

POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY



PODSTAWY TELEINFORMATYKI

ROZPOZNAWANIE OBRAZU Z GRY W WARCABY

ORAZ WIZUALIZACJA STANU GRY NA KOMPUTERZE

KORNEL KRZEŚLAK 126849

PAULINA MROZEK 126879

KAMIL SAGALARA 126865

HUBERT SPRINGER 126796

PROWADZĄCY:

MGR INŻ. PRZEMYSŁAW WALKOWIAK

Poznań, 2018

Spis treści

1 Charakterystyka ogólna projektu	3
1.1 Uzasadnienie wyboru tematu	3
2 Wymagania	3
2.1 Wymagania funkcjonalne	4
2.2 Wymagania niefunkcjonalne	4
3 Narzędzia, środowisko, biblioteki	5
3.1 Narzędzia	5
3.2 Środowisko	6
4 Diagramy UML	6
4.1 Diagram przypadków użycia	6
5 Architektura systemu.	8
5.1 Rozpoznawanie obrazu	8
5.2 Wizualizacja	10
5.3 Sprawdzanie poprawności ruchu	11
6 Historia rozgrywki	12
6.1 Zapisywanie rozgrywki	13
6.2 Wczytywanie rozgrywki	14
7 Zastosowane metody przetwarzania obrazu	15
7.1 Wykrycie planszy	15
7.2 Wykrycie pionków	19
8 Mockupy	21
9 Instrukcja użytkowania	23
10 Podział prac	27
11 Napotkane problemy	29

1 Charakterystyka ogólna projektu

Celem projektu było stworzenie aplikacji, która bazując na rozpoznawaniu obrazu wspiera graczy podczas rozgrywki gry w warcaby.

Aplikacja skierowana jest do osób, które pasjonują się grami planszowymi, w szczególności warcabami. Główną zaletą aplikacji jest wyeliminowanie oszukiwania, które ma miejsce podczas rozgrywek towarzyskich. Program poinformuje użytkowników, gdy ruch będzie niepoprawny lub też gracz wykona ruch podczas tury przeciwnika. Dzięki temu nie ma konieczności, aby gracz czujnie obserwował planszę i zapamiętywał stan gry. Dzięki modułowi sprawdzania poprawności ruchu, aplikacja może również służyć jako pomoc w rozgrwce dla początkujących graczy. Co powoduje, że jest przeznaczona dla sporej grupy odbiorców, niezależnie od poziomu umiejętności graczy.

1.1 Uzasadnienie wyboru tematu

Główna motywacją wyboru tematu było zamiłowanie autorów do gier planszowych, którzy w wolnych chwilach grają między innymi w warcaby w wersji tradycyjnej jak i w wersji online na platformie Kurnik. Kolejnym powodem była możliwość zapoznania się z tematem rozpoznawania obrazów, który współcześnie jest szeroko rozwijany staje się coraz częściej wykorzystywany w projektach informatycznych oraz nauka nowego języka programowania - Python. Połączenie tych dwóch elementów sprawiło, że projekt był wykonywany z przyjemnością.

2 Wymagania

Aby osiągnąć zamierzony cel systemu oraz poprawne działanie aplikacji konieczne jest dokładne określenie konkretnych wymagań wobec tworzonego projektu. Wymagania te możemy podzielić na wymagania funkcjonalne i niefunkcjonalne, które zostały przedstawione w poniższym rozdziale.

2.1 Wymagania funkcjonalne

Wymagania funkcjonalne skupiają się na usługach oferowanych przez aplikacje. Opisują poszczególne funkcje (czynności, operacje, usługi) wykonywane przez system.

Do wymagań funkcjonalnych należą poniższe wytyczne:

- Konfiguracja parametrów - wybór zakresu wykrywania barw.
- Możliwość zapisania rozgrywki.
- Możliwość odtworzenia zapisanego stanu gry w trybie krokowym.
- Wyświetlenie listy prawidłowych ruchów dla aktywnego gracza.

2.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują ograniczenia, przy zachowaniu których system powinien realizować swe funkcje.

Do wymagań niefunkcjonalnych należą poniższe wytyczne:

- aplikacja desktopowa przeznaczona dla systemu operacyjnego Windows,
- aplikacja przeznaczona dla dwóch użytkowników,
- aplikacja nie wymaga połączenia z internetem,
- wymagany jest telefon z aplikacją DroidCam,
- komunikacja z użytkownikiem w języku polskim:
 - menu, formularz, komunikaty o błędach, podpowiedzi,
- plansza o rozmiarze 8x8 (64 pola),
- każdy gracz ma 12 pionków,
- kolor pionków o kolorze innym niż czarny lub biały,
- przechowywanie stanu gry jako plansza w formacie JSON.

3 Narzędzia, środowisko, biblioteki

W tym rozdziale zostały przedstawione narzędzia i środowisko wykorzystywane przez autorów. Zostały także przedstawione biblioteki niezbędne do prawidłowego działania aplikacji.

Podczas wyboru technologii i narzędzi wykorzystywanych w projekcie kierowano się głównie ich prostotą i popularnością. Istotnym kryterium wyboru było również to, aby narzędzia były darmowe oraz żeby posiadały obszerną dokumentację, która pozwoli na szybkie wyszukiwanie potrzebnych informacji.

3.1 Narzędzia

Logika aplikacji zaimplementowana została w języku Python. Podczas zajęć z przedmoitu obieralnego na czwartnym semestrze studiów grupa miała wprowadzenie do technik przetwarzania obrazów w języku Python, dlatego też postanowiono kontynuować pracę w tym języku oraz tej technologii.

Do rozpoznawania obrazu wykorzystywana była biblioteka Open CV, która jest obecnie najpopularniejszą, wieloplatformową biblioteką do przetwarzania obrazów oraz wideo w czasie rzeczywistym. Posiada ona bardzo dobrą dokumentację oraz wiele darmowych poradników, które znacznie ułatwiały prace nad projektem.

Do przechwycenia obrazu z kamery telefonu została wykorzystana darmowa aplikacja mobilna o nazwie DroidCam Wireless Webcam, która jest ogólnodostępna w Google Play. Pozwala ona szybkie i proste przekazywanie obrazu z telefonu komórkowego do komputera.

Interfejs graficzny wykonano przy użyciu biblioteki graficznej Tkinter, opartej na bibliotece Tk. Grupa nie miała wcześniej doczynienia z aplikacjami graficznymi w języku Python dlatego zdecydowano się na tę prostą w obsłudze bibliotekę.

Narzędzia, które zostały wykorzystane do wspierania i ułatwienia pracy zespołowej to:

- system Trello.com do organizacji zadań,
- system kontroli wersji Git wraz z repozytorium na stronie GitHub.com,
- system Overleaf.com do tworzenia dokumentacji w języku Latex,

- program Slack do komunikacji między członkami zespołu,
- program Visual Paradigm do tworzenia schematów oraz diagramów.

3.2 Środowisko

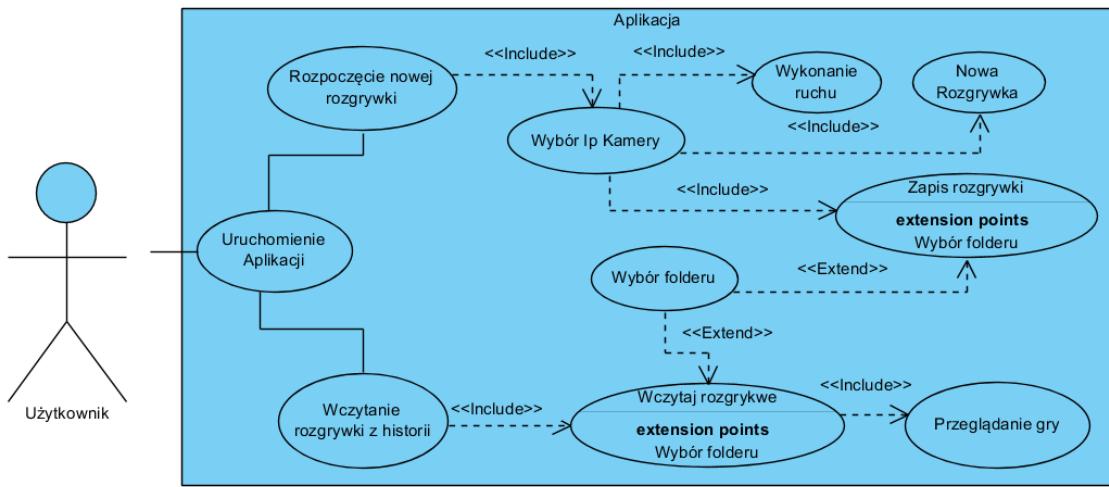
Zintegrowanym środowiskiem programistycznym jakie wybrano do stworzenia aplikacji jest PyCharm Community 2017. Jest to produkt stworzony przez firmę JetBrains oferującą wiele funkcjonalności dla języka programowania Python.

4 Diagramy UML

Diagramy UML przedstawiają w sposób graficzny działanie całego systemu. Jest to graficzny system wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych. UML to notacja umożliwiająca zaprezentowanie systemu w sposób graficzny, w postaci diagramów. Modele zapisane w języku UML prezentują system od ogółu do szczegółu, umożliwiając oglądanie modelu systemu z wybraną w danym momencie dokładnością. W zależności od wybranego diagramu można obejrzeć model systemu z różną szczegółowością.

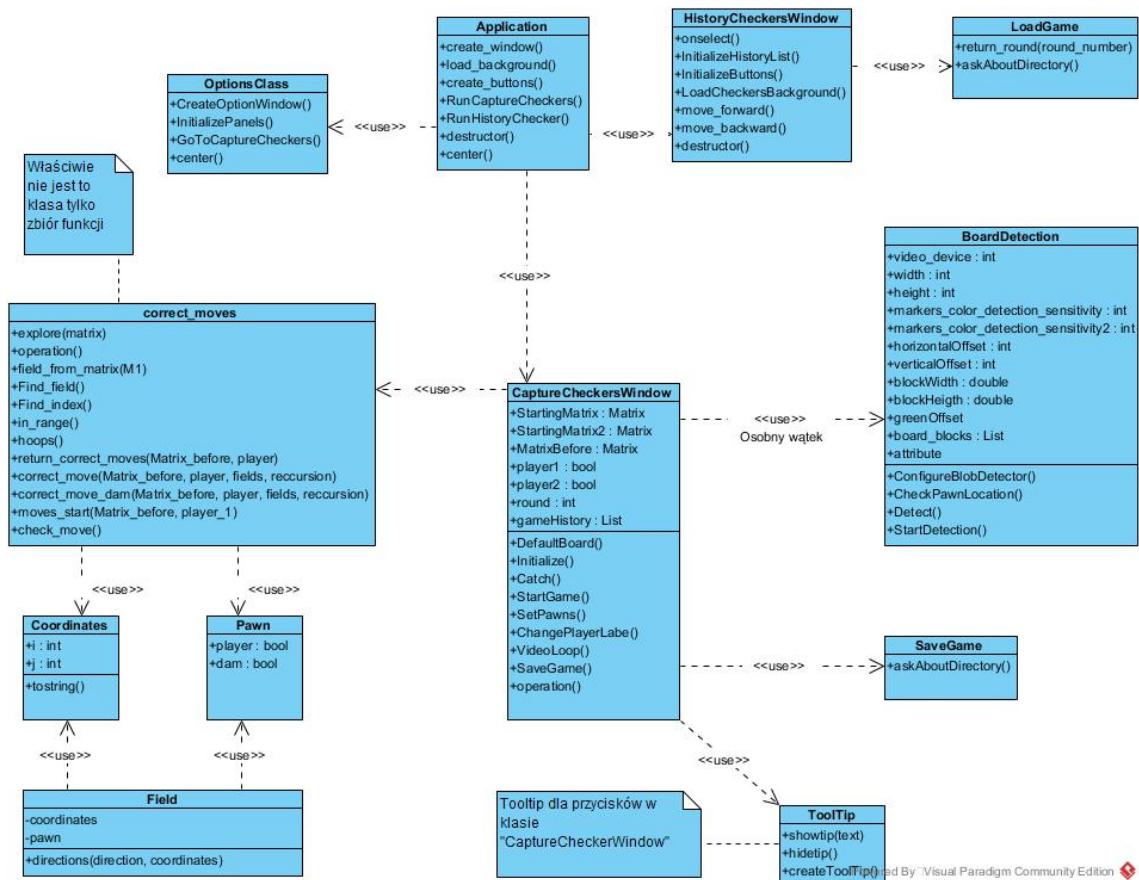
4.1 Diagram przypadków użycia

Poniższy diagram przypadków użycia przedstawia w sposób graficzny funkcjonalności systemu oferowane przez aplikację.



Rysunek 1: Diagram przypadków użycia dla aplikacji.

Poniżej został przedstawiony diagram klas pokazujący hierarchie oraz najważniejsze metody.



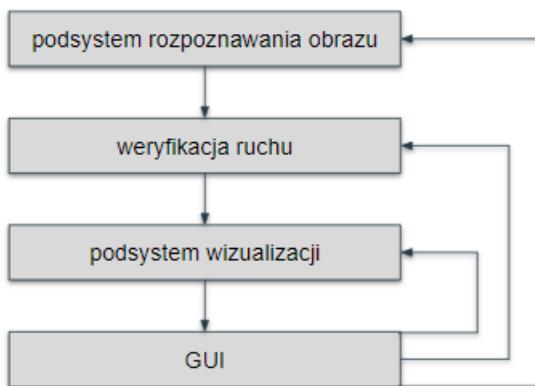
Rysunek 2: Diagram klas.

5 Architektura systemu.

Program jest podzielony na trzy współpracujące ze sobą moduły, które wymagane są do poprawnego funkcjonowania całej aplikacji. Wyróżnić można następujące moduły:

- podsystem rozpoznawania i przechwytywania obrazu,
- system weryfikacji poprawności ruchu,
- podsystem wizualizacji.

Do prawidłowego działania aplikacji niezbędne jest połączenie funkcjonalności implementowanych przez każdy z nich a następnie zaprezentowanie całej użyteczności na ekranie użytkownika. Poniżej przedstawiony został opis poszczególnych podsystemów.



Rysunek 3: Architektura systemu.

Warto zaznaczyć, że cały podsystem rozpoznawania obrazu działa niezależnie na osobnym wątku. Pozostałe moduły odczytują aktualne parametry detekcji w wybranym momencie (np. gdy użytkownik naciśnie odpowiedni przycisk).

5.1 Rozpoznawanie obrazu

PRZECHWYTYWANIE OBRAZU Z KAMERY

Funkcję kamery, która monitoruje planszę oraz trwającą rozgrywkę w warcaby, pełni telefon komórkowy z systemem operacyjnym Android oraz zainstalowaną aplikacją DroidCam Wireless Webcam.

By nawiązać łączność komputera z urządzeniem mobilnym, konieczne jest połączenie się za pośrednictwem sieci LAN lub WiFi. Do poprawnego nawiązania połączenia wymagane jest, aby użytkownik poprawnie skonfigurował aplikację poprzez określenie adresu IP urządzenia (telefonu komórkowego) oraz portu, który posłuży do komunikacji.

WYKRYWANIE PLANSZY

W celu poprawnego wykrycia planszy gry konieczne jest umieszczenie na narożnikach tak zwanych markerów, które po odpowiednim przetworzeniu pozwolą na prezyjne przechwycenie tablicy gry. W zależności od wybranych ustawień w panelu konfiguracyjnym aplikacji, markery mogą być rozmiieszczone w różnych odległościach od narożników planszy.

Najefektywniejszą metodą wykrycia oraz przetwarzania markerów okazała się być metoda zwana detekcją blobów.

Blob to grupa połączonych pikseli obrazu, które mają wspólną właściwość, którą może być na przykład wartość skali szarości. Detekcja musi być wykonana na ośmioróżkowym obrazie, czyli obrazie w skali szarości lub na obrazie binarnym, wtedy metoda polega na wykrywaniu czarnych obiektów. Detekcja blobów pozwala na wykrywanie dowolnego kształtu.

Biblioteka OpenCV pozwala na detekcję blobów i filtrację na podstawie wielu właściwości. Wykrywanie może odbywać się na podstawie koloru (jasny, ciemny), rozmiaru obiektu lub kształtu danego obiektu (kolistość, wypukłość, bezwładność).

Kolejnym etapem przetwarzania obrazu jest wycięcie fragmentu obrazu na podstawie środków ciężkości znalezionych markerów oraz wyrównanie obrazu za pomocą przekształceń afiniycznych. Nowo uzyskany obraz zostaje niejawnie podzielony na 64 równe bloki reprezentujące pojedyncze pole na planszy.

WYKRYWANIE I ROZRÓŻNIANIE PIONKÓW GRACZY

Detekcja pionków odbywa się przy pomocy tej samej metody co wykrywanie

markerów - detekcji blobów.

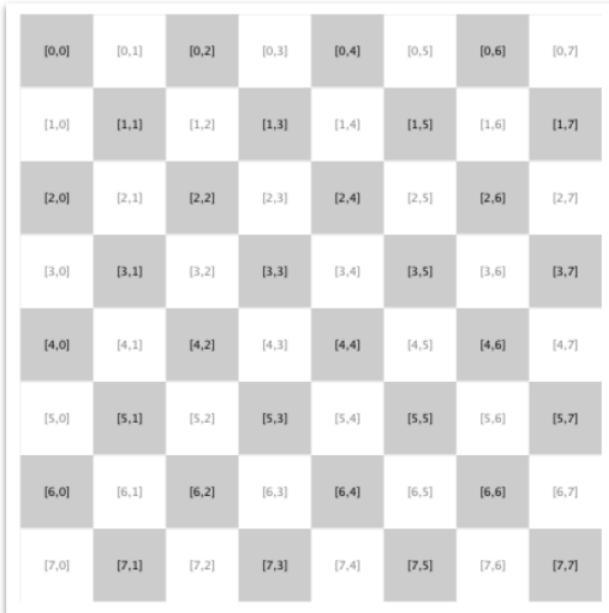
Każdy pionek ma sprawdzaną pozycję i na tej podstawie jest przypisywany do odpowiedniego bloku. Każdy blok może przyjmować następujące wartości:

- 0 - puste pole,
- 1 - zwykły pionek gracza nr 1,
- 3 - damka gracza nr 1,
- 2 - zwykły pionek gracza nr 2,
- 4 - damka gracza nr 2 .

Lista odpowiednio przygotowanych bloków zostaje dalej wysłana do podsystemu wizualizacji i poprawności ruchu. Zakłada się, że pierwszeństwo ruchu należy do gracza nr 1, czyli do niebieskich pionków. Szczegółowe informacje na temat metod detekcji zostaną wyjaśnione w następnym rozdziale.

5.2 Wizualizacja

Głównym zadaniem modułu wizualizacji jest mapowanie macierzy cyfr dostarczanej przez moduł rozpoznawania obrazów na komputerową reprezentację planszy gry. Otrzymywana macierz obrazuje ułożenie pionków na planszy. Dane dostarczane przez moduł sprawdzania poprawności ruchu są interpretowane i w zależności od wyniku, wizualizacja jest aktualniana, bądź pojawia się komunikat o konieczności cofnięcia ruchu, gdyż był on niezgodny z założonymi zasadami.



Rysunek 4: Mockup ekranu historii rozgrywki.

5.3 Sprawdzanie poprawności ruchu

Istnieje wiele różnych odmian gry w warcaby. Główne różnice wynikają przede wszystkim z:

- rozmiarów planszy (od 8 do 12 pól szerokości bądź długości),
- liczby pionów (od 8 do 30 na zawodnika),
- zasad bicia,
- sposobu poruszania się damki.

W projekcie tym zespół zdecydował się na sprawdzanie poprawności wykonywanego ruchu według zasad gry w warcaby w wariantie niemieckim, gdzie:

- rozmiar planszy wymosi 8x8,
- pionki poruszają się po czarnych polach,
- bicie jest obowiązkowe, gracz wybiera dowolne bicie z możliwych,
- bicie możliwe do przodu i do tyłu,
- damka porusza się o dowolną liczbę pól,
- damka po biciu musi zatrzymać się na pierwszym polu za zbitym pionkiem.

W module zaimplementowane są dwie główne metody, jedna dla zwykłych pionków i druga dla pionków, które są damkami. Podczas tury danego gracza generowana jest lista wszystkich możliwych sekwencji ruchów, jakie mogą być wykonane. Ze względu na obowiązek bicia, metody te wykorzystują mechanizm rekurencji. Jeżeli zostało wykryte bicie, z listy usuwane są pojedyncze ruchy. Pola na liście zapisane są w postaci litera+cyfra (np. pole o współrzędnych (0,0) zapisane jest jako A8). Pierwszym elementem na liście jest pozycja, z której rozpoczyna się wykonywanie sekwencji ruchów. Gdy gracz wykona ruch i zatwierdzi go odpowiednim przyciskiem, moduł porównuje plansze sprzed wykanania ruchu i po wykonaniu ruchu. Następnie sprawdzane jest czy wykryty ruch znajduje się na liście prawidłowych ruchów. Rezultat przekazywany jest do modułu wizualizacji, który w zależności od otrzymanego wyniku wykonuje odpowiednie operacje.

6 Historia rozgrywki

Aplikacja pozwala na zapisywanie i wczytywanie odbytej partii do/z pliku zewnętrznego. Zapisywany plik jest w formacie JSON i zapisywane są tam takie dane jak:

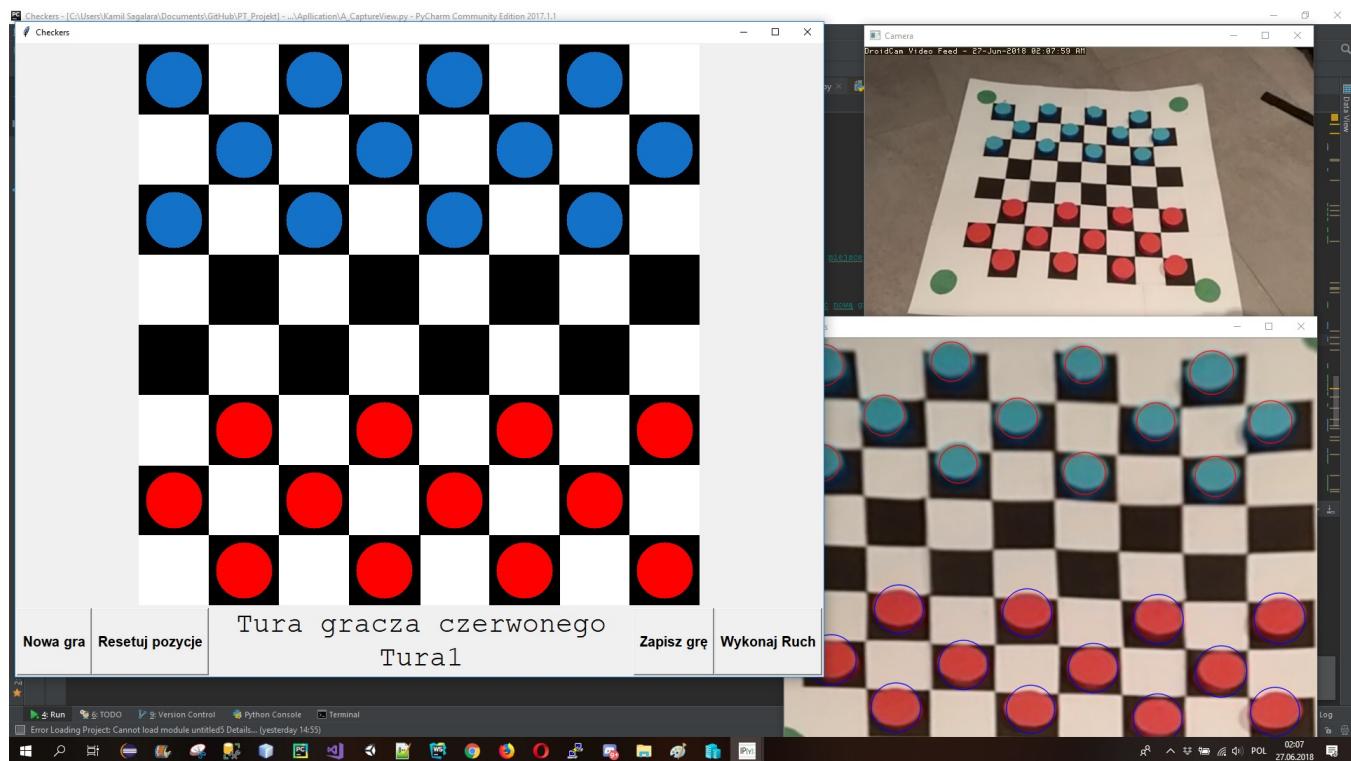
- nazwa gry,
- data partii,
- pseudonim pierwszego gracza,
- pseudonim drugiego gracza,
- historia rozgrywki,
 - numer rundy,
 - stan planszy w danej rundzie.

```
{
    "game_name": "Trial",
    "date": "Jun 1 2018",
    "player1_name": "Andrei",
    "player2_name": "Alex",
    "game_history": [
        {
            "round": "0",
            "pawns": [
                [1,0,1,0,1,0,1,0],
                [0,1,0,1,0,1,0,1],
                [1,0,1,0,1,0,1,0],
                [0,0,0,0,0,0,0,0],
                [0,0,0,0,0,0,0,0],
                [0,2,0,2,0,2,0,2],
                [2,0,2,0,2,0,2,0],
                [0,2,0,2,0,2,0,2]
            ]
        },
    ],
}
```

Rysunek 5: Przykładowy nagłówek pliku z zapisaną grą.

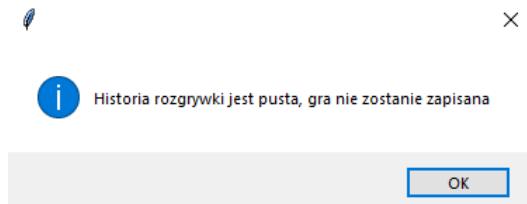
6.1 Zapisywanie rozgrywki

Zapisanie rozgrywki jest możliwe podczas trwania nowej rozgrywki, po kliknięciu na przycisk “zapisz grę”.



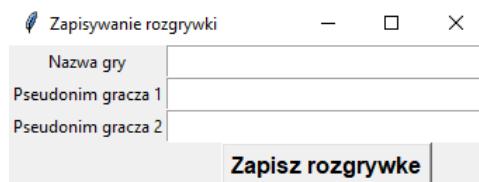
Rysunek 6: Ekran rozgrywki.

Zapisanie rozgrywki jest możliwe po wykonaniu chociaż jednego ruchu w przeciwnym razie użytkownikowi zostanie wyświetlony komunikat.



Rysunek 7: Komunikat o braku możliwości zapisania rozgrywki.

Jeżeli historia rozgrywki nie jest pusta a użytkownik chce zapisać grę, zostanie poproszony o wypełnienie dodatkowych danych zapisywanych wraz z przebiegiem rozgrywki.



Rysunek 8: Okno wprowadzania dodatkowych informacji do zapisywanej rozgrywki.

6.2 Wczytywanie rozgrywki

Aplikacja pozwala na wczytanie wcześniej zapisanej rozgrywki i odbywa się to poprzez wybranie opcji ‘wczytaj z historii’ podczas startu aplikacji. Następnie użytkownik zostanie poproszony o wskazanie lokalizacji pliku z zapisaną grą.



Rysunek 9: Ekran startowy aplikacji.

7 Zastosowane metody przetwarzania obrazu

W poniższym rozdziale opisano zastosowane metody przetwarzania obrazu wykorzystane w projekcie. Metody wykrywania opierają się głównie na rozpoznawaniu kolorów. Obrazy domyślnie są przetwarzane w formacie RGB, nie jest to jednak wygodny sposób wykrywania barw. Rozwiązaniem jest konwersja na przestrzeń barw HSV, gdzie wartość barwy jest przechowywana w jednej zmiennej.

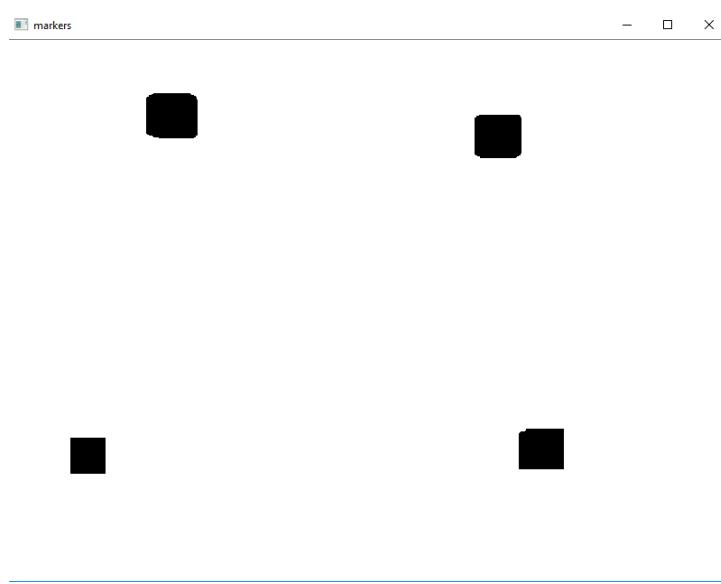
7.1 Wykrycie planszy

W celu poprawnego wykrycia pola gry na rogach planszy zostały umieszczone markery koloru zielonego. Obraz wstępnie jest konwertowany do przestrzeni barw HSV. Używając wartości HSV łatwiej jest ustawić zakres koloru, ponieważ informacja o barwie jest zawarta w jednej zmiennej (w przypadku RGB są to 3 zmienne), pozostałe wartości mają niewielkie znaczenie przy detekcji. Następnie mając odpowiednio dobraną wartość koloru można wykryć na obrazie tylko zielone obiekty. W wyniku tej operacji powstaje obraz binarny (czarno-biały), gdzie zielone obiekty

są reprezentowane białym kolorem. W celu pozbycia się zakłóceń i wygładzenia kształtów obraz zostaje przetworzony z użyciem dylatacji i erozji. Następnie obraz binarny jest odwracany, tak aby obiekty były reprezentowane kolorem czarnym.



Rysunek 10: Plansza z ustawnionymi markerami.



Rysunek 11: Wykryte markery w postaci obrazu binarnego.

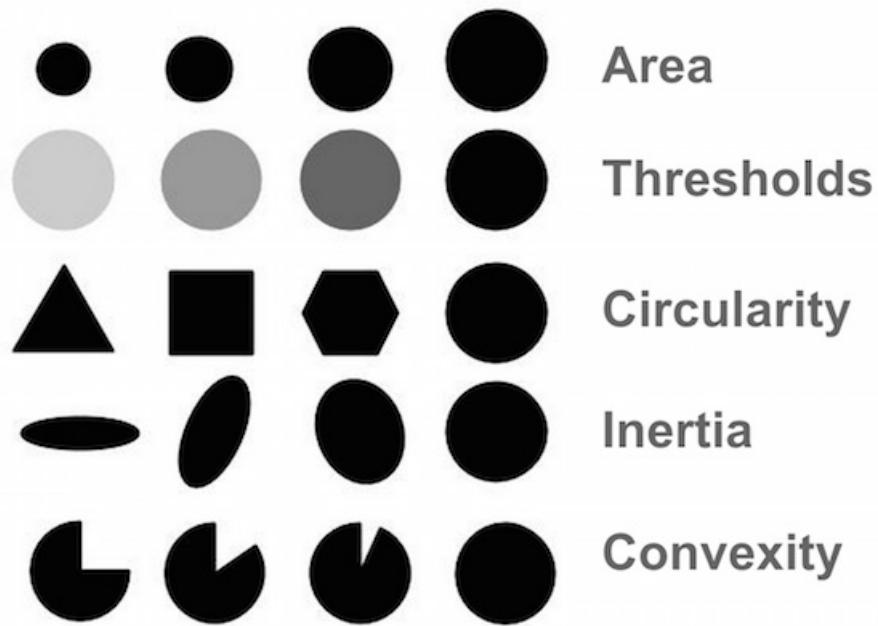
Na tak przygotowanym ekranie można zastosować detekcję blobów, która polega na znalezieniu zbioru pikseli o podobnych cechach - w tym przypadku cechą jest taki

sam kolor. Zaletą tej metody jest możliwość wykrycia nieregularnych kształtów, pod warunkiem, że stanowią jednolitą całość (bez szumów i nachodzących się obiektów). Wykryte bloby mają w sobie informację o pozycji na ekranie.



Rysunek 12: Przykładowe bloby o nieregularnych kształtach.

Algorytm ten wymaga wstępnej konfiguracji. Zmiana parametrów konfiguracji pozwala na dostosowanie detekcji do konkretnego obrazu. Przykładowe parametry zostały przedstawione na rysunku poniżej.



Rysunek 13: Kryteria detekcji blobów.

```
def ConfigureBlobDetector(self):
    # Setup SimpleBlobDetector parameters.
    params = cv2.SimpleBlobDetector_Params()

    params.filterByColor = True
    params.blobColor = 0

    # Change thresholds
    params.minThreshold = 10;
    params.maxThreshold = 200;

    # Filter by Area.
    params.filterByArea = True
    params.minArea = 400

    # Filter by Circularity
    params.filterByCircularity = True
    params.minCircularity = 0.1

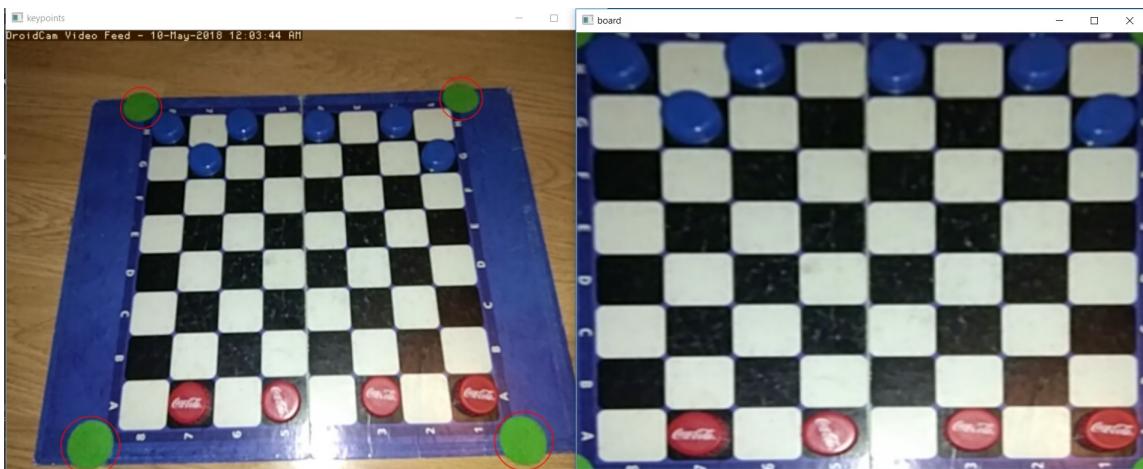
    # Filter by Convexity
    params.filterByConvexity = True
    params.minConvexity = 0.87

    # Filter by Inertia
    params.filterByInertia = True
    params.minInertiaRatio = 0.01

    return cv2.SimpleBlobDetector_create(params)
```

Rysunek 14: Przykładowa konfiguracja w języku Python.

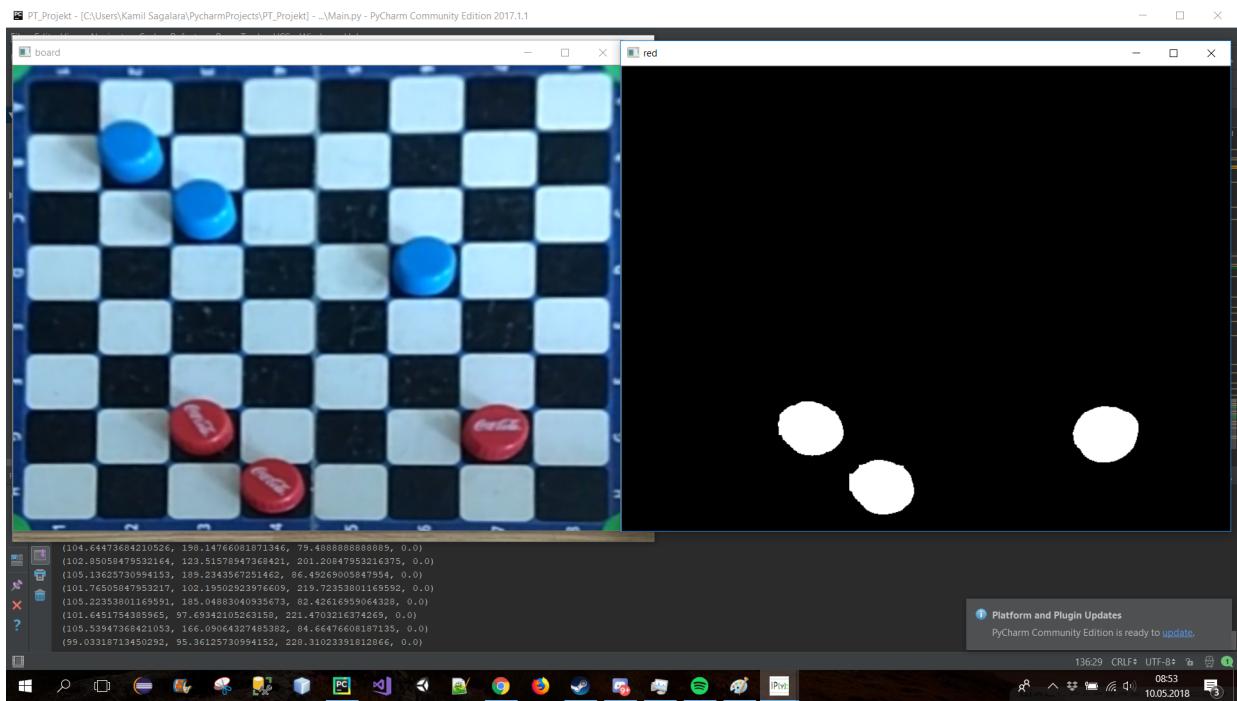
Dzięki detekcji blobów można wyznaczyć krawędzie planszy i wyodrębnić ją z obrazu. Następnie aby wyrównać szachownicę, należy zastosować przekształcenie perspektywiczne - bez tego nie będzie możliwy podział na równe bloki. Zastosowanie przekształcenia perspektywistycznego pozwala na umieszczenie kamery pod innym kątem niż kąt prosty - obraz i tak będzie wyrównany.



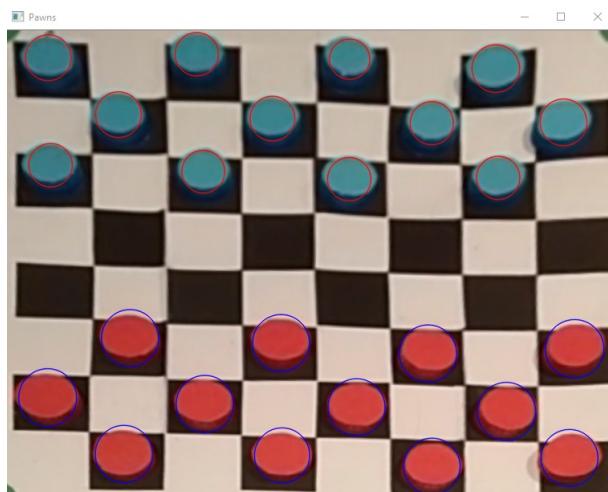
Rysunek 15: Plansza po przekształceniu perspektywicznym.

7.2 Wykrycie pionków

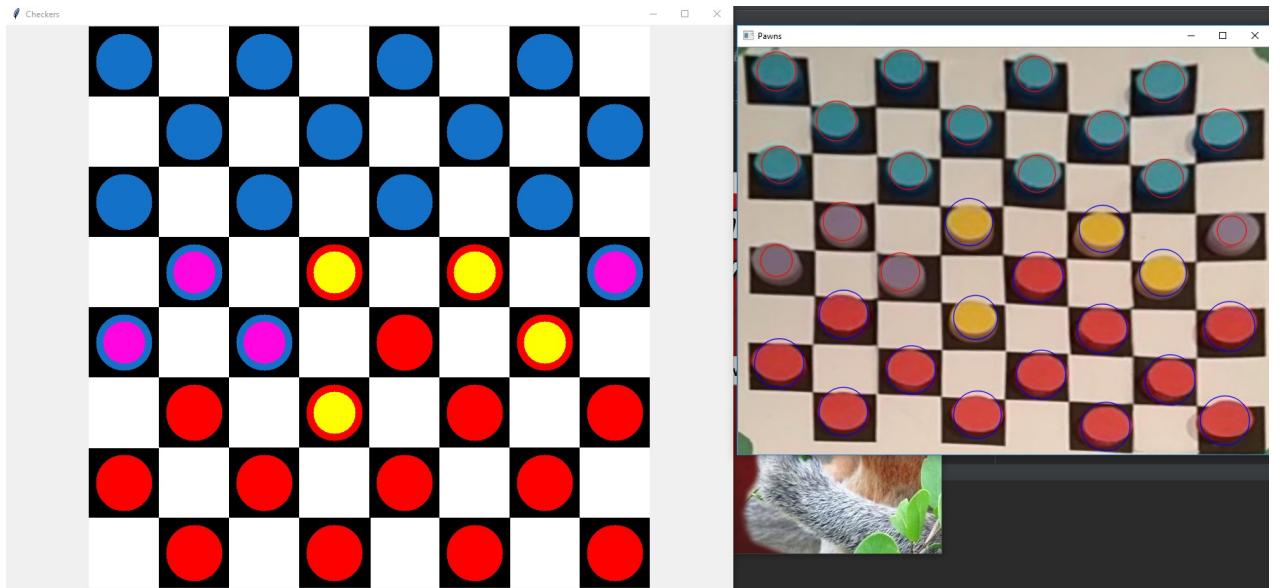
Podział wyciętej planszy polega na tym, że wysokość i szerokość obrazu jest zostaje podzielona przez 8. Wartości odpowiednich przedziałów są zapisywane do tablicy. Wygląda to w ten sposób, że uzyskujemy 64 przedziały, które reprezentują każdą kratkę planszy. Do wykrycia pionków została zastosowana ta sama metoda, co w przypadku wykrywania planszy - detekcja blobów. Najpierw dla każdego koloru pionka jest tworzony obraz binarny. Tak samo jak przy markerach, jest używana dyfuzja i erozja w celu usunięcia szumów, a następnie maska zostaje odwrócona. Dla każdej utworzonej maski zostają wykryte bloby. Następnie pozycje blobów są przydzielane do przedziałów z wcześniej utworzonej tablicy. W ten sposób uzyskujemy informację o kolorze występującym w danym przedziale - czyli znamy już dokładną lokalizację pionków. Każdy kolor jest reprezentowany przez inną liczbę całkowitą, a elementy tablicy bez przydzielonego koloru mają wartość domyślną 0.



Rysunek 16: Detekcja czerwonych pionków.



Rysunek 17: Pionki zaznaczone na podstawie detekcji blobów.

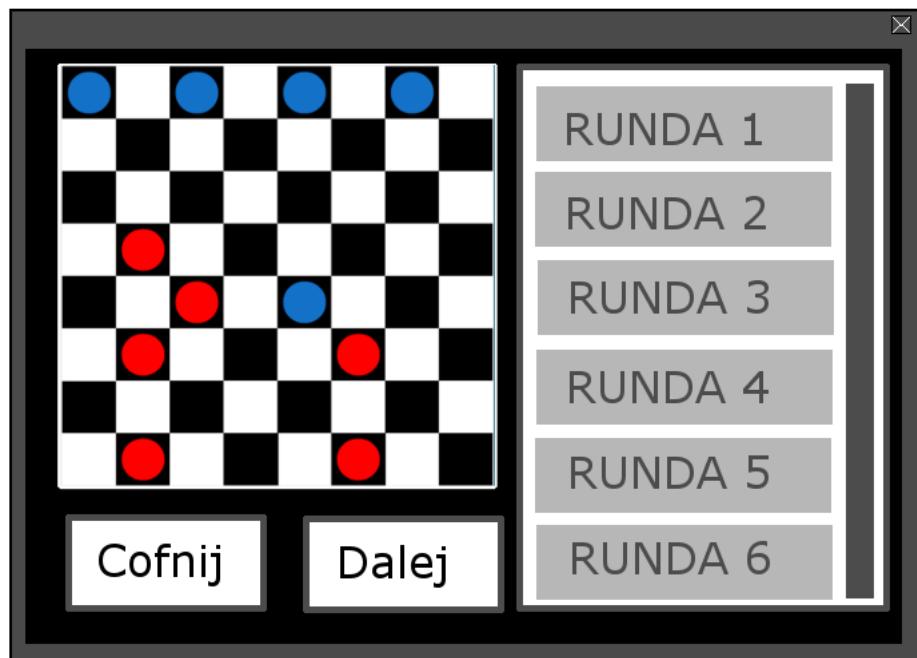


Rysunek 18: Prawidłowa detekcja różnych kolorów

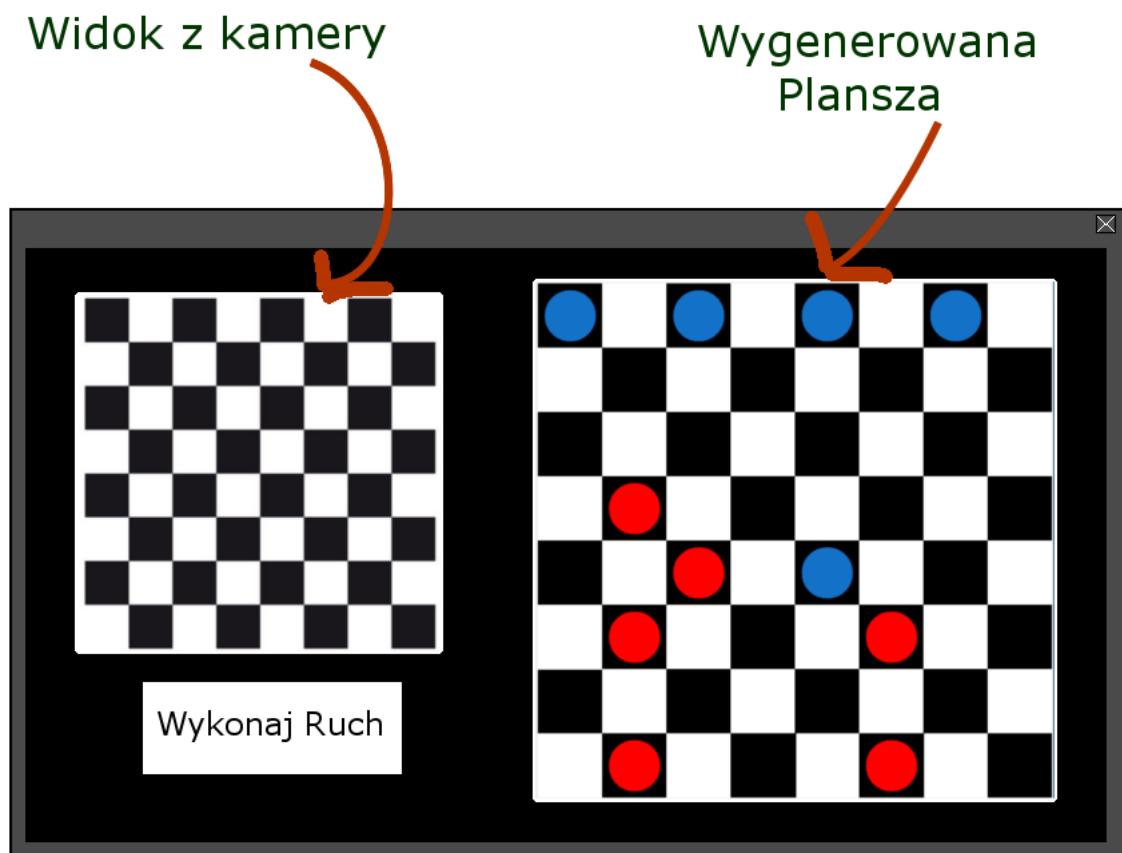
Jak widać na powyższych rysunkach, wszystkie pionki są wykryte, a ich pozycja jest ustalona z dużą precyzją.

8 Mockupy

Poniżej przedstawione są projekty interfejsów wykonane przed przystąpieniem do właściwej implementacji projektu.



Rysunek 19: Mockup ekranu historii rozgrywki.



Rysunek 20: Mockup właściwego ekranu rozgrywki.

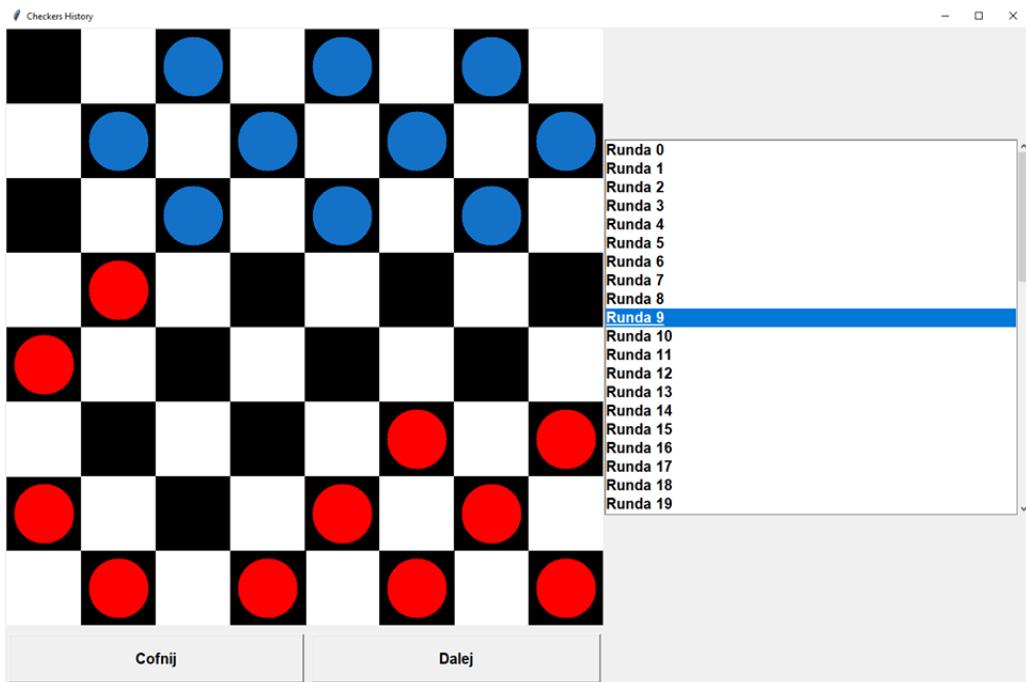
9 Instrukcja użytkowania

Po uruchomieniu aplikacji użytkownik ma możliwość rozpoczęcia nowej sesji lub wczytania zapisanego w historii stanu gry w celu analizy i prześledzenia konkretnej rozgrywki. Ekran startowy zawiera przyjazny dla użytkownika interfejs umożliwiający wybór trybu programu - rozgrywkę w czasie rzeczywistym lub podgląd historii gier.



Rysunek 21: Ekran startowy po uruchomieniu aplikacji.

W przypadku wyboru wczytania rozgrywki poprzez naciśnięcie przycisku “Wczytaj z Historii”, na ekranie wyświetlana jest aktualna plansza gry oraz zapisana historia ruchów przechowywana w formacie JSON.



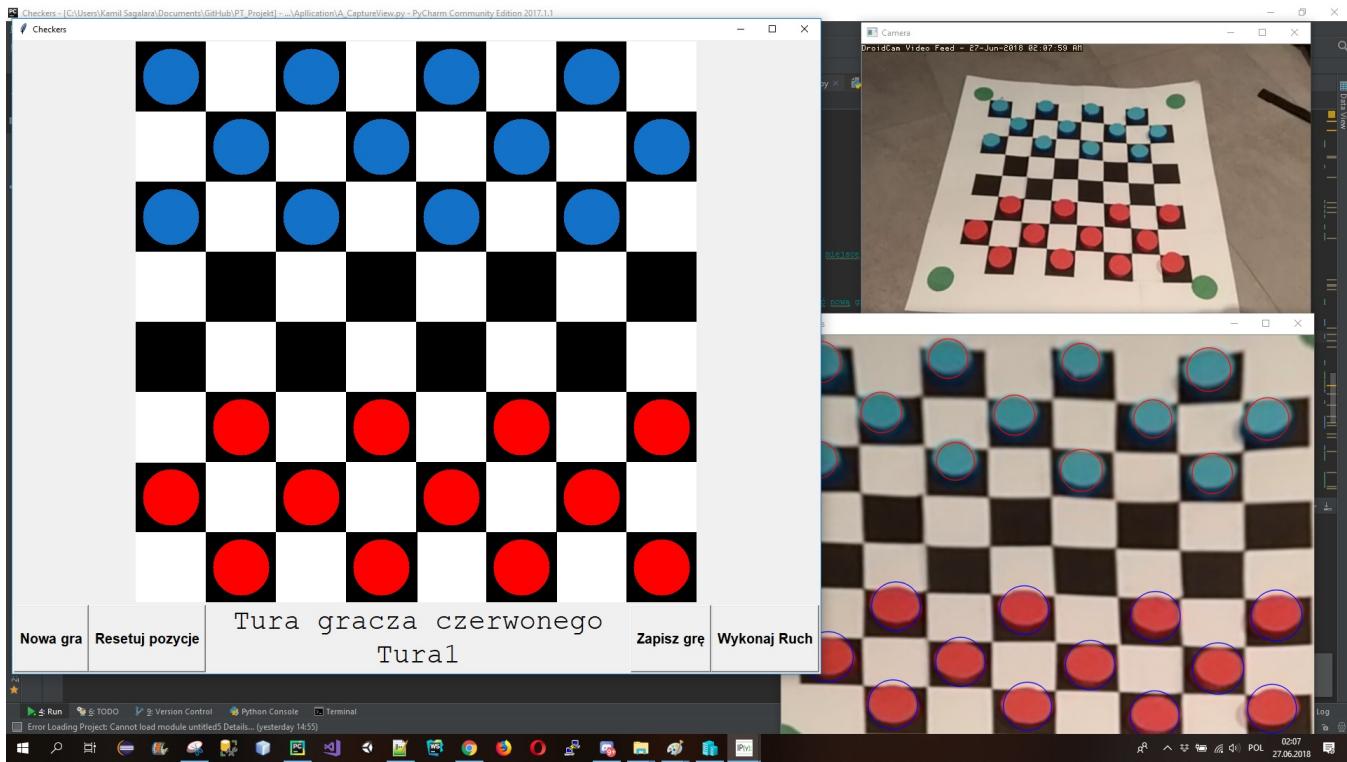
Rysunek 22: Historia wczytanej rozgrywki.

Po naciśnięciu przycisku “Rozpocznij nową rozgrywkę” użytkownik jest przekierowywany do okna konfiguracji obrazu przechwytywanego z kamery za pomocą aplikacji DroidCam.



Rysunek 23: Okno łączenia z kamerą.

Następnie po kliknięciu przycisku “Przejdź do GRY” wyświetlany jest właściwy obraz rozgrywki zawierający obraz kamery oraz jego komputerowe odwzorowanie.



Rysunek 24: Ekran rozgrywki.

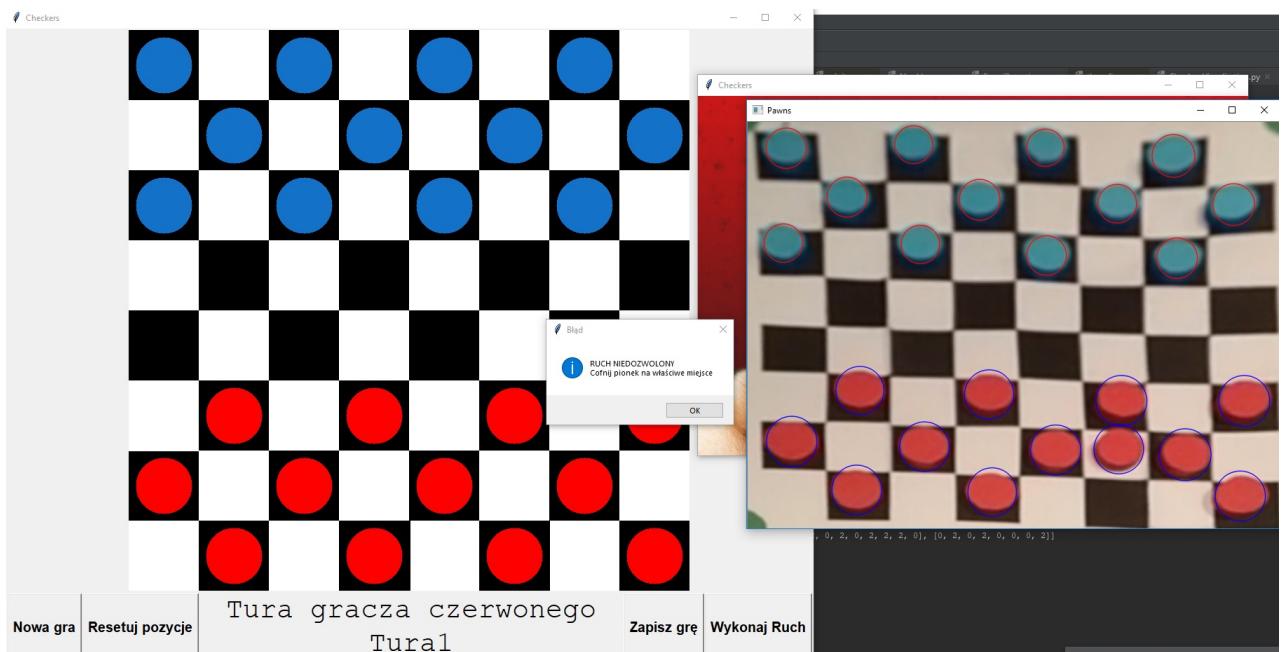
Podczas rozgrywki wyświetlane są trzy okna: obraz z kamery, wycięta plansza z zaznaczonymi pionkami oraz okno symulacji umożliwiające sterowanie rozgrywką. Dzięki temu, że okno z planszą ma zaznaczone aktualnie wykryte pionki, użytkownik może łatwo zauważać potencjalne błędy. Źle ustawiona kamera lub niewłaściwe oświetlenie może przekłamać detekcję, ale każdy błąd jest widoczny w czasie rzeczywistym.

Okno symulacji zawiera cztery przyciski:

- Nowa gra - przechwytuje obraz planszy i sprawdza czy jest zgodny ze startowym rozstawieniem pionków. Dodatkowo kasuje historię poprzedniej rozgrywki z pamięci RAM.
- Resetuj pozycję - przechwytuje aktualny obraz planszy bez żadnych konsekwencji - używane tylko wtedy gdy wystąpi nieoczekiwany błąd w ustawieniu pionków. Użytkownik ma możliwość poprawienia ustawień.

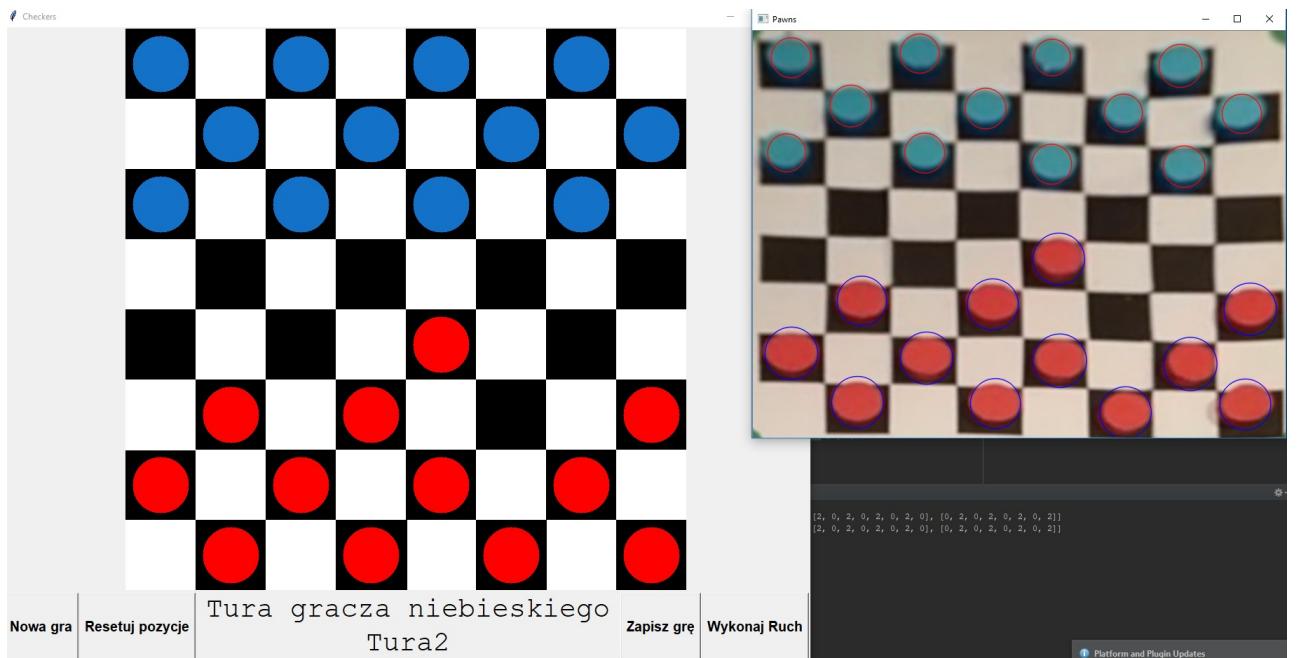
- Zapisz grę - zapisuje wszystkie wykonane ruchy do pliku JSON. Można zapisywać niedokończone rozgrywki.
- Wykonaj ruch - przechwytuje aktualny stan planszy i sprawdza poprawność ruchu. W przypadku błędного ruchu wyświetla odpowiedni komunikat oraz nie aktualizuje okna symulacji.

Aktywny gracz wykonuje swój ruch i potwierdza przyciskiem *Wykonaj ruch*. Dalej, w zależności od wyniku weryfikacji poprawności ruchu, zmieniany jest aktywny gracz lub wyświetlany jest komunikat o konieczności wykonania innego ruchu. Na dole ekranu jest wyświetlana informacje na temat tego, który gracz wykonuje ruch oraz jaka jest aktualna tura. Jeżeli na planszy zabraknie pionków jednego z graczy, gra automatycznie się zakończy.



Rysunek 25: Komunikat o błędny ruchu.

Jak widać na powyższym rysunku, błędny ruch nie aktualizuje symulacji. Po udanym ruchu kolejka przechodzi do następnego gracza, a informacja o turze zostaje zaktualizowana.



Rysunek 26: Stan programu po udanym ruchu.

10 Podział prac

W poniższej tabeli przedstawiono podział prac zespołu.

Tablica 1: Podział prac.

Osoba	Zakres wykonywanych prac
Paulina Mrozek	<ul style="list-style-type: none"> ● Tworzenie dokumentacji ● Utworzenie interfejsu graficznego aplikacji. ● Opracowanie systemu sprawdzania poprawności wykonywanych ruchów zgodnie konwencją niemiecką. ● Testowanie aplikacji i poprawa błędów.
Kamil Sagalara	<ul style="list-style-type: none"> ● Tworzenie dokumentacji ● Implementacja segmentu odpowiedzialnego za przetwarzanie obrazu przechwyconego z kamery. ● Poprawa modułu sprawdzania ruchu. ● Testowanie aplikacji i poprawa błędów.
Kornel Krześlak	<ul style="list-style-type: none"> ● Tworzenie dokumentacji ● Implementacja segmentu odpowiedzialnego za przetwarzanie obrazu przechwyconego z kamery. ● Testowanie aplikacji i poprawa błędów.
Hubert Springer	<ul style="list-style-type: none"> ● Tworzenie dokumentacji ● Implementacja segmentu odpowiedzialnego za odzwierciedlenie i wizualizacje ruchu pionków. ● Opracowanie systemu wczytywania i zapisywania gry. ● Testowanie aplikacji i poprawa błędów.

11 Napotkane problemy

Na każdym etapie realizacji projektu pojawiały się błędy i problemy. Część z nich udało się wyeliminować. Niektóre wymusiły na autorach niewielkie zmiany w początkowo założonej koncepcji.

WYKRYWANIE PLANSZY I PIONKÓW

Dużym problemem była poprawna detekcja planszy oraz pionków. Autorzy aplikacji testowali wiele różnych metod oraz sposobów oferowanych przez bibliotkę OpenCV, m.in. wykrywanie konturów, wykrywanie okręgów oraz środków ciężkości. Początkowo do wykrywania planszy próbowało wykorzystać metodę wykrywania konturów oraz wykrywanie okręgów. Niestety obie metody okazały się być nieskuteczne. Wykrycie okręgów możliwe było tylko wtedy, kiedy kształt był idealnym okręgiem, co byłoby niemożliwe ze względu na różne nachylenia kamery.

Najlepszym rozwiązaniem okazała się detekcja blobów (dowolnych kształtów mających wspólne właściwości), która została zastosowana do wykrywania krańcowych markerów planszy oraz samych pionków.

SPRAWDZENIE POPRAWNOŚCI RUCHÓW

Początkowo implementacja sprawdzania poprawności ruchu nie zakładała wykorzystania mechanizmu rekurencji. Jednak szybko okazało się że jest on niezbędny w związku z zasadą obowiązkowego wielobicia.

POŁĄCZENIE MODUŁÓW

Największym wyzwaniem było zintegrowanie zaimplementowanych modułów w jedną, spójnie działającą całość.

12 Perspektywa rozwoju

W poniższej części została przedstawiona perspektywa dalszego rozwoju aplikacji.

- Dodanie większej liczby wariantów gry w warcaby.
- Możliwość konfiguracji kolorów.
- Ustawienie parametrów detekcji.
- Stworzenie bardziej rozbudowanego interfejsu.
- Dodanie ograniczenia czasowego na wykonanie ruchu oraz całą rozgrywkę.
- Dodanie możliwości automatycznego odtwarzania kolejnych rund wczytanej rozgrywki.
- Możliwość ustawienia wyświetlanego pseudonimu gracza.
- Możliwość wczytania gry z historii i kontynuacja rozgrywki.
- Podpowiadanie ruchu.
- Zastosowanie sieci neuronowej do klasyfikacji pionków.