Dokumentacja Techniczna

CHECKERS DETECTOR

Wykrywanie pionków oraz obrazowanie stanu gry w warcaby

Paulina Kurpisz, Norbert Nowak

Wprowadzenie

Poniższa dokumentacja zawiera opis oraz założenia projektu “Checkers detector”, czyli programu obrazującego stan gry w warcaby. Projekt składa się z modułu który rozpoznaje obraz przechwycony z kamery a następnie wykrywa planszę z pionkami. Następnie wyświetlany jest aktualny stan planszy.

Opis programu

Celem programu jest przechwycenie i wyświetlenie przygotowanej planszy (plansze przed rozpoczęciem gry należy odpowiednio skalibrować) oraz pionków na niej naniesionych w odpowiednich odstępach czasu. Pierwszym krokiem do poprawnego działania programu jest skonfigurowanie planszy przy pomocy umieszczenia żółtych pionków w jej rogach, następnie umiejscowienie zielonych i niebieskich pionków na planszy.

Zakres

Dokument zawiera szczegółowe zasady działania programu. Zawarte w nim są również informacje na temat implementacji oraz komunikacji między modułami. Moduły występujące w programie:

- przetwarzanie obrazu
- warstwa wizualna
- obsługa wirtualnej planszy

Pojęcia

Znajdują się tutaj opisane pojęcia znajdujące się w poniższej dokumentacji:

- Macierz - tabela zawierająca pola na których znajdują się pionki
- Macierz projekcji - wygenerowana macierz w celu poprawnego przechwytywania planszy
- BGR - tablica kolorów tworzona na podstawie palety RGB
- Wirtualna plansza - tworzona przez program
- UI - *ang. user interface* - interfejs graficzny

Uzasadnienie wyboru tematu

Zainteresowało nas rozpoznawanie obrazu przechwyconego z kamery internetowej oraz możliwości od górnego sprawdzenia przestrzegania narzuconych zasad. (Tutaj np. zasady przestrzegane w grze w warcaby).

Podział prac

Bazując na umiejętnościach poszczególnych członków rozłożono w poniższy sposób prace nad projektem:

Temat	**Wykonawca**
Zapoznanie się z dokumentacją OpenCV	Wszyscy
Założenie projektu oraz repozytorium	Paulina
Przetwarzanie obrazu do macierzy	Norbert
Generowanie planszy	Paulina
Stworzenie macierzy projekcji	Norbert
Połączenie modułów	Wszyscy
Poprawki	Wszyscy
Dokumentacja	Wszyscy

Funkcjonalności oferowane przez aplikację

W projekcie zostały zrealizowane:
- Pobranie zdjęć w określonym odstępie czasu
Co pewien określony czas zostają pobrane zdjęcia z kamery internetowej.

- Weryfikacja jakości/poprawności obrazu
Weryfikacja czy jakość obrazu jest wystarczająca do poprawnego działania aplikacji.

- Rozpoznanie położenia pionów na planszy
Piony rozpoznawane na planszy są rozpoznawane po ich kolorze oraz kształcie. Aby piony zostały wykryte, muszą być koloru zielonego i niebieskiego.

- Rozpoznanie położenia planszy
Plansza jest rozpoznawana za pomocą ustawienia w rogach markerów koloru żółtego.

- Odnalezienie zmian na planszy
Gdy gracz zmieni pozycje pionka zmiana ta jest znaleziona oraz wyświetlona.

- Aktualizacja wyświetlanego obrazu
Wyświetlany obraz jest aktualizowany na bieżąco w określonym odstępie czasu.

Wymagania pozafunkcjonalne

Do poprawnego działania aplikacji niezbędne jest spełnienie tych wymagań.

Wymagania produktowe

Poniżej zostały przedstawione wymagania produktowe:

- Użytkownik musi posiadać kamerę internetową
- Aplikacja oraz jej interfejs powinien być dostępny w języku angielskim
- Aplikacja nie powinna wyświetlać żadnych błędów
- Działanie aplikacji powinno być płynne
- Czas uruchamiania aplikacji powinien być nie większy niż 20 sekund
- Użytkownik musi posiadać system Windows 10 lub MacOS
- Użytkownik musi mieć JDK zainstalowaną na komputerze
- Aplikacja powinna być niezawodna
- Minimum 512 MB wolnej pamięci operacyjnej
- Szachownica o dowolnym wymiarze
- Pionki o dwóch kolorach

Wymagania organizacyjne

Poniżej zostały przedstawione wymagania organizacyjne:

- Aplikacja wykonana jest za pomocą JavaFX
- Do przechwytywanie obrazu użyta została biblioteka OpenCV
- Aplikację można wdrożyć w każdej chwili
- Jako repozytorium został użyty system kontroli wersji Github
- Aplikacja napisała została za pomocą środowiska intellij

Wymagania zewnętrzne

Poniżej zostały przedstawione wymagania zewnętrzne:

- Użytkowanie aplikacji powinno być łatwe i intuicyjne aby każdy użytkownik mógł w pełni z niej korzystać
- Aplikacja powinna mieć poprawną obsługę błędów, czytelną dla użytkownika

Wymagania funkcjonalne

W projekcie skupiliśmy się na zrealizowaniu dwóch poniższych celów:

- Rozpoznawanie obrazu z planszy na podstawie przechwyconego obrazu
- Wizualizacja przebiegu gry na komputerze
- Odzwierciedlenie kolorów pionków na wirtualnej planszy
- Odpowiednie wykadrowanie oraz wyświetlenie przechwyconej planszy

Użytkownicy

W naszej aplikacji jest tylko jeden typ użytkownika który ma dostęp do wszystkich jego funkcji bez logowania się. Dużą rolę odgrywa świat zewnętrzny oraz poprawne naświetlenie stanowiska gry. Użytkownik powinien tylko włączyć oraz skalibrować program a w dalszych częściach zająć się grą w warcaby.

Wybrane technologie oraz środowisko

JavaFX

Java FX - jest domyślną biblioteką definiowania graficznego interfejsu użytkownika w języku Java począwszy od wersji 8. Zastąpiła ona starsze biblioteki takie jak Swing i AWT. Charakteryzuje się przede wszystkim dobrą separacją warstw, wsparciem dla architektury MVC (Model View Controller) i udostępnia do tego użyteczne narzędzie o nazwie Scene Builder.

Java

Java – obiektowy język programowania stworzony przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems. Java jest językiem tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną.

Biblioteka OpenCV

OpenCV – biblioteka funkcji wykorzystywanych podczas obróbki obrazu, oparta na otwartym kodzie i zapoczątkowana przez Intela. Biblioteka ta jest wieloplatformowa, można z niej korzystać w Mac OS X, Windows jak i Linux. Autorzy jej skupiają się na przetwarzaniu obrazu w czasie rzeczywistym.

IntelliJ IDEA

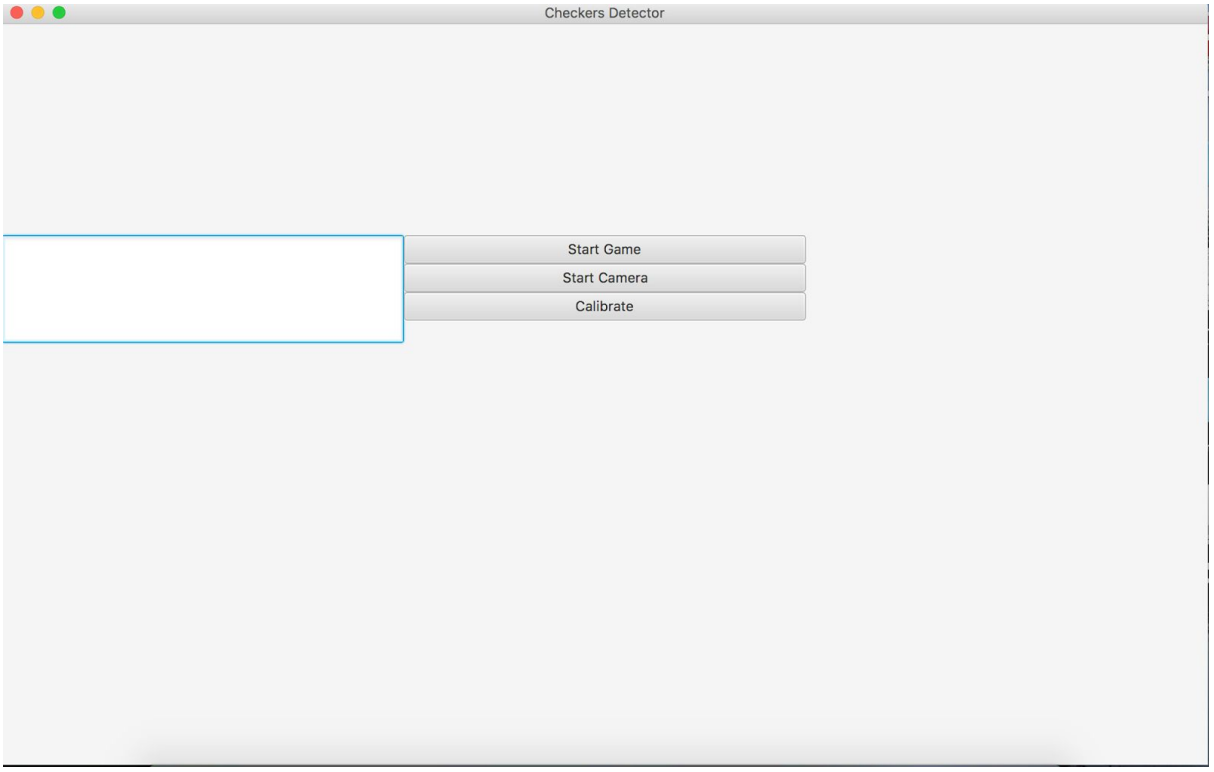
IntelliJ IDEA – komercyjne zintegrowane środowisko programistyczne (IDE) dla Javy firmy JetBrains.
Wersja IDEA (12.0) zawiera szybszy kompilator, wsparcie Java 8, narzędzie projektowania interfejsów dla Androida, wsparcie frameworka Play 2.0 dla Javy i Scala.

Scene Builder

Podstawowym narzędziem do tworzenia aplikacji w Javie FX jest Scene Builder. Jest odpowiedzialny za wygenerowanie pliku FXM i część wizualną aplikacji.

Instrukcja obsługi aplikacji

W tym dziale zostanie przedstawiony szczegółowy poradnik jak korzystać z aplikacji. Po włączeniu aplikacji ukazuje się następujące okno:



Dostępne są trzy buttony:

- Start Game
- Start Camera
- Calibrate

Przyciski należy użyć w odpowiedniej kolejności:

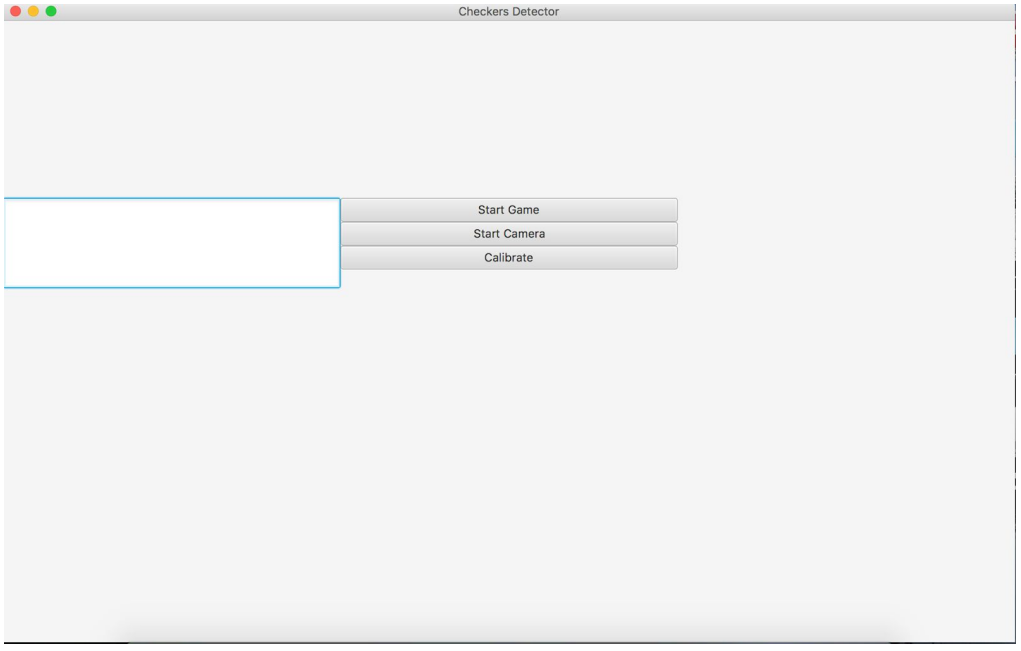
- Najpierw należy kliknąć przycisk “**Start Camera**” w celu włączeniu kamery internetowej
- Następnie należy umieścić żółte pionki odpowiadające za markery w rogach planszy
- Kolejnie należy kliknąć przycisk “**Calibrate**” aby zostało sprawdzone czy pozycja planszy jest odpowiednia, jeśli nie pozycja planszy zostanie zmieniona
- Gdy wszystko zostało ustawione poprawnie należy nacisnąć przycisk “**Start Game**”

Opis poszczególnych funkcji programu

- Wykrywanie planszy na przechwyconym obrazie odbywa się przy pomocy ustawionych w rogach żółtych pionków. Zostają one wyszukane oraz na ich podstawie ustalamy położenie planszy.
- Wykrywanie pionków polega na wyszukaniu ich na planszy po ich kolorze przy pomocy biblioteki OpenCV.
- Odwzorowanie stanu planszy na macierzy odbywa się poprzez określenie dla każdego znalezionego pionka jego koloru oraz pozycji na planszy. Dla różnych pionków przypisane są różne wartości ustalone.

Interfejs aplikacji

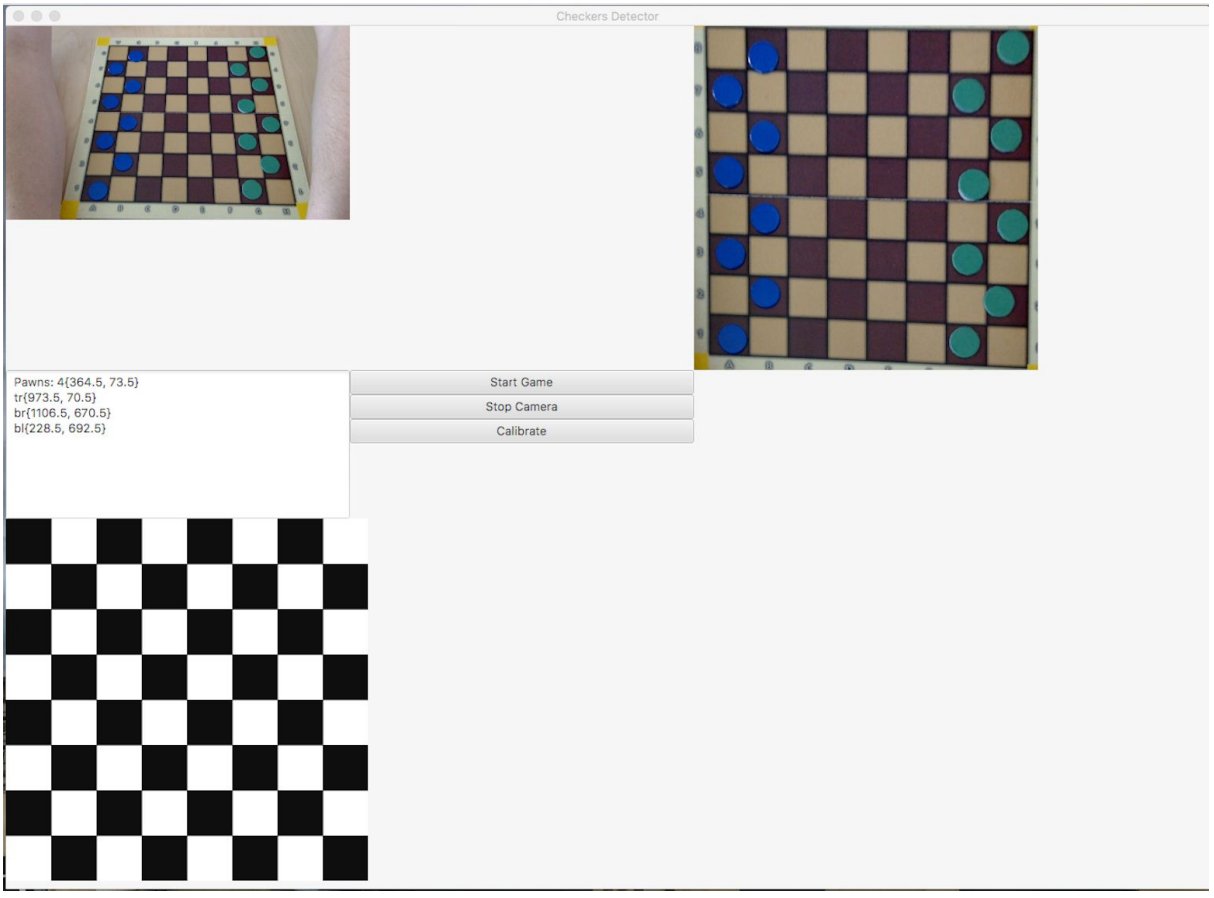
Główny panel po włączeniu aplikacji:



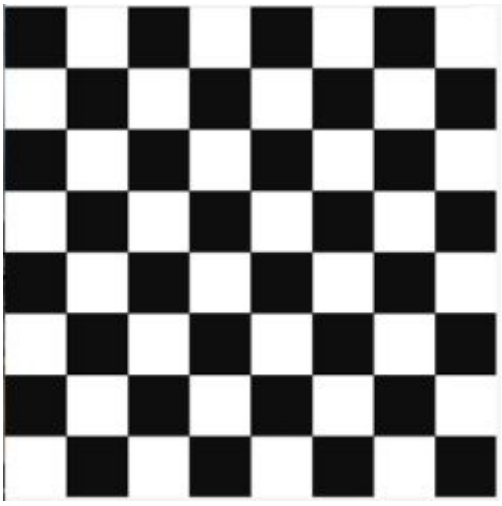
Składa się on z miejsca na komunikaty odnośnie pozycji pionów oraz trzech rodzajów przycisków:

- Start Game - odpowiadający za rozpoczęcie przechwytywania i odwzorowywania stanu gry
- Start Camera - odpowiadający za włączenie kamery internetowej
- Calibrate - odpowiadający za kalibracje planszy

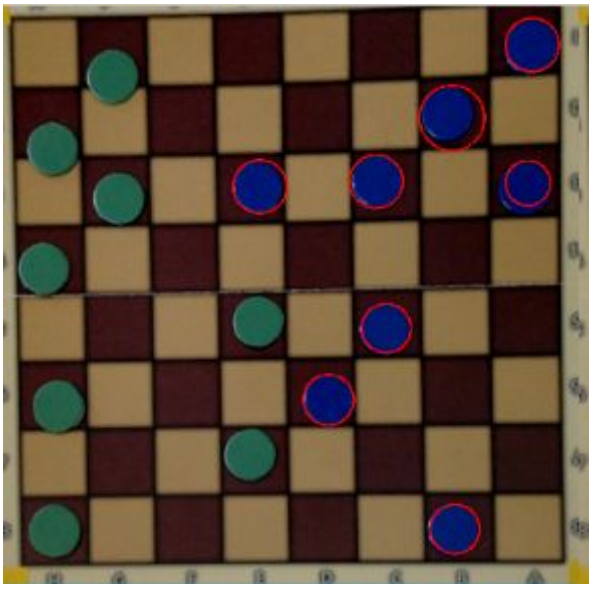
Po włączeniu kamery oraz kalibracji planszy mamy taki oto widok:



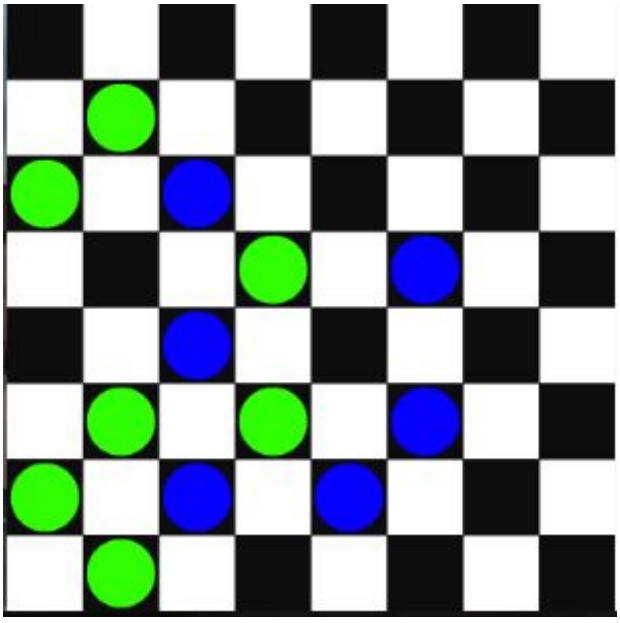
Wygenerowana plansza:



Gdy zostaną rozpoznane pionki na planszy zostaną, one zaznaczone czerwonymi okręgami:



Rozpoznane pionki są wyświetlane na wygenerowanej planszy:



W oknie dialogowym zostanie wyświetlona ilość rozpoznanych pionków danego koloru na planszy:

Pozycja żółtych markerów oraz lokalizacja rozpoznanych pionków:

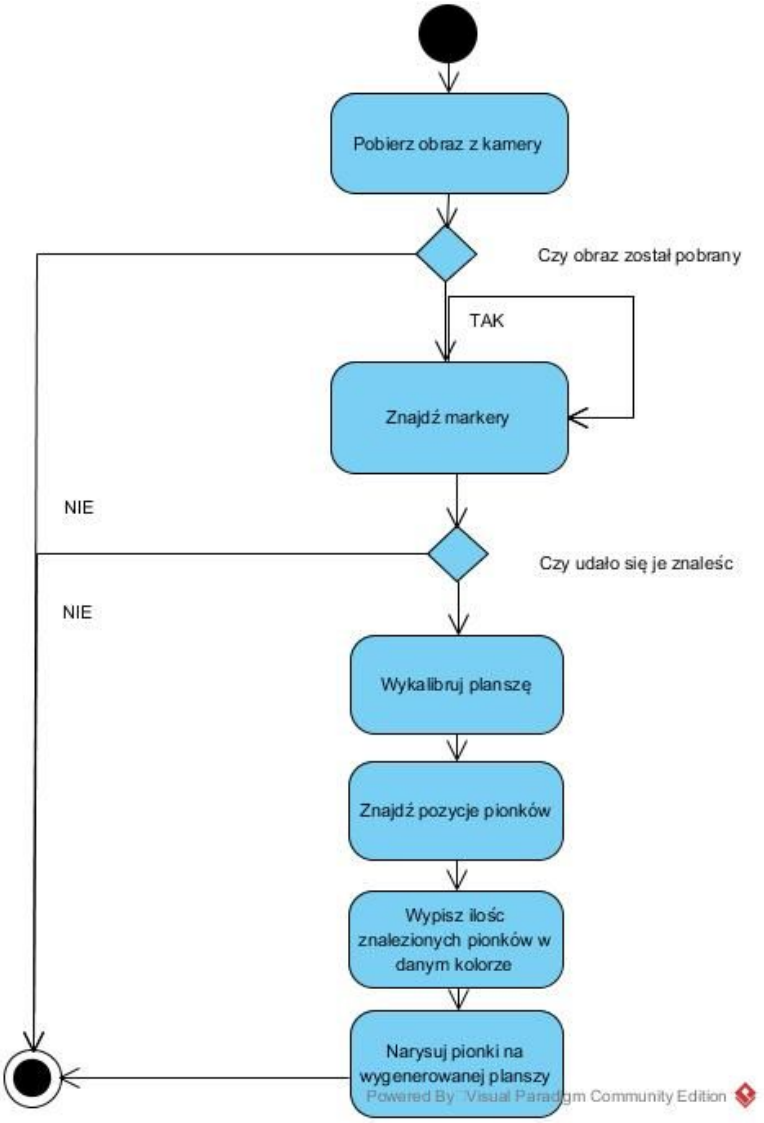
Logika działania algorytmu wykrywania pionków

W celu wykrycia oraz naniesienia pionków na plansze wykrywane są poszczególne kroki:

1. Na podstawie wygenerowanej macierzy tworzona jest pusta tabela obrazująca początkowy stan gry
2. Na wyżej wymienioną macierz nakładany jest filtr odpowiedniego koloru
3. Odfiltrowany obraz jest rozmywany w celu zmniejszenia zakłóceń/szumów
4. Na przetworzonym obrazie wyszukiwane są okręgi
5. Odszukane okręgi konwertowane są do tabeli
6. Do pustej tablicy opisującej stan gry wpisywane wpisywane są zajęte pola na podstawie pozycji oraz koloru
7. Na wirtualną planszę gry nakładane są wirtualne pionki.
8. Plansza jest czyszczona

Model aplikacji

Diagram aktywności przedstawiający ogólny schemat działania programu oraz zaimplementowaną logikę prezentuje się następująco:



Implementacja

Poniższy dział zawiera implementację w Javie z wykorzystaniem biblioteki OpenCV. Podrozdziały prezentują poszczególne moduły/klasy.

Kolor

Specjalnie stworzona klasa przechowująca zakresy kolorów wykorzystywane przez inne funkcje. Celem tej klasy jest ujednolicenie oraz uproszczenie przekazywanie wartości kolorów w zakresie programu.

```
public class Color {

    public Color(Integer B1, Integer G1, Integer R1, Integer B2, Integer G2, Integer R2) {...}

    public Color(Integer color) {...}

    public Color(Integer B1, Integer G1, Integer R1) {...}

    private List<Integer> minValues;
    private List<Integer> maxValues;
```

- List<Integer> … - lista przechowująca minimalne lub maksymalne wartości z tabeli kolorów BGR
- Color(...) - konstruktory przypisujące wartości do list minValues, maxValues
-

Kwadrat

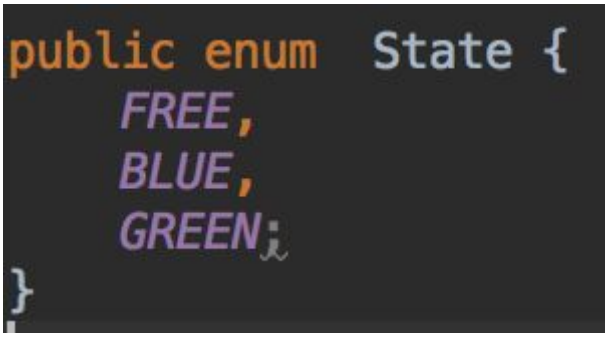
Podobnie do powyższej klasy jej funkcja jest typowo pomocnicza. Klasa ta wykorzystywana jest podczas generowania macierzy projekcji.

```
public class Square {
    public Point topLeft;
    public Point topRight;
    public Point bottomLeft;
    public Point bottomRight;

    public Square(Point topLeft, Point topRight, Point bottomRight, Point bottomLeft) {...}

    public List<Point> getPoints() { return Arrays.asList(topLeft, topRight, bottomRight, bottomLeft); }
}
```

- Point … - punkty reprezentujące rogi kwadratu
- List<Point> getPoints - zwraca listę wszystkich punktów kwadratu

Stan

Enum reprezentujący stan gry oraz kolory pionków.



- FREE - wolne pole
- BLUE - pola gracza koloru niebieskiego
- GREEN - pola gracza koloru zielonego

Klasa zarządzająca

Funkcja inicjująca podstawowe komponenty takie jak widok oraz kontroler zarządzający widokami. Tworzy również instancję klasy obsługującej przetwarzanie obrazu. Wybierana jest również tutaj biblioteka wykorzystywana przez resztę programu.

```
public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "sample.fxml"));
        VBox rootElement = loader.load();
        rootElement.setStyle("-fx-background-color: whitesmoke;");
        Scene scene = new Scene(rootElement);
        primaryStage.setTitle("Checkers Detector");
        primaryStage.setScene(scene);
        MainController controller = loader.getController();
        controller.imageProcessing = new ImageProcessing(controller);
        primaryStage.show();
    }

    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        launch(args);
    }
}
```

Główny kontroler

Klasa zarządzająca logiką działania programu oraz wywołująca inne funkcję w celu uzyskania odpowiednich wyników działania programu.

```
public class MainController {
    @FXML
    public Button startCameraButton;
    @FXML
    public Button nextMoveButton;
    @FXML
    public Button calibrateButton;

    @FXML
    public ImageView cameraView;
    @FXML
    public ImageView captureView;
    @FXML
    public ImageView calibrateView;
    @FXML
    public ImageView checkersBoardView;
    @FXML
    public TextArea infoTextArea;

    public ImageProcessing imageProcessing;
```

Przypisanie zmiennych występujących w UI.

- @FXML jest to adnotacja oznaczająca że dana zmienna występuje w konfiguracji interfejsu graficznego
- ImageProcessing instancja klasy zajmującej się przetwarzaniem obrazów

```
Color firstPlayer = new Color( B1: 100,  G1: 150,  R1: 0,  B2: 140,  G2: 255,  R2: 255); //BLUE;
Color borderEdges = new Color( B1: 20,  G1: 100,  R1: 100,  B2: 30,  G2: 255,  R2: 255); //YELLOW
Color secondPlayer = new Color( B1: 50,  G1: 32,  R1: 16,  B2: 80,  G2: 245,  R2: 245); //GREEEN;
Color currentColor;
int gameState = 0;
```

Inicjalizacja kolorów obu graczy, oraz markerów ułożonych w celu wykrycia przestrzeni gry.

```
49      @FXML
50      public void startCamera() {
51          Board board = new Board();
52          DrawBoard drawBoard = new DrawBoard(board);
53          checkersBoardView.setImage(mat2Image(drawBoard.drawBackground(board)));
54          if (!imageProcessing.cameraActive) {
55              imageProcessing = new ImageProcessing( mainController: MainController.this);
56              System.out.println(imageProcessing.capture);
57              imageProcessing.capture.open( index: 0);
58              if (imageProcessing.capture.isOpened()) {
59                  imageProcessing.cameraActive = true;
60
```

Podobnie jak wykorzystano adnotacje @FXML, jednak w tym miejscu oznacza on deklaracje wciśnięcia przycisku. Poniżej opisano działanie, na podstawie linii kodu:

- 51 - Deklaracja klasy board
- 52 - Deklaracja klasy drawBoard opisanej poniżej
- 53 - Narysowanie wirtualnej planszy
- 54 - Główny if, sprawdzający czy kamera jest włączona
- 55 - Inicjalizacja klasy zarządzania obrazem, wywoływany co wyłączenie kamery w celu zapobiegania wystąpienia NullPointerException (JAVA :/)
- 56 - Wypisanie id kamery w celu sprawdzenia czy wybrana została dobra

- 57 - Uruchomienie przechwytywania obrazu
- 58 - Sprawdzenie warunku czy przechwytywanie jest gotowe
- 59 - Ustawienie odpowiedniej flagi

```
61  TimerTask frameGrabber = () -> {
64          Mat frame = imageProcessing.grabFrame();
65          Mat topView = frame;
66          Image image = mat2Image(frame);
67          Image topViewImage = null;
68          if (imageProcessing.generatedPerspectiveMatrix == false) {
69              imageProcessing.findPerspectiveMatrix(frame, borderEdges);
70          } else {
71              topView = imageProcessing.topView(frame, imageProcessing.perspectiveMatrix);
72              topViewImage = mat2Image(topView);
73              calibrateView.setImage(topViewImage);
74              calibrateView.setFitWidth(380);
75              calibrateView.setFitHeight(400);
76              calibrateView.setPreserveRatio(true);
77          }
78          cameraView.setImage(image);
79          cameraView.setFitWidth(380);
80          cameraView.setFitHeight(400);
81          cameraView.setPreserveRatio(true);
```

- 61 - Wątek zajmujący się uruchamianiem funkcji w odstępach czasu
- 64 - Zmienna w której przechowywana jest dalej obrabiana klatka
- 65 - Zmienna pomocnicza
- 66 - Przetworzenie obrazu w celu wyświetlenia
- 67 - Zmienna pomocnicza
- 68 - Sprawdzenie warunku czy istnieje macierz projekcji
- 69 - Wywołanie funkcji odpowiedzialnej za znalezienie macierzy
- 70 - Jeżeli macierz istnieje to wykonywany jest poniższy kod
- 71 - Przypisanie widoku bazującego na macierzy projekcji
- 72 - Przetworzenie obrazu na możliwy do wyświetlenia
- 73 - Przypisanie obrazu do UI
- 74 - Ustawienie szerokości
- 75 - Ustawienie wysokości
- 76 - Ustawienie funkcji odpowiedzialnej za utrzymanie proporcji obrazu
- 77-81 - Opisane wyżej

```
83          if (gameState != 0) {
84              Mat circleFrame = imageProcessing.findCircles(topView, currentColor.getMinValues(),
85                      currentColor.getMaxValues(),  view: true);
86              Image circleImage = mat2Image(circleFrame);
87              captureView.setImage(circleImage);
88              captureView.setFitWidth(380);
89              captureView.setFitHeight(400);
90              captureView.setPreserveRatio(false);
91
92              Mat firstPlayerPawns = imageProcessing.findCircles(topView, firstPlayer.getMinValues(),
93                      firstPlayer.getMaxValues(),  view: false);
94              Mat secondPlayerPawns = imageProcessing.findCircles(topView, secondPlayer.getMinValues(),
95                      secondPlayer.getMaxValues(),  view: false);
96              drawBoard.clearBoard();
97              board.cleanBoard();
```

- 83 - Sprawdzenie czy status gry jest różny niż 0
- 81 - Nałożenie na obraz wykrytych okręgów
- 82-90 - Opisane wyżej
- 92 - Wyszukanie pionków gracza pierwszego na podstawie okręgów
- 93 - Wyszukanie pionków gracza drugiego na podstawie okręgów
- 94 - Oczyszczenie graficznej planszy
- 95 - Oczyszczenie tablicy double z wartościami

```
98                          for (double[] pawn : getPawnPositions(firstPlayerPawns)) {
99                              drawBoard.putPawn(pawn, board, State.BLUE);
100                         }
101                         for (double[] pawn : getPawnPositions(secondPlayerPawns)) {
102                             drawBoard.putPawn(pawn, board, State.GREEN);
103                         }
104
105                         Mat backGround = drawBoard.drawGame(board);
106                         checkersBoardView.setImage(mat2Image(backGround));
107                         checkersBoardView.setFitWidth(400);
108                         checkersBoardView.setFitHeight(400);
109                         checkersBoardView.setPreserveRatio(false);
110                     }
111                 };
113                 imageProcessing.timer.schedule(frameGrabber, delay: 0, period: 100);
114                 this.startCameraButton.setText("Stop Camera");
```

- 98 - Pętla iterująca wszystkie pozycje pionków
- 99 - Rysowanie pionków na planszy
- 101, 102 - Opisane powyżej
- 105-109 - Opisane powyżej
- 113 - Ustawienie timera który zajmuje się uruchamianiem programu w odpowiednich przedziałach czasu
- 114 - Zmiana wyświetlanego tekstu na przycisku

```
115                 } else {
116                     System.err.println("Impossible to open the camera connection...");
117                 }
118             } else {
119                 imageProcessing.cameraActive = false;
120                 startCameraButton.setText("Start Camera");
121                 if (imageProcessing.timer != null) {
122                     imageProcessing.timer.cancel();
123                     imageProcessing.timer = null;
124                 }
125                 imageProcessing.capture.release();
126                 cameraView.setImage(null);
127                 captureView.setImage(null);
128             }
129         }
```

- 116 - Wyświetlanie informacji w konsoli w momencie gdy kamera działa niepoprawnie
- 119 - Zmiana flagi aktywności kamery
- 120 - Ustawienie tekstu przycisku
- 121 - Sprawdzenie wartości timera
- 122 - Wyłączenie timera
- 125 - Wyłączenie przechwytywania obrazu

```
public Image mat2Image(Mat frame) {
    MatOfByte buffer = new MatOfByte();
    Imgcodecs.imencode( ext: ".png", frame, buffer);
    return new Image(new ByteArrayInputStream(buffer.toArray()));
}
```

Funkcja przetwarzająca obraz z kamery do obrazu możliwego do wyświetlenia w oknie programu.

```java
public List<double[]> getPawnPositions(Mat mat) {
    List<double[]> result = imageProcessing.matToDouble(mat);
    for (double[] i : result) {
        i[0] = i[0] / 100;
        i[1] = i[1] / 100;
    }
    return result;
}
```

Funkcja zajmująca się umieszczaniem pionków w tabeli.

```java
@FXML
public void calibrate() { imageProcessing.generatedPerspectiveMatrix = false; }
```

Deklaracja zachowania przycisku *calibrate* który ustawia flagę macierzy projekcji co powoduje ponowne jej generowanie.

```java
@FXML
public void nextMove() {
    switch (gameState) {
        case 1: {
            currentColor = firstPlayer;
            gameState = 2;
            break;
        }
        case 2: {
            currentColor = secondPlayer;
            gameState = 1;
            break;
        }
        case 0: {
            currentColor = firstPlayer;
            gameState = 1;
            break;
        }
    }
}
```

Przycisk *Start Game* który zmienia stan gry oraz kolor obecnie wyświetlanych na środkowym ekranie kółek gracza.

Rysowanie planszy

Klasa w której zaimplementowane są funkcje obsługujące prezentacje stanu gry na planszy.

```
public Integer SCREEN_SIZE;
public Integer pawnRadius;
public Integer fieldSize;
public Color bluePawn;
public Color greenPawn;
public Color lightField;
public Color darkField;
public Mat dispGameBoard;
public Mat boardBackground;

public DrawBoard(Board board) {
    this.SCREEN_SIZE = 800;
    this.pawnRadius = ((Double) (SCREEN_SIZE * 0.9 / 16)).intValue();
    this.fieldSize = SCREEN_SIZE / board.SIZE;
    this.bluePawn = new Color( B1: 255,  G1: 0,  R1: 0);
    this.greenPawn = new Color( B1: 0,  G1: 255,  R1: 43);
    this.darkField = new Color(16);
    this.lightField = new Color(256);
    this.dispGameBoard = new Mat(new Size(SCREEN_SIZE, SCREEN_SIZE), CvType.CV_8UC3);
    this.boardBackground = drawBackground(board);
}
```

Zmienne oraz ich deklaracja w konstruktorze, wykorzystywane przez całą klasę.

- SCREEN_SIZE - rozmiar ekranu, w tym przypadku 800.
- pawnRadius - promień pionka generowanego na planszy.
- fieldSize - rozmiar pola wyliczany na podstawie całkowitego rozmiaru ekranu oraz ilości pól w jednej kolumnie.
- bluePawn - zadeklarowany kolor pionków niebieskich na podstawie tablicy kolorów BGR.
- greenPawn - zadeklarowany kolor pionków zielonych na podstawie tablicy kolorów BGR.
- darkField - czarny kolor pól gry.
- lightField - jasny kolor pól gry
- dispGameBoard - wyświetlana plansza graficzna
- boardBackground - zmienna przechowująca czysty stan planszy

```
public Mat drawBackground(Board board) {
    List<Integer> lF = lightField.getMinValues();
    List<Integer> dF = darkField.getMinValues();

    Imgproc.rectangle(dispGameBoard, new Point( x: 0,  y: 0), new Point(SCREEN_SIZE, SCREEN_SIZE),
            new Scalar(dF.get(0), dF.get(1), dF.get(2)),  thickness: -1);
    for (int x = 0; x < board.SIZE; x++) {
        for (int y = 0; y < board.SIZE; y++) {
            if (y % 2 == x % 2) {
                Imgproc.rectangle(dispGameBoard, new Point( x: x * fieldSize,  y: y * fieldSize),
                        new Point( x: (x + 1) * fieldSize,  y: (y + 1) * fieldSize),
                        new Scalar(lF.get(0), lF.get(1), lF.get(2)),  thickness: -1);
            }
        }
    }
    return dispGameBoard;
}
```

drawBackground - funkcja rysująca pustą planszę. W pierwszej kolejności tworzony jest pusty kwadrat o kolorze czarnym. W pętli nakładane są mniejsze kwadraty koloru białego co daje efekt planszy.

```
public void drawPawns(Board board) {
    List<Integer> bP = bluePawn.getMinValues();
    List<Integer> gP = greenPawn.getMinValues();
    for (int x = 0; x < board.SIZE; x++) {
        for (int y = 0; y < board.SIZE; y++) {
            List<Integer> currentPawnColor = new ArrayList<>();
            if (board.boardState[x][y] == State.GREEN) {
                currentPawnColor = gP;
            } else if (board.boardState[x][y] == State.BLUE) {
                currentPawnColor = bP;
            } else {
                continue;
            }
            Imgproc.circle(dispGameBoard,
                    new Point(
                            x: ((x + .5)) * fieldSize,
                            ((y + .5) * fieldSize)),
                    pawnRadius, new Scalar(currentPawnColor.get(0), currentPawnColor.get(1), currentPawnColor.get(2)
                    ),  thickness: -1);
        }
    }
}
```

drawPawns - funkcja rysująca pionki na planszy wirtualnej. Położenie pionków znajduje się w przekazywanej przez argument klasie *board.* Kolory przypisywane są do dwóch list. Dwie główne pętle tworzą punkty x, y które są przez późniejsze ify sprawdzane jakiego koloru jest pionek na planszy. Ostatnim ruchem jest nałożenie okręgów na planszę.

```
public void putPawn(double[] pawn, Board board, State state) {
    int x = (int) (pawn[0]);
    int y = (int) (pawn[1]);
    if (x % 2 == y % 2)
        board.boardState[x][y] = state;
}
```

putPawn - funkcja nakładająca pionki na planszę, pionki nakładane są na pola czarne.

Przetwarzanie obrazu

```
public ImageProcessing(MainController mainController) {
    this.capture = new VideoCapture();
    this.cameraActive = false;
    this.timer = new Timer();
    this.perspectiveMatrix = null;
    this.generatedPerspectiveMatrix = false;
    this.mainController = mainController;
}
```

Nazwa klasy oraz zmienne przez nią wykorzystywane:

- SCREEN_SIZE - rozmiar ekranu
- EDGE_SIZE - długość krawędzi, wykorzystywane w celu poprawnego kalibrowania markerów
- timer - wyżej opisywana zmienna używana do kontroli działania programu
- cameraActive - flaga oznaczająca czy kamera jest aktywna
- perspectiveMatrix - macierz projekcji
- generatedPerspectiveMatrix - flaga oznaczająca stan macierzy
- mainController - instancja głównego kontrolera stworzona w celu wyświetlania komunikatów


```
public ImageProcessing(MainController mainController) {
    this.capture = new VideoCapture();
    this.cameraActive = false;
    this.timer = new Timer();
    this.perspectiveMatrix = null;
    this.generatedPerspectiveMatrix = false;
    this.mainController = mainController;
}
```


Konstruktor przyjmujący w argumencie instancje kontrolera. Jego zadaniem jest przypisanie wartości do wszystkich zmiennych wykorzystywanych przez daną klasę.

```
private void mat2Hsv(Mat src, Mat dst) { cvtColor(src, dst, Imgproc.COLOR_BGR2HSV); }
```

Funkcja zajmująca się transformacją modelu przestrzeni kolorów obrazu do Hsv (*ang. Hue Saturation Value*). Jako argumenty przyjmuje obraz źródłowy oraz wyjściowy. Jej głównym zadaniem jest wykonanie wbudowanej w OpenCV funkcji *cvtColor.*

```java
public Mat grabFrame() {
    Mat frame = new Mat();

    if (this.capture.isOpened()) {
        try {
            this.capture.read(frame);

        } catch (Exception e) {
            System.err.print("ERROR");
            e.printStackTrace();
        }
    }
    return frame;
}
```

Funkcja zajmująca się przechwytywaniem klatki z kamery w danym momencie. Wbudowana w OpenCV funkcja *read* przyjmuje jako argument zwracany obraz. Jej wywołanie powoduje możliwość wystąpienie wyjątku, dlatego zastosowany został try...catch.

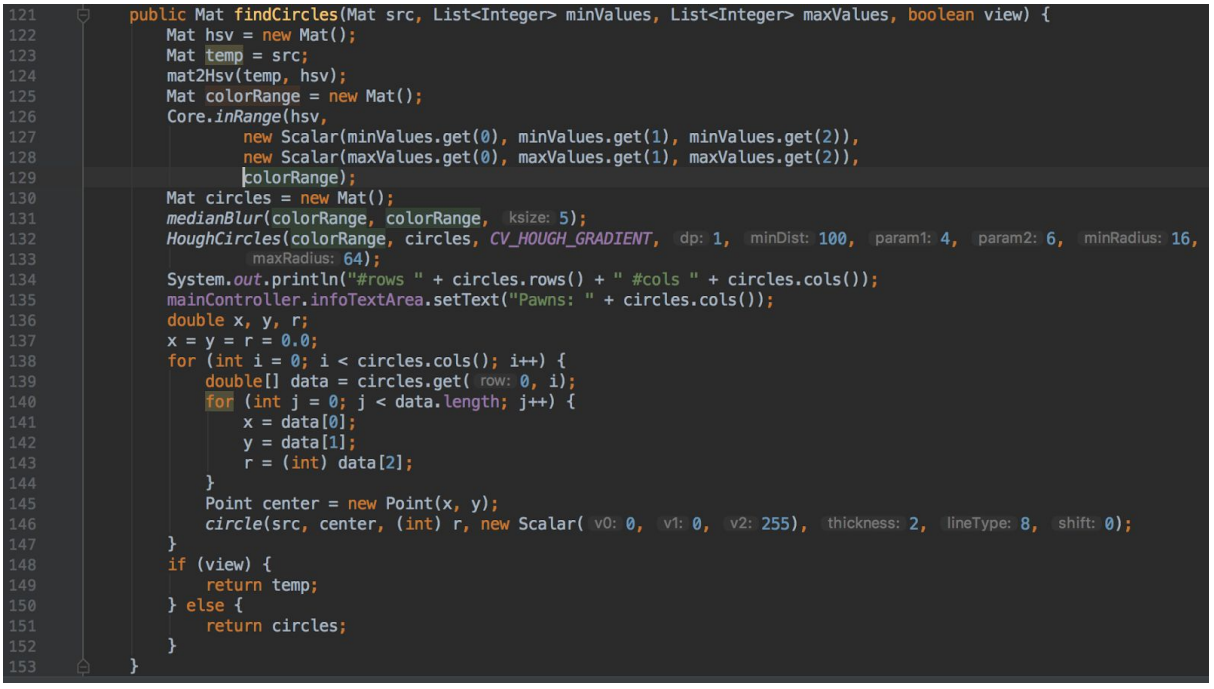
```java
63  public void findPerspectiveMatrix(Mat frame, Color color) {
64      Mat boardMarkers = findCircles(frame, color.getMinValues(), color.getMaxValues(), view: false);
65      Mat perspectiveMatrix;
66      Square board;
67      System.out.println(boardMarkers.cols());
68      if (boardMarkers != null && boardMarkers.cols() == 4) {
69          Mat boardMakersNew = boardMarkers;
70          Square square = new Square(
71                  new Point( x: 0,  y: 0),
72                  new Point( x: SCREEN_SIZE - 1,  y: 0),
73                  new Point( x: SCREEN_SIZE - 1,  y: SCREEN_SIZE - 1),
74                  new Point( x: 0,  y: SCREEN_SIZE - 1));
75          board = findCorners(boardMakersNew);
76          perspectiveMatrix =
77                  Imgproc.getPerspectiveTransform(Converters.vector_Point2f_to_Mat(board.getPoints()),
78                  Converters.vector_Point2f_to_Mat(square.getPoints()));
79          this.perspectiveMatrix = perspectiveMatrix;
80          this.generatedPerspectiveMatrix = true;
81      }
82  }
```

Funkcja wyszukująca macierz projekcji, na podstawie czterech markerów umieszczonych w rogach planszy. Poniższa analiza kodu dotyczy poszczególnych linijek:

- 64 - odszukanie markerów wyznaczających zakres planszy
- 65 - pomocnicza macierz projekcji
- 66 - powyżej przedstawiony kwadrat oznaczający rogi planszy
- 67 - wyświetlenie ilości kolumn odszukanych w linii 64
- 68 - sprawdzenie czy wykryta została poprawna ilość markerów
- 69 - pomocnicza zmienna
- 70-74 - przypisanie wartości punktów
- 75 - przekształcenie rogów planszy by były w odpowiedniej kolejności
- 76 - stworzenie macierzy projekcji na podstawie odpowiednio uporządkowanych punktów
- 79 - ustawienie macierzy projekcji
- 80 - zmiana stanu flagi obrazującej stan macierzy

```java
public Mat topView(Mat frame, Mat perspectiveMatrix) {
    Mat wrappedPerspective = new Mat();
    Imgproc.warpPerspective(frame, wrappedPerspective, perspectiveMatrix, new Size(SCREEN_SIZE, SCREEN_SIZE));
    return wrappedPerspective;
}
```

Funkcja zmieniająca perspektywę obrazu, działa tylko gdy rzeczywista plansza jest ustawiona pod kątem nie większym niż 45 stopni.



```java
121     public Mat findCircles(Mat src, List<Integer> minValues, List<Integer> maxValues, boolean view) {
122         Mat hsv = new Mat();
123         Mat temp = src;
124         mat2Hsv(temp, hsv);
125         Mat colorRange = new Mat();
126         Core.inRange(hsv,
127                 new Scalar(minValues.get(0), minValues.get(1), minValues.get(2)),
128                 new Scalar(maxValues.get(0), maxValues.get(1), maxValues.get(2)),
129                 colorRange);
130         Mat circles = new Mat();
131         medianBlur(colorRange, colorRange,  ksize: 5);
132         HoughCircles(colorRange, circles, CV_HOUGH_GRADIENT,  dp: 1,  minDist: 100,  param1: 4,  param2: 6,  minRadius: 16,
133                  maxRadius: 64);
134         System.out.println("#rows " + circles.rows() + " #cols " + circles.cols());
135         mainController.infoTextArea.setText("Pawns: " + circles.cols());
136         double x, y, r;
137         x = y = r = 0.0;
138         for (int i = 0; i < circles.cols(); i++) {
139             double[] data = circles.get( row: 0, i);
140             for (int j = 0; j < data.length; j++) {
141                 x = data[0];
142                 y = data[1];
143                 r = (int) data[2];
144             }
145             Point center = new Point(x, y);
146             circle(src, center, (int) r, new Scalar( v0: 0,  v1: 0,  v2: 255),  thickness: 2,  lineType: 8,  shift: 0);
147         }
148         if (view) {
149             return temp;
150         } else {
151             return circles;
152         }
153     }
```

findCircles jest to funkcja wyszukująca na planszy okręgów na podstawie koloru. Poniżej przedstawiony opis bazuje na liniach kodu widocznych w lewym rogu zdjęcia.

- 122 - inicjalizacja obrazu
- 123 - przypisanie obrazu do zmiennej pomocniczej
- 124 - zmiana modelu przestrzeni kolorów
- 125 - zmienna przechowująca kolory
- 126-129 - zmiana obrazu na taki który wyświetla tylko miejsca których kolor odpowiada tym w argumencie
- 130 - zmienna przechowująca odnalezione okręgi
- 131 - wykonanie rozmycia w celu wyeliminowania szumów
- 132 - wykorzystanie wbudowanej w OpenCV funkcji wyszukującej okręgi
- 135 - ustawienie napisu w UI
- 136 - utworzenie zmiennych przechowujących wartości punktów
- 137 - przypisanie zmiennym wartości 0.0
- 138 - główna pętla iterująca wszystkie wyszukane kolumny
- 139 - utworzenie i przypisanie zmiennej danych punktu
- 140-144 - przypisanie do wcześniej ustalonej tablicy danych
- 145 - utworzenie nowego punktu
- 146 - nałożenie okręgu na obraz

```java
public List<double[]> matToDouble(Mat mat) {
    List<double[]> doubles = new ArrayList<>();
    for (int i = 0; i < mat.cols(); i++) {
        doubles.add(mat.get( row: 0, i));
    }
    return doubles;
}
```

Funkcja transformująca obraz do listy punktów w postaci tablic liczb zmiennoprzecinkowych.

FXML

Warstwa graficzna aplikacji napisana w xml.

```xml
<VBox xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controllers.MainController">
   <children>
      <HBox>
         <children>
            <ImageView id="cameraView" fx:id="cameraView" fitHeight="200.0" fitWidth="380.0" pickOnBounds="true"
                       preserveRatio="true" />
            <ImageView id="captureView" fx:id="captureView" fitHeight="200.0" fitWidth="380.0" pickOnBounds="true"
                       preserveRatio="true" />
            <ImageView id="calibrateView" fx:id="calibrateView" fitHeight="200.0" fitWidth="380.0" pickOnBounds="true"
                       preserveRatio="true" />
         </children>
      </HBox>
      <HBox>
         <children>
            <TextArea id="infoTextArea" fx:id="infoTextArea" prefHeight="164.0" prefWidth="380.0" />
            <VBox>
               <children>
                  <Button id="nextMoveButton" fx:id="nextMoveButton"
                          mnemonicParsing="false" onAction="#nextMove" prefHeight="12.0" prefWidth="380.0" text="Start Game" />
                  <Button fx:id="startCameraButton" mnemonicParsing="false"
                          onAction="#startCamera" prefHeight="12.0" prefWidth="380.0" text="Start Camera" />
                  <Button fx:id="calibrateButton" mnemonicParsing="false" onAction="#calibrate"
                          prefHeight="12.0" prefWidth="380.0" text="Calibrate" />
               </children>
            </VBox>
         </children>
      </HBox>
      <HBox>
         <children>
            <ImageView id="checkersBoardView" fx:id="checkersBoardView" depthTest="DISABLE" fitHeight="400.0"
                       fitWidth="400.0" pickOnBounds="true" preserveRatio="true"></ImageView>
         </children>
      </HBox>
   </children>
</VBox>
```

Na samej górze jest napisana kontrolera do którego odnosi się ten plik

Następnie są definicje pól (w kolejności od góry pliku)

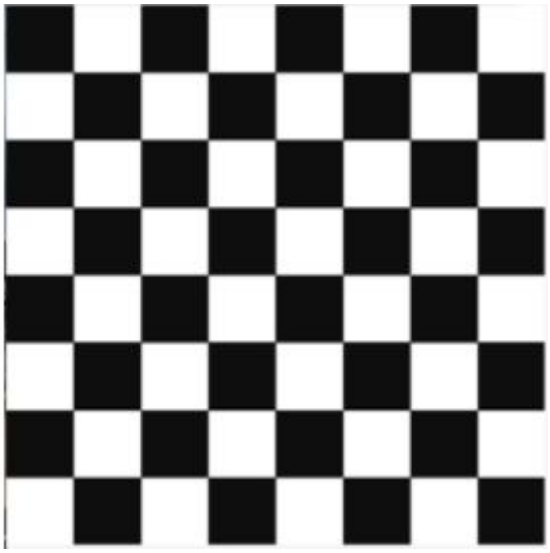
- Pole w którym wyświetla się obraz kamery
- Pole w którym wyświetla się obraz z kamery z zaznaczonymi wychwyconymi pionkami
- Pole w którym wyświetla się obraz skalibrowanej planszy

Poniżej w pliku przedstawione jest umiejscowienie buttonów oraz ich atrybuty

- button nextMoveButton który odpowiada za początek gry
- button startCameraButton który odpowiada za włączenie kamery internetowej
- button calibrateButton który odpowiada za kalibrację planszy

Wszystkie te buttony mają wysokość “12.0” oraz szerokość “300.0”

Na samym dole pliku jest definicja obrazka planszy o rozmiarach 400.0 x 400.0



Podsumowanie

Poniższy rozdział jest podsumowaniem całego projektu. Został on podzielony na cztery podrozdziały które opisują początkowe założenia które zostały spełnione oraz niespełnione. Na zakończenie przedstawione zostały napotkane problemy. Dzięki projektowi grupa poznała aspekty biblioteki OpenCV związane z przetwarzaniem obrazu oraz wykrywaniem kolorów/kształtów. Dużym plusem jest również poznanie pracy z systemem kontroli wersji które ułatwia kontrolę nad projektem.

Spełnione założenia

Głównym założeniem projektu które zostało spełnione było przechwytywanie planszy gry w warcaby w czasie rzeczywistym oraz przedstawienie stanu gry na wirtualnej planszy. Bardzo pomocny okazał się system kontroli wersji dzięki któremu bezproblemowo było można sprawdzać różne rozwiązania nie bojąc się o utratę działającego kodu.

Niespełnione założenia

Logiczna część gry - sprawdzanie czy ruch został wykonany w sposób poprawny według reguł gry w warcaby oraz brak zamiany funkcjonalność pionków.

Napotkane problemy

Jednym z większych problemów działania aplikacji są czynniki zewnętrzne a mianowicie oświetlenie miejsca gry. Kolejnym problemem napotkanym podczas rozwoju projektu były klucze SSH do GitHub'a które wymagały ciągłego resetowania z niewiadomych przyczyn na Macbook'.

Dalszy rozwój

W celu skomercjalizowania projektu należałoby wprowadzić logikę oraz lepszą obsługę błędów występujących w systemie. Dobrym krokiem byłoby również wprowadzenie bazy danych ze statystykami graczy.

Bibliografia

http://docs.opencv.org/2.4/index.html
http://docs.oracle.com/javase/8/docs/
http://docs.oracle.com/javase/8/javase-clienttechnologies.html