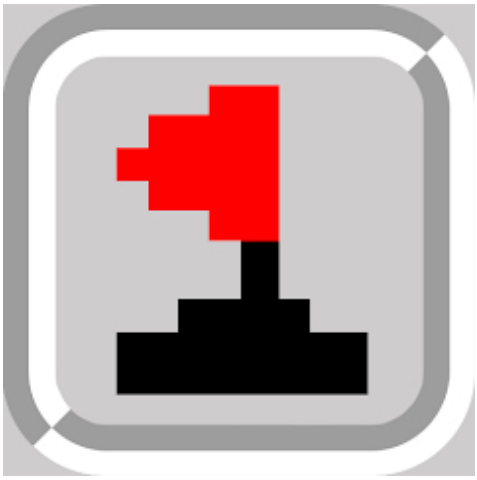
Projekt z zakresu programowania
Saper



Autorzy:

Artur Kidaj, Kamil Kopiński, Krystian Knysz

Politechnika Lubelska
Wydział podstaw techniki
Inżynieria i analiza danych
II rok

Spis treści

1 Opis aplikacji

Stworzona aplikacja ma na celu zapewnienie rozrywki użytkownikowi, którego zadaniem jest odkryć i omijać miny ukryte na planszy. Gracz klikając na pola na planszy odkrywa, czy dane pole jest bezpieczne, czy zawiera minę. Jeśli gracz odkryje pole z miną, przegrywa grę. Jeśli gracz odkryje wszystkie pola bez min, wygrywa grę. W miarę odkrywania pól, gracz może wyznaczać obszary, w których na pewno nie ma min, co pomaga mu w dalszym odkrywaniu planszy. Gra wymaga myślenia strategicznego i przewidywania, aby uniknąć min i zakończyć grę sukcesem.

Po uruchomieniu aplikacji wyświetli nam się menu główne (patrz: Rysunek 1)
Składa się ono z trzech przycisków:

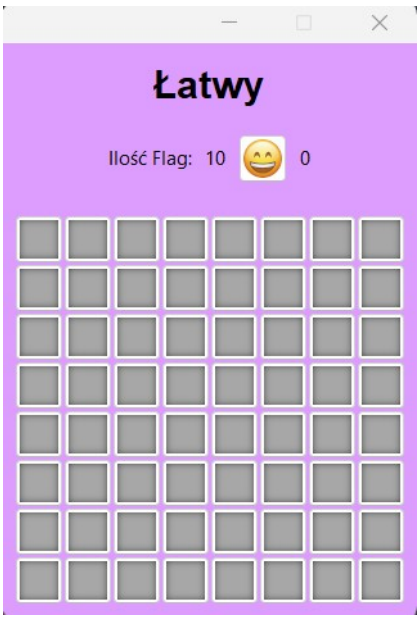
1. Łatwy
2. Średni
3. Trudny

Naciśnięcie jednego z nich przekierowuje nas do planszy z grą z wcześniej wybranym poziomem trudności.

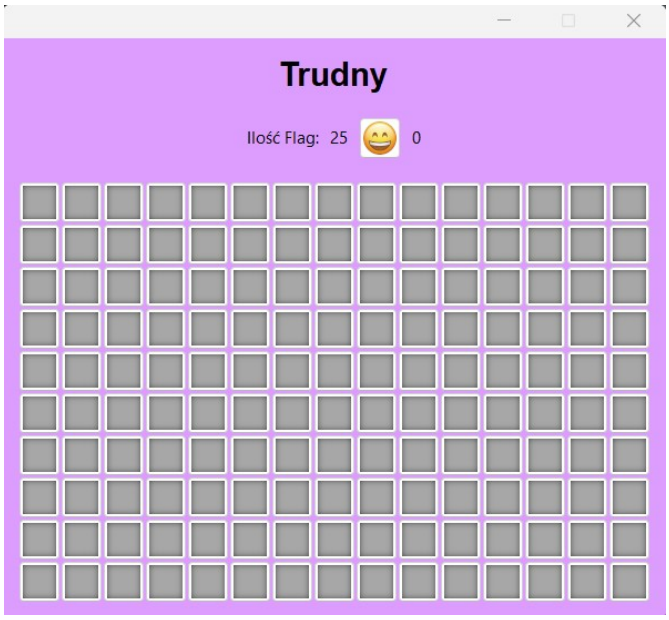
Rysunek 1: Menu główne

2 Plansze z grą

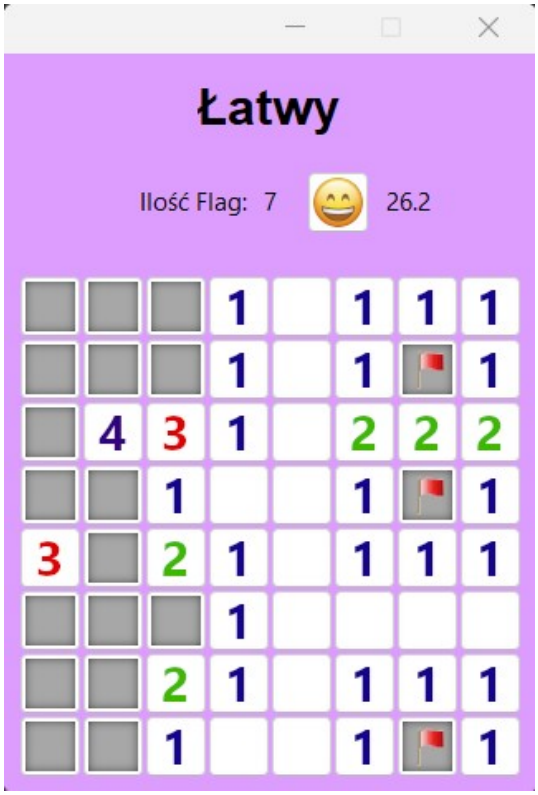
Po wybraniu odpowiedniego dla nas poziomu trudności (”Łatwy”, ”Średni”, ”Trudny”) zostajemy przekierowani do odpowiedniej planszy z grą.
Poziom ”Łatwy” (patrz: Rysunek 2) posiada planszę 8 x 8 oraz 10 bomb,
Poziom ”Średni” (wygląda identycznie jak poziom Łatwy) posiada planszę 10 x 10 oraz 15 bomb,
Poziom ”Trudny” (patrz: Rysunek 3) posiada planszę 10 x 15 oraz 25 bomb.
Po kliknięciu lewym przyciskiem, na któreś z pól ma ono 3 możliwości przemiany, albo zamieni się na numer (1-9) oznaczający ile wokół niego jest bomb, albo na pole puste, wtedy odkrywają się wszystkie pola wokół niego, lub ostatecznie na bombę, która kończy grę. Kliknięcie prawym przyciskiem na jakiekolwiek nie wybrane jeszcze pole spowoduje pojawienie się tam flagi czyli uwagi dla gracza, że w tym miejscu może być bomba. Nad planszą do gry zaczynając od lewej znajdują się: napis ”Ilość flag”, który dynamicznie pokazuje ile jeszcze zostało nam flag do rozdysponowania, obok niego jest grafika uśmiechniętej buzi po kliknięciu w którą nastąpi reset gry, natomiast po porażce zmienia się ona na smutną, po prawej jest licznik czasu naszej gry. Natomiast nad nimi znajduje się napis oznaczający na jakim poziomie trudności aktualnie gramy. Tak wygląda przykładowa plansza w trakcie gry na poziomie ”Łatwym” (patrz: Rysunek 4), a tak po wywołaniu bomby (patrz: Rysunek 5).



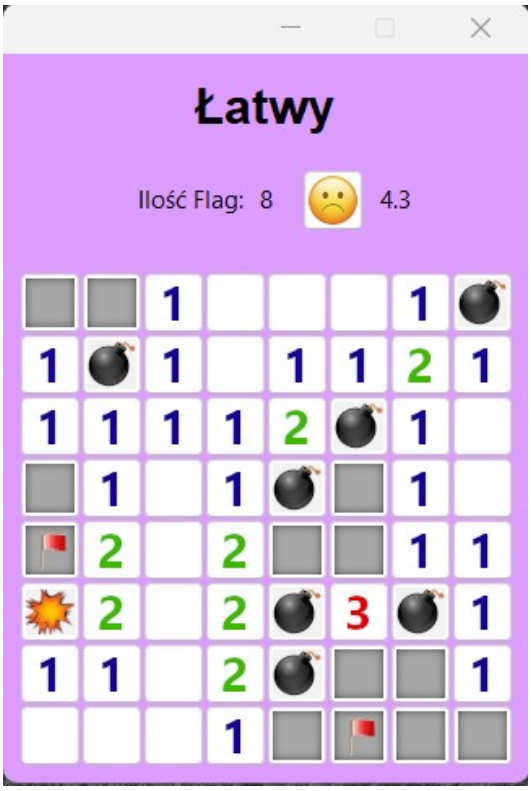
Rysunek 2: Poziom Łatwy



Rysunek 3: Poziom Trudny



Rysunek 4: Plansza podczas gry



Rysunek 5: Plansza po przegranej

3 Zaimplementowane klasy

3.1 Klasa pole

```
    class pole
{
    public:
        pole();
        virtual ~pole();
        int GetWartosc(){return wartosc;};
        int GetNumer(){return numer;};
        int GetZajete(){return zajete;};
        void SetWartosc(int n){wartosc = n;};
        void SetNumer(int n){numer = n;};
        void SetZajete(int z){zajete = z;};
        void ZwiekszWartosc(){wartosc++;};

    private:
        int wartosc;
        int numer;
        int zajete;
};
```

Klasa "pole" służy do opisywania pojedynczego pola w grze "Saper". Zawiera ona kilka funkcji i zmiennych takich jak wartość pola, numer pola i informację, czy jest ono już zajęte przez minę. Funkcje te pozwalają na uzyskanie informacji o polu oraz na zmianę jego wartości. Zmienna "wartość" jest używana do opisywania liczby min wokół pola, "numer" jest identyfikatorem pola, a "zajęte" informuje, czy pole zostało już zajęte przez minę

3.2 Klasa Gra

```
    class Gra
{
    public:
        Gra(int B);
        virtual ~Gra();
        void Zresetuj(int L);
        bool Wygrana();

        void Zmniejsz(){LPol--;};
        void ZmniejszF(){Flagi--;};
        void ZwiekszF(){Flagi++;};
        void SetStanGry(int n){StanGry = n;};
        int GetLiczby(int n){return Liczby[n];};
        int GetStanGry(){return StanGry;};
        int GetFlagi(){return Flagi;};
    protected:

    private:
        int Bomby;
        int Flagi;
        int LPol;
        int *Liczby;
        int StanGry;

};
```

Klasa "Gra" służy do opisywania stanu gry. Zawiera ona kilka funkcji i zmiennych takich jak liczba min, liczba flag, pozostała liczba nieodkrytych pól, tablica zawierająca informacje o liczbie min wokół poszczególnych pól i stan gry (np. wygrana lub przegrana). Funkcje te pozwalają na uzyskanie informacji o stanie gry oraz na jego zmianę. Zmienna "Bomby" jest używana do opisywania liczby min w grze, "Flagi" jest używana do opisywania liczby flag, które zostały już ustawione, "LPol" opisuje liczbę pozostałych nieodkrytych pól, "Liczby" to tablica zawierająca informacje o liczbie min wokół poszczególnych pól, a "StanGry" jest zmienną, która określa, czy gra została już wygrana lub przegrana.

```
Gra::Gra(int B){
    srand(time(0));
    Bomby = B;
    Liczby = new int[B];
}

Gra::~Gra()
{
    delete []Liczby;
}

void Gra::Zresetuj(int L){
    StanGry = 0;
    Flagi = Bomby;
    LPol = L;
    int i = 0;
    int j = 0;
    while(i<Bomby){
        Liczby[i] = rand()%(LPol-1);
        while(j<i){
           if(Liczby[i] == Liczby[j])
                i--;
            j++;
        }
        i++;
        j=0;}
}

bool Gra::Wygrana(){
    if(LPol-Bomby == 0)
        return true;
    return false;
}
```

Konstruktor jest używany do inicjalizacji obiektu tej klasy. Inicjalizuje on zmienną "Bomby" z wartością przekazaną jako argument i alokuje pamięć dla tablicy "Liczby". Funkcja "srand(time(0))" jest używana do inicjalizacji generatora liczb losowych.

Funkcja "Zresetuj(int L)" jest używana do resetowania stanu gry. Ustawia ona zmienną "StanGry" na 0, "Flagi" na wartość "Bomby", "LPol" na wartość przekazaną jako argument "L" i wypełnia tablicę "Liczby" liczbami losowymi.

Funkcja "Wygrana()" jest używana do sprawdzenia, czy gra została wygrana. Sprawdza, czy liczba pozostałych nieodkrytych pól jest równa 0, a jeśli tak, zwraca wartość true, w przeciwnym wypadku zwraca wartość false.

4 Zaimplementowane funkcje

4.1 Funkcja reset

Celem funkcji reset planszy a następnie przygotowanie jej do rozgrywki.

```
    BitmapButton64->SetBitmap(rysunki[13]);
    Timer1.Stop();
    ile_razy = 0;
    G->Zresetuj(64);

    StaticText3 -> SetLabel("10");
    StaticText4 -> SetLabel("0");


    for (int i = 0; i<=63;i++){
        plansza[i] -> SetBitmap(rysunki[0]);
    }
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            P[i][j].SetZajete(0);
            P[i][j].SetWartosc(0);}
    }

    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            for(int m=0;m<10;m++){
                if(P[i][j].GetNumer()==G->GetLiczby(m)){
                    P[i][j].SetWartosc(9);
                    for(int k = -1; k<=1; k++){
                        for(int l = -1; l<=1; l++){
                            if((i+k>=0)&&(i+k<8)&&(j+l>=0)&&(j+l<8))
                                P[i+k][j+l].ZwiekszWartosc();}}}}}
}
```

Najpierw zmieniany jest rysunek resetu oraz zatrzymywany i resetowany jest stoper. Zostaje wykonana funkcja Zresetuj o parametrze równej liczbie pól w danym trybie. Następnie wszystkim bitmapom zostaje ustawiony rysunek zakrytego pola, po czym wszystkim polom zostaje ustawiony status niezajęty a wartość jest zerowana. W dalszej kolejności poruszając się po tablicy dwuwymiarowej sprawdzane jest, czy id tego pola równe jest z jedną z wylosowanych liczb znajdujących się w klasie gra. Jeśli tak to oznacza to, że na tym polu znajduje się bomba. Wartość tego pola zostaje ustawiona na 9 a wszystkim sąsiadującym polom wartość zostaje zwiększona o jeden. Potem funkcja kończy działanie.

4.2 Funkcja odkryj

Celem funkcji jest odkrycie klikniętego pola.

```
void dialogLatwy::odkryj(int x,int y){
    int id = 8*y+x;
    if(P[x][y].GetZajete()==0){
        P[x][y].SetZajete(1);
        if(P[x][y].GetWartosc()>=9){
            for(int i=0;i<8;i++){
                for(int j=0;j<8;j++){
                    if((P[i][j].GetWartosc()>=9)&&(P[i][j].GetZajete()==0)){
                        plansza[j*8+i]->SetBitmap(rysunki[10]);
                    }
                    else if((P[i][j].GetZajete()==2)&&(P[i][j].GetWartosc()<9))
                        plansza[j*8+i]->SetBitmap(rysunki[15]);
                }
            }
            plansza[id]->SetBitmap(rysunki[11]);
            Timer1.Stop();
            G->SetStanGry(2);
            BitmapButton64->SetBitmap(rysunki[14]);
        }
        else if(P[x][y].GetWartosc()==0){
                G->Zmniejsz();
                plansza[id] -> SetBitmap(rysunki[1]);
            for(int k = -1; k<=1; k++){
                for(int l = -1; l<=1; l++){
                    if((x+k>=0)&&(x+k<8)&&(y+l>=0)&&(y+l<8)){
                        odkryj(x+k,y+l);
                      }
                }
            }
        }
        else{
            plansza[id] -> SetBitmap(rysunki[P[x][y].GetWartosc()+1]);
            G->Zmniejsz();
        }
    }
}
```

Najpierw tworzona jest zmienna id reprezentująca id pola wyliczonego ze współrzędnych. Następnie sprawdzane jest, czy pole nie zostało zajęte. Jeśli tak funkcja kontynuuje działanie. W przeciwnym wypadku funkcja kończy działanie. Potem pole zostaje zajęte i sprawdzane jest, czy kliknięto pole z bombą. Jeśli tak to przechodząc przez wszystkie pola sprawdzamy najpierw czy dane pole zostało nieodkryte i znajduje się w nim bomba a następnie czy oflagowane zostały pola nie zawierające bomb. W pierwszym przypadku zostaje ustawiony rysunek bomby a w drugim rysunek błędnego oflagowania. Po wyjściu z pętli timer zostaje zatrzymany, stan gry zostaje zmieniony na zakończony, rysunek bomby zmieniony na wybuch a rynek resetu na przegraną. Natomiast jeśli pole ma wartość równą zero to zmniejszamy to ustawiany jest rysunek pustego pola a liczba pozostałych pól zmniejsza się o jeden. Następnie wszystkie sąsiadujące pole wywołują funkcję odkryj z odpowiednimi dla siebie współrzędnymi. W przeciwnym wypadku zostaje ustawiony rysunek przedstawiający liczbę równą wartości pola a liczba pozostałych pól zmniejsza się o jeden. Po wszystkim funkcja zostaje zakończona.

5 Kolejność tworzenia programu i jego funkcji

Na początku swojej pracy stworzyliśmy wxDialog jako okno początkowe. Umieściliśmy w nim 3 przyciski "Łatwy", "Średni", "Trudny", które mają implementację prawie taką samą, mianowicie

```
    void SaperDialog::OnButton1Click(wxCommandEvent& event)
{
    dialogLatwy * dg = new dialogLatwy(this);
    dg -> ShowModal();
    delete dg;
}
```

Następnie stworzyliśmy pierwszy poziom "Łatwy", które zawiera 64 pola wxBitmapButton
Za pomocą makra dodaliśmy każdemu polu jego obsługę

```
      #define par(nr) plansza[nr] = BitmapButton##nr;
        par(0); par(1); par(2); ... par(63);
    #undef par
```

Kolejno do każdego elementu tablicy "rysunki" przyporządkowaliśmy odpowiednią grafikę

```
    rysunki[0]= wxBitmap(wxImage(_T("rysunki/kwadrat.png")));
    rysunki[1]= wxBitmap(wxImage(_T("rysunki/0.png")));
    rysunki[2]= wxBitmap(wxImage(_T("rysunki/1.png")));
    rysunki[3]= wxBitmap(wxImage(_T("rysunki/2.png")));
    rysunki[4]= wxBitmap(wxImage(_T("rysunki/3.png")));
    rysunki[5]= wxBitmap(wxImage(_T("rysunki/4.png")));
    rysunki[6]= wxBitmap(wxImage(_T("rysunki/5.png")));
    rysunki[7]= wxBitmap(wxImage(_T("rysunki/6.png")));
    rysunki[8]= wxBitmap(wxImage(_T("rysunki/7.png")));
    rysunki[9]= wxBitmap(wxImage(_T("rysunki/8.png")));
    rysunki[10]= wxBitmap(wxImage(_T("rysunki/mina.png")));
    rysunki[11]= wxBitmap(wxImage(_T("rysunki/wybuch.png")));
    rysunki[12]= wxBitmap(wxImage(_T("rysunki/flaga.png")));
    rysunki[13]= wxBitmap(wxImage(_T("rysunki/reset.png")));
    rysunki[14]= wxBitmap(wxImage(_T("rysunki/resetp.png")));
    rysunki[15]= wxBitmap(wxImage(_T("rysunki/pudlo.png")));
```

Za pomocą pętli przypisujemy do każdego elementu z tablicy "plansza" działanie na kliknięcie lewego i prawego przycisku myszy Funkcja "Connect" umożliwia powiązanie zdarzenia, takiego jak kliknięcie przycisku, z funkcją "OnBitmapButton0Click". Za pomocą funkcji "SetBitmap" ustawiamy każde pole na identyczną grafikę początkową. Na końcu dopasowujemy rozmiar okna do treści w nim zawartej dzięki funkcji "Fit()"

```
    for (int i = 0; i<=63;i++){
        Connect(plansza[i]->GetId(),wxEVT_COMMAND_BUTTON_CLICKED,(
    wxObjectEventFunction)&dialogLatwy::OnBitmapButton0Click);
        plansza[i]->Connect(wxEVT_RIGHT_DOWN,(wxObjectEventFunction)&dialogLatwy::
    OnRightClick, NULL, this);
        plansza[i] -> SetBitmap(rysunki[0]);
    }
    this->Fit();
```

Później zrobiliśmy pętle, która jest używana do tworzenia tablicy obiektów klasy "pole". Pętla zewnętrzna odpowiada za iterację wierszy, a pętla wewnętrzna odpowiada za iterację kolumn. Zmienna "num" jest używana do nadawania numeru każdemu obiektowi klasy "pole".

```
   for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            P[i][j].SetNumer(num);
            num++;}}
```

Następnie zaimplementowaliśmy funkcje: reset (patrz: (4.1)) i odkryj (patrz: (4.2)) opisane powyżej. Zaprogramowaliśmy co się dzieje po kliknięciu w BitmapButton

```
void dialogLatwy::OnBitmapButton0Click(wxCommandEvent& event)
{
    if(G->GetStanGry() == 0){
        Timer1.Start();
        G->SetStanGry(1);
    }
    if(G->GetStanGry() == 1){
        int i = event.GetId()-372;
        int X,Y;
        X = i%8;
        Y = (i-X)/8;
        odkryj(X,Y);

        if(G->Wygrana()){
            Timer1.Stop();
            wxMessageBox("Wygrana","Rezultat");
            reset();}}}
```

Celem funkcji jest odkrycie klikniętego pola.
Najpierw sprawdzane jest, czy gra nie została jeszcze rozpoczęta. Jeśli tak, to timer zostaje uruchomiony a stan gry zostaje zmieniony na „w toku". Następnie sprawdzane jest, czy gra jest w toku. Jeśli tak, to funkcja kontynuuje działanie. W przeciwnym wypadku funkcja kończy działanie. Następnie tworzone są 3 zmienne całkowite: i, X, Y. Zmienna i przechowuje id klikniętej bitmapy. Jednak po pobraniu liczby za pomocą event.GetId id jest większa od pożądanej wartości o wartość zależną od wybranego poziomu trudności (Łatwy – 373, Średni – 267, Trudny - 111). Dlatego następuje odjęcie jej. Zmienne X oraz Y przechowują współrzędne umiejscowienia w tablicy dwuwymiarowej pola utożsamianego z klikniętą bitmapą. Zmienne te są obliczane z wcześniej pobranego id. Następnie zostaje wywołana funkcja odkryj z parametrami X oraz Y. Po jej zakończeniu sprawdzana jest ewentualna wygrana. Jeśli tak to timer zostaje zatrzymany, stan gry zostaje zmieniony na zakończony oraz zostaje wywołana funkcja reset. Po wszystkim funkcja kończy działanie.

Dodaliśmy działanie prawego przycisku

```
void dialogLatwy::OnRightClick(wxCommandEvent& event)
{

    if(G->GetStanGry() == 1){
        int i = event.GetId() - 372;
        int X,Y;
        X = i%8;
        Y = (i-X)/8;

        if(P[X][Y].GetZajete()==0 && G->GetFlagi() > 0){
            P[X][Y].SetZajete(2);
            plansza[i]-> SetBitmap(rysunki[12]);
            G->ZmniejszF();
        }
        else if(P[X][Y].GetZajete()==2){
            P[X][Y].SetZajete(0);
            plansza[i]-> SetBitmap(rysunki[0]);
            G->ZwiekszF();
        }
        wxString w;
        int x = G->GetFlagi();
        w<< x;
        StaticText3 -> SetLabel(w); }}
```

Celem funkcji jest ustawienie lub zdjęcie flagi sygnalizującej naszą domniemaną spekulacje odnośnie położenia bomby.
Najpierw sprawdzane jest, czy gra jest w toku. Jeśli tak, to funkcja kontynuuje działanie. W przeciwnym wypadku funkcja kończy działanie. Następnie tworzone są 3 zmienne całkowite: i, X, Y. Zmienna i przechowuje id klikniętej bitmapy. Jednak po pobraniu liczby za pomocą event.GetId id jest większa od pożądanej wartości o wartość zależną od wybranego poziomu trudności (Łatwy – 373, Średni – 267, Trudny - 111). Dlatego następuje odjęcie jej. Zmienne X oraz Y przechowują współrzędne umiejscowienia w tablicy dwuwymiarowej pola utożsamianego z klikniętą bitmapą. Zmienne te są obliczane z wcześniej pobranego id. Następnie sprawdzane jest, czy pole nie zostało jeszcze odkryte oraz czy dostępne są flagi. Jeśli tak to pole to zostaje ustawione jako zajęte przez flagę, ustawiany jest rysunek flagi a liczba dostępnych flag zmniejsza się o jeden. Natomiast jeśli na kliknięty polu ustawiona jest już flaga, to zostaje ono ustawione jako nieodkryte, ustawiany jest rysunek zakrytego pola a liczba dostępnych flag zwiększa się o jeden. Po wszystkim uaktualniona jest liczba dostępnych flag i funkcja kończy działanie.

Dodaliśmy BitmapButton "buźkę", która resetuje grę na podstawie funkcji (patrz: Funkcja 4.1)

```
void dialogLatwy::OnBitmapButton64Click1(wxCommandEvent& event)
{
    reset();
}
```

Dodaliśmy również timer zliczający ile trwa dana rozgrywka

```
void dialogLatwy::OnTimer1Trigger(wxTimerEvent& event)
{
    ile_razy++;
```

```
    double c = ile_razy * Timer1.GetInterval()/700.0;
    czas = wxString::Format("%.1f",c);
    StaticText4 -> SetLabel(czas);
}
```

Funkcja "OnTimer1Trigger" jest funkcją zdarzeniową, która jest wywoływana po każdym przebiegu zegara Timer1. Wewnątrz funkcji obliczana jest zmienna "c", która jest równa ilości czasu upłyniętego od uruchomienia zegara.

Następnie jest tworzony łańcuch znaków "czas" za pomocą wxString::Format z zaokrągloną wartością "c". Ostatecznie tekst na etykiecie "StaticText4" jest ustawiany na wartość tego łańcucha znaków.

Na bierząco w trakcie pisania programu dodawaliśmy do biblioteki dialogLatwy.h deklarowane zmiennie

```
    wxBitmapButton *plansza[64];
    wxBitmap rysunki[16];
    void OnRightClick(wxCommandEvent& event);
    Gra *G = new Gra(10);
    pole P[8][8];

    int num = 0;
    int ile_razy = 0;
    string czas;
```

Treść poziomów "Średni" i "Trudny" wygląda identycznie jak "Łatwy" lecz ze zmianionymi gdzieniegdzie liczbami.

Procentowy udział w tworzeniu projektu

- Tworzenie funkcji:
  Artur Kidaj - 60%, Kamil Kopiński - 20%, Krystian Knysz - 20%
- Tworzenie klas:
  Artur Kidaj - 70%, Kamil Kopiński - 17%, Krystian Knysz - 13%
- Tworzenie plansz do gry:
  Artur Kidaj - 30%, Kamil Kopiński - 30%, Krystian Knysz - 40%
- Wygląd gry:
  Artur Kidaj - 20%, Kamil Kopiński - 15%, Krystian Knysz - 65%
- Tworzenie pliku instalacyjnego:
  Artur Kidaj - 10%, Kamil Kopiński - 70%, Krystian Knysz - 20%
- Tworzenie dokumentacji:
  Artur Kidaj - 20%, Kamil Kopiński - 70%, Krystian Knysz - 10%