

# Przestrzenie wizualne

projekt 1

Kamil Woźniak  
Wiktoria Opalińska  
Aleksandra Kędziora

# 1. Cel projektu

Celem projektu jest stworzenie interaktywnego systemu, który na podstawie wykrywania markerów AR (hiro, kanji) prezentuje użytkownikowi określony stan emocjonalny. System ma potencjał do pogłębiania doświadczenia emocjonalnego odbiorcy – użytkownik może doświadczyć stanów takich jak radość, miłość, smutek czy niepokój.

Nasz projekt odnosi się do dwóch kluczowych rodzajów wzmocnień, a jego głównym zadaniem jest pogłębienie wybranego stanu emocjonalnego:

- Wzmocnienie afektywne – celem jest nie tylko regulacja emocji, ale również pogłębienie i utrwalenie określonego stanu emocjonalnego. Dzięki odpowiednio dobranym elementom dźwiękowym i wizualnym, system pomaga zanurzyć się w doświadczeniu emocjonalnym, potęgując uczucia zarówno pozytywne, jak i te trudniejsze, w zależności od założonego efektu.
- Estetyczne wzmocnienie - polega na zastosowaniu filtrów i ikon, które wzmacniają dany stan emocjonalny oraz modyfikują sposób postrzegania obrazu. Na przykład filtr odpowiedni dla danego stanu pomaga użytkownikowi bardziej intensywnie przeżywać emocje, łącząc aspekty afektywne z percepcyjnymi. Dzięki temu wszystkie elementy systemu współpracują, tworząc spójną i artystyczną przestrzeń, która umożliwia autentyczne doświadczenie emocjonalne.

## Technologie i biblioteki

- **NyARToolkit for Processing**
- **OpenCV for processing**
- **Biblioteka Sound**

## Detekcja markerów AR

- System wykorzystuje ARToolKit do wykrywania markerów typu hiro i kanji.
- W zależności od wykrytego markera, system interpretuje kontekst emocjonalny – różnicując stany pozytywne i negatywne.

Wykorzystane markery:



index	0	1
pozytywne	miłość	radość
negatywne	niepokój	smutek

## 2. Podział pracy

### Zespół Projektowy

#### Wiktoria Opalińska - Team Leader

- Implementacja biblioteki dźwięku
- Implementacja biblioteki GUI
- Integracja wszystkich modułów projektu

#### Aleksandra Kędziora

- Stworzenie dokumentacji projektu
- Zaprojektowanie ikon

#### Kamil Woźniak

- Opracowanie i implementacja filtrów wizualnych

## 3. Biblioteki

### Użyte biblioteki

- ControlP5
- Nyar4PSG
- Video Library for Processing 3 2.0
- Sound 2.4.0
- OpenCV for Processing 0.7.0

## 4. Interface użytkownika - GUI

Moduł GUI wykorzystuje bibliotekę **ControlP5** dla Processing, która umożliwia tworzenie elementów interfejsu graficznego.

Dodawane są dwa przyciski:

- Stan pozytywny – ustawiony z przesunięciem w lewo od środka.
- Stan negatywny – ustawiony z przesunięciem w prawo od środka.

Dla obu przycisków ustawiana jest domyślna kolorystyka (czarne tło i biały tekst) oraz czcionka

```
1 void GUI_setup(){
2   cp5 = new ControlP5(this);
3
4   // tworzenie przycisków oraz pozycje startową
5   int buttonWidth1 = 150;
6   int buttonHeight = 35;
7   int startX = width/2;
8   int startY = height - buttonHeight - 20;
9
10
11  // Dodajemy pierwszy przycisk "Stan pozytywny"
12  b1 = (Button) cp5.addButton("Stan pozytywny")
13      .setPosition(startX - 180, startY)
14      .setSize(buttonWidth1, buttonHeight)
15      .setColorBackground(color(0))
16      .setColorForeground(color(255));
17
18  // Ustawiamy czcionkę napisu na przycisku "Stan pozytywny"
19  b1.getCaptionLabel().setFont(createFont("Arial", 14));
20
21  // Dodajemy drugi przycisk "Stan negatywny"
22  b2 = (Button) cp5.addButton("Stan negatywny")
23      .setPosition(startX + 30, startY)
24      .setSize(buttonWidth1, buttonHeight)
25      .setColorBackground(color(0))
26      .setColorForeground(color(255));
27
28  // Ustawiamy czcionkę napisu na przycisku "Stan negatywny"
29  b2.getCaptionLabel().setFont(createFont("Arial", 14));
30
31 }
```

Dodatkowo, kolory przycisku, który został kliknięty, są zmieniane, aby wizualnie zaznaczyć, że jest aktywny.

```
32 // Funkcja obsługująca kliknięcia przycisków
33 void controlEvent(ControlEvent event) {
34     // Pobieramy nazwę przycisku, który wywołał zdarzenie
35     String buttonName = event.getController().getName();
36
37     b1.setColorBackground(color(0));
38     b1.setColorForeground(color(255));
39     b2.setColorBackground(color(0));
40     b2.setColorForeground(color(255));
41
42     // Sprawdzamy, który przycisk został kliknięty:
43     if (buttonName.equals("Stan pozytywny")) {
44         icon_set = 1;
45         // Zmieniamy kolory przycisku, aby wskazać, że jest aktywny:
46         b1.setColorBackground(color(150));
47         b1.setColorForeground(color(0));
48     } else if (buttonName.equals("Stan negatywny")) {
49         icon_set = 2;
50         b2.setColorBackground(color(150));
51         b2.setColorForeground(color(0));
52     }
53 }
```

## 5. Ikonki - icons

Do wykrywania markerów wykorzystano bibliotekę ARToolKit (konkretnie klasę **MultiMarker**), która umożliwia inicjalizację kamery oraz dodawanie markerów na podstawie plików wzorcowych (np. "patt.hiro" i "patt.kanji").

W module ikon zastosowano następujące funkcje:

- **drawMilosc()** – generuje kształt serca 2D - przy użyciu funkcji beginShape(), vertex() i bezierVertex().
- **drawSmutek()** – tworzy ikonę smutku przypominającą chmurkę (wykorzystanie funkcji sphere() nakładających się na siebie).
- **drawRadosc()** – rysuje słońce przy użyciu sphere() oraz tworzy promienie poprzez dodanie obręczy z obróconych prostokątów (rotateZ() i box()),
- **drawNiepokoj()** – tworzy nieregularną kompozycję brył (box()) z rotacjami i przesunięciami, co wizualnie oddaje stan niepokoju.

## Ikonka serca - miłość

- **scale(1, -1)** - skaluje układ współrzędnych, odwracając go w osi Y, by dopasować orientację figury do sceny AR.
- **beginShape(); ... endShape(CLOSE)** - rozpoczyna i kończy definiowanie kształtu niestandardowego (Shape).
- **vertex(0, -20);** – Pierwszy punkt kształtu.
- **bezierVertex(...);** – Dodaje zakrzywione segmenty (krzywe Beziera). Rysuje górną, zaokrągloną część serca.
- **CLOSE w endShape** zamyka kształt, łącząc ostatni punkt z pierwszym.

```
1 void icons_setup() {
2
3   println(MultiMarker.VERSION);
4   cam = new Capture(this, 640, 480);
5   nya = new MultiMarker(this, width, height, "camera_para.dat", NyAR4PsgConfig.CONFIG_PSG);
6   nya.addARMarker("patt.hiro", 80); // id=0
7   nya.addARMarker("patt.kanji", 80); // id=1
8   nya.addARMarker("h.patt", 80); // id=2
9   nya.addARMarker("a.patt", 80); // id=3
10  cam.start();
11 }
12 // rysowanie serca
13 void drawMilosc() {
14   scale(1, -1);
15   fill(255, 0, 0);
16   noStroke();
17   beginShape();
18   vertex(0, -20);
19   bezierVertex(-25, -40, -50, 0, 0, 40);
20   bezierVertex(60, 0, 25, -40, 0, -20);
21   endShape(CLOSE);
22 }
```

## Ikonka chmury - smutek

Główna sfera:

- **sphereDetail(16)** - Określa szczegółowość sfer rysowanych przez funkcję **sphere()**. Większa liczba oznacza więcej wielokątów i gładszą powierzchnię.

Pierwszy blok mniejszych sfer:

- **pushMatrix();** – Zapisuje aktualną macierz transformacji. Pozwala to na tymczasowe przesunięcia, obroty itp. bez wpływu na kolejne elementy.
- **translate(25, 0, 0);** – Przesuwa układ współrzędnych o 25 w osi X (w prawo), 0 w osi Y i 0 w osi Z. Dzięki temu sfera pojawi się obok głównej.

```

24 // rysowanie chmurki
25 void drawSmutek() {
26     stroke(63, 67, 69);
27     fill(54, 55, 56);
28     sphereDetail(16);
29
30     // Główna (największa) sfera
31     pushMatrix();
32     sphere(40);
33     popMatrix();
34
35     // Kilka mniejszych sfer, przesuniętych względem głównej
36     pushMatrix();
37     translate(-25, 0, 10);
38     sphere(30);
39     popMatrix();
40
41     pushMatrix();
42     translate(25, 0, -10);
43     sphere(30);
44     popMatrix();
45
46     pushMatrix();
47     translate(10, -15, 0);
48     sphere(20);
49     popMatrix();
50
51     pushMatrix();
52     translate(-10, -15, 0);
53     sphere(20);
54     popMatrix();
55 }

```

## Ikona słońca -radość

```
56 // rysowanie słońca
57 void drawRadosc() {
58     fill(255, 204, 0);
59     noStroke();
60     sphereDetail(24);
61     sphere(25);
62
63     //promienie
64     int numRays = 12; // liczba promieni
65     for (int i = 0; i < numRays; i++) {
66         pushMatrix();
67         // obrot promieni
68         rotateZ(TWO_PI * i / numRays);
69
70         // przesuwamy promienie
71         translate(30, 0, 0);
72
73         fill(247, 247, 161);
74
75         // promienie jako box
76         box(8, 40, 8);
77         popMatrix();
78     }
79 }
```



## Ikonka box - niepokój

```
80 // box złożony z boxów
81 void drawNiepokoj() {
82     stroke(0, 23, 59);
83
84     // główny box
85     fill(2, 62, 158);
86     box(60);
87
88
89     // Box doł
90     pushMatrix();
91     translate(0, -20, 0);
92     rotateX(PI/4);
93     fill(0, 68, 255);
94     box(50);
95     popMatrix();
96
97     // Box 2 lewo góra
98     pushMatrix();
99     translate(20, 20, 0);
100     rotateY(PI/3);
101     fill(0, 68, 255);
102     box(40);
103     popMatrix();
104
105     // Box 3: prawo góra
106     pushMatrix();
107     translate(-20, 20, 0);
108     rotateZ(PI/6);
109     fill(0, 68, 255);
110     box(40);
111     popMatrix();
112
113     // Box 4:
114     pushMatrix();
115     translate(0, 0, 20);
116     rotateX(PI/6);
117     fill(0, 68, 255);
118     box(40);
119     popMatrix();
120 }
```

## 6. Muzyka i efekty dźwiękowe - music, sound

Projekt zawiera dwa podejścia do generowania efektów dźwiękowych – jeden oparty na odtwarzaniu wcześniej nagranych plików dźwiękowych, a drugi na generowaniu dźwięków w czasie rzeczywistym przy użyciu oscylatorów i szumu.

<https://www.youtube.com/playlist?list=PLRvIwc3GKqTbgOD8vBYXvTSQZHy2to8Sr> - film z działającym dźwiękiem

### 1. Odtwarzanie dźwięków (SoundFile) - music

```
1 import processing.sound.*;
2
3 SoundFile loveSound, sunshineSound, anxietySound, sadnessSound;
4
5
6 // ładuje pliki dźwiękowe (np. "love.mp3", "sunshine.mp3", "anxiety.wav", "sadness.mp3")
7 void music_setup() {
8     loveSound = new SoundFile(this, "love.mp3");
9     sunshineSound = new SoundFile(this, "sunshine.mp3");
10    anxietySound = new SoundFile(this, "anxiety.wav");
11    sadnessSound = new SoundFile(this, "sadness.mp3");
12 }
13
14 void playEffectMusic(int iconSet, int marker) {
15     stopAllSounds(iconSet, marker); // zatrzymanie dźwięków, aby nie zaszło nakładanie się na siebie dźwięku
16
17     if (iconSet == 1 && marker == 0) {
18         if (!loveSound.isPlaying()){
19             loveSound.loop();
20         }
21     } else if (iconSet == 1 && marker == 1) {
22         if (!sunshineSound.isPlaying()){
23             sunshineSound.loop();
24         }
25     } else if (iconSet == 2 && marker == 0) {
26         if (!anxietySound.isPlaying()){
27             anxietySound.loop();
28         }
29     } else if (iconSet == 2 && marker == 1) {
30         if (!sadnessSound.isPlaying()){
31             sadnessSound.loop();
32         }
33     }
34 }
```

```

36 // przy aktywacji jednego dźwięku zatrzymywany jest dźwięk przeciwnego stanu
37 void stopAllSounds(int iconSet, int marker) {
38
39     if (iconSet == 1){
40         anxietySound.stop();
41         sadnessSound.stop();
42         if (marker == 0){
43             sunshineSound.stop();
44         } else if (marker == 1){
45             loveSound.stop();
46         }
47     } else if (iconSet == 2){
48         loveSound.stop();
49         sunshineSound.stop();
50         if (marker == 0){
51             sadnessSound.stop();
52         } else if (marker == 1) {
53             anxietySound.stop();
54         }
55     }
56 }
57

```

## 2. Generowanie dźwięków w czasie rzeczywistym (Oscylatory i White Noise) - sound

W drugiej części kodu ponownie importujemy `processing.sound.*`, tym razem w celu stworzenia efektów dźwiękowych za pomocą generatorów dźwięku:

### **SinOsc (sinusoidalne oscylatory):**

Obiekty **sunOsc**, **loveOsc** oraz **anxietyOsc** generują czyste tony, których częstotliwość i amplituda mogą być dynamicznie modyfikowane.

### **WhiteNoise:**

Obiekt **noise** generuje biały szum, który jest wykorzystywany jako efekt dźwiękowy dla stanu negatywnego (u nas dla niepokoju).

```

1 import processing.sound.*;
2 // Deklaracja oscylatorów (SinOsc) i generatora szumu (WhiteNoise)
3 SinOsc sunOsc, loveOsc, anxietyOsc;
4 WhiteNoise noise;
5
6 void sound_setup() {
7     //size(800, 600);
8     // Deklaracja oscylatorów (SinOsc) i generatora szumu (WhiteNoise)
9     sunOsc = new SinOsc(this);
10    loveOsc = new SinOsc(this);
11    anxietyOsc = new SinOsc(this);
12    noise = new WhiteNoise(this);
13 }

```

Funkcja odtwarza efekt dźwiękowy na podstawie wybranego stanu emocjonalnego (icon\_set) i markera (index) przy użyciu oscylatorów SinOsc oraz WhiteNoise z biblioteki processing.sound:

**Stan pozytywny (icon\_set == 1):**

- **Marker 1** : Odtwarzany jest sunOsc z losową częstotliwością, nadając dynamiczny, zmienny ton.
- **Marker 0**: Odtwarzany jest loveOsc ustawiony na 330 Hz, co kojarzy się z ciepłem miłości.

**Stan negatywny (icon\_set == 2):**

- **Marker 0**: Uruchamiany jest anxietyOsc z losową częstotliwością, tworząc niepokojący ton.
- **Marker 1**: Generowany jest biały szum.

```
14 // Funkcja odtwarzająca efekt dźwiękowy w zależności od stanu emocjonalnego (icon_set) i indeksu markera
15 void playEffectSound(int icon_set, int index) {
16     // Dla stanu pozytywnego (icon_set == 1)
17     if (icon_set == 1 && index == 1) {
18         sunOsc.freq(220 + random(10, 50)); // Ustawia losową częstotliwość w określonym zakresie
19         sunOsc.amp(0.3); // Ustawia głośność oscylatora
20         sunOsc.play(); // Odtwarza dźwięk
21         delay(500); // Czeka 500 ms
22         fadeOut(sunOsc); // Stopniowo wygasa dźwięk
23     }
24     else if (icon_set == 1 && index == 0) {
25         loveOsc.freq(330);
26         loveOsc.amp(0.5);
27         loveOsc.play();
28         delay(200);
29         fadeOut(loveOsc);
30     }
31     // Dla stanu negatywnego (icon_set == 2)
32     else if (icon_set == 2 && index == 0) {
33         anxietyOsc.freq(random(100, 200));
34         anxietyOsc.amp(0.5);
35         anxietyOsc.play();
36         delay(200);
37         fadeOut(anxietyOsc);
38     }
39
40     else if (icon_set == 2 && index == 1) {
41         noise.amp(0.2);
42         noise.play();
43         delay(500);
44         fadeOut(noise);
45     }
46 }
```

Po krótkim opóźnieniu dźwięk stopniowo zanika dzięki funkcji fadeOut.

```

47 // Funkcja wygaszania dźwięku dla oscylatora SinOsc
48 void fadeOut(SinOsc osc) {
49     for (float i = 0.2; i > 0; i -= 0.02) {
50         osc.amp(i);    // Stopniowe zmniejszanie amplitudy
51         delay(50);     // Opóźnienie dla efektu wygaszania
52     }
53     osc.stop();        // Zatrzymuje odtwarzanie po wygaszeniu
54 }
55 // Funkcja wygaszania dźwięku dla generatora WhiteNoise
56 void fadeOut(WhiteNoise noise) {
57     for (float i = 0.2; i > 0; i -= 0.02) {
58         noise.amp(i);
59         delay(50);
60     }
61     noise.stop();
62 }

```

\

## 7. Efekty wizualne - visual

W projekcie wykorzystane zostały również efekty wizualne, nakładane na cały obraz, co ma prowadzić do zwiększenia immersji użytkownika. Nadając jednocześnie scenie spójny i wyrazisty charakter dopasowany do stanu emocjonalnego.

Zmienne użyte w programie:

```
18 // Visual
19 Capture cam;
20 MultiMarker nya;
21 boolean markerDetected = false;
22
23 // Effects' parameters
24 float sunEffect = 0;
25 final float FADE_SPEED = 0.05;
26 final float WARMTH_INTENSITY = 1.5;
27 final float SUNLIGHT_GLOW = 1.8;
28
29 int cellSize = 20;
30 float loveGlow = 0;
31 float glowSpeed = 0.05;
32 ArrayList<Heart> hearts = new ArrayList<Heart>();
33
34 float bwEffect = 0;
35
36 int cols, rows;
37 float pulseIntensity = 0;
38 float pulseSpeed = 0.1;
39 boolean increasing = true;
40
41 class Heart {
42     float x, y;
43     float size;
44     float speed;
45     color heartColor;
46 }
```

```

46
47 Heart[] {
48     x = random(width);
49     y = height + 20;
50     size = random(15, 40);
51     speed = random(1, 3);
52     heartColor = color(
53         220 + random(35),
54         50 + random(100),
55         100 + random(100),
56         150 + random(105)
57     );
58 }
59
60 void update() {
61     y -= speed;
62     x += random(-0.5, 0.5);
63 }
64
65 void display() {
66     noStroke();
67     fill(heartColor);
68     beginShape();
69     vertex(x, y);
70     bezierVertex(x - size/2, y - size/2, x - size, y + size/3, x, y + size);
71     bezierVertex(x + size, y + size/3, x + size/2, y - size/2, x, y);
72     endShape();
73 }
74
75 boolean isOffScreen() {
76     return y < -size;
77 }
78 }
79 OpenCV opencv;
80 PImage edgeImage;
81

```

## Efekt słoneczny

Na cały ekran nakładany jest ciepły filtr, który nadaje obrazowi złote tony. W efekcie tym wykorzystywany jest bufor graficzny (PGraphics) do przetwarzania obrazu z kamery. Modyfikowane są piksele – wzmacniane są kanały czerwony i zielony, a dodawany jest złoty tint. Następnie, przy użyciu trybu mieszania SCREEN, rysowana jest elipsa symulująca promienie słoneczne, a na końcu nakładana jest delikatna winieta, która skupia uwagę na centrum obrazu.

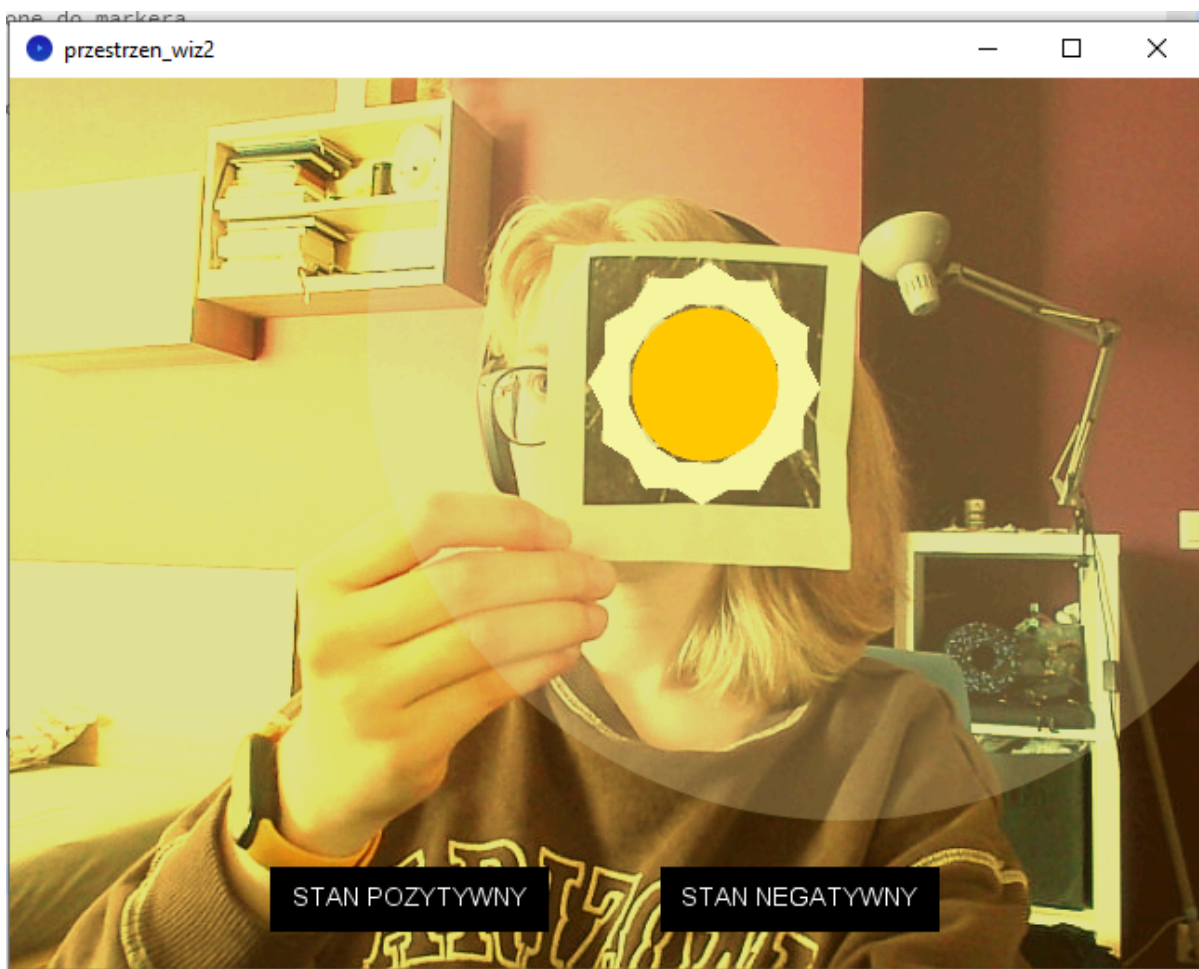
```
69 // Efekt słoneczny: nakłada ciepły, złoty filtr z efektem promieni słonecznych i winietą
70 void applySunshineEffect() {
71     PGraphics pg = createGraphics(width, height);
72     pg.beginDraw();
73     pg.image(cam, 0, 0);
74
75     pg.loadPixels();
76     for (int i = 0; i < pg.pixels.length; i++) {
77         color c = pg.pixels[i];
78         float r = red(c);
79         float g = green(c);
80         float b = blue(c);
81
82         // Wzmocnienie czerwonych i żółtych tonów
83         r = min(r * (1.0 + sunEffect * 0.7), 255);
84         g = min(g * (1.0 + sunEffect * 0.4), 255);
85         b = b * (1.0 - sunEffect * 0.3);
86
87         // Dodanie złotego efektu
88         float warmth = sunEffect * WARMTH_INTENSITY;
89         r = min(r + 30 * warmth, 255);
90         g = min(g + 15 * warmth, 255);
91
92         pg.pixels[i] = color(r, g, b);
93     }
94     pg.updatePixels();
95
96     // Dodajemy efekt promieni słonecznych, jeśli intensywność efektu jest wystarczająca
97     if (sunEffect > 0.3) {
98         pg.blendMode(SCREEN);
99         pg.fill(255, 220, 150, 80 * sunEffect);
100         pg.noStroke();
101         pg.ellipse(width * 0.7, height * 0.3, width * 0.8, width * 0.8);
102         pg.blendMode(BLEND);
103     }
104 }
```



```

104
105 // robienie winiety
106 pg.fill(0, 0, 0, 30 * sunEffect);
107 pg.noStroke();
108 pg.beginPath();
109 pg.vertex(0, 0);
110 pg.vertex(width, 0);
111 pg.vertex(width, height);
112 pg.vertex(0, height);
113 pg.beginContour();
114 pg.vertex(width*0.1, height*0.1);
115 pg.vertex(width*0.9, height*0.1);
116 pg.vertex(width*0.9, height*0.9);
117 pg.vertex(width*0.1, height*0.9);
118 pg.endContour();
119 pg.endShape();
120
121 pg.endDraw();
122
123 image(pg, 0, 0);
124 }

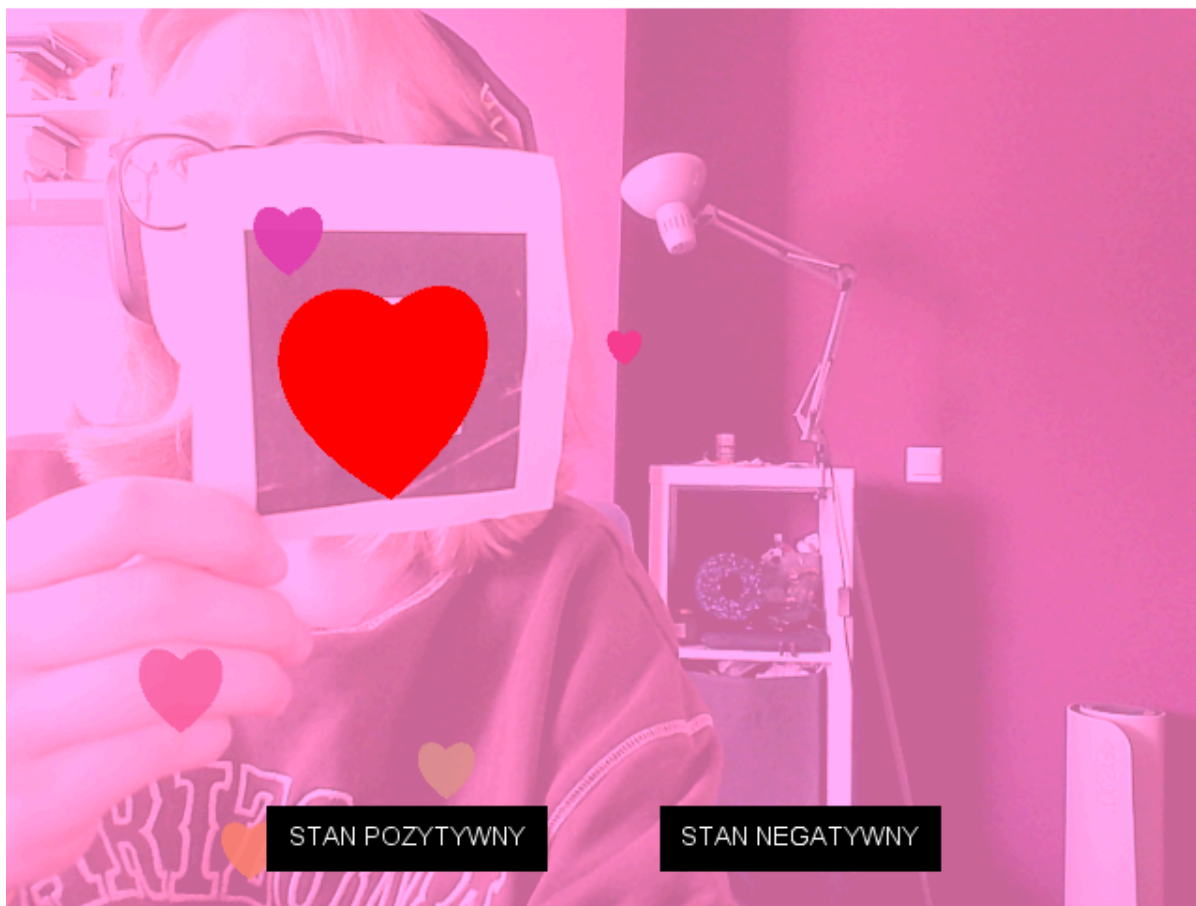
```



## Efekt miłości:

Obraz z kamery jest przetwarzany, gdzie piksele są mieszane z odcieniami różu – oryginalne kolory są redukowane, a do nich dodawane są stałe wartości wzmacniające różowy ton. Nakładany jest efekt miękkiego blasku poprzez zastosowanie funkcji `tint()` i dynamiczne rysowanie unoszących się elementów (serduszek) jako dodatkowych obiektów.

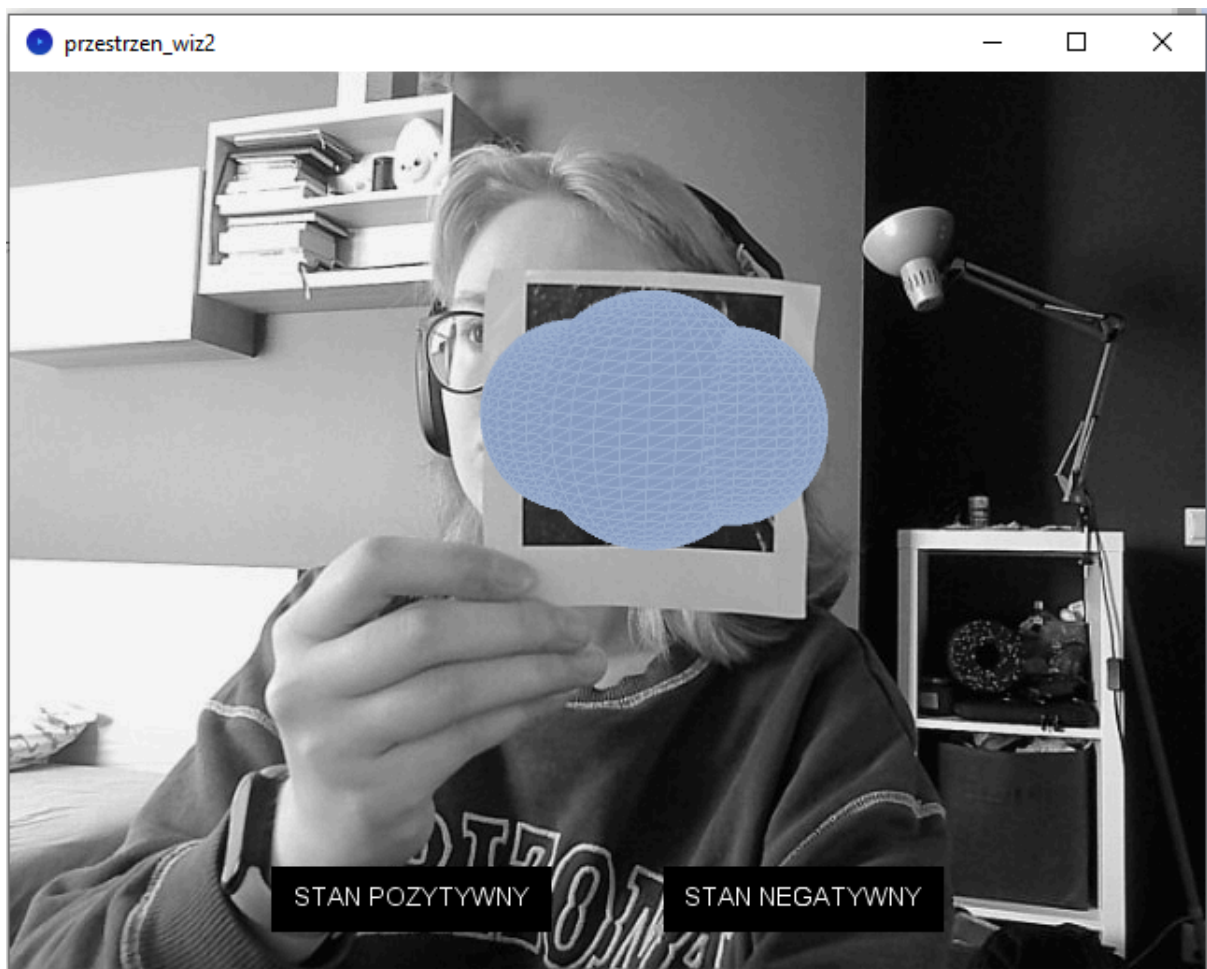
```
125
126 // Efekt miłości: nakłada różowy filtr oraz dodaje unoszące się serduszka
127 void applyLoveEffect() {
128     if (cam.available()) {
129         cam.read();
130     }
131
132     nya.detect(cam);
133     markerDetected = nya.isExist(0);
134
135     // Zwiększamy lub zmniejszamy intensywność efektu miłosnego (loveGlow) w zależności od wykrycia markera
136     if (markerDetected) {
137         loveGlow = min(loveGlow + glowSpeed, 1.0);
138
139         // dodajemy latające serduszka
140         if (frameCount % 30 == 0 && hearts.size() < 15) {
141             hearts.add(new Heart());
142         }
143     } else {
144         loveGlow = max(loveGlow - glowSpeed, 0.0);
145     }
146
147     // Jeśli efekt miłości jest aktywny, tworzymy obraz z różowym filtrem
148     if (loveGlow > 0) {
149         cam.loadPixels();
150         PImage pinkImage = createImage(cam.width, cam.height, RGB);
151         pinkImage.loadPixels();
152
153         for (int i = 0; i < cam.pixels.length; i++) {
154             color c = cam.pixels[i];
155             // Przekształcamy kolory do odcieni różu w zależności od intensywności loveGlow
156             float r = red(c) * 0.5 + 200 * loveGlow;
157             float g = green(c) * 0.3 + 100 * loveGlow;
158             float b = blue(c) * 0.4 + 150 * loveGlow;
159             pinkImage.pixels[i] = color(
160                 constrain(r, 0, 255),
161                 constrain(g, 0, 255),
162                 constrain(b, 0, 255)
163             );
164         }
165         pinkImage.updatePixels();
166
167         // Nakładamy efekt blasku
168         tint(255, 200 + 55 * loveGlow);
169         image(pinkImage, 0, 0);
170         noTint();
171
172         // Rysujemy unoszące się serduszka, które aktualizują swoją pozycję i znikają, gdy opuszczają ekran
173         for (int i = hearts.size()-1; i >= 0; i--) {
174             Heart h = hearts.get(i);
175             h.update();
176             h.display();
177             if (h.isOffScreen()) {
178                 hearts.remove(i);
179             }
180         }
181     } else {
182         image(cam, 0, 0);
183     }
184 }
185
```



## Efekt czarno-biały:

efekt czarno-biały zamienia każdy piksel na odcień szarości, a następnie pozwala stopniowo mieszać tę szarość z oryginalnymi kolorami, w zależności od wartości zmiennej **bwEffect**

```
186 void applyBlackWhiteEffect() {
187     // Tworzymy bufor graficzny
188     PGraphics pg = createGraphics(width, height);
189     pg.beginDraw();
190
191     pg.image(cam, 0, 0);
192
193     // zamieniamy kolory na szarość
194     if (bwEffect > 0) {
195         pg.loadPixels();
196         for (int i = 0; i < pg.pixels.length; i++) {
197             color c = pg.pixels[i];
198
199             float gray = red(c) * 0.299 + green(c) * 0.587 + blue(c) * 0.114;
200             // Mieszamy oryginalny kolor z wartością szarości
201             float r = lerp(red(c), gray, bwEffect);
202             float g = lerp(green(c), gray, bwEffect);
203             float b = lerp(blue(c), gray, bwEffect);
204             pg.pixels[i] = color(r, g, b);
205         }
206         pg.updatePixels();
207     }
208
209     pg.endDraw();
210
211     image(pg, 0, 0);
212 }
```



## Efekt niepokoju:

Obraz niepokoju tworzony jest przez pulsującą, dynamiczną zmianę kolorów, w której oryginalny obraz jest konwertowany do odcieni szarości, a następnie mieszany z intensywnym czerwonym kolorem. Intensywność tej czerwieni regularnie pulsuje między określonymi wartościami minimalną i maksymalną, co powoduje efekt migania.

```
213 // Efekt niepokoju: tworzy pulsujący, czerwony efekt i dodatkowo wykrywa krawędzie
214 void applyAnxietyEffect() {
215     if (cam.available()) {
216         cam.read();
217     }
218
219     nya.detect(cam);
220     markerDetected = nya.isExist(0);
221
222     // Aktualizacja intensywności efektu pulsacji
223     if (markerDetected) {
224         if (increasing) {
225             pulseIntensity += pulseSpeed;
226             if (pulseIntensity >= 1.0) {
227                 pulseIntensity = 1.0;
228                 increasing = false;
229             }
230         } else {
231             pulseIntensity -= pulseSpeed;
232             if (pulseIntensity <= 0.3) {
233                 pulseIntensity = 0.3;
234                 increasing = true;
235             }
236         }
237     } else {
238         pulseIntensity = 0;
239     }
240
241     if (markerDetected) {
242
243         cam.loadPixels();
244         PImage redImage = createImage(cam.width, cam.height, RGB);
245         redImage.loadPixels();
246
247         for (int i = 0; i < cam.pixels.length; i++) {
248             color c = cam.pixels[i];
249             // Calculate grayscale value
250             float gray = red(c) * 0.299 + green(c) * 0.587 + blue(c) * 0.114;
251             // Apply pulsing red effect
252             float r = gray + (255 - gray) * pulseIntensity;
253             float g = gray * (1.0 - pulseIntensity * 0.9);
254             float b = gray * (1.0 - pulseIntensity * 0.9);
255             redImage.pixels[i] = color(r, g, b);
256         }
257         redImage.updatePixels();
258     }
259 }
```

Dodatkowo, obraz delikatnie się skaluje, sprawiając wrażenie lekkiego zakłócenia lub drżenia. Co kilka klatek na cały obraz nakładany jest półprzezroczysty czerwony prostokąt, potęgując migotanie.



```

259 // Dodajemy efekt przeskalowania obrazu dla efektu "zakłócenia"
260 pushMatrix();
261 translate(width/2, height/2);
262 scale(1.0 + pulseIntensity * 0.05);
263 translate(-width/2, -height/2);
264 image(redImage, 0, 0);
265 popMatrix();
266
267 // Dodajemy migający czerwony overlay (efekt migotania)
268 if (frameCount % 5 == 0) { // Random migotanie
269     fill(255, 0, 0, 30 * pulseIntensity);
270     rect(0, 0, width, height);
271 }
272 } else {
273     image(cam, 0, 0);

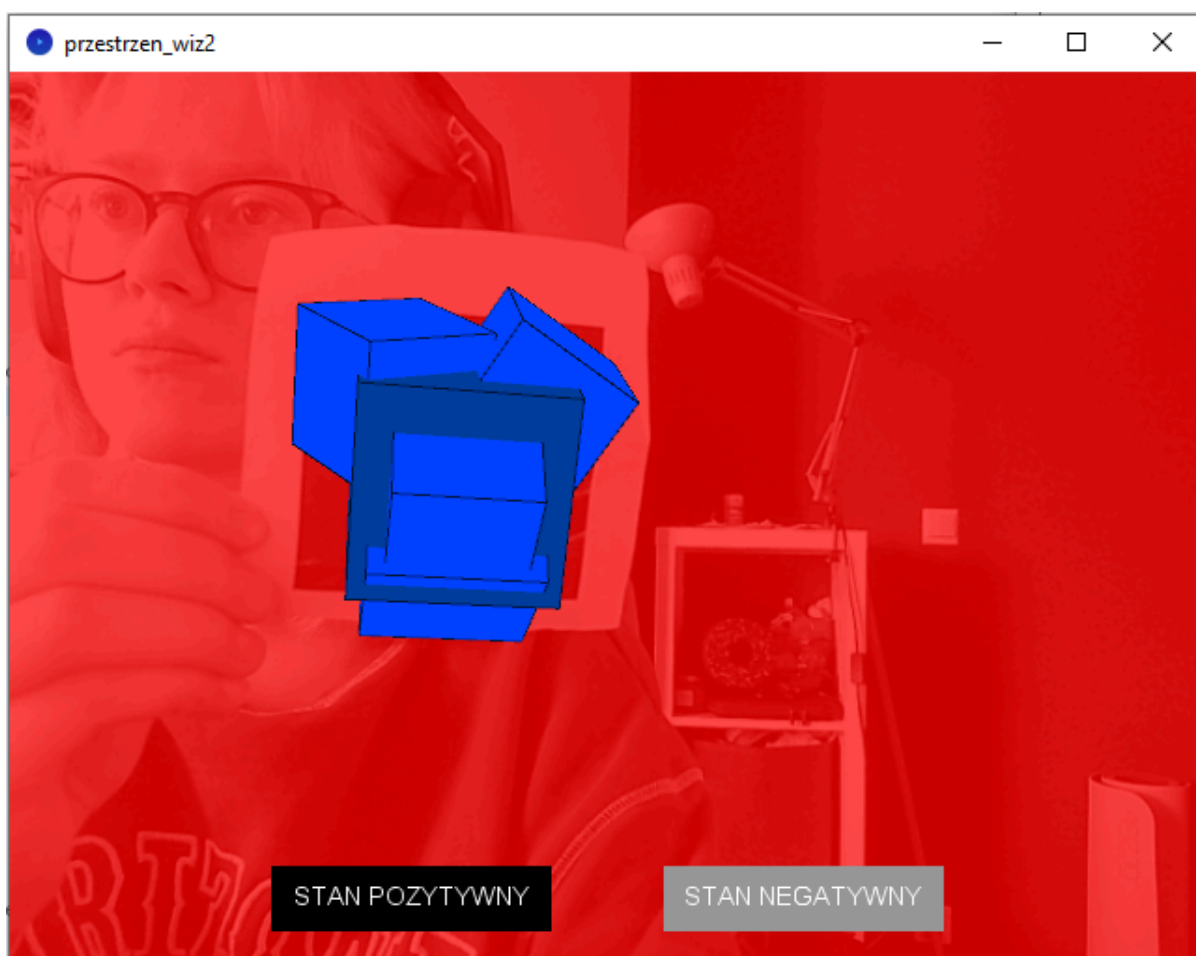
```

Na końcu, dla wzmocnienia efektu, przy pomocy biblioteki OpenCV obraz poddawany jest analizie w celu wykrycia krawędzi metodą Canny'ego. W efekcie tego powstają dodatkowe linie i kontury na obrazie, które wprowadzają wizualny chaos, potęgując uczucie niepokoju.

```

275 // Dodatkowo, używamy OpenCV do wykrywania krawędzi w obrazie z kamery:
276 opencv.loadImage(cam); // Ładujemy aktualny obraz z kamery do OpenCV
277 opencv.findCannyEdges(30, 100); // Wykrywamy krawędzie metodą Canny z progami 30 i 100
278 edgeImage = opencv.getSnapshot(); // Pobieramy obraz z wykrytymi krawędziami
279 }
280
281 // Funkcja obsługująca zdarzenia przechwytywania obrazu z kamery
282 void captureEvent(Capture c) {
283     c.read();
284 }

```



## Wyświetlanie i łączenie efektów

Dla każdego wykrytego markera rozpoczynana jest transformacja. W zależności od wykrytego markera i wybranego stanu (icon\_set) wywoływane są funkcje rysujące odpowiednie ikony (np. drawMilosc, drawRadosc, drawNiepokoj, drawSmutek).

```
1 void visual_setup(){
2     // Obliczamy liczbę kolumn i wierszy na podstawie szerokości/wyokości okna i wielkości komórki
3     cols = width / cellSize;
4     rows = height / cellSize;
5     opencv = new OpenCV(this, width, height);
6 }
7
8
9 // Główna funkcja aplikująca efekty wizualne na obraz z kamery
10 void apply_effects() {
11     // Odczyt kamery i detekcja markerów
12     if (cam.available()) {
13         cam.read();
14     }
15     // Wykrywamy markery AR za pomocą biblioteki MultiMarker
16     nya.detect(cam);
17
18     // Rysujemy tło z kamery
19     background(0);
20     nya.drawBackground(cam);
21
22     // 1. Rysowanie ikon na markerach
23     for (int i = 0; i < 4; i++) {
24         if (!nya.isExist(i)) {
25             continue;
26         }
27         // Rozpoczynamy transformację, aby przyczepić ikonę do markera
28         nya.beginTransform(i);
29
30         // Rysujemy tylko ikonki - BEZ wywoływania efektów globalnych!
31         if (i == 0 && icon_set == 1) {
32             drawMilosc();
33         } else if (i == 1 && icon_set == 1) {
34             drawRadosc();
35         } else if (i == 0 && icon_set == 2) {
36             drawNiepokoj();
37         } else if (i == 1 && icon_set == 2) {
38             drawSmutek();
39         }
40     }
41 }
```

Równolegle odtwarzane są efekty dźwiękowe (playEffectMusic, playEffectSound).

```
41 // Ewentualnie odtwarzanie dźwięku
42 playEffectMusic(icon_set, i);
43 playEffectSound(icon_set, i);
44
45 nya.endTransform();
46 }
47
```

Później nakładane są globalne efekty. Po narysowaniu ikon, na cały obraz nakładane są dodatkowe filtry.

Dla stanu pozytywnego wywoływane są np. applyLoveEffect i applySunshineEffect, a dla stanu negatywnego – applyAnxietyEffect i applyBlackWhiteEffect.



```

47
48 // 2. Nakładanie globalnych efektów na cały ekran
49 switch(icon_set) {
50     case 1:
51         // Dla icon_set 1, jeśli dany marker jest widoczny, nakładamy odpowiedni efekt:
52         if (nya.isExist(0)) {
53             applyLoveEffect();
54         }
55         if (nya.isExist(1)) {
56             applySunshineEffect();
57         }
58         break;
59     case 2:
60         if (nya.isExist(0)) {
61             applyAnxietyEffect();
62         }
63         if (nya.isExist(1)) {
64             applyBlackWhiteEffect();
65         }
66         break;
67 }
68 }

```