Politechnika Warszawska

Integracja programowa systemów multimedialnych I

Dokumentacja projektu:

Analiza gry planszowej Mastermind

Jarosław Affek

Warszawa 2018

1. Temat projektu

Zadaniem było napisanie programu w języku C++, obiektowo, z wykorzystaniem klas i zewnętrznych bibliotek. Program miał śledzić poprzez kamerę internetową (w tej funkcji komórka połączona przez Wi-Fi z komputerem) rozgrywkę w grę Mastermind i wyświetlać na ekranie podpowiedzi dla zgadującego gracza (białe i czerwone sygnalizatory). Program miał umożliwić rozgrywkę w pojedynkę z pomocą komputera i kamery. Do analizy i przetwarzania obrazu wykorzystana została biblioteka OpenCV, która dysponuje bardzo rozbudowaną bazą funkcji i klas obsługujących edycję obrazów.

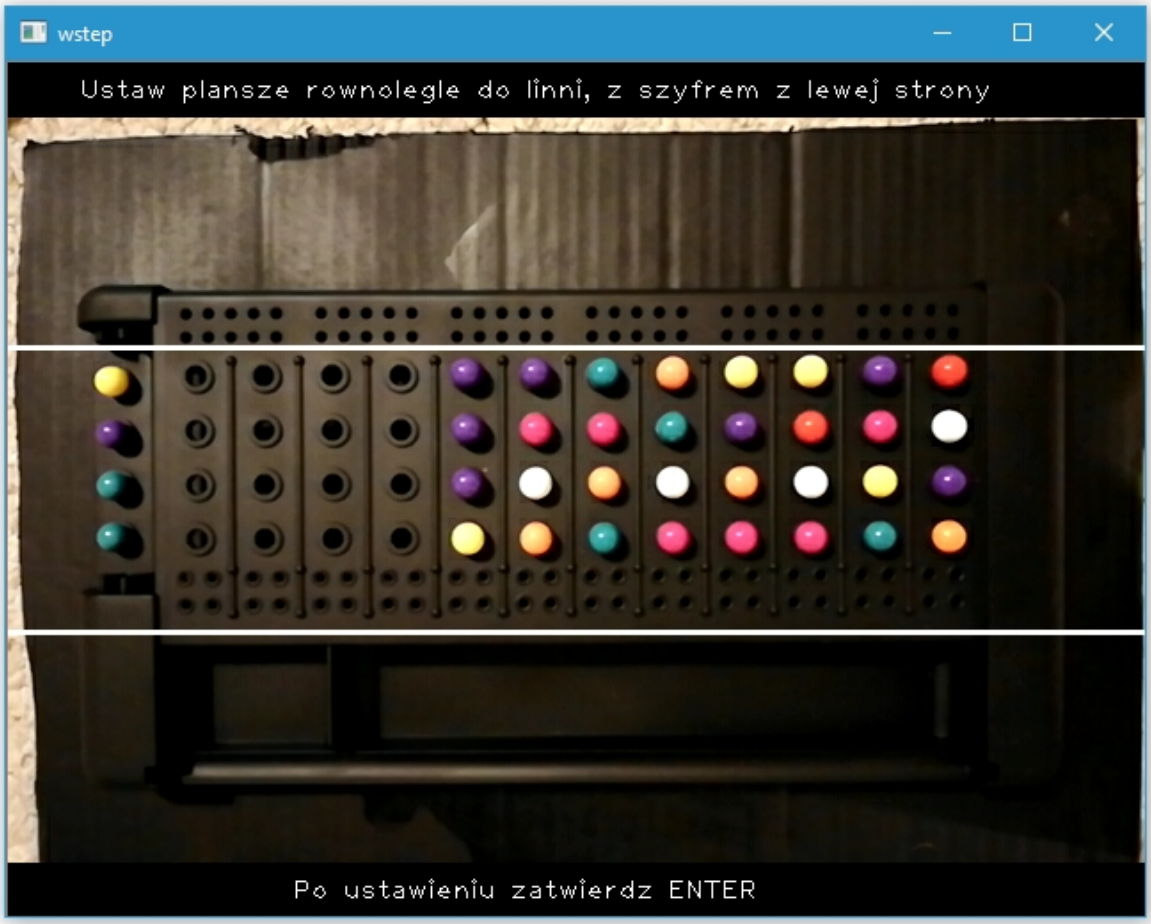
2. Napotkane problemy

Projekt opiera się na detekcji barw wybranych elementów obrazu. Aby program mógł porównywać pionki bezbłędnie należało zastosować wiele filtrów (dokładny opis w punkcie – analiza kodu). Pierwszym problemem była konieczność wyodrębnienia pionków z pozostałej części obrazu. Stało się to możliwe dzięki zastosowaniu konwersji z przestrzeni RGB na HSV i zastosowanie progowania dla wartości V (Value - jasność), dzięki temu, że pionki są jaśniejsze niż plansza udaje się wyodrębnić dość nieregularne kształty (nieregularne ze względu na materiał z jakiego są wykonane – gładki i odbijający), które następnie zaznaczane są na obrazie w skali szarości. Aby otrzymać kształty możliwie zbliżone do kół (pionków) zastosowano rozmycie (Gaussa), erozję i dylatację na obrazie w skali szarości uzyskując kształty zbliżone do kół. Kolejnym krokiem jest binaryzacja obrazu poprzez progowanie – wysoki próg zapewnia niski poziom szumów i przypadkowych odbić od planszy, które mogłyby być potraktowane jako pionek. Aby otrzymać współrzędne punktów zastosowano funkcję SimpleBlobDetector, która pozwala wyszukać obiekty o określonej powierzchni, „okrągłości", i wypełnieniu, efektem jej działania jest zbiór okręgów, których współrzędne oraz promienie pokrywają się w dużej mierze z pionkami. Mając te dane można próbkować kolory z oryginalnego obrazu w wybranych punktach. Jednak ze względu na materiał pionków (silnie odbijający) wybranie środka okręgu jako próbkę mogłoby przypadkowo trafić w „odblask" i kolor pobrany byłby zafałszowany, dlatego zastosowano średnią wartość barwy z czterech punktów równo oddalonych od środka okręgu (położonych symetrycznie „na krzyż"). Pomimo starań zniwelowania wpływu źródła światła niestety program wymaga równomiernego oświetlenia (i możliwie rozproszonego) całej planszy (nie nadają się do tego punktowe źródła światła z bliskiej odległości, ponieważ powodują one silne odblaski i różnice w barwie pionków położonych najbliżej i najdalej od źródła światła). Nawet przy stosunkowo dobrym oświetleniu problemem okazało się błędne rozpoznawanie pionków tej samej barwy, ponieważ nigdy nie udaje się osiągnąć oświetlenia idealnego. Dlatego metodą eksperymentalną wyznaczono przedziały wartości H i S z przestrzeni barw HSV, dla której przy dobrych warunkach oświetleniowych pionki są rozpoznawane prawidłowo. Kolejnym problemem z tym związanym było wykrywanie barwy czerwonej, ponieważ jej wartości Hue na kole kolorów mają wartości ok. 0 – 10 i 170 – 179 (dla Opencv, gdzie pełne 360 stopni koła jest podzielone na dwa ze względu na taką pojemność typu uchar (max 255)). Aby wyznaczyć średnią z barw oscylujących w granicach 179 i 0 konieczne było przybliżenie wartości oscylujących do stałej – 179. Głównym elementem programu jest stworzona prosta klasa reprezentująca pojedynczy pionek i przechowująca dane takie jak barwa (w różnych przestrzeniach), sumy kanałów barwnych, współrzędne i inne. Program celowo został wyposażony w kilka początkowych faz przygotowania do gry (ustawienie planszy, ROI, threshold), ponieważ założenie jest takie że program może być wykorzystywany w różnych

miejscach. Gdyby założyć, że program ma zawsze działać w określonym miejscu, z identycznym oświetleniem i zamocowanie kamery można by pozbyć się tych etapów przygotowujących i zastąpić je pewnymi stałymi, bez możliwości regulacji. Jednak większa uniwersalność programu kosztem drobnego wysiłku przy wstępnym ustawieniu programu jest moim zdaniem opłacalna.

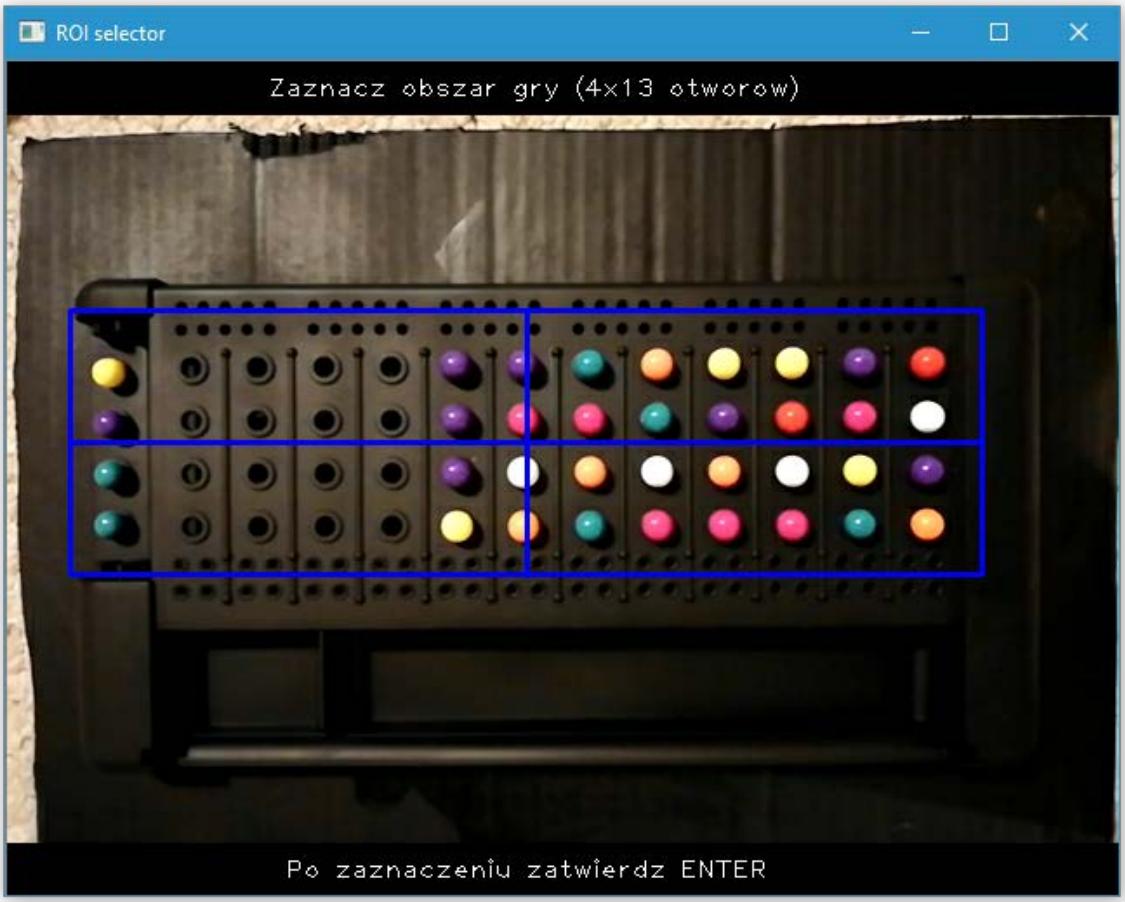
3. Instrukcja obsługi

Program umożliwia samodzielną grę w Mastermind. Po uruchomieniu programu widoczne jest okno z podglądem z kamery. Planszę do gry należy umieścić poziomo z szyfrem do odgadnięcia po LEWEJ stronie obrazu. Poziome linie naniesione na obraz mają za zadanie ułatwić ułożenie planszy poziomo – rysunek 1. Po prawidłowym ustawieniu wcisnąć ENTER.



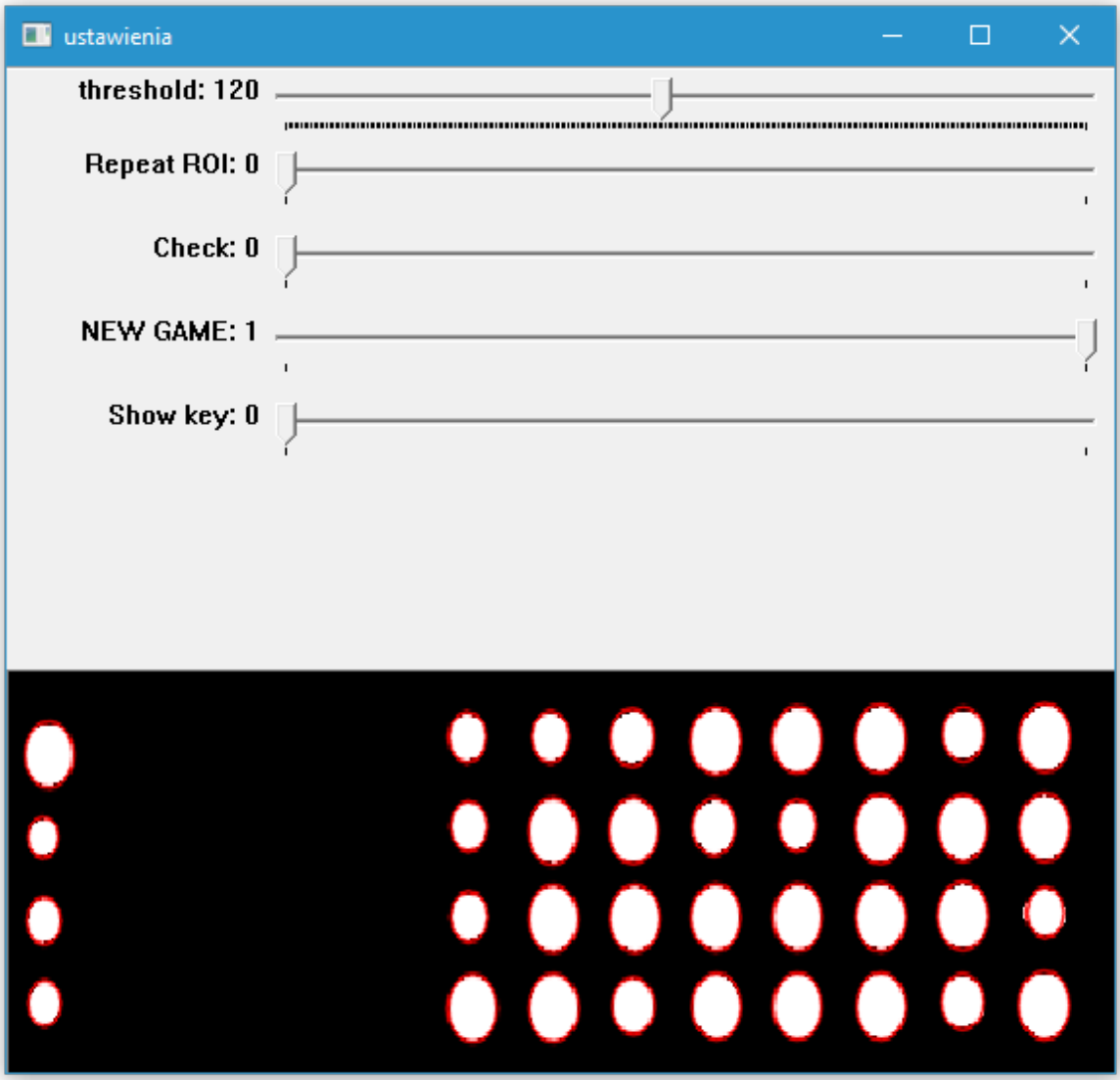
Rysunek 1

Kolejnym krokiem jest zaznaczenie tzw. ROI (Region of interest) za pomocą myszki. W tym przypadku ROI to wszystkie główne pola do gry (bez pól pomocniczych) – zgodnie z rys. 2.



Rysunek 2

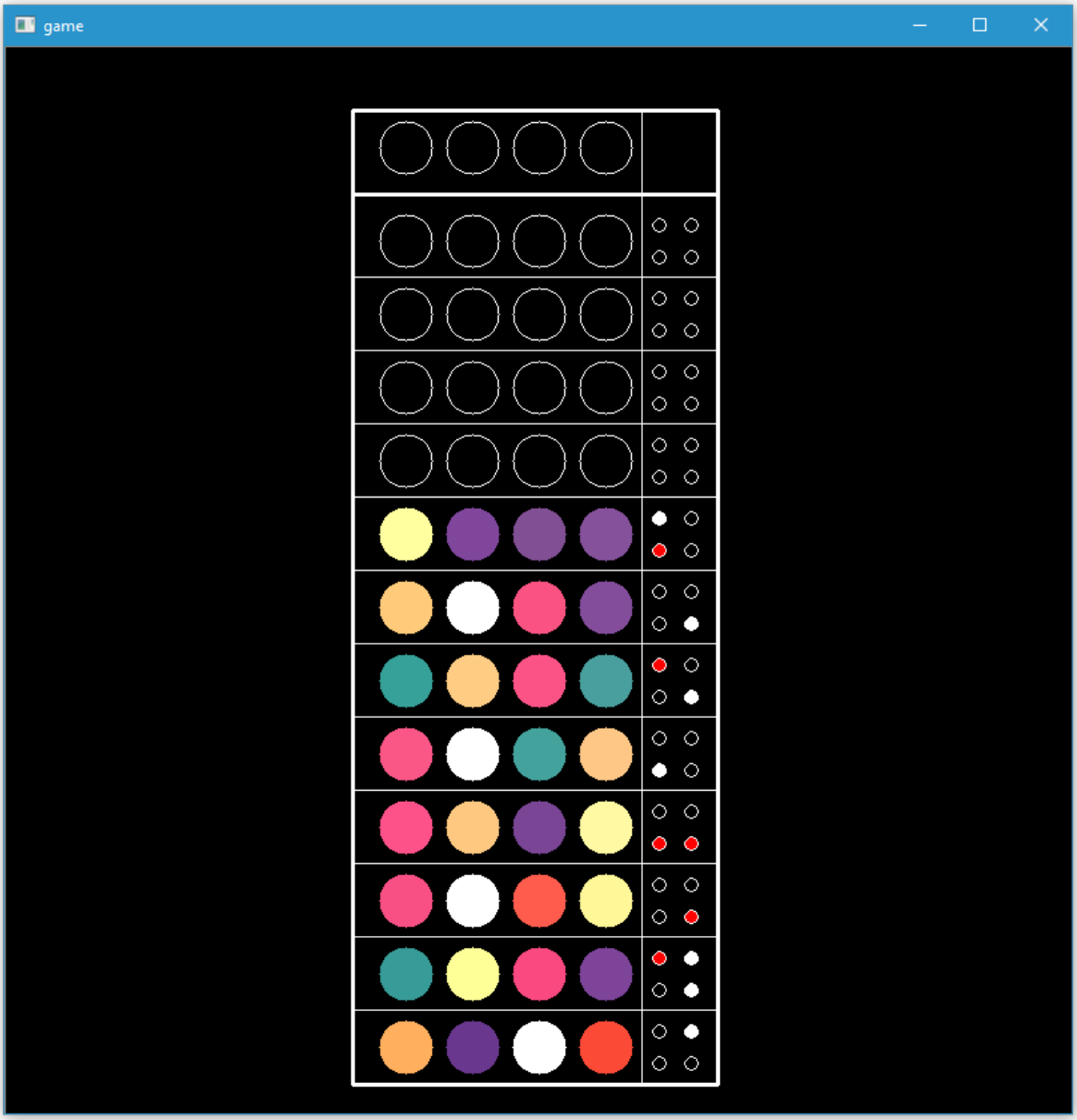
Po zatwierdzeniu ENTERem otwiera się panel gry oraz panel kontrolny – rys. 3.



Rysunek 3

Pasek threshold służy do odpowiedniego ustawienia progu jasności. Należy go ustawić w taki sposób aby wszystkie pionki na planszy do gry były widoczne w podglądzie binarnym poniżej, ale nie były widoczne szumy i inne odblaski z planszy do gry. Prawidłowe ustawienie tego suwaka można poznać po tym, że w podglądzie binarnym na dole okna wszystkie białe koła reprezentujące pionki na planszy zostały obwiedzione czerwonym okręgiem. Prawidłowe ustawienie jest zaprezentowane na rys. 3. Rozmiary kół powinny być jak najbardziej zbliżone do siebie, można to osiągnąć oświetlając planszę możliwie równomiernie. Suwak Kolejne suwaki pełnią rolę przycisków (mają tylko 2 możliwe ustawienia). Przesunięcie suwaka Repeat ROI pozwala ponownie zaznaczyć ROI. Suwak Check służy do uruchomienia algorytmu programu – aby program mógł zareagować na fizyczne zmiany rozgrywki należy zawsze wcisnąć przycisk Check. Suwak NEW GAME należy użyć gdy gracz chce rozpocząć nową grę i zmienił się szyfr konieczny do odgadnięcia (następnie zawsze należy użyć suwaka Check). Suwak Show key pozwala ujawnić na ekreanie poszukiwany szyfr (następnie zawsze należy użyć suwaka Check). Wyjście z

programu jest możliwe poprzez wciśnięcie przycisku ESC. Poprawnie wczytana plansza powinna wyglądać jak na rys. 4.



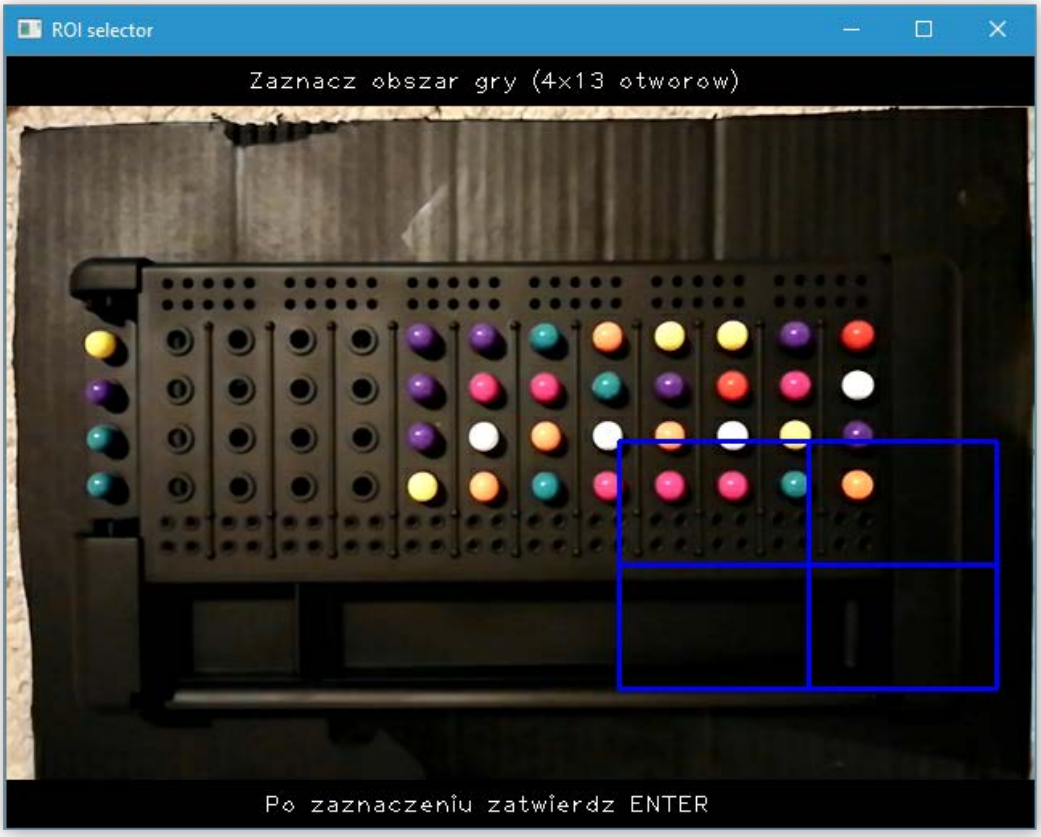
Rysunek 4

Poszukiwany szyfr zostanie odsłonięty na ekranie po użyciu suwaka Show key jak również po odgadnięciu szyfru w sekcji prób.

4. Wykonane testy

Aby zapewnić sprawną pracę programu zastosowano odpowiednią ilość mechanizmów zabezpieczających przez zawieszeniem się programu wywołanym błędnym zachowaniem użytkownika. Ze względu na sposób działania funkcji SimpleBlobDetector (wyszukuje bloby poczynając od dołu okna) konieczne jest ustawienie planszy poziomo względem okna programu. W przypadku błędnego zaznaczenia ROI lub ustawienia planszy niezgodnie z wytycznymi użytkownik otrzyma komunikat i będzie mógł poprawić zaznaczenie – rys. 3 i 4.

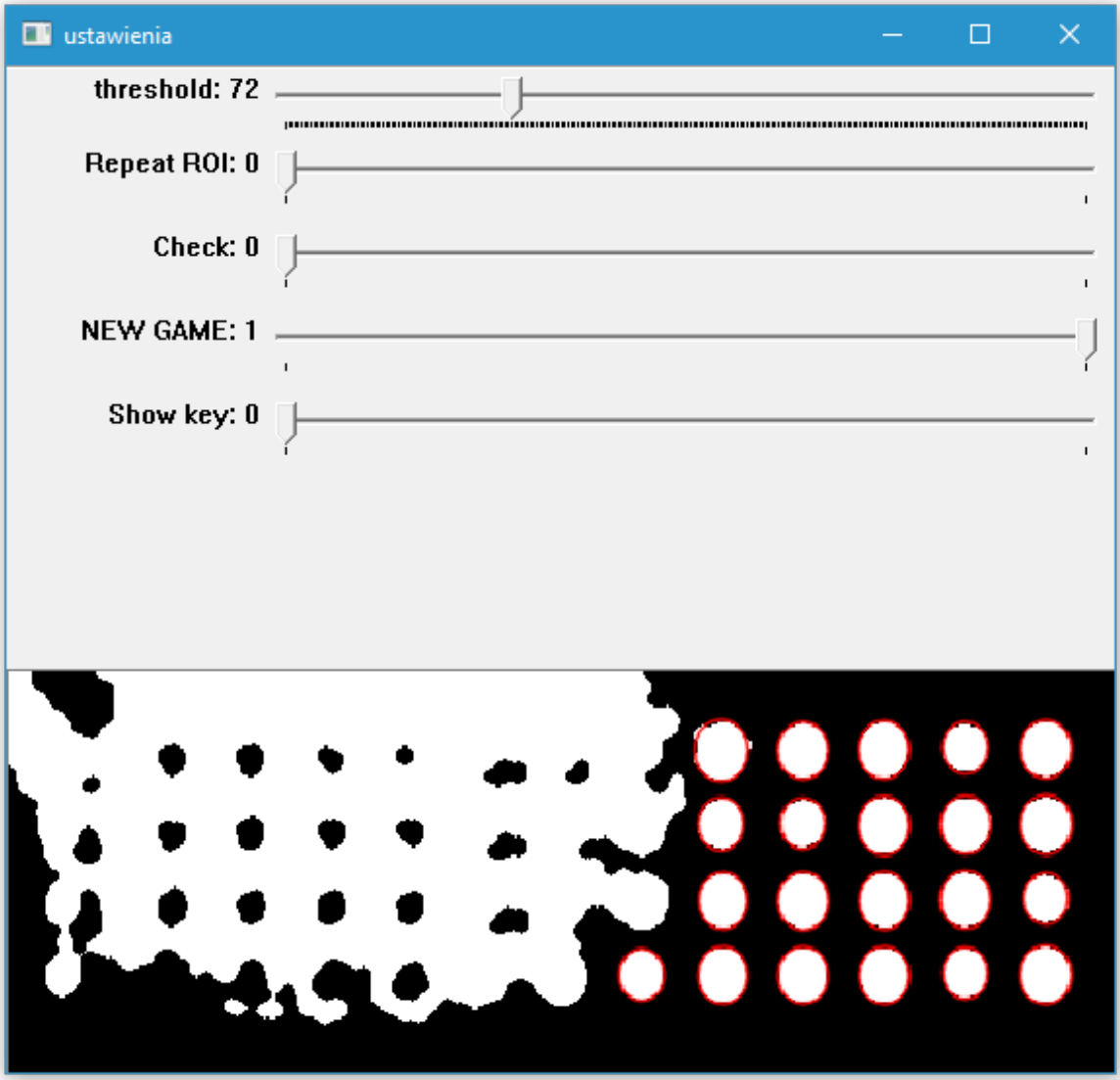
Rysunek 5 Komunikat o błędzie



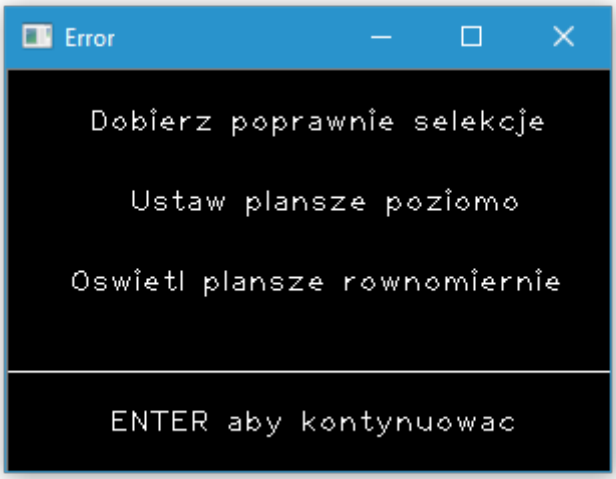
Rysunek 6 Przykład błędnego zaznaczenia

W tym momencie sprawdzana jest jedynie odpowiednia wielkość ROI. Po otwarciu panelu ustawień i ustawieniu suwaka threshold a następnie Check, program sprawdza poprawność następujących elementów: liczbę wykrytych blobów (pionki), położenie planszy, liczbę umieszczonych na planszy pionków. Jeśli wykryje błąd pojawią się komunikat i należy sprawdzić poprawność ustawienia pionków, planszy i suwaka threshold. Przykład źle

ustawionego suwaka threshold i efekt w postaci komunikatu uniemożliwiającego prawidłową pracę programu rys. 7 i 8.



Rysunek 7 Złe ustawnie suwaka threshold



Rysunek 8 Komunikat o błędzie

Program został przetestowany dla wielu różnych układów pionków i wyniki działania w zdecydowanej większości przypadków są poprawne. Błędne działanie programu wynikać może najczęściej z jakości oświetlenia planszy do gry.

5. Analiza kodu

Opisane zostaną wszystkie stworzone funkcje na potrzeby programu w kolejności ich umieszczenia w kodzie.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "opencv2/features2d.hpp"
#include "opencv2/features2d/features2d.hpp"
#include <iostream>
#include <cstdlib>

using namespace std;
using namespace cv;

class Pionek
{
public:
	float x;
	float y;
	int H;
	int S;
	int V;
	uchar B;
	uchar G;
	uchar R;
	int sHS;  // suma kanalow H i S
	float size;  //rozmiar bloba
	Point kord;
    void inicjuj(int nr_rzedu, int poz, int l_rzedow, const Mat& obraz, const Mat&
obrazhsv, const vector<KeyPoint>& bloby);
};

class Podp
{
```

```
public:
	int B;
	int G;
	int R;
	void nadaj(int red, int green, int blue);
};
```

Powyższy kod to wnętrze pliku pionek.h – czyli deklaracja 2 klas wykorzystywanych w programie – Pionek i Podp. Klasa Pionek to klasa, która odpowiada cechom jednego pionka, czyli współrzędne - x, y, składowe barwne w dwóch przestrzeniach barwnych RGB i HSV, suma poszczególnych składowych barw (sHS) – służy wykrywaniu podobnych kolorów, size – czyli rozmiar danego bloba, kord – czyli znowu współrzędne w postaci punktu oraz jedna funkcja – kontruktor. Klasa Podp zawiera jedynie składowe barw RGB, nie jest ona konieczna, ponieważ można by zastosować zwykły wektor wieloelementowy w tym celu.

```
#include "pionek.h"

using namespace std;
using namespace cv;

void Pionek::inicjuj(int nr_rzedu, int poz, int l_rzedow, const Mat& obraz, const Mat&
obrazhsv, const vector<KeyPoint>& bloby)
{
		int k = nr_rzedu + (poz * l_rzedow);
		Vec3b intensity1 = obraz.at<Vec3b>(bloby[k].pt.y + bloby[k].size / 4,
bloby[k].pt.x);
		Vec3b intensity2 = obraz.at<Vec3b>(bloby[k].pt.y - bloby[k].size / 4,
bloby[k].pt.x);
		Vec3b intensity3 = obraz.at<Vec3b>(bloby[k].pt.y, bloby[k].pt.x +
bloby[k].size / 4);
		Vec3b intensity4 = obraz.at<Vec3b>(bloby[k].pt.y, bloby[k].pt.x -
bloby[k].size / 4);
		Vec3b intensity;
		intensity.val[0] = (intensity1.val[0] + intensity2.val[0] +
intensity3.val[0] + intensity4.val[0]) / 4;
		intensity.val[1] = (intensity1.val[1] + intensity2.val[1] +
intensity3.val[1] + intensity4.val[1]) / 4;
		intensity.val[2] = (intensity1.val[2] + intensity2.val[2] +
intensity3.val[2] + intensity4.val[2]) / 4;
		Vec3b intensity5 = obrazhsv.at<Vec3b>(bloby[k].pt.y + bloby[k].size / 4,
bloby[k].pt.x);
		Vec3b intensity6 = obrazhsv.at<Vec3b>(bloby[k].pt.y - bloby[k].size / 4,
bloby[k].pt.x);
		Vec3b intensity7 = obrazhsv.at<Vec3b>(bloby[k].pt.y, bloby[k].pt.x +
bloby[k].size / 4);
		Vec3b intensity8 = obrazhsv.at<Vec3b>(bloby[k].pt.y, bloby[k].pt.x -
bloby[k].size / 4);
		Vec3b intensity0;
		int pom = -1;
		int pom1 = -1;
		if ((int)intensity5.val[0] < 20 || (int)intensity6.val[0] < 20 ||
(int)intensity7.val[0] < 20 || intensity8.val[0] < 20)
			pom = 1;
		if ((int)intensity5.val[0] > 160 || (int)intensity6.val[0] > 160 ||
(int)intensity7.val[0] > 160 || intensity8.val[0] > 160)
			pom1 = 1;
		intensity0.val[0] = (intensity5.val[0] + intensity6.val[0] +
intensity7.val[0] + intensity8.val[0]) / 4;
```

```
			intensity0.val[1] = (intensity5.val[1] + intensity6.val[1] +
intensity7.val[1] + intensity8.val[1]) / 4;
			intensity0.val[2] = (intensity5.val[2] + intensity6.val[2] +
intensity7.val[2] + intensity8.val[2]) / 4;
			x = bloby[k].pt.x;
			y = bloby[k].pt.y;
			if (pom == 1 && pom1 == 1 || (int)intensity0.val[0] < 5)
				H = 179;
			else
				H = intensity0.val[0];
			S = intensity0.val[1];
			V = intensity0.val[2];
			B = intensity.val[0];
			G = intensity.val[1];
			R = intensity.val[2];
			kord = Point(bloby[k].pt.x, bloby[k].pt.y);
			sHS = (int)H + (int)S;
}
void Podp::nadaj(int red, int green, int blue)
{
	R = red;
	G = green;
	B = blue;
}
```

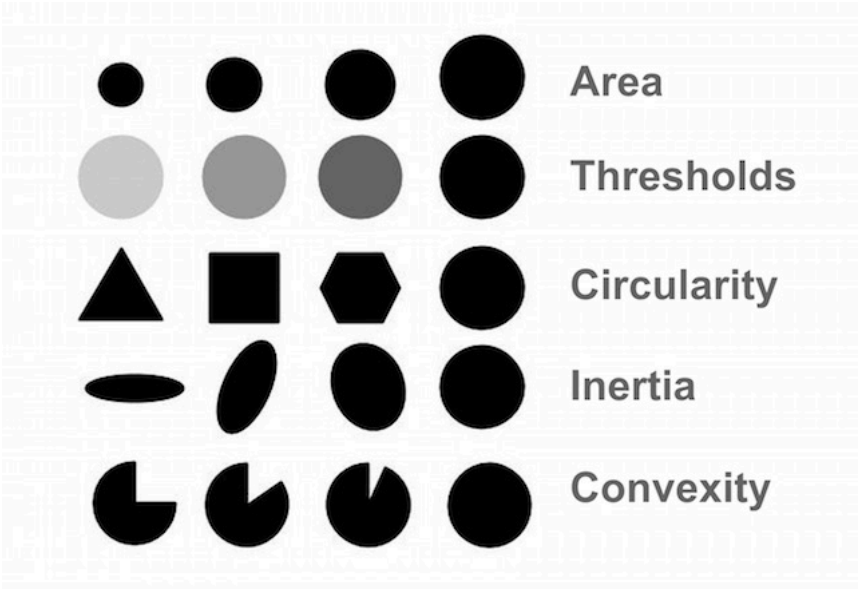
Powyższy kod to wnętrze pliku pionek.cpp. Konstruktor inicjuj na podstawie dostarczonych przy wywołaniu danych, oblicza wartości kanałów barwnych jako średnia z 4 punktów równoodległych od środka bloba. Dla koloru czerwonego i przestrzeni HSV należało dodatkowo zastosować warunki umożliwiające nadanie koloru czerwonego inaczej niż ze średniej, ponieważ wartości H dla czerwieni to ok. 0 – 7 i 175 – 179 dlatego niemożliwe było policzenie średniej z 4 punktów (przykładowo wartości 0, 1, 4, 179 dają średnią arytmetyczną daleką od koloru czerwonego). W funkcji przypisywane są też koordynaty dla poszczególnych pionków.

```
void imgtransf()
{
	int lowH = 0;
	int highH = 179;
	int lowS = 0;
	int highS = 255;
	int highV = 255;
	Mat hsvImg;
	cvtColor(imgOriginal, hsvImg, CV_BGR2HSV);
	inRange(hsvImg, Scalar(lowH, lowS, lowV), Scalar(highH, highS, highV),
threshImg);
	GaussianBlur(threshImg, threshImg, Size(3, 3), 0);
	int dilate_size = 6;
	Mat element = getStructuringElement(MORPH_ELLIPSE,Size(2 * dilate_size + 1, 2 *
dilate_size + 1),Point(dilate_size, dilate_size));
	dilate(threshImg, threshImg, element);
	int erosion_size = 3;
	Mat element1 = getStructuringElement(MORPH_ELLIPSE, Size(2 * erosion_size + 1, 2
* erosion_size + 1), Point(erosion_size, erosion_size));
	erode(threshImg, threshImg, element1);
	threshold(threshImg, threshImg, 230, 255, THRESH_BINARY);
}
```

Powyższa funkcja wykonuje podstawowe operacje graficzne. Obraz wejściowy z kamery jest zamieniany na przestrzeń RGB -> HSV. Następnie za pomocą progowania powstaje obraz binarny (progowanie obejmuje jedynie wartość V czyli jasność, którą regulujemy suwakiem threshold), następnie obraz jest rozmywany i wykonywana jest operacja dylatacji i erozji aby uzyskane kształty stały się bardziej zbliżone do koła. Po tych operacjach (zwłaszcza rozmycia) obraz stał się obrazem w skali szarości dlatego aby usunąć pośrednie tony i zlikwidować szumy i odblaski ponownie wykonywane jest progowanie.

```
void blobcreator()
{
	bitwise_not(threshImg, tmp);
	SimpleBlobDetector::Params params;
	params.minThreshold = 200;
	params.maxThreshold = 255;
	params.filterByArea = true;
	params.minArea = 150;
	params.filterByCircularity = true;
	params.minCircularity = 0.6;
	params.filterByConvexity = true;
	params.minConvexity = 0.6;
	params.filterByInertia = true;
	params.minInertiaRatio = 0.6;
	Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);
	detector->detect(tmp, keypoints);
	drawKeypoints(threshImg, keypoints, im_with_keypoints, Scalar(0, 0, 255),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
}
```

Powyższa funkcja to funkcja omawiana już na wstępie. Służy ona do wyodrębnienia z obrazu binarnego kształtów określonych przez powyższe parametry. Znaczenie tych parametrów najlepiej wyjaśnia rysunek 9.



Rysunek 9

Zasadniczo program ma wyszukiwać koła, jednak ze względu na wysoki współczynnik odbicia pionków należy uwzględnić dość duże wahania kształtów reprezentujących pionki, które mogą

odbiegać znacząco od koła. Efektem działania tej funkcji jest wektor okręgów (współrzędnych i promieni) – keypoints, o wartościach najbardziej zbliżonych do wyszukanych blobów. Wyszukiwanie blobów odbywa się od dołu okna.

```
void okienka()
{
	namedWindow("ustawienia", CV_WINDOW_NORMAL);
	//namedWindow("imgOriginal1", CV_WINDOW_AUTOSIZE);
	namedWindow("game", CV_WINDOW_AUTOSIZE);
	createTrackbar("threshold", "ustawienia", &lowV, 255);
	//createTrackbar("Próba", "ustawienia", &proba, 12);
	//createTrackbar("Numer", "ustawienia", &numer, 3);
	createTrackbar("Repeat ROI", "ustawienia", &zaz_pon, 1);
	createTrackbar("Check", "ustawienia", &doit, 1);
	createTrackbar("NEW GAME", "ustawienia", &nowa, 1);
	createTrackbar("Show key", "ustawienia", &pokaz, 1);
	//imshow("ustawienia", threshImg);
	imshow("ustawienia", im_with_keypoints);
	//imshow("imgOriginal1", imgOriginal1);
	imshow("game", background);

}
```

Funkcja ta odpowiada za wyświetlanie poszczególnych obrazów i suwaków w określonych oknach. Elementy w komentarzu służą do ewentualnego sprawdzenia działania programu gdyby okazało się, że nie dział on poprawnie (umożliwiają m. in. wypisanie wartości HSV w oknie poleceń dla poszczególnych pionków).

```
void sortowanie()
{
	rzad = ile / 4;
	for (int k = 0; k < ile; k = k + rzad)
		for (int i = 0; i < rzad - 1; i++)
			for (int j = 0; j < rzad - 1; j++)
				if (keypoints[j + k].pt.x > keypoints[j + 1 + k].pt.x)
					swap(keypoints[j + k], keypoints[j + 1 + k]);
}
```

Funkcja ta to najprostsze sortowanie bąbelkowe. Nie jest ono zbyt wydajne, ale przy tak niewielkiej liczbie elementów (do 13 w rzędzie) sprawdza się bardzo dobrze i jest przejrzyste. Funkcja ta jest niezbędna aby uporządkować wg współrzędnych wykryte bloby. Ponieważ wykrywanie blobów odbywa się od dołu okna to każdy rząd można uporządkować rosnąco wg współrzędnej x aby potem móc odwoływać się do danych pionków w kolejnych funkcjach. Właśnie dlatego ustawienie planszy musi być poziome, ponieważ w innym przypadku blobów nie dałoby się posortować i ich kolejność byłaby losowa.

```
void tablica()
{
	Mat imgHsv;
	cvtColor(imgOriginal, imgHsv, CV_BGR2HSV);
	pionki = new Pionek * [rzad];
	for(int i = 0; i < rzad; i++)
		pionki[i] = new Pionek[4];
	for (int i = 0; i < rzad; i++)
		for (int j = 0; j < 4; j++)
		{
			if(i != 0)
```

```
                        pionki[i][j].inicjuj(i, j, rzad, imgOriginal1, imgHsv,
keypoints);
                else
                        if (nowa == 1 && doit == 1)
                        {
                                szyfr[j].inicjuj(0, j, rzad, imgOriginal1, imgHsv,
keypoints);

                        }
        }
}
```

Funkcja tworząca tablicę dynamiczną dwuwymiarową typu Pionek, pierwsza współrzędna tablicy to nr rzędu a druga to numer pionka w sekwencji (od 0 do 3). Wielkość tablicy zależna jest od liczby wykrytych blobów (czyli liczby podejść prowadzących do odgadnięcia szyfru). Sam szyfr zapisywany jest w tablicy jednowymiarowej typu Pionek.

```
void rysuj()
{
        int koord[13];
        int koord1[13];
        int pom = 750;
        int pom1 = 750;
        for (int i = rzad - 1; i > 0; i--)
        {
                koord[i] = pom;
                pom = pom - 55;
        }
        for (int i = 12; i > 0; i--)
        {
                koord1[i] = pom1;
                pom1 = pom1 - 55;
        }
        for (int i = 12; i > 0; i--)
                for (int j = 3; j >= 0; j--)
                        circle(background, Point(300 + 50 * j, koord1[i]), 20, Scalar(255,
255, 255), 1, 8, 0);
        for(int i = rzad - 1; i > 0; i--)
                for (int j = 3; j >= 0; j--)
                        circle(background, Point(300+50*j,koord[i]), 20,
Scalar((int)pionki[i][j].B, (int)pionki[i][j].G, (int)pionki[i][j].R), -5, 8, 0);
        if (pokaz == 1)
        {
                circle(background, Point(300, 75), 20, Scalar((int)szyfr[0].B,
(int)szyfr[0].G, (int)szyfr[0].R), -5, 8, 0);
                circle(background, Point(350, 75), 20, Scalar((int)szyfr[1].B,
(int)szyfr[1].G, (int)szyfr[1].R), -5, 8, 0);
                circle(background, Point(400, 75), 20, Scalar((int)szyfr[2].B,
(int)szyfr[2].G, (int)szyfr[2].R), -5, 8, 0);
                circle(background, Point(450, 75), 20, Scalar((int)szyfr[3].B,
(int)szyfr[3].G, (int)szyfr[3].R), -5, 8, 0);
        }
        for(int i = 300; i<=450; i = i + 50)
                circle(background, Point(i, 75), 20, Scalar(255,255,255), 1, 8, 0);
        for(int i = rzad - 1; i > 0; i--)
                for (int j = 0; j < 4; j++)
                {
                        switch (j)
                        {
                        case 0:
```

```
                        circle(background, Point(490, koord[i] - 12), 5,
Scalar(key[i][j].B, key[i][j].G, key[i][j].R), -1, 2, 0);
                        break;
                    case 1:
                        circle(background, Point(514, koord[i] - 12), 5,
Scalar(key[i][j].B, key[i][j].G, key[i][j].R), -1, 2, 0);
                        break;
                    case 2:
                        circle(background, Point(490, koord[i] + 12), 5,
Scalar(key[i][j].B, key[i][j].G, key[i][j].R), -1, 2, 0);
                        break;
                    case 3:
                        circle(background, Point(514, koord[i] + 12), 5,
Scalar(key[i][j].B, key[i][j].G, key[i][j].R), -1, 2, 0);
                        break;
                    }
            }
      for (int i = 12; i > 0; i--)
            for (int j = 0; j < 4; j++)
            {
                    switch (j)
                    {
                    case 0:
                        circle(background, Point(490, koord1[i] - 12), 5,
Scalar(255, 255, 255), 1, 2, 0);
                        break;
                    case 1:
                        circle(background, Point(514, koord1[i] - 12), 5,
Scalar(255, 255, 255), 1, 2, 0);
                        break;
                    case 2:
                        circle(background, Point(490, koord1[i] + 12), 5,
Scalar(255, 255, 255), 1, 2, 0);
                        break;
                    case 3:
                        circle(background, Point(514, koord1[i] + 12), 5,
Scalar(255, 255, 255), 1, 2, 0);
                        break;
                    }
            }
      for (int i = 722; i >= 172; i = i - 55)
            line(background, Point(260,i), Point(534,i),Scalar(255, 255, 255), 1, 8,
0);
      line(background, Point(260, 110), Point(534, 110), Scalar(255, 255, 255), 2, 8,
0);
      line(background, Point(260, 47), Point(534, 47), Scalar(255, 255, 255), 2, 8,
0);
      line(background, Point(260, 778), Point(534, 778), Scalar(255, 255, 255), 2, 8,
0);
      line(background, Point(260, 47), Point(260, 778), Scalar(255, 255, 255), 2, 8,
0);
      line(background, Point(534, 47), Point(534, 778), Scalar(255, 255, 255), 2, 8,
0);
      line(background, Point(477, 47), Point(477, 778), Scalar(255, 255, 255), 1, 8,
0);
}
```

Funkcja rysująca wirtualną planszę do gry. Zależnie od postępu w wypełnianiu otworów, plansza zostaje wypełniana pionkami dużymi jak również małymi pomocniczymi. Miejsca puste również są oznaczane jako białe puste w środku okręgi. Warto zaznaczyć, że pionki główne rysowane na planszy są takiego koloru jaki został wykryty przez program. Dlatego

może się zdarzyć, że pionki teoretycznie tego samego koloru mogą na ekranie mieć minimalnie różną barwę (wynika to z różnicy oświetlenia oddalonych od siebie pionków). Często też barwa fizycznego pionka może być odbierana przez ludzkie oko jako inna niż ta wykryta przez program, właśnie dlatego cała plansza jest rysowana na ekranie aby można było korzystać z obu wersji.

```
void klucz()
{
	int **pomoc = new int *[rzad];
	for (int i = 0; i < rzad; i++)
		pomoc[i] = new int [4];
	for (int i = rzad - 1; i >= 0; i--)
		for (int j = 3; j >= 0; j--)
			pomoc[i][j] = 0;
	int **pomoc1 = new int *[rzad];
	for (int i = 0; i < rzad; i++)
		pomoc1[i] = new int[4];
	for (int i = rzad - 1; i >= 0; i--)
		for (int j = 3; j >= 0; j--)
			pomoc1[i][j] = 0;
	key = new Podp *[rzad];
	for (int i = 0; i < rzad; i++)
		key[i] = new Podp[4];
	// szukanie czerwonych
	for (int i = rzad - 1; i > 0; i--)
		for (int j = 3; j >= 0; j--)
		{
			key[i][j].nadaj(0, 0, 0);
			if ((int)pionki[i][j].H >= ((int)szyfr[j].H - 5) &&
(int)pionki[i][j].H <= ((int)szyfr[j].H + 5) && (int)pionki[i][j].S >=
((int)szyfr[j].S - 100) && (int)pionki[i][j].S <= ((int)szyfr[j].S + 120))
			{
				key[i][j].nadaj(255, 0, 0);
				pomoc[i][j] = 1;
				pomoc1[i][j] = 1;
				if(key[i][0].R == 255 && key[i][1].R == 255 && key[i][2].R
== 255 && key[i][3].R == 255) pokaz = 1;
			}
		}
	// szukanie białych
	for (int i = rzad - 1; i > 0; i--)
		for (int j = 3; j >= 0; j--)
		{
			for (int k = 3; k >= 0; k--)
			{
				if (pomoc[i][j] == 0 && pomoc1[i][k] == 0 &&
(int)pionki[i][j].H >= ((int)szyfr[k].H - 5) && (int)pionki[i][j].H <=
((int)szyfr[k].H + 5) && (int)pionki[i][j].S >= ((int)szyfr[k].S - 100) &&
(int)pionki[i][j].S <= ((int)szyfr[k].S + 120))
				{
					key[i][j].nadaj(255, 255, 255);
					pomoc[i][j] = 1;
					pomoc1[i][k] = 1;
				}
			}
		}
	for (int i = 0; i < rzad - 1; i++)
		delete[] pomoc[i];
	delete[] pomoc;
```

```
	for (int i = 0; i < rzad - 1; i++)
		delete[] pomoc1[i];
	delete[] pomoc1;
}
```

Główna funkcja programu decydująca o rozpoznaniu podobieństwa kolorów i wypełnieniu otworów pionkami określającymi prawidłowość próby. Metodą eksperymentalną dobrano wartości jakimi mogą się różnić poszczególne składowe barwne pionków aby można było je uznać za takie same. Każdy pionek porównywany jest z każdym pionkiem z szyfru i jeśli wartość Hue (odpowiadająca za barwę) nie różni się więcej niż o -+5 oraz wartość S (Saturation) nie różnic się o więcej niż -100 +120 to pionki uznawane są jako pionki o jednakowej barwie. Wartości te można modyfikować jak również dodać warunki związane z kanałami R, G, B, V oraz ich sumami w przypadku chęci używania programu w nietypowym oświetleniu. Należało również zastosować zabezpieczenie aby prawidłowość każdego pionka oznaczyć tylko raz (na biało lub czerwono). Aby uniknąć nadpisywania się wskazówek zastosowano dwie tablice pomocnicze, które określają czy dany pionek już był porównany z pozytywnym efektem z szyfrem.

```
int test()
{
	if (keypoints[0].pt.y < (keypoints[ile / 4 - 1].pt.y - 14) || keypoints[0].pt.y
>(keypoints[ile / 4 - 1].pt.y + 6))
		return 0;
	if (keypoints[0].pt.x < (keypoints[3 * ile / 4].pt.x - 8) || keypoints[0].pt.x >
(keypoints[3 * ile / 4].pt.x + 8))
		return 0;
	return 1;
}
```

Funkcja sprawdzająca czy plansza jest ułożona poziomo. Brane są pionki z przeciwległych krańców planszy i porównywane są ich współrzędne. Jeśli różnice są zbyt duże to zwracany jest błąd.

```
void errokno(int nr)
{
	while (1)
	{
		char exit = 0;
		namedWindow("Error", CV_WINDOW_AUTOSIZE);
		okienko = Mat3b(200, 300, Vec3b(0, 0, 0));
		putText(okienko, "ENTER aby kontynuowac", Point(okienko.cols / 2 - 100,
okienko.rows - 20), FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
		if (nr == 1)
		{
			putText(okienko, "Zaznacz cale", Point(40, 30),
FONT_HERSHEY_PLAIN, 2, Scalar(255, 255, 255));
			putText(okienko, "pole gry", Point(70, 70), FONT_HERSHEY_PLAIN, 2,
Scalar(255, 255, 255));
		}
		if(nr == 2)
		{
			putText(okienko, "Dobierz poprawnie selekcje", Point(40, 30),
FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
			putText(okienko, "Ustaw plansze poziomo", Point(60, 70),
FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
```

```
                    putText(okienko, "Oswietl plansze rownomiernie", Point(30, 110),
FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
             }
             line(okienko, Point(0, okienko.rows - 50), Point(okienko.cols,
okienko.rows - 50), Scalar(255, 255, 255), 1, 8, 0);
             imshow("Error", okienko);
             doit = 0;
             exit = waitKey(1);
             if (exit == 13)
             {
                    destroyWindow("Error");
                    break;
             }
      }
}
```

Funkcja, która odpowiada za wyświetlanie okien z komunikatami o błędach.

```
int WinMain()
{

      VideoCapture capWebcam(0);
      if (capWebcam.isOpened() == false)
      {
             std::cout << "error: Webcam connect unsuccessful\n";
             return(0);
      }
      capWebcam.set(CV_CAP_PROP_FRAME_WIDTH, 640);
      capWebcam.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
      while (charCheckForEscKey != 27 && capWebcam.isOpened()) {
             bool blnFrameReadSuccessfully;
             Mat poczatek;
             while (czekaj != 13)
             {
                    capWebcam.read(imgOrigin);
                    poczatek = imgOrigin;
                    line(poczatek, Point(0, poczatek.size().height / 3),
Point(poczatek.size().width, poczatek.size().height / 3), Scalar(255, 255, 255), 2, 8,
0);
                    line(poczatek, Point(0, 2 * poczatek.size().height / 3),
Point(poczatek.size().width, 2 * poczatek.size().height / 3), Scalar(255, 255, 255),
2, 8, 0);
                    rectangle(poczatek, Point(0, poczatek.rows - 30),
Point(poczatek.cols, poczatek.rows), Scalar(0, 0, 0), -1, 8, 0);
                    putText(poczatek, "Po ustawieniu zatwierdz ENTER",
Point(poczatek.cols / 2 - poczatek.cols / 4,poczatek.rows-10), FONT_HERSHEY_PLAIN, 1,
Scalar(255, 255, 255));
                    rectangle(poczatek, Point(0,0), Point(poczatek.cols, 30),
Scalar(0, 0, 0), -1, 8, 0);
                    putText(poczatek, "Ustaw plansze rownolegle do linni, z szyfrem z
lewej strony", Point(40, 20), FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
                    namedWindow("wstep", CV_WINDOW_AUTOSIZE);
                    imshow("wstep", poczatek);
                    czekaj = waitKey(500);
             }
             destroyWindow("wstep");
             if (lowVP != lowV || pom == 0 || doit == 1)
                    blnFrameReadSuccessfully = capWebcam.read(imgOrigin);
             else
                    blnFrameReadSuccessfully = capWebcam.read(imgOrigintmp);
             if (!blnFrameReadSuccessfully || imgOrigin.empty()) {
                    cout << "error: frame can't read \n";
```

```
                        break;
                }
                imgOriginal = imgOrigin;
                if (pom == 0 || zaz_pon == 1)
                {
                        zaz_pon = 0;
                        int szer = 0;
                        int wys = 0;
                        while ((szer < (25*imgOrigin.cols/32) || wys <
(imgOrigin.rows/4)))
                        {
                                rectangle(imgOrigin, Point(0, poczatek.rows - 30),
Point(poczatek.cols, poczatek.rows), Scalar(0, 0, 0), -1, 8, 0);
                                putText(imgOrigin, "Po zaznaczeniu zatwierdz ENTER",
Point(poczatek.cols / 2 - poczatek.cols / 4, poczatek.rows - 10), FONT_HERSHEY_PLAIN,
1, Scalar(255, 255, 255));
                                rectangle(imgOrigin, Point(0, 0), Point(poczatek.cols, 30),
Scalar(0, 0, 0), -1, 8, 0);
                                putText(imgOrigin, "Zaznacz obszar gry (4x13 otworow)",
Point(150, 20), FONT_HERSHEY_PLAIN, 1, Scalar(255, 255, 255));
                                r = selectROI(imgOrigin);
                                destroyWindow("ROI selector");
                                imgOriginal = imgOrigin(r);
                                szer = imgOriginal.size().width;
                                wys = imgOriginal.size().height;
                                if (szer < (25 * imgOrigin.cols / 32) || wys <
(imgOrigin.rows / 4))
                                {
                                        errokno(1);
                                        capWebcam.read(imgOrigin);
                                }
                                pom = 1;
                        }
                }
                else
                        imgOriginal = imgOrigin(r);
                if (lowVP != lowV || doit == 1)
                {
                        imgtransf();
                        blobcreator();
                        imgOriginal1 = imgOriginal.clone();
                        if (doit == 1)
                        {
                                background = pombg.clone();
                                ile = keypoints.size();
                                cout << ile << endl;
                                if (ile % 4 != 0 || ile < 4)
                                {
                                        errokno(2);
                                        goto blad;
                                }
                                sortowanie();
                                if (test() == 0)
                                {
                                        errokno(2);
                                        cout << "Dobierz poprawnie selekcje i ustaw plansze
poziomo  \n";
                                        goto blad;
                                }
                                tablica();
                                nowa = 0;
                                if (proba >= rzad - 1) proba = rzad - 1;
```

```
                        if (numer >= 3) numer = 3;
                    /* if (proba != 0)
                        {
                                cout << "H: " << (int)pionki[proba][numer].H << "  "
<< "S: " << (int)pionki[proba][numer].S << "  " << "V: " <<
(int)pionki[proba][numer].V << "  " << "HS: " << (int)pionki[proba][numer].sHS << "  "
<< "X: " << pionki[proba][numer].x << "  " << "Y: " << pionki[proba][numer].y << endl;
                                circle(imgOriginal1, pionki[proba][numer].kord, 20,
Scalar((int)pionki[proba][numer].B, (int)pionki[proba][numer].G,
(int)pionki[proba][numer].R), 10, 8, 0);
                        }
                        else
                        {
                                cout << "H: " << (int)szyfr[numer].H << "  " << "S:
" << (int)szyfr[numer].S << "  " << "V: " << (int)szyfr[numer].V << "  " << "HS: " <<
(int)szyfr[numer].sHS << "  " << "X: " << szyfr[numer].x << "  " << "Y: " <<
szyfr[numer].y << endl;
                                circle(imgOriginal1, szyfr[numer].kord, 20,
Scalar((int)szyfr[numer].B, (int)szyfr[numer].G, (int)szyfr[numer].R), 10, 8, 0);
                        }*/
                        klucz();
                        rysuj();
                        lowVP = lowV;
                        numer1 = numer;
                        proba1 = proba;
                        for (int i = 0; i < rzad; i++)
                                delete[] pionki[i];
                        delete[] pionki;
                        for (int i = 0; i < rzad; i++)
                                delete[] key[i];
                        delete[] key;
                        doit = 0;
                }
                okienka();


        }
        blad:
        doit = 0;
        charCheckForEscKey = waitKey(1);
        if (zaz_pon == 1)
                destroyAllWindows();
    }
    return(0);
}
```

Funkcja main najpierw otwiera obraz video z kamery i sprawdza czy poprawnie został wczytany i dostosowuje go do przewidzianych przez program rozmiarów. Następnie w dużej pętli wykonywane są wszystkie czynności – każde wykonanie pętli to jedno odświeżenie programu (wszystkich okien). W pętli tej najpierw wyświetlane są początkowe okna, w których należy określić ustawienie i obszar ROI, następnie sprawdzana jest poprawność ustawienia ROI, w razie błędu wywoływane jest okno z błędem. Następnie wywoływane są po kolei wszystkie omówione wyżej funkcje, jednakże tylko wtedy gdy użyty został suwak Check. Na końcu zwalniana jest pamięć zajęta przez tablice dynamiczne. Część w komentarzu pozostała celowo, ponieważ dzięki niej można wyświetlić poszczególne odczytane kolory na oryginalnym zdjęciu z kamery jak również wypisać dane dotyczące danego koloru (składowe jak również współrzędne punktów). Ta część była wykorzystana do

dobierania parametrów w funkcji klucz(). Działanie programu może zostać zakończone przez naciśnięcie przycisku ESC.

6. Dodatkowe materiały

W skład projektu wchodzą pliki programowe, niniejsza dokumentacja, plik z wyeksportowanym kodem (pdf) oraz filmik demonstrujący działanie programu.