Podstawy Teleinformatyki
Rozpoznawanie obrazu z gry w Warcaby oraz wizualizacja stanu gry na komputerze

Marcin Orczyk Natalia Popielarz Piotr Wołyński

28 czerwca 2018

Marcin Orczyk	126804	marcin.orczyk5@gmail.com
Natalia Popielarz	126803	natalia.popielarz@student.put.poznan.pl
Piotr Wołyński	126832	piotr.wolynski@student.put.poznan.pl

Spis treści

1 Charakterystyka projektu **3**
1.1 Opis projektu . . . 3
1.2 Opis gry . . . 3
1.3 Uzasadnienie wyboru tematu . . . 3
1.4 Moduły projektu i podział prac . . . 4
1.5 Narzędzia . . . 4

2 Wymagania projektu **4**
2.1 Wymagania funkcjonalne . . . 4
2.2 Wymagania niefunkcjonalne . . . 5

3 Architektura rozwiązania **5**

4 Opis implementacji **5**
4.1 Opis implementacji modułu rozpoznawania obrazu . . . 5
4.2 Opis interfejsu użytkownika . . . 11
4.3 Opis implementacji logiki gry w Warcaby . . . 15
4.4 Opis implementacji systemu wizualizacji . . . 16

5 Instrukcja użytkowania aplikacji **23**
5.1 Pełna instrukcja obsługi aplikacji . . . 23
5.2 Skrócona instrukcja obsługi aplikacji . . . 24

6 Podsumowanie **25**
6.1 Napotkane problemy . . . 25
6.2 Dlaczego C++ jest lepszy od C#? . . . 27
6.3 Podział prac . . . 28
6.4 Cele zrealizowane . . . 29
6.5 Perspektywa rozwoju . . . 29

1 Charakterystyka projektu

1.1 Opis projektu

Projekt zakłada stworzenie programu do cyfrowej wizualizacji stanu gry w Warcaby, którego zadaniem będzie przeniesienie stanu planszy fizycznej do komputera oraz weryfikacja poprawności ruchów wykonywanych przez graczy. Nad planszą do gry w Warcaby umieszczona będzie kamera. Moduł do rozpoznawania obrazu z kamery wyszuka pozycje pionków i przekaże je do modułu weryfikacji, który sprawdzi, czy wykonano poprawny ruch. Jeśli tak, dane o ruchu przekazywane zostaną do modułu wizualizacji, w przeciwnym wypadku gracze otrzymają informację o wykonaniu błędnego ruchu. Moduł wizualizacji będzie przedstawiać aktualny, poprawny stan planszy fizycznej w postaci cyfrowej.

Realizowany program pozwoli na zrewolucjonizowanie transmisji meczów w Warcaby. Dzięki programowi będzie istniała możliwość transmisji rozgrywek na żywo dla osób, które posiadają łącze internetowe o niskiej przepustowości, limity transmisji danych bądź urządzenia mobilne o małej wydajności (np. z aplikacjami napisanymi w języku Java). Dzięki temu oprogramowaniu, osoby takie będą mogły w czasie rzeczywistym (o ile transmisję danych w sieciach pakietowych opartych o protokoły Transmission Control Protocol/User Datagram Protocol /Internet Protocol można nazwać transmisją w czasie rzeczywistym) śledzić postępy ulubionych zawodników. Program cechuje się wyjątkowo niskim zapotrzebowaniem na zasoby sprzętowe i sieciowe, ponieważ umożliwia przesyłanie jedynie informacji o planszy (takich jak rozmieszczenie pionków czy informacje o pionkach takich jak ich kolor i rodzaj — czy jest to zwykły pionek, czy damka). Nie jest wymagana transmisja całego obrazu, co jest realizowane w klasycznych systemach telewizji internetowej. Dzięki temu nie jest konieczne posiadanie szybkiego dostępu do Internetu ani nie trzeba angażować procesora komputera w dekodowanie sygnału audio i wideo.

1.2 Opis gry

Program umożliwia wizualizację planszy, na której rozgrywany jest mecz w Warcaby. Na potrzeby programu, klasyczne zasady gry zostały uogólnione tak, aby w grę mogły grać jednocześnie trzy osoby. Dwie spośród osób wchodzących w skład drużyny rozgrywają między sobą tradycyjną partię gry. Zadaniem trzeciej osoby jest podtrzymanie kamery tak, aby znalazła się ona nad planszą, a krawędzie planszy były równoległe do krawędzi obrazu. Dodatkowym celem grającej pary osób jest zakończenie gry, zanim osoba podtrzymująca kamerę nie będzie mogła dalej wykonywać swojego zadania z powodu bólu ręki.W przypadku braku trzeciej osoby trzymającej kamerę niezbędne jest posiadanie statywu lub innego mechanizmu umożliwiającego utrzymanie kamery nad planszą.

1.3 Uzasadnienie wyboru tematu

Temat projektu został wybrany zgodnie przez wszystkich członków zespołu, głównie ze względu na wspólne zainteresowanie grami komputerowymi oraz planszowymi. Projekt ten łączy w sobie tematykę gier, przetwarzania obrazów, programowania reguł i zasad oraz integracji różnych technologii takich jak C#, Python

oraz istniejące aplikacje webowe. Dzięki realizacji programu, członkowie zespołu mogli znacznie rozwinąć nabytą na innych zajęciach wiedzę związaną z przetwarzaniem i analizowaniem obrazu pochodzącego z internetowych kamer RGB (w szczególności tak zwanego obrazu ruchomego, składającego się z ciągłego strumienia klatek obrazu).

1.4 Moduły projektu i podział prac

Z uwagi na złożoność projektu, konieczne jest wyodrębnienie na etapie projektowania modułów, z których powinna składać się końcowa aplikacja. Projekt będzie składać się z następujących modułów:

- Moduł odpowiedzialny za zdigitalizowanie stanu planszy fizycznej.
- Moduł odpowiedzialny za weryfikację logiki gry.
- Moduł odpowiedzialny za wyświetlenie gry w aplikacji webowej.[1]

1.5 Narzędzia

Powodzenie realizacji projektu w dużej mierze zależy od zespołu, który go tworzy, jak również od narzędzi, którymi będą posługiwać się programiści oraz wszystkie osoby zaangażowane w tworzenie aplikacji. Ze względu na preferencje członków zespołu oraz ich indywidualne doświadczenie wybrane zostały następujące narzędzia:

- Język programowania C#.
  - Środowisko *Visual Studio 2017*.
  - Biblioteka „OpenCV".
- Język programowania Python.
  - Biblioteka *Selenium*.
- System kontroli wersji Github.com.
- Edytor tekstu Overleaf języka TeX (Dokumentacja).

2 Wymagania projektu

Przed przystąpieniem do prac nad projektem konieczne jest określenie wymagań, jakie powinna spełniać finalna aplikacja.

2.1 Wymagania funkcjonalne

Wymagania funkcjonalne skupiają się na funkcjonalnościach oferowanych przez aplikację takich jak:

- Cyfrowa wizualizacja rzeczywistej rozgrywki.
- Weryfikacja poprawności ruchów gracza.

[1] Uwaga: Do wizualizacji zostanie wykorzystana gotowa aplikacja www.kurnik.pl

2.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają właściwości całej aplikacji. Wynikają z potrzeb użytkownika, a także preferencji twórców. Aby aplikacja działała poprawnie konieczne jest spełnienie określonych warunków. Niezbędne wyposażenie użytkownika to:

- Kamera internetowa.
- Plansza do gry w Warcaby klasyczne (8 x 8) oraz pionki w 4 różnych kolorach, innych niż kolory pól na planszy.
- System operacyjny Windows 10 (na starszych systemach program nie był testowany).
- Monitor o rozdzielczości minimalnej 1920 x 1080.

3 Architektura rozwiązania

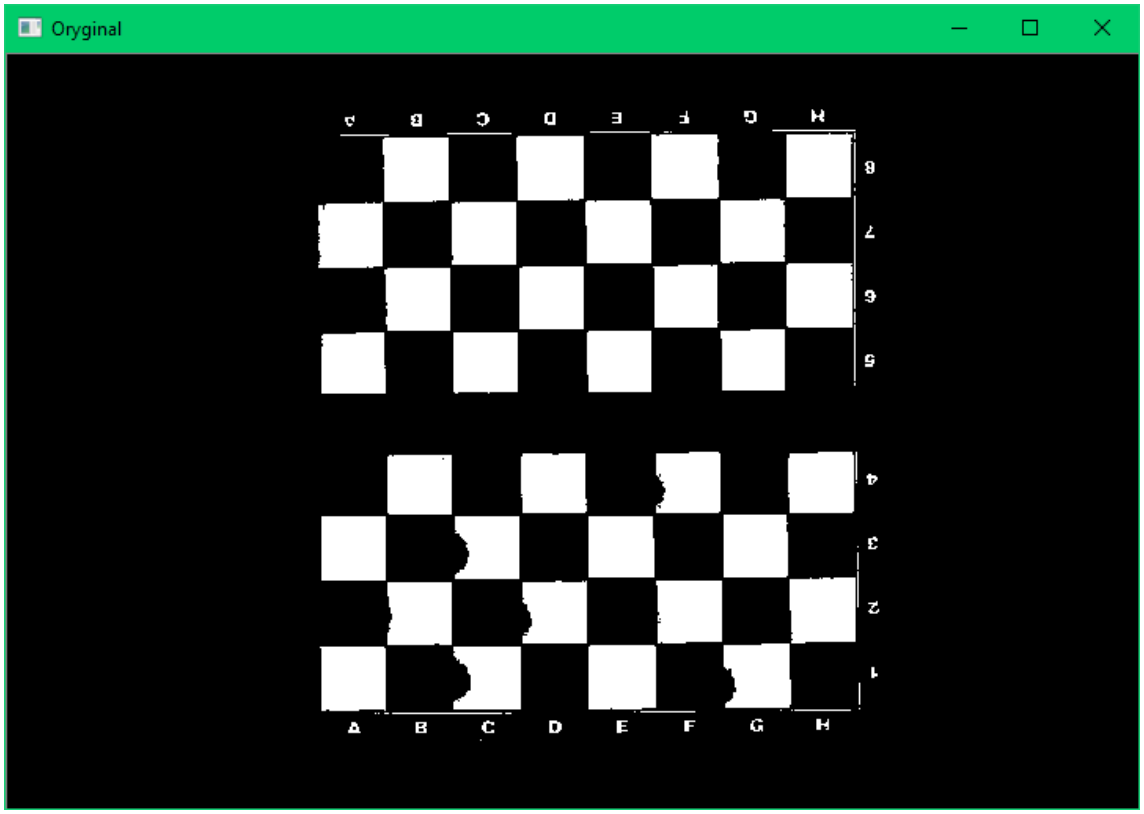
Aplikacja składa się z trzech modułów. Moduł wizyjny przy pomocy kamery odczytuje pozycje pionków na planszy i przekazuje stan planszy do modułu walidacyjnego. Po pozytywnej walidacji moduł wizyjny przekazuje dany ruch do modułu wyświetlającego rozgrywkę.

4 Opis implementacji

Program został przygotowany z wykorzystaniem środowisk *Visual Studio 2017* oraz *Visual Studio Code*, a większość testów została przeprowadzona na systemie operacyjnym *Windows 10* (gdzie zostały wykonane zrzuty ekranu, wchodzące w skład niniejszej dokumentacji).

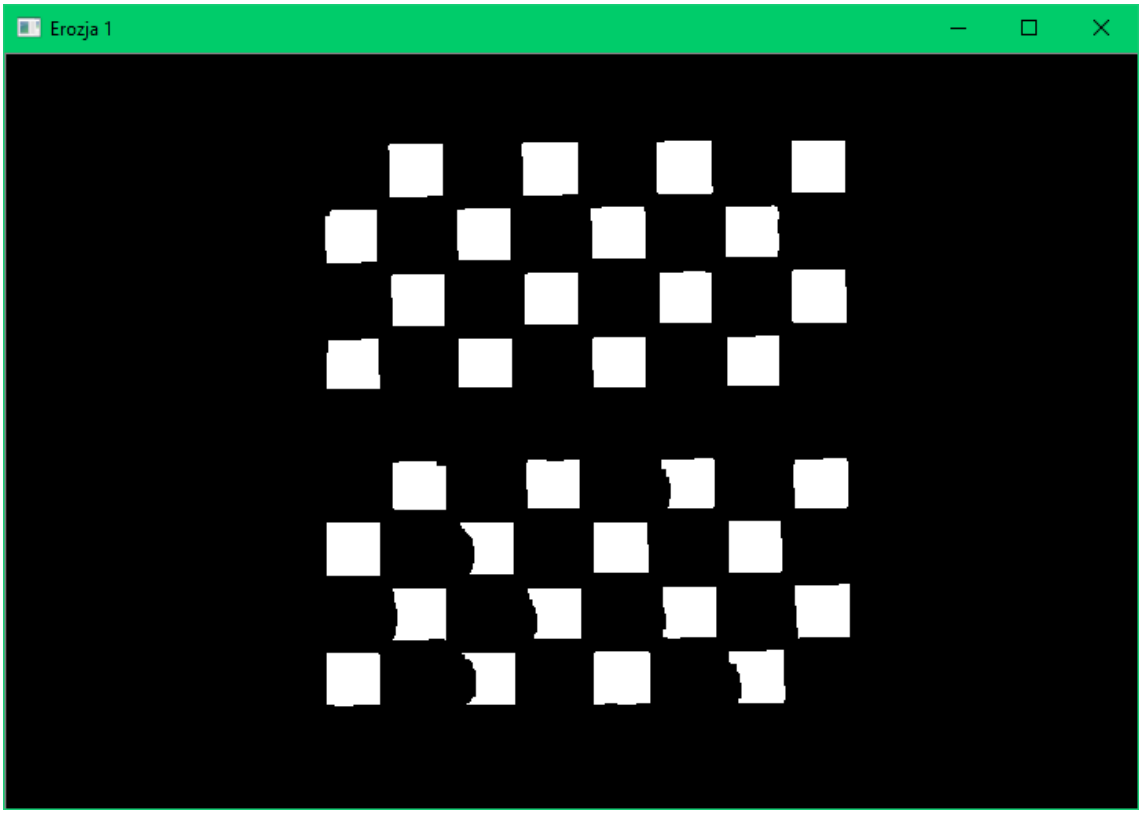
4.1 Opis implementacji modułu rozpoznawania obrazu

Najważniejszą częścią modułu rozpoznawania obrazu jest metoda AnalizujPlansze. Metoda ta przetwarza obraz kolorowy na czarno-biały, pozostawiając na nim elementy określone za pomocą parametrów przekazanych do tej funkcji. Metoda jako argumenty przyjmuje przetwarzany obraz oraz minimalną i maksymalną wartość koloru, zapisane w systemie HSV, oraz zwraca rozpoznane kontury. Na początku, z obrazu kolorowego usuwane są piksele, których kolory nie zawierają się w zakresie parametrów przekazanych do funkcji (piksele te są zerowane, a więc otrzymują kolor czarny). Następnie następuje konwersja kolorów z HSV (Hue-Saturation-Value, pol. Odcień-Nasycenie-Wartość) na skalę szarości, a później progowanie obrazu, dzięki czemu uzyskiwany jest obraz binarny, którego piksele są wyłącznie białe albo czarne. Progowanie następuje według zasady, że wszystkie czarne piksele pozostają czarne, natomiast pozostałe piksele stają się białe. Przykładowy obraz, uzyskiwany po wykonaniu powyższych operacji, przedstawiony jest na rysunku 1.



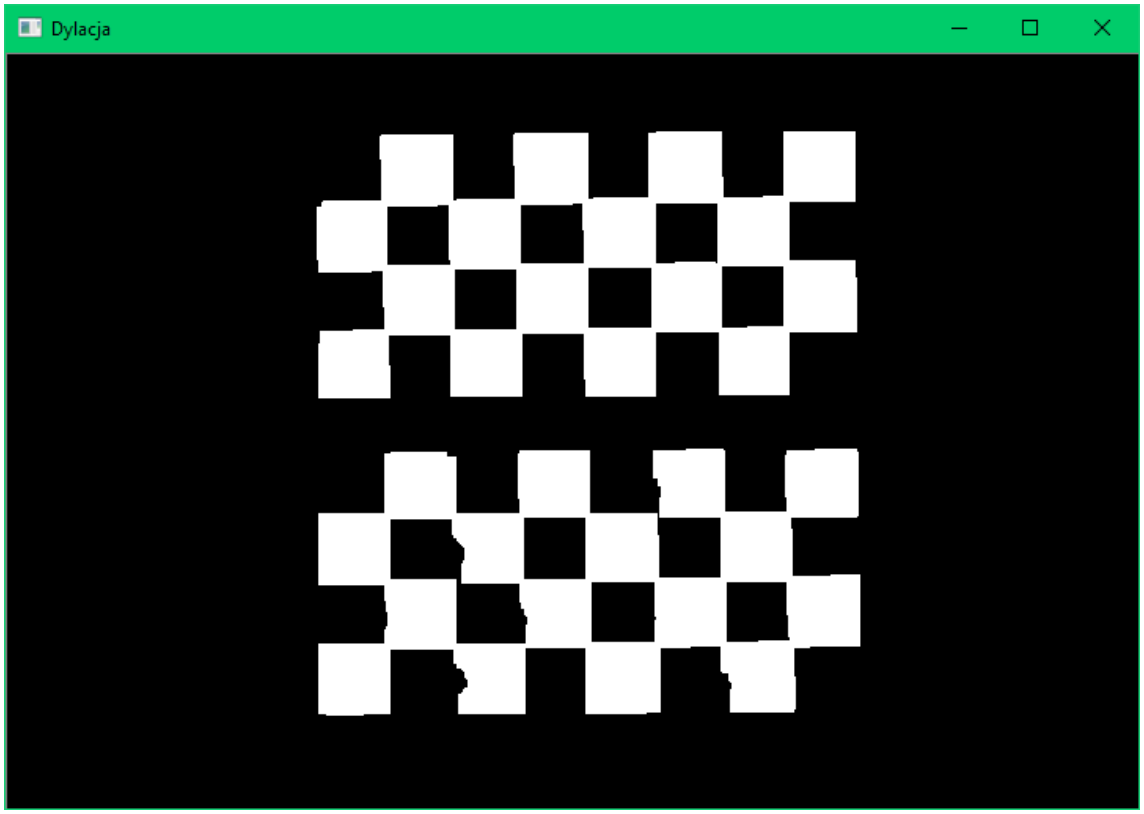
Rys. 1: Obraz z kamery po wycięciu kolorów z określonego zakresu i po progowaniu.

Jak widać, na obrazie znajduje się wiele postrzępionych krawędzi, pojedyncze piksele czarne lub niewielkie grupy takich pikseli, a także białe krawędzie planszy czy oznaczenia cyfrowe i literowe. Aby pozbyć się tych niepożądanych elementów, obraz poddawany jest procesowi erozji. W jego wyniku, z obrazu usunięte zostają niewielkie białe grupy pikseli, natomiast białe pola planszy stają się mniejsze, poza tym są od siebie oddzielone. Przykładowy wynik został pokazany na rysunku 2.



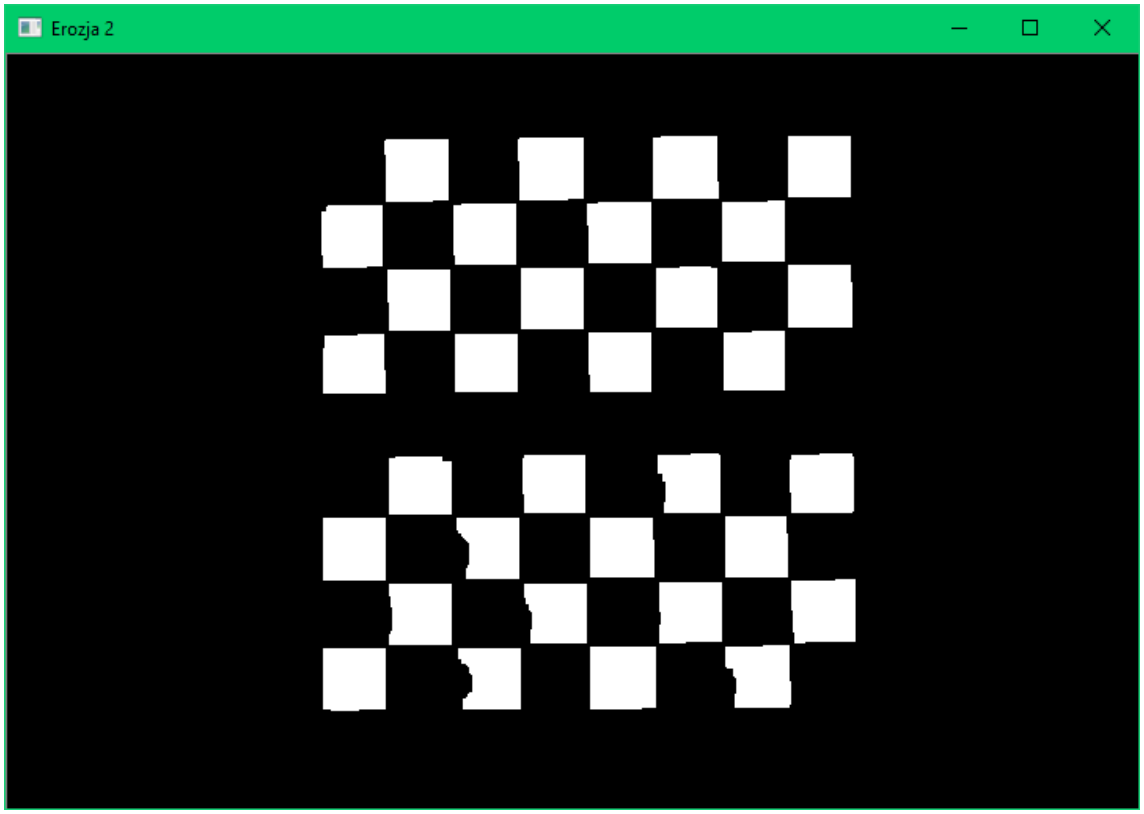
Rys. 2: Przetwarzany obraz po wykonaniu erozji.

Pomniejszenie pól jest dodatkowym, niechcianym efektem ubocznym erozji. Aby go zniwelować, obraz poddaje się procesowi dylacji. Powoduje on, że białe pola stają się większe i bardziej przypominają swoim rozmiarem rzeczywiste pola na warcabnicy. Przykładowy wynik został pokazany na rysunku 3.



Rys. 3: Przetwarzany obraz po wykonaniu dylacji.

Problemem dylacji jest fakt, że po przetworzeniu obrazu, pola są ze sobą połączone narożnikami. Utrudniłoby to znacznie (o ile nie uniemożliwiło całkowicie) prawidłowe rozpoznanie pól, dlatego obraz ponownie poddawany jest erozji, jednak już z inną wielkością elementu strukturalnego. Powoduje to oddzielenie od siebie poszczególnych pól, które zachowują rozmiar zbliżony do ich rozmiaru na rzeczywistej planszy. Obraz ten, pokazany na rysunku 4, zostaje przekazany do bibliotecznej funkcji rozpoznawania konturów. Rozpoznane kontury są zwracane przez funkcję AnalizujObraz, która w tym momencie kończy działanie.



Rys. 4: Przetwarzany obraz po drugim wykonaniu erozji.

W klasie służącej do rozpoznawania obrazu, znajdują się publiczne metody RozpoznajPola oraz RozpoznajPionki. Wywołanie pierwszej metody jest wymagane przed wywołaniem metody RozpoznajPionki, poza tym, jeśli założyć, że plansza nie przesuwa się i przez cały czas gry pozostaje w jednym miejscu, metodę RozpoznajPola wystarczy wywołać tylko raz w ciągu całej gry. Funkcja RozpoznajPionki, wywołuje metodę AnalizujPlansze, przekazując jej przedział kolorów dla pól białych. Metoda sprawdza liczbę rozpoznanych konturów — jeśli jest ona równa 32 (ilość pól jednego koloru na standardowej planszy o rozmiarze 8x8), metoda kontynuuje działanie; w przeciwnym wypadku, następuje zakończenie pracy, ponieważ nie ma ona sensu (oznacza to, że np. plansza jest nieprawidłowo oświetlona, źle skalibrowana lub użytkownik właśnie wykonuje ruch, zasłaniając ręką część planszy). Po rozpoznaniu konturów, wykonywane są złożone obliczenia matematyczne mające na celu obliczenie współrzędnych poszczególnych pól. Najpierw, na podstawie najmniejszych oraz największych wartości współrzędnych X i Y obszarów zajmowanych przez pola, obliczana jest średnia wysokość i szerokość białego pola. Wartości te (najmniejsze i największe współrzędne X i Y każdego pola) są zapisywane jako granice pól białych. Następnie, kolekcja rozpoznanych pól sortowana jest rosnąco względem najmniejszej współrzędnej Y obszaru pola. Później, w pętli następuje przetworzenie pól — w każdej iteracji przetwarzany jest jeden wiersz (4 białe pola), posortowane w kolejności rosnącej według najmniejszej współrzędnej X obszaru. W tablicy zawierającej granice pól, uzupełniane są informacje o białych i czarnych polach. Jeśli wiersz zaczyna się od czarnego pola (czyli odległość lewej krawędzi pierwszego białego pola od lewej krawędzi planszy jest w przybliżeniu równa średniej szerokości pola), pierwszemu polu czarnemu przypisywane są współrzędne X — minimalna równą współrzędnej

X lewej krawędzi planszy, a maksymalna równa minimalnej wartości X pola białego, znajdującego się po prawej stronie. Dla pozostałych pól czarnych, wartości współrzędnych X to współrzędne sąsiednich pól białych, a współrzędne Y to wartości pola białego, znajdującego się po lewej stronie. Jeśli wiersz rozpoczyna się polem białym (odległość lewej krawędzi pierwszego pola białego od lewej krawędzi planszy jest w przybliżeniu równa 0), wykonywane są podobne operacje jak powyżej, z tą różnicą, że najbardziej skrajne pole czarne po prawej stronie otrzymuje maksymalną współrzędną X równą współrzędnej X prawej krawędzi planszy. Po zrealizowaniu powyższego algorytmu, uzyskiwana jest mapa planszy, zawierająca, dla każdego białego i czarnego pola, cztery współrzędne — minimalne i maksymalne wartości X i Y, określające, że w wyznaczanym za ich pomocą zakresie znajduje się pole. Granice pól białych obliczane są na podstawie położeń krawędzi w rozpoznanym obrazie, natomiast granice pól czarnych na podstawie leżących w ich sąsiedztwie (po lewej i prawej stronie) pól białych.

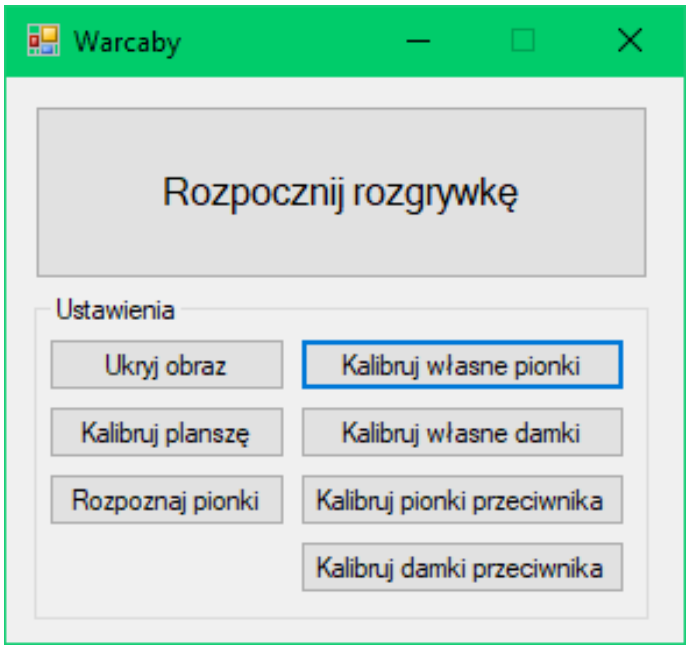
Kolejną metodą publiczną jest metoda RozpoznajPionki, która służy do wyznaczenia pozycji pionków na planszy. Funkcja wymaga wcześniejszego wywołania metody rozpoznającej pola. Opisywana metoda czterokrotnie wywołuje metodę AnalizujPlansze, dla czterech kolorów pionków (pionki zwykłe i damki obu graczy). Następnie, dla każdego rozpoznanego konturu, funkcja oblicza środek konturu. Odbywa się to za pomocą znalezienia minimalnej i maksymalnej wartości X i Y współrzędnych wchodzących w skład konturu, a następnie przez obliczenie środka prostokąta utworzonego na podstawie tych współrzędnych. W kolejnym kroku następuje przeszukanie dwuwymiarowej tablicy przechowującej granice pól i sprawdzenie, czy współrzędne środka rozpoznanego konturu pionka mieszczą się w zakresie któregoś z pól. Jeśli tak, do zbioru rozpoznanych pionków dodawany jest obiekt zawierający współrzędne tego pionka (czyli indeksy x oraz y w tablicy zawierającej pola) oraz typ pionka. Na końcu funkcji, zbiór ten jest zamieniany na tablicę i zwracany. Powyższy sposób działania, oprócz tego, że jest bardzo prosty, jest również niezawodny. Jeśli jakiś pionek znajduje się poza planszą, zostanie on pominięty w powyższym procesie, ponieważ nie mieści się na obszarze zajmowanym przez żadne z rozpoznanych pól. Oznacza to, że zbite bądź zapasowe pionki można umieścić obok planszy i nawet, jeśli znajdą się one w kadrze, nie będą mieć żadnego wpływu na działanie programu i algorytm rozpoznawania pozycji. Dotyczy to również innych elementów znajdujących się w kadrze czy niejednorodnego koloru tła. Powyższej zasady nie można jednak stosować w przypadku pól — jeśli w kadrze znajdą się ciała obce w kolorze takim, jak pola, nazywane umownie „polami białymi", spowoduje to błędy i nieprawidłowości w działaniu programu.

W opisywanej klasie znajdują się również mniej ważne, ale również użyteczne metody pomocnicze, jak np. metoda Kalibruj. Do funkcji należy przekazać parametr oznaczający typ kalibrowanego elementu. Metoda ta wyświetla okno kalibracji i podglądu obrazu z kamery i służy do wczytania i zapisania zakresów kolorów oraz do wyświetlania w sposób ciągły obrazu z kamery, przetworzonego przez metodę AnalizujObraz na podstawie aktualizowanych na bieżąco zakresów kolorów, pobranych od użytkownika. Metoda PokazObraz wyświetla obraz pobrany z kamery z nałożonymi na niego rozpoznanymi konturami. Funkcja ta, za pomocą metody AnalizujObraz, pobiera a następnie wyświetla, z użyciem linii różnych kolorów i grubości 5 pikseli, kontury pięciu elementów występujących w programie (cztery rodzaje pionków oraz pola białe). Poza tym, za pomocą linii o grubości 1 piksela, wyświetlane są

granice pól. Klasa zawiera również metody służące do zapisania i wczytania ustawień programu (zakresy kolorów dla poszczególnych elementów) oraz metodę WczytajObraz, która zajmuje się pobraniem jednej klatki obrazu z kamery i konwersją koloru z BGR na HSV.

4.2 Opis interfejsu użytkownika

Po uruchomieniu aplikacji wyświetlane jest proste okno główne, przedstawione na rysunku 5, zawierające kilka przycisków.

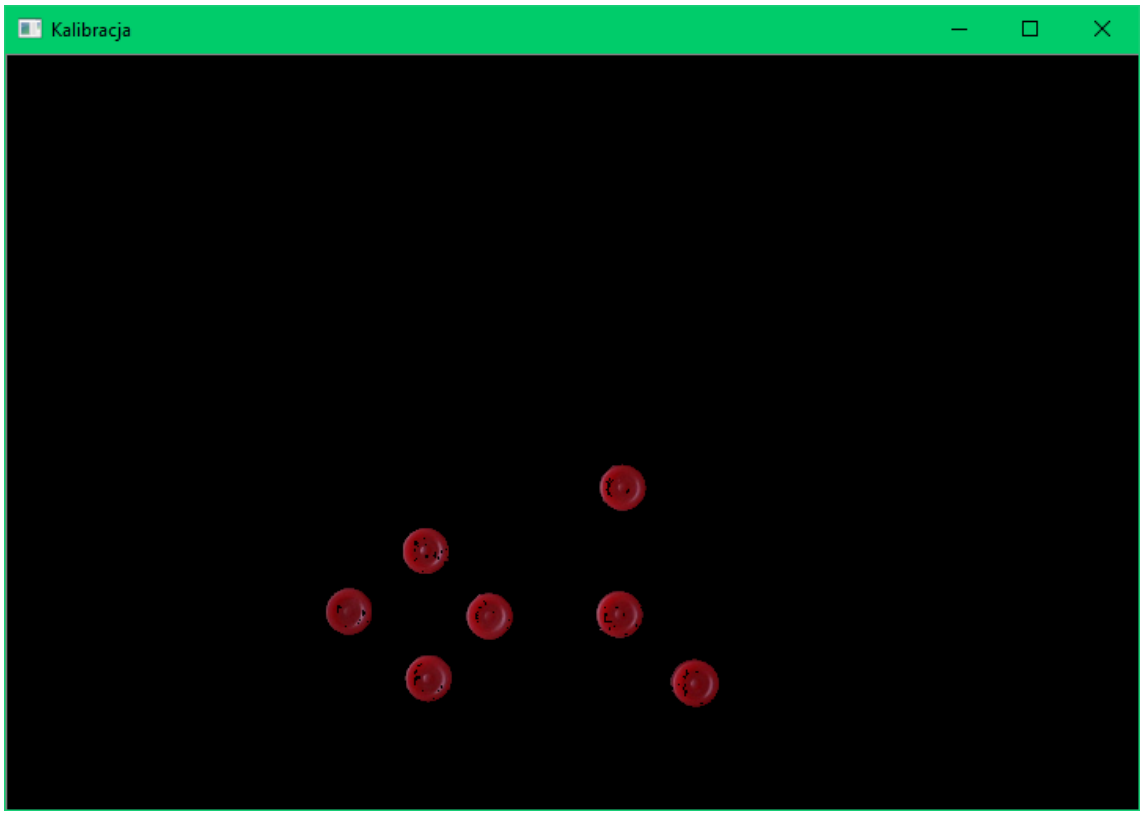


Rys. 5: Główne okno programu.

Najważniejszym przyciskiem, który, z powodu swojej wielkości, od razu rzuca się w oczy, jest przycisk służący do rozpoczęcia nowej rozgrywki. Po kliknięciu tego przycisku, w programie uruchamiana jest funkcja mająca za zadanie przygotować środowisko do wyświetlania stanu gry. Funkcja ta, między innymi, inicjalizuje moduł służący do sprawdzania poprawności ruchu oraz uruchamia dwa nowe procesy. W procesach tych rozpoczęte zostaje wykonywanie programu interpretera języka Python. W każdym z tych procesów uruchamiany jest jeden plik ze skryptem Pythona. Zadaniem skryptów jest uruchomienie przeglądarki Google Chrome, połączenie się z witryną kurnik.pl i wyświetlanie na niej rozgrywki.

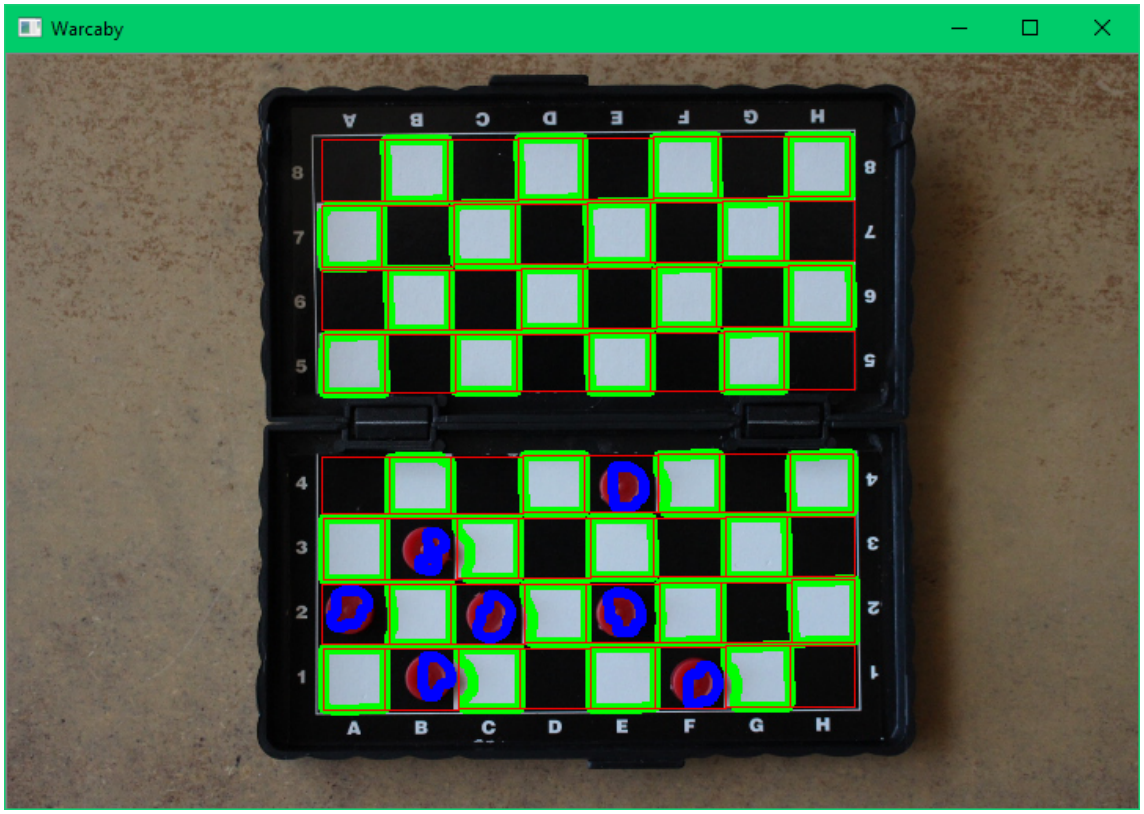
Po inicjalizacji następuje wejście do głównej pętli programu. W pętli tej najpierw następuje odczytanie z obrazu z kamery aktualnego stanu planszy i rozmieszczenia pionków. Następnie sprawdzana jest poprawność danych — czy zawartość planszy została odczytana poprawnie (nie wystąpiły problemy, takie jak np. zasłonięcie ręką planszy) oraz czy ruchy wykonane przez graczy były poprawne w sensie zasad gry. Ponieważ poprawne odczytanie obrazu potrafi sprawić wiele problemów, w celach testowych wyświetlane jest dodatkowe okienko, przedstawiające odczytany stan planszy, czyli rozmieszczenie pionków oraz ich kolory, przez co można określić, z jakiego rodzaju pionkiem program ma do czynienia (do którego gracza należy pionek oraz czy jest to zwykły pionek, czy damka).

Pod przyciskiem umożliwiającym rozpoczęcie analizowania rozgrywki, wyświetlona jest sekcja umożliwiająca skonfigurowanie programu. Na samej górze, po lewej stronie znajduje się przycisk o nazwie „Pokaż obraz” lub „Ukryj obraz” (nazwa zmienia się w zależności od stanu programu). Kliknięcie przycisku powoduje wyświetlenie okna zawierającego podgląd na żywo obrazu z kamery, np. taki jak na rysunku 6.



Rys. 6: Obraz po poprawnej kalibracji czerwonych pionków.

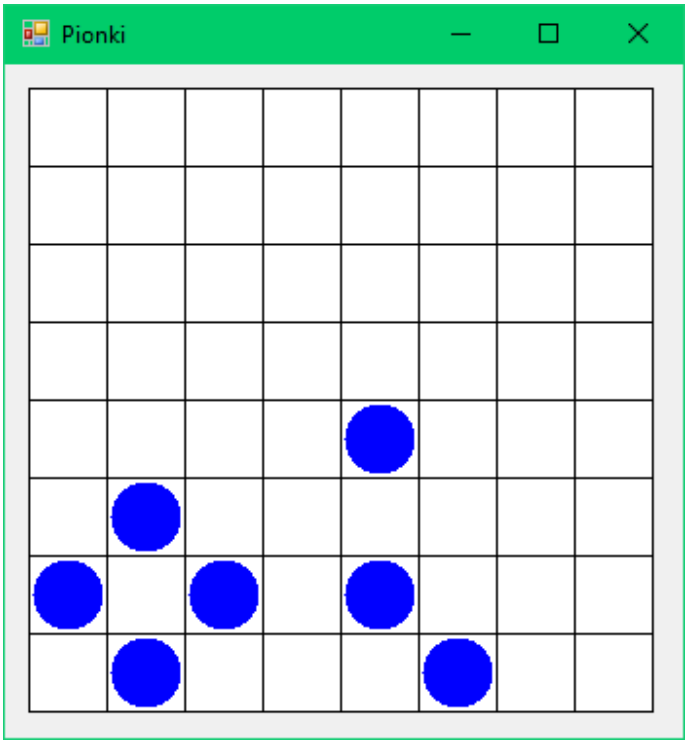
Na rysunku 7 zaznaczone są informacje o rozpoznanych konturach — różnymi kolorami zaznaczone są kontury poszczególnych pionków oraz pól białych. Poza tym, za pomocą czerwonych, cienkich linii wyświetlone zostają granice pól.



Rys. 7: Okno podglądu obrazu z zaznaczonymi konturami rozpoznanych elementów oraz z granicami pól.

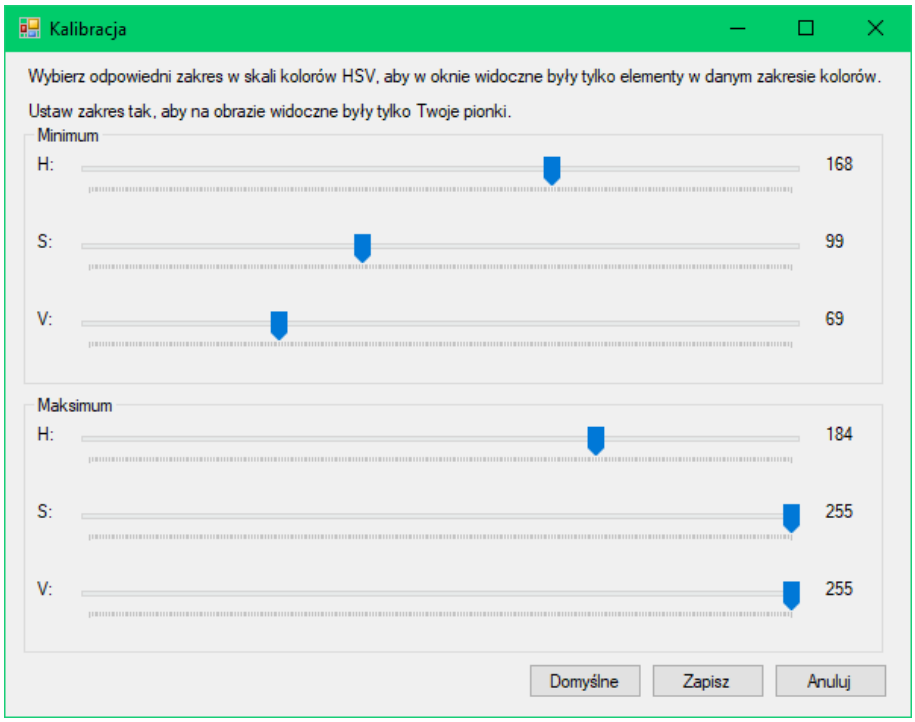
Granice te są obliczane na podstawie rozpoznanych konturów pól białych i są używane do wyznaczenia, na którym dokładnie polach znajdują się konkretne pionki. Specyfika biblioteki OpenCV powoduje, że zwykłe zamknięcie okna staje się czynnością trudną. Po kliknięciu przyciski „Zamknij", umieszczony w prawym górnym rogu okna, użytkownik klikający spodziewa się, że okno zostanie zamknięte. Tak się jednak nie dzieje. Okno wprawdzie rzeczywiście znika, jednak natychmiast pojawia się nowe, z taką samą zawartością. Dlatego, aby zamknąć okno programu, należy kliknąć przycisk Ukryj obraz, znajdujący się w oknie głównym.

Na dole lewej kolumny umieszczony został przycisk służący do rozpoznania pionków. Jeśli konfiguracja programu została przeprowadzona poprawnie i program jest w stanie wykryć pionki, po kliknięciu przycisku pojawi się okno takie jak na rysunku 8 przedstawiające aktualny stan planszy. Okno to było używane przez twórców programu do sprawdzenia poprawności implementacji algorytmów rozpoznawania obrazu, jednak jest również przydatne dla zwykłych użytkowników programu celem oceny poprawności konfiguracji.



Rys. 8: Okno wyświetlające wirtualną planszę z rozpoznanymi pionkami.

W środkowym wierszu lewej kolumny przycisków znajduje się przycisk umożliwiający kalibrację planszy. Po jego kliknięciu wyświetlone zostaje okno służące do kalibracji parametrów programu, takie jak na rysunku 9.



Rys. 9: Okno kalibracji zakresu kolorów.

Przycisk ten ma takie samo działanie, jak cztery przyciski umieszczone w prawej kolumnie — ich zadaniem jest możliwość zmiany ustawień programu. Przyciski te służą odpowiednio do kalibracji planszy (mówiąc bardziej szczegółowo, do kalibracji białych pól na warcabnicy), do kalibracji pionków jednego z graczy, damek jednego z graczy, pionków drugiego z graczy oraz damek drugiego z graczy. Po kliknięciu jednego z przycisków pojawiają się dwa okna. W jednym z nich zostaje umieszczony obraz na żywo z kamery, z zastosowaniem wprowadzanych parametrów. Drugie okno zawiera sześć suwaków, służących do ustalenia zakresu kolorów HSV.

Pierwsza grupa suwaków umożliwia wybranie dolnych zakresów parametrów, natomiast grupa położona poniżej umożliwia wybranie górnych zakresów parametrów. W oknie ponadto pojawia się komunikat, o tym, jakie dokładnie elementy należy skonfigurować. Zadaniem użytkownika jest takie ustawienie suwaków, aby na podglądzie obrazu z kamery wyświetlała się tylko konkretna grupa elementów (na przykład, pionki w jednym kolorze). W oknie konfiguracji dostępne są trzy przyciski. Pierwszy z nich (patrząc od lewej strony), umożliwia przywrócenie ustawień domyślnych. W takim przypadku dolny zakres parametrów ustawiany jest na zero, natomiast górny na wartość 255. Drugi przycisk umożliwia zastosowanie wybranych ustawień w programie oraz zamyka okna ustawień, oraz podglądu kamery, Ostatni przycisk, pozwala na anulowanie wprowadzonych ustawień i zamknięcie okien bez ich zapisywania.

Program umożliwia zapisanie ustawień konfiguracyjnych w pliku, który domyślnie pojawia się na pulpicie i posiada nazwę Ustawienia.dat. Plik ten jest plikiem binarnym i charakteryzuje się niewielkim zużyciem miejsca na dysku. W pliku zapisywane są kolejno zakresy kolorów (w skali HSV) dla poszczególnych rodzajów pionków oraz dla pól planszy. Plik ten jest zapisywany przy zamykaniu programu, natomiast wczytywany podczas uruchamiania aplikacji.

4.3 Opis implementacji logiki gry w Warcaby

Implementacja logiki gry w Warcaby została wykonana przy użyciu języka C# oraz środowiska Visual Studio 2017. Bardzo ważnym elementem projektu jest klasa Move_Detector, która wraz z metodą DetectMove, po odebraniu obrazu z kamery, porównuje go z dotychczasowym stanem planszy i określa, czy ruch został wykonany, podając współrzędne początkowe oraz końcowe, czy nie, rzucając wyjątek.

Iterując po całej planszy, spośród wszystkich pól które zmieniły swój stan (zwartość początkowa pola, różni się od zawartości końcowej), na listę możliwych pól początkowych dodawane są te, z których gracz zabrał pionek, a na listę możliwych pól końcowych dodawane są wszystkie pola, na których pojawił się pionek. Jeżeli pole zmieniło swój stan, ale nie kwalifikuje się do żadnego z powyższych przypadków (np w miejsce czerwonego pionka położyliśmy czarny), rzucany jest wyjątek o niekompatybilności stanów początkowego i końcowego planszy.

Sprawdzane jest także czy np. gracz nie wykonał żadnego ruchu. Z planszy może zniknąć wiele pionków, ale tylko jeden pionek może się pojawić. Następnie na listę prawdopodobnie przesuniętych pionków dodawane są wszystkie te, które zniknęły z danego pola, pojawił się na drugim polu, a ich kolory są takie same. Jeżeli istnieje dokładnie jeden taki pionek, to następuje symulacja, czy dany ruch w ogóle mógł być wykonany, a następnie porównanie, czy plansza wynikowa jest zgodna z planszą otrzymaną od systemu rozpoznawania obrazu z kamery. Jeśli wszystko przebiegło

pomyślnie, zwracana jest tablica współrzędnych zawierająca pozycję początkową oraz końcową pionka. Dane te przekazywane są następnie do systemu wizualizacji.

Na chwilę uwagi zasługuje również klasa Draughts_checkers, która odpowiada za całą logikę gry czyli stan planszy, kontrolę tur graczy, weryfikację poprawności ruchów czy kontrolę i znajdowanie najdłuższego bicia. Metoda Make_move weryfikuje, czy trwa tura gracza wykonującego ruch, czy na polu początkowym znajduje się pionek, a pole końcowe jest puste, czy porusza on własnym pionkiem lub czy ruch wykonywany jest tylko na polach czarnych.

Po zweryfikowaniu poprawności powyższych warunków następuje szukanie najdłuższych możliwych bić dla danego gracza, ponieważ bicie jest obowiązkowe oraz należy wykonać jedno spośród możliwych najdłuższych bić. Jeżeli gracz może wykonać bicie, następuje walidacja ruchu podanego przez gracza, a jeśli spełnia on wszystkie warunki poprawnej gry, wtedy ruch jest wykonywany. W przypadku, gdy gracz nie ma dostępnych bić, weryfikowana jest jedynie poprawność ruchu.

4.4 Opis implementacji systemu wizualizacji

System wizualizacji został wykonany w języku Python z wykorzystaniem biblioteki *Selenium*. Aby zwiększyć czytelność kodu i zaoszczędzić czas w przypadku ewentualnych zmian struktury aplikacji webowej zastosowany został wzorzec projektowy *Page Object Pattern*. Aplikacja została stworzona na wzór architektury klient - serwer. Serwer jest odpowiedzialny za inicjalizację rozgrywki w tym utworzenie pokoju, w którym rozgrywany będzie mecz a także za wysłanie zaproszenia do klienta. Zadaniem klienta jest dołączenie do już utworzonego pokoju i zaakceptowanie zaproszenia wysłanego przez serwer. Moduł wizualizacji wywoływany jest przez moduł rozpoznawania obrazu i po pozytywnej walidacji ruchu wykonywanej przez moduł walidacji przekazywane są mu współrzędne x, y $\in <0,7>$. Aby ruch został wykonany należy podać współrzędne początkowe i współrzędne końcowe.

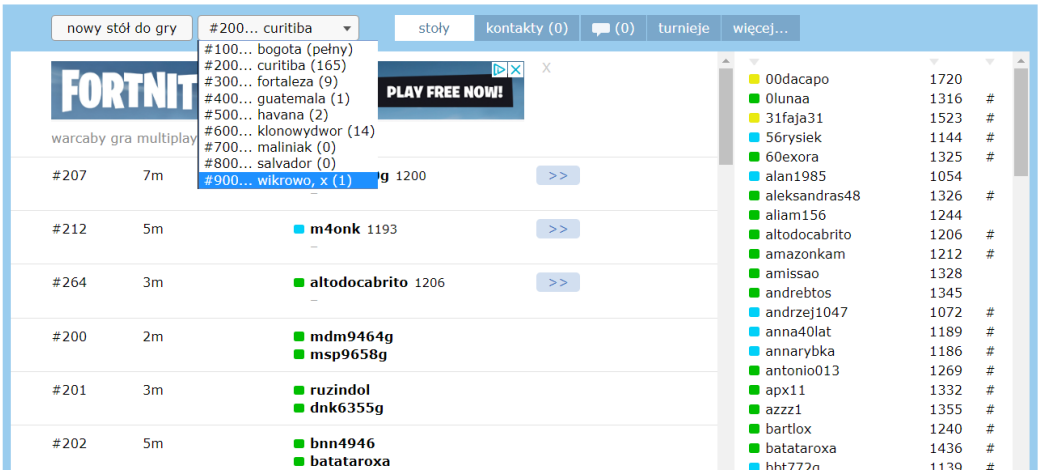
Przejdźmy jednak do szczegółów implementacji. Cały proces rozpoczynany jest przez otworzenie przeglądarki ze stroną *https://www.kurnik.pl/warcaby*. Na rysunku 10 podane zostały login i hasło serwera. Kilka początkowych kroków jest identycznych zarówno dla klienta jak i dla serwera. Poszczególne kroki wykonują się równolegle - to znaczy tworzone są dwie instancje przeglądarki.

Rys. 10: Ekran logowania.

Rysunek 11 przedstawia sytuację po zalogowaniu. Button *START-WARCABY* musi zostać wybrany aby przejść dalej. Ten krok jest identyczny zarówno dla klienta jak i dla serwera.

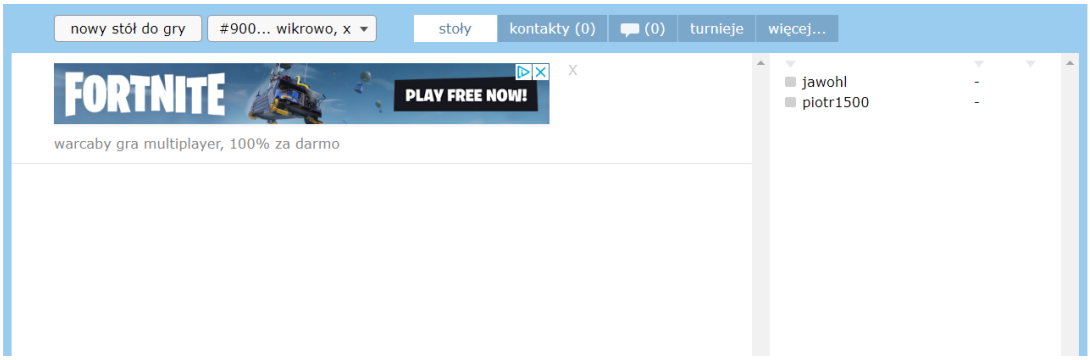
Rys. 11: Ekran po zalogowaniu.

Na rysunku 12 z dropdown'u wybierany jest rodzaj pokoju na *#900... wikrowo*. Na tym etapie kończą się kroki identyczne dla klienta i dla serwera.



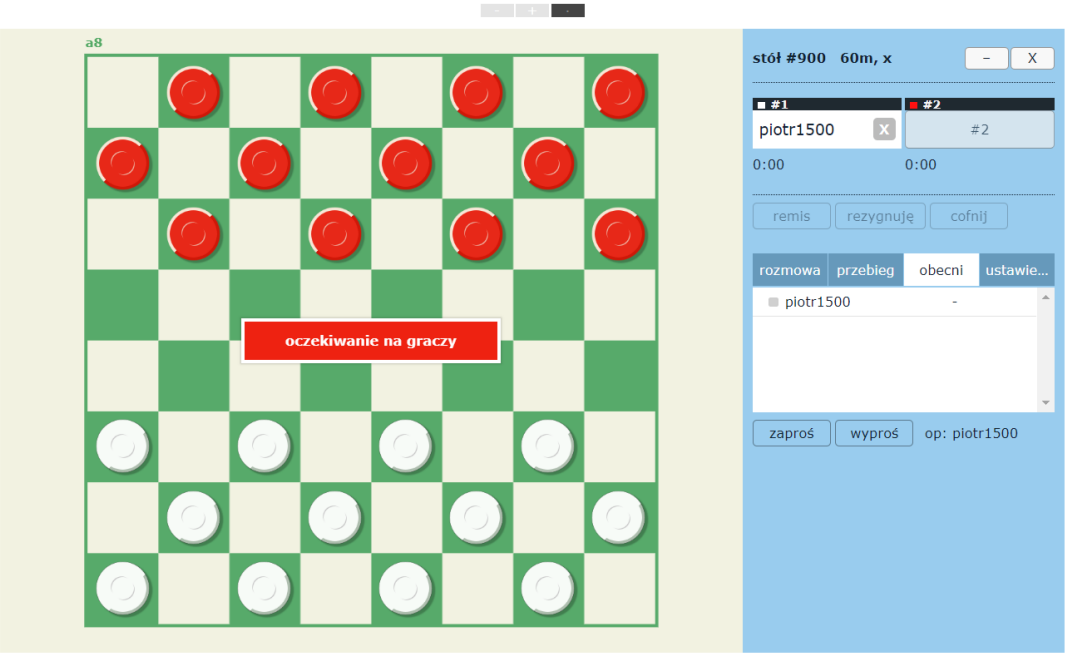
Rys. 12: Ekran wyboru rodzaju pokoju.

Na obecnym etapie serwer poprzez button *nowy stół do gry* tworzy pokój, w którym rozgrywany będzie mecz. Jest to również miejsce oczekiwania na zaproszenie przez klienta od serwera.



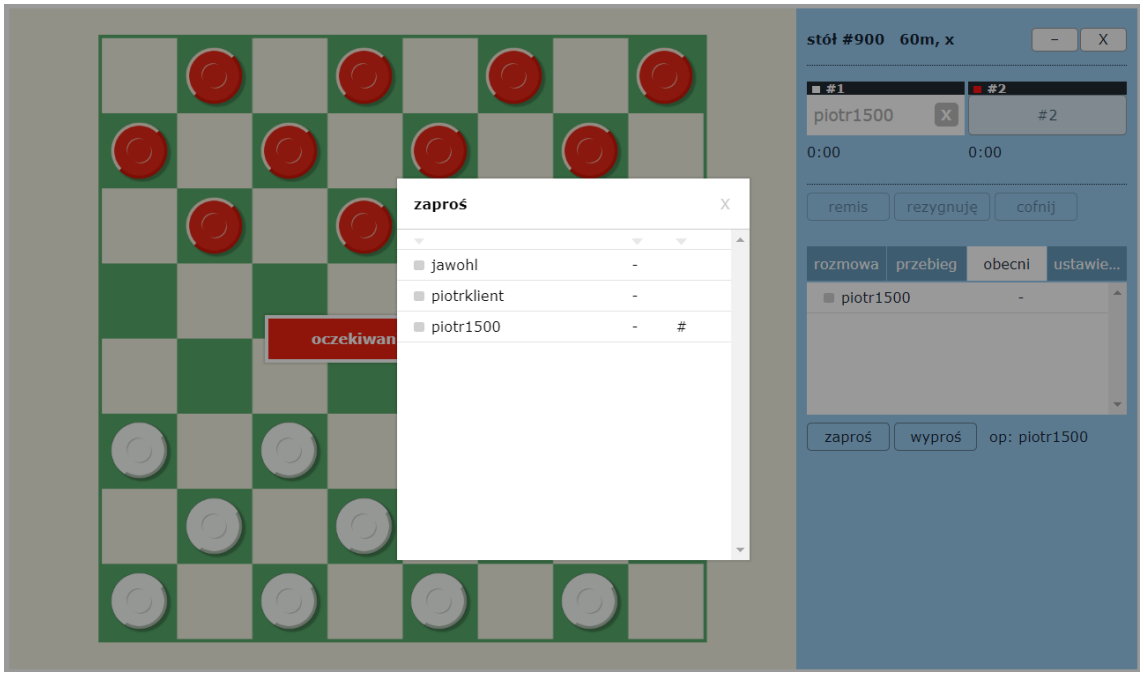
Rys. 13: Ekran tworzenia nowego pokoju.

Na rysunku 14 przedstawiono etap, podczas którego serwer wybiera stronę rozgrywki, a później poprzez button *zaproś* zaprasza *piotrklient* do gry. Dodatkowo sprawdza czy żaden inny gracz nie zajął miejsca oczekiwanego *piotrklient*. Gdyby tak się stało - wyrzuca go.



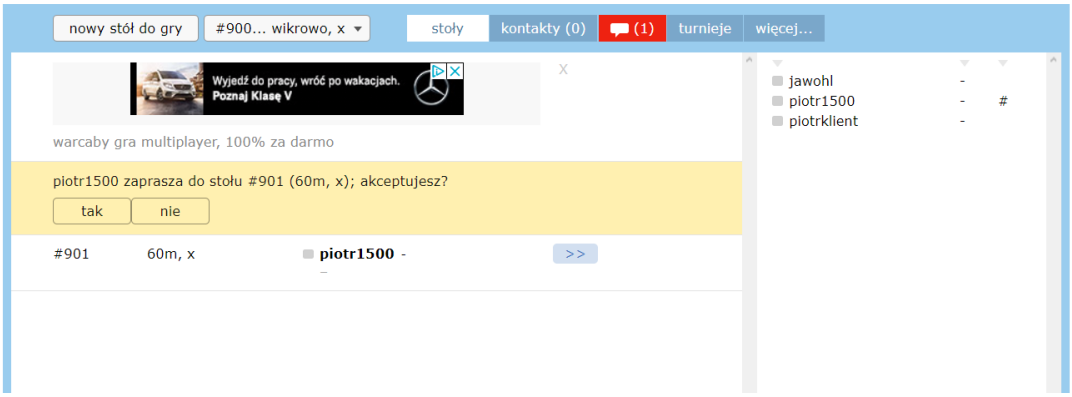
Rys. 14: Ekran po utworzeniu pokoju.

Na rysunku 15 przedstawiono listę graczy, którzy obecnie mogą zostać zaproszeni przez serwer do aktualnej rozgrywki. Naturalnym jest, że do rozgrywki zostanie zaproszony *piotrklient*.



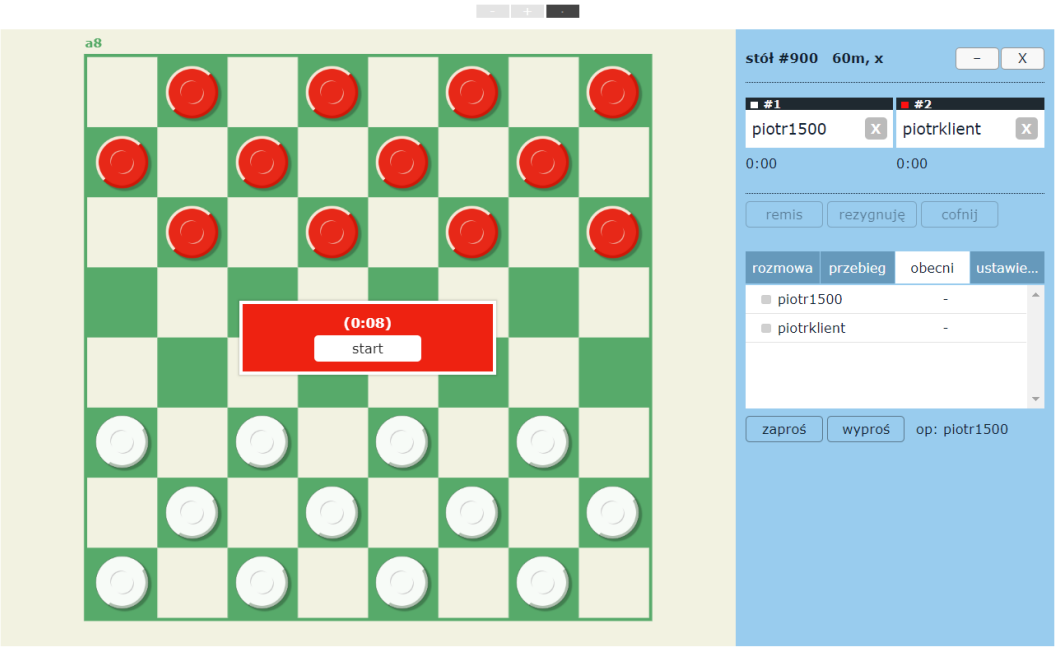
Rys. 15: Ekran wyboru gracza do rozgrywki.

Rysunek 16 przedstawia ekran klienta w momencie otrzymania zaproszenia od serwera. Klient może odrzucić zaproszenie lub z niego skorzystać. Oczywiście przyjmuje je, aby rozgrywka była możliwa.



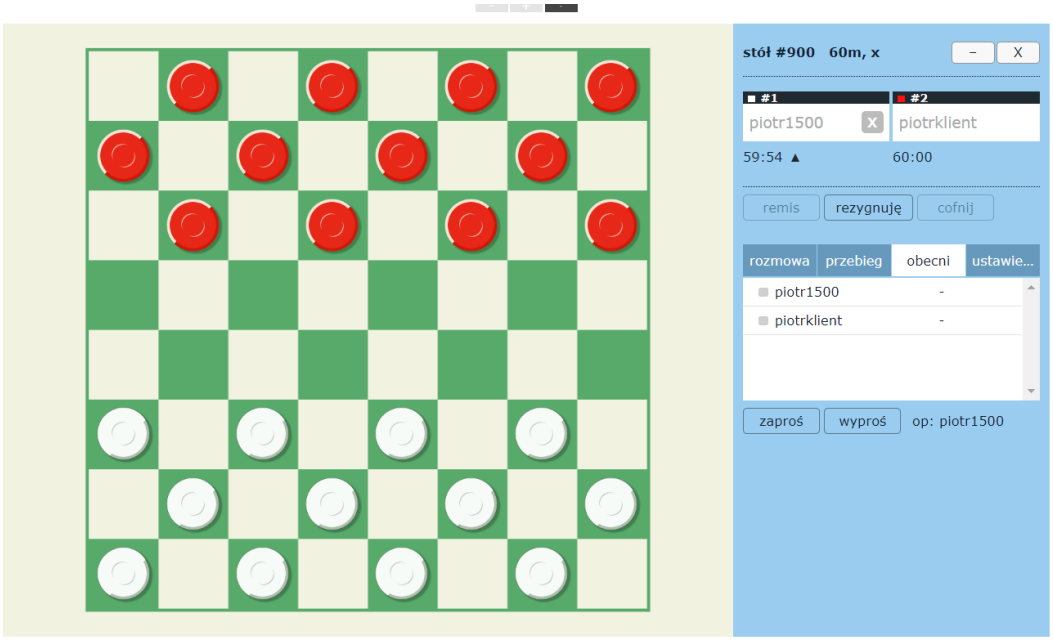
Rys. 16: Ekran klienta w momencie dostania zaproszenia.

Rysunek 17 przedstawia sytuację w momencie przyjęcia zaproszenia przez klienta. Serwer i klient mają 10 sekund na potwierdzenie gotowości do rozegrania partii. Po potwierdzeniu gotowości rozgrywka będzie możliwa.



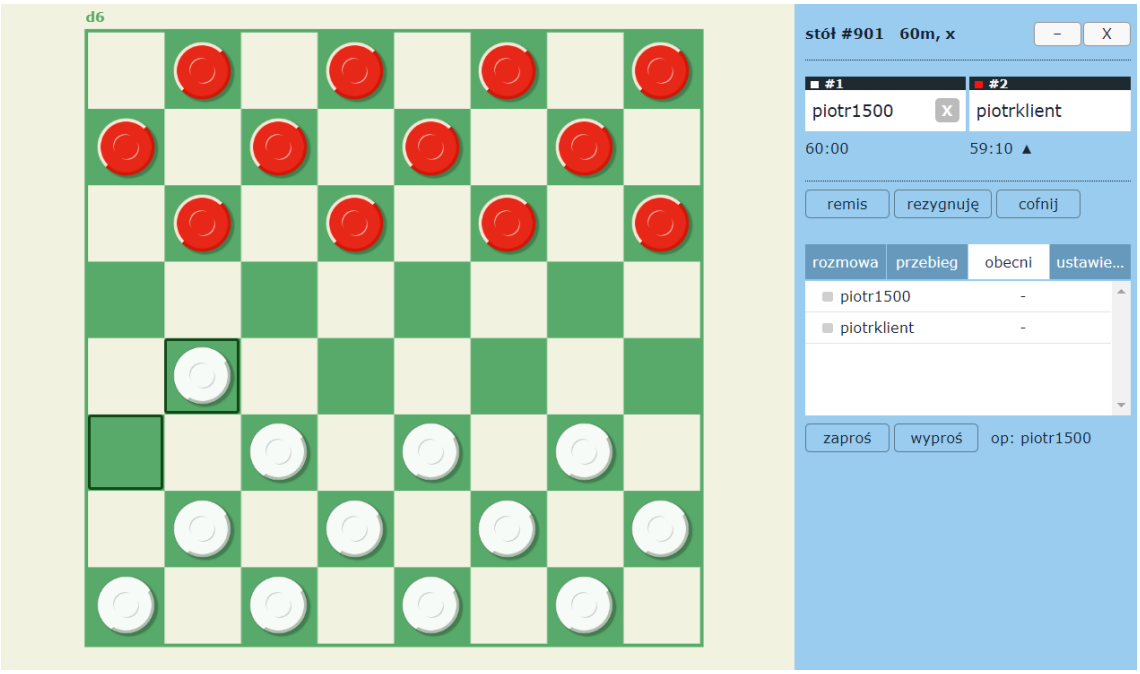
Rys. 17: Ekran po przyjęciu zaproszenia przez klienta.

Rysunek 18 przedstawia sytuację kiedy obie strony potwierdziły chęć rozgrywki. W tym momencie moduł oczekuje na współrzędne aby wizualizować rozgrywkę.



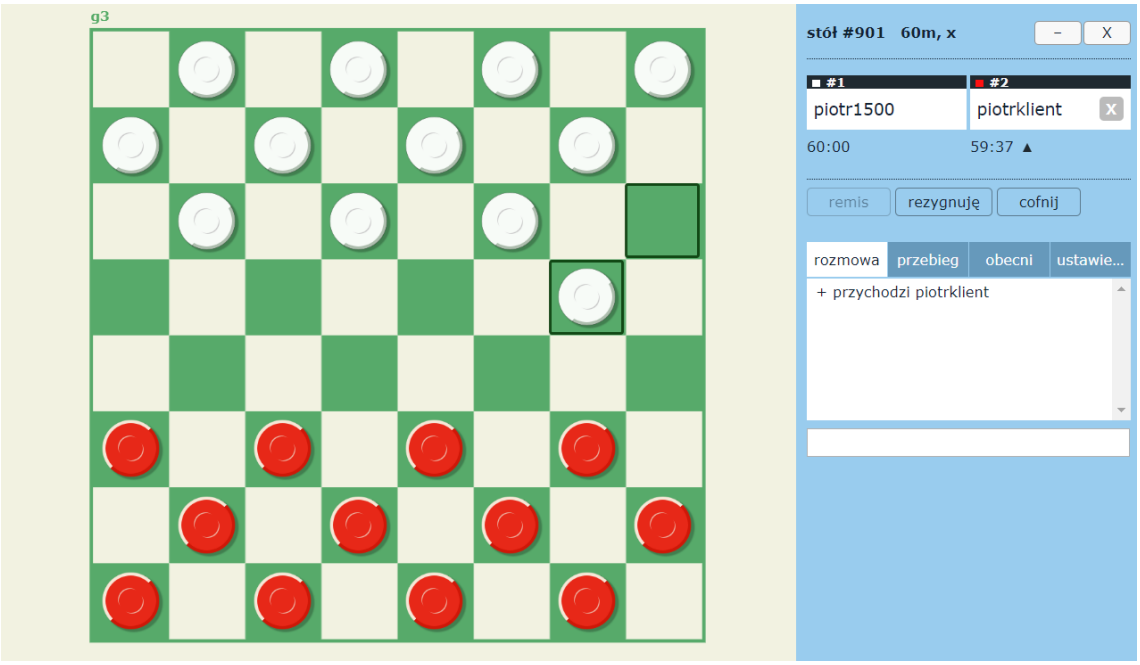
Rys. 18: Ekran po pomyślnym nawiązaniu sesji.

Rysunek 19 przedstawia planszę po wykonaniu ruchu gracza z białymi pionkami. Istotnym elementem implementacyjnym jest fakt iż chcąc wykonać ruch podajemy współrzędne z punktu widzenia gracza wykonującego ruch.



Rys. 19: Widok z perspektywy serwera - białe pionki.

Rysunek 20 przedstawia planszę z perspektywy klienta. Istotnym elementem implementacyjnym jest fakt iż chcąc wykonać ruch podajemy współrzędne z punktu widzenia gracza wykonującego ruch.



Rys. 20: Widok z perspektywy klienta - czerwone pionki.

5 Instrukcja użytkowania aplikacji

Aby użytkownik mógł korzystać z aplikacji konieczne jest dostarczenie mu instrukcji obsługi programu. Podczas pierwszego kontaktu z aplikacją ważne jest, by użytkownik wiedział, jakie czynności musi wykonać, ale również by rozumiał, dlaczego je wykonuje oraz jaki mają one wpływ na działanie programu. Użytkownik regularnie korzystający z aplikacji nie potrzebuje pełnego opisu wszystkich czynników, a jedynie prostą listę niezbędnych kroków wymaganych do uruchomienia programu. W tym celu stworzone zostały pełna instrukcja obsługi aplikacji oraz skrócona instrukcja obsługi aplikacji.

5.1 Pełna instrukcja obsługi aplikacji

Przed uruchomieniem programu, należy przygotować środowisko służące do gry. Należy przejść w miejsce, gdzie istnieje możliwość równomiernego oświetlenia planszy (na przykład do piwnicy, w której zainstalowane są odpowiednie źródła światła). Nie należy kierować żadnego źródła światła w stronę planszy — może to spowodować, że na planszy pojawią się odbicia, poza tym plansza będzie oświetlona nierównomiernie. W różnych miejscach planszy będzie panować różne natężenie oświetlenia, w efekcie czego elementy, które teoretycznie są w takim samym kolorze (na przykład, identyczne czerwone pionki), będą mieć inne odcienie kolorów lub nawet inne kolory, przez co kolory teoretycznie takich samych elementów zostaną rozpoznane jako znacząco różniące się, przez co może nie być możliwe prawidłowe analizowanie planszy przez program. Najlepszym rozwiązaniem jest zastosowanie silnego źródła światła i rozproszenie promieni świetlnych w pomieszczeniu. Wszelkie cienie czy inne problemy ze światłem mogą wpływać na niepoprawne działanie programu.

Przed przystąpieniem do kolejnych działań zainstaluj Pythona 3.x a także bibliotekę *Selenium*. Folder *Display* służący do wizualizacji rozgrywki należy umieścić w ścieżce *C:\Users\Piotr\Desktop\Display*. Dodatkowo w folderze *Display* powinien znajdować się aktualny plik *ChromeDriver.exe*. Jest on niezbędny do działania modułu i należy pobrać go ze strony *http://chromedriver.chromium.org*.

Do rozgrywki należy przygotować odpowiednią planszę oraz pionki. Wymagane jest, aby połowa pól na planszy (w klasycznych warcabach mających kolor biały) oraz pionki miały różne kolory, tak aby możliwe było rozpoznanie ich na podstawie kolorów. Wymagane jest więc, aby w kadrze kamery znalazły się elementy o co najmniej sześciu różnych kolorach — pola na warcabnicy, cztery kolory pionków (pionki obu graczy oraz należące do tych zawodników damki) oraz co najmniej jeden dodatkowy kolor, w którym będą pozostałe pola na warcabnicy oraz tło (na przykład blat stołu lub inny element, na którym została umieszczona plansza). Zalecanym sposobem przygotowania stanowiska jest zastosowanie planszy posiadającej pola białe i czarne, pionków w czterech różnych kolorach oraz czarnego tła. Przed uruchomieniem programu, na planszy należy rozmieścić (w dowolnym porządku) co najmniej po jednym pionku w każdym kolorze, a nad planszą należy umieścić kamerę RGB (tak, aby krawędzie planszy były prostopadłe do krawędzi kadrów obrazu). Po wykonaniu czynności, na komputerze można uruchomić prezentowane oprogramowanie w celu przeprowadzenia kalibracji oraz oficjalnego uruchomienia programu.

Kamerę nad planszą należy ustawić tak, aby krawędzie planszy były równoległe do krawędzi obrazu przechwytywanego z kamery. Kolejnym warunkiem jest aby

w lewym górnym rogu planszy znajdowało się pole białe. Po uruchomieniu programu wyświetli się okno główne aplikacji. W oknie tym można dokonać konfiguracji programu, tak aby dostosować go do posiadanej kamery oraz do panujących aktualnie warunków oświetleniowych. Na początku, należy przeprowadzić kalibrację obrazu przez kliknięcie na przyciski „Kalibruj planszę", „Kalibruj własne pionki", „Kalibruj pionki przeciwnika" oraz „Kalibruj damki przeciwnika". Po kliknięciu każdego z tych przycisków wyświetlą się dwa okna — jedno zawiera podgląd obrazu z kamery, natomiast drugie suwaki umożliwiające zmianę parametrów. Zadaniem użytkownika jest takie ustawienie suwaków, aby na obrazie z kamery pozostały widoczne tylko wybrane elementy (pionki oraz damki, w przypadku kalibracji planszy białe pola na warcabnicy). Program umożliwia zapisanie wprowadzonych ustawień, dzięki czemu nie trzeba ich zmieniać po każdym uruchomieniu programu (jednak oczywiście, jeśli program zostanie ponownie uruchomiony w innych warunkach oświetleniowych, wtedy wybrane ustawienia mogą stać się nieaktualne, przez co może wystąpić konieczność ponownej konfiguracji programu).

W oknie konfiguracji istnieje też możliwość wyjścia z okna bez zapisywania konfiguracji lub przywrócenia ustawień domyślnych, jeśli użytkownik zgubi się podczas wprowadzania ustawień i nie będzie w stanie znaleźć prawidłowych wartości. Po poprawnej konfiguracji każdego z pięciu elementów, istnieje możliwość przetestowania wprowadzonych parametrów. W tym celu, należy kliknąć przycisk „Pokaż obraz". Zostanie wtedy wyświetlony podgląd z kamery z zaznaczonymi konturami rozpoznanych elementów. Aby zamknąć okno podglądu, należy kliknąć przycisk „Ukryj obraz" (jest to ten sam przycisk, który posłużył do otwarcia okna, jednak ze zmienionym opisem wyświetlanym na elemencie). Okna wyświetlane przez bibliotekę graficzną nie zamykają się przy próbie zamknięcia ich przyciskiem „X" znajdującym się w prawym górnym rogu ekranu. Można również wybrać i kliknąć przycisk „Rozpoznaj pionki", dzięki czemu powinno wyświetlić się okno wyświetlające zawartość planszy. Jeśli okno nie zostanie wyświetlone lub rozmieszczenie pionków nie zgadza się z rzeczywistym stanem planszy, oznacza to, że kalibracja nie została przeprowadzona poprawnie. W takim przypadku, należy powtórzyć proces kalibracji.

Jeśli wszystkie operacje powiodą się, należy przygotować planszę do rozgrywki. Na planszy należy rozmieścić pionki na czarnych polach tak, by na górze planszy znajdowały się pionki gracza rozpoczynającego rozgrywkę, a na dole pionki drugiego gracza. Należy ponownie sprawdzić, czy w lewym górnym rogu planszy znajduje się białe pole.

Dopiero teraz można kliknąć na duży, wyróżniający się przycisk „Rozpocznij rozgrywkę". Po chwili uruchomią się dwa okna przeglądarki Google Chrome, w których wczytana zostanie strona www.kurnik.pl, służąca do wizualizacji rozgrywki. W oknach przeglądarki, aplikacja zaloguje się na dwa różne konta użytkownika we wspomnianym serwisie, po czym nastąpi połączenie wspomnianych dwóch użytkowników. Od tego momentu możliwa będzie gra w Warcaby na fizycznej planszy. Przesuwanie pionki zostaną rozpoznane przez program oraz wyświetlone w serwisie kurnik.pl.

5.2 Skrócona instrukcja obsługi aplikacji

Skrócona instrukcja obsługi aplikacji przedstawia krótko najważniejsze punkty użytkowania programu. Przeznaczona jest ona dla użytkowników, którzy uprzednio

zapoznali się z pełną instrukcją obsługi programu.

1. Przygotuj planszę do gry w Warcaby oraz pionki.
2. Zadbaj o równomierne oświetlenie planszy.
3. Folder *Display* umieść w ścieżce *C:\Users\Piotr\Desktop\Display*.
4. Sprawdź czy w folderze *Display* znajduje się aktualny plik *ChromeDriver.exe*.
5. Upewnij się, że używasz Pythona w wersji 3.6.
6. Upewnij się, że zainstalowałeś bibliotekę *Selenium*.
7. Ustaw kamerę nad planszą tak, by z perspektywy kamery, na górze znajdowały się pionki „białe" (pionki gracza, który wykonuje pierwszy ruch), a na dole pionki przeciwnika (który wykonuje ruch jako drugi).

6 Podsumowanie

Celem pracy było stworzenie zautomatyzowanego systemu umożliwiającego wizualizację rozgrywki w Warcaby. Zadanie udało się wykonać całkowicie. Istotnym i bardzo rozwijającym elementem naszej pracy była integracja wielu technologii takich jak programy napisane w językach C#, Python czy gotowych aplikacji webowych. Wbrew naszym przewidywaniom nie przysporzyło to większych problemów, co potwierdza czas integracji naszych modułów, który wyniósł 2.5 godziny. Elementem dodatkowym, który w przyszłości warto byłoby dodać, jest automatyczna konfiguracja doboru parametrów HSV progowania obrazu w celu prawidłowego rozpoznawania pionków oraz planszy.

6.1 Napotkane problemy

Realizowany program służy do analizy i rozpoznawania obrazu z kamery. Zadanie to wydaje się łatwe, zwłaszcza że użyta zostaje gotowa biblioteka OpenCV. Okazuje się jednak, że przetwarzanie obrazu jest procesem bardzo złożonym i skomplikowanym. Jednym z problemów jest wybór sprzętu. Stosowane powszechnie kamery internetowe RGB potrafią zapewnić obraz o jedynie przeciętnej jakości. Ponadto, same kamery mają nieszczególnie zadowalające parametry sprzętowe. Między innymi, podczas nagrywania obrazu w pomieszczeniu, gdzie jest niezbyt duże natężenie światła, na obrazie pojawia się wyraźny szum ISO. Do tego dochodzi fakt, że obraz przesyłany między kamerą a komputerem za pośrednictwem portu Universal Serial Bus (pol. Uniwersalna Magistrala Szeregowa) jest kodowany za pomocą algorytmów kompresji stratnych, z zastosowaniem niskich parametrów (np. niska wartość prędkości transmisji bitów).

Wszystko to powoduje, że obraz, który trafia do algorytmów programu, jest mocno zaszumiony. Powoduje to trudności z rozpoznawaniem obrazu — jeśli np. na planszy znajduje się pionek w jednolitym, czerwonym kolorze, w przetwarzanym obrazie sąsiednie piksele będą różnić się kolorem. W efecie czego, podczas procesu kalibracji nie da się ustawić parametrów tak, aby z obrazu wyciąć inne piksele niż piksele pionków. Na kalibrowanych pionkach pojawiają się czarne, wycięte piksele,

które posiadają inny kolor niż rzeczywisty kolor pionka (z powodu szumów i strat związanych z kompresją).

Kolejnym problemem jest wysoki poziom abstrakcji bibliotek i automatyczna konfiguracja parametrów kamery. Ustawienia automatyczne są bardzo przydatną funkcjonalnością urządzeń optycznych, takich jak kamery czy aparaty cyfrowe. Dzięki nim, nie trzeba znać się zbyt dobrze na fotografii, ani, w przypadku posiadania takich umiejętności, nie trzeba poświęcać dużej ilości czasu na ustawienie wszelkich parametrów i można szybko wykonać zdjęcie. Fotografowie zapewne zgodzą się z twierdzeniem, że bardzo przydatna jest możliwość ręcznej zmiany parametrów w używanym urządzeniu, dzięki czemu można zrobić udane zdjęcie, w przypadku gdy automatyka zawodzi lub ustawia parametry poprawne, lecz niechciane przez fotografa (jeśli np. fotograf chce uzyskać na zdjęciu specjalne efekty). Podczas tworzenia programu, zespół programistów spotkał się z problemem automatycznej konfiguracji parametrów optycznych internetowej kamery RGB. Jeden z członków zespołu próbował przeprowadzić konfigurację obrazu pochodzącego z kamery. Chwilę przed tym, zanim konfiguracja została ostatecznie wybrana, w kamerze zmienił się balans bieli. Z tego powodu, wszystkie elementy na obrazie miały zupełnie inne odciennie kolorów. Spowodowało to, że precyzyjnie ustawiane parametry stały się bezużyteczne i konieczna stała się ponowna konfiguracja programu.

Najważniejszym czynnikiem, wpływającym na jakość działania programu jest oświetlenie. Jego istnienie oraz kolor, natężenie i właściwości światła są w stanie w drastyczny sposób wpłynąć na działanie programu. Załóżmy, że na stole wewnątrz pomieszczenia leży książka w białej okładce. Poszczególne punkty wchodzące w skład powierzchni okładki znajdują się w różnej odległości od okna. Na książkę pada rozproszone światło, wpadające do pokoju przez odsłonięte okno. W oknie znajduje się roślina liściasta, która częściowo zasłania wpadające światło. Wszystko to powoduje, że okładka jest oświetlona w sposób wysoce nierównomierny. Człowiek na takie różnice nie zwraca uwagi, dla niego książka ma po prostu białą okładkę. Oprogramowanie komputera nie potrafi jednak stwierdzić, jaki kolor w rzeczywistości ma dany obiekt, i czy różnica koloru wynika np. z posiadania przed przedmiot zróżnicowanego koloru, czy po prostu na różne punkty obiektu pada światło o różnym natężeniu, ponadto pojawiają się cienie.

Wszystko to powoduje, że analizowane obiekty muszą być oświetlone równomiernie, nie mogą posiadać żadnych różnic w oświetleniu ani cieni. Poza tym, parametry światła nie mogą zmieniać się w czasie - jeśli skonfiguruje się program dla pewnych wartości oświetlenia, zmiana tych wartości powoduje, że program traci możliwość wykrycia pionków (program np. nie będzie mógł rozpoznać innych odcieni skonfigurowanych kolorów) Uzyskanie takich warunków może być bardzo trudne - nie powinno się polegać na naturalnym świetle, ponieważ jest ono zmienne w czasie. Powodem jest ruch obrotowy Ziemi wokół własnej osi. Sprawia to, że w ciągu doby plansza jest oświetlana światłem o różnym kolorze oraz o natężeniu, który zależy od pory dnia czy położenia planszy w przestrzeni względem kierunków świata czy innych przedmiotów, mogących wpływać na jakość i kierunek padania światła. Uniemożliwia to poprawne ustawienie parametrów programu lub powoduje konieczność częstej ich zmiany. Dlatego zaleca się oświetlenie planszy i pionków za pomocą sztucznego źródła światła, które jest niezmienne, i niedopuszczenie do pomieszczenia, w którym odbywa się eksploatacja programu, światła pochodzącego z innych, niestabilnych źródeł. Źródło światła powinno być tak umieszczone i skierowane, aby

w odpowiedni sposób oświetlało warcabnicę oraz pionki i nie wprowadzało zbędnych zakłóceń do obrazu.

W końcowej fazie pracy nad projektem pojawił się problem dotyczący modułu wizualizacji. Okazało się, że jego działanie stało się niemożliwe ze względu na zmianę struktury serwisu - www.kurnik.pl. Dzięki zastosowaniu wzorca projektowego *Page Object Pattern* naprawienie problemu nie zajęło wiele czasu. Poprawki dotyczyły lokalizowania elementów webowych, czyli w praktyce niezbędna była tylko zmiana dwóch ścieżek xpath.

6.2 Dlaczego C++ jest lepszy od C#?

Podczas wyboru narzędzi projektowych niezbędnych do realizacji zadania, zespół miał duży dylemat, czy wybrać język C++, czy C#. Zdecydowano się ostatecznie na rozwijanie umiejętności programowania w języku C# z chęci poznania różnic względem języka C++ oraz utwierdzenia się w myśli, że język C++ jest lepszy. Za wyższością języka C++ przemawiają następujące argumenty: Język C++ uczy pełnego zrozumienia działania komputera oraz pamięci w komputerze poprzez jawne używanie wskaźników na obiekty, oraz referencji.

Podczas prac związanych z projektowaniem programu, były uwzględnione możliwości stworzenia programu w jednym z wielu różnych języków programowania; jedną z propozycji był język C++. Wybór na C# padł z powodu propozycji stworzenia modułu do weryfikacji ruchów właśnie w tym języku. Gdyby moduł do rozpoznawania obrazu powstał w C++, cała aplikacja byłaby zbyt zróżnicowana pod względem użytych technologii.

W przypadku modułu do rozpoznawania obrazu, w zasadzie nie byłoby różnicy między wyborem C# a C++. Używana w programie biblioteka OpenCV posiada możliwość użycia zarówno w języku C++ (w którym jest napisana), jak i w C# (za pomocą nakładki EmguCV). Między innymi, ze względu na podobną składnię obu języków, a także z powodu użycia praktycznie tych samych funkcji, kod programu, niezależnie od tego, w jakim języku by powstał, wyglądałby bardzo podobnie. O wyborze C# przesądziła jednak łatwość stworzenia interfejsu użytkownika. Tworzona aplikacja jest rozbudowanym programem, poza tym rozpoznawanie obrazu jest złożonym procesem, zależnym od wielu czynników zewnętrznych, dlatego program został wyposażony w bogate możliwości konfiguracji. Wymagało to stworzenia rozbudowanego interfejsu użytkownika, co jest znacznie łatwiejsze (i wymaga przeznaczenia na tę czynność zdecydowanie mniej czasu) w języku programowania C# niż w C++.

Zaletą języka C++ jest możliwość jego użycia na wielu platformach — na wielu systemach operacyjnych (jak np. Linux czy Windows) oraz na wielu różnych platformach sprzętowym (począwszy od prostych, ośmiobitowych mikrokontrolerów rodziny ATtiny, przez bardziej rozbudowane mikroprocesory rodziny ARM, aż po wydajne i złożone procesory architektury x64. Podczas opracowywania języka C++, jego autorzy bez wątpienia musieli pójść na wiele kompromisów — np. wbudowany mikrokontroler nie ma nawet części takich możliwości graficznych, jak np. system Windows z wydajną kartą graficzną i pakietem DirectX. Zapewne z tego właśnie powodu oraz z powodu chęci zachowania małych rozmiarów narzędzi niezbędnych do rozwoju oprogramowania w tym języku, twórcy biblioteki standardowej języka C++ nie zdecydowali się na umieszczenie w niej pewnych elementów, takich jak

funkcje służące do tworzenia graficznych interfejsów użytkownika oraz funkcje umożliwiające współpracę z tymi interfejsami. Oznaczałoby to konieczność znalezienia dodatkowych bibliotek graficznych.

Pewną trudnością jest fakt, że język C++ jest wprawdzie językiem wysokiego poziomu (w porównaniu z na przykład, asemblerem albo językiem maszynowym), jednak istnieją też języki posiadające wyższy poziom abstrakcji niż C++. Wykonanie interfejsu użytkownika w tym języku byłoby zapewne procesem znacznie bardziej złożonym i czasochłonnym, niż przy użyciu innych technologii. Jedną z rozważanych możliwości było wykorzystanie wspomnianej już biblioteki OpenCV, która posiada możliwość tworzenia graficznych interfejsów użytkownika. Narzędzia te są proste w obsłudze, jednak zapewniają jedynie niewielką kontrolę nad wynikowym interfejsem. Innym przykładem biblioteki, z którą miał już do czynienia co najmniej jeden z członków zespołu programistycznego tworzącego omawiany program, jest Windows API. Biblioteka ta jest dość trudna i skomplikowana w obsłudze, przez co stworzenie w niej interfejsu mogłoby w zasadzie być oddzielnym modułem, przygotowanym przez jedną z osób wchodzących w skład zespołu (w takim przypadku, składającego się z czterech członków).

Ostatecznie, po bardzo długich i burzliwych dyskusjach (podczas których wystąpiły również pewne straty materialne, związane z użyciem argumentów fizycznych), jako oficjalna technologia wykonania programu wybrany został język C# (znany również pod nazwami, takimi jak C Sharp lub C Płotek), działający na platformie .NET Framework. Dzięki jego zastosowaniu, tworzenie aplikacji przebiegło względnie szybko i sprawnie. Poza tym nie trzeba było przejmować się kwestiami takimi jak na przykład wydajność — stworzenie biblioteki OpenCV w języku C++ (pierwotnie w języku C) zapewnia wystarczająco wysoką wydajność, której najprawdopodobniej nie udałoby się uzyskać, gdyby bibliotek przetwarzania obrazu OpenCV została napisana od razu w języku C#.

6.3 Podział prac

Realizacja projektu byłaby niemożliwa, gdyby nie pełne zaangażowanie wszystkich członków zespołu. Każdy wywiązał się rzetelnie ze swoich zadań i aktywnie wspierał resztę zespołu w trakcie trwania prac nad projektem. Podział zadań przedstawiał się następująco:

- Marcin Orczyk.
  - Moduł do rozpoznawania obrazu z kamery i tworzenie cyfrowego odpowiednika planszy fizycznej.
- Natalia Popielarz.
  - Moduł logiki gry w Warcaby i weryfikacji poprawności ruchu wraz z wykrywaniem wykonania ruchu.
- Piotr Wołyński.
  - Moduł wizualizacji stanu gry na komputerze.

Ponadto, cała grupa intensywnie pracowała nad opracowaniem sprawozdania, dzięki czemu jest ono długie i rozwlekłe niczym programy napisane w Asemblerze.

6.4 Cele zrealizowane

Dzięki systematycznej pracy zrealizowaliśmy wszystkie zamierzone cele. Aplikacja działa, a zespół jest wdzięczny za możliwość jej wykonania.

6.5 Perspektywa rozwoju

Aplikację w przyszłości można rozwinąć o dodatkowe funkcjonalności np. dodanie zautomatyzowanego ustawiania parametrów ekspozycji w taki sposób, aby nie trzeba konfigurować ich ręcznie. Innym usprawnieniem aplikacji mogłoby być obsługa długich bić, bez konieczności wykonywania każdego pomniejszego bicia pojedynczo.