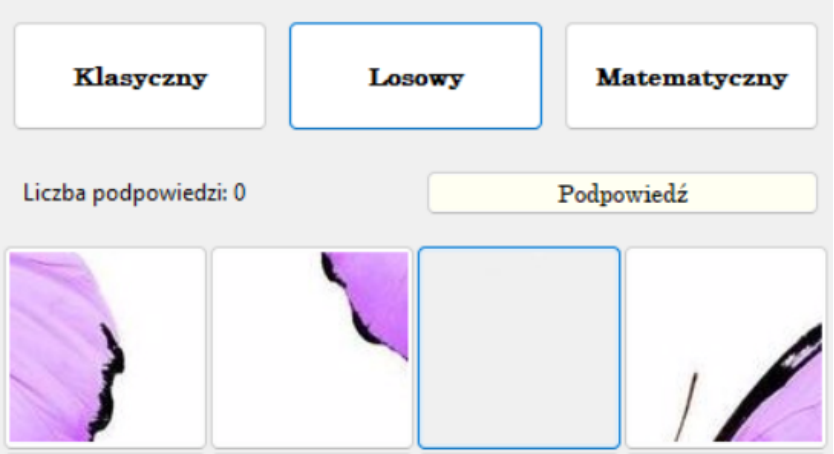
Zauważyłyśmy, że nasz Timer, który liczy ile czasu gracz poświęca na ułożenie łamigłówki nie działał poprawnie (był zbyt wolny). [sprawdzane stoperem]

Zastanawiałyśmy się, dlaczego to działa niepoprawnie. Rozwiązałyśmy ten problem (metodą prób i błędów) poprzez zmianę wartości 1000.0 na 544.0.

```
1    void ukladankaDialog::OnTimer2Trigger(wxTimerEvent& event)
2    {
3     licznik++;
4     auto czas = (Timer2.GetInterval()*licznik)/544.0;
5     wxString w = wxString::Format(wxT("%.1f"), czas);
6     StaticText3->SetLabel(w);
7    }
```

### 5.3. Podświetlane elementy

Przy poprawianiu wyglądu graficznego naszej aplikacji napotkałyśmy problem, którego nie udało nam się rozwiązać do tej pory. A mianowicie chodzi o podświetlanie elementów.




Rysunek 14: Problem z podświetlaniem elementów

Zauważyłyśmy, że elementy, które wybieramy są podświetlane na niebiesko. W naszym odczuciu ten kolor nie do końca współgrał z resztą kolorystyki i chciałyśmy go zmienić. Niestety mimo poszukiwań i prób, nie udało nam się znaleźć odpowiedzi na rozwiązanie tej zagadki.

## 6. Bibliografia

- Opracowanie własne
- Zdjęcia do układanki pobrane są z Google Grafika, oraz z prywatnych źródeł .