Podstawy Teleinformatyki
Informatyka, semestr VI
Politechnika Poznańska

PUZZLE-STACKER

podpowiadarka do puzzli
z wykorzystaniem biblioteki OpenCV

Autorzy :
Łukasz Wolniak
Dominik Krystkowiak
Szymon Zieliński

Spis treści

1. Charakterystyka ogólna projektu

Aplikacja ma wspomagać proces układania puzzli. Podstawowa wersja aplikacji pozwala na wczytanie obrazu niepoukładanych puzzli, a potem, gdy posiadamy obraz, jaki będą prezentować puzzle po ułożeniu, aplikacja ma możliwość wskazania, gdzie konkretny puzzel z obrazu puzzli nieułożonych znajduje się na obrazie. W przypadku gdy nie posiadamy obrazu puzzli po ułożeniu, to aplikacja oferuje opcje, aby sam spróbował go poukładać za użytkownika.
Zespół zdecydował się na wykonanie tego projektu z powodu chęci wykorzystania poznanej wiedzy teoretycznej i praktycznej na przedmiocie “Przetwarzanie obrazów i systemy wizyjne” w większym projekcie.

2. Wymagania

Wymagania funkcjonalne:

- możliwość wczytania i zapisywania obrazów w programie,
- dobieranie parametrów przez użytkownika, dzięki którym program odpowiednio rozpozna wszystkie niepoukładane puzzle z obrazu,
- wskazywanie miejsca na ułożonym obrazie, gdzie dany osobny puzzel powinien się znajdować,
- przejrzysty interfejs graficzny,
- układanie puzzli.

Wymagania niefunkcjonalne:

- zapewnienie odpowiedniej wydajności czasowej i pamięciowej,
- interfejs w języku polskim,
- aplikacja desktopowa przeznaczona na systemy Windows, Linux, Mac OS
- nie jest wymagane połączenie z Internetem,
- aparat wykonujący zdjęcia w zadowalającej jakości.

3. Środowisko, narzędzia, biblioteki

Do implementacji aplikacji został wykorzystany język programowania Python, w wersji 3.5. Wybrano ten język z powodu łatwego użycia funkcji z OpenCV (nie wymagane jest zainstalowanie dodatkowego wrappera jak w językach C# (Emgu CV) oraz Java (JavaCV)).

Środowisko, biblioteki oraz narzędzia wykorzystywane do budowy aplikacji :

- Środowisko - Pycharm Community Edition.
- Biblioteki:
    - OpenCV 3.4.1.15 - biblioteka napisana w języku C umożliwiająca przetwarzanie oraz obróbkę obrazów w czasie rzeczywistym. Oparta na otwartym kodzie i zapoczątkowana przez firmę Intel.
    - Matplotlib 2.2.2 - rozszerzenie numeryczne NumPy - wsparcie dla operacji na dużych, wielowymiarowych tablicach i macierzach, a także duży zbiór zaawansowanych funkcji matematycznych do obsługi tych macierzy.
    - PyQT 5 - biblioteka umożliwiająca tworzenie interfejsów okienkowych opartych o międzyplatformowy framework QT.
    - PIL - rozszerzenie dla Pythona które dodaje obsługę grafiki np. otwieranie, modyfikowanie, zapisywanie.
- Narzędzia:
    - Repozytorium - Git - hosting GitHub.
    - Visual Paradigm - oprogramowanie wykorzystywane do tworzenia diagramów UML (np. przypadków użycia).
    - QT Designer - oprogramowanie zawarte w wieloplatformowym środowisku programistycznym QT Creator. Pozwala na stworzenie graficznego interfejsu użytkownika (GUI).

4. Funkcje programu

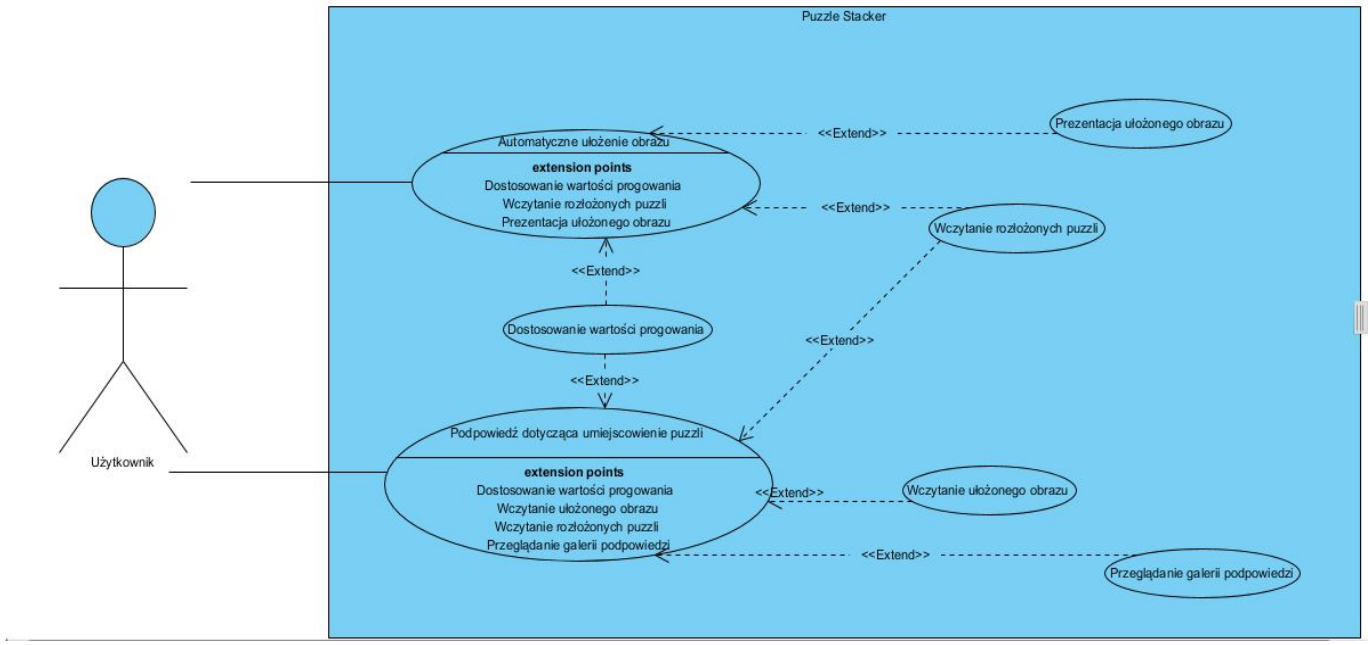
Rysunek 4.1 przedstawia diagram przypadków użycia, pokazujące przegląd możliwych działań w aplikacji Puzzle Stacker.
Głównymi operacjami w systemie jest:

- automatyczne ułożenie puzzli (jeżeli nie posiadamy gotowego obrazu np. z pudełka),
- podpowiedź dotyczącą umiejscowienia danego puzzla.

Dodatkowo system wspierany jest przez następujące funkcjonalności (umożliwiające wykonanie głównych zadań):

- wycięcie puzzla,
- wczytanie zdjęcia ułożonych i rozłożonych puzzli,
- zmiana ustawień progowania wczytanego zdjęcia,

- określenie pozycji danego puzzla na obrazku.



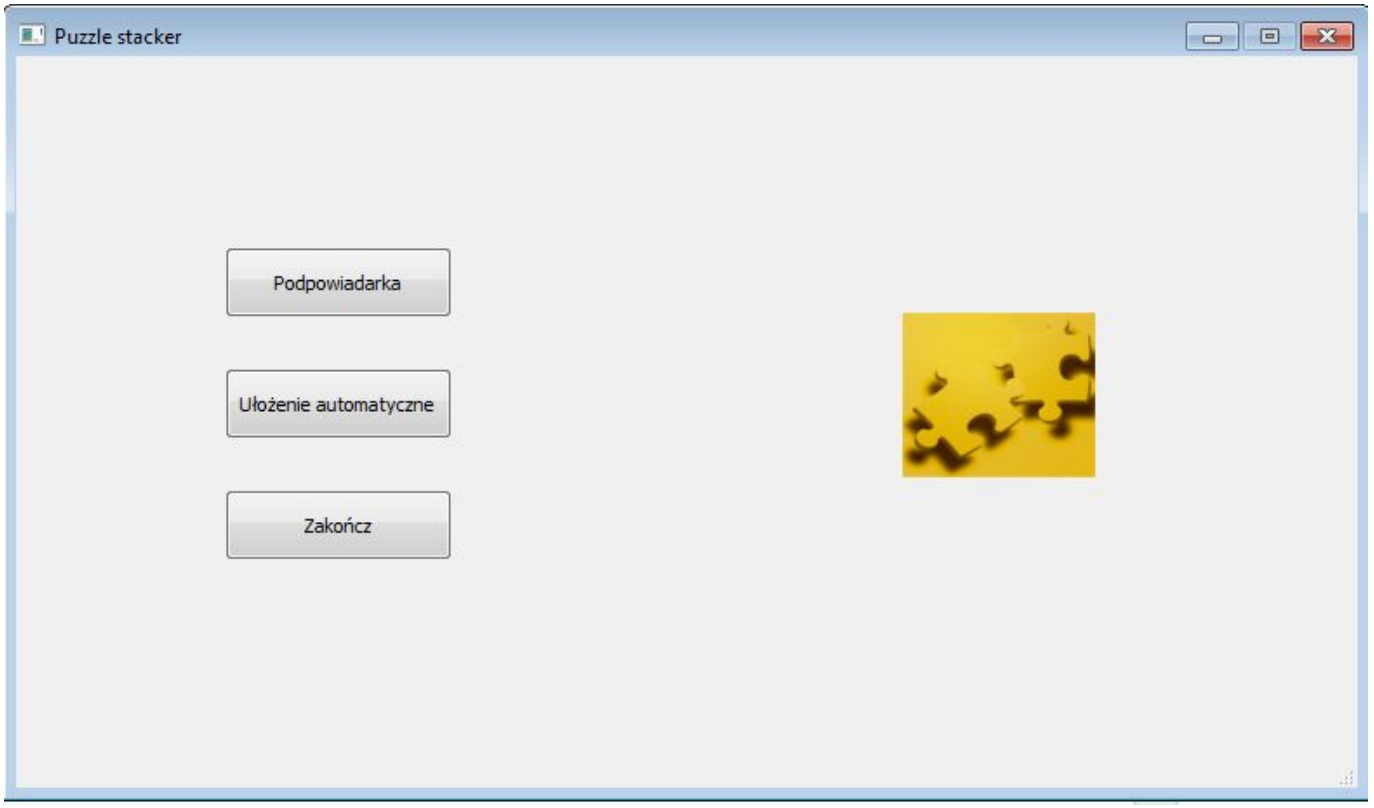
Rys. 4.1.: Wczytanie obrazu do programu

5. Projekt interfejsu graficznego

W rozdziale zostały przedstawione interfejsy graficzne programu wykonane przy użyciu programu QT Designer. Każdy przycisk posiada podpowiedź w formie dymku, który ujawnia się po wskazaniu kursorem elementu.
Rysunki pokazują gotowe interfejsy:

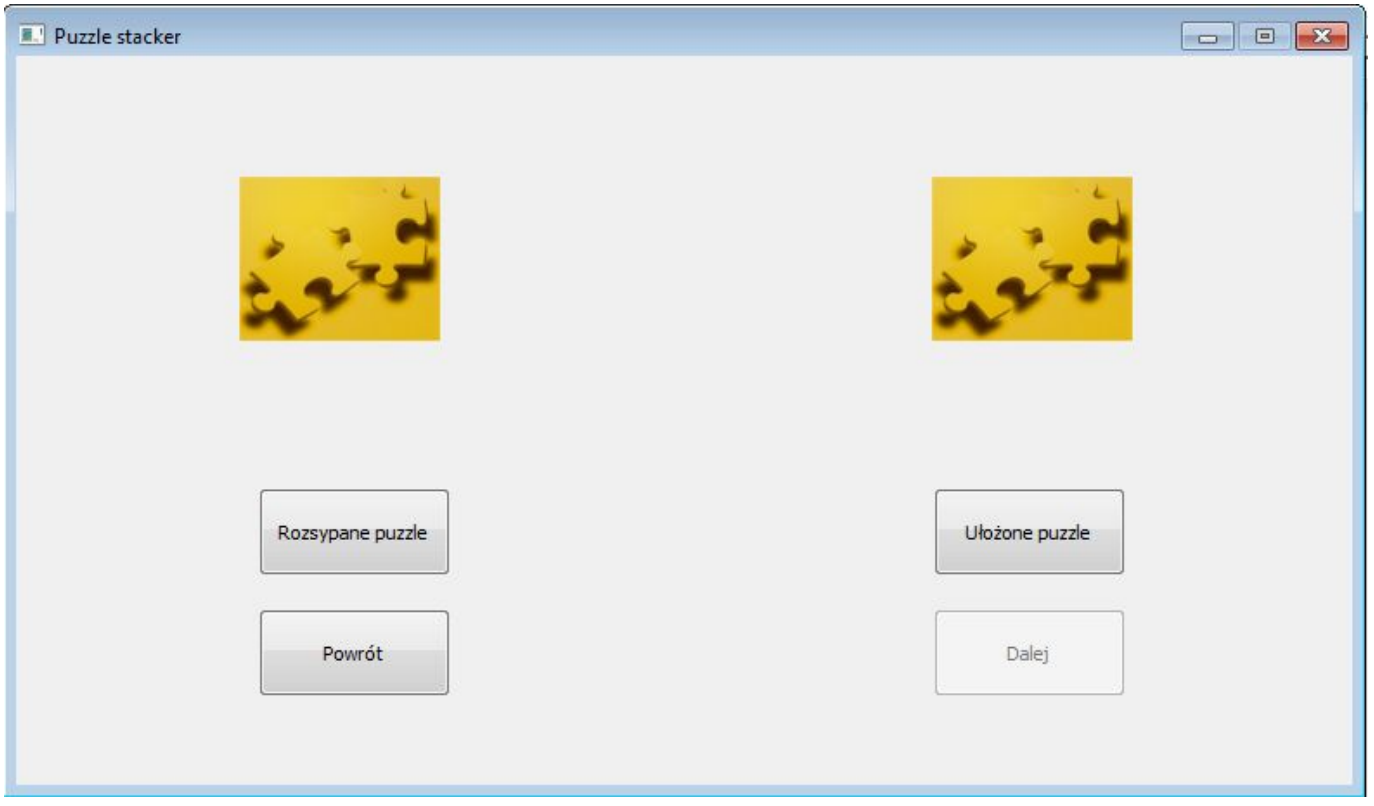
- interfejs główny
- interfejs wczytania dwóch obrazów
- interfejs wyboru obrazu
- interfejs dostosowania parametrów progowania
- interfejs pomocy
- interfejs wskazywania podpowiedzi
- interfejs wczytania obrazu
- interfejs prezentacji wyniku automatycznego układania

Rysunek 5.1. przedstawia ekran główny, który użytkownik widzi jako pierwszy po uruchomieniu programu. Z tego widoku użytkownik może przejść do dwóch trybów pracy aplikacji oraz może zakończyć działanie programu. Pierwszym trybem pracy (podstawowym - przycisk “Podpowiadarka”) jest wskazywanie przez program podpowiedzi na podstawie rozsypanych puzzli oraz zdjęcia ułożonego obrazu (np. z pudełka w którym znajdują się puzzle).



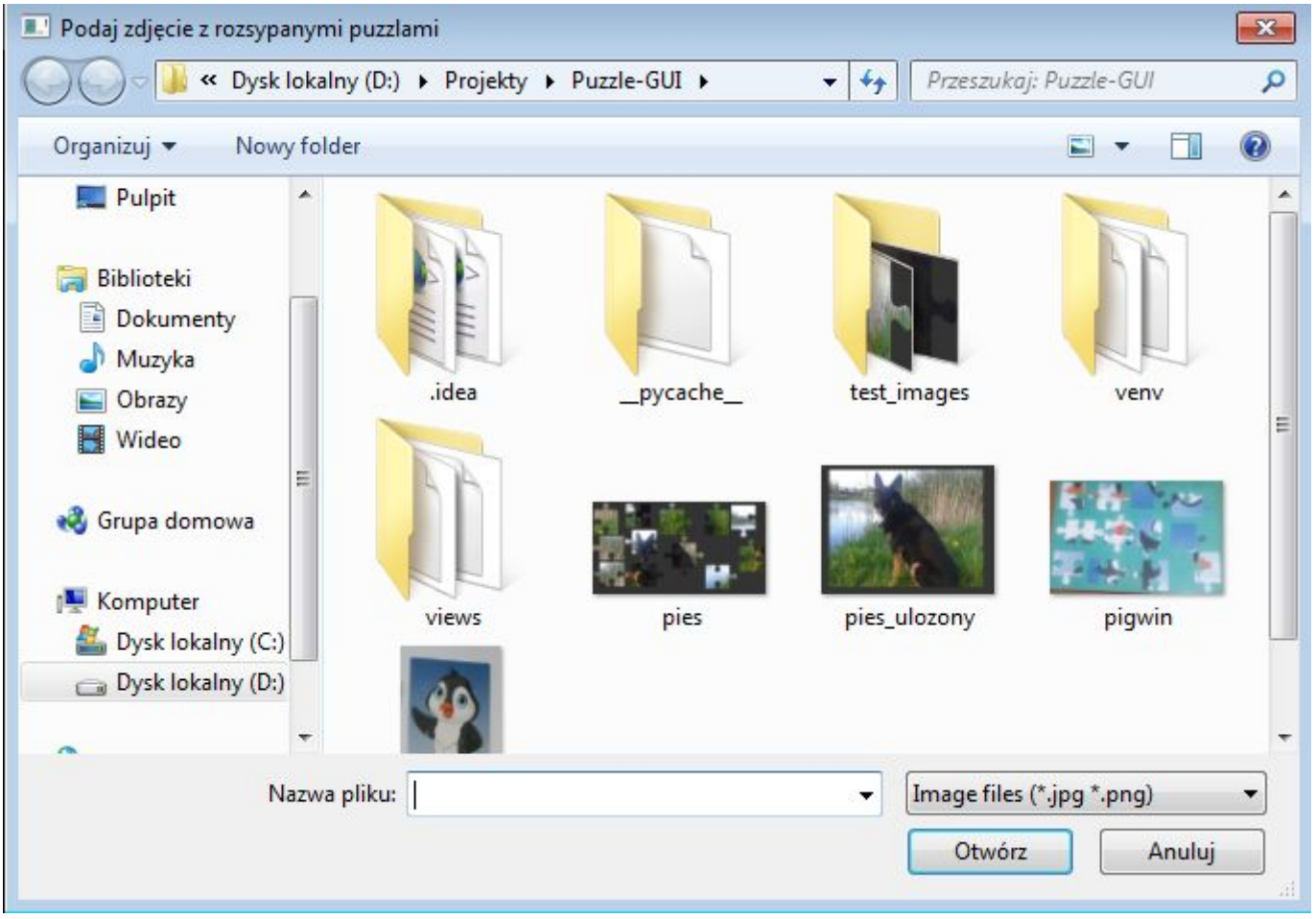
rys. 5.1. Interfejs główny

Rysunek 5.2. przedstawia interfejs wczytania dwóch obrazów. Widok składa się z czterech przycisków. Przycisk “Powrót” umożliwia użytkownikowi przejście do ekranu głównego. Natomiast przyciski “Rozsypane puzzle” oraz “Ułożone puzzle” umożliwiają wczytanie zdjęć/obrazów. Miniaturki wczytanych obrazów prezentowane są poprzez zastąpienie żółtych elementów interfejsu. Przycisk “Dalej” uaktywnia się w momencie wczytania obu wymaganych ilustracji i po kliknięciu przenosi użytkownika do ekranu dostosowania parametrów.



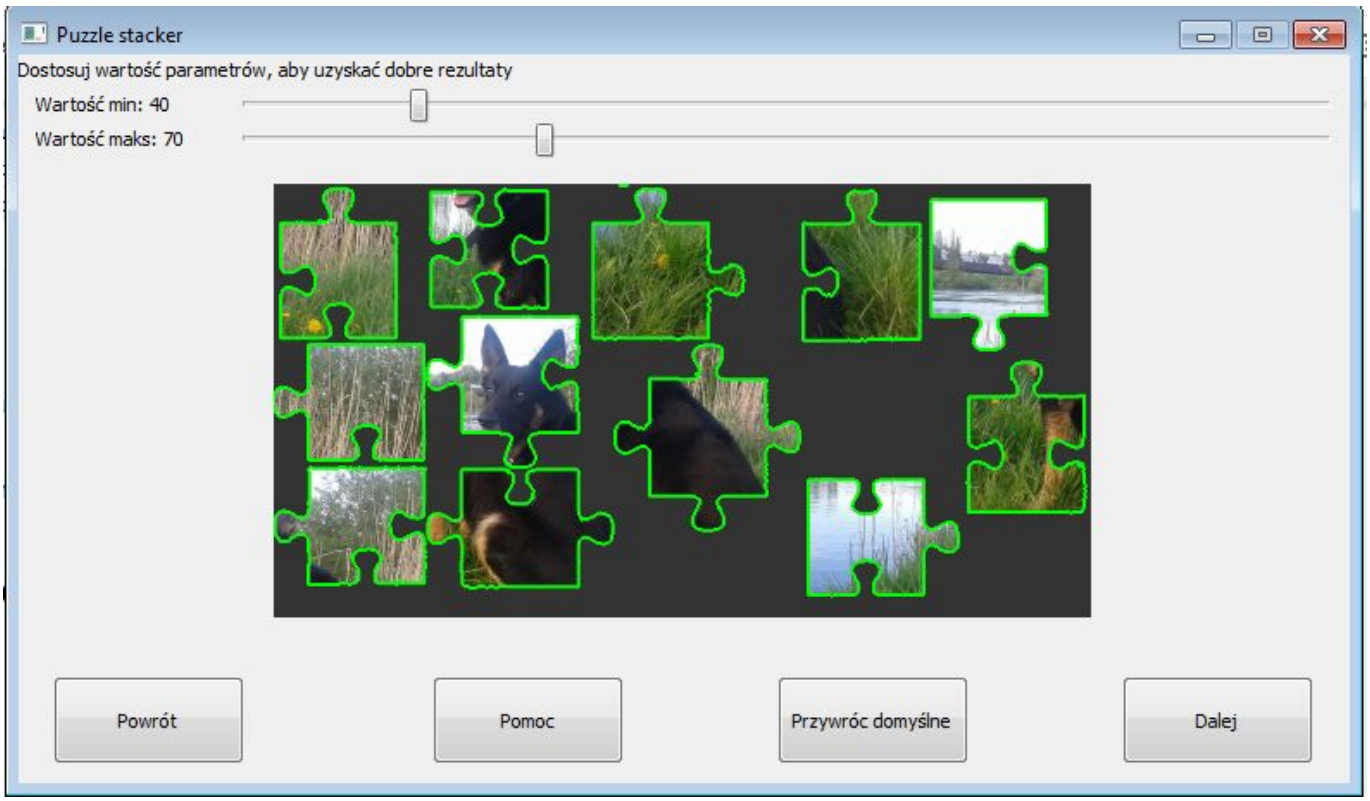
rys. 5.2. Interfejs wczytania dwóch obrazów

Rysunek 5.3. przedstawia widok, który zostaje wyświetlony po kliknięciu na przyciski odpowiedzialne za wczytanie obrazu. Poprzez ten interfejs użytkownik może swobodnie przechodzić pomiędzy folderami oraz wskazać plik graficzny. Program obsługuje wyłącznie pliki w formacie JPG oraz PNG. Pasek tytułowy podpowiada użytkownikowi jaką grafikę powinien wskazać (na rysunku 5.3. “Podaj zdjęcie z rozsypanymi puzzlami”, analogicznie dla ułożonych puzzli “Podaj zdjęcie z ułożonymi puzzlami).



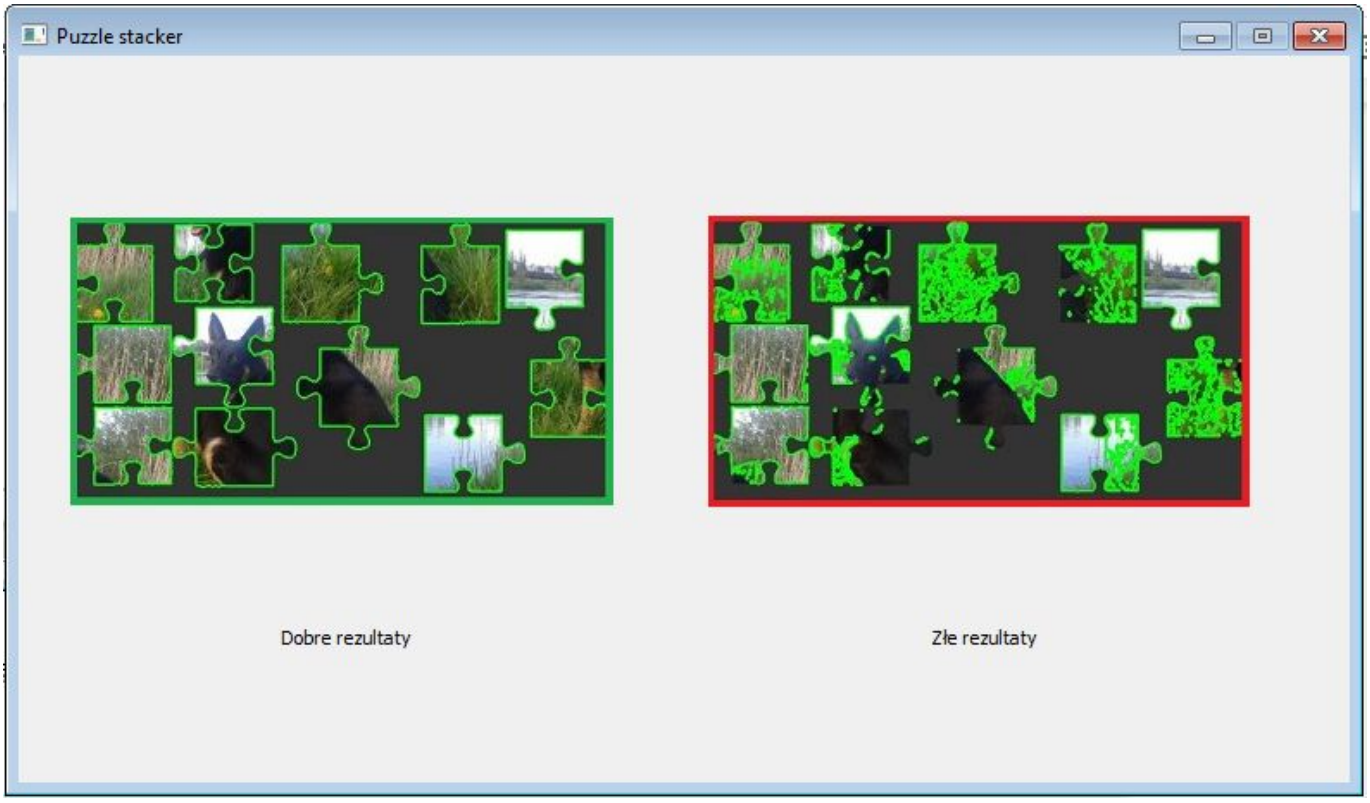
rys. 5.3. Interfejs wyboru obrazu

Rysunek 5.4. przedstawia interfejs wspólny dla obu trybów aplikacji. Dzięki temu widokowi użytkownik może dobrać ręcznie parametry progowania poprzez zmianę pozycji trackbaru (skok o jeden punkt). Jeżeli użytkownik chce powrócić do ustawień domyślnych zdefiniowanych przez twórców aplikacji może to zrobić poprzez kliknięcie przycisku “Przywróć domyślne” (automatycznie wartości trackbarów zmieniane są z wartości ustalonych przez użytkownika na wartości 40 oraz 70). Przycisk “Pomoc” umożliwia wyświetlenie dodatkowego okna aplikacji z pomocą dotyczącą użytkowania tego widoku.



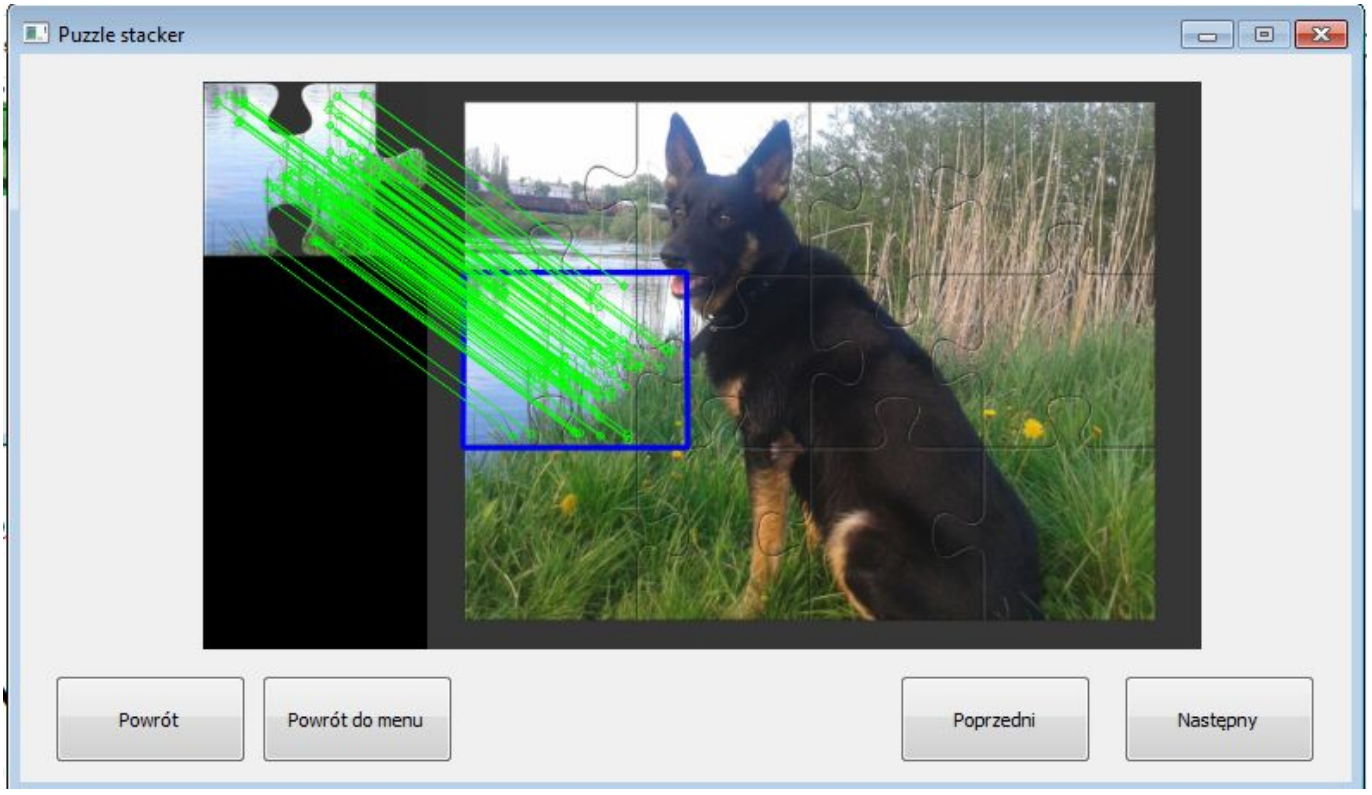
rys. 5.4. Interfejs dostosowania parametrów progowania

Rysunek 5.5. przedstawia pomoc dotyczącą korzystania z ekranu dostosowania parametrów progowania. Pokazuje jak wygląda obraz z dobrze dobranymi parametrami (umożliwiającymi uzyskanie rezultatów) oraz obraz z nieodpowiednimi wartościami. Obie grafiki obwiedzione są odpowiednio zieloną i czerwoną linią.



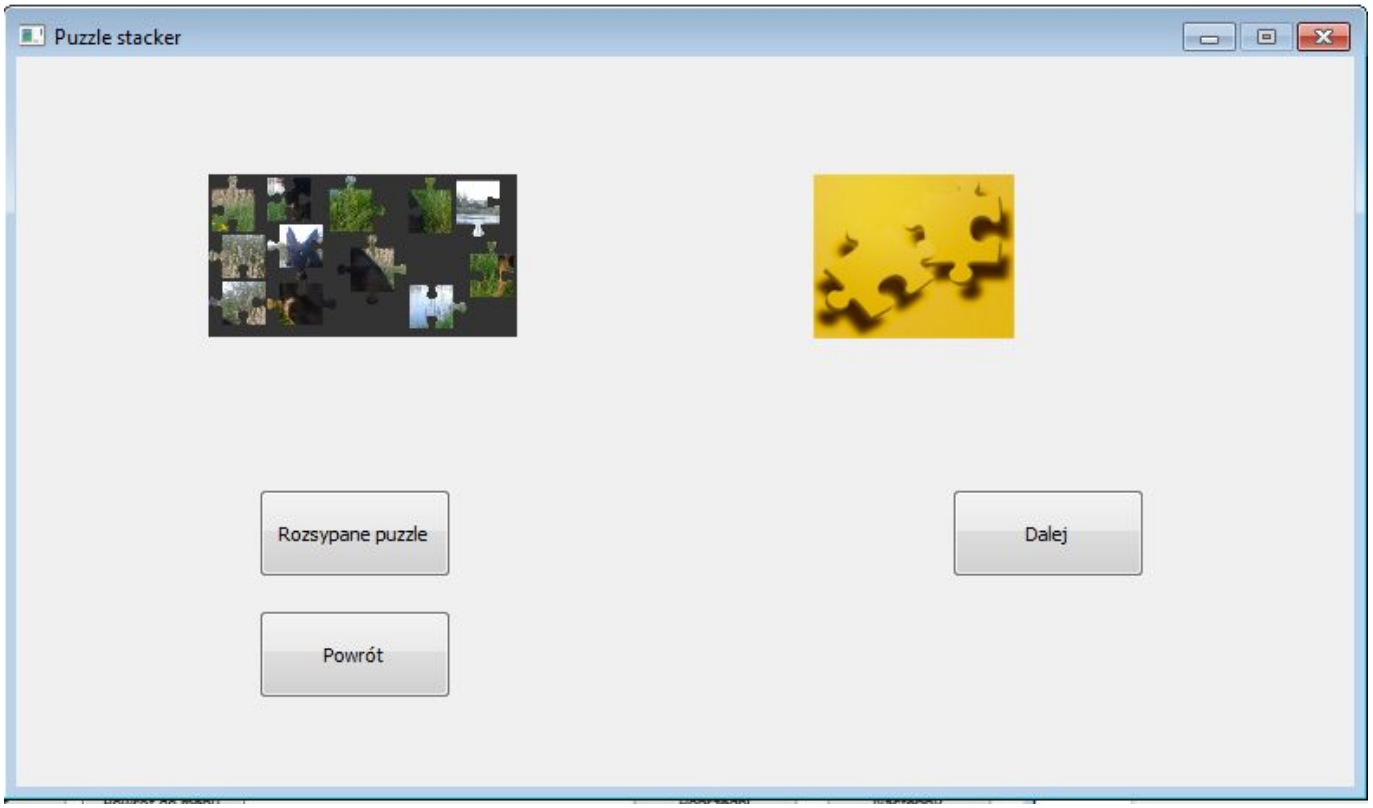
rys. 5.5. Interfejs pomocy

Ilustracja 5.6. przedstawia ekran w którym użytkownik może przeglądać wyniki (podpowiedzi) zwrócone przez działanie programu. Rezultaty prezentowane są poprzez umieszczenie puzzla w lewym górnym rogu, obraz ułożony w prawym, a dopasowanie poprzez niebieski prostokąt oraz zielone linie. Wyniki prezentowane są w formie galerii, a przejście pomiędzy zdjęciami odbywa się poprzez dwa przyciski (“Poprzedni” oraz “Następny).



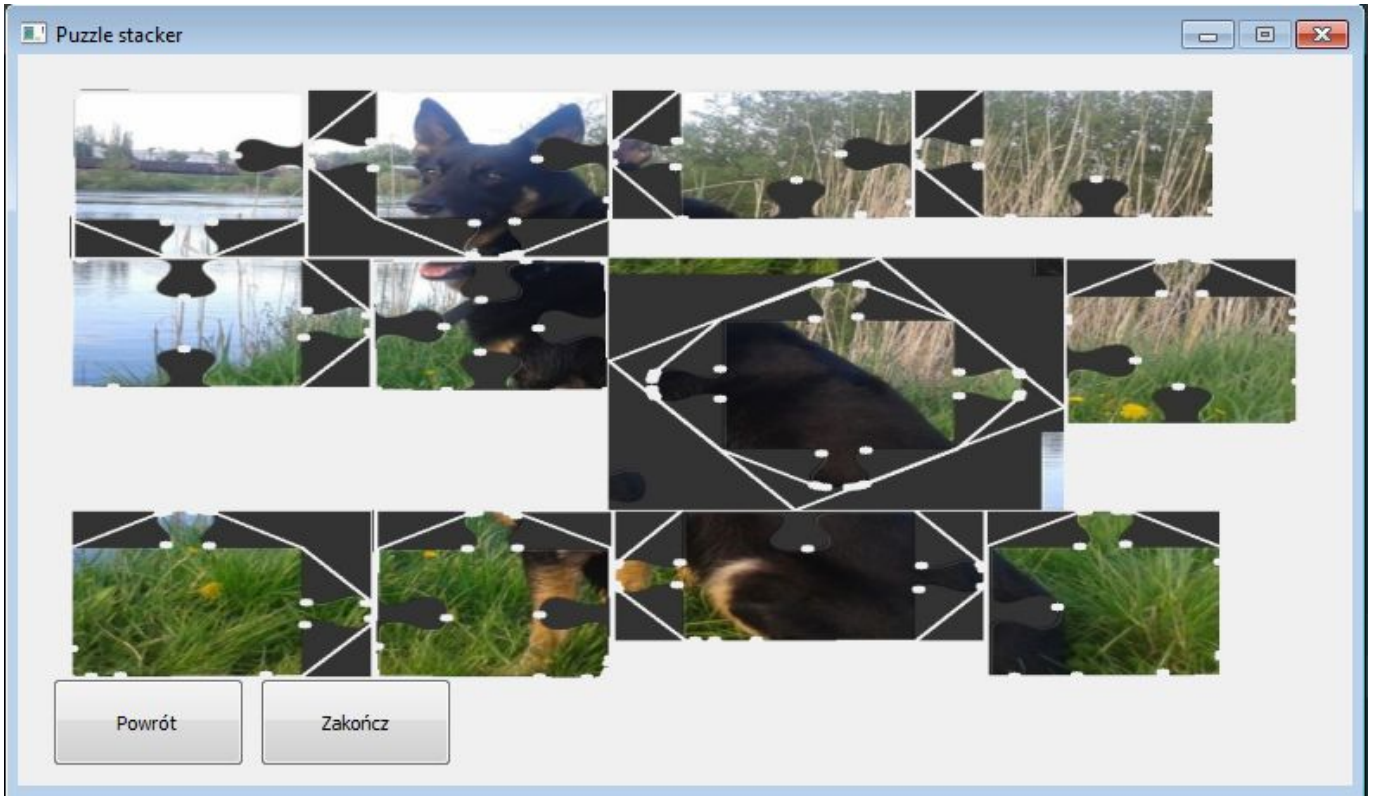
rys. 5.6. Interfejs wskazywania podpowiedzi

Rysunek 5.7. przedstawia interfejs wczytania rozłożonych puzzli. Widok składa się z trzech przycisków. Przycisk “Powrót” umożliwia użytkownikowi przejście do ekranu głównego. Natomiast przycisk “Rozsypane puzzle” umożliwia wczytanie zdjęć/obrazów. Miniaturki wczytanych obrazów prezentowane są poprzez zastąpienie żółtego elementu interfejsu. Przycisk “Dalej” uaktywnia się w momencie wczytania ilustracji i po kliknięciu przenosi użytkownika do ekranu dostosowania parametrów.



rys. 5.7. Interfejs wczytania obrazu

Ilustracja 5.8 prezentuje widok, w którym prezentowany jest wynik funkcji automatycznego układania puzzli na podstawie rozsypanych elementów. Oprócz obejrzenia rezultatu użytkownik może przejść do poprzedniego okna oraz powrócić do ekranu głównego aplikacji.



rys. 5.8. interfejs prezentacji wyniku automatycznego układania

6. Rodzaje progowania

Rozdział prezentuje wyniki uzyskane dla puzzli przez różne rodzaje progowania, których porównanie oraz wybór najlepszego było kluczowe przy obróbce obrazu.

a. Progowanie

Progowanie jest to metoda polegająca na ustaleniu pewnej wartości progowej i porównaniu wartości każdego piksela obrazu z określoną wartością progu (piksele jaśniejsze od wyznaczonego progu otrzymują jedną wartość np. 0, a ciemniejsze przeciwną np. 255). Stosowany do oddzielenia obiektów pierwszoplanowych od tła. Wynikiem tej metody jest obraz w formie binarnej.

rys. 6.1. Progowanie

b. Progowanie adaptacyjne

W tej metodzie obraz dzielony jest na obszary, którego rozmiar dopasowany jest do rozmiaru obiektów. W każdym wydzielonym fragmencie liczone są lokalne charakterystyki (średnia, minimalna i maksymalna wartość intensywności), które będą wykorzystane do obliczenia progu lokalnego. Rysunek 6.2. prezentuje wynik progowania adaptacyjnego.

rys. 6.2. Progowanie adaptacyjne

c. Progowanie metodą Otsu

Metoda automatycznie ustawia próg (progowanie globalne) i ma dobre wyniki dla obrazów, w których histogram ma rozkład bimodalny (tło i obiekt pierwszego planu).

rys. 6.3. Progowanie metodą Otsu

d. Wnioski

Wykonano wiele testów dla różnych obrazów, parametrów oraz rodzajów progowania. Zdecydowano się na progowanie adaptacyjne, które zwracało zadowalające rezultaty.

7. Detektory i deskryptory cech

W rozdziale zostały przedstawione wyniki testów dla poszczególnych detektorów i deskryptorów cech. Detektory cech wykorzystywane są do wskazywania charakterystycznych punktów w obrazie, natomiast deskryptory cech służą do opisywania cechy w oparciu o charakterystykę sąsiedztwa cech wykrytych przez detektor.

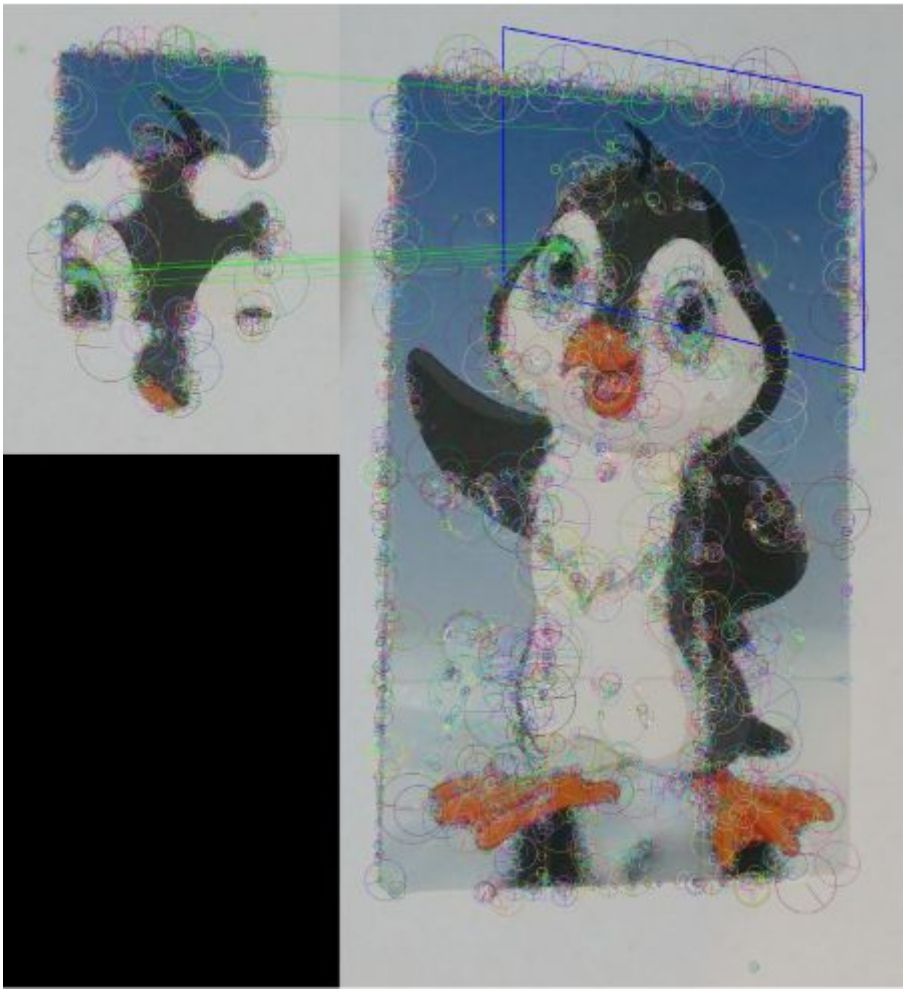
a. SIFT

Detektor i deskryptor cech Scale-Invariant Feature Transform jest to metoda częściowo odporna częściowo na zmiany i przekształcenia na przykład: rotacja, zmiany oświetlenia i skali. Dodatkowo opisuje cechy w celu ich późniejszego dopasowania.

rys. 7.1. Detekcja i deskrypcja za pomocą SIFT

b. SURF

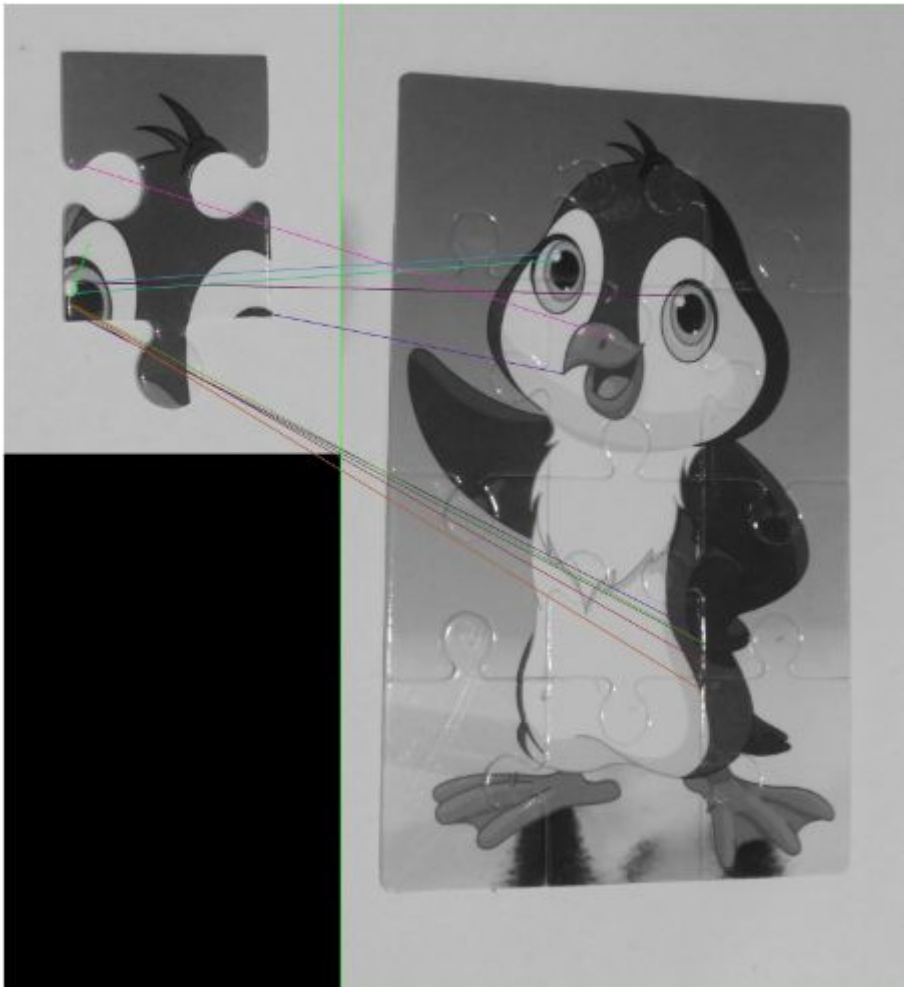
Speeded-Up Robust Features został stworzony w celu zniwelowania największej wady algorytmu SIFT, czyli prędkością wykonywania algorytmu.



rys. 7.2. Detekcja i deskrypcja za pomocą SURF

c. ORB

Oriented FAST, Rotated BRIEF używa detektora FAST do wykrywania cech w wielu skalach oraz deskryptora BRIEF do opisu cech charakterystycznych.



rys. 7.3. Detekcja i deskrypcja za pomocą ORB

d. Wnioski

W wykonanych testach lepsze wyniki prezentował detektor i deskryptor SURF, jednak w końcowej wersji aplikacji został zastosowany SIFT. W celu wykorzystania tego algorytmu została zainstalowana biblioteka opencv_contrib.

8. Najważniejsze metody i fragmenty kodu aplikacji

Odczytanie obrazu z pliku

Pierwszym krokiem umożliwiającym pracę z obrazami, jest odczytanie gotowego obrazu z systemu plików komputera. Użyto tu do tego funkcji z biblioteki OpenCV - imread, gdzie jako parametr tej funkcji podaje się ścieżkę do obrazu, jaki ma zostać odczytany, można podać również samą nazwę pliku, wtedy funkcja będzie obrazu o tej nazwie szukać w folderze, gdzie znajduje się plik z programem zawierającym dane polecenie.

```
filename = 'pies.JPG'
base_image_color = cv2.imread(filename)
```

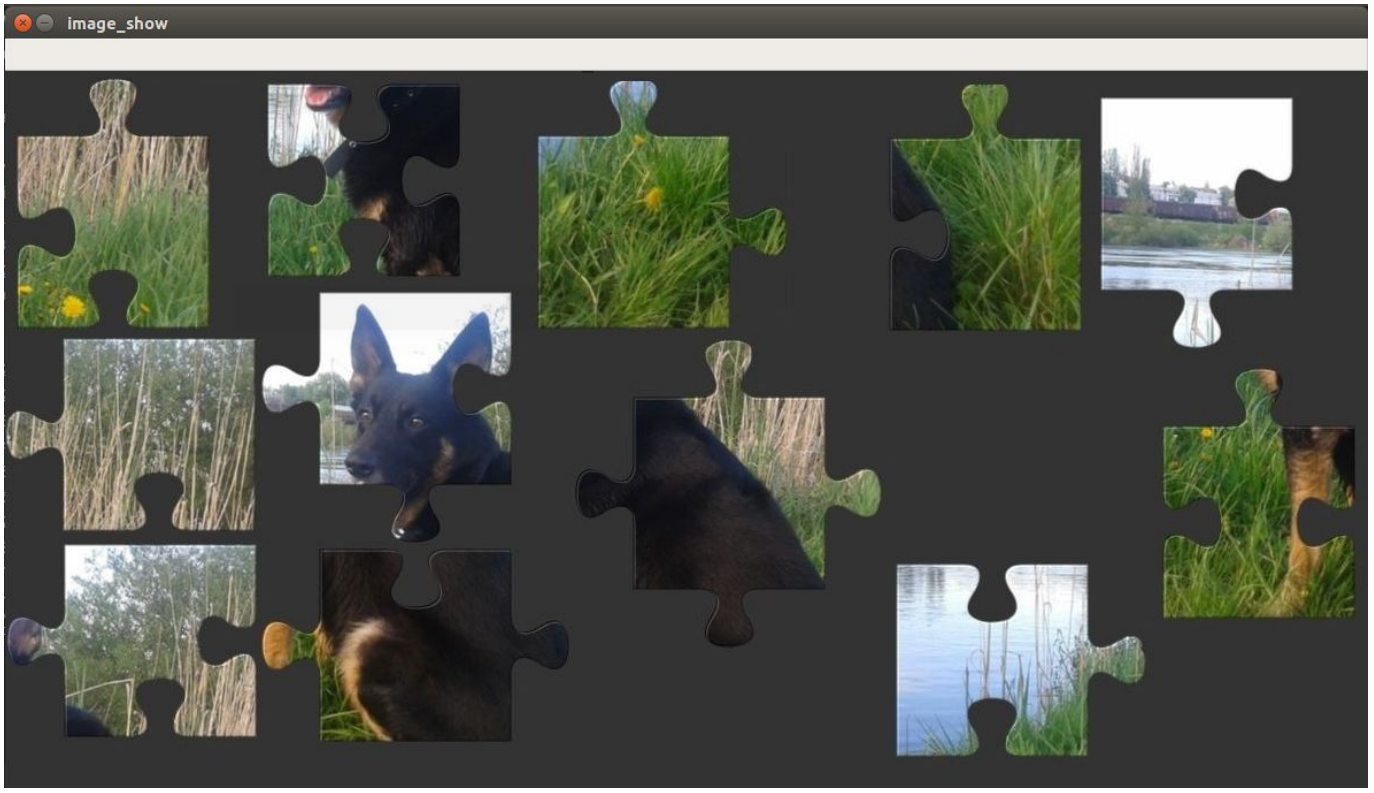
rys. 8.1. Wczytanie obrazu do programu

W celach deweloperskich, stosowany był również fragment kodu poniżej, w celu walidacji, czy obraz został odpowiednio przetworzony. Fragment ten powoduje wyświetlenie obrazu na ekranie oraz wstrzymanie wykonywania programu, dopóki użytkownik nie wciśnie dowolnego klawisza klawiatury.

```
cv2.imshow('image_show', base_image_color)
cv2.waitKey()
```

rys. 8.2. Funkcja wyświetlająca obraz na ekranie

Poniżej można zobaczyć jak wyglądał nasz dopiero co załadowany obraz, wyświetlony na ekranie przez powyższą funkcję.



rys. 8.3. Wyświetlenie obrazu na ekranie za pomocą biblioteki OpenCV

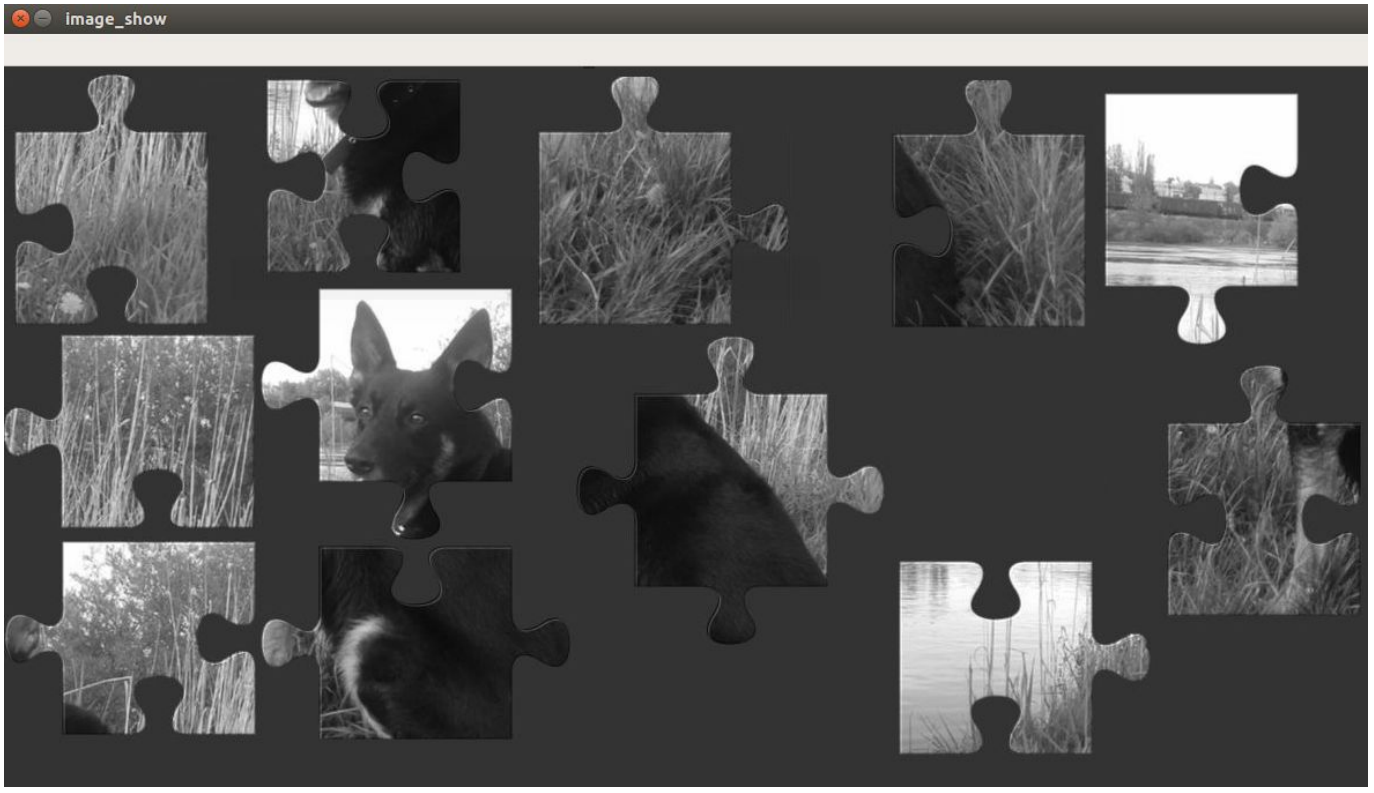
Konwersja na skalę szarości

Do niektórych operacji na obrazach bardziej funkcjonalne są obrazy w skali szarości, czyli w trybie kolorów, gdzie występują jedynie piksele czarne, białe, oraz cała gamma pikseli pośrednich między nimi. Obrazy takie mogą być skuteczniej przetwarzanie przez operacje, takie jak np. progowanie, czy detekcja krawędzi. Na rysunku poniżej przedstawione zostało, jak można za pomocą biblioteki OpenCV przekształcić obraz w taki tryb kolorów.

```
base_image_gray = cv2.cvtColor(base_image_color, cv2.COLOR_BGR2GRAY)
```

rys. 8.4. Funkcja przekształcająca obraz kolorowy na skalę szarości

Poniżej pokazane jest, jak wygląda obraz po takim przekształceniu.



rys. 8.5. Obraz po przekształceniu w skalę szarości

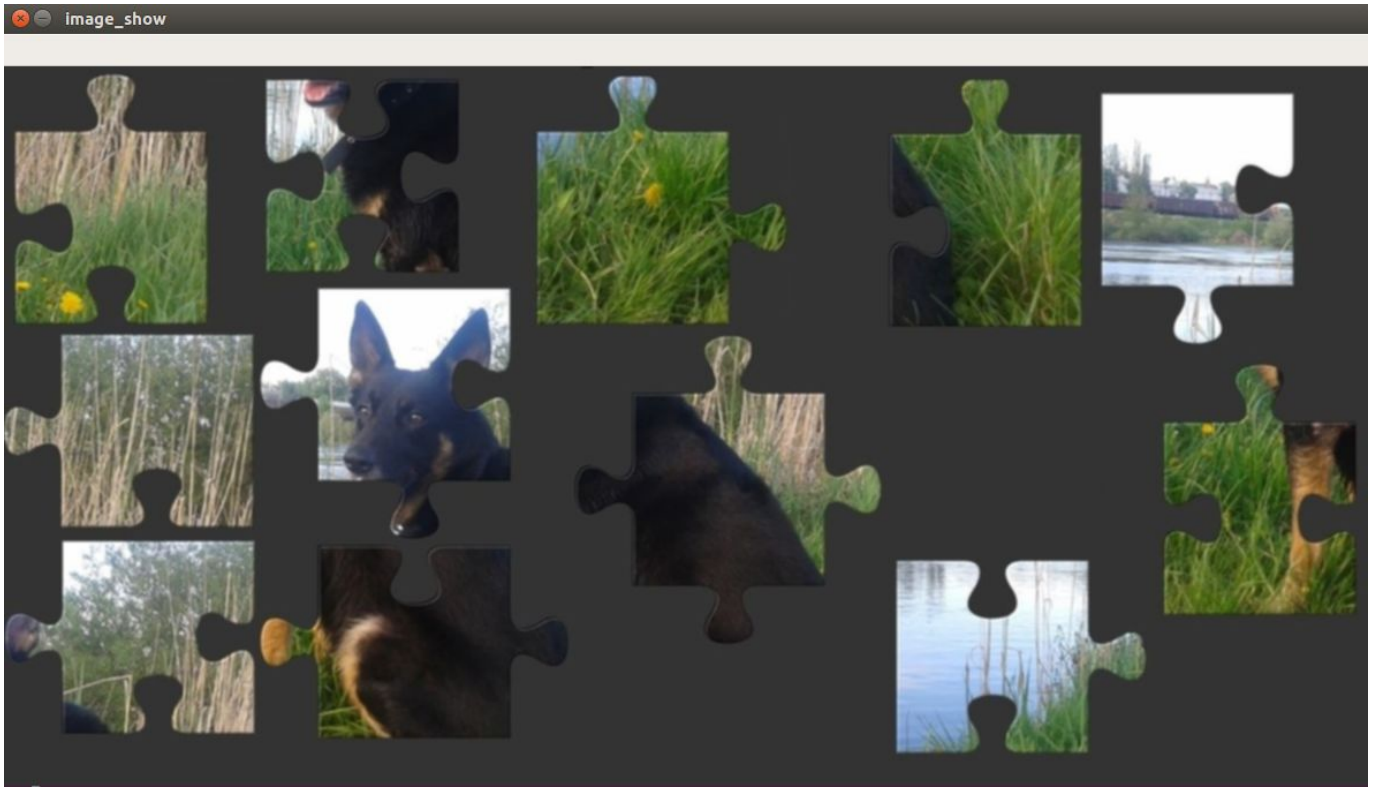
Rozmycie gaussowskie

Jedną z przydatnych operacji podczas działania na obrazach, jest rozmycie obrazu. Rozmycie pozwala na zmniejszenie szumów i zakłóceń w obrazie lub w celu zamazania detali. Poniżej przedstawiony został fragment kodu z projektu, który wykonuje rozmycie gaussowskie na obrazie.

```
blured_img = cv2.GaussianBlur(base_image_gray, (kernel_size, kernel_size), 1.5)
```

rys. 8.6. Funkcja stosująca rozmycie gaussowskie na obraz

Rysunek 8.7 Przedstawiony poniżej pokazuje przykładowy wynik po użyciu rozmycia gaussowskiego.



rys. 8.7. Obraz po rozmyciu gaussowskim

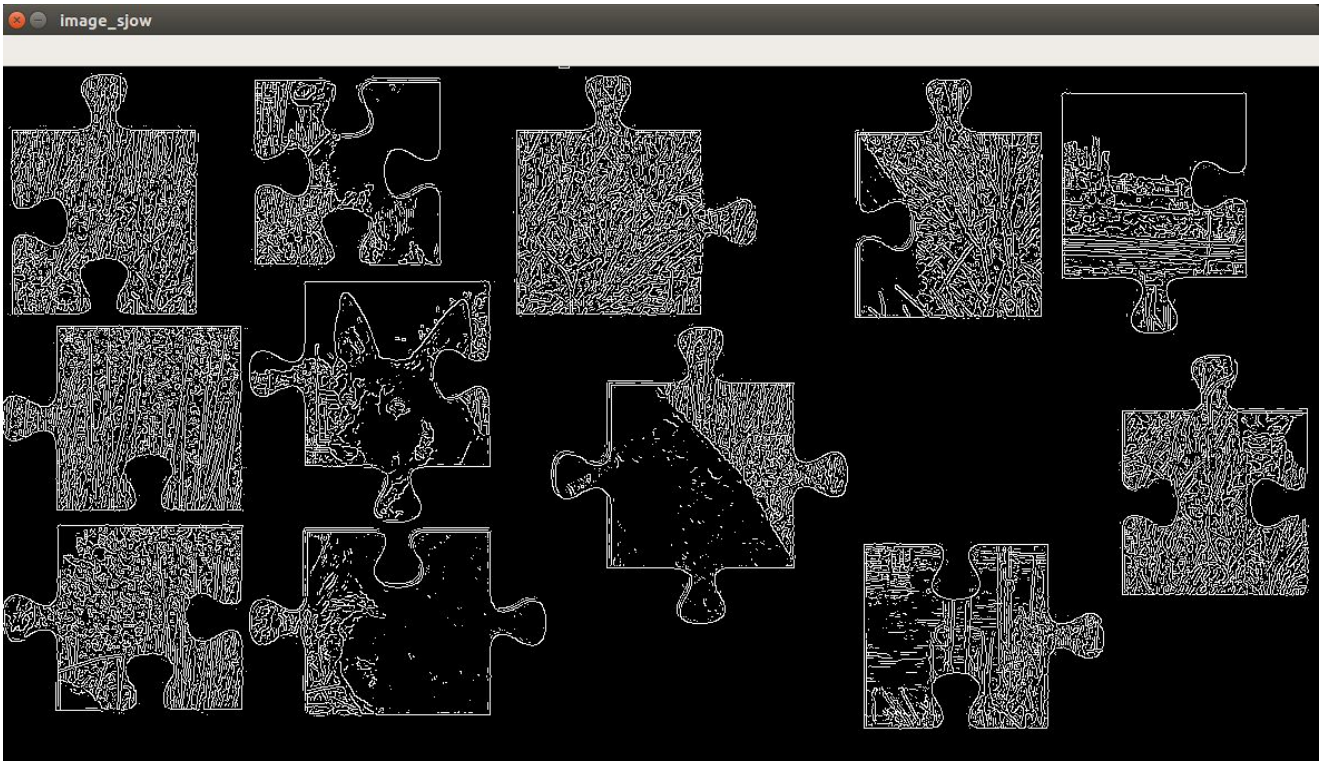
Detekcja krawędzi algorytmem Cannego

W celu detekcji krawędzi puzzli z obrazu w projekcie został użyty algorytm Canny'ego, który wartości do progowania pobiera z trackbarów, których używamy w interfejsie graficznym aplikacji. Aby algorytm nie zwracał nam krawędzi jakiś drobnych zanieczyszczeń z obrazu, to jako parametr funkcji w OpenCV, która przetworzy nam obraz algorytmem Canny'ego, podajemy obraz rozmyty,który został wcześniej poddany rozmyciu gaussowskiemu.

```
canny_img = cv2.Canny(blured_img, threshold1, threshold2)
```

rys. 8.8. Funkcja zwracająca obraz po przetworzeniu algorytmem Canny'ego

Rysunek 8.9 przedstawia przykładowy wynik po zadziałaniu na obraz tą funkcją.



rys. 8.8. Obraz po przetworzeniu algorytmem Canny’ego

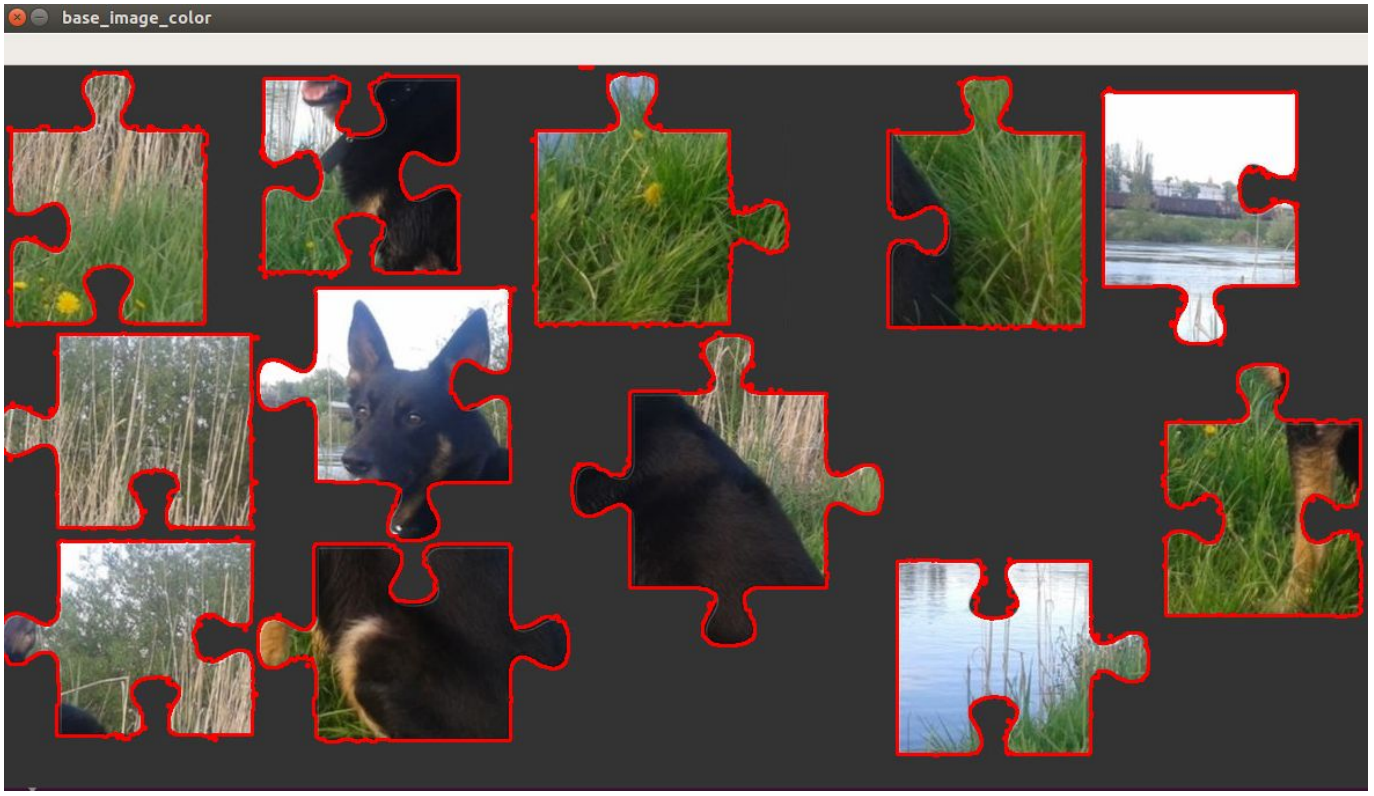
Znajdowanie konturów

Aby odnaleźć współrzędne konturów puzzli, w projekcie została użyta funkcja z OpenCV - findContours, gdzie jako jej parametr, podany został obraz po przetworzeniu algorytmem Canny’ego po dodatkowym rozmyciu.

```
image_contours, contours, hierarchy = cv2.findContours(blur_image(canny_img), cv2.RETR_EXTERNAL,
                                                       cv2.CHAIN_APPROX_SIMPLE)
```

rys. 8.9. Funkcja znajdowania konturów

Rysunek 8.10 przedstawia, jak prezentuje się obraz, po nałożeniu konturów uzyskanych przez powyższą funkcję(Rysunek 8.9) na obraz.



rys. 8.10. Obraz po nałożeniu na niego konturów uzyskanych dzięki funkcji cv2.findContours

Wycinanie pojedynczych puzzli

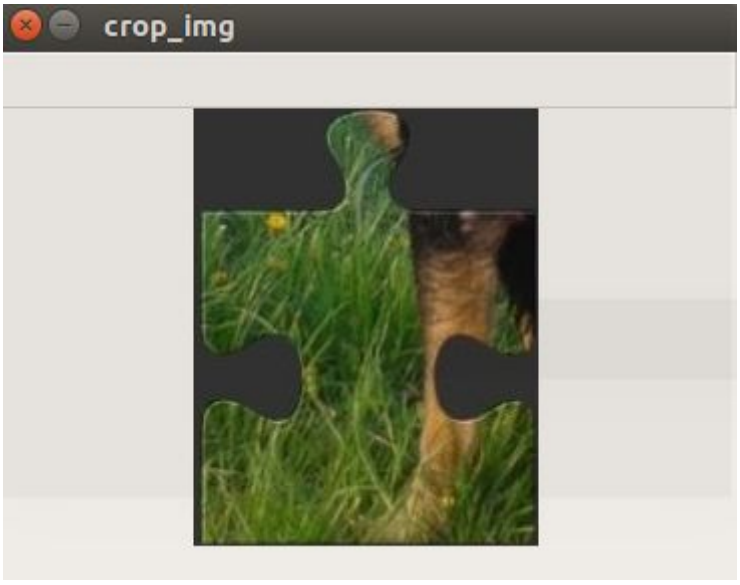
Rysunek 8.11 pokazuje fragment kodu, który wycina jeden konkretny puzzel z obrazu. Za pomocą funkcji cv2.minAreaRect, do której jako parametr podajemy kontur pojedynczego puzzla, otrzymujemy prostokąt, który “otacza” wszystkie podane mu kontury.
Następnie przetwarzamy powstały prostokąt, na odpowiadające mu współrzędne na obrazie(2 i 3 linia kodu z rysunku). Następnie współrzędne te zostają ustawione w odpowiedniej kolejności, oraz obraz zostaje wycinany(ostatnia linia kodu na obrazie, działanie na macierzy).

```
rect = cv2.minAreaRect(contour)
box = cv2.boxPoints(rect)
box = np.int0(box)
x_values = [box[0][0], box[1][0], box[2][0], box[3][0]]
y_values = [box[0][1], box[1][1], box[2][1], box[3][1]]
x1 = min(x_values)
x2 = max(x_values)
y1 = min(y_values)
y2 = max(y_values)
crop_img = image[y1:y2, x1:x2]
```

rys. 8.11. Kod powodujący wycięcie pojedynczego puzzla

Rysunek 8.12 oraz 8.13 pokazują 2 przykładowe puzzle, po wycięciu ich z obrazu za pomocą kodu z rysunku 8.11.

rys. 8.12. Pojedynczy puzzel po wycięciu



rys. 8.13. Pojedynczy puzzel po wycięciu

Do próby ułożenia całego obrazu z pojedynczych puzzli, przy jednoczesnym posiadaniu obrazu puzzli tylko nieposkładanych został stworzony algorytm, który poprzez detekcję krawędzi oraz ich analizę próbuje odpowiednio je ułożyć, poniżej są opisane dwie główne fazy realizacji tego założenia - wykrywanie wypustek puzzli oraz ich układanie.

Wykrywanie wypustek puzzli

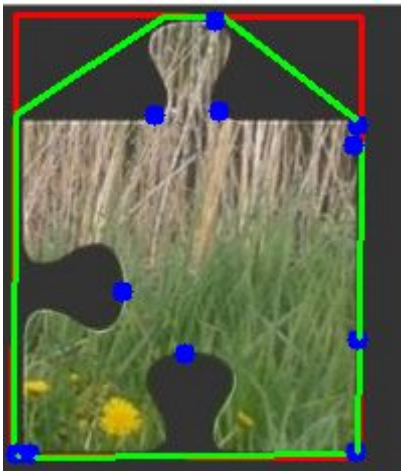
Do wykrycia wypustek puzzla, najpierw program korzysta z określenia krzywych oraz defektów otaczających puzzla. Późniejsza analiza krzywych nie bierze pod uwagę tych, których długość jest mniejsza niż 10% długości badanego puzzla. Wykrycie krzywych oraz defektów jest realizowane za pomocą funkcji z biblioteki OpenCV, przedstawionych na rysunku poniżej(rys. 8.14)

```
hull = cv2.convexHull(contour, returnPoints=False)

defects = cv2.convexityDefects(contour, hull)
```

rys. 8.14. Pojedynczy puzzel po wycięciu

Przykładowy efekt można zobaczyć na obrazie poniżej, gdzie krzywe są narysowane kolorem zielonym, a defekty kolorem niebieskim.



rys. 8.15. Pojedynczy puzzel po naniesieniu wykrytych krzywych i defektów

Wykrywanie wklęsłych oraz prostych ścian puzzla

Gdy wykryte wcześniej krzywe na danej ścianie puzzla są nachylone pod względem osi OX lub osi OY pod kątem mniejszym niż 10°, to wtedy dana ściana puzzla jest prosta lub wklęsła. Żeby określić, która konkretnie, program analizuje opisane wyżej defekty(niebieskie punkty na rys. 8.15). Gdy jakiś defekt leży w większym oddaleniu od krzywej, to wtedy badana ściana jest wklęsła, w przeciwnym wypadku ściana jest prosta. Opisany algorytm jest realizowany za pomocą kodu pokazanego na rysunku poniżej(rys. 8.16)

```
def is_on_start_and_end_line(self, threshold):
    degree, distance = self.define_slope_angle_and_distance(threshold=threshold)
    # horizontally
    if (0 <= degree <= 10) or (170 <= degree <= 190) or (350 <= degree <= 360):
        # is straight
        if (abs(self.start[1] - self.far[1]) < threshold) or (abs(self.end[1] - self.far[1]) < threshold):
            if self.start[0] > self.end[0]:
                return "straight_top"
            else:
                return "straight_down"
        else:
            # is concave
            if self.far[1] < self.end[1] and self.far[1] < self.start[1]:
                return "down_concave"
            elif self.far[1] > self.end[1] and self.far[1] > self.start[1]:
                return "top_concave"
            else:
                return "IDK"
    # perpendicularly
    else:
        # is straight
        if (abs(self.start[0] - self.far[0]) < threshold) or (abs(self.end[0] - self.far[0]) < threshold):
            if self.start[1] > self.end[1]:
                return "straight_right"
            else:
                return "straight_left"
        else:
            # is concave
            if self.far[0] < self.end[0] and self.far[0] < self.start[0]:
                return "right_concave"
            elif self.far[0] > self.end[0] and self.far[0] > self.start[0]:
                return "left_concave"
```

rys. 8.16. Kod realizujący wykrywanie płaskich i wklęsłych ścian puzzla

Wykrywanie wypukłych ścian puzzla

Gdy wykryte wcześniej krzywe na danej ścianie puzzla są nachylone pod kątem większym niż 10° w odniesieniu do osi OX oraz osi OY, to sprawdzane jest, jak są nachylone pod względem osi OX, z uwzględnieniem kierunku, po którym idzie krzywa(zawsze idzie odwrotnie do wskazówek zegara, przyjmując środek w środku puzzla). Poniższy rysunek(rys. 8.16) pokazuje, jak ustawiana jest flaga dla pojedynczej krzywej, gdy spełni opisane powyżej warunki.

```
if 0 < degree < 90:
    down_or_right = True
if 90 < degree < 180:
    top_or_right = True
if 180 < degree < 270:
    top_or_left = True
if 270 < degree < 360:
    down_or_left = True
```

rys. 8.17.Określenie flagi nachylenia krzywej względem puzzla.

Po ustaleniu flag, badane są wszystkie krzywe, para flag jest prawdziwa, to puzzel wypukły jest tam, gdzie obie te flagi wskazują. Dodatkowo, gdy wszystkie flagi będą prawdziwe, to wtedy istnieje możliwość, że puzzel ma 4 lub 3 wypukłe ściany, albo 2 wypukłe ściany po przeciwnych stronach. Wtedy algorytm dodatkowo analizuje, czy inne ściany nie zostały już wcześniej określone(wypukłość jest badana na końcu). Powyższy algorytm jest przedstawiony na poniższym fragmencie kodu(rys. 8.18)

```
# end of loop, define convex
if down_or_left and down_or_right and top_or_right and top_or_left:
    if down_or_right and down_or_left and self.down_tab is None:
        self.down_tab = "convex"
    if down_or_right and top_or_right and self.right_tab is None:
        self.right_tab = "convex"
    if top_or_right and top_or_left and self.top_tab is None:
        self.top_tab = "convex"
    if down_or_left and top_or_left and self.left_tab is None:
        self.left_tab = "convex"
else:
    if down_or_right and down_or_left:
        self.down_tab = "convex"
    if down_or_right and top_or_right:
        self.right_tab = "convex"
    if top_or_right and top_or_left:
        self.top_tab = "convex"
    if down_or_left and top_or_left:
        self.left_tab = "convex"
```

rys. 8.18. Określenie wypukłej ściany puzzla

Układanie puzzli

Po określeniu kształtu każdego puzzla, do ich ułożenia program inicjalizuje tablicę dwuwymiarową, w której w każdym pojedynczym polu, będzie umieszczony jeden puzzel. Następnie wyznaczane są narożniki puzzli(mają po dwie proste ściany) i umieszczane w odpowiednich polach. Operacje te są pokazane na rysunku poniżej(rys. 8.19)

```
# create array of size of puzzles(one array value is one puzzle)
final_puzzles = [[0 for x in range(final_puzzles_width)] for y in range(final_puzzles_height)]

for corner in corners_puzzles:
    if corner.top_tab == "straight":
        if corner.right_tab == "straight":
            final_puzzles[0][final_puzzles_width - 1] = corner
        else:
            final_puzzles[0][0] = corner
    else:
        if corner.right_tab == "straight":
            final_puzzles[final_puzzles_height - 1][final_puzzles_width - 1] = corner
        else:
            final_puzzles[final_puzzles_height - 1][0] = corner
```

rys. 8.19. Ułożenie narożników w macierzy

Po ułożeniu narożników w macierzy, algorytm w pętli iteruje po każdym elemencie macierzy, próbując dopasować w dane miejsce puzzla, uwzględniając otoczenie(granice macierzy, oraz puzzle obok). Gdy w dane miejsce pasuje więcej niż jeden puzzel, to program wybiera pierwszy możliwy wariant, a pozostałe zapisuje w tablicy. Gdy w przyszłości dojdzie do sytuacji, gdzie w pozostałe miejsca nie pasują żadne pozostałe puzzle, to program zabiera z tablicy jedną z zapisanych możliwości, i ją kontynuuje. Na rysunku poniżej(rys. 8.20), przedstawiona jest operacja wypełniania tablicy, oraz obsługi konfliktów. Funkcja complete_field wypełnia jedno pole w macierzy z ułożonymi puzzlami, oraz zwraca pozostałe możliwości, jeśli istnieją, lub

informacje o błędzie dopasowania, która powoduje rozpoczęcie kolejnej możliwości ułożenia puzzli

```
additional_final_puzzles = []
x = 0
y = 0
while x < final_puzzles_width:
    while y < final_puzzles_height:
        if final_puzzles[y][x] == 0:
            additional_final = complete_field(final_puzzles, unfitted_puzzles, x, y, final_puzzles_width,
                                              final_puzzles_height)
            if additional_final is not None and additional_final != "ERROR":
                additional_final_puzzles.append(additional_final)

            if additional_final == "ERROR":
                additional_tuple = additional_final_puzzles.pop()

                final_puzzles = additional_tuple[0]
                x = additional_tuple[1]
                y = additional_tuple[2]
        y += 1
    x += 1
    y = 0
```

rys. 8.20. Układanie puzzli w macierzy

Po ułożeniu puzzli trzeba je odpowiednio ułożyć na obrazie, do tego celu program wykorzystuje bibliotekę PIL. Program iteruje po każdym puzzlu w macierzy i umieszcza go odpowiednio na puzzlu. Początkowy obraz jest większy od rzeczywistości, i jest on ucinany po ostatecznym ułożeniu puzzli. Dane operacje są przedstawione na rysunku poniżej(rys. 8.21)

```
image = Image.new("RGBA", (width + 500, height))
actual_height = 0
actual_width = 0
max_width = 0
for y in range(len(final_puzzles)):
    y_values = []
    for x in range(len(final_puzzles[y])):
        temp_image = Image.fromarray(final_puzzles[y][x].crop_image)
        image.paste(temp_image, (actual_width, actual_height))
        width, height = temp_image.size
        actual_width += width

        y_values.append(height)
    if actual_width > max_width:
        max_width = actual_width
    actual_width = 0
    actual_height += max(y_values)

open_cv_image = np.array(image)
trimmed_image = open_cv_image[0:actual_height, 0:max_width]
```

rys. 8.21. Układanie puzzli na obrazie

Przykładowy wynik ułożonych puzzli na obrazie jest przedstawiony na rysunku poniżej (rys. 8.22).

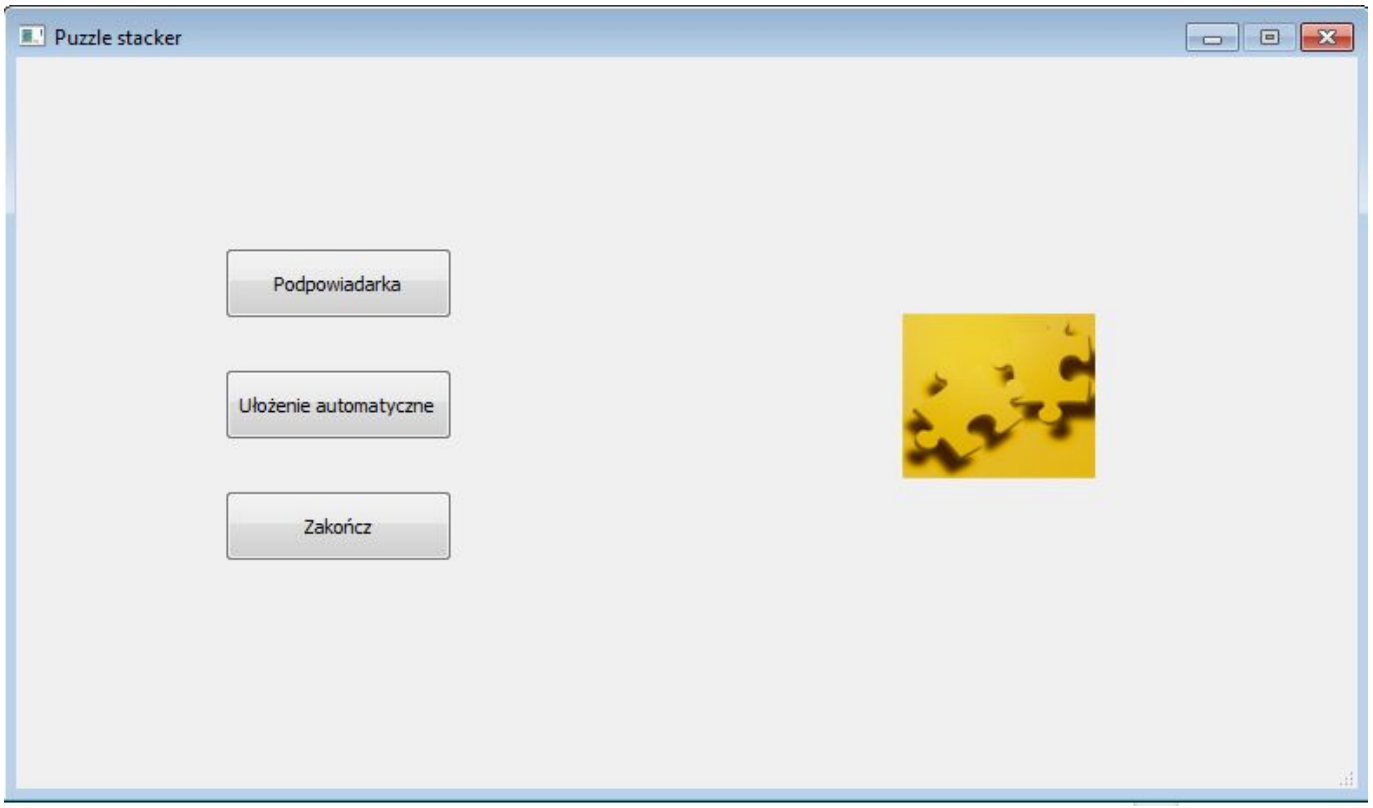
rys. 8.21. Ułożone automatycznie puzzle

9. Instrukcje użytkowania

W rozdziale zostanie przedstawiona krok po kroku instalacja aplikacji oraz przykładowe wykorzystanie programu. Do użytkowania aplikacji wymagany jest Python w wersji 3, mysz komputerowa oraz podstawowa znajomość wiersza poleceń. Instrukcja pokazuje typowe użytkowanie aplikacji.
Kolejne kroki instalacji aplikacji:

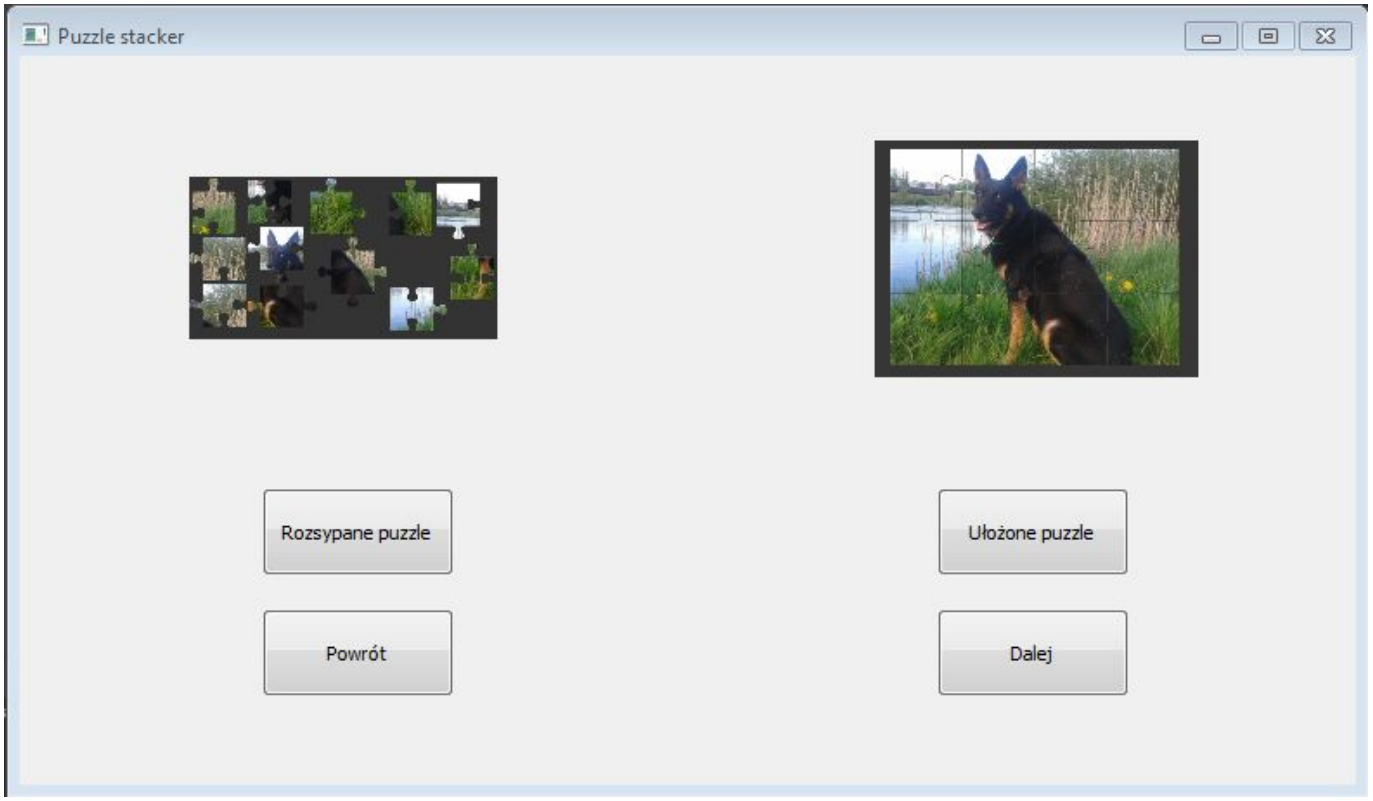
- pobranie programu z repozytorium
  git clone https://github.com/NormanLecter/puzzle-stacker.git
- instalacja wymaganych bibliotek
  pip install -r requirements.txt
- uruchomienie programu
  python app.py

Po wykonaniu wszystkich komend użytkownikowi wyświetlany jest ekran główny aplikacji. Z tego miejsca użytkownik ma możliwość wybrania dwóch trybów: podpowiedzi dotyczące umiejscowienia puzzla oraz automatycznego ułożenia obrazu przez program.



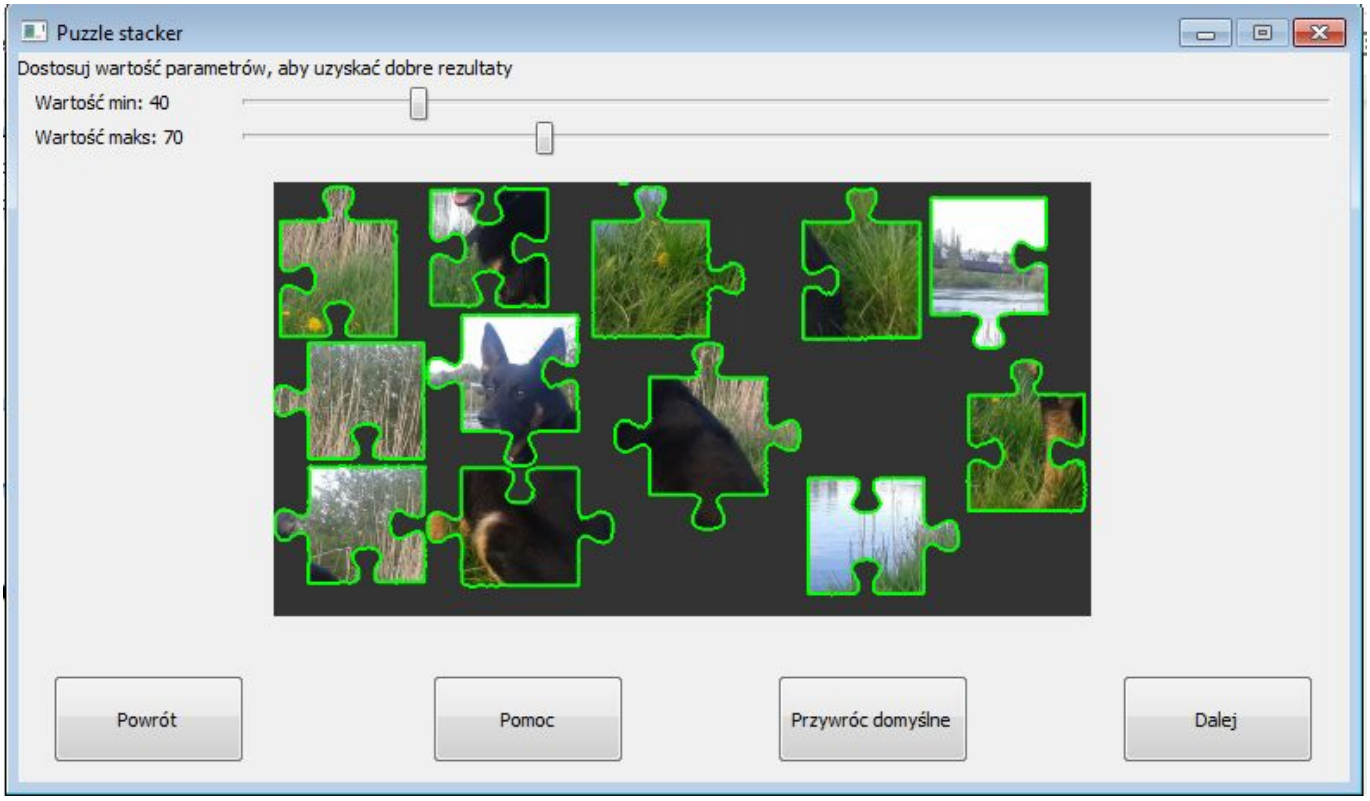
rys. 9.1. Ekran główny aplikacji

Po wybraniu przez użytkownika odpowiedniego trybu pracy, wyświetlany jest ekran wyboru plików (rysunek 9.2. dla opcji podpowiadarki wymagane dwa pliki, dla układania automatycznego jeden plik). Użytkownik wskazuje pliki graficzne poprzez proste okno (wyświetlane są wyłącznie foldery oraz pliki w formacie PNG i JPG).



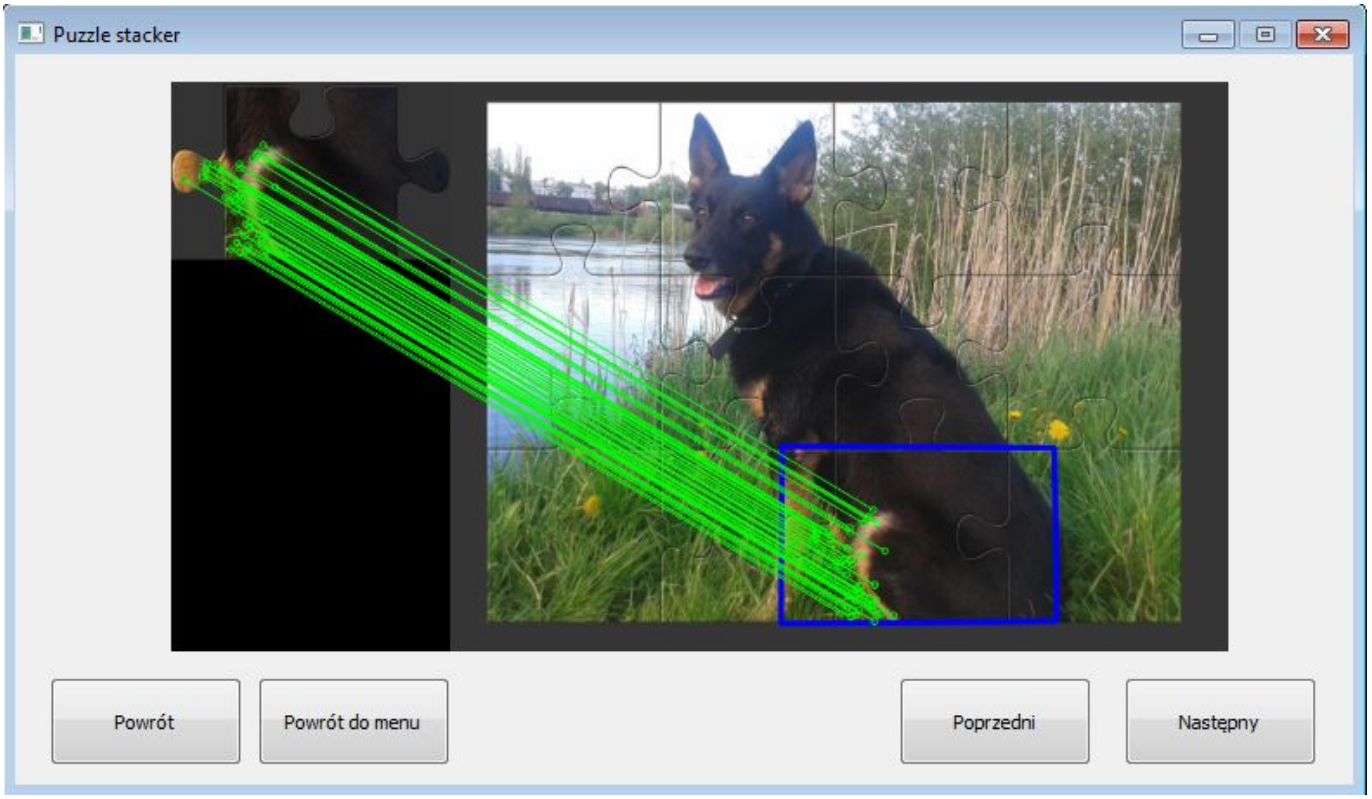
rys. 9.2. Ekran wczytania obrazów

Po wczytaniu wymaganych plików oraz kliknięciu na przycisk dalej użytkownik może dostosować parametry do jakości zdjęcia oraz tła.



rys. 9.3. Ekran dostosowania parametrów

Po dostosowaniu parametrów oraz przejściu do następnego widoku, użytkownik aplikacji może przeglądać otrzymane wyniki.



rys. 9.4. Ekran przeglądania wyników

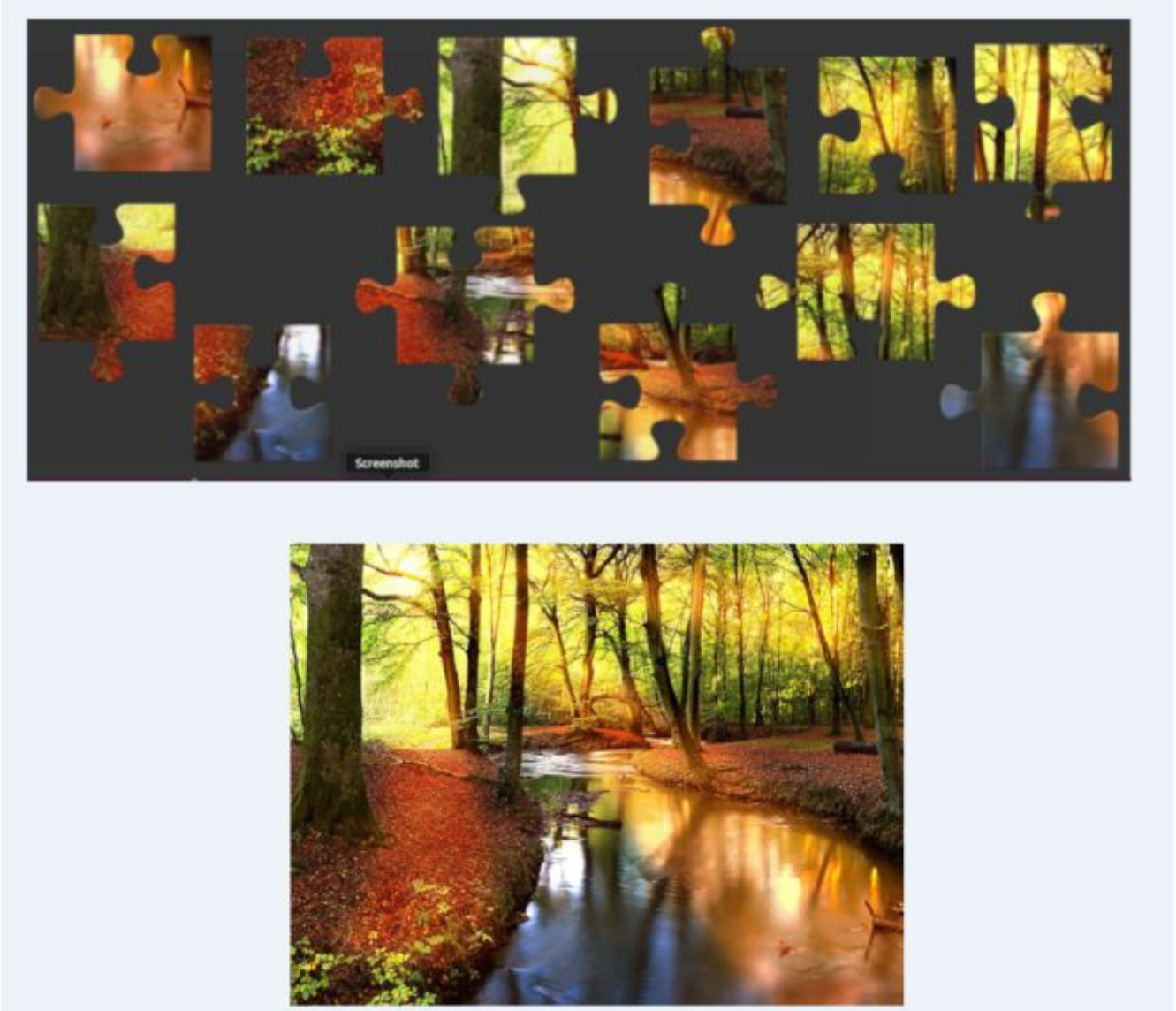
Podobny przebieg użytkowania aplikacji jest dla drugiej funkcji programu (automatycznego układania) z kilkoma różnicami:

- ekran wczytania obrazów (rysunek 9.2.) umożliwia wczytanie wyłącznie jednej grafiki,
- ekran przeglądania wyników (rysunek 9.4.) wyświetla jeden wynik (ułożony obraz).

10. Testy aplikacji

Rozdział prezentuje testy obu funkcji programu, do których wykorzystano obrazy pobrane ze strony http://www.puzzleonline.eu oraz zdjęcia wykonane przez autorów aplikacji.

Rysunek 10.1. prezentuje puzzle pobrane ze strony internetowej, które zostały wycięte przez program komputerowy.



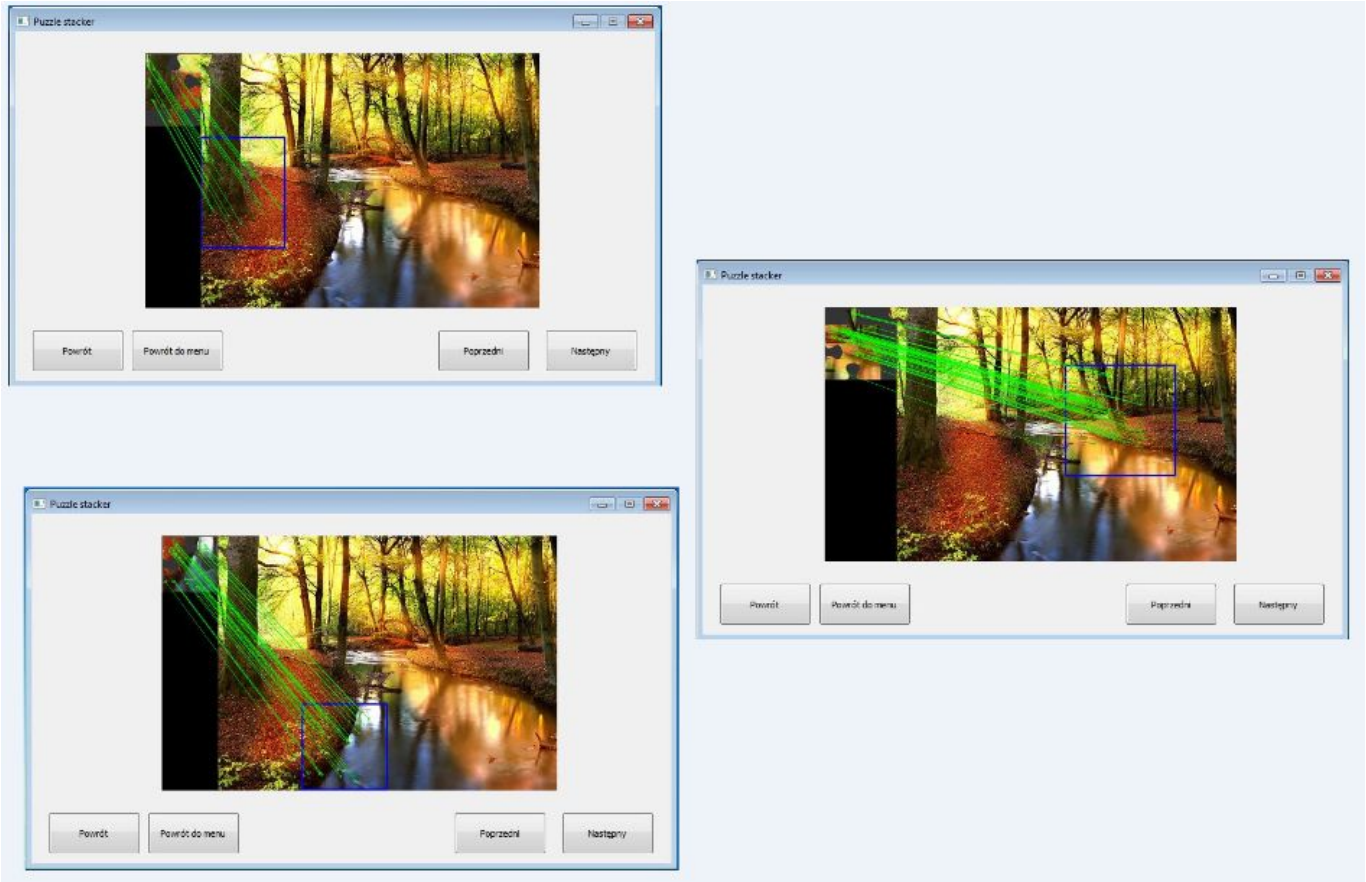
rys. 10.1. Puzzle pobrane ze strony http://www.puzzleonline.eu oraz ułożony obraz

Rysunek 10.2. prezentuje zdjęcie puzzli wykonanych przez autorów aplikacji na zielonym tle.

rys. 10.2. Zdjęcia puzzli oraz ułożonego obrazu przez autorów aplikacji

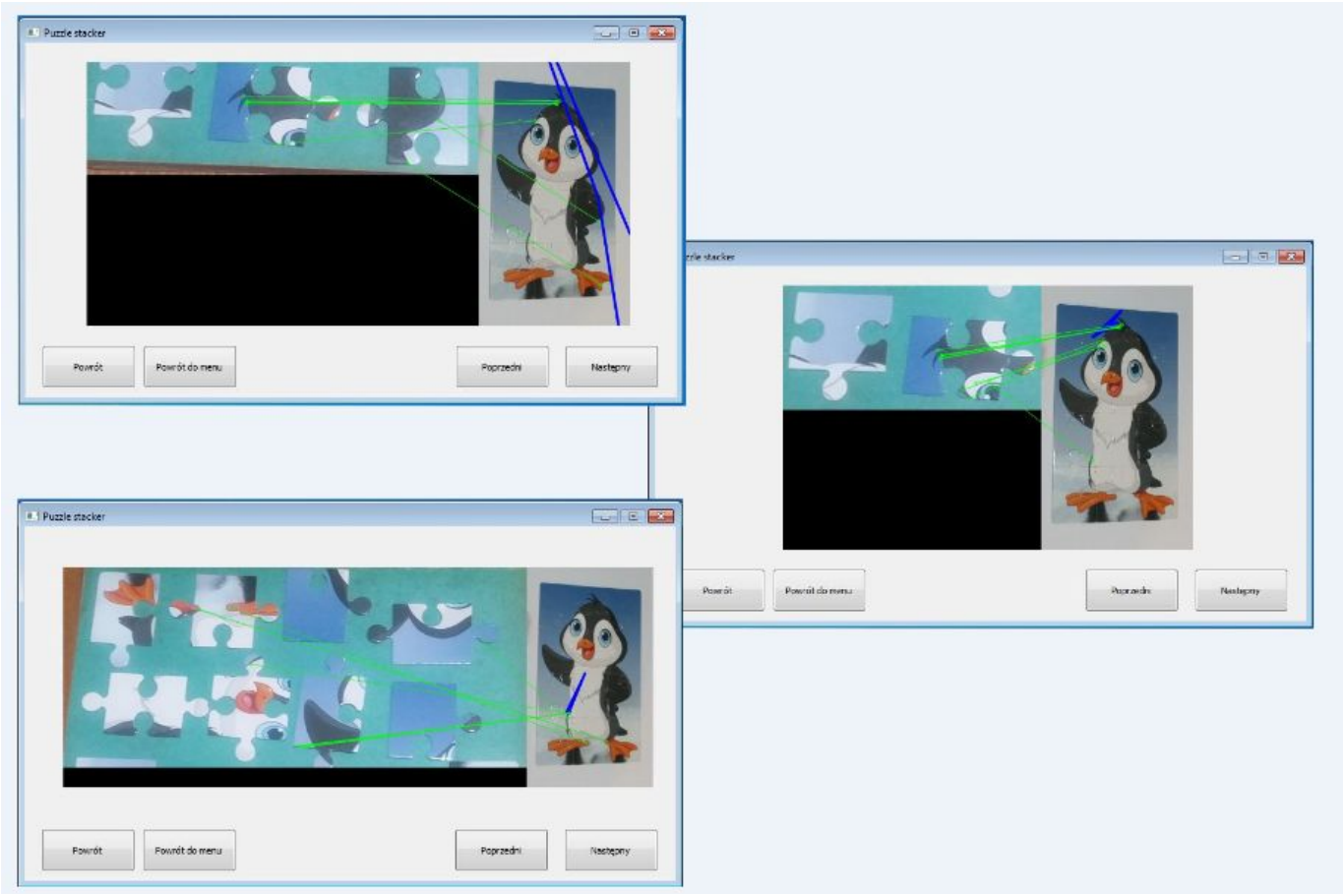
a. Testy podpowiadarki

Pierwszym wykonanym testem był test przy użyciu danych ze strony internetowej. Udało się odnaleźć 10 puzzli w tym dwa zostały źle wycięte (jednak mimo to, zostały poprawnie wskazane).



rys. 10.3. Wyniki podpowiadarki z użyciem danych z Internetu.

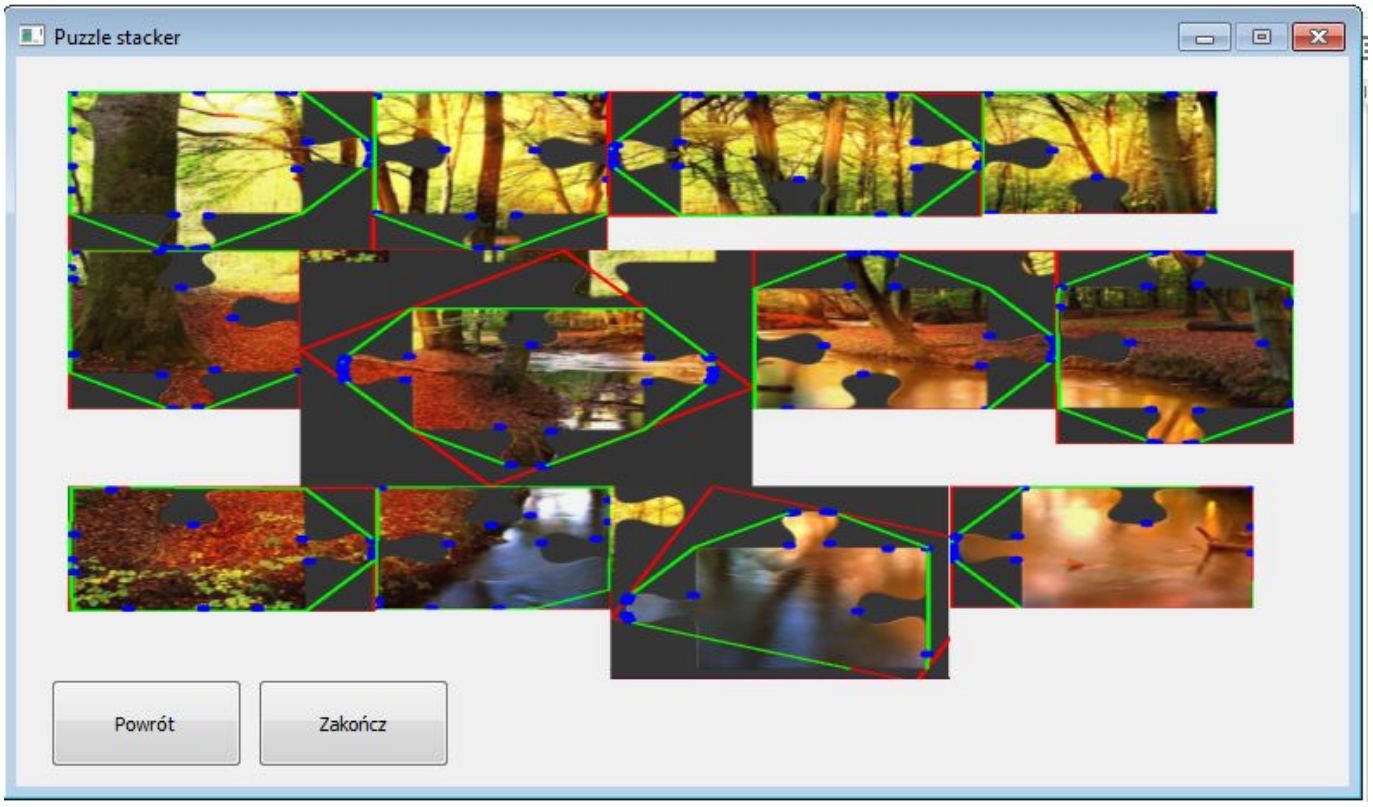
Drugi test został przeprowadzony ze zdjęciami wykonanymi za pomocą aparatu telefonicznego. Program wyświetlił trzy rezultaty, jednak puzzli zostało wskazanych więcej. Było to prawdopodobnie spowodowane słabym oświetleniem oraz kiepską jakością zdjęcia.



rys. 10.4. Wyniki dla zdjęć puzzli wykonanych aparatem.

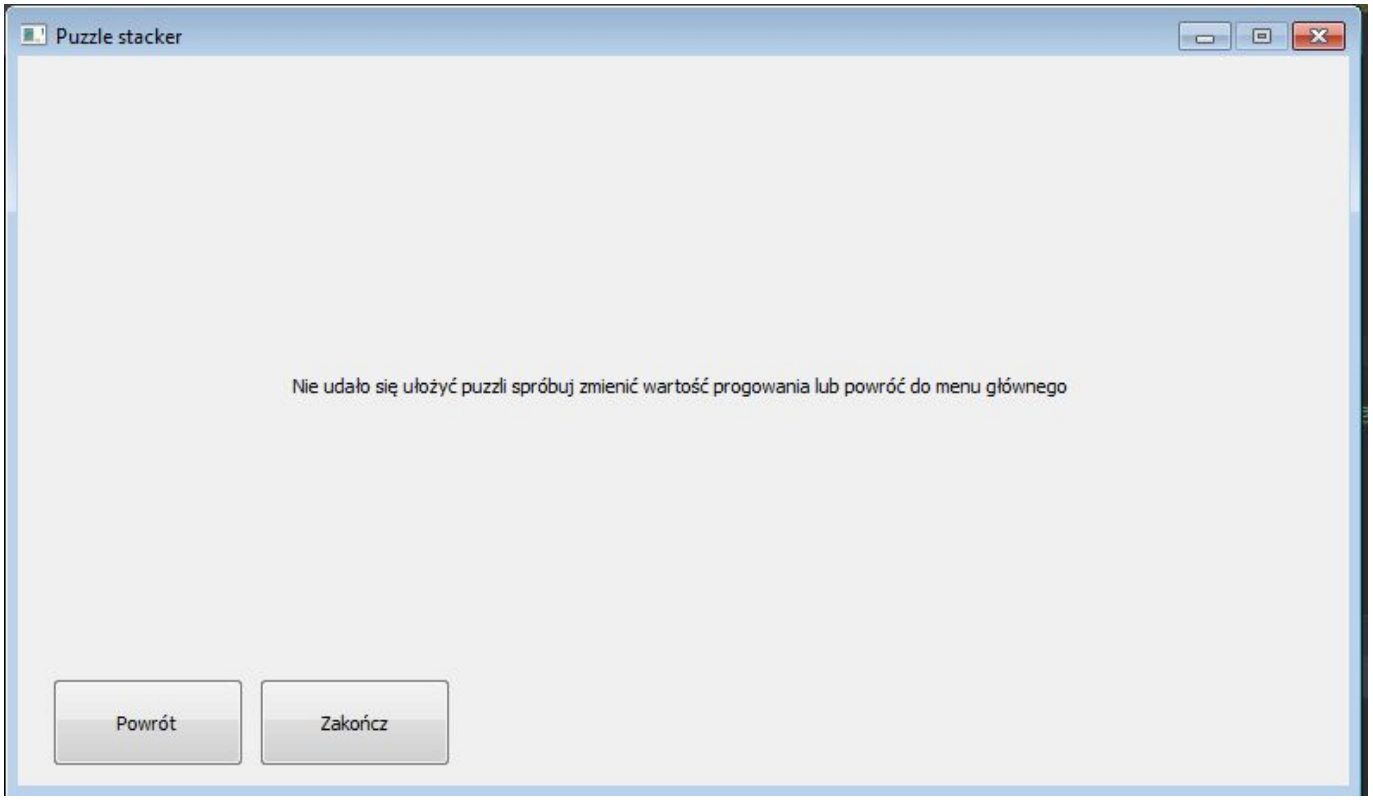
b. Testy automatycznego układania

Pierwszym wykonanym testem podobnie jak w podrozdziale 10.a. było sprawdzenie wyników dla puzzli pobranych z witryny internetowej. Po załadowaniu pliku do programu, aplikacji udało się zwrócić poprawny wynik (rysunek 10.5.).



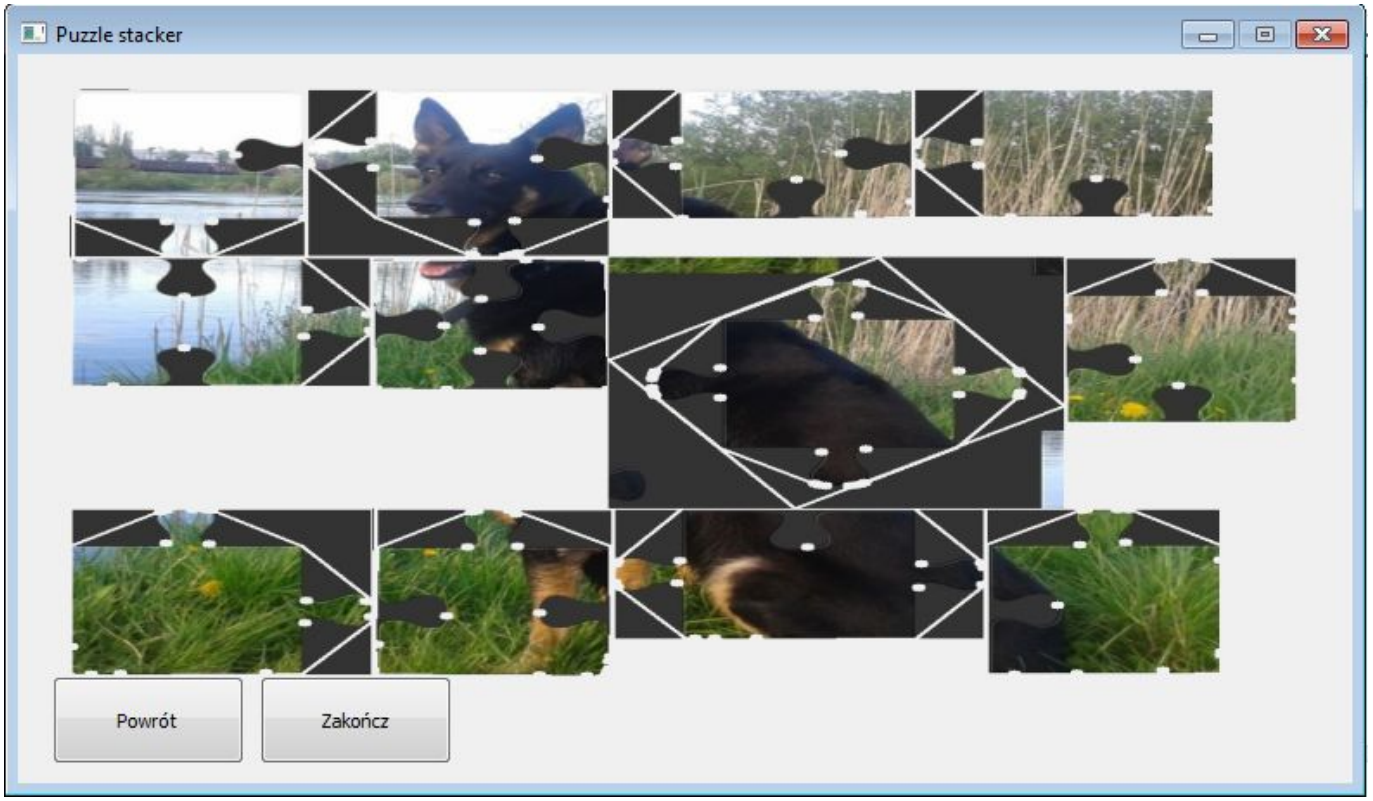
rys. 10.5. Ułożony rysunek dla puzzli wyciętych komputerowo.

Kolejny test przeprowadzony został na zdjęciach wykonanych aparatem komórkowym. Po wczytaniu obrazu oraz ustaleniu odpowiedniego progowania nie udało się ułożyć obrazu przez aplikację.



rys. 10.6. Wynik zwrócony dla danych wykonanych telefonem.

Po nieudanej próbie postanowiono wykonać testy dla kolejnego obrazu pobranego z Internetu. Wynik był zadowalający co pokazuje rysunek 10.7.



rys. 10.7. Wynik dodatkowego testu dla danych pobranych ze strony internetowej.

c. Wnioski

Program działa zadowalająco dla zdjęć pobranych z Internetu (zarówno automatycznie układa oraz wskazuje podpowiedzi dotyczące prawidłowego umiejscowienia puzzla). Natomiast dla zdjęć robionych przy użyciu aparatu program nie może ułożyć całego obrazu oraz wyodrębnić wszystkich puzzli. Rozwiązaniem tego problemu przez użytkownika mogłoby być robienie zdjęcia pojedynczym puzzlom (program musiałby wyłącznie oddzielić puzzel od tła). Jednak ten pomysł nie umożliwiłby ułożenie całego obrazu (twórcy aplikacji mogliby dodać funkcję wczytywania i zapamiętywania kilku zdjęć puzzli).

11. Podsumowanie

W trakcie powstawania udało się zrealizować wszystkie zamierzone cele oraz dodano dodatkową funkcjonalność tj. automatyczne układanie puzzli.
W trakcie realizacji napotkano kilka problemów związanych między innymi z ucinaniem tekstu przycisków w systemie Linux. Kolejnym problemem było debugowanie błędów podczas integracji interfejsu (w tym przypisywanie funkcji do przycisków, przejścia między widokami) z funkcjami wykorzystującymi bibliotekę OpenCV. Zamiast wyświetlać w konsoli potencjalne miejsce w którym znajduje się problematyczny fragment kodu wyłącznie, pojawiał się komunikat “Python.exe przestał działać”. Poradzono sobie z tym problemem poprzez dodanie odpowiedniej funkcji do programu (rysunek 11.1).

```
def catch_exceptions(t, val, tb):
    QMessageBox.critical(None,
                                    "An exception was raised",
                                    "Exception type: {}".format(t))
    old_hook(t, val, tb)


old_hook = sys.excepthook
sys.excepthook = catch_exceptions
```

rys. 11.1. Funkcja oraz zmienne umożliwiające “łapanie” błędów.

Możliwe perspektywy rozwoju aplikacji:

- skalowanie aplikacji (napisy na przyciskach),
- umożliwienie wczytanie kilku zdjęć.

Tabela 11.1 Podział prac

Łukasz Wolniak	• stworzenie interfejsu graficznego przy użyciu QT Designer • detekcja puzzli na zdjęciu z ułożonymi puzzlami (zastosowanie detektora i deskryptora cech)
Szymon Zieliński	• znajdowanie konturów (progowanie, odszumianie, filtracja obrazu) • automatyczne układanie puzzli
Dominik Krystkowiak	• zapewnienie funkcjonalności interfejsu graficznego (współpraca między widokami) • integracja OpenCV z PyQT (dostosowanie funkcji do interfejsu)