

**Projekt nr 13 z przedmiotu**  
**Wprowadzenie Do Grafiki Komputerowej:**  
*„Lustrzane puzzle”*

**Michał Matyl**  
**Michał Nowak**  
**Angelika Maja Szymańska**

## Spis treści

Spis treści .....	2
1. Tytuł projektu i autorzy projektu .....	3
2. Opis projektu .....	3
3. Założenia wstępne przyjęte w realizacji projektu .....	3
4. Analiza projektu .....	3
4.1 Specyfikacja danych wejściowych .....	3
4.2 Opis oczekiwanych danych wyjściowych .....	4
4.3 Zdefiniowanie struktur danych .....	4
4.4 Specyfikacja interfejsu użytkownika .....	4
4.5 Wyodrębnienie i zdefiniowanie zadań .....	5
4.6 Decyzja o wyborze narzędzi programistycznych .....	5
5. Podział pracy i analiza czasowa .....	5
6. Opracowanie i opis niezbędnych algorytmów .....	6
6.1. Wczytywanie pliku obrazu z dysku .....	6
6.2. Obróbka wczytanego pliku obrazu .....	6
6.3. Tasowanie współrzędnych, lustrzane odbicia puzzli .....	6
6.4. Wyświetlanie puzzli .....	6
6.5. Śledzenie ruchu wskaźnika myszy .....	7
6.6. Wyznaczanie sąsiadów puzzla pod wskaźnikiem myszy .....	7
6.7. Śledzenie kliknięć .....	7
6.8. Wyznaczanie punktacji .....	7
6.9. Generator obrazów .....	7
6.10. Zapis danych do pliku .....	7
6.11. Odczyt danych z pliku .....	8
7. Kodowanie .....	8
7.1. Schemat blokowy .....	8
7.2. Opis klasy Gra: zmienne i funkcje .....	9
8. Testowanie .....	16
9. Wdrożenie, raport i wnioski .....	16

## **1. Tytuł projektu i autorzy projektu**

Celem projektu „*Lustrzane puzzle*” jest napisanie gry komputerowej polegającej na ułożeniu obrazka z pomieszanych elementów. Do zespołu pracującego nad projektem należą:

Michał Matyl

Michał Nowak

Angelika Szymańska

## **2. Opis projektu**

„*Lustrzane puzzle*” jest grą komputerową, w której gracz ma za zadanie ułożenie obrazka z jego pomieszanych części. Po wczytaniu obrazka zostaje on podzielony na mniejsze części, które dodatkowo mogą być odbite w pionie, poziome lub w obu tych kierunkach jednocześnie. Jeżeli użytkownik wskaże myszką na puzzel wyświetlane są czerwone kwadraciki, które reprezentują możliwe operacje na określonej grupie puzzli, lub na pojedynczym puzzlu. Kliknięcie jednego z kwadracików powoduje pokazanie możliwych ruchów, czyli obrotów, lub zmiany pozycji elementów układanki. Koniec gry następuje po ułożeniu obrazka, lub po przerwaniu gry przez użytkownika.

## **3. Założenia wstępne przyjęte w realizacji projektu**

- wczytanie obrazka wybranego przez użytkownika
- obracanie pojedynczych puzzli, grupy puzzli
- generowanie obrazka
- przywoływanie prawidłowo ułożonego obrazka przez wciśnięcie klawisza [Tab]
- wybieranie przez gracza liczby części, na jakie obrazek ma zostać podzielony (liczba kolumn i wierszy)
- zmienianie koloru puzzli na czarno-białe
- rozjaśnianie/ściemnianie po najechaniu na puzzel, oraz zwiększanie/zmniejszanie szybkości tych zmian, oraz możliwość wyłączenia tej opcji
- pomoc do programu

## **4. Analiza projektu**

### **4.1 Specyfikacja danych wejściowych**

Dane wejściowe to plik graficzny. Program ma możliwość wczytania obrazka w formacie BMP, GIF, JPEG, PCX, PNG, PNM, TIFF, oraz XPM. Obrazek jest skalowany jeżeli jego szerokość jest mniejsza niż 600px lub większa niż 800px, lub gdy jego wysokość jest mniejsza niż 500px, lub większa niż 600px.

## 4.2 Opis oczekiwanych danych wyjściowych

Po ułożeniu obrazka, gracz może zapisać swoje wyniki - liczbę punktów, oraz czas w tabeli wyników. Dziesięć najlepszych wyników zapisanych jest w pliku *highscore.dat*

## 4.3 Zdefiniowanie struktur danych

W programie niezbędne jest przechowywanie informacji o puzzlach oraz danych dotyczących akcji wykonywanych przez użytkownika. W tym celu została zaprojektowana klasa o nazwie *Gra*, zawierająca wszystkie niezbędne zmienne. Zmienne zostały zadeklarowane jako prywatne a do operacji na nich służy szereg metod, których nazwy odnoszą się do odpowiedniej zmiennej. Nazwy większości metod mają przedrostek *Set*, wskazujący, że metoda służy do ustawienia żądanej wartości danej zmiennej, oraz przedrostek *Get*, wskazujący, że metoda pozwala sprawdzić wartość danej zmiennej. Szczegółowy opis wszystkich zmiennych i metod klasy *Gra* jest zawarty w rozdziale 7.2.

## 4.4 Specyfikacja interfejsu użytkownika

Okno główne programu zbudowane jest z dwóch części: komunikacji z użytkownikiem, oraz okna gdzie rysowane są puzzle.

Części komunikacji z użytkownikiem składa się z sześciu przycisków. Po najechaniu myszką na przycisk pojawia się z informacja o jego funkcji:

— Przycisk *Wczytaj obraz* otwiera okno dialogowe pozwalające wybrać użytkownikowi obrazek. Podczas całej rozgrywki dostępne są wszystkie przyciski z okna głównego. Jeżeli w czasie gry użytkownik będzie chciał wczytać inny obrazek, lub wygenerować obrazek, pojawi się okno modalne z zapytaniem czy na pewno chce zakończyć bieżącą grę.

— Klikając przycisk *Opcje* otwiera okno w którym użytkownik programu może zwiększyć lub zmniejszyć liczbę kolumn i wierszy na które zostanie podzielony obrazek. Nie ma możliwości zmiany liczby puzzli w czasie trwającej gry. Liczbę kolumn i wierszy można zmienić od dwóch do czterdziestu. W oknie opcji można również włączać, oraz zmieniać szybkość rozjaśniania i ściemniania puzzli. Gracz może także zmienić kolor obrazka na czarno-biały.

— Po kliknięciu przycisku *Generuj obraz* program tworzy obrazek.

— Pomoc w formie strony www zawiera opis możliwości programu, jego funkcje i dostępne opcje. Pojawia się po wciśnięciu przycisku *Pomoc*.

— Przycisk *Najlepsi* otwiera okno z wynikami najlepszych graczy. Tabela zawiera imiona dziesięciu najlepszych graczy, liczbę uzyskanych przez nich punktów, czas rozgrywki, nazwę wczytanego obrazka (nazwę „*Generator*” jeżeli obrazek został wygenerowany przez program), oraz informację o podziale obrazka (liczba kolumny x liczba wierszy).

— Klikając przycisk *O!* użytkownik może dowiedzieć się podstawowych informacji o programie takich jak; autorzy, licencja, wersja programu.

Po wczytaniu pliku pojawia się okno gry z podzielonymi i pomieszanymi częściami obrazka. Po najechnaniu na dowolny puzzel pojawiają się czerwone kwadraty pokazujące graczowi dostępne ruchy. W zależności od ilości „sąsiadów” puzzla dostępne są cztery, sześć lub dziewięć możliwości obrotu i zmiany położenia elementów. Każdą dostępną zmianę pokazują strzałki pojawiające się po kliknięciu na kwadrat. Po ułożeniu obrazka, jeżeli liczba uzyskanych punktów jest wynikiem wystarczającym do znalezienia się wśród dziesięciu najlepszych graczy, program zapisze wynik w tabeli. W przeciwnym razie wynik i czas rozgrywki pokażą się w pasku poniżej okna gry.

#### 4.5 Wyodrębnienie i zdefiniowanie zadań

Całość projektu można podzielić na następujące zadania:

- wczytywanie obrazka, jego podział
- tasowanie i obracanie puzzli
- komunikacja z graczem, wygląd okien
- zapis wyników, tworzenie tabeli najlepszych i obliczanie punktacji

#### 4.6 Decyzja o wyborze narzędzi programistycznych

Projekt został napisany w języku C++, z wykorzystaniem biblioteki *wxWidgets*. Użyto środowiska *wxDev-C++*, ponieważ ma on możliwość szybkiego i prostego projektowania wyglądu aplikacji *wxFrame*, oraz umożliwia łatwe podłączanie metod obiektów do zdarzeń. Natomiast biblioteka *wxWidgets* jest bardzo dobrze opisana w dokumentacji biblioteki.

### 5. Podział pracy i analiza czasowa

praca/zadanie	kto	czas
przygotowanie wyglądu programu	MM	1 tydzień
wczytywanie obrazka, podział obrazka	MM	3 dni
obracanie, tasowanie	MM	3 dni
sterowanie	MN, MM	3 dni
dodatkowe opcje	MN, AMS	2 dni
komunikacja z graczem	AMS	2 dni
wyniki, czas, zapis wyników	AMS	2 dni
testowanie	zespół	1 dzień
poprawa błędów	MN	2 dni
testowanie	MN	1 dzień
dokumentacja	MM, AMS	3 dni

## 6. Opracowanie i opis niezbędnych algorytmów

### 6.1. Wczytywanie pliku obrazu z dysku

Wczytywanie pliku obrazu odbywa się po wystąpieniu w programie zdarzenia kliknięcia przycisku *Wczytaj obraz*. Po wystąpieniu tego zdarzenia sprawdzany jest stan rozgrywki na podstawie wartości zmiennej *gra\_w\_toku* w klasie *Gra*. Jeżeli jej wartość wskazuje, że gracz jest w trakcie rozgrywki, wówczas program zapyta czy bieżąca gra ma zostać zakończona. W przeciwnym wypadku wybrany przez gracza z dysku plik graficzny zostanie przeskalowany do wielkości nieprzekraczającej 800x600px i umieszczony w zmiennej *obrazek* w klasie *Gra*.

### 6.2. Obróbka wczytanego pliku obrazu

Na podstawie wymiarów wczytanego obrazka i zmiennych *ilosc\_puzzli\_x* oraz *ilosc\_puzzli\_y* w klasie *Gra* wyznaczone są wymiary pojedynczego puzzla i przekazywane do zmiennych *szer\_puzzla* i *wys\_puzzla* w tej klasie. Następnie w pętli z obrazu w zmiennej *obrazek* wycinane są za pomocą metody *wxImage::GetSubImage* pojedyncze puzzle i kopiowane do dwuwymiarowej tablicy *puzzle*. Jednocześnie w dwuwymiarowej tablicy *wsp\_puzzli* rejestrowane są współrzędne każdego puzzla.

### 6.3. Tasowanie współrzędnych, lustrzane odbicia puzzli

W pętli losowane są współrzędne któregoś z puzzli a następnie współrzędne puzzla danego i wylosowanego są zamienione miejscami. Następnie w pętli na podstawie dwóch rzutów monetą ustalane są lustrzane odbicia każdego puzzla. W pierwszym rzucie: orzeł – lustrzane odbicie pionowe, reszka – brak odbicia pionowego. Wyniki drugiego rzutu decydują analogicznie o odbiciach poziomych. Wyniki losowania zapamiętywane są w dwuwymiarowej tablicy *obroty*.

### 6.4. Wyświetlanie puzzli

Puzzle wyświetlane są w głównym oknie po wystąpieniu zdarzenia związanego z odrysowaniem okna. W pierwszej kolejności po wystąpieniu tego zdarzenia sprawdzany jest stan rozgrywki na podstawie wartości zmiennej *gra\_w\_toku* w klasie *Gra*.

Jeżeli jej wartość wynosi 0, wówczas w głównym oknie wyświetlany jest tylko interfejs użytkownika.

Jeżeli wartość wskazuje 1, w oknie wyświetlane są puzzle lub wzór układanego obrazka. Puzzle wyświetlane są w pętli za pomocą kontekstu okna *wxBufferedPaintDC*. Każdy puzzle z dwuwymiarowej tablicy *puzzle* wyświetlany jest w miejscu ekranu wskazanym przez dwuwymiarową tablicę *wsp\_puzzli*. Dodatkowo tablice *obroty*, *rozjasniacz* oraz zmienne *bajery* i *aktualny* wskazują jakim transformacjom przed wyświetleniem ma zostać poddany każdy puzzle. Na końcu każdego przebiegu pętli w zależności od wartości zmiennych *aktualny*, *klikniecie* i *ktory\_przycisk* oraz wartości w tablicach *sasiedzi*, *przyciski* i *wsp\_przyciskow* na puzzlu pod wskaźnikiem myszy wyświetlane są odpowiednie przyciski.

W przypadku, gdy wartość zmiennej *gra\_w\_toku* wynosi 2, w oknie wyświetlany jest tylko wzór układanego obrazka.

### 6.5. Śledzenie ruchu wskaźnika myszy

Każda zmiana pozycji wskaźnika myszy w obrębie głównego okna wywołuje odpowiednie zdarzenie. Po wystąpieniu tego zdarzenia współrzędne wskaźnika porównywane są w pętli ze współrzędnymi puzzli przechowywanymi w tablicy *wsp\_puzzli*. Współrzędne puzzla pod wskaźnikiem myszy zostają zapisane w zmiennej *aktualny* w klasie *Gra*.

### 6.6. Wyznaczanie sąsiadów puzzla pod wskaźnikiem myszy

Po ustaleniu współrzędnych puzzla znajdującego się pod wskaźnikiem myszy następuje wyznaczenie współrzędnych puzzli z nim sąsiadujących. Współrzędne znalezionych sąsiadów zapamiętywane są w tablicy *sasiedzi*.

### 6.7. Śledzenie kliknięć

Kliknięcie, podobnie jak ruch wskaźnika myszki, wywołuje odpowiednie zdarzenie. Część okna, w której następuje rysowanie puzzli, reaguje na to zdarzenie tylko w wypadku rozpoczęcia rozgrywki, tj. gdy zmienna *gra\_w\_toku* przyjmuje wartość *1*. Następnie sprawdzana jest wartość zmiennej *klikniecie*. Na jej podstawie porównywane są współrzędne kliknięcia ze współrzędnymi przycisków wyświetlonych na puzzlu znajdującym się pod wskaźnikiem myszki. W tej części programu następuje zapamiętanie numeru przycisku podstawowego (czerwonego) w zmiennej *ktory\_przycisk* w klasie *Gra* oraz numeru przycisku operacji na puzzlach w zmiennej lokalnej. Na podstawie tych dwóch zmiennych wykonywana jest jednoznacznie określona operacja na puzzlach.

### 6.8. Wyznaczanie punktacji

Po rozpoczęciu gry w zmiennej *czas\_roz poczenia* zapamiętywany jest aktualny czas. W czasie gry w dwuwymiarowej tablicy *czas\_puzzli* zapisywany jest czas ostatniego prawidłowego ułożenia każdego puzzla. Po ułożeniu puzzli notowany jest aktualny czas w zmiennej *czas\_zakonczenia*. Różnica wartości zmiennych *czas\_roz poczenia* i *czas\_zakonczenia* wyznacza czas trwania gry. Liczba zdobytych przez gracza punktów wyznaczana jest na podstawie zależności:

$$(liczba\ puzzli) * sum_{i}^{liczba\ puzzli} (1800 / (czas\_roz poczenia - czas\_puzzli_i))$$

i przechowywana wraz z imieniem gracza, liczbą puzzli, nazwą pliku i czasem gry w tablicy *punktacja*.

### 6.9. Generator obrazów

Generator obrazów losuje współrzędne na krawędzi bitmapy o wymiarach 600x600px. Następnie składowe r, g i b barwy każdego piksela bitmapy ustalana jest jako  $\sin(k_i r + a_i)$ , gdzie r jest odległością danego piksela od punktu wylosowanego na początku,  $k_i$  i  $a_i$  są, odpowiednio, wylosowaną częstością i przesunięciem fazowym i-tej składowej barwy piksela. Następnie powyższa operacja z małymi modyfikacjami jest wykonywana jeszcze trzykrotnie.

### 6.10. Zapis danych do pliku

Gdy gracz zdecyduje się zamknąć program, następuje zapis informacji z tablicy *punktacja* do pliku *highscore.dat*. Każde pole z tablicy zostaje przetworzone

w taki sposób, aby jego długość wynosiła 20 znaków. Po przetworzeniu wszystkie pola zostają „sklejone” w jeden string i w takiej postaci zostają zapisane do pliku.

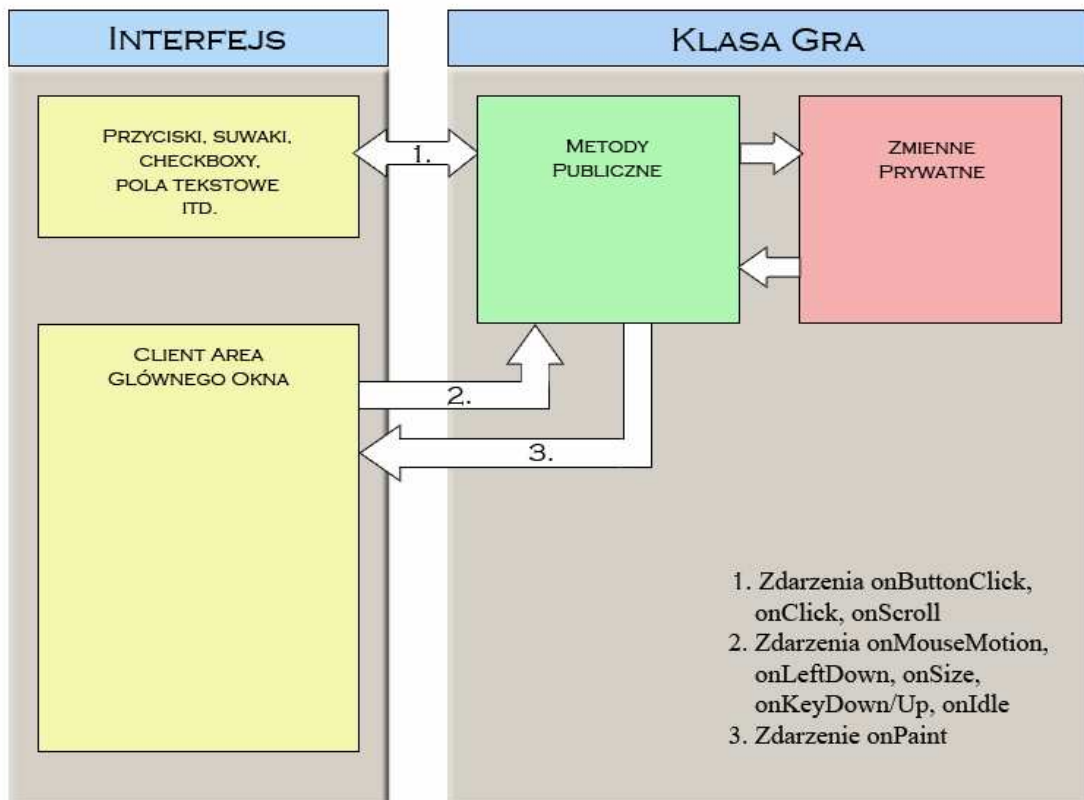
### 6.11 Odczyt danych z pliku

Odczytywanie danych z pliku *highscore.dat* polega na wykonaniu operacji odwrotnych do tych wykonywanych w czasie zapisu danych. W pierwszej kolejności cała zawartość pliku ładowana jest jako string. String ten następnie jest cięty na fragmenty o długości 20 znaków. Każdy fragment umieszczany jest w odpowiednim polu tablicy *punktacja*. Jeżeli plik *highscore.dat* nie istnieje, wówczas tablica *punktacja* wypełniana jest domyślnymi wartościami.

## 7. Kodowanie

### 7.1. Schemat blokowy

Poniżej przedstawiono główne bloki programu. Klasa *Gra* komunikuje się z główną aplikacją poprzez zdarzenia generowane przez użytkownika. Każde zdarzenie wygenerowane przez użytkownika i wewnętrzne zdarzenia klasy inkrementują zmienną *zmiana*. Zmienna ta jest dekrementowana po wystąpieniu zdarzenia *onPaint*. Jednocześnie wartość zmiennej *zmiana* jest śledzona w zdarzeniu *onIdle*, które kończy każdy ciąg innych zdarzeń. Gdy wartość zmiennej *zmiana* jest większa od zera, *onIdle* wywołuje zdarzenie *onPaint*. Tego typu zapętlenie pozwala uzyskać w grze efekt animacji i jednocześnie zatrzymuje działanie aplikacji, gdy gracz nie podejmuje żadnej akcji.



## 7.2. Opis klasy Gra: zmienne i funkcje

Poniżej przedstawiony jest opis klasy *Gra*. Wszystkie zmienne zostały zadeklarowane jako prywatne. Do operacji na nich służą metody publiczne.

– Zmienne:

### **wxPoint aktualny**

Przechowuje współrzędne x i y puzzla pod wskaźnikiem myszy. Jeżeli wskaźnik znajduje się poza obszarem, na którym wyświetlane są puzzle, współrzędne przyjmują wartości (-1,-1).

### **int bajery**

Przyjmuje wartość 0 lub 1. W funkcji *DrawPuzzle()* wyłącza lub włącza rozjaśnianie puzzli.

### **long czas\_roz poczenia**

Przechowuje czas rozpoczęcia gry wyrażony w sekundach.

### **long czas\_zakonczenia**

Przechowuje czas zakończenia gry wyrażony w sekundach.

### **long czas\_puzzli[50][50]**

Przechowuje czas ostatniego prawidłowego ułożenia każdego puzzla wyrażony w sekundach.

### **wxString filename**

Nazwa wczytanego pliku.

### **int gra\_w\_toku**

Przyjmuje wartość 0, 1 lub 2. 0 – program oczekuje na wczytanie pliku graficznego lub wygenerowanie obrazu, 1 – wczytano plik lub wygenerowano obraz, gracz jest w trakcie układania puzzli, 2 – gracz ułożył puzzle.

### **int ilosc\_puzzli\_x**

Ile kolumn puzzli, 2–40.

### **int ilosc\_puzzli\_y**

Ile wierszy puzzli, 2–40.

### **wxString imie\_gracza**

Przechowuje imię gracza.

### **int klikniecie**

Przyjmuje wartość 0 lub 1. W funkcji *DrawPuzzle()* określa czy na aktualnym puzzlu wyświetlać przyciski podstawowe (czerwone) czy przyciski operacji.

### **int ktory\_przycisk**

Przechowuje numer przycisku podstawowego (czerwonego), który został kliknięty. Przyjmuje wartości 0-8. 0, 2, 4, 6 to numery przycisków narożnych, odpowiednio, lewego górnego, prawego górnego, prawego dolnego, lewego dolnego. 1, 3, 5, 7 to numery przycisków, odpowiednio, górnego, prawego, dolnego i lewego. 8 to numer przycisku środkowego. W funkcji *DrawPuzzle()* określa, który zestaw przycisków operacji ma zostać wyświetlony na puzzlu pod wskaźnikiem myszy.

### **wxImage obrazek**

Przechowuje wygenerowany lub wczytany z pliku obraz.

### **wxPoint obroty[50][50]**

Zawiera informacje o lustrzanych odbiciach w osi pionowej (*obroty[i][j].x*) i w osi poziomej (*obroty[i][j].y*) każdego puzzla. Każda składowa może przyjąć wartość 0 (brak odbicia) lub 1 (odbicie).

### **wxFile plik**

Obiekt służący do operacji odczytu/zapisu pliku *highscore.dat*

### **wxImage puzzle[50][50]**

Zawiera fragmenty wczytanego/wygenerowanego obrazu

### **wxImage przyciski[4][6]**

Zawiera obrazy przycisków: *przyciski[0][ ]* – zawiera obrazy przycisków podstawowych (czerwonych), generowane w programie, *przyciski[1][ ]* – obrazy sześciu przycisków dla operacji wykonywanych na grupie czterech puzzli, *przyciski[2][ ]* – obrazy trzech przycisków dla operacji wykonywanych na dwóch puzzlach, *przyciski[3][ ]* - obrazy dwóch przycisków dla operacji na jednym puzzlu.

### **wxString punktacja[10][5]**

Zawiera 10 najlepszych wyników gry. *punktacja[ ][0]* - imię gracza, *punktacja[ ][1]* – liczba puzzli, *punktacja[ ][2]* – nazwa pliku, *punktacja[ ][3]* – czas gry, *punktacja[ ][5]* – liczba zdobytych punktów.

### **int puzzle\_bw**

Przyjmuje wartość 0 lub 1. W funkcji *DrawPuzzle()* określa czy puzzle zostaną wyświetlone jako czarno-białe.

### **int rozjasniacz[50][50]**

Każdy element tej tablicy przyjmuje wartość 0-80. W funkcji *DrawPuzzle()* wszystkie składowe barwy każdego piksela puzzla *ij* zwiększane są o wartość *rozjasniacz[i][j]*. Jednocześnie wartość ta jest inkrementowana, jeżeli jest to puzzel pod wskaźnikiem myszy, dla pozostałych jest dekrementowana.

### **wxPoint sasiedzi[8]**

Zawiera współrzędne ośmiu sąsiadów puzzla pod wskaźnikiem myszy. Jeżeli którego z sąsiadów nie ma, jego współrzędne zostają oznaczone przez (-1,-1).

Sąsiedzi numerowani są od 0-7 zgodnie z ruchem wskazówek zegara, zaczynając od górnego lewego.

**int szer\_puzzla**

Szerokość puzzla w px.

**int szybkość\_sciemniania**

Wartość o jaką dekrementowany jest *rozjasniacz[i][j]*. Przyjmuje wartości 1-8

**int szybkość\_rozjasniania**

Wartość o jaką inkrementowany jest *rozjasniacz[i][j]*. Przyjmuje wartości 1-8

**int wciśnięto\_tab**

Przyjmuje wartość 1 gdy zostanie wciśnięty klawisz [Tab]. 0 przy zwolnieniu klawisza.

**wxPoint wsp\_przyciskow[4][9]**

Zawiera współrzędne przycisków. *wsp\_przyciskow[0][]* – współrzędne dziewięciu przycisków podstawowych (czerwonych), *wsp\_przyciskow[1][]* – współrzędne sześciu przycisków operacji na grupie czterech puzzli, *wsp\_przyciskow[2][]* – współrzędne trzech przycisków operacji na dwóch puzzlach, *wsp\_przyciskow[3][]* – współrzędne dwóch przycisków operacji na jednym puzzlu.

**wxPoint wsp\_puzzli[50][50]**

Zawiera współrzędne każdego z puzzli.

**int wylosowano**

Przyjmuje wartość 0 lub 1. Ustawiany na 1 po przetasowaniu współrzędnych puzzli

**int wymiar\_przycisku**

Zawiera wymiar przycisku podstawowego (czerwonego) – 15. Przycisk podstawowy wyświetlany jest jako kwadrat o boku 15px.

**int wys\_puzzla**

Wysokość puzzla w px.

**wxPoint w\_oknie**

Zawiera współrzędne początku pomocniczego układu współrzędnych. Zapewnia wyświetlanie puzzli w środku okna po zmianie jego rozmiarów.

**int zapisano\_punkty**

Przyjmuje wartość 0 lub 1. Zapobiega wyświetlaniu w nieskończoność okna, w którym gracz wpisuje imię.

**int zmiana**

„Stos” zdarzeń gry. Przyjmuje wartości  $\geq 0$ . Każde zdarzenie wygenerowane przez użytkownika lub zwiększenie/zmniejszenie rozjaśnienia puzzli inkrementuje stos. Każde zdarzenie odrysowania okna dekrementuje stos.

– Metody:

**Gra(void)**

Konstruktor. Ustawia domyślne parametry gry.

**void SetAktualny(int a, int b)**

**wxPoint GetAktualny()**

Set...() ustawia współrzędne (x,y) puzzla pod wskaźnikiem myszy w zmiennej *aktualny*. Następnie wywołuje funkcje *SetWspolrzednePrzyciskow()* i *SetSasiedzi()*. Get...() zwraca wartość zmiennej *aktualny*.

**void SetBajery(int stan)**

**int GetBajery()**

Ustawia/zwraca zmienną *bajery*.

**void SetCzasRozpoczecia()**

**long GetCzasRozpoczecia()**

Set...() z rozpoczęciem gry zapisuje aktualny czas (*wxGetLocalTime*) w zmiennej *czas\_rozpoczecia*. Get...() zwraca jej wartość.

**void SetCzasZakonczenia()**

**long GetCzasZakonczenia()**

Wykonuje w/w operacje na zmiennej *czas\_zakonczenia* w momencie ułożenia puzzli.

**void SetCzasPuzzli(int a, int b)**

**long GetCzasPuzzli(int a, int b)**

**void ResetCzasPuzzli(int a, int b)**

Set...() ustawia czas ułożenia (*wxGetLocalTime()* – *czas\_rozpoczecia*) puzzla nr *ab* w tablicy *czas\_puzzli[a][b]*, gdy puzzel znajdzie się na właściwym miejscu układanki. Get...() pozwala sprawdzić czas ułożenia puzzla *ab*. Reset...() zapisuje w tablicy *czas\_puzzli[a][b]* wartość 0, gdy puzzel nie znajduje się na właściwym miejscu układanki.

**void SetFilename(wxString f)**

**wxString GetFilename()**

Set...() ustawia zmienną *filename*. Get...() zwraca zmienną *filename* lub string „Generator”.

**void SetGraWToku(int stan)**

**int GetGraWToku()**

Ustawia/zwraca wartość zmiennej *gra\_w\_toku*

**void SetIloscPuzzliX(int ilosc)**  
**int GetIloscPuzzliX()**  
Ustawia/zwraca zmienną *ilosc\_puzzli\_x*.

**void SetIloscPuzzliY(int ilosc)**  
**int GetIloscPuzzliY()**  
Ustawia/zwraca zmienną *ilosc\_puzzli\_y*.

**void SetImieGracza(wxString im)**  
**wxString GetImieGracza()**  
Ustawia/zwraca zmienną *imie\_gracza*.

**void SetKlikniecie(int w)**  
**int GetKlikniecie()**  
Ustawia/zwraca zmienną *klikniecie*.

**void SetKtory(int w)**  
**int GetKtory()**  
Ustawia/zwraca zmienną *ktory\_przycisk*

**void SetLosoweWspolrzedne()**  
Tasuje współrzędne puzzli w tablicy *wsp\_puzzli*.

**void SetLosoweObroty()**  
Dwa rzuty monetą. Pierwszy decyduje o lustrzanym odbiciu puzzla względem osi pionowej. Drugi o obiciu względem osi poziomej.

**void SetObrazek(wxImage obr)**  
**wxImage GetObrazek()**  
Set...() kopiuje obraz do zmiennej *obrazek*. Następnie wyznacza wymiary pojedynczego puzzla (*SetSzerokosc/WysokoscPuzzla()*), sprawdza czy przyciski podstawowe zmieszczą się wewnątrz puzzla i ewentualnie dokonuje zmniejszenia ich rozmiaru, generuje przyciski podstawowe i wczytuje przyciski operacji z dysku (*SetPrzyciski()*), dzieli obrazek na puzzle (*SetPuzzle()*) i notuje czas rozpoczęcia gry (*SetCzasRozpoczecia()*). Get...() zwraca obraz przechowywany w zmiennej *obrazek*.

**void SetObrotX(int x, int y )**  
**void SetObrotY(int x, int y)**  
Zapisują wartość 0 lub 1 w tablicy *obroty[x][y].x* i *obroty[x][y].y*

**void OpenPlik()**  
**void SavePlik()**  
Open...() wczytuje informacje z pliku *highscore.dat* do tablicy *punktacja*. Save...() zapisuje informacje z tablicy *punktacja* do pliku.

**void SetPuzzle()**

**wxImage GetPuzzle(int x, int y)**

Set...() wycina z obrazu w zmiennej *obrazek* pojedyncze puzzle i umieszcza je w tablicy *puzzle*.

Get...() zwraca obraz puzzla z tablicy *puzzle[x][y]*.

**void SetPrzyciski()**

Generuje obraz przycisku podstawowego w postaci czerwonego kwadratu i ładuje obrazy przycisków operacji z katalogu *gfx*. Obrazy zostają skopiowane do tablicy *przyciski*.

**void SetPunktacja()**

**wxString GetPunktacja(int a, int b)**

Set...() wyznacza liczbę punktów po zakończeniu gry i zapisuje wraz z imieniem gracza, liczbą puzzle, nazwą pliku i czasem w tablicy *punktacja*

Get...() zwraca string z tablicy *punktacja[a][b]*.

**void SetPuzzleBW(int stan)**

**int GetPuzzleBw()**

Ustawia/zwraca zmienną *puzzle\_bw*.

**void SetSasiedzi()**

**wxPoint GetSasiad(int a)**

Set...() przeszukuje tablicę *wsp\_puzzli* i wyznacza sąsiadów puzzla pod wskaźnikiem myszy (*aktualny*). Wynik zapisuje w tablicy *sasiedzi*.

Get...() zwraca współrzędne sąsiada nr a.

**void SetSzybkoscSciemniania(int szybkosc)**

**void SetSzybkoscRozjasniania(int sz)**

**int GetSzybkoscRozjasniania()**

**int GetSzybkoscSciemniania()**

Ustawiają/zwracają zmienne *szybkosc\_sciemniania* i *szybkosc\_rozjasniania*.

**void SetWcisnietoTab(int stan)**

**int GetWcisnietoTab()**

Ustawia/zwraca zmienna *wcisnieto\_tab*.

**void SetWspolrzednePrzyciskow()**

**wxPoint GetWspolrzednePrzycisku(int a, int b)**

Set...() wyznacza współrzędne przycisków podstawowych i przycisków operacji na podstawie współrzędnych puzzla pod wskaźnikiem myszy (*aktualny*). Współrzędne każdego przycisku przechowywane są w tablicy *wsp\_przyciskow*.

Get...() zwraca współrzędne przycisku w tablicy *wsp\_przyciskow[a][b]*.

**void SetWspolrzednePuzzli()**

**wxPoint GetWspolrzednePuzzla(int ax, int by)**

Set...() wyznacza współrzędne każdego z puzzli w układzie współrzędnych, którego początek definiuje zmienna *w\_oknie*. Dokonuje tasowania współrzędnych puzzli i losowania obrotów, jeśli jeszcze się nie odbyły.

Get...() zwraca współrzędne puzzla nr *axby*.

**void SetWylosowano(int stan)**

Ustawia zmienną *wylosowano*.

**void SetWOknie(wxPoint wsp)**

**wxPoint GetWOknie()**

Ustawia/zwraca zmienną *w\_oknie*.

**void SetZapisanoPunkty(int stan)**

**int GetZapisanoPunkty()**

Ustawia/zwraca zmienną *zapisano\_punkty*.

**void SetZmiana(int stan)**

**int GetZmiana()**

Ustawia/zwraca zmienną *zmiana*.

**void DrawImage(wxBufferedPaintDC \*my\_dc, int x=0, int y=0)**

Wyświetla obraz w zmiennej *obrazek* w głównym oknie za pomocą *wxBufferedPaintDC*.

**void DrawPuzzle(wxBufferedPaintDC \*my\_dc, int x=0, int y=0)**

Wyświetla wszystkie puzzle z tablicy *puzzle*. Przed wyświetleniem wykonywane są wszystkie transformacje związane z lustrzanymi odbiciami i parametrami gry.

**void GenerujObrazek()**

Wykonuje operacje opisane w rozdziale 6.9.

**void OperujNaPuzzlach(int p1, int p2)**

Wykonuje na puzzlach operację jednoznacznie określoną przez numer przycisku podstawowego (*p1*) i numer przycisku operacji (*p2*).

**void ResetGame()**

Ustawia domyślne parametry gry, gdy użytkownik zdecyduje się przerwać grę.

**int SprawdzStanGry()**

Sprawdza czy wszystkie puzzle mają czasy różne od zera w tablicy *czas\_puzzli*. Jeżeli tak, to znaczy, że obraz został poprawnie ułożony i gra zostanie zakończona przez zwrócenie wartości 1.

## **8. Testowanie**

Testowanie każdej funkcji odbywało się po jej implementacji. Jeżeli efekt działania funkcji był zgodny z oczekiwaniami, pisana była kolejna metoda. W przeciwnym razie błąd zostawał od razu naprawiany. Niektóre metody wymagały napisania funkcji testujących, co pozwalało szybko odnaleźć błąd.

Taka metoda testowania nie jest wygodna, ponieważ trudno przewidywać, jak będą wyglądały dalsze fragmenty programu, oraz jak zostaną rozwiązane problemy pojawiające się w trakcie implementacji. Przez to wiele funkcji po napisaniu zostało poprawionych i zmienionych.

Dzięki temu że kod programu został podzielony na małe moduły uniknięto sporo problemów z późniejszymi zmianami w metodach.

## **9. Wdrożenie, raport i wnioski**

W programie zawarte są wszystkie wstępnie przyjęte wymagania. Zrealizowano również dodatkowe pomysły, które pojawiły się w trakcie pisania programu.

Dodatkowo w przyszłości można:

- dodać możliwość zapisania i wczytania gry
- umożliwić graczowi cofanie kilku ostatnich ruchów
- uporządkować kod programu, refaktoryzować kod