

# Dokumentacja projektu **WARCABY**

---

Prowadzący zajęcia: mgr inż. Przemysław Walkowiak

Rok 3 Informatyki na Wydziale Elektrycznym, tryb stacjonarny

**Temat projektu :** *Rozpoznawanie obrazu z gry warcaby  
oraz wizualizacja stanu gry na komputerze*

## Dane osobowe grupy

JUSTYNA KAWCZYŃSKA

JOANNACHROMIŃSKA

IWONAMARAŚKIEWICZ

## Spis treści

Cel aplikacji .....	2
Uzasadnienie wyboru tematu .....	2
Założenia projektowe .....	2
Wykorzystane technologie .....	2
Wykres Gantta .....	4
Wymagania funkcjonalne aplikacji .....	4
Zadania osiągnięte w projekcie .....	8
Interfejs aplikacji .....	9
Zaimplementowane funkcjonalności .....	11
Opis pól i zmiennych w programie .....	11
Opis metod używanych w programie .....	13
Testowanie .....	29
Stanowisko .....	35

## Cel aplikacji

Celem pracy było stworzenie systemu zapisującego przebieg gry planszowej – warcaby, wykorzystując informację jaką dostarcza obraz wideo. Program ma za zadanie rozpoznać poszczególne figury na planszy wraz z ich ruchami. Warcaby wymagają poza rozróżnieniem koloru figur również jakiego rodzaju to figura (zwykły pionek czy tzw. „damka”). Informacje te są istotne i już na wstępnych etapach planowania musiały być wzięte pod uwagę. Zestawy warcabów posiadają cechy wspólne jak plansza na bazie kwadratu, niezmienna ilość pól (64 pola ułożone 8x8 z na przemian leżącymi kolorami) taka sama ilość i rodzaj figur oraz położenie początkowe. Jednak występują istotne różnice uniemożliwiające stworzenie uniwersalnego programu działającego z wszystkimi zestawami warcabów. Poszczególne zestawy różnią się między sobą kolorami pól (pola ciemne mogą być koloru czarnego lub w jednym z odcieni brązu), rodzajem materiału z jakiego są wykonane. Należało więc zastosować takie mechanizmy przetwarzania obrazu aby w jak największym stopniu uwzględnić wpływ tych czynników na działanie programu dając jak największą uniwersalność. Dlatego aplikacja przez nas stworzona została dopasowana tylko do konkretnego zestawu warcabów (w przyszłości prawdopodobnie powrócimy do projektu i udoskonalimy jego działanie oraz postaramy się stworzyć go uniwersalnym).

## Uzasadnienie wyboru tematu

Rejestracja, przetwarzanie i analiza obrazu jako techniki stosowane współcześnie coraz częściej w wielu dziedzinach życia.

## Założenia projektowe

- Stworzenie systemu zapisującego przebieg gry planszowej na przykładzie gry w warcaby, wykorzystując informację jaką dostarcza obraz wideo.
- Rozpoznawanie przez program poszczególnych pionków na planszy wraz z ich ruchami oraz dokonywanie zapisu – wizualizacja stanu gry na komputerze.

## Wykorzystane technologie

Przystępując do realizacji projektu wybrano odpowiednie narzędzia (tj. język programowania, urządzenia rejestrujące obraz), dzięki którym możliwe było łatwe i szybkie przeprowadzenie procesu rejestracji gry w warcaby, analizy i rozpoznania ruchów.

Do tworzenia projektu został wybrany język C# (Visual Studio 2015), ponieważ aplikacje napisane w nim nie wymagają dodatkowych programów aby mogły działać, a dzięki prostocie i możliwościom nadaje się do realizacji zagadnień związanych z przetwarzaniem obrazu. Głównym powodem wyboru tego języka

programowania były biblioteki, które mogliśmy wykorzystać w programie w celu interpretacji obrazu na komputerze (OpenCV) oraz możliwość wykorzystania techniki 3D w celu pokazania aktualnego stanu gry (SharpGL, biblioteki umożliwiającej skorzystanie z OpenGL w aplikacji napisanej w języku C# w technologii .Net). Wybrałyśmy do tego graficzną wersję programowania tego języka (Windows Form) w celu uzyskania programu zbliżonego wyglądem do gry komputerowej. Wersja z przyciskami i odpowiednimi okienkami w znacznym stopniu ułatwia nawigację oraz wygląd programu.

Biblioteka OpenCV w znacznym stopniu ułatwiły nam interpretację obrazu odczytanego z kamery. Po pierwsze możliwe było odczytywanie obrazu z kamery „na żywo”. Po drugie po umieszczeniu obrazu w odpowiednim miejscu w programie możliwe było zinterpretowanie i wyznaczenia pól czarnych używanych do gry (kalibracja planszy). Po trzecie możliwe było określenie położenia pionków na planszy.

## Biblioteka OpenCV

OpenCV jest to zbiór funkcji i aplikacji połączonych w jedną bibliotekę, stworzonych na potrzeby grafiki komputerowej w celu obróbki obrazu. Jest doskonałym narzędziem wykorzystywanym w systemach wizyjnych czasu rzeczywistego dzięki swojej wydajności, a zarazem prostocie obsługi. Stworzona została przez firmę Intel. Wersje OpenCV wcześniejsze niż 2.0 były napisane w języku C, i stworzone tak aby miały charakter obiektowy. Osiągnięte zostało to poprzez utworzenie trzech statycznych struktur IplImage, CvMat, CvArr reprezentujących obraz wraz z osobnymi funkcjami operującymi na tych strukturach.

Biblioteka daje możliwości:

- odczyt i zapis obrazu z kamery lub pliku
- tworzenie interfejsu użytkownika (okna, suwaki, obsługa myszy)
- operacje na macierzach i wektorach
- filtracja obrazu
- transformacja obrazu
- dostęp do pikseli lub obszarów obrazu
- wykrywanie ruchu
- wykrywanie obiektów
- rozpoznawanie kształtów
- rysowanie na obrazie

## Wykres Gantta

Zadania	Podzadania	Osoba	Termin rozpoczęcia	Termin zakończenia
Projektowanie interface'u		Joanna Chromińska	T1	T2
Projektowanie aplikacji		Justyna Kawczyńska	T1	T3
Testowanie aplikacji		Justyna Kawczyńska	T8	T9
Tworzenie aplikacji	Utworzenie logiki aplikacji	Justyna Kawczyńska Joanna Chromińska Iwona Maraśkiewicz	T3	T5
	Utworzenie interfejsu graficznego aplikacji		T6	T8
	Poprawki		T8	T9
Tworzenie dokumentacji		Iwona Maraśkiewicz Justyna Kawczyńska Joanna Chromińska	T1	T10

## Wymagania funkcjonalne aplikacji

### 1. Identyfikator wymagania : K001 (Kamera)

**Treść :** Możliwość sczytania obrazu z kamery w czasie rzeczywistym

**Istotność dla odbiorcy :** 2

**Reprezentowany punkt widzenia :** developer

**Ważność dla konstrukcji całego oprogramowania :** krytyczne

**Przewidywana stabilność/podatność na zmiany :** stabilne

**Wymagane wejście :** obraz przesyłany z kamery

**Sposób przetwarzania :** pobranie obrazu wysyłanego przez kamerę w czasie rzeczywistym

**Sposób przetwarzania użytkownik :** wciśnięcie przycisku „START”

**Oczekiwane wyjście :** wyświetlenie obrazu z kamery w czasie rzeczywistym

**Uwagi dotyczące testowania :** sprawdzenie poprawności oraz jakości wyświetlanego obrazu

2. *Identyfikator wymagania* : K002 (Kamera)

*Treść* : Możliwość sczytania koordynatów planszy z obrazu przesyłanego przez kamerę w czasie rzeczywistym

*Istotność dla odbiorcy* : 2

*Reprezentowany punkt widzenia* : developer

*Ważność dla konstrukcji całego oprogramowania* : krytyczne

*Przewidywana stabilność/podatność na zmiany* : niestabilne

*Wymagane wejście* : obraz przesyłany z kamery

*Sposób przetwarzania* : sczytanie koordynatów narożników planszy z obrazu wysyłanego przez kamerę w czasie rzeczywistym

*Sposób przetwarzania użytkownik* : wciśnięcie przycisku „START”

*Oczekiwane wyjście* : koordynaty planszy

*Uwagi dotyczące testowania* : sprawdzenie poprawności pobranych danych

3. *Identyfikator wymagania* : W001 (Wizualizacja)

*Treść* : Wizualizacja planszy

*Istotność dla odbiorcy* : 3

*Reprezentowany punkt widzenia* : użytkownik

*Ważność dla konstrukcji całego oprogramowania* : krytyczne

*Przewidywana stabilność/podatność na zmiany* : stabilne

*Wymagane wejście* : punkty środków pól planszy obliczone na podstawie obrazu przesyłanego przez kamerę w czasie rzeczywistym (na podstawie koordynatów planszy)

*Sposób przetwarzania* : obliczenie wielkości planszy, wielkości pól oraz położenia pól w odpowiednich miejscach na zwizualizowanej planszy

*Sposób przetwarzania użytkownik* : wciśnięcie przycisku „SPRAWDŹ PLANSZĘ”

*Oczekiwane wyjście* : wyświetlenie zwizualizowanej planszy gry w warcaby

*Uwagi dotyczące testowania* : sprawdzenie poprawności wyświetlonej planszy, wielkości planszy – czy pola obliczone na podstawie koordynatów sczytanych z obrazu wysyłanego przez kamerę zgadzają się z przypisanymi koordynatami na planszy wirtualnej, czy pola na planszy wirtualnej zostały odpowiednio przypisane

4. *Identyfikator wymagania* : K003 (Kamera)

*Treść* : Możliwość sczytania pozycji pionków z obrazu przesyłanego przez kamerę w czasie rzeczywistym

*Istotność dla odbiorcy : 2*

*Reprezentowany punkt widzenia : developer*

*Ważność dla konstrukcji całego oprogramowania : krytyczne*

*Przewidywana stabilność/podatność na zmiany : niestabilne*

*Wymagane wejście : obraz przesyłany z kamery*

*Sposób przetwarzania :* czytanie pozycji położonych pionków na planszy z obrazu wysyłanego przez kamerę w czasie rzeczywistym

*Sposób przetwarzania użytkownik : wciśnięcie przycisku „WYKRYJ PIONKI”*

*Oczekiwane wyjście : współrzędne położenia pionków na planszy*

*Uwagi dotyczące testowania : sprawdzenie poprawności pobranych danych*

#### 5. *Identyfikator wymagania : W002 (Wizualizacja)*

*Treść : Wizualizacja pionków na planszy*

*Istotność dla odbiorcy : 3*

*Reprezentowany punkt widzenia : użytkownik*

*Ważność dla konstrukcji całego oprogramowania : krytyczne*

*Przewidywana stabilność/podatność na zmiany : stabilne*

*Wymagane wejście : punkty współrzędne pionków na planszy oraz odczytanie rodzaju i koloru pionka pobrane (obliczone) na podstawie obrazu przesyłanego przez kamerę w czasie rzeczywistym*

*Sposób przetwarzania : obliczenie miejsca położenia pionków w odpowiednich miejscach na zwizualizowanej planszy, dopasowanie odczytanego koloru pionka do właściciela oraz rodzaju pionków*

*Sposób przetwarzania użytkownik : wciśnięcie przycisku „WYKRYJ PIONKI”*

*Oczekiwane wyjście : wyświetlenie zwizualizowanej pionków na planszy*

*Uwagi dotyczące testowania : sprawdzenie poprawności wyświetlanych pionków – ich położenia, wyświetlanego rodzaju, koloru (przypisania pionka do odpowiedniego rodzaju) – czy koordynaty czytane z obrazu wysyłanego przez kamerę zgadzają się z przypisanymi koordynatami na planszy wirtualnej, czy pionki na planszy wirtualnej zostały odpowiednio przypisane*

#### 6. *Identyfikator wymagania : W003 (Wizualizacja)*

*Treść : Aktualizacja planszy – położenia pionków na planszy*

*Istotność dla odbiorcy : 3*

*Reprezentowany punkt widzenia : użytkownik*

*Ważność dla konstrukcji całego oprogramowania : krytyczne*

*Przewidywana stabilność/podatność na zmiany* : niestabilne

*Wymagane wejście* : koordynaty położenia pionków na planszy oraz odczytanie rodzaju i koloru pionka pobrane (obliczone) na podstawie obrazu przesyłanego przez kamerę w czasie rzeczywistym

*Sposób przetwarzania* : obliczenie miejsca położenia pionków w odpowiednich miejscach na zwizualizowanej planszy, dopasowanie odczytanego koloru pionka do właściciela oraz rodzaju pionków

*Sposób przetwarzania użytkownik* : wciśnięcie przycisku „RUCH PRZECIWNIKA”

*Oczekiwane wyjście* : zaktualizowanie położenia pionków na planszy – wyświetlenie nowej wizualizacji planszy zgodnej z rzeczywistym obrazem

*Uwagi dotyczące testowania* : sprawdzenie poprawności wyświetlanych pionków – ich położenia, wyświetlanego rodzaju, koloru (przypisania pionka do odpowiedniego rodzaju) – czy współrzędne sczytane z obrazu wysyłanego przez kamerę zgadzają się z przypisanymi współrzędnymi na planszy wirtualnej, czy pionki na planszy wirtualnej zostały odpowiednio przypisane

## 7. *Identyfikator wymagania* : S001 (Stan gry)

*Treść* : Wyświetlenie informacji o stanie gry

*Istotność dla odbiorcy* : 2

*Reprezentowany punkt widzenia* : użytkownik

*Ważność dla konstrukcji całego oprogramowania* : ważne

*Przewidywana stabilność/podatność na zmiany* : niestabilne

*Wymagane wejście* : współrzędne położenia pionków na planszy oraz odczytanie rodzaju i koloru pionka pobrane (obliczone) na podstawie obrazu przesyłanego przez kamerę w czasie rzeczywistym

*Sposób przetwarzania* : zliczenie ilości pionków każdego rodzaju

*Sposób przetwarzania użytkownik* : wciśnięcie przycisku „WYKRYJ PIONKI” lub „RUCH PRZECIWNIKA”

*Oczekiwane wyjście* : informacje o obecnym stanie gry – wyświetlenie liczby pionków i damek należących do gracza zielonego oraz pionków i damek należących do gracza czerwonego.

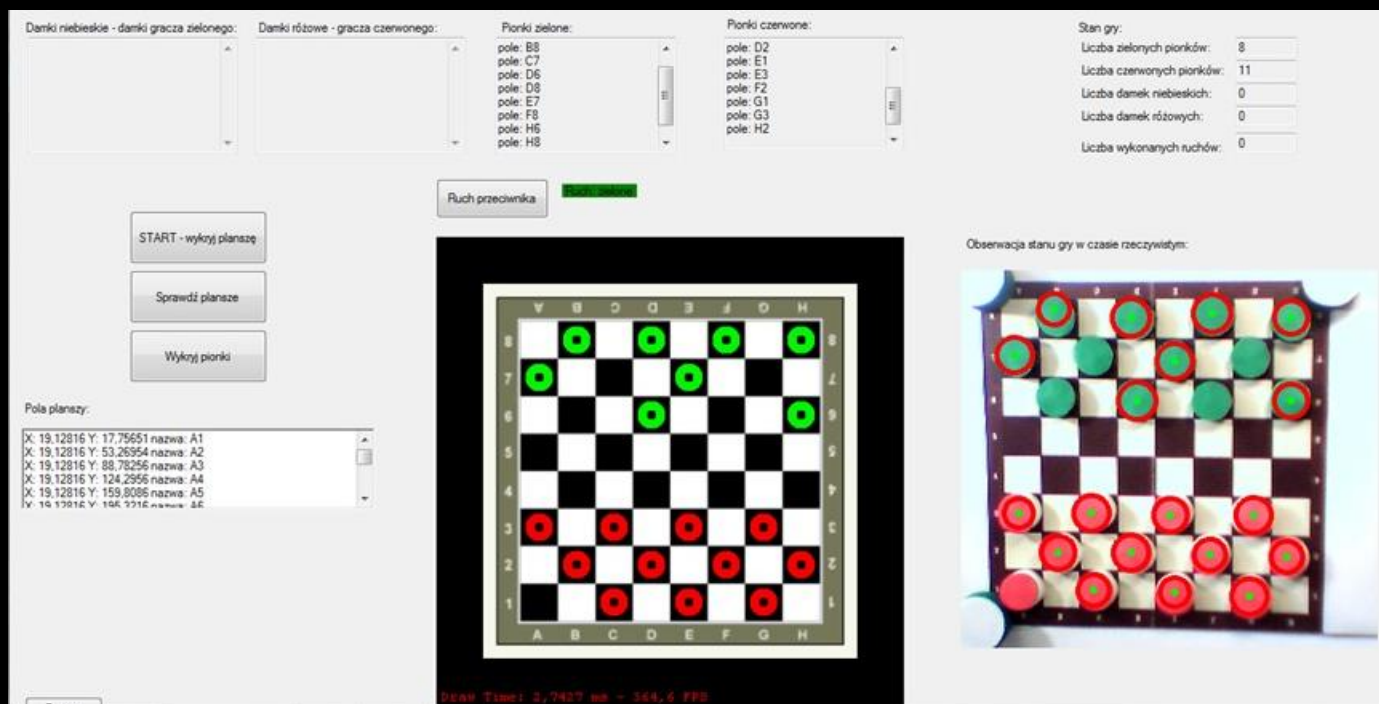
*Uwagi dotyczące testowania* : sprawdzenie poprawności liczby wyświetlanych pionków – według rodzaju, koloru (przypisania pionka do odpowiedniego rodzaju)

## Zadania osiągnięte w projekcie

- Zaprojektowanie interfejsu
- Zintegrowanie kamery internetowej ze środowiskiem OpenCV:
  - obserwacja stanu gry w czasie rzeczywistym
  - rozpoznawanie pojawiających się przed kamerą okrągłych obiektów – pionków gry (skorzystanie z metody HoughCircles)
  - wypisywanie ich pozycji (współrzędnych X, Y) oraz promienia okręgu
- Dodanie do projektu graficznej biblioteki SharpGL:
  - narysowanie obiektu przedstawiającego planszę gry
  - nałożenie tekstury szachownicy na planszę
- Rozpoznawanie planszy:
  - dodanie trzech różnokolorowych koordynatów na rogach planszy oraz określenie ich pozycji
  - wyliczenie współrzędnych środków pól planszy
  - przyporządkowanie poszczególnym polom planszy nazw za pomocą współrzędnych szachownicy (np. A3, E1, H8, itp.)
- Rysowanie pionków na planszy wirtualnej – wizualizacja stanu gry
  - rysowanie okrągłych obiektów (pionków) w polach szachownicy odpowiadającym rzeczywistemu ułożeniu pionków na planszy (z wykorzystaniem biblioteki SharpGL)
- Wypisywanie informacji o stanie gry:
  - liczba pozostałych pionków na planszy
  - liczba pozostałych pionków na planszy gracza czerwonego
  - liczba pozostałych pionków na planszy gracza zielonego
  - liczba pozostałych damek różowych na planszy gracza czerwonego
  - liczba pozostałych damek niebieskich na planszy gracza zielonego
  - liczba wykonanych ruchów
  - pozycje (nazwy współrzędnych szachownicy) pionków gracza zielonego
  - pozycje (nazwy współrzędnych szachownicy) pionków gracza czerwonego
  - pozycje (nazwy współrzędnych szachownicy) różowych damek gracza czerwonego
  - pozycje (nazwy współrzędnych szachownicy) niebieskich damek gracza czerwonego



## Interfejs aplikacji



Interfejs aplikacji składa się z dwóch głównych części, tj. część odpowiedzialnej za wizualną reprezentację stanu gry oraz część odpowiedzialną za reprezentację gry w postaci danych.

Część wizualna składa się z okna wyświetlającego obraz z kamery w czasie rzeczywistym, gdzie można zauważyć oznaczone okręgami pionki – program identyfikuje pionki na planszy sprawdzając ich położenie, oraz okno reprezentujące stan gry w formie planszy 2D z zidentyfikowanymi pionkami.

Część przedstawiająca użytkownikom dane stanu gry składa się z okna obecnych pionków na planszy: pionki zielone oraz czerwone (tu: lista na jakich polach znajdują się pionki gracza), pionki niebieskie (damki gracza zielonego) oraz pionki różowe (damki gracza niebieskiego). Na tą część składa się także okno „Pola planszy”, gdzie obrazowane są współrzędne środków pól sczytanych przez program z planszy, np. Pole (jego punkt środkowy), o nazwie A4, ma współrzędne  $X = 19.12016$ ;  $Y = 124.29156$ . W prawym górnym rogu znajdują się także „Stan gry”, która składa się ze spisu liczby pionków oraz damek każdego z graczy.

Na interfejs aplikacji składają się również 4 przyciski odpowiadające kolejno za :



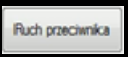
„START – wykryj planszę” – uruchamia część programu odpowiedzialną za połączenie z kamerą, sczytanie obrazu z kamery oraz znalezienie planszy, czyli sczytanie koordynatów narożników planszy, dzięki którym określa się wielkość planszy, jej położenie oraz można określić kolejne pola.

Sprawdź planszę

„Sprawdź planszę” – uruchamia proces odpowiedzialny za wyliczenie szerokości i długości planszy oraz na tej podstawie obliczenie współrzędnych środków pól planszy. Dzięki tym czynnościom i pobranych danych możliwe jest następnie narysowanie planszy wirtualnej. Program wykonuje wizualizację samej planszy – na obiekt narysowany przy użyciu graficznej biblioteki SharpGL nakłada teksturę (gotową bitmapę). Dopasowuje odpowiednio umiejscowienie pól. Aby podczas położenia pionka np. na polu H3 program miał możliwość zwizualizowania tego pionka na takim samym polu (H3) na planszy wirtualnej.

Wykryj pionki

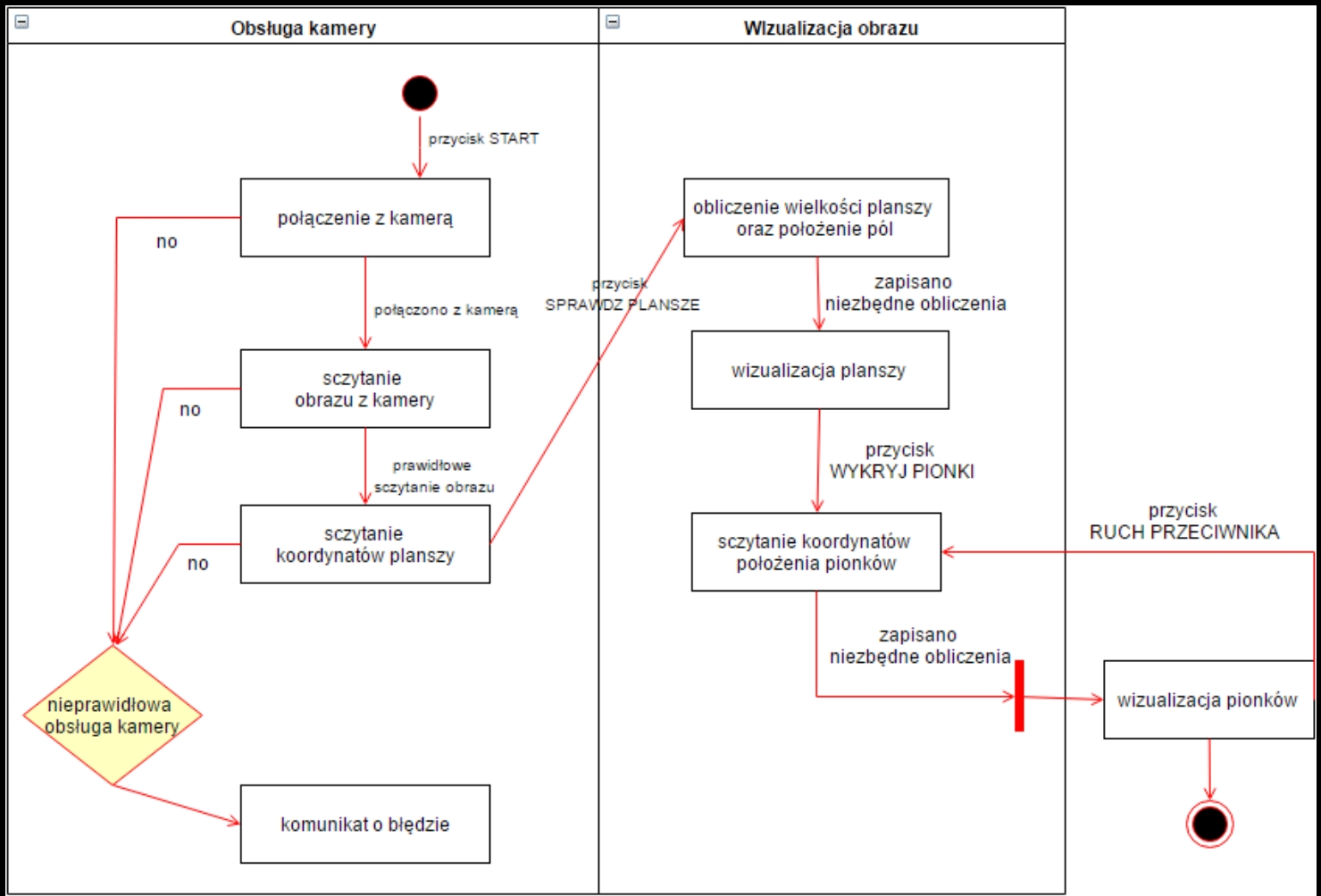
„Wykryj pionki” – przycisk ten uruchamia proces szytywania z obrazu pobieranego z kamery w czasie rzeczywistym położenia pionków oraz ich identyfikacji. Program lokalizuje kolejne pionki na szachownicy, której obraz jest pobierany/szytywany z kamery – określa pole, na jakim pionek się znajduje oraz określa typ pionka (jakiego jest koloru – do jakiego gracza należy, oraz czy jest to zwykły pionek czy damka). Następnie rysuje pionki na wirtualnej szachownicy w odpowiednim miejscu.

Ruch przeciwnika

„Ruch przeciwnika” – przycisk służący do zakomunikowania programowi o zakończonym ruchu. Podobnie jak w rzeczywistych turniejach warcabowych i szachowych gracze po zakończonym ruchu wciskają przycisk – służyć to ma możliwości rozbudowy aplikacji o zegar i możliwość „przegranej” poprzez koniec czasu oraz uruchamia on odpowiednie funkcje programu polegające przede wszystkim na rysowaniu pionków – gdy gracz zakomunikuje koniec ruchu wczytywana zostaje ponownie pozycja wszystkich pionków na planszy.



„Ruch zielone” – informują, który gracz zielony czy czerwony obecnie wykonuje ruch.



## Zaimplementowane funkcjonalności

### Opis pól i zmiennych w programie

*capWebcam* - obiekt typu Capture; służy do pobierania obrazu z kamery.

*przechwytywanie* - pole typu bool pełniące rolę flagi - znacznika. Jeśli ustawione jest na wartość true, to rzeczywisty obraz jest przechwytywany.

*zdjOryginalne* - pole typu Image, które służy do wizualizacji obrazu przechwyconego z kamery.

*zdjTworzone* - pole typu Image, w którym widać obraz przechwycony z kamery w odcieniach szarości. Odcienie szarości zastosowano w celu zredukowania szumów. Zmienna początkowo wykorzystywana w programie, następnie zrezygnowano z obrazowania filtrowanego obrazu ze względu skupienie się na intuicyjnym, przejrzystym i wygodnym dla użytkownika interfejsie.

*pola* - tablica dwuwymiarowa przechowująca obiekty typu Pole. Wpisywane są do niej pozycje X oraz Y środków pól planszy.

*tablica\_X\_zielone* - tablica typu float, w której przechowywane są wartości X dla każdego zielonego pionka znajdującego się na planszy.

*tablica\_Y\_zielone* - tablica typu float, w której przechowywane są wartości Y dla każdego zielonego pionka znajdującego się na planszy.

*tablica\_X\_czerwone* - tablica typu float, w której przechowywane są wartości X dla każdego czerwonego pionka znajdującego się na planszy.

*tablica\_Y\_czerwone* - tablica typu float, w której przechowywane są wartości Y dla każdego czerwonego pionka znajdującego się na planszy.

*tablica\_X\_niebieskie* - tablica typu float, w której przechowywane są wartości X dla każdego niebieskiego pionka znajdującego się na planszy.

*tablica\_Y\_niebieskie* - tablica typu float, w której przechowywane są wartości Y dla każdego niebieskiego pionka znajdującego się na planszy.

*tablica\_X\_zolte* - tablica typu float, w której przechowywane są wartości X dla każdego żółtego pionka znajdującego się na planszy.

*tablica\_Y\_zolte* - tablica typu float, w której przechowywane są wartości Y dla każdego żółtego pionka znajdującego się na planszy.

*gorny\_lewy\_X* , *gorny\_lewy\_Y* , *dolny\_lewy\_X* , *dolny\_lewy\_Y*, *gorny\_prawy\_X* , *gorny\_prawy\_Y* - zmienne typu float, w których przechowywane są współrzędne wierzchołków planszy. Wartości przypisywane są przy pomocy tzw. koordynatów, czyli punktów (okręgów o określonych kolorach) w trzech rogach planszy. Środek każdego koordynata ma współrzędne X i Y. Po dodaniu/odjęciu (w zależności od położenia koordynata) do tych współrzędnych długości promienia okręgu, którym jest koordynat, otrzymamy współrzędne początku planszy w każdym z trzech wierzchołków.

*rozmiar\_pola\_wys* - zmienna typu float, przyjmująca wartość wysokości pojedynczego pola

*rozmiar\_pola\_szer* - zmienna typu float, przyjmująca wartość szerokości pojedynczego pola

*polowa\_szer\_pola* - zmienna typu float, przyjmująca wartość połowy szerokości pojedynczego pola, służy do obliczania współrzędnych środka każdego pola na planszy

*polowa\_wys\_pola* - zmienna typu float, przyjmująca wartość połowy wysokości pojedynczego pola, służy do obliczania współrzędnych środka każdego pola na planszy

*czy\_wykrywanie\_pionkow* - pole typu logicznego pełniące rolę flagi - znacznika. Jeśli ma wartość true - wykonywana jest funkcja wykrywająca pionki na planszy.

*kolejna* - pole typu logicznego pełniące rolę flagi

Zmienne, do których przypisywane są pozycje X, Y oraz promień koordynatów umiejscowionych w rogach planszy. Zmienne te stosowane są w metodzie obliczającej współrzędne X i Y pól planszy na podstawie koordynatów:

*raz\_X* ,

*dwa\_X* ,

*trzy\_X* ,

*raz\_Y* ,

*dwa\_Y* ,  
*trzy\_Y* ,  
*raz\_prom* ,  
*dwa\_prom* ,  
*trzy\_prom* ,

*wysokosc* - zmienna typu float. Zawiera wysokość planszy obliczoną na podstawie koordynatów.

*szerokosc* - zmienna typu float. Zawiera szerokość planszy obliczoną na podstawie koordynatów.

*texture* - obiekt typu Texture służący do teksturowania. Dzięki niemu nakładana zostaje tekstura przedstawiająca szachownicę na obiekt narysowany za pomocą funkcji OpenGL.

### Opis metod używanych w programie

*SharpGLForm* - funkcja, w której inicjalizowane są wszystkie komponenty oraz uruchamiana zostaje kamera.

Najważniejsze elementy w tej funkcji:

```
try
{
    capWebcam = new Capture(1);
}
catch (NullReferenceException except)
{
    return;
}
SharpGL.OpenGL gl = this.openGLControl.OpenGL;

gl.Enable(OpenGL.GL_TEXTURE_2D);

texture.Create(gl, "C://Users/dom/Desktop/01_szachownica.bmp");
```

W tym miejscu tworzymy instancję obiektu *Capture*. Do konstruktora przekazujemy wartość 1, za względu na to, że korzystamy z kamery dodatkowej, podłączonej przez USB. W przypadku kamery wbudowanej nie byłoby potrzeby podawania żadnej wartości.

Tworzony jest też główny obiekt *SharpGL*, a następnie ustawiana widoczność tekstury, która zostanie utworzona poniżej.

*proces\_wykryj\_plansze(object sender, EventArgs arg)* - metoda służąca do wykrywania planszy wywoływana po wciśnięciu przycisku 'Wykryj plansze'. Poza wykrywaniem planszy następuje tutaj wyliczenie szerokości i długości szachownicy oraz wyliczenie pól planszy. Metoda wywołuje funkcję „wylicz”.

W metodzie tej następuje czytanie trzech różnokolorowych koordynatów w kolorach RGB - czerwonego, zielonego i niebieskiego. Są one przymocowane do planszy na stałe - przesuwają się razem z planszą, dzięki czemu w przypadku ponownego obliczania wymiarów planszy pozostaną one takie same.

Kamera wyznacza dla okręgów, którymi są koordynaty, pozycję X i Y oraz wyznacza ich promień. Korzysta z wbudowanych metod dostarczonych przez bibliotekę OpenCV, tj. QueryFrame – przechwycenie obrazu z kamery i wczytywanie, InRange – ustalenie zakresu czytywanych kolorów, SmoothGaussian – stosowanie filtru w celu wygładzenia i rozmycia, HoughCircles- wykrywanie okręgów o określonych kolorach, cvCircle – rysowanie małego zielonego okręgu w środku wykrytego obiektu, Draw – narysowanie czerwonego okręgu wokół wykrytego obiektu (wokół zielonego okręgu).

Dzięki funkcjom biblioteki OpenCV możemy określić kolor, który chcemy czytywać (w naszym przypadku czerwony, zielony, niebieski) oraz rozpoznawać pojawiające się przed kamerą okrągłe obiekty – koordynaty oraz pionki (skorzystanie z metody HoughCircles).

Funkcje biblioteki OpneCV, przeznaczone do wykrywania okręgów z przechwyconego przez kamerę obrazu, stosują próg Canny – operator wykrywania, wykorzystujący algorytm wielostopniowy w celu wykrycia szeregu różnych krawędzi. Jest to inaczej metoda detekcji krawędzi występujących w obrazie. Ponadto algorytm Cann’ego można stosować w wielu różnych środowiskach, a jego parametry pozwalają na takie modyfikacje, aby dostosować go do rozpoznawania krawędzi o różnych cechach, w zależności od aktualnych oczekiwań. Właściwości te ułatwiły nam wybór metody rozpoznawania okrągłych pionków gry oraz umożliwiły realizację głównych założeń i zadań projektowych, jak też uzyskanie najważniejszej funkcjonalności programu – czytywanie obrazu z gry warcaby oraz realizację wizualizacji stanu gry na komputerze.

Funkcja zawiera:

*InRange* – sprawdza, czy elementy obrazu leżą pomiędzy dwoma zmiennymi skalarnymi: wartością minimalną i maksymalną dla kolorów: niebieskiego, czerwonego oraz zielonego.

```
zdjTworzone = zdjOryginalne.InRange(new Bgr(153, 76, 0), new Bgr(255, 153, 51));
```

```
zdjTworzone2 = zdjOryginalne.InRange(new Bgr(0, 0, 255), new Bgr(153, 153, 255));
zdjTworzone3 = zdjOryginalne.InRange(new Bgr(0, 153, 76), new Bgr(178, 255, 102));
```

Zastosowanie filtru Gaussian - rozmycie w celu wygładzenia

```
zdjTworzone = zdjTworzone.SmoothGaussian(9);
zdjTworzone2 = zdjTworzone2.SmoothGaussian(9);
zdjTworzone3 = zdjTworzone3.SmoothGaussian(9);
```

Wykrywanie okręgów o określonych kolorach

```
CircleF[] circles = zdjTworzone.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0];
CircleF[] circles2 = zdjTworzone2.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0];
CircleF[] circles3 = zdjTworzone3.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0];
```

Pętle *foreach* dla okręgów każdego koloru, w których następuje wpisywanie do odpowiednich zmiennych współrzędnych wykrytych okręgów na planszy.

`foreach (CircleF circle in circles) //dla okręgów niebieskich`

```
{
    raz_X = circle.Center.X;
    raz_Y = circle.Center.Y;
    raz_prom = circle.Radius;

    CvInvoke.cvCircle(zdjOryginalne, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new
MCvScalar(0, 255, 0), -1, LINE_TYPE.CV_AA, 0);

    zdjOryginalne.Draw(circle, new Bgr(Color.Red), 3);
}
```

```
foreach (CircleF circle in circles2) // dla czerwonych - to samo co powyżej
{
    dwa_X = circle.Center.X;
    dwa_Y = circle.Center.Y;
    dwa_prom = circle.Radius;

    CvInvoke.cvCircle(zdjOryginalne, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new
MCvScalar(0, 255, 0), -1, LINE_TYPE.CV_AA, 0);

    zdjOryginalne.Draw(circle, new Bgr(Color.Red), 3);
}

foreach (CircleF circle in circles3) // dla zielonych - to samo co powyżej
{
    trzy_X = circle.Center.X;
    trzy_Y = circle.Center.Y;
    trzy_prom = circle.Radius;

    CvInvoke.cvCircle(zdjOryginalne, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new
MCvScalar(0, 255, 0), -1, LINE_TYPE.CV_AA, 0);

    zdjOryginalne.Draw(circle, new Bgr(Color.Red), 3);
}

ibOriginal.Image = zdjOryginalne;

delay = 1;
}
```

*wylicz()* - metoda służąca do wyliczania (współrzędnych i rozmiaru) pól na planszy na podstawie wykrytych koordynatów.

Każdemu polu planszy nadajemy nazwę, wyznaczoną na podstawie współrzędnych szachownicy.



```
if (raz_X == 0 || dwa_X == 0 || trzy_X == 0)
{
    MessageBox.Show("Nie wykryto planszy! Wybierz - wykryj plasze!");
}
else
{
    Pole pole = new Pole();
    gorny_lewy_X = dwa_X + dwa_prom;
    gorny_lewy_Y = dwa_Y + dwa_prom;

    dolny_lewy_X = raz_X + raz_prom;
    dolny_lewy_Y = raz_Y - raz_prom;

    gorny_prawy_X = trzy_X - trzy_prom;
    gorny_prawy_Y = trzy_Y + trzy_prom;

    wysokosc = dolny_lewy_Y - gorny_lewy_Y;
    szerokosc = gorny_prawy_X - gorny_lewy_X;

    MessageBox.Show("Wykryto planszę! Można rozpocząć grę!");
    MessageBox.Show("Wysokosc = " + wysokosc.ToString() + ". Szerokosc = " +
szerokosc.ToString());

    //rozmiar każdego pola
    rozmiar_pola_szer = szerokosc / 8;
    rozmiar_pola_wys = wysokosc / 8;

    polowa_szer_pola = rozmiar_pola_szer / 2;
    polowa_wys_pola = rozmiar_pola_wys / 2;

    float nowa_polowa_szer = polowa_szer_pola;
    float nowa_polowa_wys = polowa_wys_pola;

    char nazwa_litera = 'A';
    int nazwa_int = 1;
```

```
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        pola[i, j] = new Pole();
        pola[i, j].X = polowa_szer_pola + i * rozmiar_pola_szer;
        pola[i, j].Y = polowa_wys_pola + j * rozmiar_pola_wys;

        pola[i, j].nazwa += nazwa_litera;
        pola[i, j].nazwa += nazwa_int;

        ++nazwa_int;
    }
    ++nazwa_litera;
    nazwa_int = 1;
}

richTextBox_pola.Clear();
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        richTextBox_pola.AppendText("X: " + pola[i, j].X.ToString());
        richTextBox_pola.AppendText(" Y: " + pola[i, j].Y.ToString());
        richTextBox_pola.AppendText(" nazwa: " + pola[i, j].nazwa + "\n");
    }
}
}
```

*button\_wykryj\_plansze\_Click(object sender, EventArgs e)* - metoda wywołująca funkcję "proces\_wykryj\_plansze" oraz ustawiająca flagę "przechwytywanie".

```
Application.Idle += proces_wykryj_plansze;
przechwytywanie = true;
MessageBox.Show("Wykrywanie planszy!");

if (delay == 1)
{
    if (raz_X == 0.0 || dwa_X == 0.0 || trzy_X == 0.0)
    {
        MessageBox.Show("Nie wykryto planszy! Proszę poprawić położenie planszy!");
        Application.Idle += proces_wykryj_plansze;
        przechwytywanie = true;
    }
    else
    {
    }
}
else
{
    MessageBox.Show("Ponowne wykrywanie planszy!");
}
```

*openGLControl\_OpenGLDraw(object sender, RenderEventArgs e)* - główna funkcja OpenGL rysująca obiekty. Jest cyklicznie odświeżana w czasie uruchomienia programu, w związku z czym możliwa jest właściwa wizualizacja stanu gry – metoda periodycznie rysuje pionki na planszy wirtualnej, w miejscach odpowiadającym pozycjom pionków na planszy rzeczywistej. Zmiana położenia jakiegokolwiek pionka rzeczywistego spowoduje narysowanie go w odpowiednim miejscu na szachownicy w programie.

Główne jej zadania to rysowanie szachownicy oraz nałożenie na nią tekstury oraz rysowanie pionków, do czego wykorzystuje wiele metod z biblioteki OpenGL.

Tworzenie szachownicy:

```
SharpGL.OpenGL gl = this.openGLControl.OpenGL;
gl.Enable(OpenGL.GL_TEXTURE_2D);
gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);
```

```
gl.LoadIdentity();
gl.Translate(-4.0f, -4.0f, 0.0f);
texture.Bind(gl);

gl.Begin(OpenGL.GL_QUADS);
gl.Color(255.0f, 255.0f, 255.0f);
gl.TexCoord(0.0f, 1.0f);
gl.Vertex(0.0f, 0.0f, 1.0f);
gl.TexCoord(1.0f, 1.0f);
gl.Vertex(8.0f, 0.0f, 1.0f);
gl.TexCoord(1.0f, 0.0f);
gl.Vertex(8.0f, 8.0f, 1.0f);
gl.TexCoord(0.0f, 0.0f);
gl.Vertex(0.0f, 8.0f, 1.0f);
gl.End();
```

Rysowanie pionków:

W macierzy "pola" znajdują się pola (czerwony, zielony, niebieski, różowy) typu boolowskiego określające czy na danym polu znajduje się pionek oraz jakiego koloru on jest.

Wykorzystujemy pola macierzy do określenia jakiego koloru jest pionek oraz na jakiej pozycji narysować go na planszy wirtualnej w programie.

```
if (pola[0, 0] != null)
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            if (pola[i, j].czerwony == true)
            {
                var quadratic = gl.NewQuadric();
                gl.Color(255.0f, 0, 0);
                gl.PushMatrix();
```

```
        gl.Translate(1.2f + i*0.8 , 1.2f+j*0.8 , 2.0f);
        gl.Cylinder(quadratic, 0.30f, 0.30f, 0.20f, 32, 1);
        gl.Disk(quadratic, 0.10f, 0.30f, 32, 1);
        gl.PopMatrix();

    }
    else if (pola[i, j].zielony == true)
    {
        var quadratic = gl.NewQuadric();
        gl.Color(0, 153.0f, 0);
        gl.PushMatrix();
        gl.Translate(1.2f + i * 0.8, 1.2f + j * 0.8, 2.0f);
        gl.Cylinder(quadratic, 0.30f, 0.30f, 0.20f, 32, 1);
        gl.Disk(quadratic, 0.10f, 0.30f, 32, 1);
        gl.PopMatrix();
    }
    else if (pola[i, j].niebieski == true)
    {
        var quadratic = gl.NewQuadric();
        gl.Color(0, 0, 153.0f);
        gl.PushMatrix();
        gl.Translate(1.2f + i * 0.8, 1.2f + j * 0.8, 2.0f);
        gl.Cylinder(quadratic, 0.30f, 0.30f, 0.20f, 32, 1);
        gl.Disk(quadratic, 0.10f, 0.30f, 32, 1);
        gl.PopMatrix();
    }
    else if (pola[i, j].rozowy == true)
    {
        var quadratic = gl.NewQuadric();
        gl.Color(255.0f, 0, 127.0f);
        gl.PushMatrix();
        gl.Translate(1.2f + i * 0.8, 1.2f + j * 0.8, 2.0f);
        gl.Cylinder(quadratic, 0.30f, 0.30f, 0.20f, 32, 1);
        gl.Disk(quadratic, 0.10f, 0.30f, 32, 1);
        gl.PopMatrix();
    }
```

```
        }  
    }  
}  
}  
}  
}
```

*button1\_Click(object sender, EventArgs e)* - metoda zliczająca liczbę wykonanych ruchów. Jeśli gracze chcą, aby ich ruchy były liczone, powinni wciskać przycisk po każdym ruchu.

Metoda wyświetla również informację, który gracz wykonuje aktualny ruch.

```
if (flaga == 0)  
{  
    label_ruch.BackColor = Color.Green;  
    label_ruch.Text = "Ruch: zielone";  
    flaga = 1;  
    liczba_wykonanych_ruchow++;  
    textBox_stan_ruchy_wykonane.Text = liczba_wykonanych_ruchow.ToString();  
}  
else if (flaga == 1)  
{  
    label_ruch.BackColor = Color.Red;  
    label_ruch.Text = "Ruch: czerwone";  
    flaga = 0;  
    liczba_wykonanych_ruchow++;  
    textBox_stan_ruchy_wykonane.Text = liczba_wykonanych_ruchow.ToString();  
}
```

*proces\_wykryj\_pionki(object sender, EventArgs arg)* - metoda odpowiedzialna za wykrywanie pionków i damek znajdujących się na planszy.

Działa podobnie jak metoda wykrywająca koordynaty (*proces\_wykryj\_plansze*), jednakże czytuje okręgi w kolorach: czerwonym, zielonym, różowym oraz niebieskim.

W metodzie tej jest również zaimplementowane wypisywanie informacji o stanie gry:

- pozycji pionków czerwonych i zielonych,
- pozycji damek różowych i niebieskich,
- liczbie pionków na planszy,
- liczbie wykonanych ruchów.

W metodzie tej korzystamy również z funkcji biblioteki Opencv, przeznaczonych do wykrywania okręgów z przechwyconego przez kamerę obrazu, które stosują próg Canny – operator wykrywania, wykorzystujący algorytm wielostopniowy w celu wykrycia szeregu różnych krawędzi. Jest to inaczej metoda detekcji krawędzi występujących w obrazie. Ponadto algorytm Canny’ego można stosować w wielu różnych środowiskach, a jego parametry pozwalają na takie modyfikacje, aby dostosować go do rozpoznawania krawędzi o różnych cechach, w zależności od aktualnych oczekiwań. Właściwości te ułatwiły nam wybór metody rozpoznawania okrągłych pionków gry oraz umożliwiły realizację głównych założeń i zadań projektowych, jak też uzyskanie najważniejszej funkcjonalności programu – sczytywanie obrazu z gry warcaby oraz realizację wizualizacji stanu gry na komputerze.

```
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        pola[i, j].zielony = false;
        pola[i, j].czerwony = false;
        pola[i, j].rozowy = false;
        pola[i, j].niebieski = false;
    }
}
```

```
SharpGL.OpenGL gl = this.openGLControl.OpenGL;
```

```
zdyOryginalne = capWebcam.QueryFrame(); // wczytywanie obrazu przechwyconego z kamery
if (zdyOryginalne == null)
    return;
```

```
zdyTworzone2 = zdyOryginalne.InRange(new Bgr(0, 0, 255), new Bgr(153, 153, 255));
```

```
zdzTworzone = zdjOryginalne.InRange(new Bgr(0, 153, 76), new Bgr(178, 255, 102));  
zdzTworzone3 = zdjOryginalne.InRange(new Bgr(153, 76, 0), new Bgr(255, 153, 51));  
zdzTworzone4 = zdjOryginalne.InRange(new Bgr(127, 0, 255), new Bgr(153, 151, 255));
```

```
zdzTworzone = zdjTworzone.SmoothGaussian(9);  
zdzTworzone2 = zdjTworzone2.SmoothGaussian(9);  
zdzTworzone3 = zdjTworzone3.SmoothGaussian(9);  
zdzTworzone4 = zdjTworzone4.SmoothGaussian(9);
```

```
CircleF[] circles = zdjTworzone.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0]; //
```

zielone

```
CircleF[] circles2 = zdjTworzone2.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0]; //
```

czerwone

```
CircleF[] circles3 = zdjTworzone3.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0]; //
```

niebieskie

```
CircleF[] circles4 = zdjTworzone4.HoughCircles(new Gray(85), new Gray(40), 2, 30, 1, 30)[0]; //
```

rozowe

Pętla *foreach* wykrywająca zielone pionki na planszy oraz obliczająca ich pozycje. Dla każdego koloru pionków zaimplementowana jest osobna pętla.

```
foreach (CircleF circle in circles)
```

```
{
```

```
    zielony_X = circle.Center.X;
```

```
    zielony_Y = circle.Center.Y;
```

```
    zielony_prom = circle.Radius;
```

```
    CvInvoke.cvCircle(zdjOryginalne, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new  
    MCvScalar(0, 255, 0), -1, LINE_TYPE.CV_AA, 0);
```

```
    zdjOryginalne.Draw(circle, new Bgr(Color.Red), 3);
```

```
    double d2_min = 123456789;
```

```
    int i_min = 0;
```

```
    int j_min = 0;
```



```
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        double dx = (circle.Center.X - pola[i, j].X) - 2*polowa_szer_pola;
        double dy = (circle.Center.Y - pola[i, j].Y) - 2*polowa_wys_pola;
        double d2 = dx * dx + dy * dy;

        if(d2 < d2_min)
        {
            d2_min = d2;
            i_min = i;
            j_min = j;
        }
    }
}

if (d2_min < 5000000000)
{
    if (j_min == 7)
    {
        j_min = 0;
    }
    else if (j_min == 6)
    {
        j_min = 1;
    }
    else if (j_min == 5)
    {
        j_min = 2;
    }
    else if (j_min == 4)
    {
        j_min = 3;
    }
}
```

```
        else if (j_min == 3)
        {
            j_min = 4;
        }
        else if (j_min == 2)
        {
            j_min = 5;
        }
        else if (j_min == 1)
        {
            j_min = 6;
        }
        else if (j_min == 0)
        {
            j_min = 7;
        }
        pola[i_min, j_min].zielony = true;
    }
    else
    {
    }
}
```

Wypisywanie informacji o stanie gry:

```
int liczba_zielonych = 0;
int liczba_czerwonych = 0;
int liczba_niebieskich = 0;
int liczba_rozowych = 0;
polazielone.Clear();
polaczerwone.Clear();
damki_czerwone.Clear();
damki_zielone.Clear();
for (int i = 0; i < 8; i++)
{
```

```
for (int j = 0; j < 8; j++)
{
    if (pola[i, j].zielony == true)
    {
        polazielone.AppendText("pole: " + pola[i, j].nazwa + "\n");
        liczba_zielonych++;
    }
    else if (pola[i, j].czerwony == true)
    {
        polaczerwone.AppendText("pole: " + pola[i, j].nazwa + "\n");
        liczba_czerwonych++;
    }
    else if (pola[i, j].niebieski == true)
    {
        damki_zielone.AppendText("pole: " + pola[i, j].nazwa + "\n");
        liczba_niebieskich++;
    }
    else if (pola[i, j].rozowy == true)
    {
        damki_czerwone.AppendText("pole: " + pola[i, j].nazwa + "\n");
        liczba_rozowych++;
    }
}
}
```

```
textBox_stan_zielone_pionki.Text = liczba_zielonych.ToString();
textBox_stan_czerwone_pionki.Text = liczba_czerwonych.ToString();
textBox_damki_niebieskie.Text = liczba_niebieskich.ToString();
textBox_damki_rozowe.Text = liczba_rozowych.ToString();
```

```
ibOriginal.Image = zdjOryginalne;
```

*button\_pionki\_Click(object sender, EventArgs e)* - metoda, w której wywoływany jest proces odpowiedzialny za wykrywanie pionków ("proces\_wykryj\_pionki").

```
int licznik = 0;
if (przechwytywanie == true && czy_wykrywanie_pionkow == true && kolejna == true )
{
    Application.Idle -= proces_wykryj_pionki;
    przechwytywanie = false;
    kolejna = false;
}
else
{
    Application.Idle += proces_wykryj_pionki;
    przechwytywanie = true;
    czy_wykrywanie_pionkow = true;
    kolejna = true;
    licznik++;
}
```

button\_pomoc\_Click(object sender, EventArgs e) - przycisk, po wciśnięciu którego wyświetlana jest informacja o zasadach gry oraz korzystania z programu.

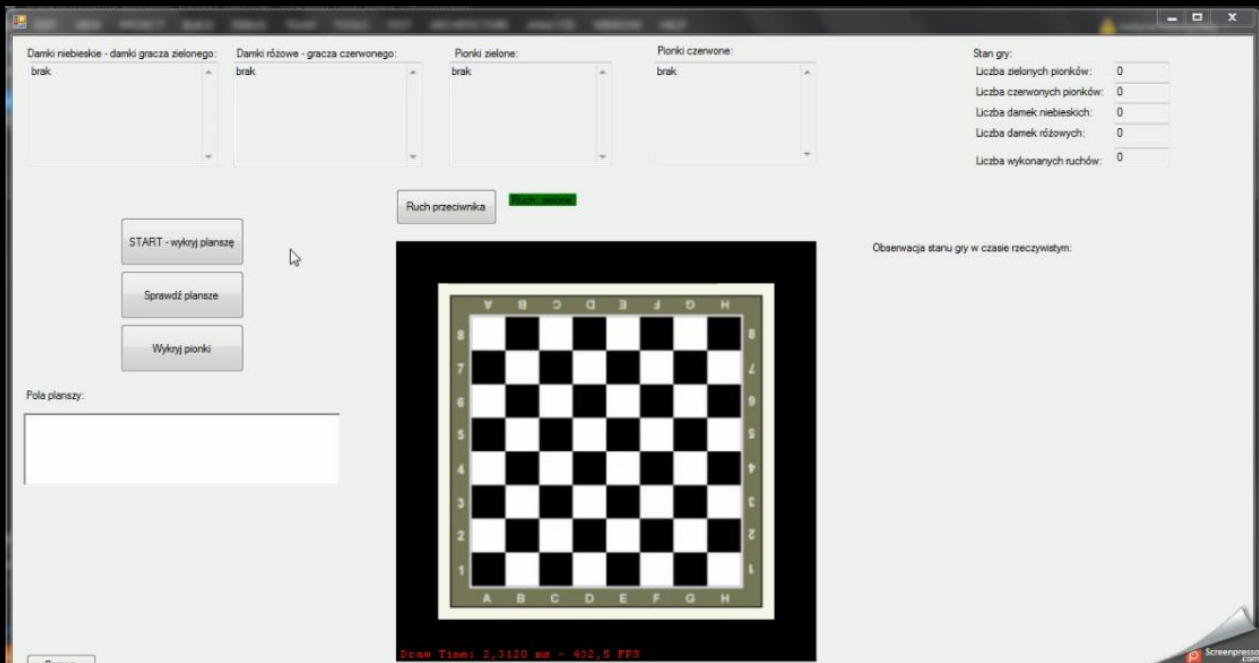
Pole - klasa opisująca pojedyncze pole planszy. Zawiera informacje tj. współrzędne środka pola X i Y, nazwę oraz kolor pionka, który znajduje się na danym polu lub czy jest ono puste.

```
public class Pole
{
    public float X = 0;
    public float Y = 0;
    public string nazwa = "";
    public bool czerwony = false;
    public bool zielony = false;
    public bool rozowy = false;
    public bool niebieski = false;
```

}

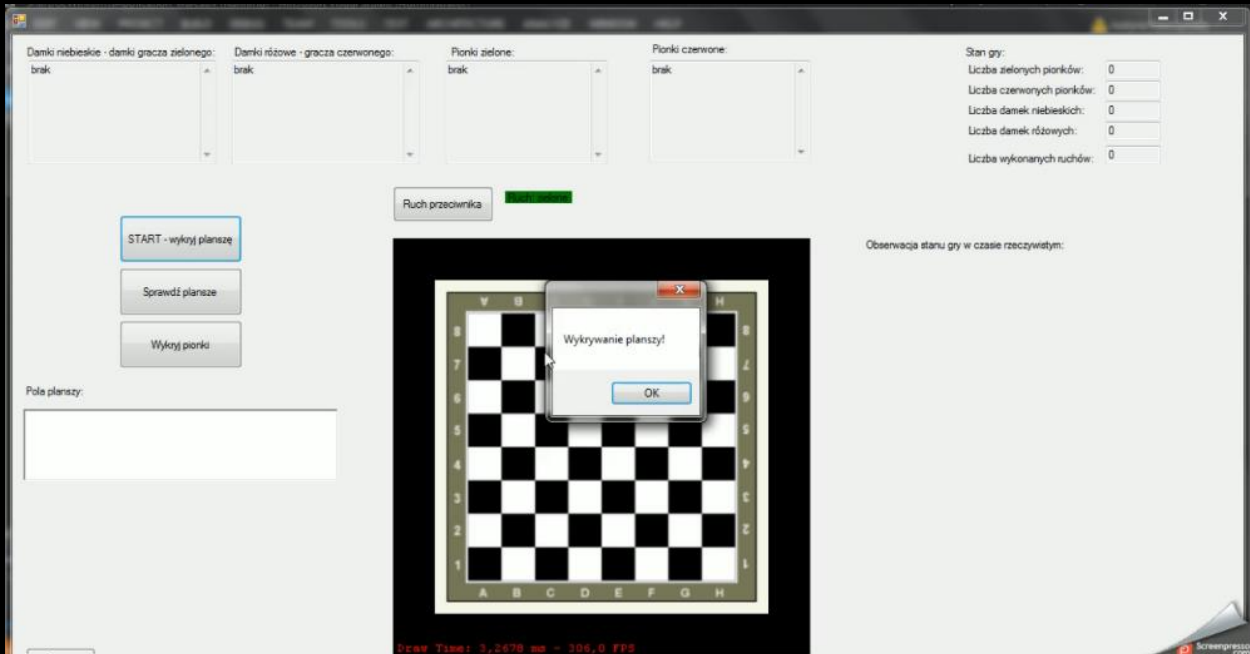
## Testowanie

### 1. Uruchomienie programu.



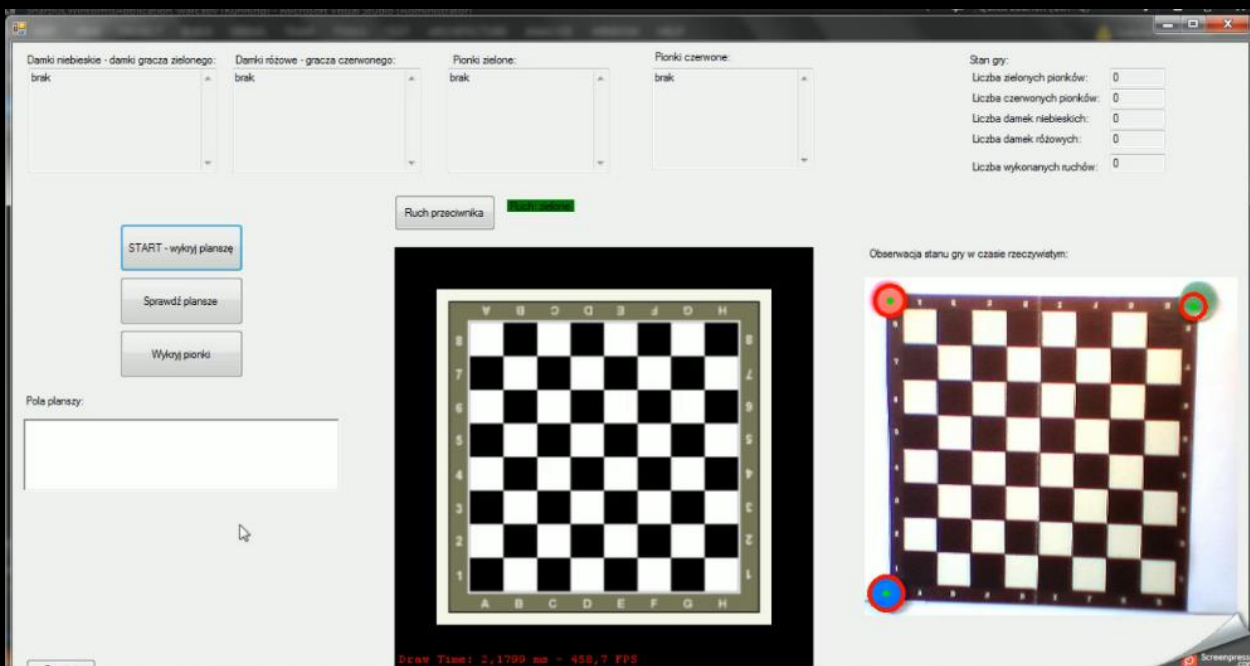
Rys.1. Początkowy wygląd programu.

## 2. Wybór opcji „START – wykryj planszę” – rozpoczęcie gry.



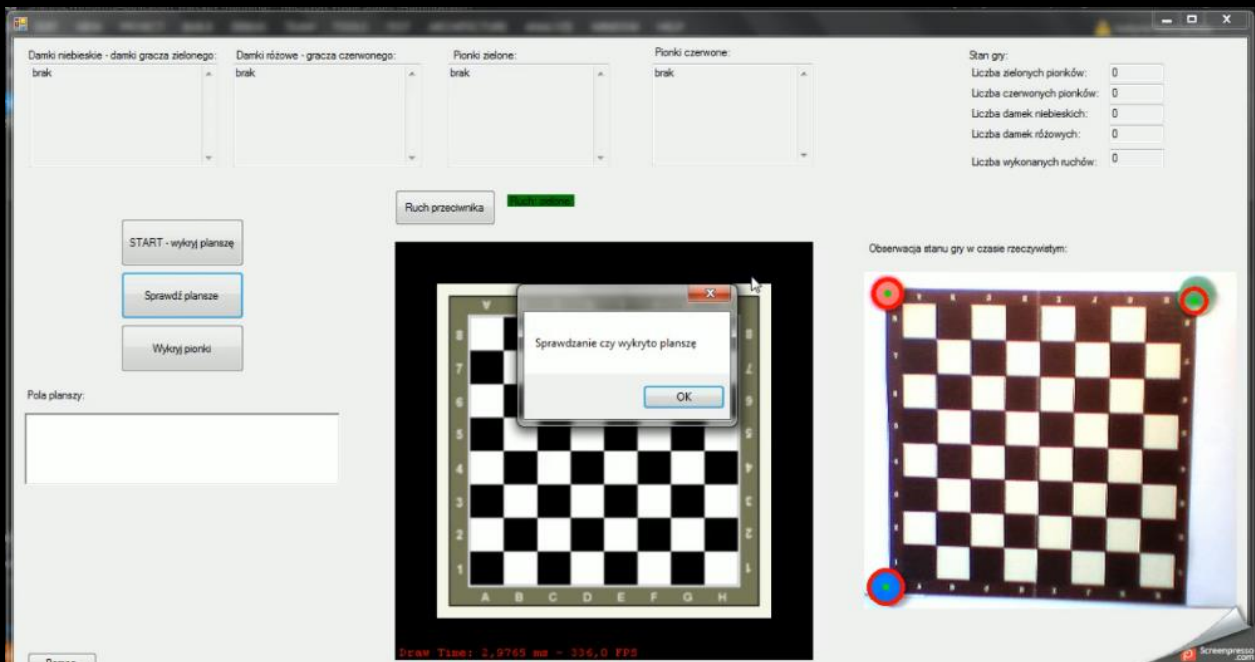
Rys.2. Po wyborze przycisku „START – wykryj planszę” pojawienie się komunikatu z informacją o rozpoczęciu wykrywania planszy.

## 3. Wykrywanie planszy.



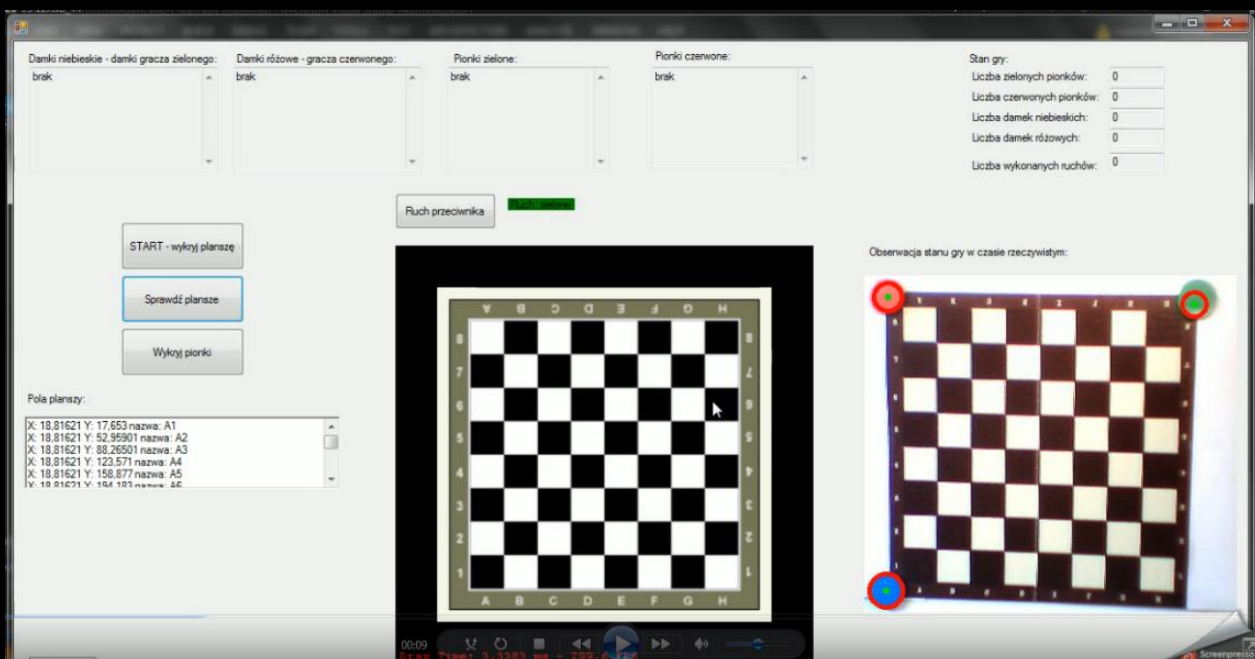
Rys.3. Pojawienie się w prawej części okna programu planszy rzeczywistej – obrazu przechwyconego przez kamerę. Widoczne trzy różnokolorowe koordynaty na rogach planszy oraz narysowane okręgi wokół rozpoznanych obiektów.

#### 4. Sprawdzenie planszy – przycisk „Sprawdź planszę”.



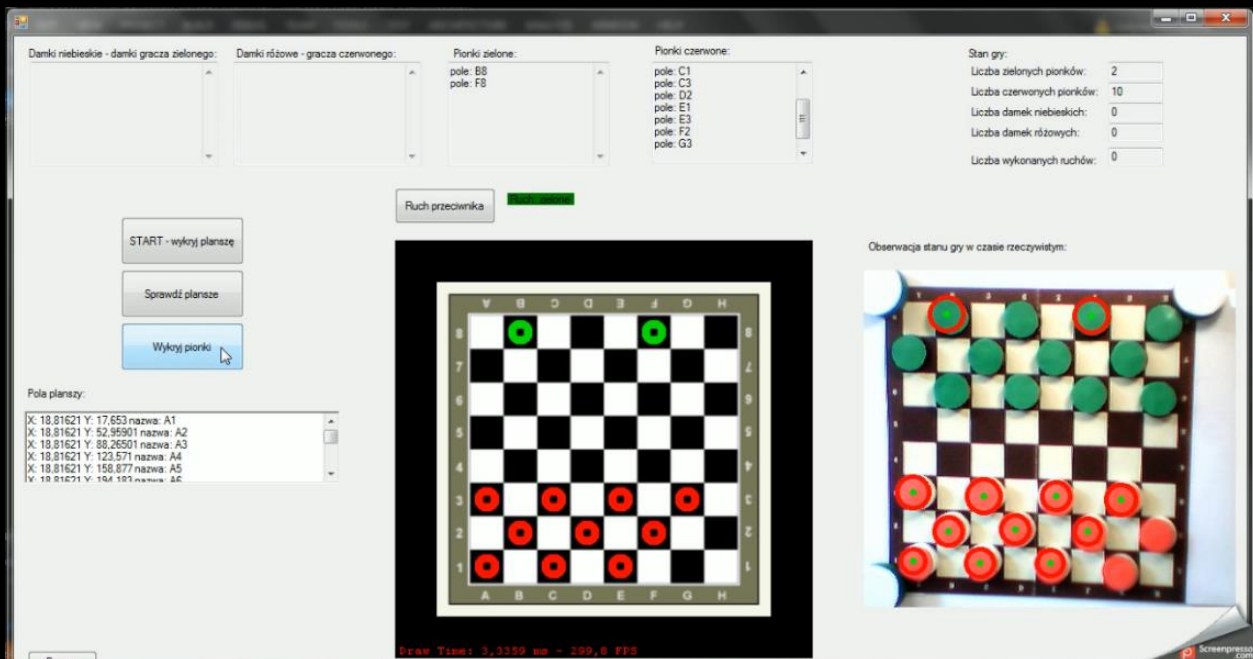
Rys.4. Pojawienie się komunikatu o rozpoczęciu sprawdzenia, czy plansza została poprawnie wykryta.

#### 5. Pojawienie się nazw wykrytych pól planszy oraz ich współrzędnych.



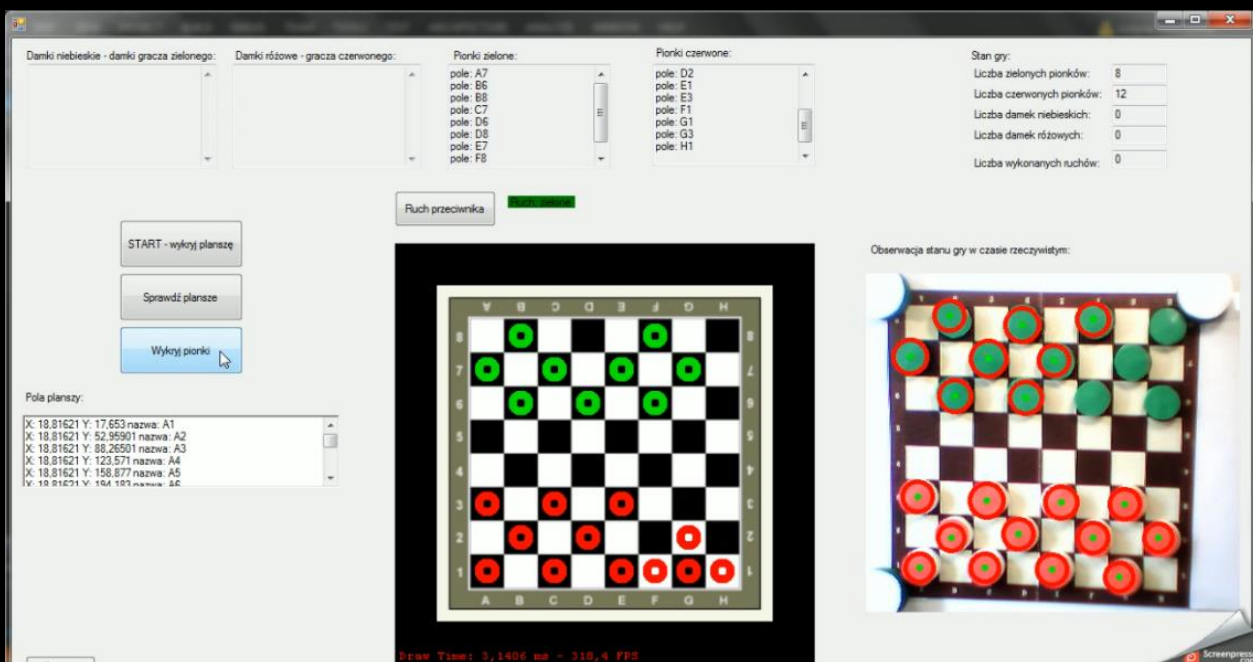
Rys.5. Pojawienie się współrzędnych pól wykrytej planszy oraz ich nazwy, nadane odpowiednio według współrzędnych danej szachownicy.

## 6. Wykrywanie pionków – przycisk „Wykryj pionki”.



Rys.6. Możliwość obserwacji stanu gry w czasie rzeczywistym. Pojawienie się na planszy przechwyconego obrazu przez kamerę internetową – szachownicy z pionkami gracza czerwonego oraz pionkami gracza zielonego.

## 7. Wizualizacja stanu gry.

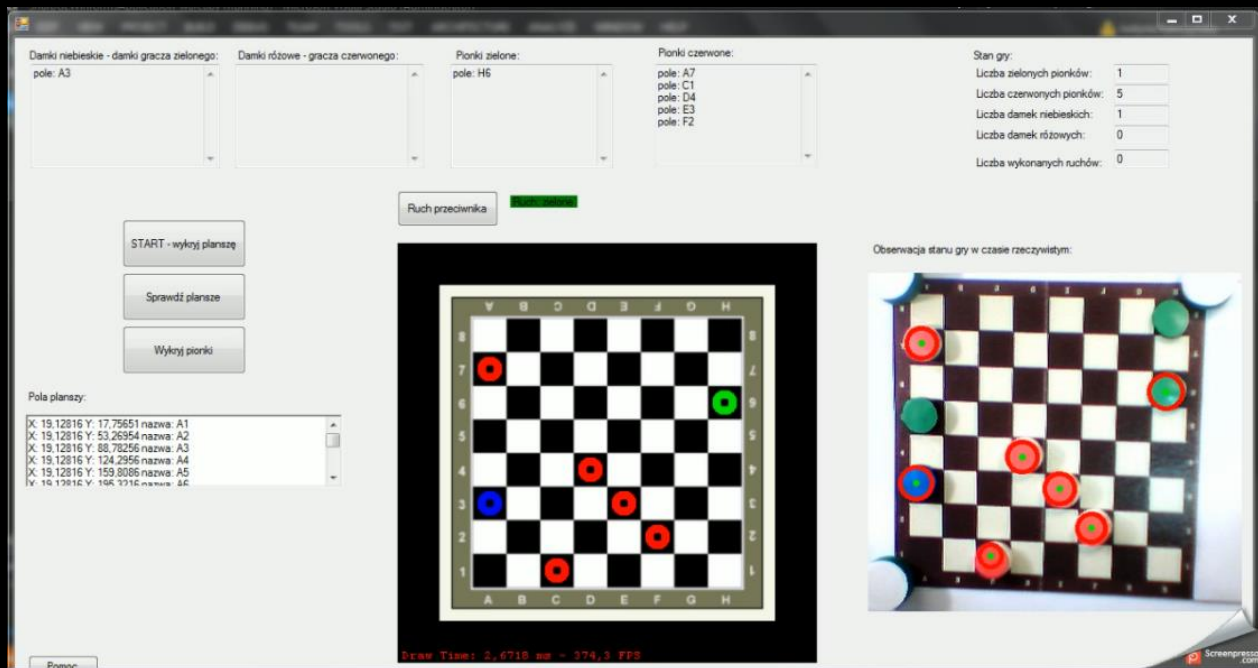


Rys.7. Wizualizacja stanu gry – na planszy wirtualnej w programie zostają narysowane pionki na pozycjach odpowiadającym pozycjom pionków na planszy rzeczywistej, w kolorze czerwonym i



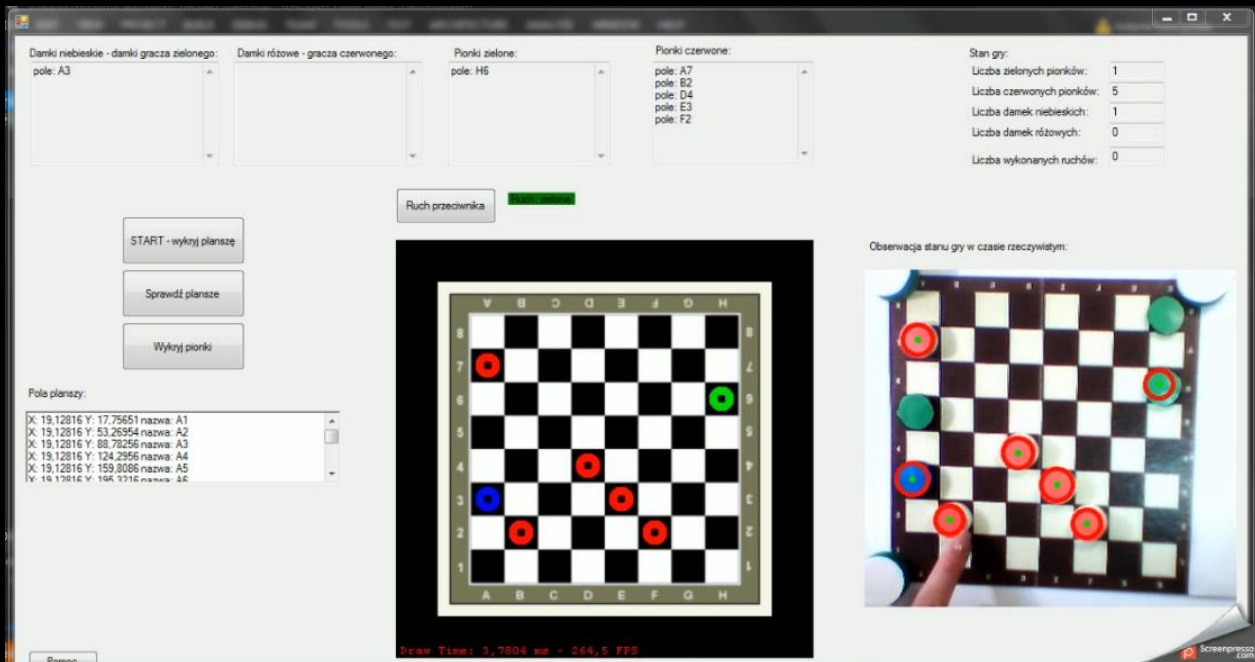
zielonym. W górnej części programu widoczne informacje o stanie gry – wypisanie pozycji pionków gracza zielonego oraz gracza czerwonego oraz pojawienie się informacji o ilości poszczególnych pionków na planszy.

## 8. Pojawienie się pionka typu „damka” na szachownicy.



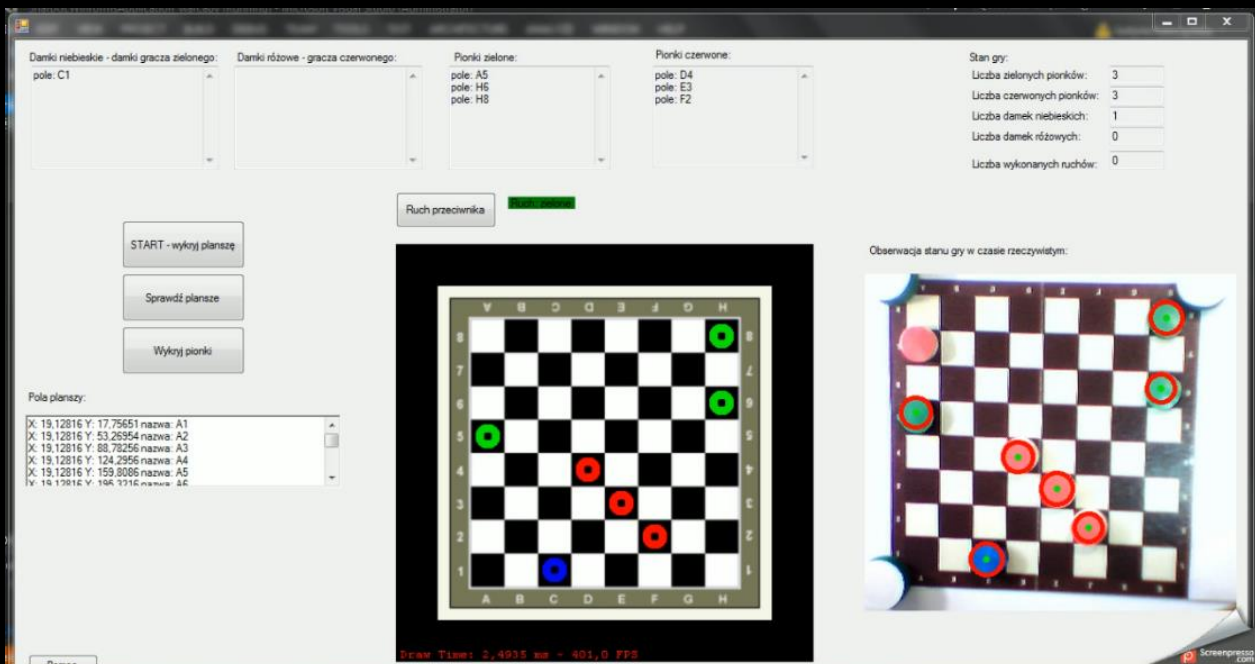
**Rys.8.** Trwanie gry – pojawienie się pionka niebieskiego typu „damka” gracza zielonego. Wypisanie stanu gry – informacji na jakich pozycjach znajdują się poszczególne pionki zielone i czerwone, jak również pionek niebieski – „damka” oraz wypisanie informacji o liczbie poszczególnych pionków na planszy.

## 9. Trwanie gry – zmiana pozycji pionków.



**Rys.9.** Trwanie gry – gracz czerwony zmienia pozycję swojego pionka. Narysowanie pionka w nowej pozycji na wirtualnej szachownicy w programie oraz wypisanie informacji o aktualnym stanie gry – nowej pozycji pionka czerwonego oraz wypisanie informacji o liczbie pozostałych pionków na planszy.

## 10. Trwanie gry – „bicie” pionka.



**Rys.10.** Trwanie gry – Pionek typu „damka” zbija pionek gracza czerwonego. Zmiana liczby pionków czerwonych oraz zmiana pozycji pionka niebieskiego typu „damka”. Zobrazowanie zmian na planszy

wirtualnej w programie oraz wypisanie nowych, aktualnych informacji o stanie gry w górnej części programu.

## Stanowisko

Na stanowisko składają się odpowiednie elementy:

- stojak wykonany z aluminiowej wygiętej rurki – pręta oraz drewnianej podstawki (widoczny na poniższych zdjęciach)
- plansza gry – szachownica o 64 polach (8x8) równych rozmiarów z trzema koordynatami na jej rogach w następujących kolorach: czerwonym, zielonym i niebieskim (widoczna głównie na Rys.14., jak również na Rys.11,13 i 15),
- kamera internetowa podłączona do komputera portem USB, przymocowana na stojaku, skierowana w kierunku planszy (widoczne na Rys.11 oraz Rys.12)
- laptop z uruchomioną aplikacją – programem przeznaczonym do rozpoznawania stanu gry warcaby oraz wizualizacji gry.



Rys.11. Stojak z umocowaną kamerą internetową skierowaną w kierunku planszy.



Rys.12. Kamera umocowana na stojaku.

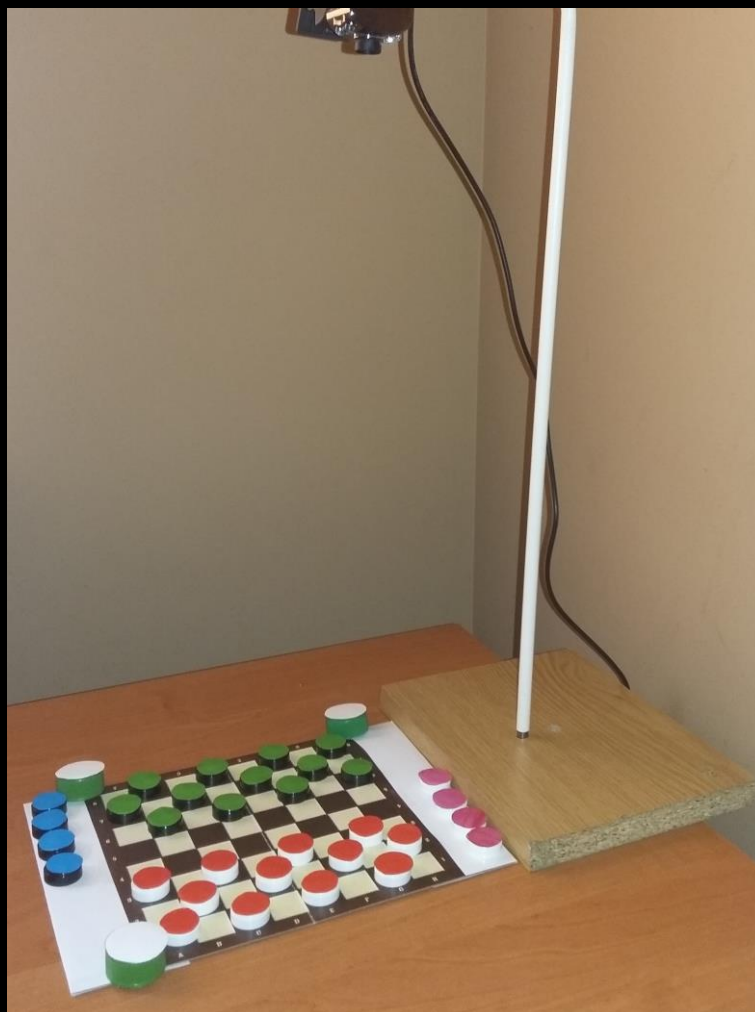


Rys.13. Plansza – szachownica 8x8 z trzema różnokolorowymi koordynatami umiejscowionymi w rogach planszy.





Rys.14. Plansza z pionkami gracza czerwonego i zielonego oraz damki niebieskie dla gracza zielonego i różowe dla gracza czerwonego.



Rys.15. Kamera umocowana na stojaku i sczytująca obraz – planszę z pionkami gry warcaby.