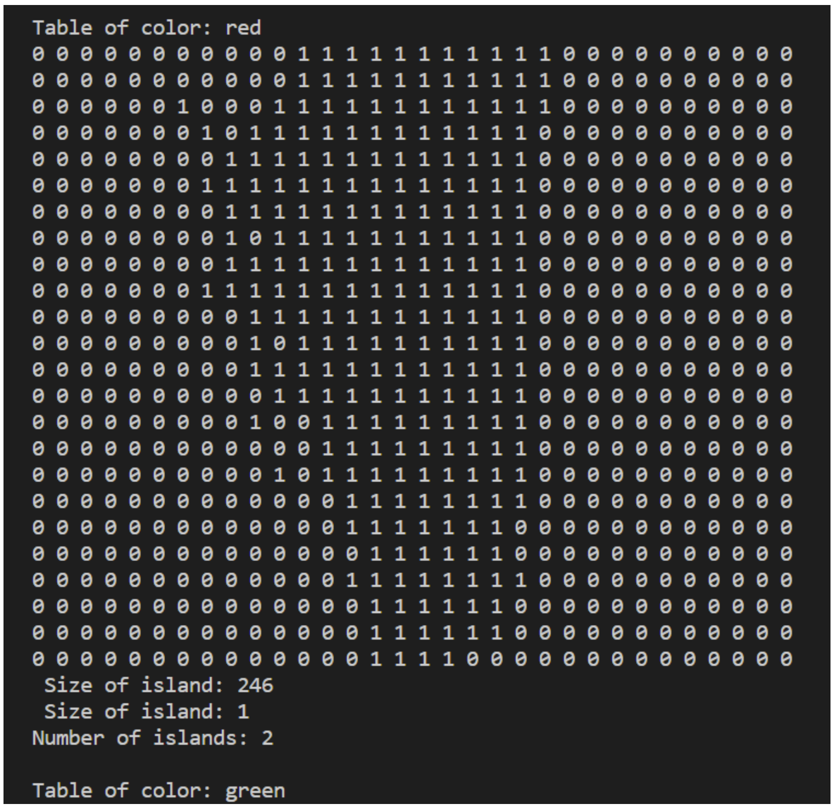
Andrzej Cieślik 03.02.2023

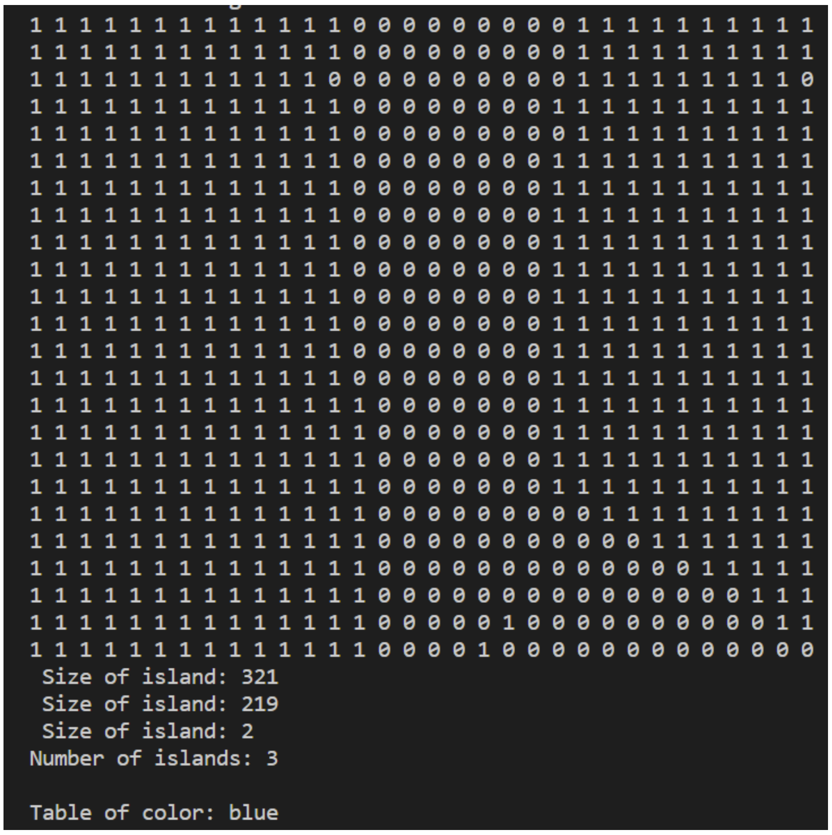
Laboratorium: obliczanie maksymalnej powierzchni wg kolorów z obrazu kamery ESP32-CAM

1. **Widok z konsoli:**



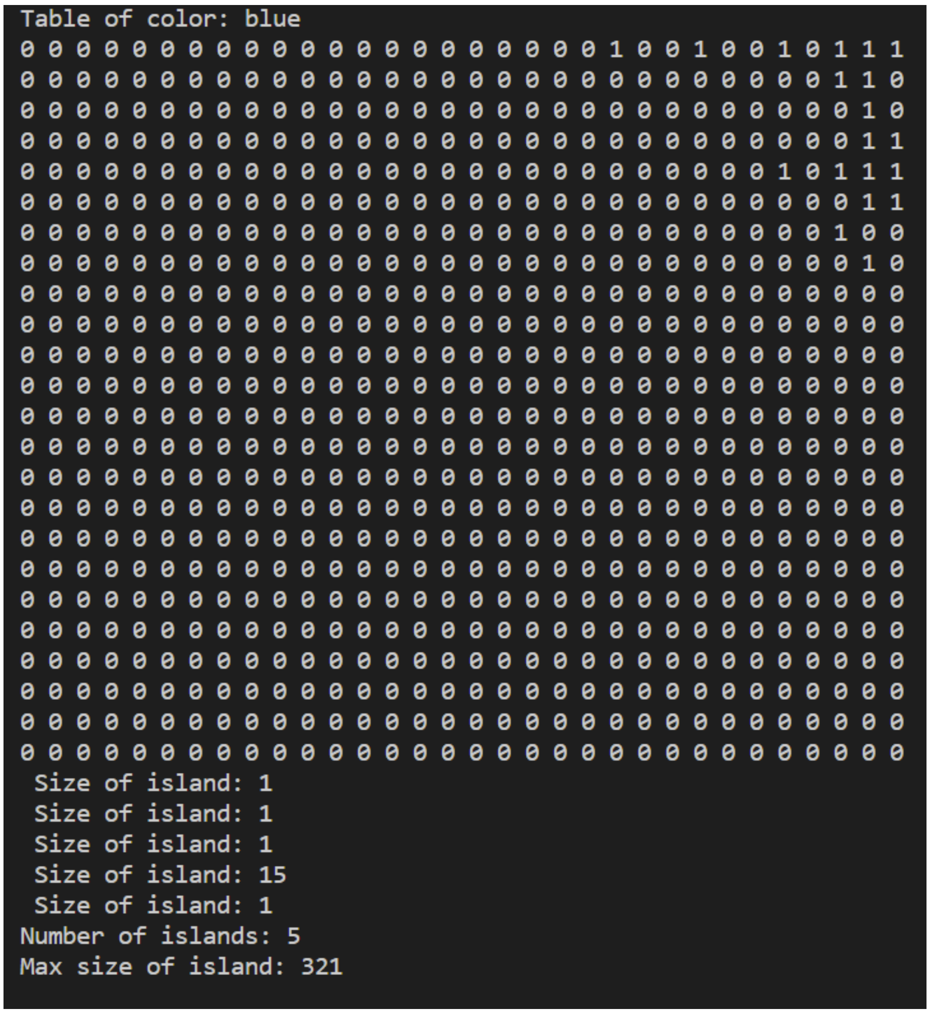
```
Table of color: red
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Size of island: 246
 Size of island: 1
Number of islands: 2

Table of color: green
```



```
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 Size of island: 321
 Size of island: 219
 Size of island: 2
Number of islands: 3

Table of color: blue
```



```
Table of color: blue
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Size of island: 1
 Size of island: 1
 Size of island: 1
 Size of island: 15
 Size of island: 1
Number of islands: 5
Max size of island: 321
```

2. Kod:

```cpp
#include <WiFiClient.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <esp_camera.h>
#include <Arduino.h>
#include <esp_timer.h>
#include <FS.h>
#include "ESPAsyncWebServer.h"

//for find the islands in table
#include <cstdint>
#include <iostream>
#include <vector>
#include <queue>
#include <cstring>
using namespace std;

#define CAMERA_MODEL_AI_THINKER
#define FRAME_SIZE FRAMESIZE_QQVGA
#define SOURCE_WIDTH 160
#define SOURCE_HEIGHT 120
#define BLOCK_SIZE 5
#define DEST_WIDTH (SOURCE_WIDTH / BLOCK_SIZE)
#define DEST_HEIGHT (SOURCE_HEIGHT / BLOCK_SIZE)

```

```
int max_island = 0;

// Below arrays detail all eight possible movements from a cell
// (top, right, bottom, left, and four diagonal moves)
//sprawdzanie kratek dookola kratki - dla algorytmu BFS
int row[] = { -1, -1, -1, 0, 1, 0, 1, 1 };
int col[] = { -1, 1, 0, -1, -1, 1, 0, 1 };

//for parameter in function countIslands()
const int HEIGHT = DEST_HEIGHT;
const int WIDTH = DEST_WIDTH;

const char* PARAM_INPUT_1 = "input1";
const char* PARAM_INPUT_2 = "offset";

String inputMessage;
int prog=128;
uint16_t rgb_frame[DEST_HEIGHT][DEST_WIDTH][3] = { 0 };
uint16_t frame[DEST_HEIGHT][DEST_WIDTH] = { 0 };          //ramka dodana dla analizy kolorow

uint16_t redFrame[DEST_HEIGHT][DEST_WIDTH] = { 0 };
uint16_t greenFrame[DEST_HEIGHT][DEST_WIDTH] = { 0 };
uint16_t blueFrame[DEST_HEIGHT][DEST_WIDTH] = { 0 };

int offset=20;
#include "camera_pins.h"

#include <SD.h>
#include <SPIFFS.h>
#define DIODA 33
#define CAMERA_MODEL_AI_THINKER     //wybor modelu kamery

#include "camera_pins.h"

const char* ssid = "Q6_2862";
const char* password = "1278MartaStyle+x12";

AsyncWebServer server(80);  //uzycie serwera asynchronicznego http na porcie 80

//prosta strona www z miejscem na obraz z kamery
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <h2>ESP Image Web Server</h2>
    <form action="/get">
    prog: <input type="text" name="input1">
    <input type="submit" value="Submit">
  </form><br>
      <form action="/get">
    offset: <input type="text" name="offset">
    <input type="submit" value="Submit">
  </form><br>
  </body>
</html>)rawliteral";

void grab_image(uint8_t *source, int len) {
     for (int y=0;y<DEST_HEIGHT;y++)
       {
         for (int x=0;x<DEST_WIDTH;x++)
          {
            rgb_frame[y][x][0]=0;
            rgb_frame[y][x][1]=0;
```

```
           rgb_frame[y][x][2]=0;

         }
      }
   for (size_t i = 0; i < len; i += 2)
   {
   const uint8_t high = source[i];
   const uint8_t low  = source[i+1];
   const uint16_t pixel = (high << 8) | low;

   const uint8_t r = (pixel & 0b1111100000000000) >> 11;
   const uint8_t g = (pixel & 0b0000011111100000) >> 6;
   const uint8_t b = (pixel & 0b0000000000011111);
       const size_t j = i / 2;
       const uint16_t x = j % SOURCE_WIDTH;
       const uint16_t y = floor(j / SOURCE_WIDTH);
       const uint8_t block_x = floor(x / BLOCK_SIZE);
       const uint8_t block_y = floor(y / BLOCK_SIZE);
       rgb_frame[block_y][block_x][0] += r;
       rgb_frame[block_y][block_x][1] += g;
       rgb_frame[block_y][block_x][2] += b;
   }
  }

void fotka()   //funkcja do wykonania zdjecia
{

   camera_fb_t * fb = NULL;
   fb = esp_camera_fb_get();  //uruchomienie kamery
   if (!fb) {
       Serial.println("Camera capture failed");
    }
      grab_image(fb->buf,fb->len);
      //Serial.printf("%d %d \n",DEST_HEIGHT,DEST_WIDTH);
      for (int y=0;y<DEST_HEIGHT;y++)
      {
        for (int x=0;x<DEST_WIDTH;x++)
         {
             if ((rgb_frame[y][x][0]>(rgb_frame[y][x][1]+offset))&&(rgb_frame[y][x][0]>(rgb_frame[y][x][2]+offset))&&(rgb_frame[y][x][0]>prog))
             {
               //Serial.print("CC");
               frame[y][x] = 'C';
             }
             else if ((rgb_frame[y][x][1]>(rgb_frame[y][x][0]+offset))&&(rgb_frame[y][x][1]>(rgb_frame[y][x][2]+offset))&&(rgb_frame[y][x][1]>prog))
             {
              //Serial.print("ZZ");
              frame[y][x] = 'Z';
             }
             else if ((rgb_frame[y][x][2]>(rgb_frame[y][x][0]+offset))&&(rgb_frame[y][x][2]>(rgb_frame[y][x][1]+offset))&&(rgb_frame[y][x][2]>prog))
             {
              //Serial.print("NN");
              frame[y][x] = 'N';
             }
             //else  Serial.print(" ");
         }
         Serial.printf("\n");
      }
     Serial.printf("\n");
     Serial.printf("\n");
     Serial.printf("\n");
     Serial.printf("\n");
   esp_camera_fb_return(fb);
}

void generateFramesByColor(){            //wypełnij 3 tablice wartościami dla odpowiednich kolorów
 for (int y=0;y<DEST_HEIGHT;y++)
      {
        for (int x=0;x<DEST_WIDTH;x++)
         {
               if(frame[y][x] == 'C')
               {
                 redFrame[y][x] = 1;
               }
               if(frame[y][x] == 'Z')
               {
                 greenFrame[y][x] = 1;
               }
               if(frame[y][x] == 'N')
               {
                 blueFrame[y][x] = 1;
               }
               //Serial.print(redFrame[y][x]);

         }
         //Serial.printf("\n");
      }
}

bool isSafe(vector<vector<int>> const &mat, int x, int y,
        vector<vector<bool>> const &processed)
{
    return (x >= 0 && x < mat.size()) && (y >= 0 && y < mat[0].size()) &&
        mat[x][y] && !processed[x][y];
}

void BFS(vector<vector<int>> const &mat, vector<vector<bool>> &processed, int i, int j) //zastosowano algorytm BFS (przeszukiwanie wszerz)
{
    int size_of_island = 1;

    // create an empty queue and enqueue source node
    queue<pair<int, int>> q;
    q.push(make_pair(i, j));

    // mark source node as processed
    processed[i][j] = true;

    // loop till queue is empty

    while (!q.empty())
    {
        // dequeue front node and process it
        int x = q.front().first;
        int y = q.front().second;
        q.pop();

        // check for all eight possible movements from the current cell
        // and enqueue each valid movement

        int b =0;
        for (int k = 0; k < 8; k++)
        {
            // skip if the location is invalid, or already
```

```cpp
            // processed, or is 0
            if (isSafe(mat, x + row[k], y + col[k], processed))
            {
                size_of_island++;
                // mark it as processed and enqueue it
                processed[x + row[k]][y + col[k]] = 1;
                q.push(make_pair(x + row[k], y + col[k]));

            }
        }
    }
    Serial.print(" Size of island: ");
    Serial.print(size_of_island);
    Serial.println("");

    if(size_of_island > max_island)     //jak wyspa jest wieksza niż obecnie najwieksza to zastap
    {
      max_island = size_of_island;
    }
}

int countIslands(vector<vector<int>> const &mat)      //liczenie ilości wysp dla tablicy wektorowej 2D
{
    // base case
    if (mat.size() == 0) {
        return 0;
    }

    // `M × N` matrix
    int M = mat.size();
    int N = mat[0].size();

    // stores if a cell is processed or not
    vector<vector<bool>> processed(M, vector<bool>(N));

    int island = 0;
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            // start BFS from each unprocessed node and increment island count
            if (mat[i][j] && processed[i][j] == 0)
            {
                BFS(mat, processed, i, j);
                island++;
            }
        }
    }

    return island;
}

void count(uint16_t table[][DEST_WIDTH], int height, int width, string name)
{
Serial.println("");
cout << "Table of color: " << name << endl;

vector<vector<int> > vec;
vector<int> vectorRows;
    for (int y=0;y<DEST_HEIGHT;y++)
    {
        //jeden rząd z tablicy redFrame do vectora v1
        for (int x=0;x<DEST_WIDTH;x++)
        {
          vectorRows.push_back(table[y][x]);       //np. {0,0,1,1,1,1,0,0} dla wiersza y = 0, wrzuc do vec, potem  dla wiersza y = 1 , 2..
        }
        //vector rzędu do vectora 2D
        vec.push_back(vectorRows);
        vectorRows = {}; //wyczyść wiersz po wpisaniu
    }

    for (int i = 0; i < vec.size(); i++)
    {
        for (int j = 0; j < vec[i].size(); j++)
            cout << vec[i][j] << " ";
        cout << endl;
    }
    cout << "Number of islands: " << countIslands(vec) << endl;
}

void findMaxIsland()                      //wyświetl najwiekszy rozmiar wysp z wszystkich wysp i skasuj wartość dla przyszłych przejść pętli loop()
{
    Serial.print("Max size of island: ");
    Serial.print(max_island);
    max_island = 0;
}

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0; //definicja portow, do ktorych podlaczona jest kamera
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_RGB565;
  config.frame_size = FRAME_SIZE;
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);   //inicjacja kamery
  if (err != ESP_OK) {
    Serial.printf("B��d inicjacji kamery numer: 0x%x", err);
    return;
  }
  Serial.printf("kamera ok");

  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAME_SIZE);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Po��czono z WIFI");

  Serial.print("Kamera gotowa wejd� na adres: 'http://");
  Serial.println(WiFi.localIP());

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });

 server.on("/fotka", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send(SPIFFS, "/photo.jpg", "image/jpg");
  });
  server.begin();

  server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputParam;
    // GET input1 value on <ESP_IP>/get?input1=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
      inputMessage = request->getParam(PARAM_INPUT_1)->value();
      inputParam = PARAM_INPUT_1;
      prog=inputMessage.toInt();
    }
    request->send(200, "text/html", "HTTP GET request sent to your ESP on input field ("
                                     + inputParam + ") with value: " + inputMessage +
                                     "<br><a href=\"/\">Return to Home Page</a>");

     if (request->hasParam(PARAM_INPUT_2)) {
      inputMessage = request->getParam(PARAM_INPUT_2)->value();
      inputParam = PARAM_INPUT_2;
      offset=inputMessage.toInt();
    }
    request->send(200, "text/html", "HTTP GET request sent to your ESP on input field ("
                                     + inputParam + ") with value: " + inputMessage +
                                     "<br><a href=\"/\">Return to Home Page</a>");
    });
}

void loop() {
  delay(10000);
  fotka();
  generateFramesByColor();
  count(redFrame, DEST_HEIGHT,DEST_WIDTH, "red");
  count(greenFrame, DEST_HEIGHT,DEST_WIDTH, "green");
  count(blueFrame, DEST_HEIGHT,DEST_WIDTH, "blue");
  findMaxIsland();
}
```