

# Odtwarzanie dokumentu zniszczonego przez niszczarkę

## Spis treści

<a href="#">Opis aplikacji</a>	2
<a href="#">Funkcjonalność aplikacji</a>	2
<a href="#">Podział zadań</a>	2
<a href="#">Harmonogram realizacji projektu</a>	4
<a href="#">Metodologia projektowania i implementowania</a>	5
<a href="#">Rysunek poglądowy aplikacji</a>	6
<a href="#">Specyfikacja ważniejszych algorytmów</a>	7
<a href="#">Wykorzystane narzędzia i technologie</a>	8
<a href="#">Omówienie implementacji</a>	9
<a href="#">Instrukcja użytkowania aplikacji</a>	17
<a href="#">Testy</a>	18
<a href="#">Wyniki testów:</a>	18
<a href="#">Średnie wyniki:</a>	19
<a href="#">Rozkłady wyników:</a>	21
<a href="#">Testowanie na przykładowych obrazach</a>	23
<a href="#">Lena</a>	23
<a href="#">Obraz wejściowy</a>	23
<a href="#">Obraz wyjściowy</a>	24
<a href="#">Color shapes</a>	24

<a href="#">Obraz wejściowy</a>	24
<a href="#">Obraz wyjściowy</a>	25
<a href="#">Real text</a>	28
<a href="#">Obraz wejściowy</a>	28
<a href="#">Obraz wyjściowy</a>	29
<a href="#">Podsumowanie</a>	30
<a href="#">Repozytorium</a>	30
<a href="#">Spis literatury</a>	30

## Opis aplikacji

Aplikacja ma za zadanie odtwarzać dokumenty, pocięte przez niszczarkę. Program wykrywa poszczególne odcięte fragmenty dokumentu na skanie, a następnie próbuje złożyć je w całość. Pozwala wyeksportować rezultat swojej pracy do nowego pliku graficznego oraz wyświetlić go na ekranie.

## Funkcjonalność aplikacji

- Wczytywanie pociętego dokumentu z pliku graficznego z formacie rastrowym (BMP).
- Analiza obrazu wejściowego w celu wyodrębnienia poszczególnych fragmentów pociętego dokumentu.
- Odrzucenie krawędzi, które są białe z powodu nierównego oddarcia/ucięcia.
- Poszukiwanie układu fragmentów, w którym są do siebie jak najlepiej dopasowane pod względem kształtu i kolorów na krawędziach.
- Zapis złożonego przez aplikację dokumentu do pliku graficznego (bmp).
- Opcja pomocy, służąca do objaśniania odpowiednich funkcji programu

## Podział zadań

- 1 - Wczytywanie/zapis bitmap.
- 1 - Wyodrębnianie fragmentów dokumentu.
- 3 - Wyszukiwanie uszkodzonych krawędzi fragmentów.
- 2 - Wyszukiwanie układu fragmentów które są do siebie dopasowane

- 3 - Pomoc do programu.

Legenda:<sup>1</sup>

- 1 - Krzysztof Jerzyński
- 2 - Piotr Siupa

---

<sup>1</sup> Numery działają jak flagi i mogą być sumowane bitowo.

## Harmonogram realizacji projektu

02.03-23.03.2016	Zapoznanie się z tematem zadania. Podział zadań.
24.03-06.04.2016	Konfiguracja środowiska programistycznego do obsługi OpenCV. Pisanie prototypów funkcji.
07.04-20.04.2016	Pisanie funkcji odpowiedzialnych za wczytywanie obrazu wejściowego, zapis obrazu wynikowego oraz wyszukiwanie pociętych kawałków dokumentu/obrazu.
21.04-04.05.2016	Filtrowanie elementów obrazka, wyodrębnianie największych kawałków.
05.05-18.05.2016	Wykrywanie obróconych elementów i ustawianie ich pod kątem prostym. Wycinanie wyodrębnionych kawałków i zapis do pliku jako bitmapy.
19.05-01.06.2016	Implementacja algorytmu ewolucyjnego. Wstępne układanie elementów na podstawie cech. Wyświetlanie ułożonych fragmentów w oknie wynikowym.
02.06-15.06.2016	Układanie elementów na podstawie określonych cech. Testowanie oprogramowania. Opracowanie testów. Praca nad dokumentacją.

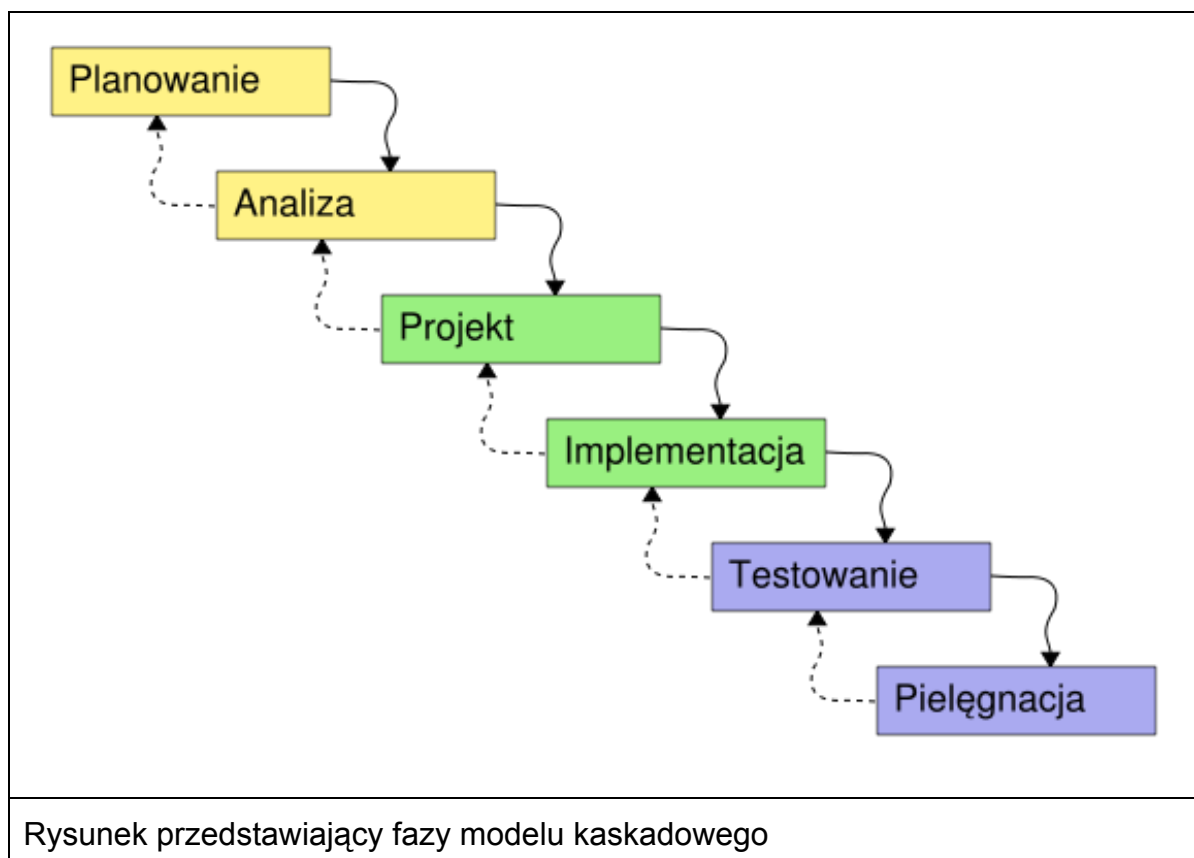
# Metodologia projektowania i implementowania

## Wybrany model kaskadowy projektowania i implementowania aplikacji.

Wybór tejże metodologii wyniknął z potrzeb projektowych, wielkości projektu oraz tego, że aby rozpocząć implementację projektu, należy w pierwszej kolejności zadbać o takie aspekty, jak planowanie i analiza zagadnienia wraz z zespołem, a dopiero potem przystąpienie do właściwej pracy.

### Iteracyjny model kaskadowy składa się z następujących kroków:

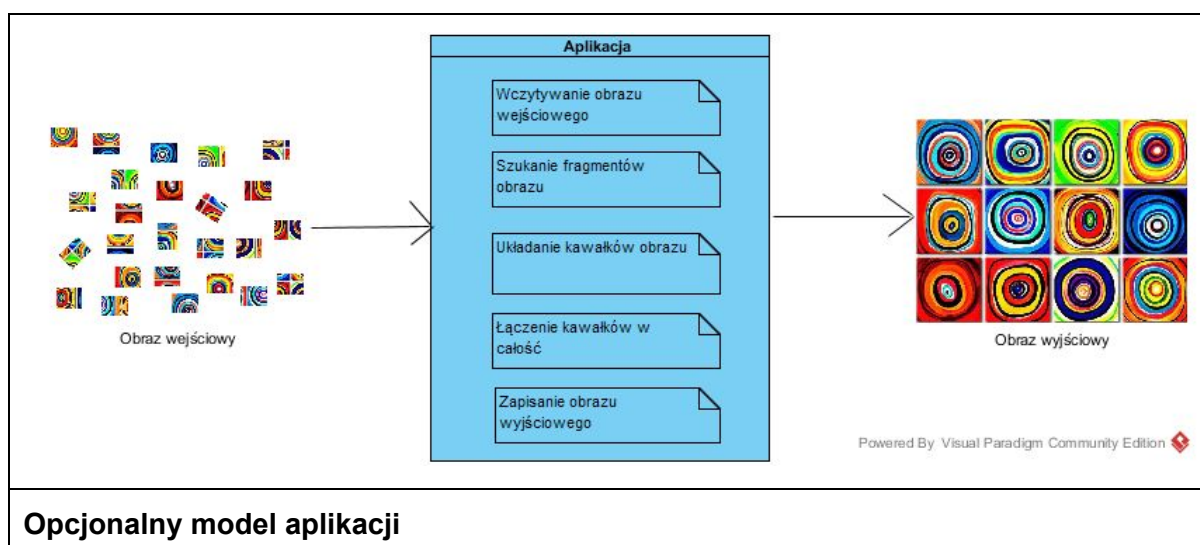
1. Planowanie systemu (w tym specyfikacja wymagań).
2. Analiza systemu (w tym analiza wymagań i studium wykonalności).
3. Projekt systemu (poszczególnych struktur itp.).
4. Implementacja (wytworzenie kodu).
5. Testowanie.
6. Pielęgnacja kodu.



Jeśli któraś z faz zaowocuje niesatysfakcjonującym produktem, to cofamy się wykonując kolejne iteracje aż do momentu kiedy otrzymamy satysfakcjonujący produkt na końcu schodków.

Mimo tego, iż jest rzadko używanym modelem, to spełnia większość naszych założeń ze względu na swoją prostotę i nieskomplikowaną złożoność projektu.

## Rysunek poglądowy aplikacji



# Specyfikacja ważniejszych algorytmów

## ➤ Algorytm wyodrębniania pociętych kawałków z obrazu wejściowego

**Wejście:** Obraz wejściowy, współrzędne kawałków obrazu

**Wyjście:** Wyodrębnione fragmenty obrazu

**Metoda:**

1. Utworzenie listy do przechowywania kawałków.
2. Utworzenie pętli for iterującej po elementach struktury zawierającej współrzędne kawałków obrazu:
  - a. wyodrębnienie minimalnego obszaru zajmowanego przez każdy pocięty obrazek z obrazu wejściowego z wykorzystaniem współrzędnych
  - b. pobranie wartości kąta pod jakim jest ułożony kawałek
  - c. wyznaczenie różnicy pomiędzy kątem 90 stopni, a kątem pobranym
    - i. obrót fragmentu o kąt wynikający z różnicy kątów
  - d. wycięcie obróconego kawałka
  - e. dodanie wyciętego elementu do listy wynikowej
3. Zwrócenie listy wyodrębnionych fragmentów obrazu.

## ➤ Algorytm ewolucyjny

**Wejście:** Wyodrębnione fragmenty obrazu, obrócone w taki sposób, żeby ich krawędzie były jednocześnie krawędziami macierzy, w których są zapisane

**Wyjście:** Fragmenty obrazu z określonymi pozycjami i obrotami.

**Metoda:**

1. Utworzenie wejściowej generacji osobników poprzez losowanie pozycji i obrotów fragmentów obrazu
2. Wielokrotne wykonanie kroku ewolucji (w pętli):
  - a. Utworzenie kolejnej generacji
  - b. Przeprowadzenie selekcji
  - c. Sprawdzenie, czy został spełniony jeden z warunków zakończenia
    - i. limit czasu
    - ii. limit kroków, w których nie udało się uzyskać poprawy
3. Wybór najlepszego osobnika z końcowej generacji

## ➤ Algorytm łączenia fragmentów obrazu w całość

**Wejście:** Uporządkowane fragmenty obrazu z ustalonymi pozycjami

**Wyjście:** Obraz wyjściowy (zapisywany do pliku graficznego oraz wyświetlany w oknie wynikowym)

**Metoda:**

1. Obróć fragmenty obrazu o zadany kąt 90, 180 lub 270 stopni.
2. W pętli (0...ilość kawałków): Wyszukaj min wartość współrzędnej X pozycji fragmentu obrazka.
3. W pętli (0...ilość kawałków): Wyszukaj min wartość współrzędnej Y pozycji fragmentu obrazka.
4. W pętli (0...ilość kawałków): Wykonaj odejmowanie każdej współrzędnej X z minimalną współrzędną X lokalizacji kawałka na osi.
5. W pętli (0...ilość kawałków): Wykonaj odejmowanie każdej współrzędnej Y z minimalną współrzędną Y lokalizacji kawałka na osi.
6. Utwórz zmienne do przechowywania wymiarów prostokątów, w których będą wyświetlone kawałki obrazu
7. W pętli (0...ilość kawałków):
  - a. pobierz wartość x
  - b. pobierz wartość y
  - c. pobierz wartość szerokości obrazka
  - d. pobierz wartość wysokości obrazka
  - e. zsumuj wartości szerokości i wysokości wszystkich kawałków i przypisz je do nowej zmiennej.
8. Utwórz macierz wynikową, której wysokość będzie sumą wysokości wszystkich kawałków obrazu, a szerokość sumą szerokości wszystkich kawałków obrazu.
9. W pętli (0...ilość kawałków):
  - a. utwórz tymczasową macierz do przechowywania kawałków obrazu i przypisz go do niej
  - b. dokonaj rozszerzenia obrazka do wymiarów przechowującego go prostokąta
  - c. skopiuj przetransformowany element do macierzy wynikowej
10. Zwróć macierz wynikową

## Wykorzystane narzędzia i technologie

- Biblioteka do analizy obrazu: **OpenCV**
- Język programowania: **C++**
- Środowisko programowania: **Microsoft Visual Studio 2015**
- Interpreter poleceń **bash**



# Omówienie implementacji

## ➤ Algorytm wyodrębniania pociętych kawałków z obrazu wejściowego

```
// funkcja przyjmuje w parametrach oryginalny obraz wejściowy image typu cv::Mat oraz
// strukturę rectangles_t w której przechowywane są współrzędne ułożenia kawałków na osi X,Y

pieces_t cropImages(cv::Mat image, rectangles_t squares)
{
    pieces_t listCropImage;
    listCropImage.reserve(squares.size());

    const std::string name = "./pieces/piece";
    const std::string extension = ".png";
    for (size_t i = 0; i < squares.size(); i++)
    {
        // utworzenie obiektu "obracanego prostokąta" dzięki któremu będzie można "równno"
        // wyciąć kawałek , uprzednio obracając go tak , żeby jego kąt wynosił 90 stopni
        cv::RotatedRect rect = cv::minAreaRect(cv::Mat(squares[i]));
        // utworzenie macierzy
        cv::Mat M, rotated, cropped;
        // pobranie kąta kawałka
        float angle = rect.angle;
        cv::Size rect_size = rect.size;
        if (rect.angle < -45.) {
            angle += 90.0;
            std::swap(rect_size.width, rect_size.height);
        }
        // pobranie obracanej macierzy
        M = getRotationMatrix2D(rect.center, angle, 1.0);
        // wykonanie transformacji
        warpAffine(image, rotated, M, image.size(), cv::INTER_CUBIC);
        // wycięcie kawałka obrazu
        getRectSubPix(rotated, rect_size, rect.center, cropped);
        /* // opcja zapisu do pliku o wskazanym folderze ( tymczasowo zakomentowana)
        std::string filename = name + std::to_string(i) + extension;
        cv::imwrite(filename, cropped);
        */
        // dodanie wyciętego obrazka do listy wynikowej
        listCropImage.push_back(cropped);
    }
    //showImage(subimage);
    //cout << "#" << i << " rectangle x:" << rectangle.x << " y:" << rectangle.y <<
    " " << rectangle.width << "x" << rectangle.height << endl;

    return listCropImage; // zwrócenie listy wynikowej
}
}
```

**Algorytm wyodrębniania pociętych kawałków z obrazu wejściowego - kod C++**

## ➤ Algorytm ewolucyjny

### ○ Funkcja, tworząca nową generację osobników

Generacja jest tworzona w większości poprzez krzyżowanie osobników z poprzedniej populacji. Niektórzy osobnicy są jednak tworzeni wyłącznie poprzez

mutację, a jeszcze mniejsza ich ilość jest całkowicie losowa. Są również zachowani wszyscy osobnicy z poprzedniej generacji.

```
void Generation::createNewIndividuals(const unsigned mutationCount, pieces_t &pieces)
{
    const size_t initialSize = individuals.size();
    for (size_t i = 0; i != CROSSES_NUMBER; ++i)
    {
        const size_t x = rand() % initialSize;
        size_t y = rand() % (initialSize - 1);
        if (y >= x)
            ++y;
        threadPool.startNew([this, mutationCount, &pieces, x, y]
        {
            Individual individualCopy1;
            Individual individualCopy2;
            {
                std::lock_guard<std::mutex> lock(mutex);
                individualCopy1 = *individuals[x];
                individualCopy2 = *individuals[y];
            }
            individualCopy1.mutate(mutationCount);
            individualCopy2.mutate(mutationCount);
            Individual newIndividual(individualCopy1, individualCopy2);
            individual_ptr_t newIndividualPointer(new
Individual(std::move(newIndividual)));
            {
                std::lock_guard<std::mutex> lock(mutex);
                individuals.push_back(std::move(newIndividualPointer));
            }
        });
    }
    for (size_t i = 0; i != MUTATION_NUMBER; ++i)
    {
        const size_t x = rand() % initialSize;
        threadPool.startNew([this, mutationCount, x]
        {
            Individual newIndividual;
            {
                std::lock_guard<std::mutex> lock(mutex);
                newIndividual = *individuals[x];
            }
            newIndividual.mutate(mutationCount);
            individual_ptr_t newIndividualPointer(new
Individual(std::move(newIndividual)));
            {
                std::lock_guard<std::mutex> lock(mutex);
                individuals.push_back(std::move(newIndividualPointer));
            }
        });
    }
    for (size_t i = 0; i != RANDOM_NUMBER; ++i)
    {
        threadPool.startNew([this, &pieces]
        {
            individual_ptr_t newIndividualPointer(new Individual(pieces));
            {
                std::lock_guard<std::mutex> lock(mutex);
                individuals.push_back(std::move(newIndividualPointer));
            }
        });
    }
    threadPool.joinAll();
}
```

```
}
```

## Funkcja, tworząca nową generację osobników

### ○ Funkcja przeprowadzająca selekcję

Przez selekcję zawsze przechodzi 10 najlepszych osobników. Każdy z pozostałych ma 20% szans na przeżycie.

```
void Generation::doSelection()
{
    //Compute rating of every individual in multiple threads (the operation is very
    time-consuming)
    for (const individual_ptr_t &individual : individuals)
        threadPool.startNew([&individual]{ individual->getRating(); });
    threadPool.joinAll();
    //Sort individuals by ratings (ascending)
    std::sort(individuals.begin(), individuals.end(), [](const individual_ptr_t &i1,
    const individual_ptr_t &i2) -> bool { return i1->getRating() < i2->getRating(); });
    //Delete (randomly) most of the individuals which aren't in best 10
    for (size_t i = 0; i != individuals.size() - SELECTION_SIZE;)
        if (rand() % 100 < SELECTION_LEAVE_PERCENT)
            ++i;
        else
            individuals.erase(individuals.begin() + i);
}
```

## Funkcja przeprowadzająca selekcję

### ➤ Klasa puli wątków

Jest to prosta implementacja puli wątków, przeznaczona do wykorzystania przez jeden wątek nadrzędny na raz. Służy ona temu, aby nie trzeba było w kółko tworzyć i usuwać krótko-żyjących wątków, ponieważ są to kosztowne operacje.

```
class ThreadPool
{
public:
    typedef std::function<void()> task_t;

    ThreadPool() : ThreadPool(std::thread::hardware_concurrency()) {}
    ThreadPool(const unsigned threadCount);
    ~ThreadPool();

    void startNew(const task_t &task);
    void joinAll() { std::unique_lock<std::mutex> lock(mutex); cv.wait(lock, [this]{
    return workingThreads == 0; }); }

    unsigned size() const { return threads.size(); }

private:
    void threadFunction();

    std::vector<std::thread> threads;
    std::mutex mutex;
    std::condition_variable cv;
    task_t newTask;
```

```

    bool closingThreads;
    unsigned workingThreads;

public:
    ThreadPool(const ThreadPool &) = delete;
    ThreadPool& operator=(const ThreadPool &) = delete;
};

ThreadPool::ThreadPool(const unsigned threadCount) :
    threads(),
    mutex(),
    cv(),
    newTask(),
    closingThreads(false),
    workingThreads(0)
{
    threads.reserve(threadCount);
    for (unsigned i = 0; i != threadCount; ++i)
        threads.push_back(std::thread([this]{ threadFunction(); }));
}

ThreadPool::~~ThreadPool()
{
    {
        std::unique_lock<std::mutex> lock(mutex);
        newTask = task_t([]{});
        closingThreads = true;
    }
    cv.notify_all();
    for (std::thread &thread : threads)
        thread.join();
}

void ThreadPool::startNew(const task_t &task)
{
    std::unique_lock<std::mutex> lock(mutex);
    cv.wait(lock, [this]{ return !bool(newTask); });
    newTask = task;
    lock.unlock();
    cv.notify_all();
    lock.lock();
    cv.wait(lock, [this]{ return !bool(newTask); });
}

void ThreadPool::threadFunction()
{
    for (;;)
    {
        task_t task;
        {
            std::unique_lock<std::mutex> lock(mutex);
            cv.wait(lock, [this]{ return bool(newTask); });
            if (closingThreads)
                break;
            ++workingThreads;
            task = newTask;
            newTask = task_t();
        }
        cv.notify_all();
        task();
        {
            std::lock_guard<std::mutex> lock(mutex);
            --workingThreads;
        }
    }
}

```

```
        cv.notify_all();
    }
}
```

## Klasa "ThreadPool"

### ➤ Algorytm łączenia fragmentów obrazu w całość

```
//funkcja pomocnicza dla mergePieces, powoduje obrócenie kawałka obrazu o zadany kąt
90,180 lub 270 stopni

void rotate_90n(cv::Mat &piece, Rotation angle)
{
    switch (angle)
    {
        case Rotation::R0:
            break;
        case Rotation::R90:
        {
            cv::transpose(piece, piece);
            cv::flip(piece, piece, 0);
            break;
        }
        case Rotation::R180:
        {
            cv::flip(piece, piece, -1);
            break;
        }
        case Rotation::R270:
        {
            cv::transpose(piece, piece);
            cv::flip(piece, piece, 1);
            break;
        }
    }
}

// wywołanie wcześniej wspomnianej funkcji dla każdego kawałka

void rotateArrangedPieces(arrangedPieces_t &arrangedPieces)
{
    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        rotate_90n(arrangedPieces[i].piece, arrangedPieces[i].rotation);
    }
}

// wyszukaj minimalną wartość współrzędnej X , iterując po wszystkich kawałkach obrazu

int searchMinX(arrangedPieces_t &arrangedPieces)
{
    int minX = std::numeric_limits<int>::max();

    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        if (arrangedPieces[i].position.first < minX)
            minX = arrangedPieces[i].position.first;
    }
}
```

```

    }

    return minX;
}

// wyszukaj minimalną wartość współrzędnej Y , iterując po wszystkich kawałkach obrazu
int searchMinY(arrangedPieces_t &arrangedPieces)
{
    int minY = std::numeric_limits<int>::max();

    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        if (arrangedPieces[i].position.second < minY)
            minY = arrangedPieces[i].position.second;
    }

    return minY;
}

// szuka maksymalnej liczby kolumn obrazu wynikowego potrzebnych w późniejszym czasie do
// odpowiedniego wyświetlania fragmentów obrazka
int CountCols(arrangedPieces_t &arrangedPieces)
{
    int maxX = std::numeric_limits<int>::min();

    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        if (arrangedPieces[i].position.first > maxX)
            maxX = arrangedPieces[i].position.first;
    }

    return maxX + 1;
}

// szuka maksymalnej liczby wierszy obrazu wynikowego potrzebnych w późniejszym czasie
// do odpowiedniego wyświetlania fragmentów obrazka
int CountRows(arrangedPieces_t &arrangedPieces)
{
    int maxY = std::numeric_limits<int>::min();

    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        if (arrangedPieces[i].position.second > maxY)
            maxY = arrangedPieces[i].position.second;
    }

    return maxY + 1;
}

// funkcja w której dokonuje się odejmowania wartości współrzędnych X i Y każdego kawałka
// od minimalnej wartości współrzędnych X i Y, żeby punkt odniesienia względem osi X i Y był
// (0,0)
void subtractionMinXAndMinY(arrangedPieces_t &arrangedPieces, int minX, int minY)
{
    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        arrangedPieces[i].position.first -= minX;
        arrangedPieces[i].position.second -= minY;
    }
}

```

```

}

// funkcja do łączenia fragmentów obrazka w całość przyjmująca w parametrze referencję do
// pociętych kawałków

cv::Mat mergePieces(arrangedPieces_t &arrangedPieces)
{
    std::cout << "Merging pieces into a single output image..." << std::endl;

    // obracanie obrazka o zadany kąt
    rotateArrangedPieces(arrangedPieces);

    // szukanie min X i min Y
    int minX = searchMinX(arrangedPieces);
    int minY = searchMinY(arrangedPieces);

    // odejmowanie wartości współrzędnych arrangedPieces z minX oraz minY
    subtractionMinXAndMinY(arrangedPieces, minX, minY);

    // obliczyć wartości każdej kolumny i każdego wiersza

    std::vector<int> sizeEveryCols;
    std::vector<int> sizeEveryRows;

    std::vector<int> widthEveryCols;
    std::vector<int> heightEveryRows;

    // maksymalne wartości kolumn oraz wierszy

    const int ColsCount = CountCols(arrangedPieces);
    const int RowsCount = CountRows(arrangedPieces);

    sizeEveryCols.resize(ColsCount, 0);
    sizeEveryRows.resize(RowsCount, 0);

    // obliczanie rozmiaru każdego elementu ( szerokość , wysokość ) przechowującego
    // fragmenty pociętego obrazu

    for (const ArrangedPiece &i : arrangedPieces)
    {
        const int columnNumber = i.position.first; // kolumna
        const int width = i.piece.cols; // wiersz

        const int rowsNumber = i.position.second;
        const int height = i.piece.rows;

        sizeEveryCols[columnNumber] = std::max(width, sizeEveryCols[columnNumber]);
        sizeEveryRows[rowsNumber] = std::max(height, sizeEveryRows[rowsNumber]);
    }

    sizeEveryCols.insert(sizeEveryCols.begin(), 0);
    for (size_t i = 1; i < sizeEveryCols.size(); i++)
    {
        widthEveryCols.push_back(sizeEveryCols[i]);
        sizeEveryCols[i] += sizeEveryCols[i - 1];
    }
    sizeEveryRows.insert(sizeEveryRows.begin(), 0);
    for (size_t i = 1; i < sizeEveryRows.size(); i++)
    {
        heightEveryRows.push_back(sizeEveryRows[i]);
    }
}

```

```

        sizeEveryRows[i] += sizeEveryRows[i - 1];
    }

    // utworzenie macierzy wynikowej, która przyjmuje w rozmiarze sumaryczną wartość
    rozmiaru kolumn i wierszy

    cv::Mat matrixOutput(sizeEveryRows.back(), sizeEveryCols.back(),
        arrangedPieces[0].piece.type(), cvScalar(255, 255, 255));

    for (size_t i = 0; i < arrangedPieces.size(); i++)
    {
        const int columnNumber = arrangedPieces[i].position.first;
        const int rowsNumber = arrangedPieces[i].position.second;

        const int columnWidth = widthEveryCols[columnNumber];
        const int columnHeight = heightEveryRows[rowsNumber];

        const cv::Rect rectangle(sizeEveryCols[columnNumber], sizeEveryRows[rowsNumber],
            columnWidth, columnHeight);

        cv::Size size(columnWidth, columnHeight);
        cv::Mat resizedPiece(columnWidth, columnHeight, arrangedPieces[0].piece.type(),
            cvScalar(255, 255, 255));
        cv::resize(arrangedPieces[i].piece, resizedPiece, size);
        /*
        std::cout << "Rectangle-> Height: " << rectangle.height << " Width: " <<
        rectangle.width << " x=" << rectangle.x << " y=" << rectangle.y << std::endl;
        std::cout << "Matrix-> Height: " << matrixOutput.rows << " Width: " <<
        matrixOutput.cols << std::endl;
        std::cout << "ResizedPiece-> Height: " << resizedPiece.rows << " Width: " <<
        resizedPiece.cols << std::endl;
        */
        // "kopiowanie" kawałka do macierzy wynikowej na ustalonych wcześniej współrzędnych
        resizedPiece.copyTo(matrixOutput(rectangle));
    }

    return matrixOutput;
}

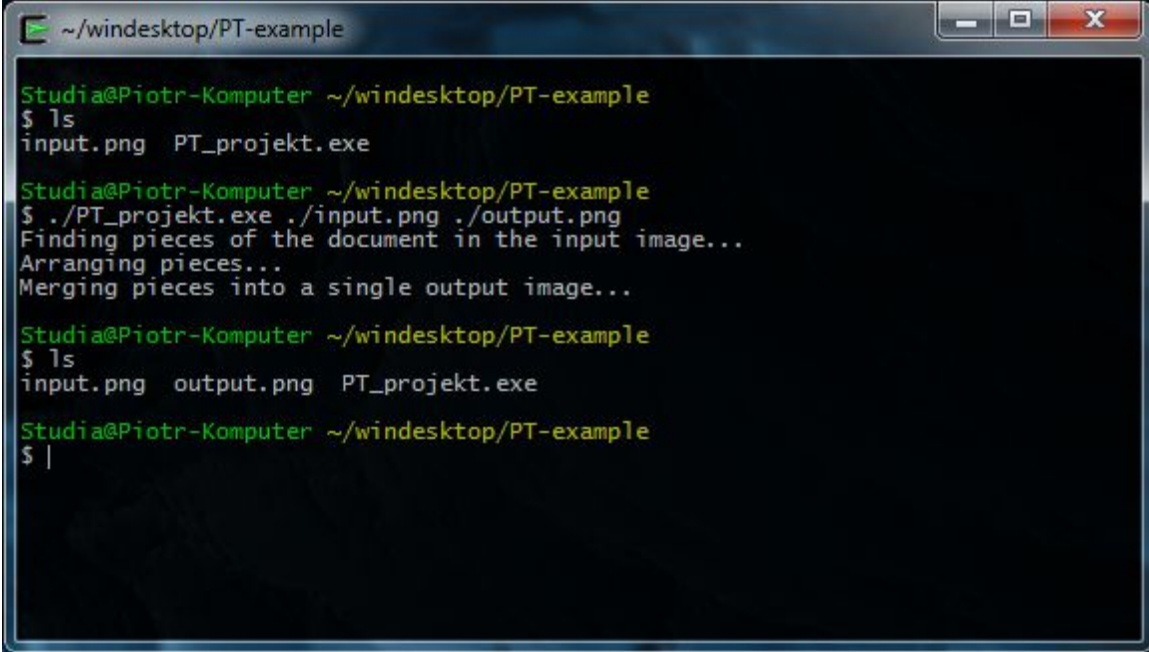
```

**Algorytm łączenia fragmentów obrazu w całość - kod C++**



# Instrukcja użytkowania aplikacji

Aplikacja działa w wierszu poleceń. Przyjmuje ona dwa argumenty, które są nazwami kolejno pliku wejściowego i wyjściowego. Po uruchomieniu program odczytuje obraz wejściowy, analizuje go i przekształca do postaci wyjściowej, a następnie zapisuje. Po zakończeniu pracy program zamyka się natychmiast, bez konieczności interakcji z użytkownikiem.



```
~/windesktop/PT-example

Studia@Piotr-Komputer ~/windesktop/PT-example
$ ls
input.png  PT_projekt.exe

Studia@Piotr-Komputer ~/windesktop/PT-example
$ ./PT_projekt.exe ./input.png ./output.png
Finding pieces of the document in the input image...
Arranging pieces...
Merging pieces into a single output image...

Studia@Piotr-Komputer ~/windesktop/PT-example
$ ls
input.png  output.png  PT_projekt.exe

Studia@Piotr-Komputer ~/windesktop/PT-example
$ |
```

**Przykład użycia programu**

# Testy

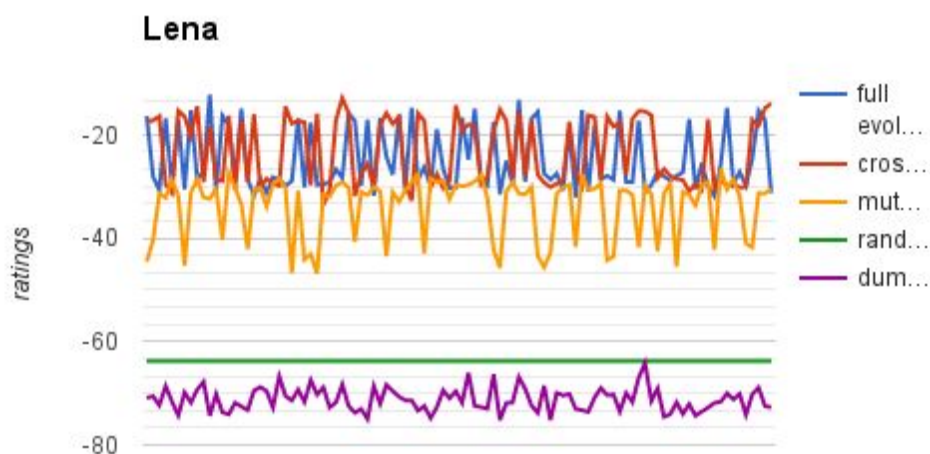
Testowaliśmy program na 3 różnych zbiorach testowych. Dla każdego ze zbiorów sprawdziliśmy działanie pełnego algorytmu, oraz poszczególnych jego elementów w celach porównawczych:

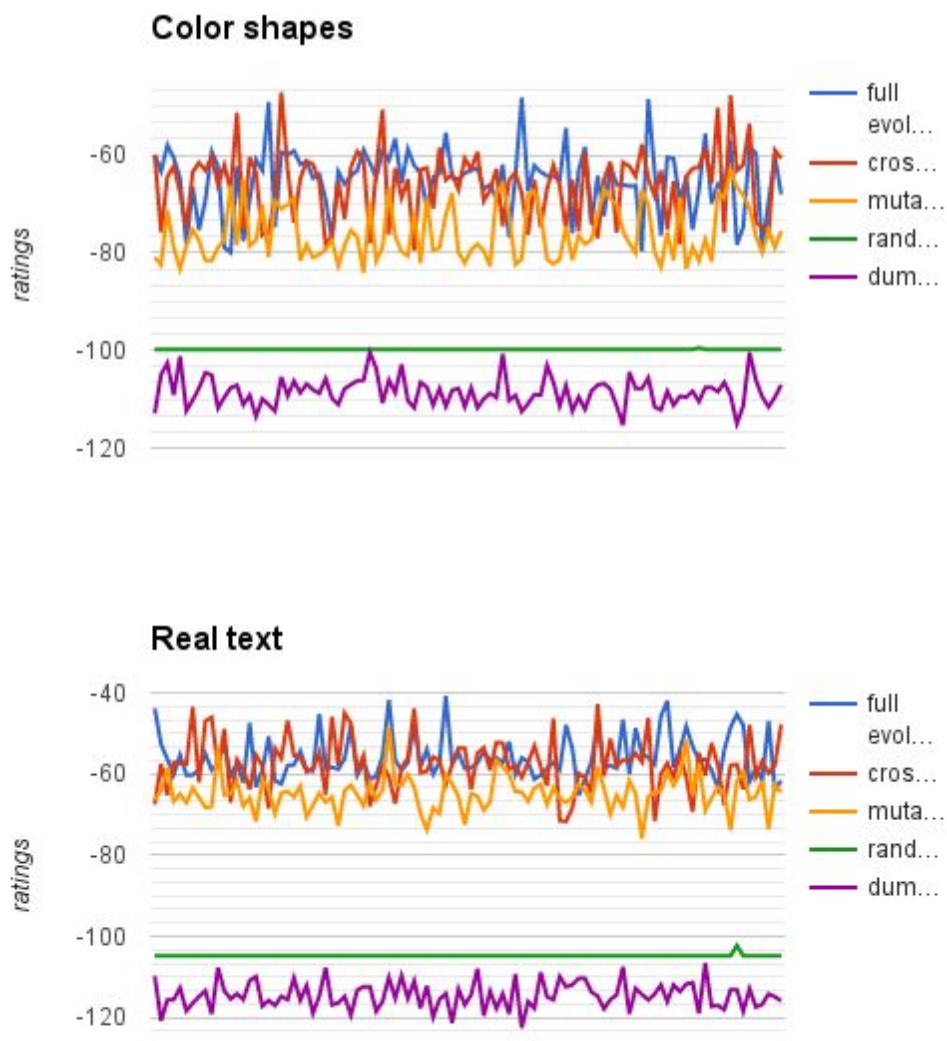
- pełen algorytm ewolucyjny
- wyłącznie krzyżowanie osobników z małego, losowego zbioru wejściowego
- wyłącznie mutacja osobników z małego, losowego zbioru wejściowego
- wybór najlepszego z losowych osobników z dużego zbioru
- całkowicie losowy osobnik

Program bazuje na liczbach pseudolosowych i dlatego wyniki testów powtórzonego testów mogą się bardzo różnić między sobą. Z tego powodu każdy z 15 testów wykonaliśmy 100 razy w celu uzyskania miarodajnych wyników.

## Wyniki testów:

Wyniki testów dla każdego z obrazów zostały umieszczone na jednym wykresie, aby ułatwić wizualne porównanie. Nie są one jednak ze sobą powiązane i każdy z testów leżących na tej samej współrzędnej osi X, był rozpoczęty z innym ziarnem.

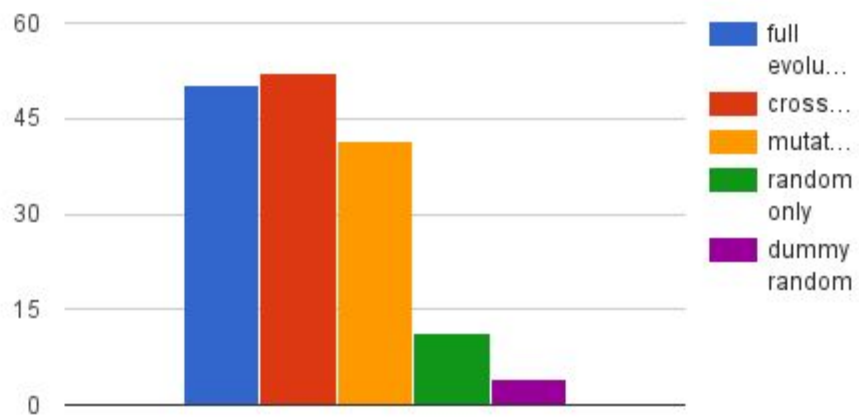




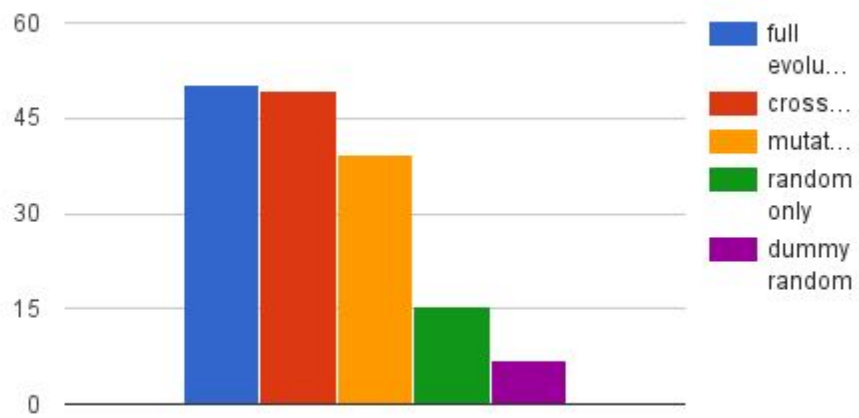
## Średnie wyniki:

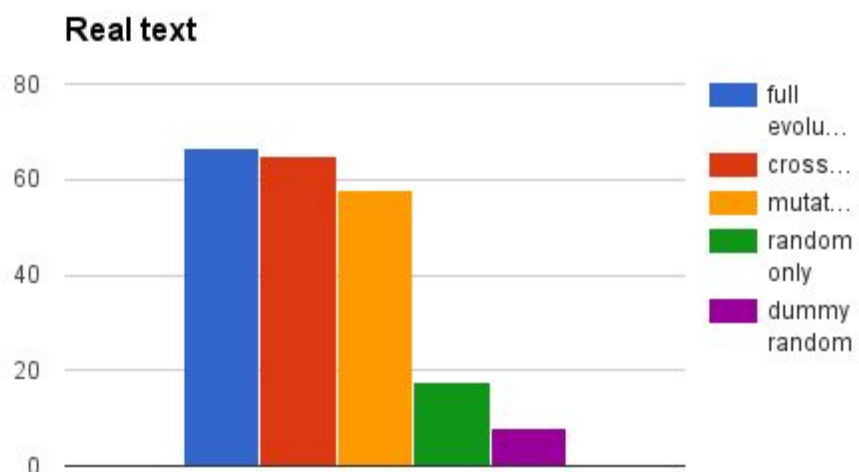
W celu poprawienia czytelności, przedstawione wyniki zostały znormalizowane, poprzez odjęcie od każdej ze średnich najmniejszego ze wszystkich wyników, uzyskanego w przedstawianej serii testów.

Lena



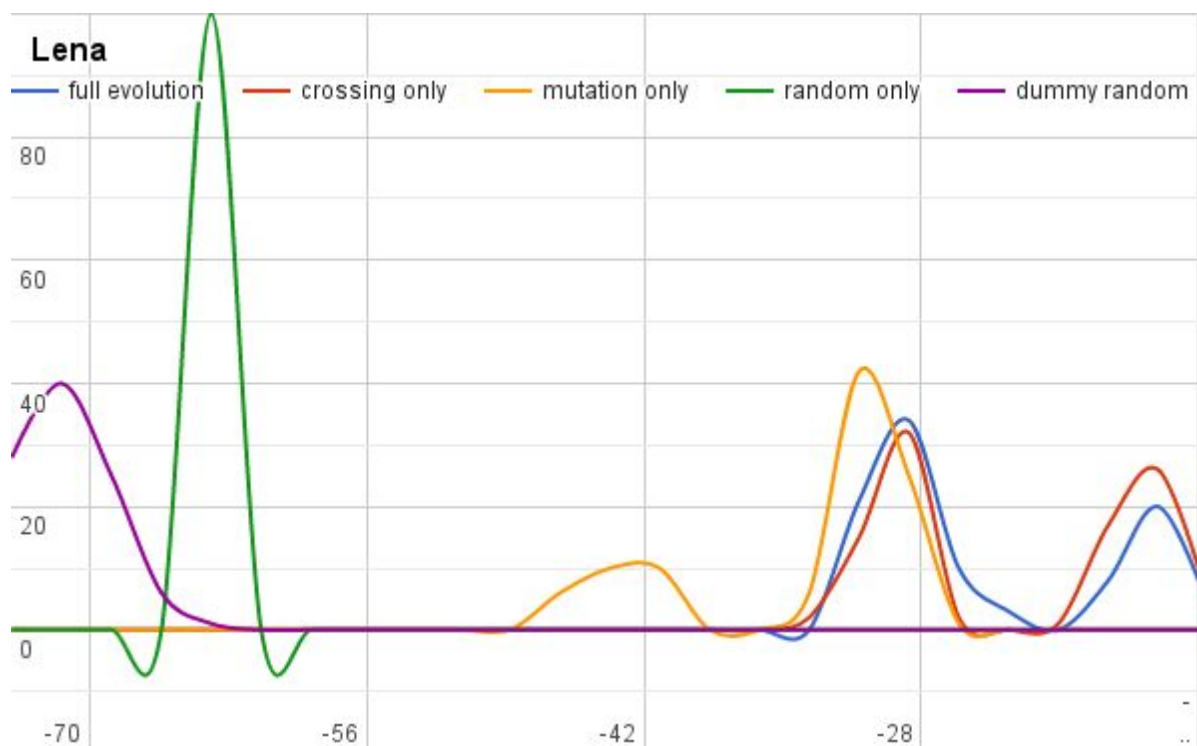
Color shapes

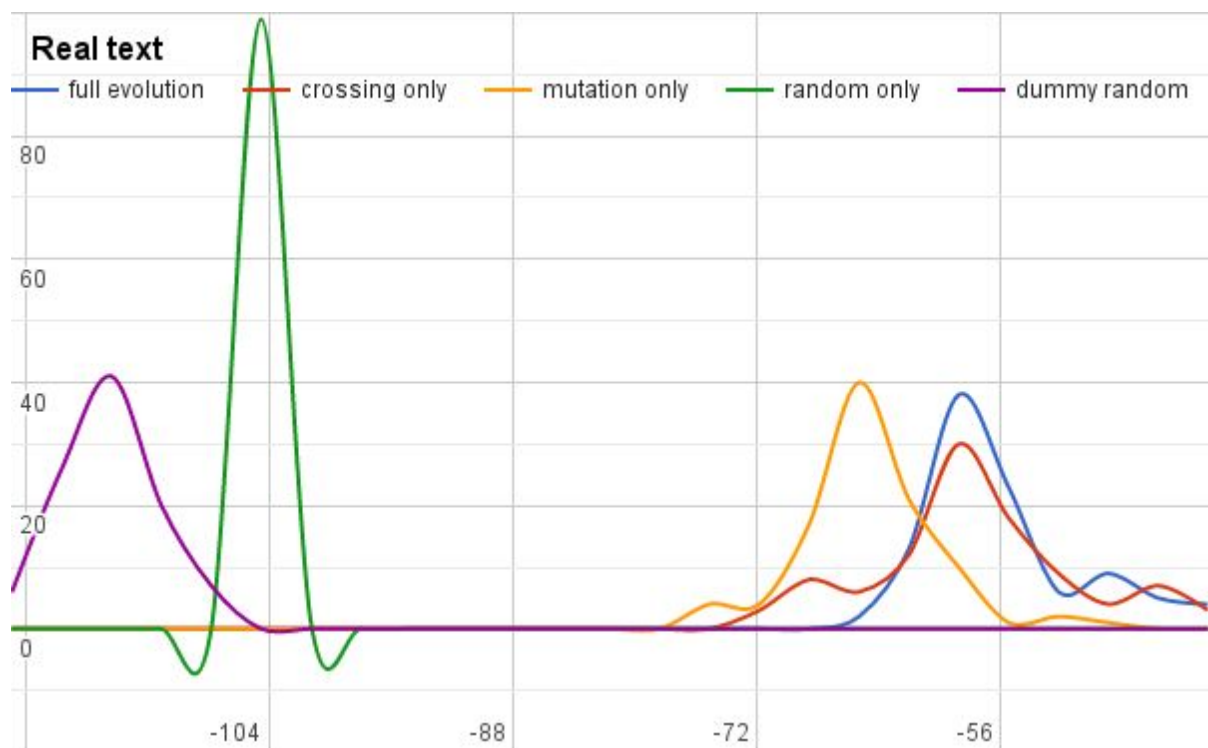
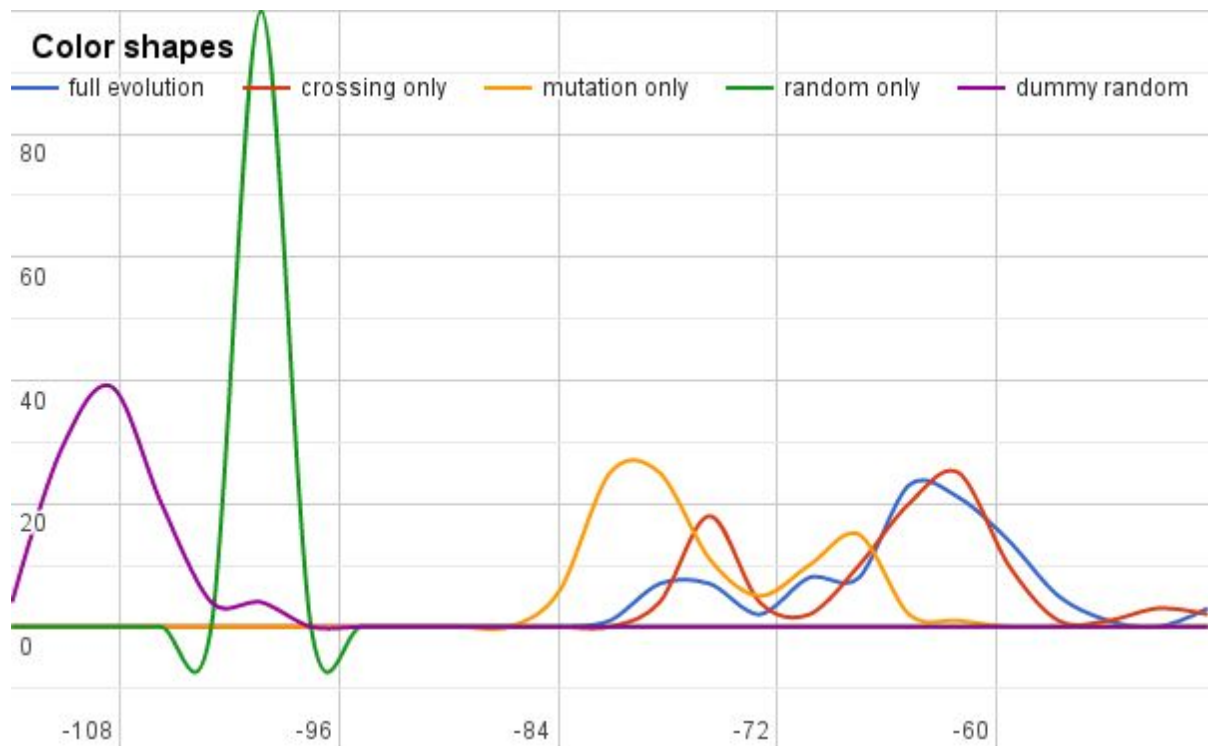


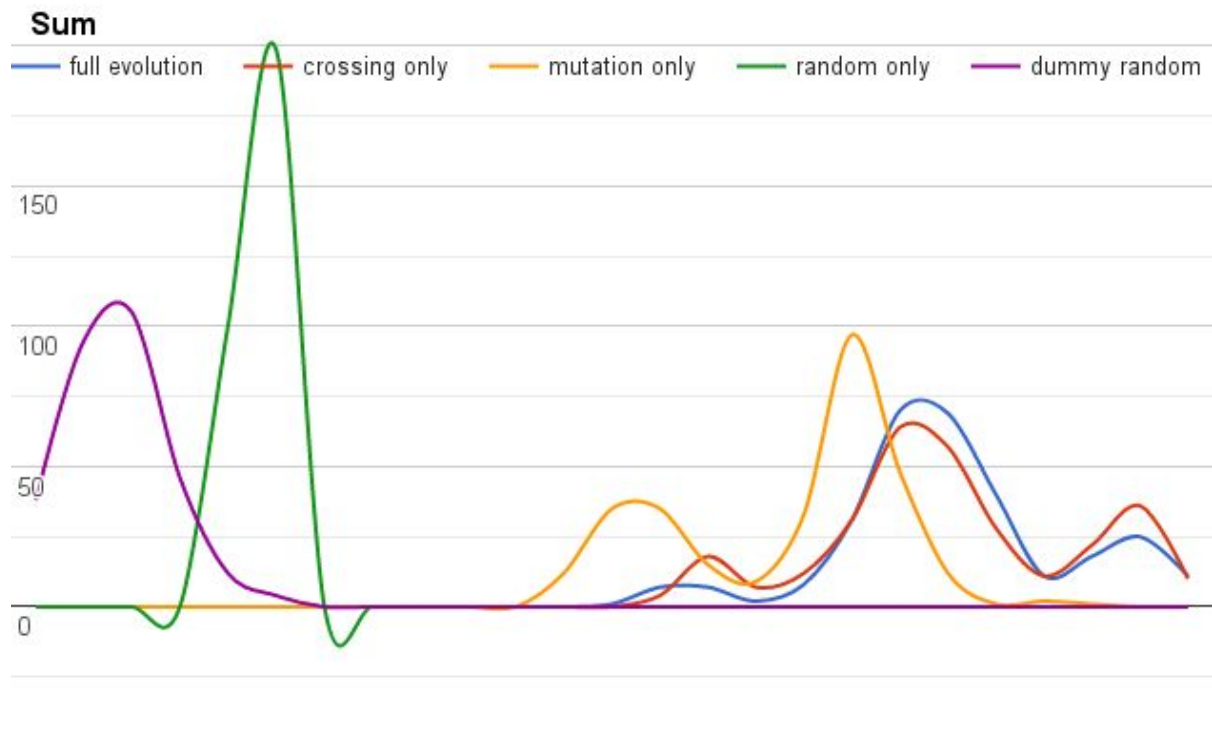


## Rozkłady wyników:

Rozkłady przedstawiają częstotliwość występowanie wyników o określonej jakości w każdym z wariantów programu. Wykresy zostały uzyskany poprzez podziały przestrzeni wyników na 25 przedziałów o jednakowej długości.



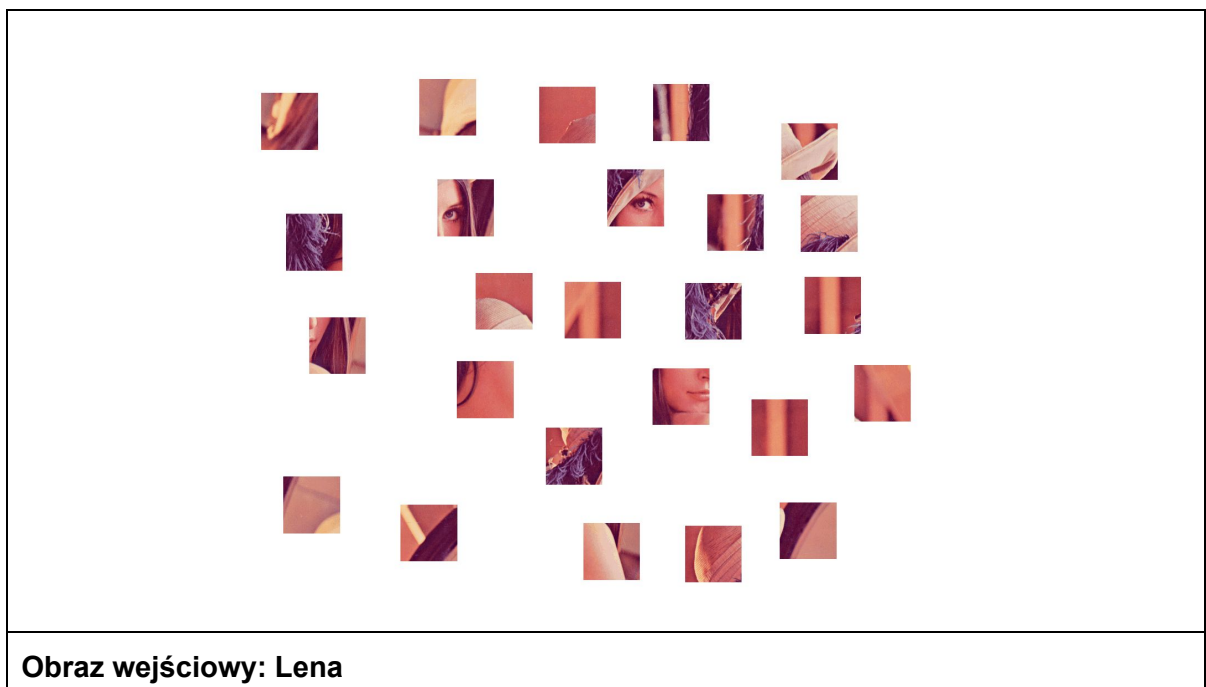




## Testowanie na przykładowych obrazach

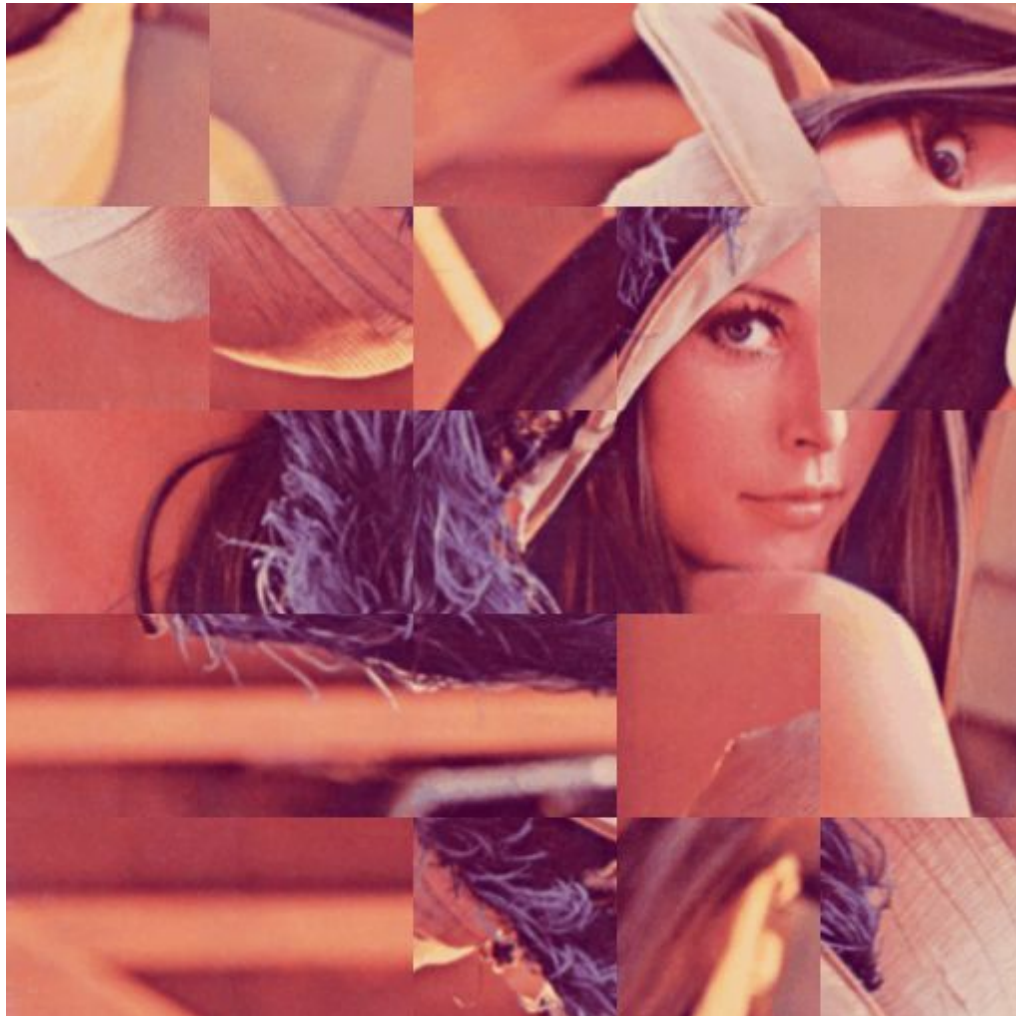
Lena

➤ Obraz wejściowy





➤ Obraz wyjściowy

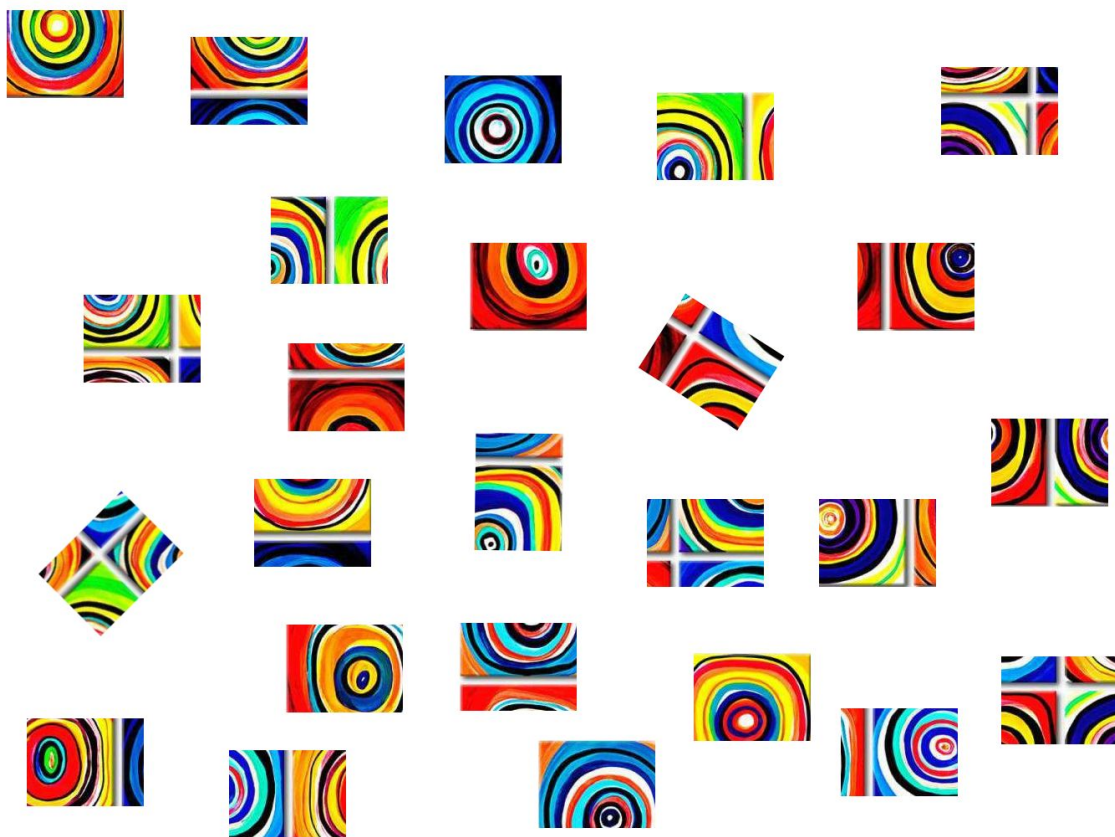


**Test: Lena**

## Color shapes

➤ Obraz wejściowy





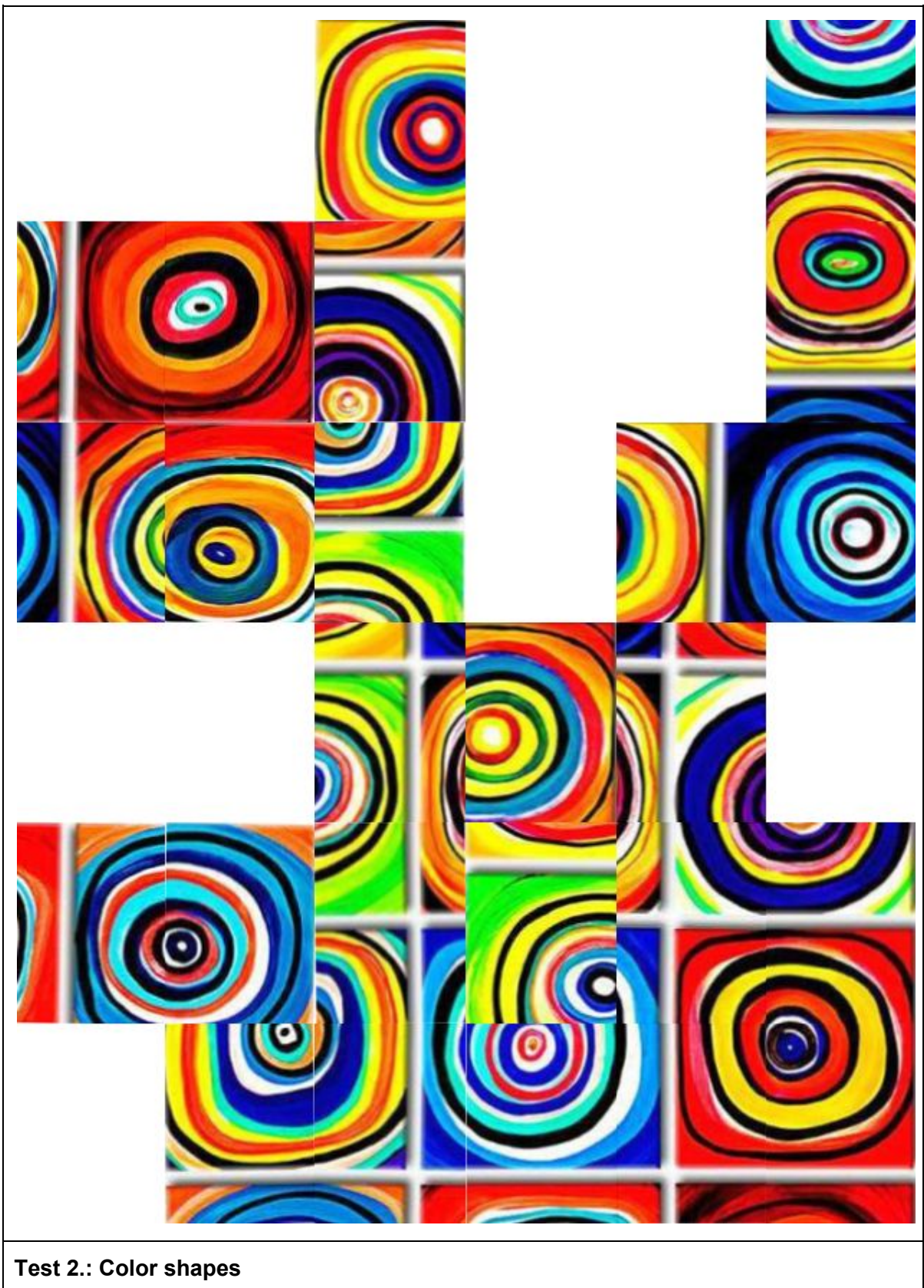
**Obraz wejściowy: Color shapes**

➤ **Obraz wyjściowy**



Test 1.: Color shapes

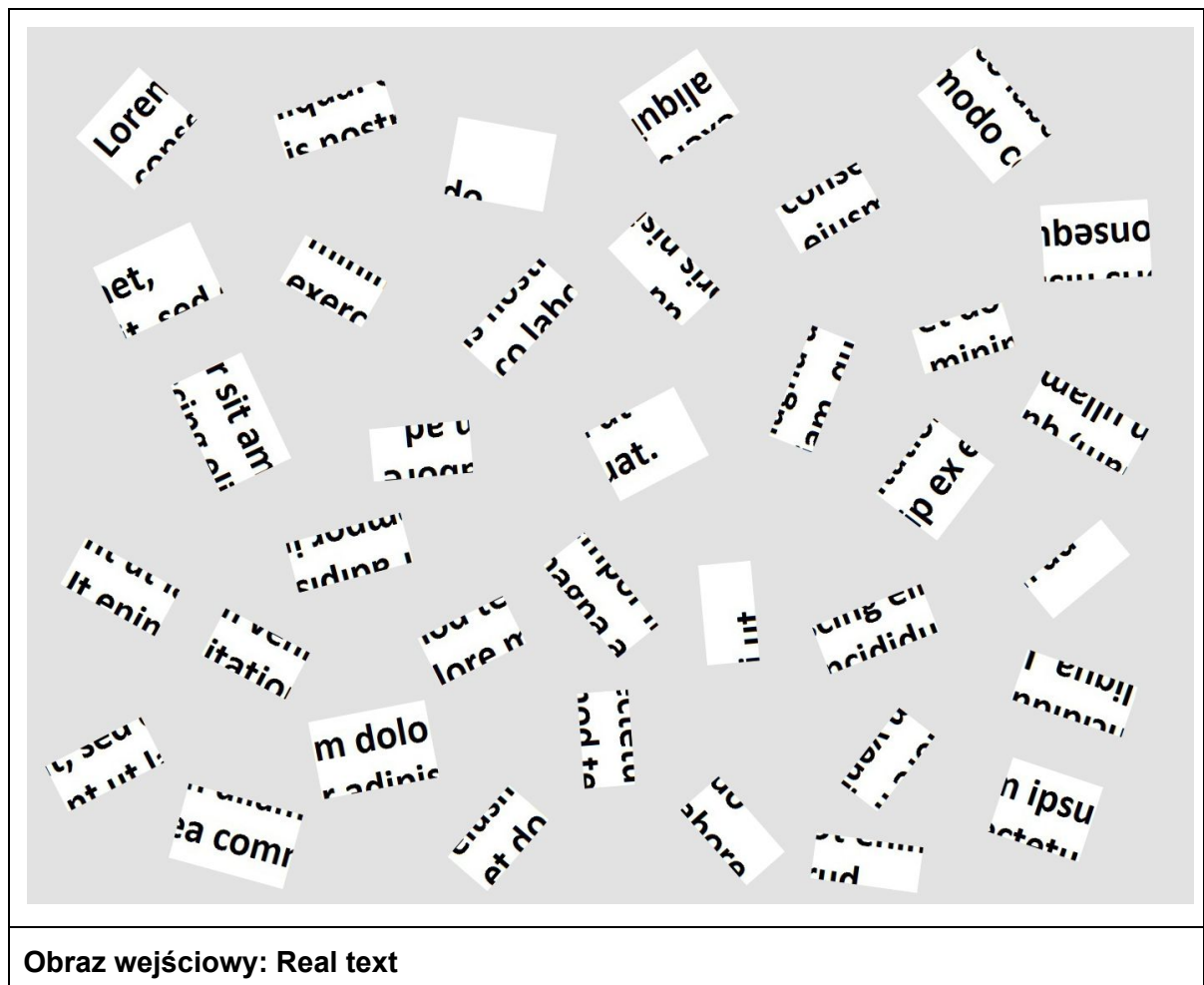




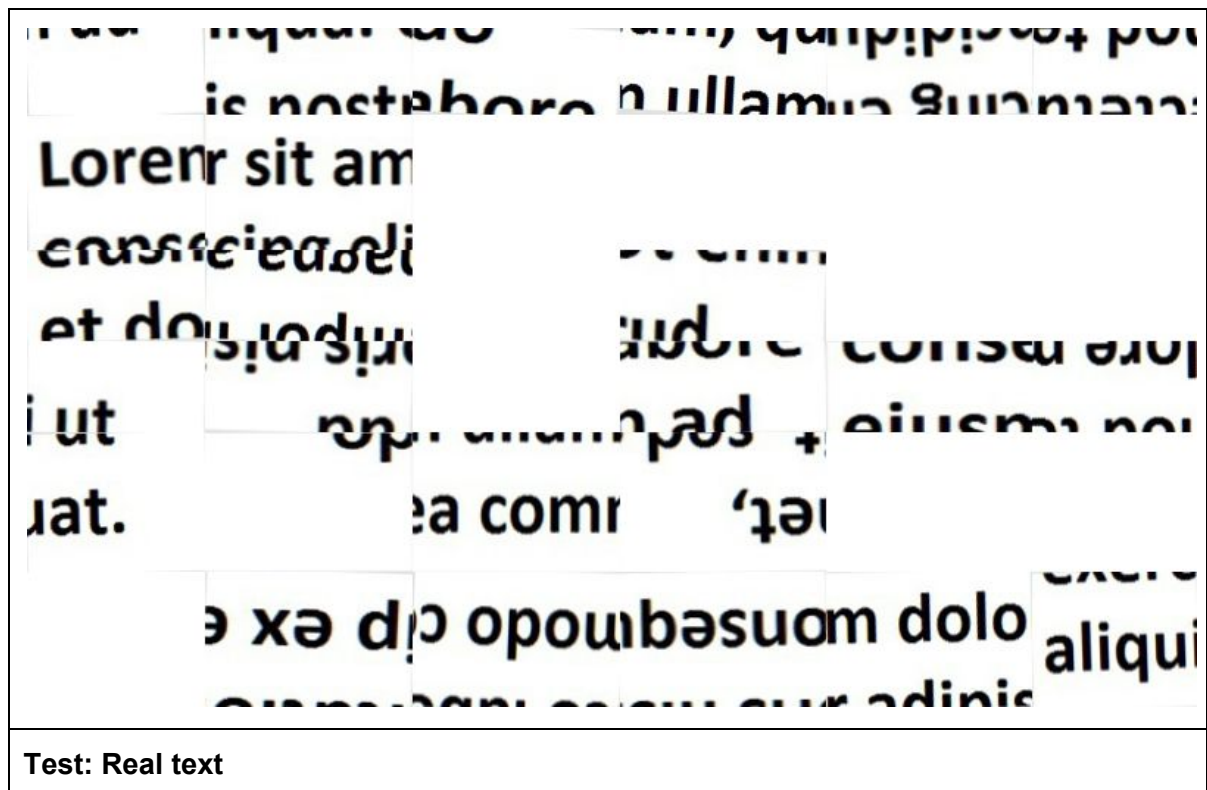
Test 2.: Color shapes

Real text

➤ Obraz wejściowy



➤ Obraz wyjściowy



# Podsumowanie

Wszystkie zaplanowane przez nas prace zostały wykonane według harmonogramu i wcześniejszych założeń.

W przyszłości projekt może być nadal rozwijany, np. poprzez dodanie bardziej precyzyjnych algorytmów dopasowujących do siebie elementy lub modyfikację interfejsu programu.

# Repozytorium

- <https://github.com/inf106605/PT>

# Spis literatury

- <http://stackoverflow.com/questions/28224938/what-is-the-correct-way-to-straighten-a-rotated-rectangle-with-opencv-python>
- <http://codesanswer.com/question/215617-how-to-straighten-a-rotated-rectangle-area-of-an-image-using-opencv-in-python>
- <http://stackoverflow.com/questions/11148807/how-to-convert-this-c-code-to-c-in-opencv>
- <http://answers.opencv.org/question/3609/opencv-cut-image-from-the-other-image/>
- <http://stackoverflow.com/questions/23134304/crop-out-part-from-images-findcontours-opencv-java>
- <http://opencv-srf.blogspot.com/>
- [https://en.wikipedia.org/wiki/Evolutionary\\_algorithm](https://en.wikipedia.org/wiki/Evolutionary_algorithm)