Projekt
Winiarz Filip 116891
Gałecki Bartosz 116885

08.06.2016

Podstawy teleinformatyki – dokumentacja

Wykrywanie stanu rozgrywki w warcabach przy pomocy OpenCV

1. **Wstęp**

• ogólna charakterystyka zadania (tematyki pracy)

Celem projektu było stworzenie aplikacji umożliwiającej rozmieszczenie pionków graczy na podstawie przechwyconego z kamery obrazu szachownicy.

Program ma za zadanie rozmieścić na wirtualnej szachownicy pionki graczy na podstawie obrazu.

W programie wykorzystywane są pionki kolorów czerwonego i zielonego.

Do wykrycia szachownicy używane są żółte markery.

Projekt został wykonany w środowisku Eclipse.

Tematyka obejmuje problem analizy obrazu z wykorzystaniem biblioteki OpenCV oraz wykorzystania biblioteki graficznej JavaFX.

• uzasadnienie podjęcia tematu.

Temat został przez nas wybrany ponieważ zainteresowało nas zagadnienie analizy obrazu. Biblioteka OpenCV była dla nas nowością oraz wyzwaniem. Zawiera ona szereg metod, które można zastosować w bardzo wielu sytuacjach i dziedzinach życia. Ponadto praca z OpenCV daje dużo satysfakcji - nie jest to typowe implementowanie kolejnej biurowej aplikacji, wymaga kreatywnego myślenia i umiejętności znajdywania zależności w obrazach i algorytmach. Efekty pracy także są interesujące do obserwacji - obrazy przedstawiające obraz wejściowy z zarysowanymi konturami detekcji lub zbinaryzowany obraz przedstawiający wykryte krawędzie i kontury. Ogólnie, jest to biblioteka mająca duży potencjał i zastosowanie w wielu ciekawych i nietypowych sytuacjach i algorytmach. Szerokie i niestandardowe zastosowanie bibliotek i algorytmów przetwarzania obrazu - oto główny powód wyboru tego tematu.

Biblioteka JavaFX, wykorzystywane przy pracy z nią pliki FXML oraz kreator to rozbudowane narzędzia. Pisząc aplikacje ułatwiają one zaprojektowanie przejrzystego interfejsu. Biblioteka ta znajdzie zastosowanie w przyszłych projektach wykorzystujących język Java.

2. **Cel i zakres pracy**

• przeznaczenie i zadania (podstawowe funkcje) projektowanego systemu

Do zadań aplikacji należą między innymi:

- przechwytywanie obrazu z kamery wbudowanej w laptop lub zewnętrznej kamery internetowej,
- analiza przechwyconego obrazu pod względem kolorów,
- przechowywanie wykrytych kolorów,
- wykrycie położenia pionków na szachownicy,

- zapisywanie historii ruchów,
- przedstawienie położenia pionków w postaci graficznej,
- ustawienia przechwytywanych kolorów i zapisywanie profilów HSV,

• **udział poszczególnych członków zespołu w realizacji zadania**

- Zadania wspólne:
  - zaplanowanie funkcjonalności systemu
  - ogólne określenie podejścia przy konstrukcji algorytmu wykrywającego
  - wstępne zaprojektowanie interfejsu aplikacji
- Filip Winiarz:
  - dopracowanie i implementacja algorytmu obliczającego współrzędne pionków i pól
  - implementacja algorytmu identyfikującego współrzędne pionków z poszczególnymi polami
  - implementacja zapisu i odczytu plików XML
- Bartosz Gałecki:
  - realizacja interfejsu aplikacji
  - wyświetlanie wyników algorytmu wykrywającego pionki
  - implementacja i wyświetlanie historii ruchów

3. **Metodyka konstruowania systemu**

• metody pracy zespołowej

Podczas prac korzystaliśmy głównie z metodologii kaskadowej.

Prace zostały podzielone na główne części a następnie stopniowo implementowane.

• metody modelowania

Do zamodelowania przypadków użycia wykorzystaliśmy program VisualParadigm

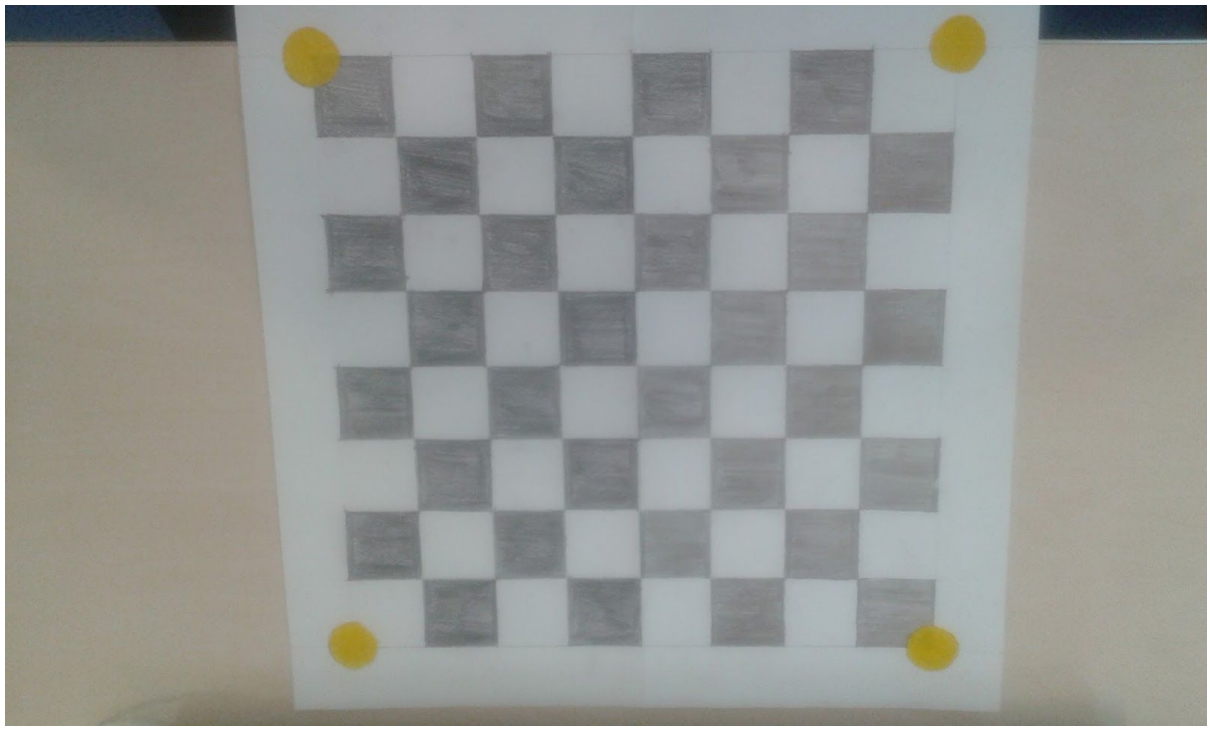
• środki implementacji (narzędzia, środowiska, platformy programowania)

Do wykonania zadania wykorzystaliśmy:

a) Narzędzia

- Kamera internetowa Creative VF0770

- Czarno-biała szachownica z żółtymi markerami



- Pionki czerwone

- Pionki zielone

b) Środowisko programistyczne

- Eclipse
- Version: Mars.2 Release (4.5.2)
- Build id: 20160218-0600

c) Wykorzystane biblioteki

- JavaFX - biblioteka graficzna, używana przez nas do projektu interfejsu
  http://docs.oracle.com/javase/8/javafx/api/toc.htm
  Wersja 2.3.0
- OpenCV - biblioteka służąca do analizy obrazu, wykorzystana głównie do wykrycia w obrazie obiektów o określonych kształtach i kolorach
  http://docs.opencv.org/
  Wersja 3.1.0
- Apache commons-io - biblioteka udostępniająca m.in. operacje odczytu/zapisu plików XML.

Wersja 2.4

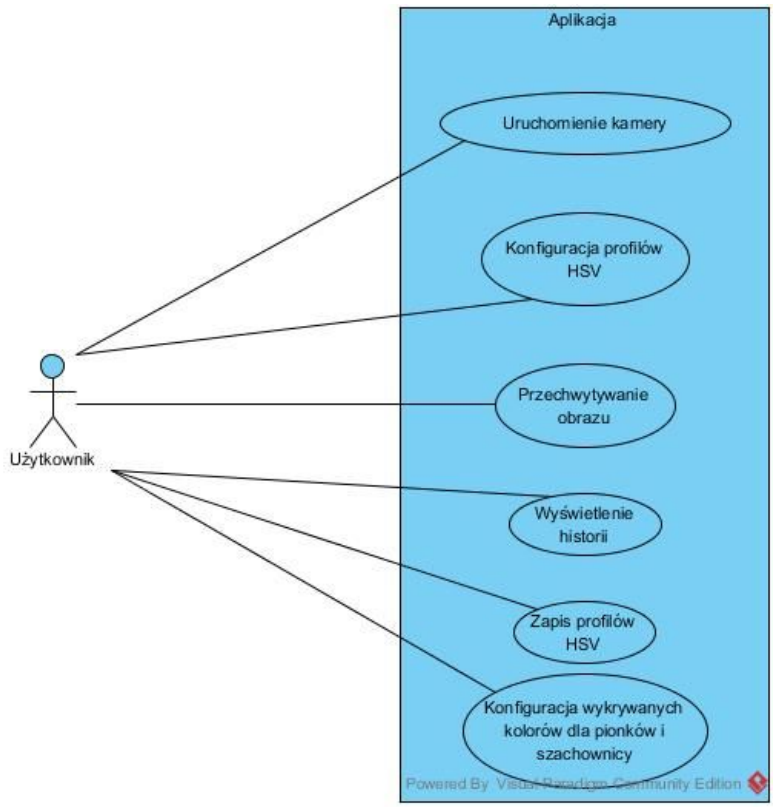
d) Kontrola wersji

- Assembla
  https://www.assembla.com/

e) Inne programy

- Visual Paradigm Community Edition
  Wersja 13
- Debut Video Capture Software
  Wersja 3.0.1

4. **Przypadki użycia**



5. **Implementacja**

• opis podstawowych struktur danych i struktur sterowania (podprogramów)

Struktura programu składa się z:

- pliku z rozszerzeniem .fxml - opisuje wszystkie elementy interfejsu użytkownika aplikacji - ich typ, właściwości, layout oraz ID i nazwy przypisanych do nich akcji
- klasa typu Controller - powiązana z plikiem .fxml, inicjalizuje elementy interfejsu, implementuje związane z nimi akcje i listenery oraz modyfikuje ich parametry w czasie działania programu
- klasa Main - wywołuje metodę launch(), która uruchamia program, inicjalizuje scenę oraz obiekt typu Controller.
- klasy danych - zawierają dane oraz metody niezbędne do funkcjonowania algorytmów. Są to klasy:
  - ObjRecognitionController - oprócz funkcji kontrolera, klasa ta definiuje wiele metod obliczających pozycje pionków i krawędzi, tworzy obiekty innych klas danych i przechowuje listy ich referencji
  - CapturedFrame - klasa zawierająca informacje o pojedynczej przechwyconej klatce obrazu z kamery, takie jak rozmiar obrazu oraz współrzędne wykrytych pól i pionków
  - BoardSquare - klasa opisująca stan każdego z pól szachownicy - jego numer a także informację czy pole jest zajęte, i jeśli tak, to przez którego z graczy.
- klasy konwertujące i przetwarzające - są to klasy zawierające statyczne metody umożliwiające konwersję danych oraz odczyt/zapis do pliku. Są 2 takie klasy:
  - ColorConverter - klasa zawierająca metody pozwalające na konwersję modeli barw ekstraktowanych z pikseli, z modelu RGB na HSV i na odwrót.
  - XmlParser - klasa pozwalająca na odczyt i zapis danych w formie plików XML oraz przetwarzanie ich do postaci użytecznej dla innych klas oraz ich metod. XmlParser wykorzystywany jest przy odczycie pliku konfiguracyjnego programu oraz odczycie/zapisie profili zakresu kolorów HSV.
- zasoby statyczne - umieszczone w podfolderach projektu, wśród nich wyróżnić można:
  - grafiki - wykorzystywane w wyświetlaniu planszy oraz pionków
  - plik konfiguracyjny - pozwala wskazać urządzenie, które będzie użyte w przchwytywaniu obrazu
  - profile HSV - definiują zakresy detekcji koloru dla markerów oraz pionków obu graczy. Są odczytywane automatycznie przy wyborze nazwy profilu z listy w interfejsie programu.

• testy (weryfikacja poprawności działania, ocena efektywności).

Film prezentujący działanie aplikacji:
https://www.youtube.com/watch?v=UE8qF8VibsE&feature=youtu.be

• implementacja (przedstawienie fragmentów ważniejszych algorytmów).

- metoda grabFrame() - przechwytuje klatkę z kamery w postaci macierzy bajtów, wykonuje na niej operacje takie jak rozmycie, dylatacja czy erozja, tworzy trzy obrazy w postaci binarnej maski (osobno dla markerów i pionków każdego z graczy), która będzie podstawą

dla algorytmu wykrywającego obiekty. Na przeanalizowanym obrazie wywołuje metodę findAndDrawMesh(), która znajduje punkty na obrazie (markery i pionki) i rysuje ich pozycje. Metoda zwraca obraz po przekształceniu z macierzy bajtów

```
            Imgproc.blur(frame, blurredImage, new Size(7, 7));

            Imgproc.cvtColor(blurredImage, hsvImage, Imgproc.COLOR_BGR2HSV);

            Scalar minValuesChessboard = new Scalar(this.hueStart1.getValue(), this.saturationStart1.getValue(),
                    this.valueStart1.getValue());
            Scalar maxValuesChessboard = new Scalar(this.hueStop1.getValue(), this.saturationStop1.getValue(),
                    this.valueStop1.getValue());
            Scalar minValuesPlayer1 = new Scalar(this.hueStart2.getValue(), this.saturationStart2.getValue(),
                    this.valueStart2.getValue());
            Scalar maxValuesPlayer1 = new Scalar(this.hueStop2.getValue(), this.saturationStop2.getValue(),
                    this.valueStop2.getValue());
            Scalar minValuesPlayer2 = new Scalar(this.hueStart3.getValue(), this.saturationStart3.getValue(),
                    this.valueStart3.getValue());
            Scalar maxValuesPlayer2 = new Scalar(this.hueStop3.getValue(), this.saturationStop3.getValue(),
                    this.valueStop3.getValue());

            Core.inRange(hsvImage, minValuesChessboard, maxValuesChessboard, mask1);
            Core.inRange(hsvImage, minValuesPlayer1, maxValuesPlayer1, mask2);
            Core.inRange(hsvImage, minValuesPlayer2, maxValuesPlayer2, mask3);

            Mat dilateElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(24, 24));
            Mat erodeElement = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(12, 12));

            Imgproc.erode(mask1, morphOutput1, erodeElement);
            Imgproc.erode(mask1, morphOutput1, erodeElement);
            Imgproc.erode(mask2, morphOutput2, erodeElement);
            Imgproc.erode(mask2, morphOutput2, erodeElement);
            Imgproc.erode(mask3, morphOutput3, erodeElement);
            Imgproc.erode(mask3, morphOutput3, erodeElement);
```

```
                Imgproc.dilate(mask1, morphOutput1, dilateElement);
                Imgproc.dilate(mask1, morphOutput1, dilateElement);
                Imgproc.dilate(mask2, morphOutput2, dilateElement);
                Imgproc.dilate(mask2, morphOutput2, dilateElement);
                Imgproc.dilate(mask3, morphOutput3, dilateElement);
                Imgproc.dilate(mask3, morphOutput3, dilateElement);

                // show the partial output
                this.onFXThread(this.maskImage1.imageProperty(), this.mat2Image(morphOutput1));
                this.onFXThread(this.maskImage2.imageProperty(), this.mat2Image(morphOutput2));
                this.onFXThread(this.maskImage3.imageProperty(), this.mat2Image(morphOutput3));

                // find the tennis ball(s) contours and show them
                frame = this.findAndDrawMesh(morphOutput1, morphOutput2, morphOutput3, frame);

                // convert the Mat object (OpenCV) to Image (JavaFX)
                imageToShow = mat2Image(frame);
            }

        } catch (Exception e) {
            // log the (full) error
            System.err.print("ERROR");
            e.printStackTrace();
        }
    }

    return imageToShow;
}
```

- metoda findAndDrawMesh() - przyjmuje jako argument 3 obrazy-maski oraz oryginalny obraz z kamery. Rysuje krawędzie szachownicy, onzacza kolorowymi konturami markery oraz pionki. Zwraca obraz z wyrysowanymi konturami.

```
    Imgproc.findContours(mask1, mask1Contours, hierarchy1, Imgproc.RETR_CCOMP, Imgproc.CHAIN_APPROX_SIMPLE);
    Imgproc.findContours(mask2, mask2Contours, hierarchy2, Imgproc.RETR_CCOMP, Imgproc.CHAIN_APPROX_SIMPLE);
    Imgproc.findContours(mask3, mask3Contours, hierarchy3, Imgproc.RETR_CCOMP, Imgproc.CHAIN_APPROX_SIMPLE);
    // if any contour exist...
    if (hierarchy1.size().height > 0 && hierarchy1.size().width > 0) {
        for (int idx = 0; idx >= 0; idx = (int) hierarchy1.get(0, idx)[0]) {
            Imgproc.drawContours(frame, mask1Contours, idx, new Scalar(250, 0, 0));

            if (!currentlyProcessedFrame.player1Points.isEmpty()) {
                Imgproc.drawContours(frame, lmopPlayer1, 0, new Scalar(0, 250, 0), 5);
            }
            if (!currentlyProcessedFrame.markerPoints.isEmpty()) {
                Imgproc.drawContours(frame, lmopMarkers, 0, new Scalar(0, 0, 250), 5);
            }
            if (!currentlyProcessedFrame.allPoints.isEmpty()) {
                Imgproc.drawContours(frame, lmopAll, 0, new Scalar(250, 250, 250), 5);
            }
        }
    }
    if (hierarchy2.size().height > 0 && hierarchy2.size().width > 0) {
        for (int idx = 0; idx >= 0; idx = (int) hierarchy2.get(0, idx)[0]) {
            Imgproc.drawContours(frame, mask2Contours, idx, new Scalar(0, 0, 250));
        }
    }
    if (hierarchy3.size().height > 0 && hierarchy3.size().width > 0) {
        for (int idx = 0; idx >= 0; idx = (int) hierarchy3.get(0, idx)[0]) {
            Imgproc.drawContours(frame, mask3Contours, idx, new Scalar(0, 250, 0));
        }
    }
    return frame;
}
```

- metoda getPointsAveragePosition() - przyjmuje jako argument listę punktów oznaczających wykryte kontury i dla każdego zamkniętego kształtu wyznacza średnią, aby zamienić go w pojedynczy punkt. Zwraca współrzędne punktu.

```
private double[] getPointsAveragePosition(List<Point> points) {
    int xSum = 0;
    int ySum = 0;
    double[] result = new double[2];
    for (int i = 0; i < points.size(); i++) {
        xSum += points.get(i).x;
        ySum += points.get(i).y;
    }
    result[0] = xSum / points.size();
    result[1] = ySum / points.size();
    return result;
}
```

- metoda generateBorderPoints() - generuje zewnętrzne krawędzie pól na podstawie wykrytych markerów. Dzieli odcinek między markerami na określoną liczbę punktów, oblicza odchylenie osi x i y w pionie i poziomie między markerami i na tej podstawie wyznacza zewnętrzne krawędzie.

```
xDiff1 = (currentlyProcessedFrame.markerPoints.get(1).x - currentlyProcessedFrame.markerPoints.get(0).x) / (BOARD_X - 1);
yDiff1 = (currentlyProcessedFrame.markerPoints.get(1).y - currentlyProcessedFrame.markerPoints.get(0).y) / (BOARD_Y - 1);
xDiff2 = (currentlyProcessedFrame.markerPoints.get(2).x - currentlyProcessedFrame.markerPoints.get(1).x) / (BOARD_X - 1);
yDiff2 = (currentlyProcessedFrame.markerPoints.get(2).y - currentlyProcessedFrame.markerPoints.get(1).y) / (BOARD_Y - 1);
xDiff3 = (currentlyProcessedFrame.markerPoints.get(3).x - currentlyProcessedFrame.markerPoints.get(2).x) / (BOARD_X - 1);
yDiff3 = (currentlyProcessedFrame.markerPoints.get(3).y - currentlyProcessedFrame.markerPoints.get(2).y) / (BOARD_Y - 1);
xDiff4 = (currentlyProcessedFrame.markerPoints.get(0).x - currentlyProcessedFrame.markerPoints.get(3).x) / (BOARD_X - 1);
yDiff4 = (currentlyProcessedFrame.markerPoints.get(0).y - currentlyProcessedFrame.markerPoints.get(3).y) / (BOARD_Y - 1);

currentlyProcessedFrame.borderPoints = borders;
this.addBordersToAll();
for (int i = 1; i < BOARD_Y - 1; i++) {
    this.generatePointsHorizontal(borders.get(32 - i), borders.get(8 + i), i);
}
// debug-only
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
        currentlyProcessedFrame.allEdges[i][j] = currentlyProcessedFrame.allPoints.get(i * 9 + j);
    }
}

this.toPointList(currentlyProcessedFrame.allEdges);
```

- metoda generatePointsHorizontal() - na podstawie wygenerowanych krawędzi zewnętrznych oblicza współrzędne pozostałych pól szachownicy. Metoda addBordersToAll() dodaje zewnętrzne krawędzie do listy wszystkich punktów.

```
public void generatePointsHorizontal(Point start, Point end, int rowIndex) {
    double xDiff = 0;
    double yDiff = 0;

    xDiff = (end.x - start.x) / (BOARD_X - 1);
    yDiff = (end.y - start.y) / (BOARD_X - 1);
    for (int i = 1; i < BOARD_X - 1; i++) {
        currentlyProcessedFrame.allPoints.add(rowIndex * BOARD_X + i, new Point(start.x + i * xDiff, start.y + i * yDiff));
    }
}

public void addBordersToAll() {
    for (int i = 0; i < 9; i++) {
        currentlyProcessedFrame.allPoints.add(currentlyProcessedFrame.borderPoints.get(i));
    }
    for (int i = 1; i < 8; i++) {
        currentlyProcessedFrame.allPoints.add(currentlyProcessedFrame.borderPoints.get(32 - i));
        currentlyProcessedFrame.allPoints.add(currentlyProcessedFrame.borderPoints.get(8 + i));
    }
    for (int i = 0; i < 9; i++) {
        currentlyProcessedFrame.allPoints.add(currentlyProcessedFrame.borderPoints.get(24 - i));
    }
}
```

- metoda findAllCheckers() - wywołuje metodę findCheckerPos(), której wynik determinuje parametry wartość obiektów pól szachownicy tworzonych w tej metodzie (czy są zajęte i jeśli tak to przez którego gracza)

```
private void findAllCheckers(){
    currentlyProcessedFrame.squares = new BoardSquare[BOARD_X - 1][BOARD_Y - 1];
    for(int x = 0; x < currentlyProcessedFrame.squares.length; x++){
        for(int y = 0; y < currentlyProcessedFrame.squares[0].length; y++){
            currentlyProcessedFrame.squares[x][y] = new BoardSquare(false, 0);
        }
    }
    int[] XYPos = new int[2];
    for(int p1 = 0; p1 < currentlyProcessedFrame.player1Points.size(); p1++){
        XYPos = this.findCheckerPos(currentlyProcessedFrame.player1Points.get(p1));
        currentlyProcessedFrame.squares[XYPos[0]][XYPos[1]] = new BoardSquare(true, 1);
    }
    for(int p2 = 0; p2 < currentlyProcessedFrame.player2Points.size(); p2++){
        XYPos = this.findCheckerPos(currentlyProcessedFrame.player2Points.get(p2));
        currentlyProcessedFrame.squares[XYPos[0]][XYPos[1]] = new BoardSquare(true, 2);
    }
}
```

- metoda findCheckerPos() - identyfikuje pozycję pionków poprzez porównanie ich współrzędnych ze współrzędnymi pól szachownicy

```
private int[] findCheckerPos(Point checker){
    int[] result = new int[2];
    for(int x = 0; x < currentlyProcessedFrame.allEdges[0].length - 1; x++){//find column number
        if(currentlyProcessedFrame.allEdges[0][x].x < checker.x && currentlyProcessedFrame.allEdges[0][x+1].x > checker.x){
            result[0] = x;
        }
    }
    for(int y = 0; y < currentlyProcessedFrame.allEdges.length - 1; y++){//find row number
        if(currentlyProcessedFrame.allEdges[y][0].y < checker.y && currentlyProcessedFrame.allEdges[y+1][0].y > checker.y){
            result[1] = y;
        }
    }
    return result;
}
```

- metoda putCheckers() - na podstawie wykrytej pozycji pionków, umieszcza je na szachownicy

```
private void putCheckers(CapturedFrame frame, int mode){
    //System.out.println("p1--- " + frame.player1Points.size());
    //System.out.println("p2--- " + frame.player2Points.size());
    for(int x = 0; x < frame.squares.length; x++){
        for(int y = 0; y < frame.squares[0].length; y++){
            if(frame.squares[y][x].player == 1){
                if(mode == 0) this.boardImgRefs[x][y].setImage(checkerWhite);
                if(mode == 1) this.historyBoardImgRefs[x][y].setImage(checkerWhite);
            }
            else if(frame.squares[y][x].player == 2){
                if(mode == 0) this.boardImgRefs[x][y].setImage(checkerBlack);
                if(mode == 1) this.historyBoardImgRefs[x][y].setImage(checkerBlack);
            }
        }
    }
}
```

6. **Użytkowanie**

Aplikacja dzieli się na 3 podstrony:

- Board View - podgląd szachownicy na której umieszczane są pionki
- Camera View - podgląd rejestrowanego obrazu

- History - historia wykonanych przez graczy ruchów

Dostępne przyciski:

- Start Camera/Stop Camera - rozpoczyna i kończy pracę kamery
- Capture - przechwytuje klatkę z przechwytywanego obrazu kamery i rozpoczyna proces analizy obrazu i umieszczenia pionków na szachownicy
- Exit - kończy prace programu
- Save HSV Profile - Zapisuje aktualnie skonfigurowany profil HSV do późniejszego użycia

Dostępne suwaki (osobne dla pionków i znaczników):

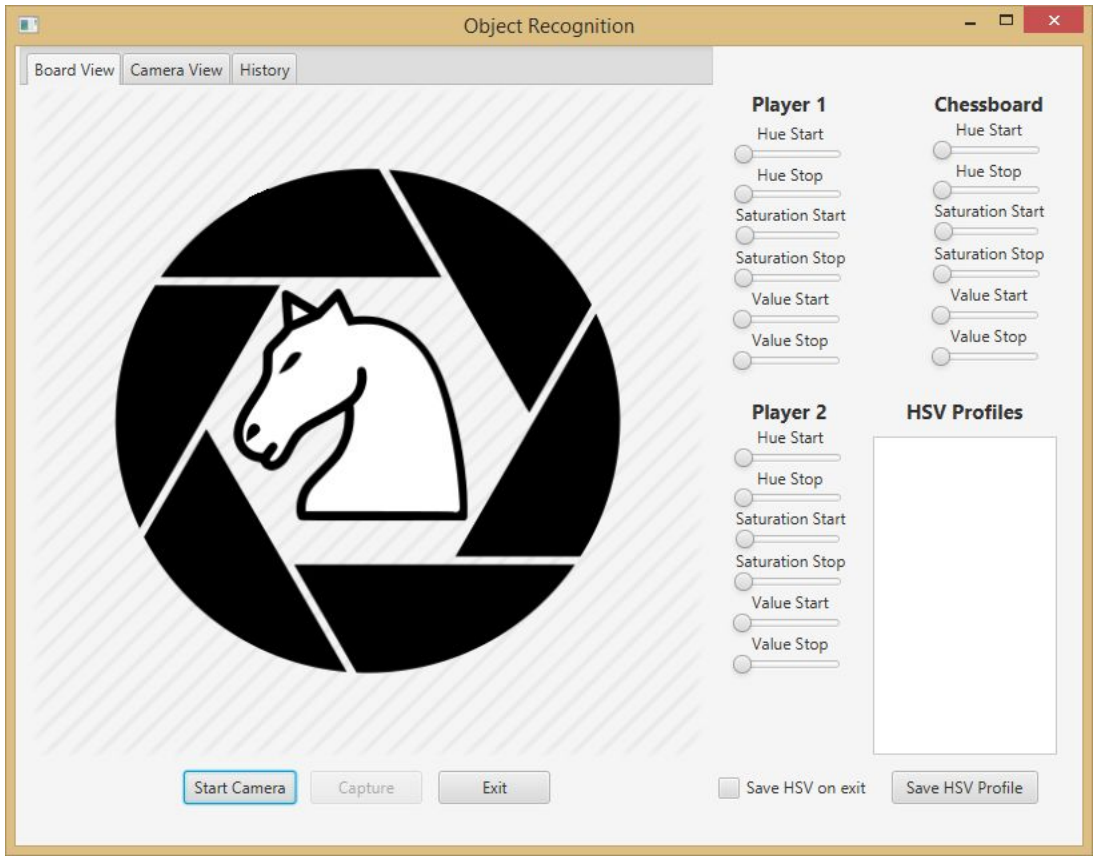
- Hue Start&Stop - określa zakres wykrywanego odcienia
- Saturation Start&Stop - określa zakres wykrywanego poziomu nasycenia
- Value Start&Stop - określa zakres wykrywanej mocy/jasności

Dostępne listy wyborów:

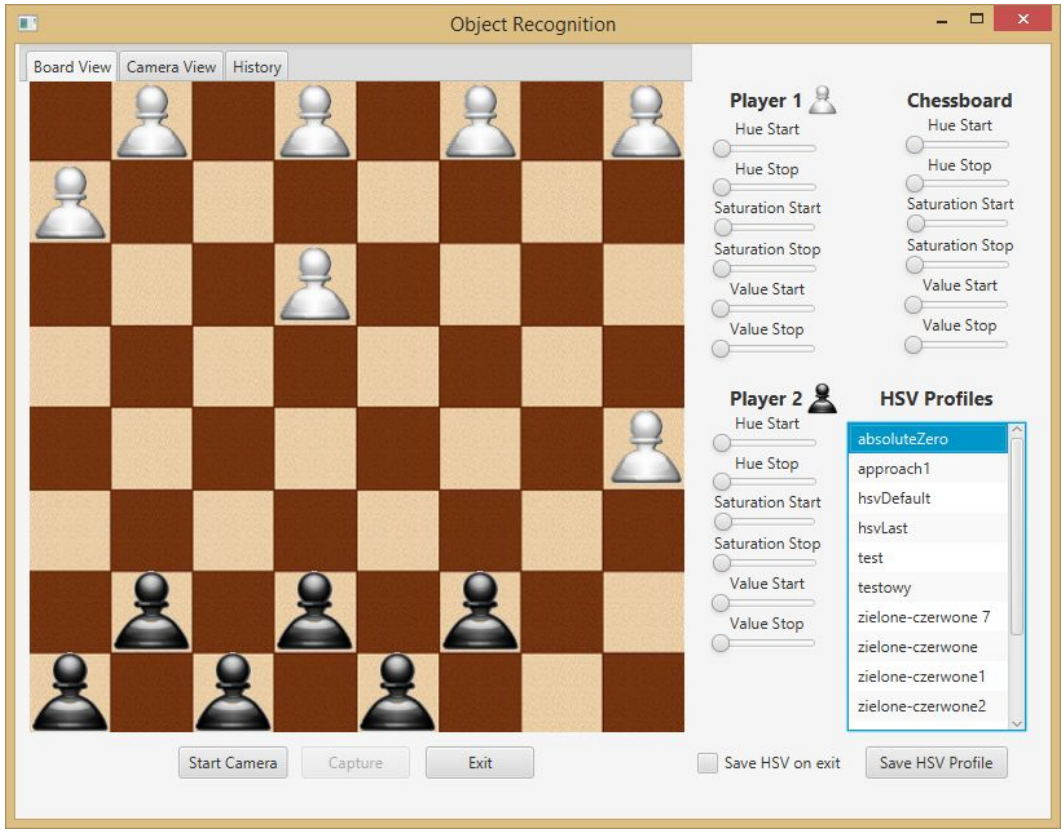
- HSV Profiles - Zdefiniowane wcześniej profile HSV
- Moves - Wykonane podczas pracy programu ruchy graczy

Zrzuty ekranowe aplikacji:

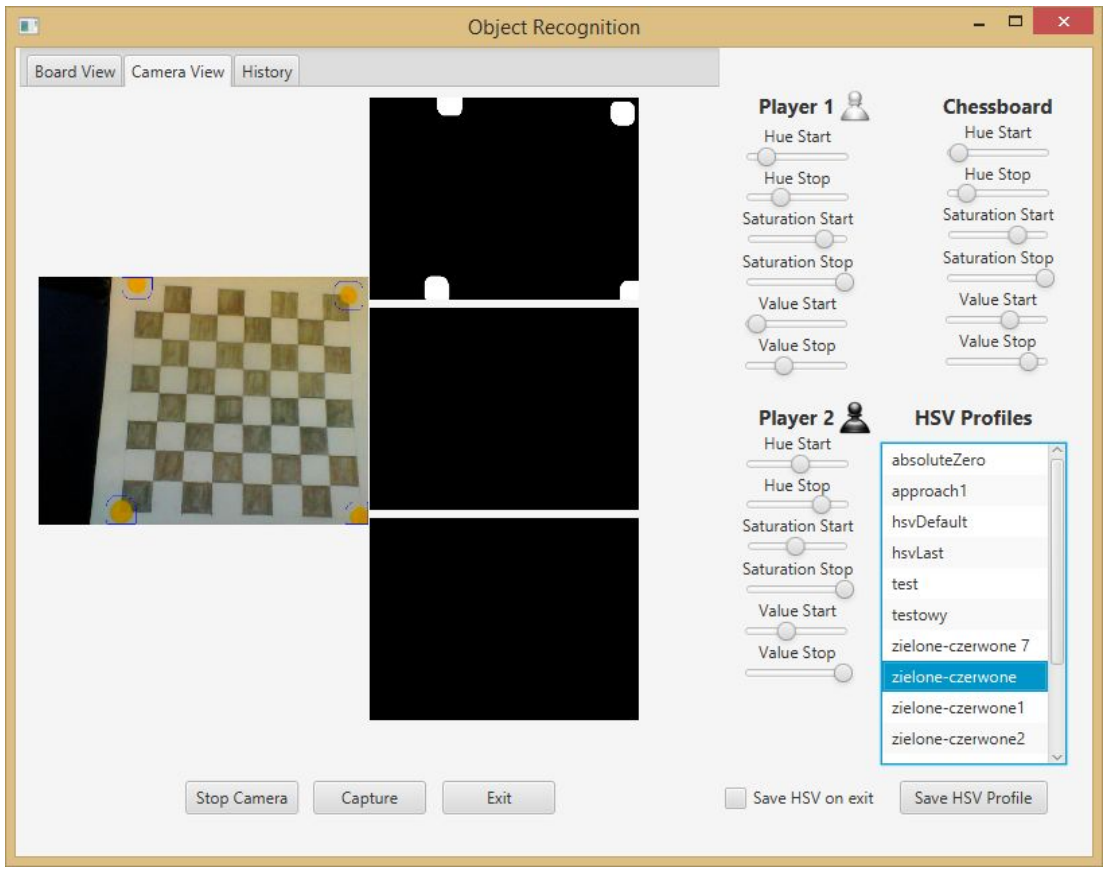
- Ekran startowy(przed włączeniem kamery)



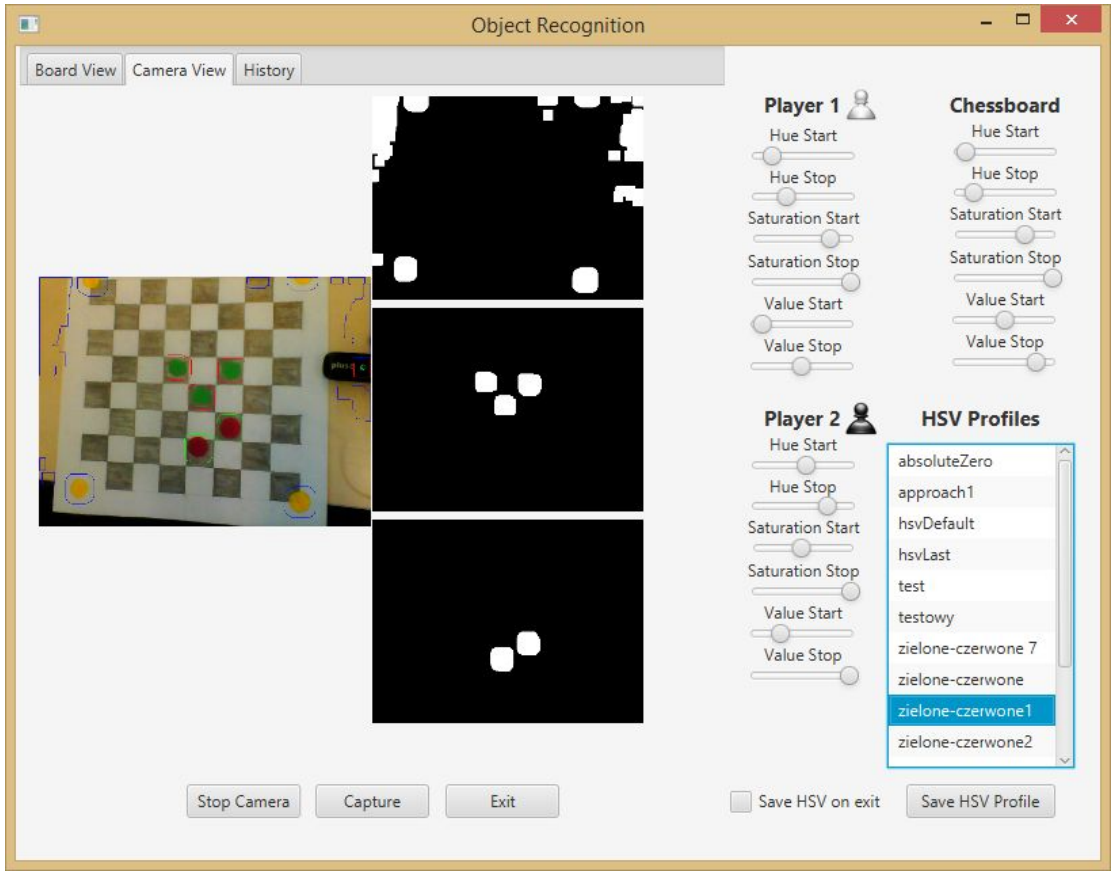
- Widok szachownicy z zaznaczonymi pionkami obu graczy



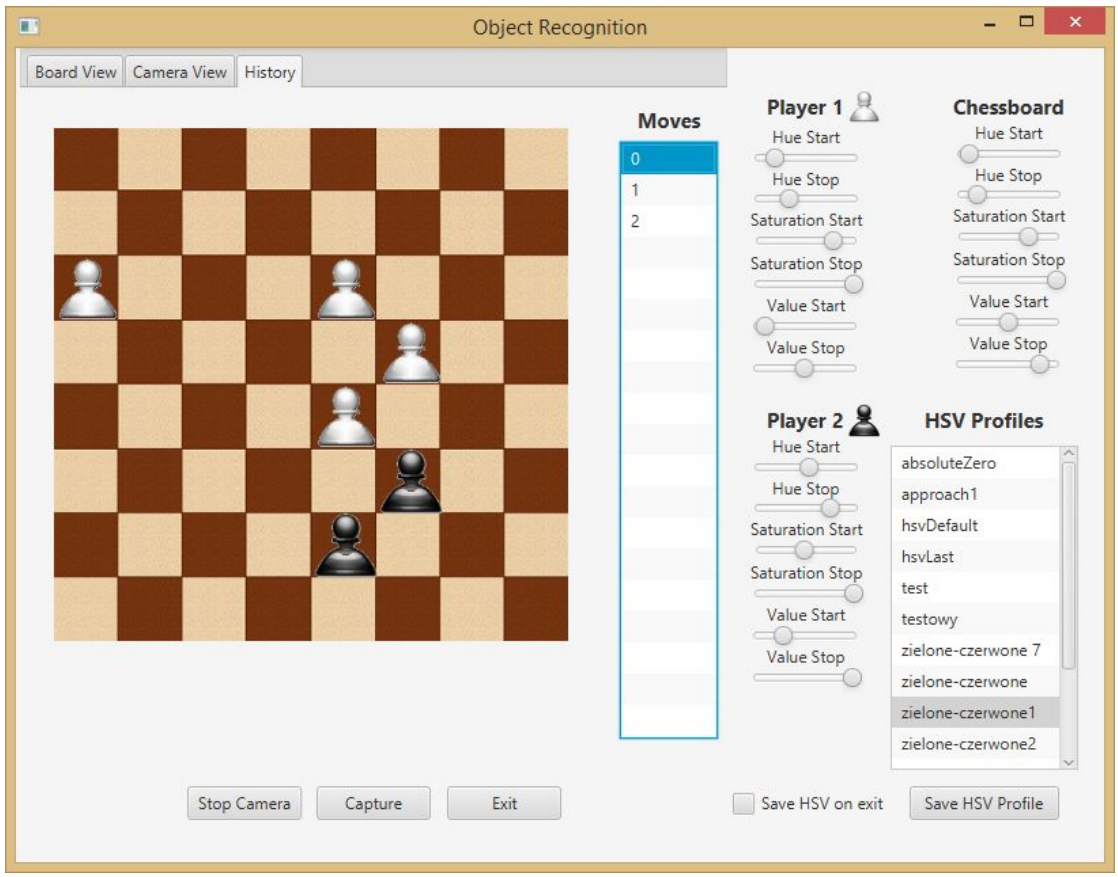
- Wykrywanie markerów



- Wykrywanie pionków i markerów



- Historia ruchów



7. **Podsumowanie**

• opis celów zrealizowanych i niezrealizowanych

- Projekt został wykonany w całości, według założeń wstępnych

• wskazanie możliwych kierunków rozbudowy systemu

- Gdyby aplikacja była rozwijana dalej, kolejnymi krokami w jej usprawnieniu byłyby :
  - usprawnienie algorytmu wykrywania konturów, aby wyeliminować niepoprawny odczyt w sytuacji gdy pionki są za blisko siebie (zlewanie się konturów)
  - automatyzacja przechwytywania obrazu (przechwytywanie czasowe lub algorytm wywołujący przechwycenie na podstawie zmiany pozycji pionków)
  - implementacja logiki warcabów - sprawdzanie, czy wykonany ruch jest poprawny
  - możliwość dobierania kolorów przez kliknięcie w dowolnym miejscu obrazu z kamery

• wnioski i obserwacje dot. pracy zespołowej.

- Prace przebiegały sprawnie przez cały semestr
- Regularne prezentacje sprawiły, że praca przez cały semestr wykonywana była płynnie i zmiany wprowadzane były regularnie

8. **Literatura**

1 - Oficjalna strona i dokumentacja OpenCV - http://opencv.org/

2 - Wikipedia, informacje odnośnie zasad gry w wracaby - https://en.wikipedia.org/wiki/Draughts