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.

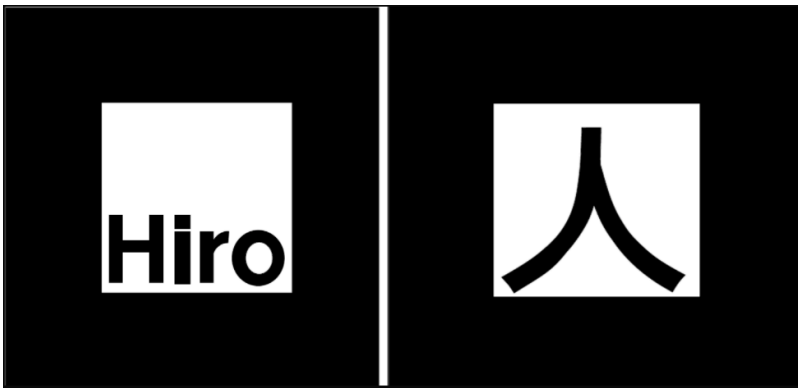
Technologie i biblioteki

- **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

```
void GUI_setup(){
cp5 = new ControlP5(this);

// tworzenie przycisków oraz pozycje startową
  int buttonWidth1 = 150;
  int buttonHeight = 35;
  int startX = width/2;
  int startY = height - buttonHeight - 20;


  // Dodajemy pierwszy przycisk "Stan pozytywny"
  b1 = (Button) cp5.addButton("Stan pozytywny")
     .setPosition(startX - 180, startY)
     .setSize(buttonWidth1, buttonHeight)
     .setColorBackground(color(0))
     .setColorForeground(color(255));

  // Ustawiamy czcionkę napisu na przycisku "Stan pozytywny"
  b1.getCaptionLabel().setFont(createFont("Arial", 14));

  // Dodajemy drugi przycisk "Stan negatywny"
  b2 = (Button) cp5.addButton("Stan negatywny")
     .setPosition(startX + 30, startY)
     .setSize(buttonWidth1, buttonHeight)
     .setColorBackground(color(0))
     .setColorForeground(color(255));

   // Ustawiamy czcionkę napisu na przycisku "Stan negatywny"
  b2.getCaptionLabel().setFont(createFont("Arial", 14));

}
```

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

```
// Funkcja obsługująca kliknięcia przycisków
void controlEvent(ControlEvent event) {
  // Pobieramy nazwę przycisku, który wywołał zdarzenie
  String buttonName = event.getController().getName();

  b1.setColorBackground(color(0));
  b1.setColorForeground(color(255));
  b2.setColorBackground(color(0));
  b2.setColorForeground(color(255));

  // Sprawdzamy, który przycisk został kliknięty:
  if (buttonName.equals("Stan pozytywny")) {
    icon_set = 1;
    // Zmieniamy kolory przycisku, aby wskazać, że jest aktywny:
    b1.setColorBackground(color(150));
    b1.setColorForeground(color(0));
  } else if (buttonName.equals("Stan negatywny")) {
    icon_set = 2;
    b2.setColorBackground(color(150));
    b2.setColorForeground(color(0));
  }
}
```

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.

```
void icons_setup() {

  println(MultiMarker.VERSION);
  cam = new Capture(this, 640, 480);
  nya = new MultiMarker(this, width, height, "camera_para.dat", NyAR4PsgConfig.CONFIG_PSG);
  nya.addARMarker("patt.hiro",80);   // id=0
  nya.addARMarker("patt.kanji",80);  // id=1
  nya.addARMarker("h.patt",80);   // id=2
  nya.addARMarker("a.patt",80);  // id=3
  cam.start();
}
// rysowanie serca
void drawMilosc() {
    scale(1, -1);
    fill(255, 0, 0);
    noStroke();
    beginShape();
      vertex(0, -20);
      bezierVertex(-25, -40, -50, 0, 0, 40);
      bezierVertex(60, 0, 25, -40, 0, -20);
    endShape(CLOSE);
}
```

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.

```
// rysowanie chmurki
void drawSmutek() {
   stroke(63, 67, 69);
  fill(54, 55, 56);
  sphereDetail(16);

  // Główna (największa) sfera
  pushMatrix();
    sphere(40);
  popMatrix();

  // Kilka mniejszych sfer, przesuniętych względem głównej
  pushMatrix();
    translate(-25, 0, 10);
    sphere(30);
  popMatrix();

  pushMatrix();
    translate(25, 0, -10);
    sphere(30);
  popMatrix();

  pushMatrix();
    translate(10, -15, 0);
    sphere(20);
  popMatrix();

  pushMatrix();
    translate(-10, -15, 0);
    sphere(20);
  popMatrix();
}
```

Ikonka słońca -radosć

```
56 // rysowanei 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         // promienei jako box
76         box(8, 40, 8);
77       popMatrix();
78     }
79 }
```

Ikonka box - niepokój

```
// box złożony z boxów
void drawNiepokoj() {
  stroke(0, 23, 59);

  // glowny box
  fill(2, 62, 158);
  box(60);


  // Box dol
  pushMatrix();
    translate(0, -20, 0);
    rotateX(PI/4);
    fill(0, 68, 255);
    box(50);
  popMatrix();

  // Box 2 lewo gora
  pushMatrix();
    translate(20, 20, 0);
    rotateY(PI/3);
    fill(0, 68, 255);
    box(40);
  popMatrix();

  // Box 3: prawo gora
  pushMatrix();
    translate(-20, 20, 0);
    rotateZ(PI/6);
    fill(0, 68, 255);
    box(40);
  popMatrix();

 // Box 4:
  pushMatrix();
    translate(0, 0, 20);
    rotateX(PI/6);
    fill(0, 68, 255);
    box(40);
  popMatrix();
}
```

\

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.

1. Odtwarzanie dźwięków (SoundFile) - music

```
import processing.sound.*;

SoundFile loveSound, sunshineSound, anxietySound, sadnessSound;


// Ładuje pliki dźwiękowe (np. "love.mp3", "sunshine.mp3", "anxiety.wav", "sadness.mp3")
void music_setup() {
  loveSound = new SoundFile(this, "love.mp3");
  sunshineSound = new SoundFile(this, "sunshine.mp3");
  anxietySound = new SoundFile(this, "anxiety.wav");
  sadnessSound = new SoundFile(this, "sadness.mp3");
}

void playEffectMusic(int iconSet, int marker) {
  stopAllSounds(iconSet, marker); // zatrzymanie dźwięków, aby nie zaszło nakładanie się na siebie dźwieku

  if (iconSet == 1 && marker == 0) {
    if (!loveSound.isPlaying()){
      loveSound.loop();
    }
  } else if (iconSet == 1 && marker == 1) {
    if (!sunshineSound.isPlaying()){
      sunshineSound.loop();
    }

  } else if (iconSet == 2 && marker == 0) {
    if (!anxietySound.isPlaying()){
      anxietySound.loop();
    }
  } else if (iconSet == 2 && marker == 1) {
    if (!sadnessSound.isPlaying()){
      sadnessSound.loop();
    }
  }
```

```
// przy aktywacji jednego dźwięku zatrzymywany jest dźwięk przeciwnego stanu
void stopAllSounds(int iconSet, int marker) {

  if (iconSet == 1){
    anxietySound.stop();
    sadnessSound.stop();
    if (marker == 0){
       sunshineSound.stop();
    } else if( marker == 1){
      loveSound.stop();
    }
  } else if (iconSet == 2){
    loveSound.stop();
    sunshineSound.stop();
    if (marker == 0){
      sadnessSound.stop();
    } else if( marker == 1) {
      anxietySound.stop();
    }
  }
}
```

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).

```
import processing.sound.*;
// Deklaracja oscylatorów (SinOsc) i generatora szumu (WhiteNoise)
SinOsc sunOsc, loveOsc, anxietyOsc;
WhiteNoise noise;

void sound_setup() {
  //size(800, 600);
// Deklaracja oscylatorów (SinOsc) i generatora szumu (WhiteNoise)
  sunOsc = new SinOsc(this);
  loveOsc = new SinOsc(this);
  anxietyOsc = new SinOsc(this);
  noise = new WhiteNoise(this);
}
```

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.

```
// Funkcja odtwarzająca efekt dźwiękowy w zależności od stanu emocjonalnego (icon_set) i indeksu markera
void playEffectSound(int icon_set, int index) {
  // Dla stanu pozytywnego (icon_set == 1)
  if (icon_set == 1 && index == 1) {
    sunOsc.freq(220 + random(10, 50));  // Ustawia losową częstotliwość w określonym zakresie
    sunOsc.amp(0.3);                    // Ustawia głośność oscylatora
    sunOsc.play();                      // Odtwarza dźwięk
    delay(500);                         // Czeka 500 ms
    fadeOut(sunOsc);                    // Stopniowo wygasza dźwięk
  }
  else if (icon_set == 1 && index == 0) {
    loveOsc.freq(330);
    loveOsc.amp(0.5);
    loveOsc.play();
    delay(200);
    fadeOut(loveOsc);
  }
    // Dla stanu negatywnego (icon_set == 2)
  else if (icon_set == 2 && index == 0) {
    anxietyOsc.freq(random(100, 200));
    anxietyOsc.amp(0.5);
    anxietyOsc.play();
    delay(200);
    fadeOut(anxietyOsc);
  }

  else if (icon_set == 2 && index == 1) {
    noise.amp(0.2);
    noise.play();
    delay(500);
    fadeOut(noise);
  }
}
```

Po krótkim opóźnieniu dźwięk stopniowo zanika dzięki funkcji fadeOut.

```
}
// Funkcja wygaszania dźwięku dla oscylatora SinOsc
void fadeOut(SinOsc osc) {
  for (float i = 0.2; i > 0; i -= 0.02) {
    osc.amp(i);   // Stopniowe zmniejszanie amplitudy
    delay(50);    // Opóźnienie dla efektu wygaszania
  }
  osc.stop();     // Zatrzymuje odtwarzanie po wygaszeniu
}
// Funkcja wygaszania dźwięku dla generatora WhiteNoise
void fadeOut(WhiteNoise noise) {
  for (float i = 0.2; i > 0; i -= 0.02) {
    noise.amp(i);
    delay(50);
  }
  noise.stop();
}
```

\

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
```

```
  Heart() {
    x = random(width);
    y = height + 20;
    size = random(15, 40);
    speed = random(1, 3);
    heartColor = color(
      220 + random(35),
      50 + random(100),
      100 + random(100),
      150 + random(105)
    );
  }

  void update() {
    y -= speed;
    x += random(-0.5, 0.5);
  }

  void display() {
    noStroke();
    fill(heartColor);
    beginShape();
    vertex(x, y);
    bezierVertex(x - size/2, y - size/2, x - size, y + size/3, x, y + size);
    bezierVertex(x + size, y + size/3, x + size/2, y - size/2, x, y);
    endShape();
  }

  boolean isOffScreen() {
    return y < -size;
  }
}
OpenCV opencv;
PImage edgeImage;
```

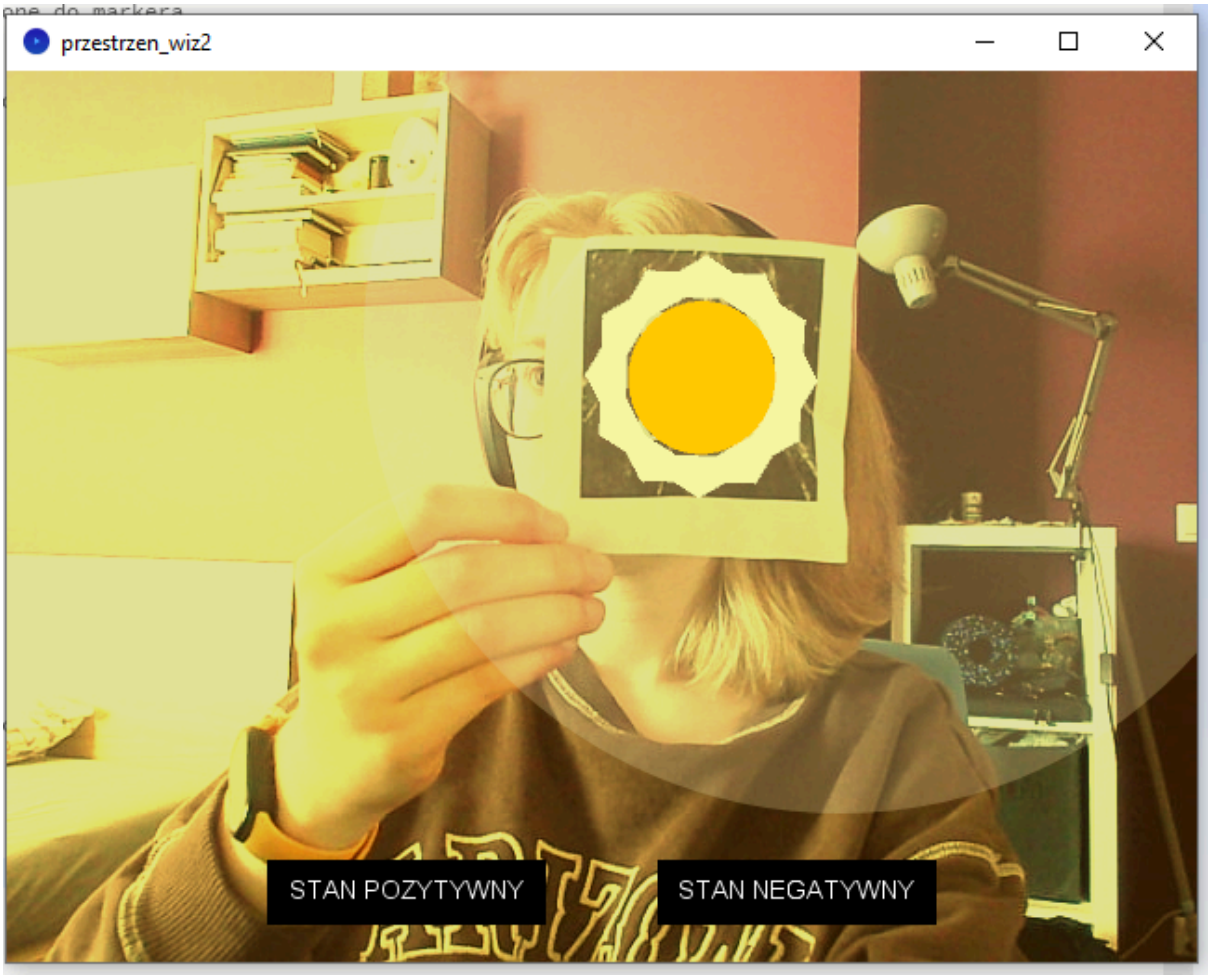
Efekt słoneczny

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
105   // robienie winiety
106   pg.fill(0, 0, 0, 30 * sunEffect);
107   pg.noStroke();
108   pg.beginShape();
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 }
```

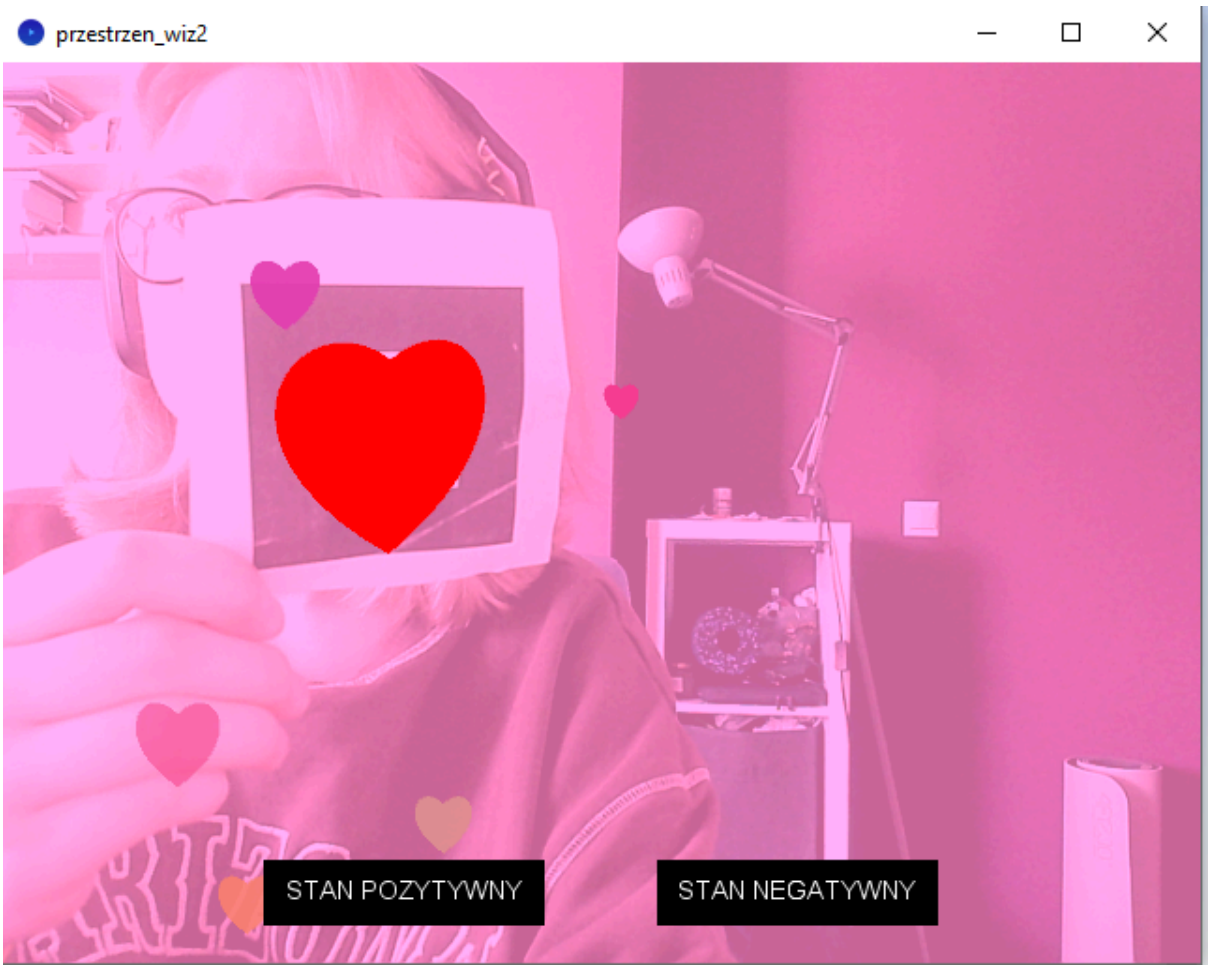
one do markera



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.

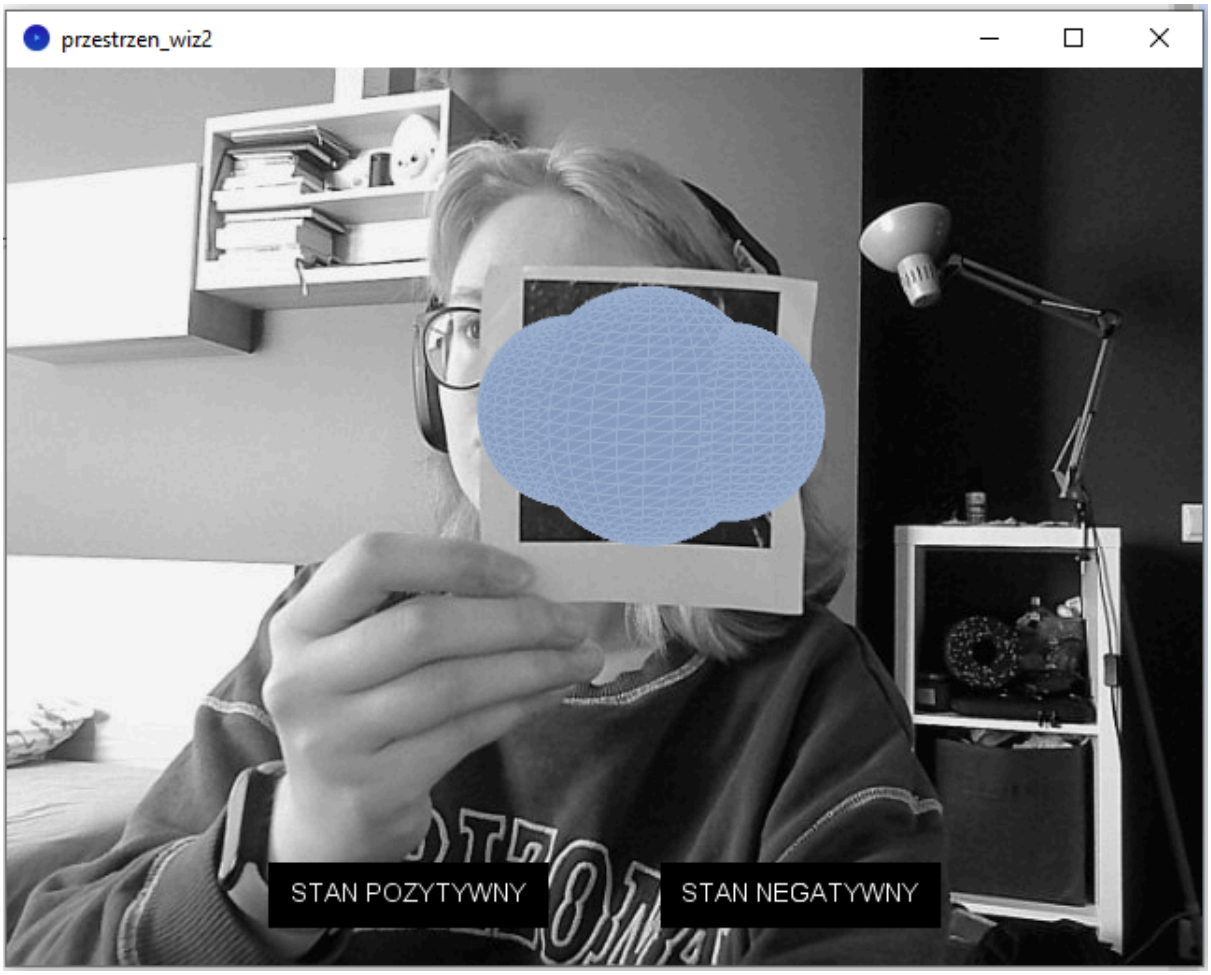
```
125
126 // Efekt miłości: nakłada różowy filtr oraz dodaje unoszące się serduszka
127 void applyLoveEffect() {
128   if (cam.available()) {
129     cam.read();
130   }
131
132   nya.detect(cam);
133   markerDetected = nya.isExist(0);
134
135   // Zwiększamy lub zmniejszamy intensywność efektu miłosnego (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 opuszczą 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
```


przestrzen_wiz2
STAN POZYTYWNY
STAN NEGATYWNY

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 }
```


przestrzen_wiz2
STAN POZYTYWNY
STAN NEGATYWNY

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.

```
// Efekt niepokoju: tworzy pulsujący, czerwony efekt i dodatkowo wykrywa krawędzie
void applyAnxietyEffect() {
  if (cam.available()) {
    cam.read();
  }

  nya.detect(cam);
  markerDetected = nya.isExist(0);

  // Aktualizacja intensywności efektu pulsacji
  if (markerDetected) {
    if (increasing) {
      pulseIntensity += pulseSpeed;
      if (pulseIntensity >= 1.0) {
        pulseIntensity = 1.0;
        increasing = false;
      }
    } else {
      pulseIntensity -= pulseSpeed;
      if (pulseIntensity <= 0.3) {
        pulseIntensity = 0.3;
        increasing = true;
      }
    }
  } else {
    pulseIntensity = 0;
  }
```

```
  if (markerDetected) {

    cam.loadPixels();
    PImage redImage = createImage(cam.width, cam.height, RGB);
    redImage.loadPixels();

    for (int i = 0; i < cam.pixels.length; i++) {
      color c = cam.pixels[i];
      // Calculate grayscale value
      float gray = red(c) * 0.299 + green(c) * 0.587 + blue(c) * 0.114;
      // Apply pulsing red effect
      float r = gray + (255 - gray) * pulseIntensity;
      float g = gray * (1.0 - pulseIntensity * 0.9);
      float b = gray * (1.0 - pulseIntensity * 0.9);
      redImage.pixels[i] = color(r, g, b);
    }
    redImage.updatePixels();
```

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

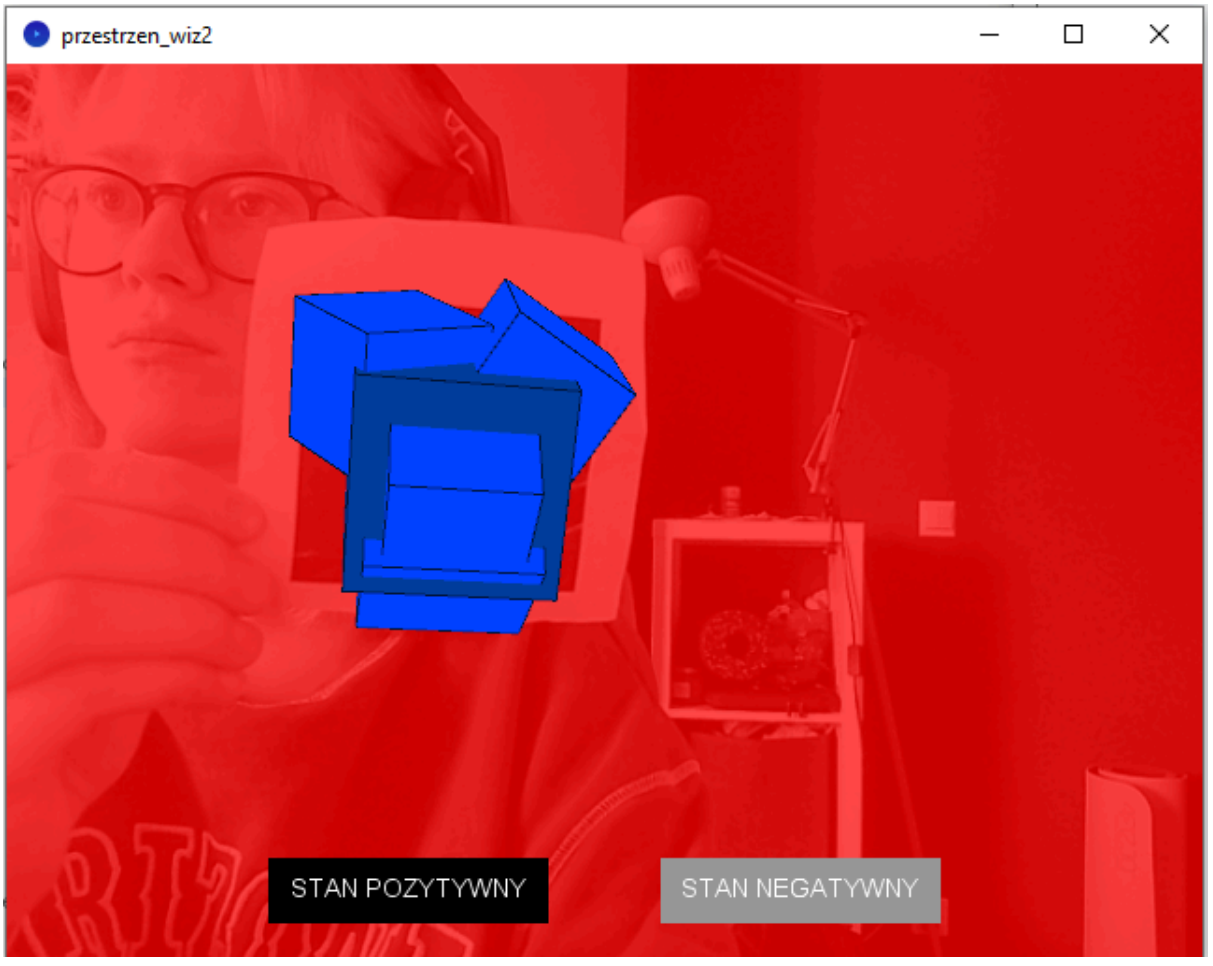
```
    // Dodajemy efekt przeskalowania obrazu dla efektu "zakłócenia"
    pushMatrix();
    translate(width/2, height/2);
    scale(1.0 + pulseIntensity * 0.05);
    translate(-width/2, -height/2);
    image(redImage, 0, 0);
    popMatrix();

    // Dodajemy migający czerwony overlay (efekt migotania)
    if (frameCount % 5 == 0) { // Random migotanie
      fill(255, 0, 0, 30 * pulseIntensity);
      rect(0, 0, width, height);
    }
  } else {
    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.

```
  // Dodatkowo, używamy OpenCV do wykrywania krawędzi w obrazie z kamery:
  opencv.loadImage(cam);                  // Ładujemy aktualny obraz z kamery do OpenCV
  opencv.findCannyEdges(30, 100);           // Wykrywamy krawędzie metodą Canny z progami 30 i 100
  edgeImage = opencv.getSnapshot();         // Pobieramy obraz z wykrytymi krawędziami
}

// Funkcja obsługująca zdarzenia przechwytywania obrazu z kamery
void captureEvent(Capture c) {
  c.read();
}
```



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).

```
void visual_setup(){
  // Obliczamy liczbę kolumn i wierszy na podstawie szerokości/wyokości okna i wielkości komórki
  cols = width / cellSize;
  rows = height / cellSize;
  opencv = new OpenCV(this, width, height);
}


// Główna funkcja aplikująca efekty wizualne na obraz z kamery
void apply_effects() {
  // Odczyt kamery i detekcja markerów
  if (cam.available()) {
    cam.read();
  }
   // Wykrywamy markery AR za pomocą biblioteki MultiMarker
  nya.detect(cam);

  // Rysujemy tło z kamery
  background(0);
  nya.drawBackground(cam);

  // 1. Rysowanie ikon na markerach
  for (int i = 0; i < 4; i++) {
    if (!nya.isExist(i)) {
      continue;
    }
    // Rozpoczynamy transformację, aby przyczepic ikone do markera
    nya.beginTransform(i);

    // Rysujemy tylko ikonki - BEZ wywoływania efektów globalnych!
    if (i == 0 && icon_set == 1) {
      drawMilosc();
    } else if (i == 1 && icon_set == 1) {
      drawRadosc();
    } else if (i == 0 && icon_set == 2) {
      drawNiepokoj();
    } else if (i == 1 && icon_set == 2) {
      drawSmutek();
    }
```

Równolegle odtwarzane są efekty dźwiękowe (playEffectMusic, playEffectSound).

```
      // Ewentualnie odtwarzanie dźwięku
     playEffectMusic(icon_set, i);
     playEffectSound(icon_set, i);

      nya.endTransform();
    }
```

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  }
```