Podstawy teleinformatyki

Projekt zespołowy

Rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze

Małgorzata Sierbin
Joanna Chojnacka
Robert Greliński
Konrad Kurcaba

Informatyka, Wydział Elektryczny

Politechnika Poznańska

Spis treści:

1. Cel pracy i motywacja

1.1 Cel pracy

Celem pracy było stworzenie systemu umożliwiającego rozpoznawanie obrazu z toczącej się gry w warcaby oraz wizualizację stanu gry na komputerze przy jednoczesnej kontroli poprawności wykonywanych ruchów względem zasad gry w warcaby. Chcieliśmy, aby rozgrywka prowadzona przez dwóch zawodników mogła być obserwowana przez tłumy ludzi. Pozwoli to na przeprowadzenie zawodów w warcaby w taki sposób, aby publiczność mogła na bieżąco śledzić postępy na ekranie projekcyjnym.

1.2 Główne moduły

Temat opiera się na połączeniu ze sobą trzech głównych modułów, jakimi są:

- podsystem do rozpoznawania obrazu wyszukujący pozycję pionków na planszy i samej planszy,
- podsystem sprawdzający poprawność wykonanych ruchów, czyli zgodność z zasadami,
- podsystem wizualizacji pionków i planszy.

Połączenie ich umożliwia poprawne działanie gry w warcaby. Wizualizacja odzwierciedla stan bieżącej rozgrywki na komputerze. Fizycznie nad planszą do grania umieszczona jest kamera podłączona do komputera. Pomocniczo wykorzystano stojak do kamery w postaci stojącej lampy, do której umocowano kamerę przy pomocy taśmy klejącej.

1.3 Motywacja

Główną motywacją wyboru tematu była chęć poznania technologii rozpoznawania obrazów i metodyki stosowanej przy analizie ruchów. Implementacja gry pozwoliła na wykorzystanie oraz rozszerzenie zdobytych podczas studiów umiejętności programistycznych oraz projektowych.

Zapoznaliśmy się z biblioteką OpenCV, co w obecnym semestrze jest umożliwione jedynie na przedmiocie projektowym z podstaw telekomunikacji. Naszym celem było rozwinięcie umiejętności programistycznych. Kierowaliśmy się zainteresowaniem do strategicznych gier planszowych. Dzięki temu możemy nie tylko dostarczyć innym rozrywkę, ale także pomóc w zrozumieniu zasad i w zwiększeniu doświadczenia w grze. Nasza praca z pewnością miała dużą wartość dydaktyczną, a jej efekty mogą posłużyć w celach rozrywkowych.

1.4 Założenia

Ustalono pewne założenia dotyczące działania aplikacji:

- pobranie obrazu przedstawiającego ustawienie pionków na planszy
- wizualizacja powyższego obrazu na ekranie aplikacji
- sprawdzenie czy wykonany ruch jest zgodny z zasadami gry w warcaby
- zakomunikowanie o błędnie wykonanym ruchu
- możliwość wstępnej konfiguracji parametrów maski

2. Autorzy projektu i podział prac

2.1 Autorzy projektu i podział prac

Prace podzieliliśmy na moduły dotyczące wizualizacji pionków i planszy, sprawdzanie poprawności wykonanych ruchów oraz rozpoznawanie obiektów i pozycji na planszy. Pozwoliło to na rozdzielenie zadań pomiędzy poszczególnymi członkami zespołu.

Autorami projektu są następujące osoby oraz przypisane im zadania składające się na całość:

- Joanna Chojnacka – wizualizacja pionków i planszy,
- Małgorzata Sierbin – wizualizacja pionków i planszy,
- Konrad Kurcaba – sprawdzenie poprawności wykonanych ruchów,
- Robert Greliński – rozpoznawanie obiektów na planszy, wykrywanie pozycji.
- Dokumentację oraz łączenie modułów opracowano wspólnie.

2.2 Podział na moduły

2.2.1 Moduł wizualizacji pionków i planszy

- zaprojektowanie interfejsu do obsługi wizualizacji
- oprogramowanie tworzenia pionków
- oprogramowanie tworzenia planszy
- możliwość wyświetlania zasad gry w warcaby tradycyjne
- przygotowanie modułu do połączenia podsystemów
- komunikacja z podsystemami
- wykonanie grafiki pionków (w dwóch wariantach kolorystycznych), planszy, ramki.

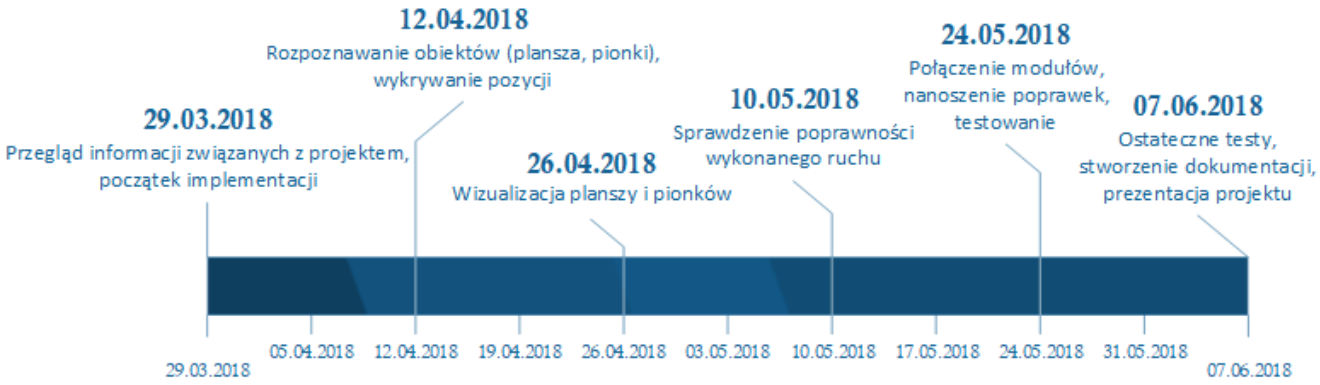
2.2.2 Moduł sprawdzania poprawności wykonania ruchów

- wprowadzenie zasad wykonywanych ruchów
- analiza poprawnego początkowego ustawienia pionków i planszy
- przechowywanie ruchów
- przygotowanie modułu do podłączenia

2.2.3 Moduł rozpoznawania obiektów na planszy, wykrywanie pozycji

- rozpoznawanie obiektów (markery, planszy, pionków)
- przekształcenia morfologiczne
- wykrywanie pozycji pionków, planszy
- dodanie możliwości konfiguracji parametrów
- przygotowanie modułu do podłączenia

2.3 Harmonogram prac



Schemat 1 *- harmonogram prac*

W celu usprawnienia i ułatwienia prac przygotowaliśmy plan według którego realizowaliśmy nasze zadania. Głównym celem jego stworzenia, było trzymanie się wyznaczonych terminów i tworzenia odpowiednich funkcjonalności w odpowiednich etapach prac. Podczas realizacji projektu harmonogram nieznacznie uległ zmianie, ponieważ nie zawsze jesteśmy w stanie przewidzieć trudności, które nas mogą zastać.

2.3.1 Przebieg prac

29.03 - Przegląd dostępnych informacji związanych z projektem, początek implementacji.

- przygotowanie harmonogramu
- wyszukanie informacji w internecie
- zapoznanie się z biblioteką OpenCV
- zapoznanie z językiem CLI
- wyszukanie i wybranie zasad gry w warcaby
- przygotowanie planszy, pionków

12.04 - Implementacja - rozpoznawanie obiektów (plansza, pionki), wykrywanie pozycji.

- wykonanie grafiki planszy i pionków
- wprowadzenie części zasad gry
- wykonanie przekształceń morfologicznych na rozpoznawanym obrazie
- testy wybrania najlepszych ustawień dla rozpoznawania obrazu

26.04 - Wizualizacja planszy i pionków.

- utworzenie interfejsu
- oprogramowanie planszy i pionków
- zastosowanie utworzonej wcześniej grafiki
- dodanie kolejnej części zasad gry
- poprawa wykrywania markerów

10.05 - Sprawdzenie poprawności wykonanego ruchu.

- dodanie kolejnej części zasad gry
- stworzenie macierzy z przechwytywanego obrazu
- pobieranie wartości macierzy do odpowiedniej pozycji pionków
- dodanie większej ilości kolorów pionków

24.05 - Nanoszenie poprawek. Testowanie.

- połączenie modułu wizualizacji i podsystemu rozpoznawania obrazów
- wprowadzenie pozostałych brakujących zasad
- poprawa błędów wynikających z połączenia modułów
- testy

7.06 - Ostateczne testy. Stworzenie dokumentacji. Prezentacja projektu.

- zmodyfikowanie interfejsu
- dodanie pasków konfiguracji zakresu kolorów
- dodanie przycisków podglądu rozpoznawania obrazów
- utworzenie dokumentacji
- połączenie modułów
- testy
- poprawa błędów wykrytych przy łączeniu
- oddanie projektu

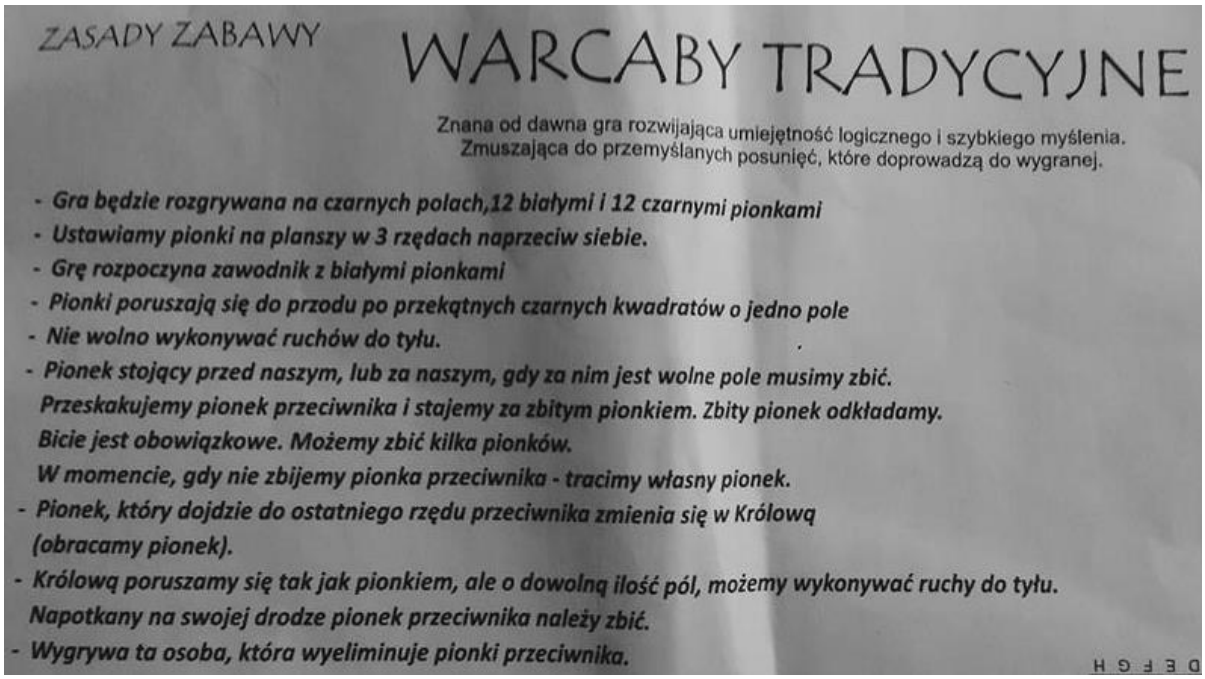
3. Funkcjonalność oferowana przez aplikację

3.1 Funkcjonalność aplikacji

Aplikacja umożliwia pobieranie obrazu z kamery ustawionej nad planszą z pionkami do gry w warcaby. Dostępny jest podgląd zarówno bezpośredniego obrazu, jak i jego wizualizacji. Istnieje możliwość konfiguracji parametrów maski podczas rozpoznawania obrazu. Za pomocą przycisku aktualizuje się wizualizację poprzez pobranie obecnego obrazu z kamery. Aplikacja informuje o złamaniu zasad poprawności ruchów. Użytkownik może dodatkowo podejrzeć obowiązujące zasady w grze. W naszym projekcie zostały wprowadzone zasady poprawności ruchów dla warcabów tradycyjnych.

3.2 Obowiązujące zasady poprawności ruchów

Podczas opracowywania zbioru zasad opieraliśmy się na posiadanych zasadach z naszej fizycznej, prywatnej gry.

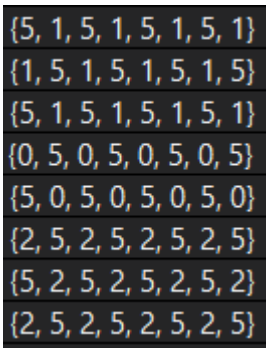


Rysunek 1 *- Zdjęcie zasad do gry w warcaby*

- Gra będzie rozgrywana na czarnych polach, 12 białymi i 12 czarnymi pionkami.
- Ustawiamy pionki na planszy w 3 rzędach naprzeciw siebie.
- Grę rozpoczyna zawodnik z białymi pionkami.
- Pionki poruszają się do przodu po przekątnych czarnych kwadratów o jedno pole.
- Nie wolno wykonywać ruchów do tyłu.
- Jeśli pionek stojący przed lub za naszym ma za sobą wolne pole, musi zostać zbity. Przeskakujemy pionek przeciwnika i stajemy za zbitym pionkiem. Zbity pionek odkładamy.
- Bicie jest obowiązkowe.

- Możemy zbić kilka pionków.
- W momencie, gdy nie zbijemy pionka przeciwnika – tracimy własny pionek.
- Pionek, który dojdzie do ostatniego rzędu przeciwnika zmienia się w Królową.
- Królową poruszamy się tak jak pionkiem, ale o dowolną ilość pól, możemy wykonywać ruchy do tyłu. Napotkany na swojej drodze pionek przeciwnika należy zbić.
- Wygrywa ta osoba, która wyeliminuje pionki przeciwnika.
- Gdy po biciu możliwe jest kolejne bicie to ruch wykonuje ta sama osoba i musi wykonać to bicie.
- Każdy gracz rozpoczyna grę z dwunastoma pionami (dwóch kolorów).
- Gra rozgrywana jest na ciemnych polach planszy o rozmiarze 8 × 8 pól.
- Piony mogą poruszać się o jedno pole do przodu po przekątnej (na ukos) na wolne pola.
- Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne).
- Zbite piony są usuwane z planszy po zakończeniu ruchu.
- Piony mogą bić zarówno do przodu, jak i do tyłu.

3.3 Macierz reprezentacji ustawienia pionków na planszy



Rysunek 2 *- macierz ustawienia pionków na planszy*

Na rysunku przedstawiono macierz zawierającą pozycje pionków na planszy względem ich rodzajów. Jest to sposób reprezentacji początkowego ustawienia pionków na planszy użytej w naszej aplikacji. Taka macierz jest transportowana pomiędzy modułami rozpoznawania obrazu, wizualizacji oraz sprawdzania poprawności.

Zastosowane oznaczenia:

- 0 - Puste czarne pole
- 1 - Pole z białym pionkiem
- 2 - Pole z czarnym pionkiem
- 5 - Białe pole

3.4 Pojęcia morfologiczne dotyczące przekształcania obrazu

3.4.1 Operacje morfologiczne

Operacje morfologiczne służą do przetwarzania obrazów. Dzięki nim możemy analizować odległości między obiektami oraz ich kształty. umożliwia to realizację złożonych systemów analizy obrazu poprzez połączenie ze sobą kilku podstawowych przekształceń morfologicznych.

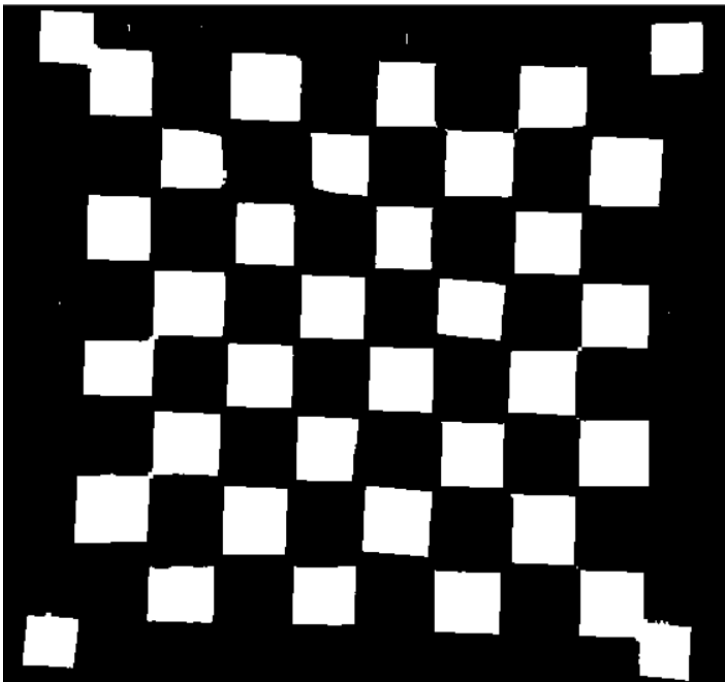
3.4.2 Przekształcenia morfologiczne cyfrowych obrazów

Są to takie przekształcenia, w wyniku których zmieniona zostaje struktura lub forma obiektu. Znajduje zastosowanie w filtracji morfologicznej, wyszukiwania informacji i analizy kształtu z wykorzystaniem elementów strukturalnych.

3.4.3 Progowanie

Metoda polega na przekształceniu obrazu kolorowego lub szarego na obraz binarny.

3.4.4 Erozja



Rysunek 3 *- Erozja na obrazie z kamery przedstawiającym planszę do gry*

Erozja inaczej oznacza zwężanie, która stosuje różnicę Minkowskiego do obrazu. Operacja polega na przyłożeniu obróconego elementu strukturalnego do każdego piksela na obrazie. Jeśli chociaż jeden piksel z otoczenia objętego przez B ma wartość zero, punkt centralny otrzymuje również wartość zero. W innym wypadku jego wartość się nie zmienia. Erozja to operacja dualna do dylatacji, jeśli zostanie wykonany negatyw obrazu operację przejdą jedna w drugą. Istotny wpływ na erozję ma wybór elementu strukturalnego.

$$I \ominus B \equiv \{p \in U_1 | \tilde{B}_p \subset I\}, \text{ gdzie } \tilde{B}_p \text{ oznacza przyłożenie } \tilde{B} \text{ do piksela p.}$$

Skutkiem erozji jest zmniejszenie obiektu, zaniknięcie wąskich gałęzi i małych elementów, pozbycie się szumu, poszerzenie się „dziur” w niespójnym obszarze i przyjmują kształt elementu strukturalnego.

3.4.5 Dylatacja

Dylatacja inaczej rozszerzenie polega na użyciu sumy Minkowskiego do obrazu:

$$I \oplus B \equiv \{p \in U_1 | \tilde{B} \cap I \neq 0\} \quad \text{lub} \quad I \oplus B \equiv \bigcup_{p \in B} (I + p)\text{, gdzie}$$

I – *element strukturalny każdego piksela na obrazie,*

B – *przyłożenie piksela na obrazie,*

p – *piksel*

Operacja przebiega w następujący sposób: przykładamy obrócony element strukturalny do każdego piksela na obrazie po kolei, następnie jeżeli chociaż jeden piksel z otoczenia B ma wartość jeden to punkt centralny też uzyskuje wartość jeden. Inaczej przypisywane jest mu wartość zero. Ważny wpływ na wynik dylatacji ma dobór elementu strukturalnego. Przy nietypowym kształcie rozrost obiektu jest kierunkowy. Jednak, jeśli punkt centralny nie jest umieszczony w środku elementu, rozrost jest niesymetryczny. Najbardziej naturalny efekt otrzymamy, gdy wykorzystamy element zbliżony do koła. Skutkiem dylatacji jest powiększenie obiektu, zniknięcie szczegółów i wypełnienie „dziur” w niespójnym obszarze. Dla uzyskania pożądanego efektu zaleca się stosowanie dylatacji wielokrotnej.

3.4.6 Otwarcie

Otwarcie polega na wykonaniu operacji dylatacji na erozji obrazu pierwotnego:

$$I \circ B = (I \ominus B) \oplus B\text{, gdzie}$$

I – *element strukturalny każdego piksela na obrazie,*

B – *przyłożenie piksela na obrazie*

Operacja nie zmienia rozmiarów obiektów obrazu oraz wygładza je poprzez usunięcie wszystkich niedoskonałości. Gdy powiększamy elementy strukturalne B powoduje to usuwanie coraz większych detalu obrazu i odwzorowywanie powstałych obszarów do elementu strukturalnego. Operacja otwarcia dodatkowo cechuje się idempotentnością oznaczającą niezmienność wyniku niezależnie od ilości powtarzania operacji.

3.5 Moduł rozpoznawania obrazu

Kroki algorytmu modułu rozpoznawania obrazu:

1. Pobranie obrazu z kamery
2. Zmiana palety kolorów z BGR na HSV
3. Progowanie na podstawie pobranych kolorów
4. Erozja na masce markerów
5. Wykrycie konturów markerów
6. Ustalenie pozycji środka markera

7. Wycięcie obrazu o kształcie prostokąta o wierzchołkach będących środkami markerów
8. Nowy obraz przekształcamy z palety kolorów BGR na HSV
9. Tworzenie dwóch nowych masek
10. Do maski białej B przypisujemy wartości kolorów zielonych oznaczających pionki białe
11. Do maski czarnej C przypisujemy wartości kolorów różowych oznaczających pionki czarne
12. Progowanie na powyższych maskach
13. Operacja erozji
14. Operacja dylatacji
15. Podzielenie planszy na 64 jednakowe elementy
16. Wykrycie co znajduje się w danym elemencie jeżeli się znajduję, na podstawie maski białej i czarnej
17. Powstałą macierz zwracamy do systemu warcabów
18. Dodanie możliwości konfiguracyjnych dotyczących parametrów maski

W celu realizacji operacji erozji wykorzystano gotową funkcję udostępnioną przez bibliotekę OpenCV o postaci *erode(source, erosion_dst1, element).* Natomiast dylację zrealizowano poprzez użycie funkcji *dilate(source, dilation_dst, element)*, która zawarta jest również w bibliotece OpenCV.

Parametry oznaczają kolejno:

source - obiekt na którym wykonywana ma być operacja

erosion_dst1 - wartość oznaczająca wybór kształtu użytego kernela, my zastosowaliśmy kernel nr 1, czyli *erosion_type=MORPH_CROSS*, co oznacza kształt wybranego kernela.

dilation_dst - jest to wartość analogiczna do *erosion_dst1*

element - określa rozmiar obiektu poddawanego operacji

3.5.1 Implementacja operacji erozji i dylacji

```
//erozja
Mat Erosion(Mat source, int erosion_elem, int erosion_size)
{
        Mat erosion_dst1;
        int erosion_type;

        if (erosion_elem == 0) {
           erosion_type = MORPH_RECT;
        }
```

```
        else if (erosion_elem == 1) {
           erosion_type = MORPH_CROSS;
        }
        else if (erosion_elem == 2) {
           erosion_type = MORPH_ELLIPSE;
        }

        Mat element = getStructuringElement(erosion_type,
                Size(2 * erosion_size + 1, 2 * erosion_size + 1),
                Point(erosion_size, erosion_size));
        erode(source, erosion_dst1, element);
        return erosion_dst1;
}

//dylacja
Mat Dilation(Mat source, int dilation_elem, int dilation_size)
{
        Mat dilation_dst;
        int typ_dylacji;

        if (dilation_elem == 0) {
           typ_dylacji = MORPH_RECT;
        }
        else if (dilation_elem == 1) {
           typ_dylacji = MORPH_CROSS;
        }
        else if (dilation_elem == 2) {
        typ_dylacji = MORPH_ELLIPSE;
        }

        Mat element = getStructuringElement(typ_dylacji,
                Size(2 * dilation_size + 1, 2 * dilation_size + 1),
                Point(dilation_size, dilation_size));
```

```
        dilate(source, dilation_dst, element);
        return dilation_dst;
}
```

Funkcja operacji dylacji została zaimplementowana w analogiczny sposób do erozji. Zastosowano również funkcję *cvtColor(img, grey_img, CV_BGR2GRAY)* służącą do konwersji obrazu do odcieni szarości.

Parametry oznaczają:

- *img* - obraz wejściowy
- *grey_img* - obraz wyjściowy w skali szarości
- *CV_BGR2GRAY* - kod konwersji oznaczający kolor

Kolejną użytą funkcją z wykorzystywanej biblioteki jest *findContours(mask1, kontury, hierarchia, cv::RETR_CCOMP, cv::CHAIN_APPROX_SIMPLE),* która służy do znajdowania konturów w obrazie binarnym.

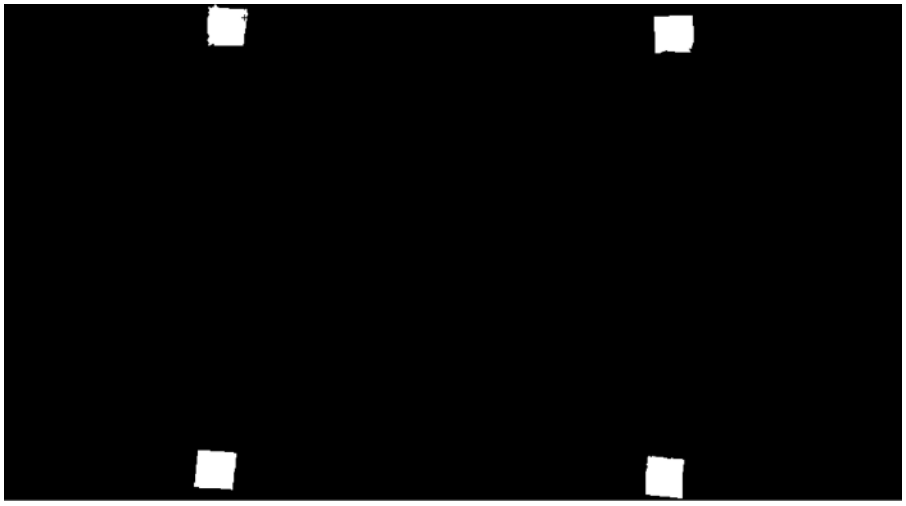
Oznaczenie parametrów:

- *mask1* - przetwarzany obraz binarny
- *kontury* - w tej zmiennej umieszczane są dane o konturach
- *hierarchia* - oznacza hierarchię konturów
- *cv::RETR_CCOMP* - oznacza, że interesują nas tylko kontury zewnętrzne obiektów
- *cv::CHAIN_APPROX_SIMPLE* - jest to wskazanie algorytmu wykorzystywanego do wykonania operacji

Wykorzystano także funkcję *inRange(planszaHSV, Scalar(150, 30, 0), Scalar(200, 255, 255), czarnePionki)*, służącą do progowania.

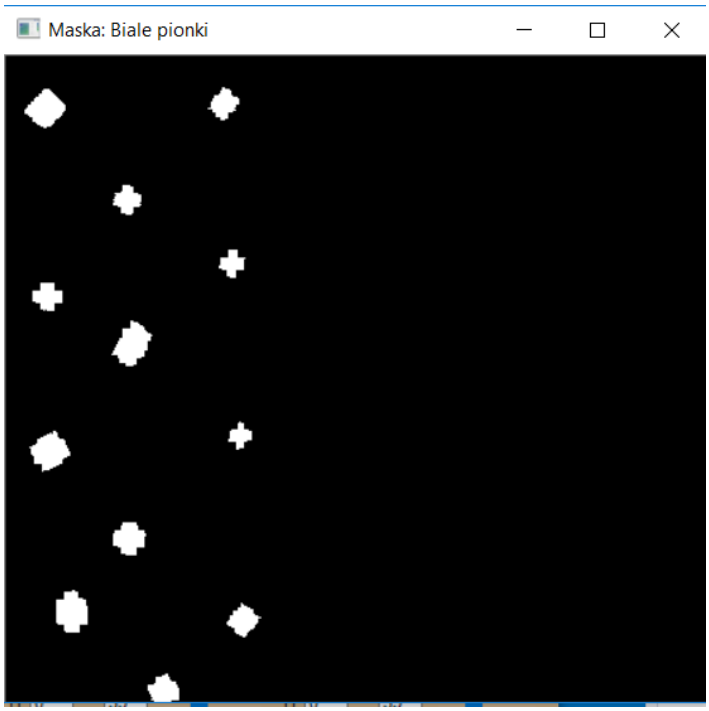
Parametry:

- *planszaHSV* - obrazek, który poddamy progowaniu
- *Scalar(150, 30, 0)* - dolna granica przedziału do którego musi należeć dany piksel aby w obrazie wyjściowym miał dany kolor
- *Scalar(200, 255, 255)* - górna granica przedziału do którego musi należeć dany piksel aby w obrazie wyjściowym miał dany kolor
- *czarnePionki* - obrazek wynikowy.



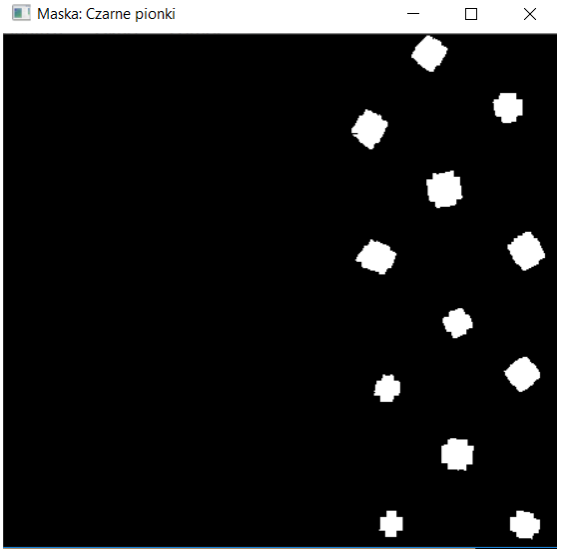
Rysunek 4 *- wykrywanie markerów*

W celu znajdowania planszy umieściliśmy na niej cztery markery, dzięki temu znajdujemy narożniki planszy i dzięki temu można ją wyciąć i precyzyjnie znajdować pionki nie przejmując się otoczeniem. Wykonano erozję na masce markerów, następnie wykryto ich kontury, kolejno ustalo pozycję środka markera i wycięto obraz o kształcie prostokąta o wierzchołkach będących środkami markerów.



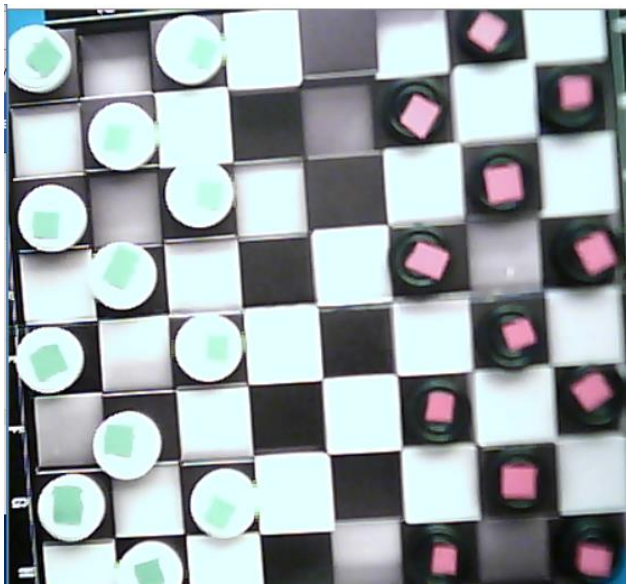
Rysunek 5 *- rozpoznany obraz po przekształceniach morfologicznych (białe pionki)*

W celu znajdowania białych pionków umieściliśmy na nich zielone znaczniki, dzięki czemu jesteśmy w stanie dokładnie wykryć ich położenie względem planszy. Zastosowano tu operacje otwarcia, dzięki której pozbyliśmy się szumu otoczenia spowodowanego różnym oświetleniem. Następnie pobraliśmy położenie obiektów względem planszy i zmapowaliśmy je do tabeli dwuwymiarowej reprezentującej położenie pionków na planszy.



Rysunek 6 *- rozpoznany obraz po przekształceniach morfologicznych (czarne pionki)*

Obraz przedstawia czynności wykonane w celu znajdowania czarnych pionków umieściliśmy na nich czerwone znaczniki, dzięki czemu jesteśmy w stanie dokładnie wykryć ich położenie względem planszy. Zastosowano tu operacje otwarcia, dzięki której pozbyliśmy się szumu otoczenia spowodowanego różnym oświetleniem. Następnie pobraliśmy położenie obiektów względem planszy i zmapowaliśmy je do tabeli dwuwymiarowej reprezentującej położenie pionków na planszy.



Rysunek 7 *- podgląd planszy*

Obraz przedstawia podgląd planszy. Znaczniki o kolorze niebieskim położone na rogach planszy służą do wycięcia planszy, którą uzyskaliśmy z obrazu z kamery. Znaczniki na pionkach służą do rozpoznawania pionków na planszy. Kolory użyte jako znaczniki to różowy dla czarnych pionków oraz zielony dla białych. Wyciętą plansze podzielono na 64 elementy.

3.6 Moduł wizualizacji

W celu jego przygotowania zrealizowano następujące punkty:

1. Zapoznano się z grą w warcaby
2. W programie graficznym Inkscape wykonano grafikę przedstawiającą planszę do gry w warcaby
3. Wykonano również grafikę pionków do gry w czterech różnych kolorach reprezentujących dwóch graczy oraz pionki damek
4. Pobranie macierzy reprezentującej ustawienie pionków od modułu rozpoznawania obrazów
5. Stworzenie interfejsu wykorzystującego CLI
6. Dodanie zbioru pictureBoxów do interfejsu oraz przypisanie ich do macierzy
7. Dodanie przycisków do interfejsu
8. Podzielenie obrazu planszy na 64 pola
9. Umieszczenie każdego pola w kolejnych pictureBoxach
10. Analiza formatu macierzy pobranej od modułu rozpoznawania obrazów
11. Opracowania grupy warunków na podstawie których zapis w macierzy jest odwzorowywany w wizualizacji
12. Zmiana poszczególnych pictureBoxów względem wartości z macierzy
13. Dodanie funkcjonalności przycisków ustawiających planszę, wywołujących moduł rozpoznawania obrazu oraz ustawiających pionki
14. Możliwość podejrzenia zasad gry
15. Dołączenie modułu sprawdzania poprawności
16. Wywołanie modułu sprawdzania poprawności po pobraniu macierzy od modułu rozpoznawania obrazu

Podczas implementacji wykonywano na bieżąco testy dodawanych funkcjonalności. Stopniowo łączono ze sobą trzy powyższe moduły. Dokonywano koniecznych modyfikacji oraz wykonano wiele testów w celu sprawdzenia poprawności połączenia modułów i ich wzajemnej współpracy. Podczas kolejnych prób dochodziły nowe błędy wynikające z implementacji oraz głównie z konfliktu typów. Sprawdzono czy wymiana danych jest wykonywana w sposób poprawny. Testowano czy aplikacja prawidłowo reaguje i czy wszystkie funkcjonalności, które działały przed połączeniem w jedną całość, wciąż działają po połączeniu.

3.7 Możliwe przyszłe ulepszenia aplikacji

- **moduł rejestracji i logowania użytkowników** – możliwość posiadania własnego konta w aplikacji byłaby dobrym sposobem do zachęcenia użytkowników do korzystania z niej,
- **ranking użytkowników** – możliwość rywalizacji między ludźmi, sprawdzanie siebie i wzrostu swoich umiejętności,
- **możliwość połączenia z innym użytkownikiem i granie na połowie swojej planszy** – kolejna funkcjonalność, która zachęca do korzystania z aplikacji, użytkownik może sprawdzić się ze swoim przyjacielem i rozstrzygnąć kto jest lepszy,
- **okienko czatu do komunikacji między graczami** – umożliwi wymianę refleksji, opinii między użytkownikami,
- **wybór kolorów pionków i planszy** – tryb dla daltonistów,

- **możliwość użycia kamery telefonu komórkowego** – korzystanie z aparatu w urządzeniu mobilnym jest dużym ułatwieniem, ponieważ to ogólnodostępne rozwiązanie oraz rozwiązuje problem z zapotrzebowaniem na statyw,
- **zmiana markerów na mniejsze** – względy estetyczne,
- **dodanie trybu gry z komputerem** – wyświetlanie ruchu wykonanego przez komputer, aby możliwe było przeprowadzenie rozgrywki bez potrzeby drugiej osoby
- **zastosowanie technologii AdMob w celu użycia reklam w aplikacji** – np. umieszczenie banerów i zarabianie na nich pieniędzy,
- **możliwość ustawienia planszy pod kątem do kamery** – rozwiązuje problem statycznego otoczenia i stałych warunków oświetlenia,
- **matowa szachownica** – rozwiązanie problemu z odbijaniem się światła od planszy
- **automatyczne dostosowanie kolorów** – uniknięcie potrzeby pobierania kolorów HSV
- **podświetlenie konkretnego błędu** – brak konieczności szukania samemu błędu
- **zastosowanie sieci neuronowych do wyboru manewru pionków** – możliwość wybierania najlepszego ruchu.

4. Wybrane technologie

4.1 Technologie

4.1.1 Język programowania

Użyliśmy jednego z najpopularniejszych języków programowania – C++ oraz CLI języka opartego na C++. CLI wykorzystaliśmy przy wykonaniu interfejsu graficznego w podsystemie wizualizacji, ponieważ nie znaleźliśmy innego rozwiązania tego problemu. C++ pozwolił nam na bezpośredni dostęp do zasobów, budowanie naszej aplikacji.

4.1.2 Środowisko programistyczne

Skorzystaliśmy z zintegrowanego środowiska programistycznego firmy Microsoft - Visual Studio w wersji 15, które jest jednym z najczęściej używanych programów do tworzenia oprogramowania konsolowego z graficznym interfejsem użytkownika.

4.1.3 Biblioteka graficzna

OpenCV to biblioteka funkcji służących do obróbki obrazu, oparta na otwartym kodzie i zapoczątkowana przez Intela. Jest wieloplatformowa - wspiera systemy Windows, Mac OS X i Linux. Wspiera języki C++, którego używamy oraz umożliwia przetwarzanie obrazu w czasie rzeczywistym. Biblioteka dostarcza struktury pozwalające na przechowywanie danych wizyjnych, oraz funkcje implementujące algorytmy służące do przetwarzania i rozpoznawania obrazów.

4.1.4 System

Aplikacja została zrealizowana w systemie Windows 10. Do uruchomienia na innych systemach wymagane jest jedynie środowisko Visual Studio.

4.1.5 Dodatkowe narzędzia

Użyliśmy systemu kontroli wersji Git, który służy do kontrolowanego wprowadzania zmian w projekcie. Pozwala na utworzenie repozytorium oraz współdzielenia go z innymi użytkownikami, sprawdza się przy pracach projektowych, ponieważ wiele osób może pracować nad jednym problemem.

4.2 Uzasadnienie wybranych technologii

Wybraliśmy język C++, ponieważ jest to język, który jest opanowany przez nasz zespół w największym stopniu oraz dokumentacja i pomoce naukowe są ogólnodostępne. CLI wybrano z powodu konieczności zastosowania interfejsu i nie znalezienia innego równie szybkiego rozwiązania. Środowisko charakteryzuje się łatwością obsługi oraz posiadamy zadowalające doświadczenie w pracy z nim. Dodaliśmy bibliotekę OpenCV, która pozwala na korzystanie z wbudowanych funkcji dotyczących rozpoznawania obrazu i wykrywania obiektów oraz była już wcześniej przez nas używana. Serwis Github służył nam do organizacji projektu i kontroli postępu projektu oraz przesyłania go pomiędzy członkami.

5. Narzędzia

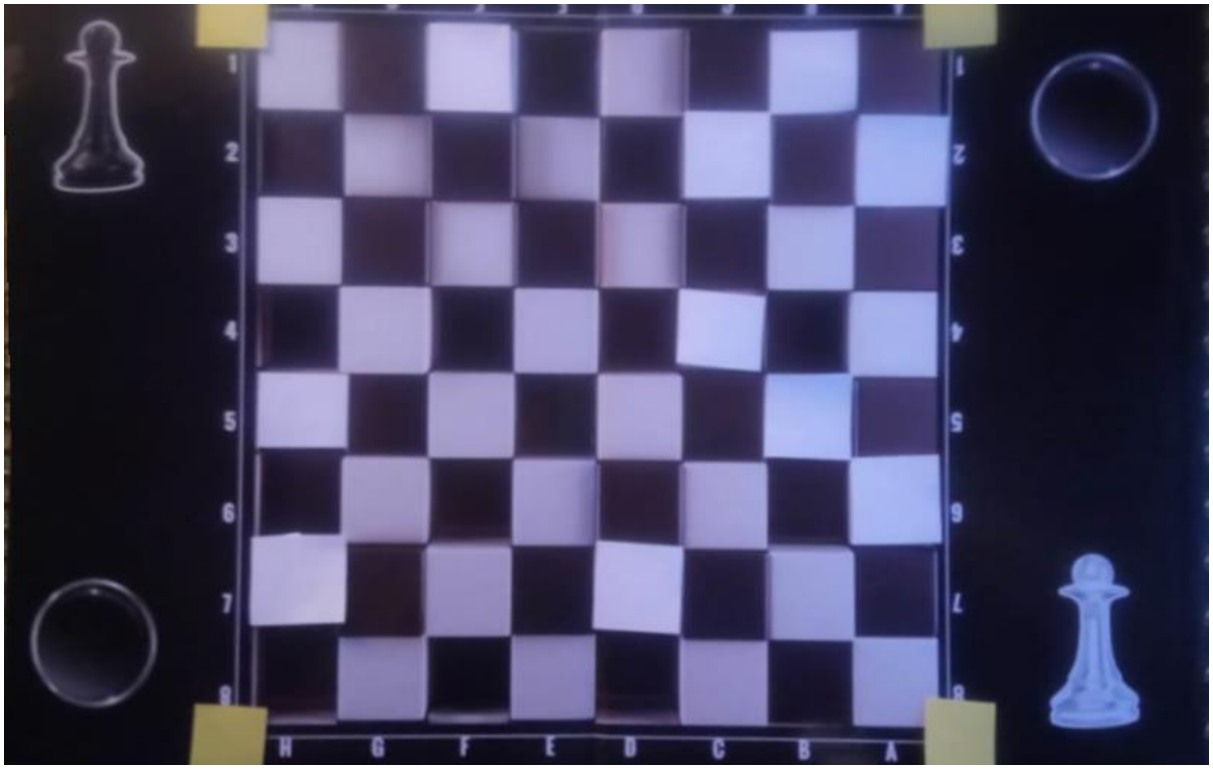
5.1. Sprzęt

5.1.2 Kamera

Rysunek 8 *- kamera wykorzystywana w projekcie*

Odpowiedzialna za odbiór danych wizyjnych jest kamera cyfrowa WEB Camera wyprodukowana przez firmę Apollo Multimedia. Umożliwia pobieranie obrazu w kolorze o rozdzielczości 2.0 Mpix. Posiada możliwość obrotu o 360°, automatyczny balans bieli, system auto korekcji kolorów, zasilana przez USB 2.0, nie wymaga zewnętrznego zasilania, nie wymaga oprogramowania. Wsparcie systemu Win10 oraz niższe.

5.1.3 Akcesoria do gry

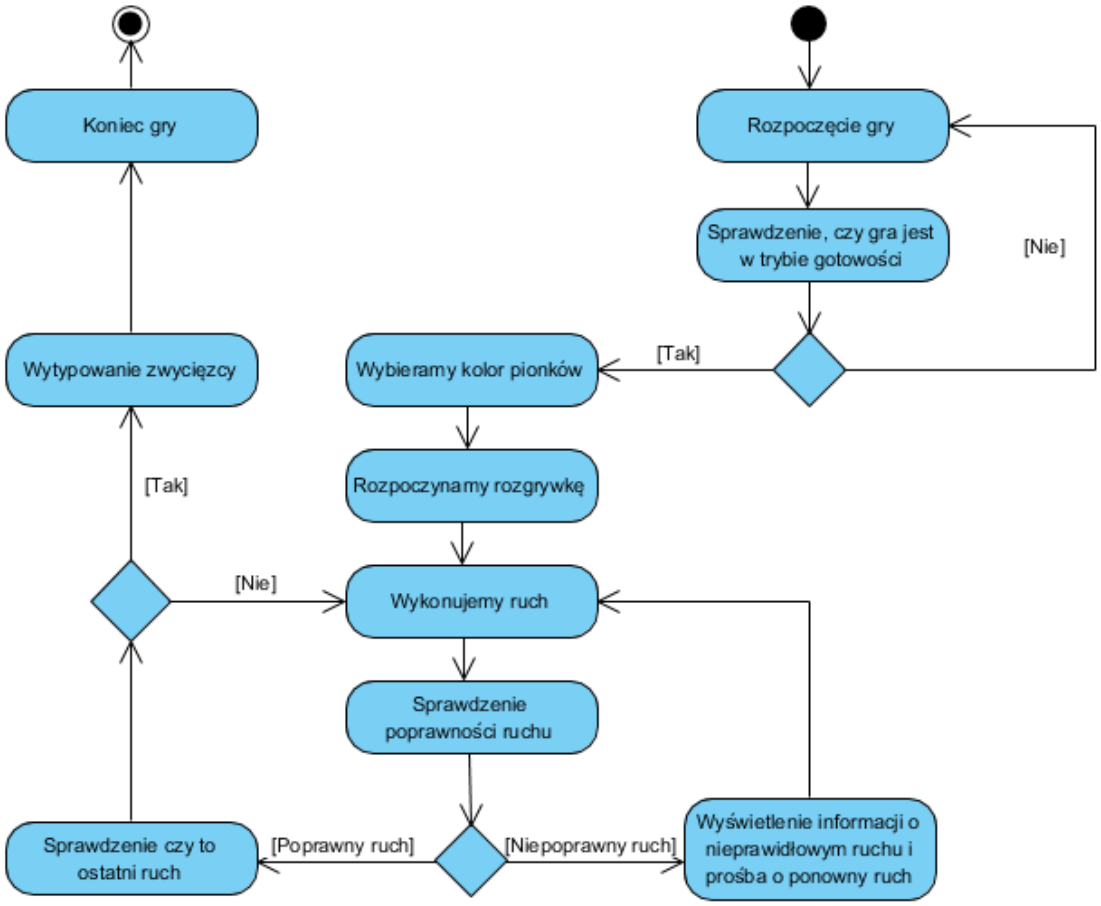


Rysunek 9 *- plansza do gry w warcaby*

Plansza jest czarno-biała i ma wymiary 8 × 8 pól, gdzie każde pole jest kwadratem o długości 3cm. Szachownica jest umieszczona na planszy z miejscem przeznaczonym na odkładanie niepotrzebnych pionków. W tradycyjnych warcabach najczęściej używane są białe i czarne pionki, a damka składa się z dwóch pionków nałożonych na siebie. Niestety w naszym przypadku takie rozwiązanie nie jest efektywne. Jest to spowodowane koniecznością rozróżniania innych kolorów przechwytywanych przez kamerę. Nasze pionki posiadają kolorowe znaczniki. Każdy pionek jest walcem o średnicy 2 cm i wysokości 0,8 cm.

6. Architektura rozwiązania

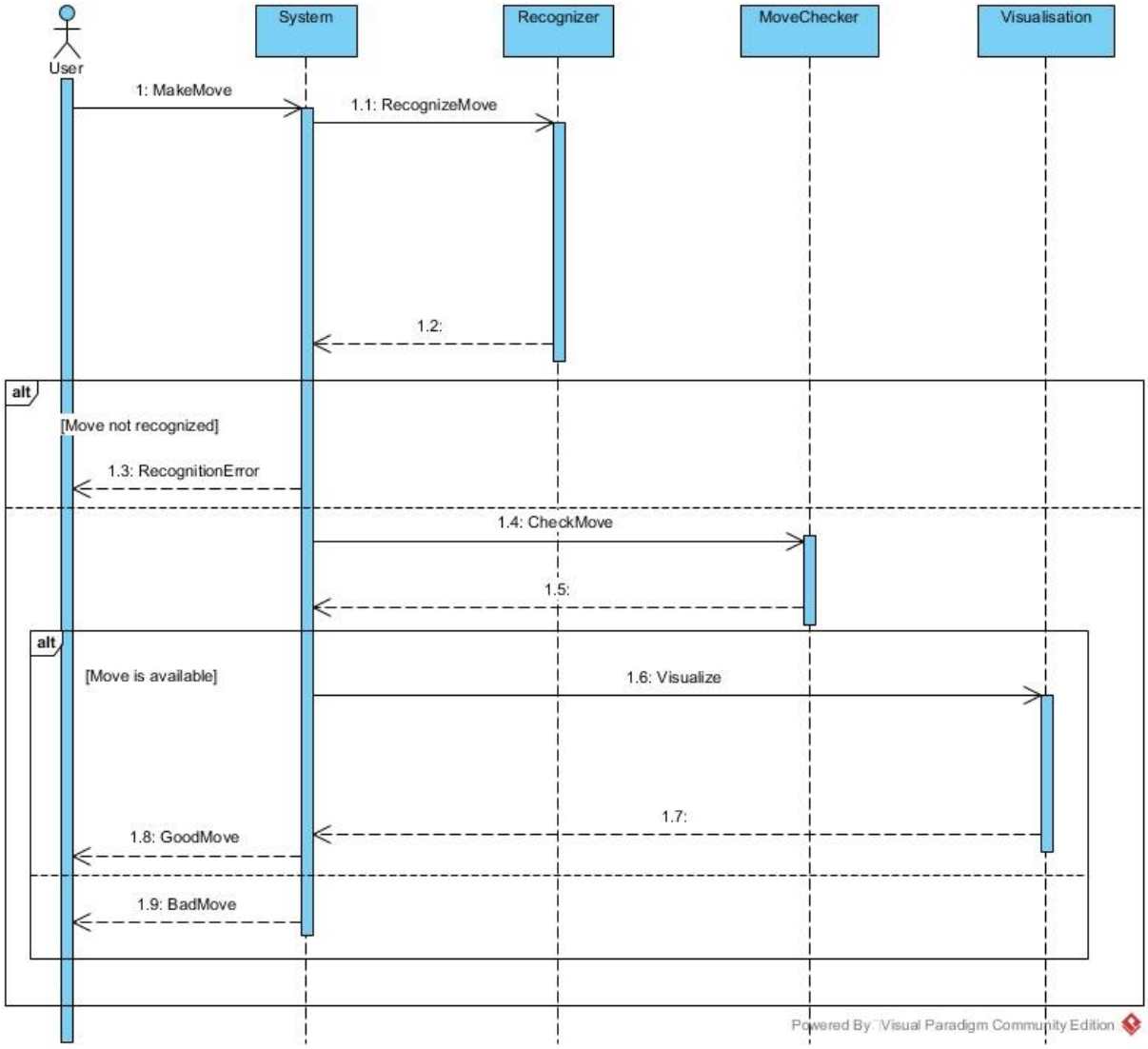
6.1 Diagram aktywności



Rysunek 10 *- diagram aktywności UML rozgrywki gry w warcaby*

Ilustracja przedstawia diagram czynności UML rozgrywki gry w warcaby. Węzeł początkowy rozpoczyna grę, następnie sprawdzamy, czy gra jest w trybie gotowości, jeżeli nie - wracamy do ponownego rozpoczęcia gry, w drugim przypadku przechodzimy do wyboru koloru pionków *(w naszej wizualizacji mamy dostępne dwa komplety kolorów)*. Kolejnym krokiem jest rozpoczęcie rozgrywki, następnie wykonujemy ruch, dalej sprawdzamy poprawność ruchu. W przypadku nieprawidłowej zmiany pozycji wyświetlamy informacje o błędnym ruchu i prośbę o ponowienie oraz przechodzimy do wykonania ruchu. Poprawny manewr oznacza przejście do czynności sprawdzającej czy to ostatnia akcja, w przypadku gdy nie - wracamy do wykonania ruchu, gdy tak - typujemy zwycięzcę oraz węzeł końcowy kończy grę.

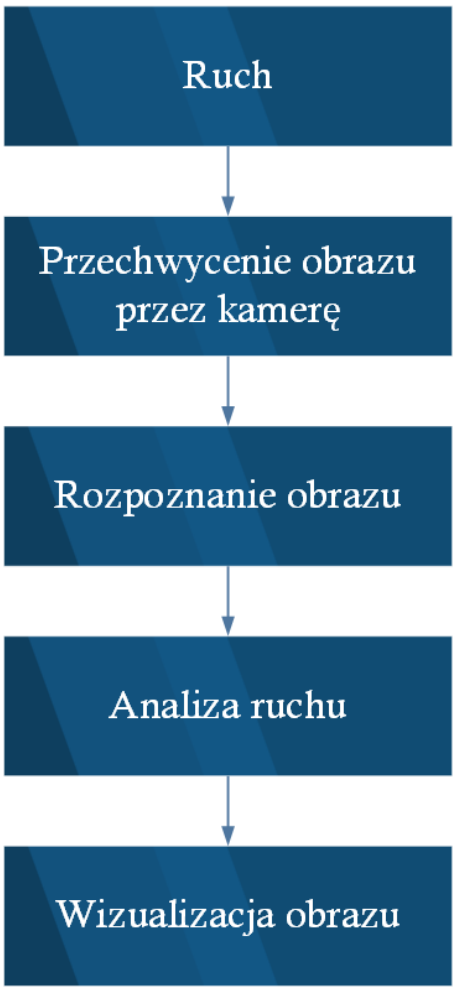
6.2 Diagram sekwencji



Rysunek 11 *- diagram sekwencji UML rozgrywki gry w warcaby*

Powyżej zamieszczono diagram sekwencji UML rozgrywki gry w warcaby. Polega on na zilustrowaniu na wykresie operacji wykonywanych w systemie. Użytkownik wykonuje ruch o czym informowany jest system, który go rozpoznaje. Dalsze działanie aplikacji zależy od rozpoznania obrazu. W sytuacji kiedy nie rozpoznano ruchu, system informuje użytkownika o błędzie. W przeciwnym wypadku system wywołuje moduł sprawdzania poprawności wykonanego ruchu. Jeśli ruch jest rozpoznany, informacja jest przesyłana do systemu wizualizacji i jest widoczna na ekranie aplikacji. System informuje użytkownika czy wykonany ruch został zrealizowany zgodnie z zasadami grania w warcaby, czy też przeciwnie.

6.3. Schemat działania



Schemat 2 *- schemat działania aplikacji*

Po wykonaniu ruchu przez gracza kamera przechwytuje obraz planszy i pozycje pionków. Następnie przekazuje macierz z pozycjami do modułu rozpoznawania obrazu, a potem do analizy ruchu. Sprawdzamy poprawność wykonania ruchu i w przypadku nie wykrycia żadnych nieprawidłowości dane są przekazywane do wizualizacji.

7. Napotkane problemy i ich rozwiązania

7.1 Problemy

7.1.1 Oświetlenie

Głównymi napotkanymi problemami są takie dotyczące oświetlenia i kamery. Obraz z kamery różni się zależnie od panujących warunków, a konieczne jest żeby odwzorowywał jak najdokładniej rzeczywistość. Należy zwrócić uwagę na padanie cienia obiektów. Dodano również możliwość konfiguracji parametrów dotyczących masek. Pozwala to dostosować kolory rozpoznawane przez kamerę. Sposoby rozwiązania problemów:

- odpowiednie oświetlenie
- stabilne ustawienie kamery
- widoczność planszy w obrazie uzyskanym z kamery
- widoczność w kamerze znaczników umieszczonych na planszy
- zapobieganie padania cienia na kamerowany obraz
- utrzymanie odpowiednich warunków
- zapobieganie pogorszeniu warunków
- nie ruszanie kamerą
- możliwość dostosowania parametrów konfiguracyjnych

7.1.2 Połączenie modułów

- CLI - błąd formatu danych
- CLI - nieprawidłowa przestrzeń nazw,
- różne zastosowania tych samych metod
- różne reprezentacje macierzy

7.1.3 CLI

Kolejnym problemem było użycie CLI, ponieważ było to nasze pierwsze użycie tej funkcjonalności. Zapoznanie i nauka zajęła nam sporo czasu, jednakże poradziliśmy sobie przy pomocy dokumentacji. Dodatkowo musieliśmy zainstalować ten pakiet na każdym komputerze. Zastosowanie tej możliwości spowodowało trudności podczas łączenia modułów projektu w jedną całość.

7.1.4 Współpraca zasad

W module sprawdzania poprawności zasad, trudnością było zaimplementowanie wszystkich zasad, żeby one ze sobą współgrały i nie kolidowały. Wykonano to przy pomocy wielokrotnych testów.

7.1.5 Wybór technologii

Ponadto zastanawialiśmy się nad wyborem odpowiednich technologii. Analizowaliśmy wybór języka programowania pomiędzy C++, Java, C# oraz Python. Rozpatrzyliśmy również różne środowiska programowania jak i sposób implementacji interfejsu oraz biblioteki graficzne.

8. Instrukcja użytkowania aplikacji

8.1 Przygotowanie

Do użycia aplikacji konieczne jest posiadanie komputera ze środowiskiem Visual Studio z zainstalowaną biblioteką OpenCV oraz rozszerzeniem CLI do języka C++. Ponadto należy posiadać planszę w kształcie szachownicy i 24 pionki do gry w warcaby po 12 danego koloru. Pionki muszą mieścić się w polu planszy. Dodatkowo konieczne jest posiadanie kamery komputerowej oraz umieszczenie jej nad planszą w taki sposób aby obejmowała całą. Kamera nie może znajdować się w niestabilnej pozycji, która groziłaby pobraniem niepoprawnego obrazu z kamery. Plansza musi zawierać markery, aby było możliwe wykrycie jej przez aplikację.

Jedynymi zadaniami, z jakimi mierzą się użytkownicy są:

- ustawienie kamery w polu widzenia,
- umieszczenie markerów na planszy,
- zapoznanie się z zasadami gry,
- obsługa aplikacji

8.2 Schemat rozrywki

Grę rozpoczyna gracz z białymi pionkami. Gracze wykonują ruchy na przemian (zgodnie z zasadami opisanymi w punkcie 3 - Obowiązujące zasady poprawności ruchów). Gra kończy się, w momencie gdy któryś z graczy zwycięża lub zostanie osiągnięty remis.
Tura pierwszego gracza rozpoczyna się od podjęcia decyzji jaki ruch zostanie wykonany, gracz chwyta pionek i przesuwa go na dane pole lub zbija inny pionek, czy wykonuje inną z dostępnych akcji, w tym czasie kamera przechwytuje obraz i moduł rozpoznawania przekazuje informację do systemu wizualizacji, aby zaktualizować podgląd należy kliknąć przycisk „Kolejny ruch”. Wszystkie niepotrzebne figury odkładamy na bok, aby nie zostały na planszy.

8.3 Środowisko

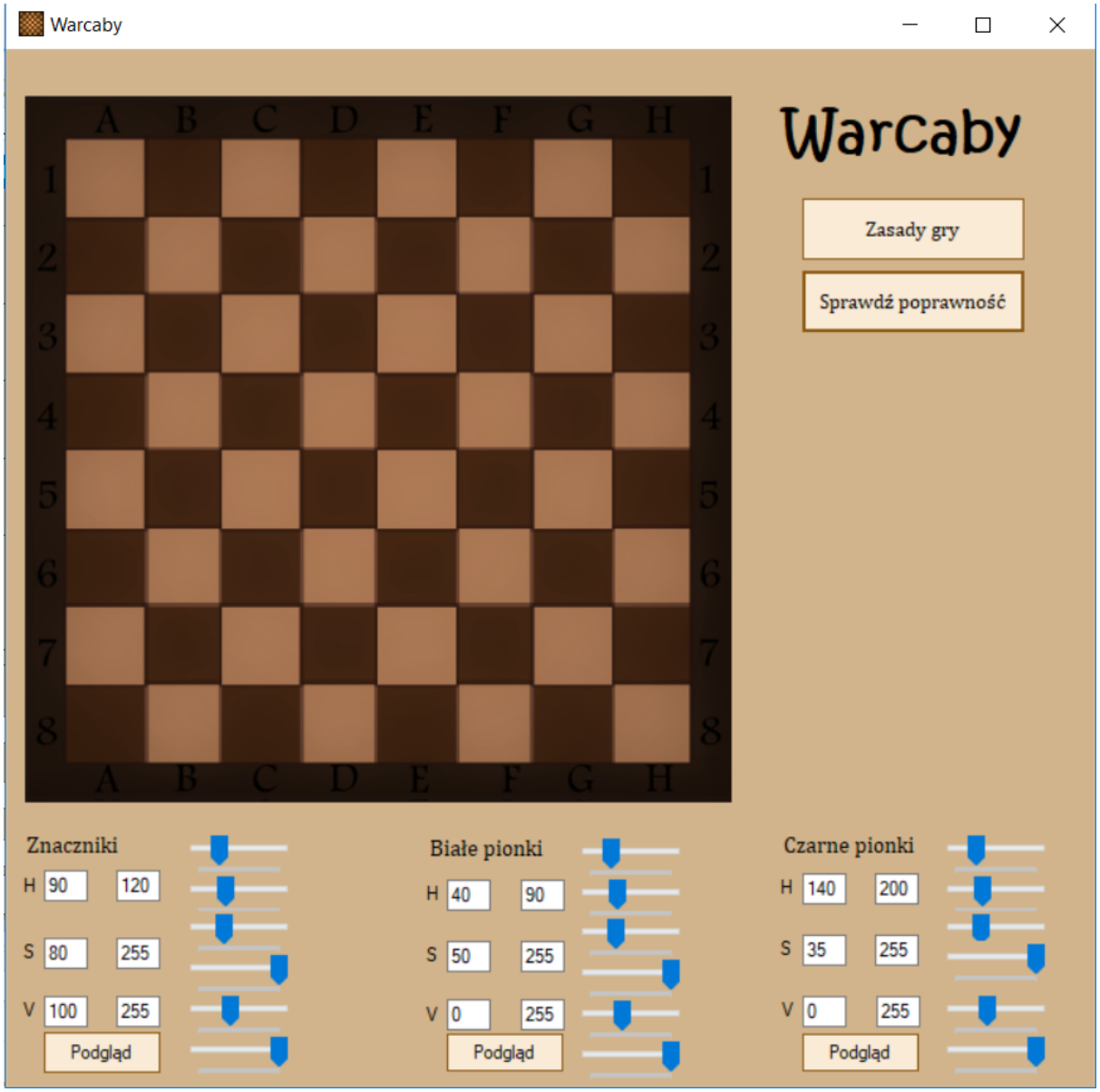
Moduł rozpoznawania obrazu musi działać w świecie rzeczywistym, gdzie w ogólnym przypadku może ulegać zmianie. Dobrym przykładem jest natężenie światła, które ulega zmianie podczas rozgrywki w zależności od pogody za oknem lub szachownica może ulec przesunięciu lub zasłonięciu przez obiekt. Spotykamy się z wieloma takimi sytuacjami i ciężko jest je obsłużyć. Spowodowało to podjęcie decyzji o wprowadzeniu dodatkowych założeń odnośnie otoczenia, w którym rozgrywa się gra.

8.3.1. Warunki zewnętrzne

Zrealizowane oprogramowanie musi działać, gdy:

- oświetlenie w pomieszczeniu nie może powodować odblasków na planszy i być stałe,
- plansza nie może wychodzić poza obszar widoczności kamery,
- szachownica cały czas znajduję się w ustalonej pozycji - nie może być pochylona, ani przekręcona

8.4 Instrukcja



Rysunek 12 *- Screen aplikacji po uruchomieniu*

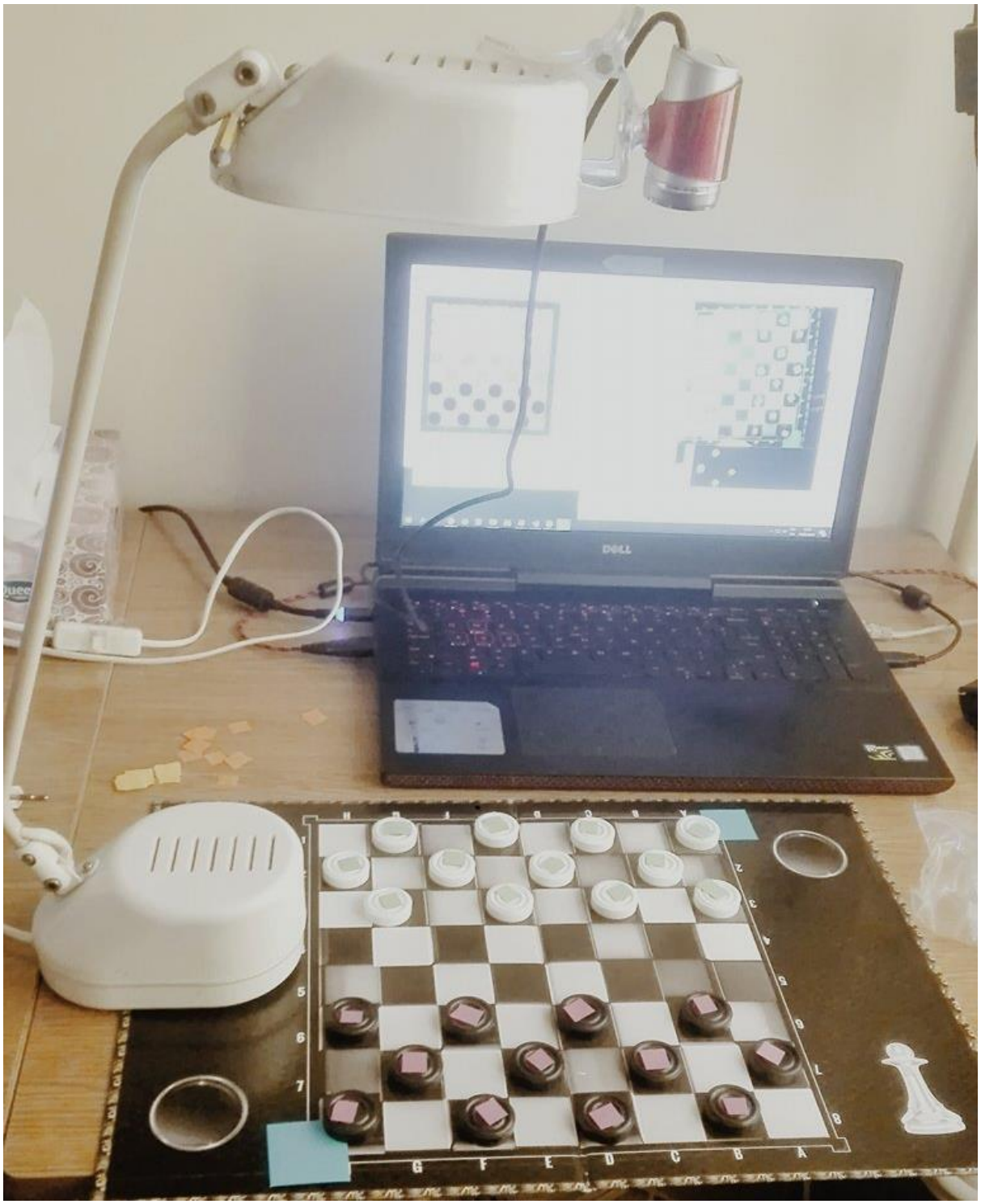
Po uruchomieniu aplikacji ukazuje nam się ekran,na którym mamy do użycia przycisk „Zasady gry” oraz „Sprawdź poprawność”. Aby zacząć grę, musimy nacisnąć drugi. Używamy go, gdy zawodnicy wykonają ruch i chcemy, aby obraz został przechwycony na komputer. Ponadto na dole znajdują się suwaki umożliwiające dokonanie konfiguracji parametrów dotyczących maski. Taka konfiguracja pomaga w dostosowaniu koloru podczas rozpoznawania obrazu oraz pomaga zapobiec problemom z oświetleniem.

Rysunek 13 *- Screen ukazujący zasady gry*

Przycisk z napisem “Zasady gry” służy do wyświetlania spisu zasad gry w warcaby tradycyjne. Po naciśnięciu ukazuje się lista z kolejno wymienionymi regułami wprowadzonymi do systemu. Gdy wykonamy niepoprawny ruch możemy w każdej chwili wrócić do tego okna i zapoznać się ponownie. Pozwala to na doskonalenie własnych umiejętności gry w warcaby

Rysunek 14 *- Screen ukazujący komunikat o błędzie*

Komunikat o błędzie pokazuje się w momencie wykonania ruchu, który nie jest zgodny z zasadami wprowadzonymi do systemu. W przypadku takiego zajścia należy cofnąć manewr oraz wykonać nowy - tym razem prawidłowy i ponownie nacisnąć przycisk “Sprawdź poprawność”. Dzięki temu użytkownik poznaje zasady gry w warcaby. Na powyższym screenie błąd pojawił się w sytuacji kiedy użytkownik przeniósł pionek na niedozwolone białe pole.



Rysunek 15 *- Zdjęcie planszy z pionkami oraz kamery*

Zdjęcie przedstawia fizyczne ustawienie planszy na biurku, na której ustawione są pionki białe i czarne, na których znajdują się dwukolorowe znaczniki. Obok planszy znajduje się wysoka lampa do której przymocowano kamerę internetową przy pomocy taśmy klejącej. Kamera jest podłączona do komputera i kameruje planszę wraz z pionami oraz przesyła ten obraz do komputera.

9. Podsumowanie

9.1. Wnioski

Celem głównym naszego projektu czyli stworzenie rozpoznawania obrazu gry w warcaby i wizualizacja stanu gry na komputerze został osiągnięty. Wszystkie wymagania zostały spełnione. System pozwala na przeprowadzenie rozgrywki i podgląd w modelu wizualizacji. Weryfikujemy wszystkie wykonane manewry oraz zgłaszamy, gdy został wykonany nieprawidłowy. Dodatkowo, system posiada mechanizmy pozwalające reagować na niektóre nieoczekiwane zdarzenia. Należy do nich sytuacja, w której szachownica zostanie znacznie przysłonięta, a także sytuacja, gdy pionek leży nieprecyzyjnie na polu. Należy poprawić nieprawidłowość i dopiero można wrócić do rozgrywki. Przeprowadziliśmy również testy, które potwierdziły działanie aplikacji.