Program Studi Teknik Informatika
Institut Teknologi Sumatera

Mata Kuliah: **Sistem/Teknologi Multimedia (IF4021)** Tugas: **Tugas Besar**

Nama:

1. **Ikhsannudin Lathief (122140137)**
2. **Rustian Afencius Marbun (122140155)**
3. **Eden Wijaya (122140187)**

Suaraku Terbang

Program Filter *Tiktok* Menggunakan Python

1 Pendahuluan

1.1 Latar Belakang

Proyek besar dengan judul "Suaraku Terbang" ini merupakan tugas akhir dari mata kuliah Sistem/Teknologi Multimedia (IF4021). Proyek ini menggunakan konsep dasar permainan yang dikontrol oleh suara (frekuensi tinggi dan frekuensi rendah). Program ini dibangun dengan menggunakan bahasa pemrograman Python dan menggunakan beberapa library seperti numpy, opencv, mediapipe, sound device, time, dan threading.
Secara sederhana, Suaraku Terbang ini terinspirasi dari game *flappy bird*, namun objek bergerak berdasarkan frekuensi suara yang ditangkap. Suaraku terbang memiliki objek berbentuk bola, jika menangkap suara dengan frekuensi tinggi, maka objek bola akan bergerak naik, jika menangkap suara dengan frekuensi rendah, maka objek bola akan bergerak turun. Selain itu, terdapat 2 pembatas pada atas dan bawah, serta memiliki 5 tingkatan level, 2 pembatas tersebut akan semakin kecil jika berhasil naik level. Ketika bola menyenentuh pembatas, maka user akan kalah dan Suaraku Terbang akan berakhir.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, umusan masalah pada proyek ini ialah :

1. Bagaimana merancang dan mengimplementasikan sistem deteksi frekuensi suara secara real-time yang mampu membedakan antara frekuensi tinggi dan rendah untuk mengontrol gerakan objek dalam permainan?

2. Bagaimana cara mengintegrasikan library Python seperti numpy, OpenCV, MediaPipe, dan SoundDevice untuk membangun fungsionalitasdan logika permainan?

3. Bagaimana sistem tingkatan level dalam proyek "Suaraku Terbang" dapat diimplementasikan untuk memberikan tantangan yang terus meningkat?,

1.3 Tujuan

Adapun tujuan dari proyek "Suaraku Terbang" ini ialah:

1. Mengimplementasikan sistem deteksi frekuensi suara real-time untuk membedakan input suara frekuensi tinggi dan rendah
2. Membangun sebuah permainan "Suaraku Terbang" yang interaktif, di mana gerakan objek bola dikendalikan sepenuhnya oleh frekuensi suara pengguna (naik untuk frekuensi tinggi, turun untuk frekuensi rendah).
3. Mengintegrasikan berbagai library Python untuk menciptakan pemrosesan audio, logika permainan, deteksi tabrakan, dan antarmuka visual.

2 Alat dan Bahan

Proyek "Suaraku Terbang" ini dikembangkan dengan menggunakan bahasa pemrograman Python dan library untuk mendukung fungsionalitasnya. Berikut adalah alat dan bahan yang digunakan:

2.1 Bahasa Pemrograman dan Lingkup Pengembangan

1. Bahasa pemrograman yang dipakai untuk membuat proyek ini adalaha bahasa pemrograman Python, tepatnya Python 3.10.
2. Visual Studio Code sebagai text editor untuk pengembangan

2.2 Library

1. opencv-python 4.11.0.86, yang digunakan untuk pemrosesan gambar dan antarmuka visual
2. mediapipe 0.10.21,
3. sounddevice 0.5.1, yang digunakan untuk menangkap dan memproses input audio
4. time dan threading, yang digunakan untuk manajemen waktu dalam permainan.

2.3 Metode dan Algoritma

1. Deteksi Nada (Pitch Detection): Menggunakan algoritma Fast Fourier Transform (FFT) untuk mengubah sinyal suara dari domain waktu ke domain frekuensi, lalu mencari frekuensi dominan dan membandingkannya dengan ambang batas (threshold) untuk mengklasifikasikan suara sebagai nada tinggi (treble) atau nada rendah (bass).
2. Deteksi Wajah (Face Detection): Fitur overlay kacamata virtual memanfaatkan metode deteksi wajah dan landmark wajah yang disediakan oleh pustaka OpenCV dan mediapipe. Secara spesifik, model deteksi wajah dari mediapipe dan model deteksi landmark wajah dipanggil untuk mengidentifikasi keberadaan wajah dan melokalisasi titik-titik kunci pada wajah, terutama di sekitar area mata. Setelah koordinat yang relevan diperoleh maka diimplementasikan algoritma clipping untuk memastikan gambar kacamata tidak keluar dari batas frame video.

3 Penjelasan Program

3.1 Arsitektur Umum dan Struktur File

Program "Suaraku Terbang" dibangun menggunakan arsitektur modular yang memisahkan tanggung jawab setiap komponen ke dalam file-file terpisah. Struktur ini memungkinkan kemudahan pengembangan, pemeliharaan, dan debugging.

3.1.1 Struktur File dan Peran Masing-masing

1. **main.py** adalah file utama yang menjalankan loop game, menginisialisasi komponen, dan mengatur alur permainan.
2. **config.py** adalah file untuk menyimpan konfigurasi global, state game, pengaturan level, dan skema warna.
3. **audio-processing.py** adalah file untuk menangani pemrosesan audio real-time dan analisis frekuensi suara.
4. **game-logic.py** adalah file yang berisi logika inti permainan seperti deteksi tabrakan.
5. **visuals.py** adalah file yang berisi fungsi-fungsi untuk menggambar elemen visual game dan UI.
6. **visuals-utils.py** adalah file yang berisi utilitas menggambar dasar seperti overlay transparan dan efek visual.
7. **particles.py** adalah file yang berisi sistem partikel untuk efek visual dinamis.

3.1.2 Dependensi Eksternal

1. **OpenCV**, untuk pemrosesan video, deteksi wajah, dan rendering visual.
2. **MediaPipe**, untuk pemrosesan video, deteksi wajah, dan rendering visual.
3. **NumPy**, untuk operasi array dan komputasi numerik.
4. **SoundDevice**, untuk perekaman audio real-time dari mikrofon.

3.2 Alur Kerja Utama Filter

Program bekerja dalam beberapa tahap utama yang berjalan secara bersamaan sebagai berikut.

3.2.1 Inisialisasi Sistem

```
cap = cv2.VideoCapture(0)
sound_processing_thread = threading.Thread(target=audio_processing.sound_thread, daemon=True)
sound_processing_thread.start()
```

Kode 1: Inisialisasi kamera dan audio

3.2.2 Loop Layar Awal

1. Menampilkan start screen dengan animasi latar belakang.
2. Menunggu input spasi untuk memulai permainan

3.2.3 Loop Game Utama

1. **Capture Frame**: Membaca frame dari webcam dan membaliknya secara horizontal
2. **Deteksi Wajah**: Menggunakan MediaPipe untuk mendeteksi landmark mata
3. **Overlay Kacamata**: Menempelkan asset kacamata sesuai posisi mata
4. **Pemrosesan Audio**: Thread terpisah menganalisis input suara secara kontinyu
5. **Update Game State**: Memperbarui posisi bola berdasarkan input suara
6. **Collision Detection**: Memeriksa tabrakan dengan penghalang
7. **Rendering**: Menggambar semua elemen visual ke frame buffer
8. **Display**: Menampilkan hasil akhir ke layar

3.3 Modul Inti dan Fungsinya

3.3.1 Pemrosesan Audio

Modul `audio_processing.py` bertanggung jawab untuk analisis audio real-time menggunakan teknik Fast Fourier Transform (FFT).

1. Fungsi **`detect_sound_direction(duration=0.1, sample_rate=44100)`**
   - Merekam audio dalam durasi singkat (100ms) untuk responsivitas
   - Menggunakan FFT untuk menganalisis spektrum frekuensi
   - Menghitung energi bass (1-150 Hz) dan treble (>150 Hz)
   - Menentukan arah gerakan berdasarkan frekuensi dominan

```
# Analisis FFT untuk mendapatkan frekuensi dominan
fft = np.fft.fft(audio_data)
freqs = np.fft.fftfreq(len(fft), 1/sample_rate)
magnitudes = np.abs(fft)
# Klasifikasi berdasarkan frekuensi
if 1 <= dominant_freq <= 150:
    current_determined_direction = "down"  # Bass
elif dominant_freq > 150:
    current_determined_direction = "up"    # Treble
```

Kode 2: Analisis FFT untuk frekuensi dominan

2. Fungsi **`sound_thread()`**
   - Berjalan sebagai daemon thread untuk pemrosesan kontinyu
   - Memperbarui variabel global `config.sound_direction` secara real-time
   - Menggunakan sleep minimal (10ms) untuk efisiensi CPU

3.3.2 Deteksi Wajah dan Overlay Kacamata

Implementasi deteksi wajah menggunakan MediaPipe Face Detection untuk akurasi tinggi dan performa optimal.

```
# Inisialisasi MediaPipe
with mp_face_detection.FaceDetection(
    min_detection_confidence=0.5
) as face_detector:
    # Konversi frame ke RGB
    image_rgb = cv2.cvtColor(webcam_frame, cv2.COLOR_BGR2RGB)
    # Deteksi wajah
    results = face_detector.process(image_rgb)
```

Kode 3: Inisialisasi MediaPipe

Algoritma Overlay Kacamata

1. **Ekstraksi Landmark**: Mengambil koordinat mata kiri dan kanan dari hasil deteksi
2. **Perhitungan Geometri**: Menghitung pusat mata dan lebar untuk scaling
3. **Transformasi Koordinat**: Konversi dari koordinat relatif ke pixel absolut
4. **Scaling Adaptif**: Menyesuaikan ukuran kacamata dengan jarak antar mata

```
# Hitung pusat dan skala kacamata
eye_center_x = (left_eye_x + right_eye_x) // 2
eye_center_y = (left_eye_y + right_eye_y) // 2
eye_width = int(2.5 * abs(right_eye_x - left_eye_x))

# Resize dan overlay kacamata
scale_factor = eye_width / kacamata.shape[1]
resized_glasses = cv2.resize(kacamata, (new_width, new_height))
webcam_frame = visuals_utils.overlay_transparent(
    webcam_frame, resized_glasses, x, y
)
```

Kode 4: Perhitungan geometri kacamata

3.3.3 Konfigurasi Global

Modul `config.py` mengatur state global menggunakan pola singleton untuk memastikan konsistensi data across modules.

1. **State Audio**

```
sound_direction = "neutral"  # up, down, neutral
sound_info = {
    "bass_energy": 0,
    "treble_energy": 0,
    "dominant_freq": 0
}
```

Kode 5: Konfigurasi state audio

2. **Sistem Level dan Skor**

```
current_level = 1
max_level = 5
current_score = 0
score_this_level = 0

# Pengaturan penghalang per level
level_barrier_settings = {
    1: (0.30, 0.70),  # Celah 40%
    2: (0.35, 0.65),  # Celah 30%
    3: (0.40, 0.60),  # Celah 20%
    4: (0.425, 0.575), # Celah 15%
    5: (0.45, 0.55)   # Celah 10%
}
```

Kode 6: Konfigurasi level dan skor

3. **Skema Warna Konsisten**

```
color_schemes = {
    "primary": (100, 150, 255),
    "secondary": (50, 200, 150),
    "accent": (255, 100, 150),
    "success": (100, 255, 100),
    "warning": (255, 200, 100),
    "danger": (255, 100, 100)
}
```

Kode 7: Skema warna

4. **Sistem Animasi dan Efek Visual**

```
ui_animations = {
    "score_pulse": 0,
    "level_glow": 0,
    "background_wave": 0
}
particles = []  # Sistem partikel dinamis
ball_trail = [] # Jejak pergerakan bola
```

Kode 8: Sistem animasi

Arsitektur ini memungkinkan modifikasi mudah pada setiap komponen tanpa mempengaruhi modul lain, serta memfasilitasi debugging dan pengembangan fitur baru.

3.3.4 Logika Permainan (`game_logic.py` dan `main.py`)

Modul ini bertanggung jawab atas mekanisme inti permainan, seperti pergerakan objek utama (bola), perhitungan skor, sistem level, dan deteksi tabrakan.

- Deteksi Tabrakan (`game_logic.py`)

Fungsi check_collision_with_barriers Fungsi ini bertugas untuk mendeteksi apakah bola bersentuhan dengan penghalang atas atau bawah. Fungsi ini menerima informasi mengenai posisi dan dimensi bola, posisi vertikal kedua penghalang, serta ketebalan penghalang. Pertama, batas vertikal bola dihitung. Kemudian, dilakukan pemeriksaan tabrakan. Bola dianggap bertabrakan dengan penghalang atas jika sisi atasnya menyentuh atau melewati batas bawah penghalang atas. Bola dianggap bertabrakan dengan penghalang bawah jika sisi bawahnya menyentuh atau melewati batas atas penghalang bawah. Fungsi ini akan mengembalikan status benar jika salah satu kondisi tabrakan terpenuhi, dan status salah jika tidak ada tabrakan.

```python
"""
File Logika Inti Game.
Berisi fungsi-fungsi yang menangani mekanika dasar permainan,
seperti deteksi tabrakan antara bola dan penghalang.
"""

def check_collision_with_barriers(ball_x, ball_y, ball_width, ball_height, top_barrier_y, bottom_barrier_y, barrier_thickness):
"""
Memeriksa apakah bola bertabrakan dengan penghalang atas atau bawah.
Mengembalikan True jika terjadi tabrakan, False jika tidak.
"""
ball_top = ball_y  # Sisi atas bola
ball_bottom = ball_y + ball_height # Sisi bawah bola

# Periksa tabrakan dengan penghalang atas
# Bola dianggap bertabrakan jika sisi atasnya menyentuh atau melewati sisi bawah penghalang atas
if ball_top <= top_barrier_y + barrier_thickness: # top_barrier_y adalah garis atas dari penghalang atas
return True

# Periksa tabrakan dengan penghalang bawah
# Bola dianggap bertabrakan jika sisi bawahnya menyentuh atau melewati sisi atas penghalang bawah
if ball_bottom >= bottom_barrier_y: # bottom_barrier_y adalah garis atas dari penghalang bawah
return True

return False # Tidak ada tabrakan
```

Kode 9: Fungsi check_collision_with_barriers

- Pergerakan Bola (`main.py`)
  Pergerakan bola dikendalikan oleh input suara yang terdeteksi. Jika ada input suara yang valid, bola akan bergerak maju secara horizontal dengan kecepatan tertentu. Kecepatan vertikal bola juga ditentukan. Jika suara yang terdeteksi bernada rendah, bola akan bergerak ke bawah. Sebaliknya, jika suara bernada tinggi, bola akan bergerak ke atas.

```python
# ...existing code...
                # Logika pergerakan bola
                ball_speed_vertical = 3 # Kecepatan vertikal bola
                ball_speed_horizontal = 3 # Kecepatan horizontal bola

                if config.sound_direction != "neutral": # Jika ada input suara
                    center_x_ball += ball_speed_horizontal # Bola bergerak ke kanan
                    if config.sound_direction == "down":
                        center_y_ball += ball_speed_vertical # Suara rendah, bola ke bawah
                    elif config.sound_direction == "up":
                        center_y_ball -= ball_speed_vertical # Suara tinggi, bola ke atas
# ...existing code...
```

Kode 10: Logika pergerakan bola

- Perhitungan Skor (`main.py`)
  Skor untuk level saat ini dihitung berdasarkan seberapa jauh bola telah bergerak secara horizontal melintasi area permainan. Progresi dihitung dari posisi horizontal bola. Skor ini bersifat proporsional terhadap jarak yang ditempuh bola dalam level tersebut, dengan batas maksimal 100 poin per level. Jika lebar area permainan adalah nol, skor level ini juga nol.

```python
# ...existing code...
```

```
                # Hitung skor untuk level ini berdasarkan posisi x bola
                if game_view_actual_width > 0:
                    progress_in_level_pixels = max(0, center_x_ball)
                    # Skor proporsional dengan jarak tempuh, maks 100
                    config.score_this_level = int((progress_in_level_pixels / game_view_actual_width) * 100)
                    config.score_this_level = min(config.score_this_level, 100) # Batasi maksimal 100
                else:
                    config.score_this_level = 0
# ...existing code...
```

Kode 11: Logika perhitungan skor

- Progres Level (`main.py`)
  Pemain dianggap menyelesaikan sebuah level ketika sisi kanan bola mencapai atau melewati batas kanan area permainan. Jika level saat ini belum merupakan level maksimal, skor dari level yang baru diselesaikan akan ditambahkan ke skor total. Kemudian, nomor level dinaikkan, skor untuk level tersebut direset menjadi nol, dan sebuah efek visual untuk menandakan naik level diaktifkan. Jika level maksimal telah tercapai, skor akhir dihitung. Setelah menyelesaikan level, posisi bola direset ke posisi awal, dan jejak visual bola sebelumnya dibersihkan.

```
# ...existing code...
                # Cek jika bola mencapai ujung kanan (level selesai)
                if center_x_ball + resized_ball_img.shape[1] >= game_view_actual_width:
                    if config.current_level < config.max_level: # Jika belum level maks
                        config.current_score += config.score_this_level # Tambah skor level ini ke total
                        config.score_this_level = 0 # Reset skor untuk level baru
                        config.current_level += 1   # Naik level
                        config.level_up_flash = 60  # Aktifkan efek kilat naik level
                        print(f"Level Up! Current Level: {config.current_level}, Total Score: {config.current_score}")
                    else: # Jika sudah level maks
                        config.current_score += config.score_this_level
                        config.score_this_level = 0
                        print(f"Max Level Reached! Final Score: {config.current_score}")
                        # Bisa tambahkan logika menang di sini
                    # Reset posisi bola untuk level baru atau setelah level maks
                    center_x_ball = 0
                    center_y_ball = actual_height // 2 - resized_ball_img.shape[0] // 2
                    config.ball_trail.clear() # Hapus jejak bola
# ...existing code...
```

Kode 12: Logika progres level

- Penanganan Tabrakan (`main.py`)
  Pada setiap frame permainan, dilakukan pengecekan tabrakan antara bola dan penghalang. Posisi vertikal penghalang atas dan bawah ditentukan berdasarkan pengaturan level saat ini. Jika tabrakan terdeteksi dan periode jeda setelah tabrakan sebelumnya sudah habis, maka status permainan akan diatur menjadi berakhir, skor akhir dihitung, dan sebuah efek visual untuk menandakan tabrakan diaktifkan. Jika periode jeda tabrakan masih aktif, durasinya akan dikurangi.

```
# ...existing code...
                # Dapatkan pengaturan penghalang untuk level saat ini
                top_barrier_factor, bottom_barrier_factor = config.level_barrier_settings[config.current_level]
                top_barrier_pos_y = int(actual_height * top_barrier_factor)
                bottom_barrier_pos_y = int(actual_height * bottom_barrier_factor)
```

```
                barrier_line_thickness = 3

                # Cek tabrakan dengan penghalang
                if collision_cooldown_timer <= 0: # Hanya cek jika tidak dalam cooldown
                    if game_logic.check_collision_with_barriers(
                        center_x_ball, center_y_ball, resized_ball_img.shape[1],
                        resized_ball_img.shape[0], top_barrier_pos_y, bottom_barrier_pos_y,
    barrier_line_thickness
                    ):
                        final_score_at_collision = config.current_score + config.
    score_this_level
                        print(f"Collision! Game Over. Final Score: {final_score_at_collision}
    ")
                        config.collision_flash = 30 # Aktifkan efek kilat tabrakan
                        config.game_over = True     # Set status game over
                else:
                    collision_cooldown_timer -= 1 # Kurangi timer cooldown
# ...existing code...
```

Kode 13: Logika penanganan tabrakan

- Batasan Posisi Bola (`main.py`)
  Untuk memastikan bola tetap berada dalam area permainan yang terlihat, posisinya dibatasi. Posisi vertikal bola dibatasi agar tidak melewati batas atas atau batas bawah layar. Demikian pula, posisi horizontal bola dibatasi agar tidak keluar dari sisi kiri atau sisi kanan area permainan.

```
# ...existing code...
                # Batasi posisi bola agar tetap di dalam layar game view
                center_y_ball = max(0, min(center_y_ball, actual_height - resized_ball_img.
    shape[0]))
# ...existing code...
                # Pastikan bola tidak keluar dari batas kiri game view setelah reset
                center_x_ball = max(0, min(center_x_ball, game_view_actual_width -
    resized_ball_img.shape[1]))
# ...existing code...
```

Kode 14: Logika batasan posisi bola

3.3.5 Penggambaran Visual dan UI

Modul ini bertanggung jawab untuk semua elemen visual yang dilihat pemain, mulai dari layar awal, antarmuka saat bermain, hingga layar game over, termasuk penggambaran bola, penghalang, dan efek-efek visual lainnya.

- `visuals_utils.py`
  File ini berisi fungsi-fungsi dasar yang membantu dalam proses penggambaran elemen-elemen grafis.
  - Fungsi `overlay_transparent` digunakan untuk menempatkan gambar yang memiliki transparansi, seperti gambar kacamata atau bola, di atas gambar latar belakang. Ia menghitung bagaimana piksel dari gambar overlay harus dicampur dengan piksel gambar latar belakang berdasarkan tingkat transparansi overlay.
    * Logika Penanganan Batas
      Sebelum melakukan blending, fungsi ini memastikan bahwa area overlay tidak keluar dari batas gambar latar belakang. Jika overlay sebagian berada di luar, ia akan dipotong (cropped) agar hanya bagian yang valid yang diproses.

```
# ...existing code...
    # Menangani jika overlay dimulai di luar batas kiri atau atas background
    if x < 0:
        overlay_crop_x_start = -x
        w_eff += x
        x = 0
    if y < 0:
        overlay_crop_y_start = -y
        h_eff += y
        y = 0

    if w_eff <= 0 or h_eff <= 0: # Jika lebar atau tinggi efektif menjadi nol atau negatif
        return background

    # Menangani jika overlay melewati batas kanan atau bawah background
    if x + w_eff > bw:
        w_eff = bw - x
    if y + h_eff > bh:
        h_eff = bh - y

    if w_eff <= 0 or h_eff <= 0: # Cek ulang setelah penyesuaian batas
        return background

    # Crop bagian overlay yang akan ditampilkan
    overlay_to_blend = overlay [
        int(overlay_crop_y_start) : int(overlay_crop_y_start + h_eff),
        int(overlay_crop_x_start) : int(overlay_crop_x_start + w_eff)
    ]
# ...existing code...
```

Kode 15: Logika penanganan batas

* Logika Blending Alpha
  Bagian inti dari fungsi ini adalah pencampuran warna piksel overlay dengan piksel background berdasarkan nilai alpha (transparansi) dari overlay. Rumus umumnya adalah: blended_color = overlay_color * alpha + background_color * (1 - alpha).

```
# ...existing code...
    # Proses blending
    alpha_overlay = overlay_to_blend[:, :, 3:] / 255.0 # Normalisasi alpha channel overlay
    color_overlay_pixels = overlay_to_blend[:, :, :3]  # Ambil channel warna (RGB) overlay

    bg_region_color = bg_region
    if bg_region.shape[2] == 4: # Jika background juga punya alpha, ambil RGB-nya saja
        bg_region_color = bg_region[:, :, :3]

    # Rumus blending alpha
    blended_color = color_overlay_pixels * alpha_overlay + bg_region_color * (1.0 - alpha_overlay)

    background[bg_y_start:bg_y_end, bg_x_start:bg_x_end, :3] = blended_color.astype(np.uint8)

    # Jika background memiliki alpha channel, update juga alpha channel-nya (opsional, tergantung kebutuhan)
    if background.shape[2] == 4:
        alpha_bg = bg_region[:, :, 3:] / 255.0 if bg_region.shape[2] == 4 else np.zeros_like(alpha_overlay)
        new_alpha_bg = alpha_overlay + alpha_bg * (1.0 - alpha_overlay)
```

```
        background[bg_y_start:bg_y_end, bg_x_start:bg_x_end, 3] = (new_alpha_bg * 255).astype(np.uint8)
# ...existing code...
```

Kode 16: Logika blending alpha

– Fungsi `draw_rounded_rectangle` digunakan untuk menggambar persegi panjang dengan sudut yang membulat, sering dipakai untuk membuat elemen UI yang terlihat lebih modern dan halus.

  * Logika Penggambaran Sudut dan Sisi (Outline)
    Jika tidak diisi (filled=False), fungsi ini menggambar empat garis lurus untuk sisi-sisi persegi panjang dan empat busur (menggunakan cv2.ellipse) untuk membuat sudut yang membulat.

```
# ...existing code...
    else: # Menggambar outline
        # Garis lurus
        cv2.line(img, (x1+radius, y1), (x2-radius, y1), color, thickness) # Atas
        cv2.line(img, (x1+radius, y2), (x2-radius, y2), color, thickness) # Bawah
        cv2.line(img, (x1, y1+radius), (x1, y2-radius), color, thickness) # Kiri
        cv2.line(img, (x2, y1+radius), (x2, y2-radius), color, thickness) # Kanan
        # Sudut (ellipse/arc)
        cv2.ellipse(img, (x1+radius, y1+radius), (radius, radius), 180, 0, 90, color, thickness) # Kiri Atas
        cv2.ellipse(img, (x2-radius, y1+radius), (radius, radius), 270, 0, 90, color, thickness) # Kanan Atas
        cv2.ellipse(img, (x1+radius, y2-radius), (radius, radius), 90, 0, 90, color, thickness)  # Kiri Bawah
        cv2.ellipse(img, (x2-radius, y2-radius), (radius, radius), 0, 0, 90, color, thickness)   # Kanan Bawah
# ...existing code...
```

Kode 17: Logika penggambaran sudut dan sisi

  * Logika Penggambaran Area Terisi (Filled)
    Jika filled=True, fungsi ini membuat sebuah mask terlebih dahulu. Mask ini diisi dengan menggambar dua persegi panjang di tengah dan empat lingkaran di sudut-sudut. Kemudian, area pada gambar utama yang sesuai dengan mask ini diisi dengan warna yang ditentukan.

```
# ...existing code...
if filled:
    # Membuat mask untuk area yang diisi
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    # Bagian tengah rectangle
    cv2.rectangle(mask, (x1+radius, y1), (x2-radius, y2), 255, -1)
    cv2.rectangle(mask, (x1, y1+radius), (x2, y2-radius), 255, -1)
    # Sudut-sudut tumpul (lingkaran)
    cv2.circle(mask, (x1+radius, y1+radius), radius, 255, -1)
    cv2.circle(mask, (x2-radius, y1+radius), radius, 255, -1)
    cv2.circle(mask, (x1+radius, y2-radius), radius, 255, -1)
    cv2.circle(mask, (x2-radius, y2-radius), radius, 255, -1)

    # Mengaplikasikan warna pada area yang di-mask
    # Perlu penanganan jika img adalah RGBA
    if img.shape[2] == 3:
        img[mask == 255] = color
    elif img.shape[2] == 4: # Jika RGBA, asumsikan color adalah (B,G,R) dan alpha 255
        img[mask == 255, :3] = color[:3]
        img[mask == 255, 3] = color[3] if len(color) == 4 else 255
```

```
# ...existing code...
```

Kode 18: Logika penggambaran area terisi

– Fungsi `draw_gradient_panel`
Fungsi ini membuat panel dengan efek gradasi warna

* Logika Pembuatan Gradasi dan Penggambaran
Gradasi dibuat secara linear dari satu warna ke warna lain sepanjang sumbu vertikal panel. Ini dicapai dengan menggunakan np.linspace untuk menghasilkan serangkaian nilai antara 0 dan 1, yang kemudian digunakan untuk menginterpolasi antara warna awal dan warna akhir. Setiap baris piksel dalam panel kemudian digambar dengan warna hasil interpolasi tersebut.

```
# ...existing code...
    # Membuat gradien linear dari color1 ke color2
    gradient = np.linspace(0, 1, y2-y1).reshape(-1, 1) # Array dari 0 ke 1 sejumlah tinggi panel
    gradient_rgb = gradient * np.array(color2) + (1-gradient) * np.array(color1) # Interpolasi warna

    # Menggambar garis horizontal dengan warna gradien
    for i in range(y2-y1):
        color = tuple(map(int, gradient_rgb[i])) # Konversi warna ke tuple integer
        cv2.rectangle(overlay, (x1, y1+i), (x2, y1+i+1), color, -1) # Gambar garis setebal 1px
# ...existing code...
```

Kode 19: Logika pembuatan gradasi dan penggambaran

* Logika Transparansi
Setelah panel gradasi digambar pada gambar overlay sementara, cv2.addWeighted digunakan untuk mencampurkannya dengan gambar img asli, dengan mempertimbangkan nilai alpha yang diberikan untuk transparansi.

```
# ...existing code...
    # Mengaplikasikan transparansi
    cv2.addWeighted(overlay, alpha, img, 1-alpha, 0, img)
# ...existing code...
```

Kode 20: Logika transparansi

– Fungsi `draw_glassmorphism_panel`
Menciptakan efek panel seperti kaca buram (glassmorphism).

* Logika Ekstraksi dan Pemburaman Latar Belakang
Bagian dari gambar utama yang akan menjadi latar belakang panel diekstrak. Kemudian, efek blur Gaussian diterapkan pada region ini. Kekuatan blur disesuaikan agar tidak menyebabkan error jika region terlalu kecil.

```
# ...existing code...
    bg_region = img[y1:y2, x1:x2].copy() # Ekstrak region latar belakang

    # Sesuaikan kekuatan blur jika region terlalu kecil
    # ... (penyesuaian blur_strength) ...

    if bg_region.shape[0] > 0 and bg_region.shape[1] > 0: # Hanya proses jika region valid
        blurred = cv2.GaussianBlur(bg_region, (blur_strength, blur_strength), 0) # Aplikasikan blur
# ...existing code...
```

Kode 21: Logika ekstraksi dan pemburaman latar belakang

* Logika Pembuatan Overlay Kaca dan Blending
  Sebuah overlay dengan warna dasar (agak gelap dengan sedikit tint biru-abu) dibuat. Overlay ini kemudian dicampur dengan latar belakang yang sudah diburamkan menggunakan cv2.addWeighted untuk menciptakan efek kaca. Hasilnya kemudian ditempatkan kembali ke gambar utama.

```
# ...existing code...
        # Membuat overlay putih semi-transparan untuk efek kaca
        overlay_color = np.ones_like(blurred) * 40 # Base color (gelap)
        overlay_color[:, :, 0] = 60  # Sedikit tint biru-abu
        overlay_color[:, :, 1] = 60
        overlay_color[:, :, 2] = 80

        # Blend overlay dengan background yang sudah diblur
        glass_effect = cv2.addWeighted(blurred, 1-alpha, overlay_color, alpha, 0)

        img[y1:y2, x1:x2] = glass_effect # Timpa region di gambar utama
# ...existing code...
```

Kode 22: Logika pembuatan overlay kaca dan blending

* Logika Penambahan Border dan Inner Glow
  Untuk memperjelas panel, sebuah border tipis digambar di sekelilingnya. Efek "inner glow" (cahaya di dalam border) juga ditambahkan dengan menggambar border lain yang lebih tipis di bagian dalam dan mencampurkannya dengan gambar panel dengan alpha rendah.

```
# ...existing code...
        # Tambahkan border tipis untuk memperjelas panel
        border_color_val = color_schemes["primary"] # Ambil dari config
        cv2.rectangle(img, (x1, y1), (x2, y2), border_color_val, 2)

        # Tambahkan inner glow (efek cahaya di dalam border)
        inner_overlay = img.copy()
        cv2.rectangle(inner_overlay, (x1+2, y1+2), (x2-2, y2-2), border_color_val,
    1) # Border lebih tipis di dalam
        cv2.addWeighted(img, 0.9, inner_overlay, 0.1, 0, img) # Blend dengan alpha
    rendah
# ...existing code...
```

Kode 23: Logika penambahan border dan inner glow

- `visuals.py`
  File ini menggunakan utilitas dari `visuals_utils.py` untuk membuat komponen visual filter.

  – Fungsi `draw_start_screen`
  Fungsi ini bertanggung jawab untuk merender tampilan layar awal permainan. Ini mencakup latar belakang animasi, panel utama dengan efek glassmorphism, judul permainan dengan efek glow yang berdenyut, dan daftar instruksi cara bermain.

  * Panel Utama dan Judul dengan Efek Glow
    Panel utama digambar menggunakan visuals_utils.draw_glassmorphism_panel. Judul "SUARAKU TERBANG" kemudian ditampilkan di atas panel ini. Untuk memberikan efek visual yang menarik, judul memiliki efek glow (cahaya) yang intensitasnya beranimasi (berdenyut) menggunakan fungsi sinus dari waktu. Efek glow ini dicapai dengan menggambar teks beberapa kali dengan ketebalan dan transparansi yang sedikit berbeda.

```
# ...existing code...
def draw_start_screen(width, height):
    """Membuat gambar untuk layar awal (start screen) game."""
```

```
    start_img = np.zeros((height, width, 3), dtype=np.uint8) # Latar hitam

    draw_animated_background(start_img, width, height) # Latar belakang animasi

    # Panel utama start screen
    panel_width_start = min(500, width - 40)
    panel_height_start = min(400, height - 40) # Sesuaikan tinggi jika perlu
    panel_x_start = (width - panel_width_start) // 2
    panel_y_start = (height - panel_height_start) // 2

    if panel_width_start > 10 and panel_height_start > 10: # Pastikan panel valid
        visuals_utils.draw_glassmorphism_panel(start_img, (panel_x_start, panel_y_start),
                                               (panel_x_start + panel_width_start, panel_y_start + panel_height_start),
                                               blur_strength=15, alpha=0.4)

    # Judul Game
    title_y_start = panel_y_start + 80 # Naikkan sedikit judul
    title_text_start = "SUARAKU TERBANG"
    (w_title, h_title), _ = cv2.getTextSize(title_text_start, cv2.FONT_HERSHEY_SIMPLEX, 1.5, 3)

    # Efek glow animasi untuk judul
    glow_intensity_start = abs(np.sin(time.time() * 2)) # Intensitas glow berdenyut
    for i in range(3): # Beberapa layer glow
        glow_alpha_start = (0.5 - i * 0.1) * glow_intensity_start
        overlay_title = start_img.copy()
        cv2.putText(overlay_title, title_text_start, (panel_x_start + (panel_width_start - w_title) // 2, title_y_start),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.5, config.color_schemes["primary"], 4 + i*2) # Glow lebih tebal
        cv2.addWeighted(start_img, 1-glow_alpha_start, overlay_title, glow_alpha_start, 0, start_img)

    cv2.putText(start_img, title_text_start, (panel_x_start + (panel_width_start - w_title) // 2, title_y_start),
                cv2.FONT_HERSHEY_SIMPLEX, 1.5, config.color_schemes["text_primary"], 3)

    # Daftar Instruksi
    instructions_start = [
        "Voice-Controlled Ball Game",
        "", # Spasi
        "How to play:",
        "    High pitch sounds = Move UP",
        "    Low pitch sounds = Move DOWN",
        "    Navigate through barriers",
        "    Reach the end to level up!",
        "    Avoid hitting barriers to keep score",
        "", # Spasi
        "Press SPACE to start"
    ]

    instruction_y_start_offset = title_y_start + h_title + 20 # Mulai instruksi di bawah judul
    line_height_instr = 25 # Jarak antar baris instruksi
    for instr_line in instructions_start:
        if instr_line: # Jika baris tidak kosong
            font_scale_instr = 0.6 if "Voice-Controlled" in instr_line else 0.5
            text_color_instr = config.color_schemes["text_primary"] if "Voice-Controlled" in instr_line else config.color_schemes["text_secondary"]
```

```
            thickness_instr = 2 if "Voice-Controlled" in instr_line else 1
            (w_instr_line, _), _ = cv2.getTextSize(instr_line, cv2.FONT_HERSHEY_SIMPLEX, font_scale_instr, thickness_instr)
            cv2.putText(start_img, instr_line, (panel_x_start + (panel_width_start - w_instr_line) // 2, instruction_y_start_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, font_scale_instr, text_color_instr, thickness_instr)
            instruction_y_start_offset += line_height_instr # Pindah ke baris berikutnya

    return start_img
# ...existing code...
```

Kode 24: Panel utama dan judul

– Fungsi `draw_split_interface`
Fungsi ini mengatur tata letak utama selama permainan berlangsung. Layar dibagi menjadi dua bagian: area tampilan game di sisi kiri dan panel informasi di sisi kanan. Panel informasi menampilkan berbagai data seperti frekuensi suara, skor, level, dan kontrol audio. Fungsi ini juga menangani efek visual kilatan pada area game saat terjadi tabrakan atau pemain naik level, serta memicu penambahan partikel.

```
def draw_split_interface(img, width, height, fps, score, level, sound_info, sound_direction, collision_flash, level_up_flash):
    """Menggambar antarmuka terpisah: tampilan game di kiri, panel info di kanan."""
    info_panel_width_split = 250  # Lebar panel informasi, hindari konflik nama
    game_view_width_split = width - info_panel_width_split # Lebar area game

    # Latar belakang panel informasi (sisi kanan)
    info_bg_color_split = (15, 15, 25) # Dari config atau spesifik
    cv2.rectangle(img, (game_view_width_split, 0), (width, height), info_bg_color_split, -1)

    # Garis pemisah
    cv2.line(img, (game_view_width_split, 0), (game_view_width_split, height), config.color_schemes["primary"], 3)

    # Pengaturan panel di sisi kanan
    panel_x_split = game_view_width_split + 5
    panel_width_split_val = info_panel_width_split - 10 # Lebar efektif panel

    # Panel Tampilan Frekuensi (atas)
    freq_height_split = 110
    draw_frequency_display(img, panel_x_split, 10, panel_width_split_val, freq_height_split, sound_info)

    # Panel Info Game (tengah)
    game_info_y_split = freq_height_split + 15
    game_info_height_split = 160
    draw_game_info_panel(img, panel_x_split, game_info_y_split, panel_width_split_val, game_info_height_split, score, level, fps)

    # Panel Kontrol Audio (bawah)
    audio_y_split = game_info_y_split + game_info_height_split + 10
    audio_height_split = height - audio_y_split - 10 # Sisa tinggi untuk panel audio
    draw_audio_control_panel(img, panel_x_split, audio_y_split, panel_width_split_val, audio_height_split, sound_info, sound_direction)

    # Efek kilat hanya pada area game view
    if collision_flash > 0:
        flash_intensity_split = collision_flash / 30.0
        # Buat overlay hanya untuk area game
```

```
        flash_overlay_split = np.zeros((height, game_view_width_split, 3), dtype=np.uint8)
        flash_overlay_split[:] = config.color_schemes["danger"]
        # Ambil slice area game dari img untuk blending
        game_area_slice = img[:, :game_view_width_split]
        cv2.addWeighted(game_area_slice, 1 - flash_intensity_split * 0.3, flash_overlay_split, flash_intensity_split * 0.3, 0, game_area_slice)

        if collision_flash == 30: # Saat pertama kali tabrakan
            particles.add_particles(game_view_width_split//2, height//2, config.color_schemes["danger"], 20)

    if level_up_flash > 0:
        flash_intensity_split = level_up_flash / 60.0
        flash_overlay_split = np.zeros((height, game_view_width_split, 3), dtype=np.uint8)
        flash_overlay_split[:] = config.color_schemes["success"]
        game_area_slice = img[:, :game_view_width_split]
        cv2.addWeighted(game_area_slice, 1 - flash_intensity_split * 0.2, flash_overlay_split, flash_intensity_split * 0.2, 0, game_area_slice)

        if level_up_flash == 60: # Saat pertama kali naik level
            particles.add_particles(game_view_width_split//2, 100, config.color_schemes["success"], 30)

    return game_view_width_split # Kembalikan lebar area game untuk logika lain
# ...existing code...
```

Kode 25: Fungsi draw_split_interface

– Fungsi `draw_modern_barriers`
Fungsi ini menggambar penghalang atas dan bawah di area permainan. Penghalang digambar dengan garis utama berwarna solid dan diberi efek glow (cahaya) di sekelilingnya untuk tampilan yang lebih modern. Efek glow dicapai dengan menggambar beberapa garis yang lebih tebal dan semi-transparan di belakang garis utama.

```
# ...existing code...
def draw_modern_barriers(img, width, height, top_y, bottom_y, thickness):
    """Menggambar penghalang (barriers) dengan gaya modern dan efek glow."""
    barrier_color = config.color_schemes["primary"]
    glow_color = tuple(c // 2 for c in barrier_color) # Warna glow lebih gelap

    # Efek glow untuk barrier
    for i in range(3): # Beberapa layer glow
        glow_thickness = thickness + (3-i) * 2 # Ketebalan glow berkurang
        alpha = 0.3 - i * 0.1 # Transparansi glow berkurang
        overlay = img.copy()
        # Gambar garis glow
        cv2.line(overlay, (0, top_y), (width, top_y), glow_color, glow_thickness)
        cv2.line(overlay, (0, bottom_y), (width, bottom_y), glow_color, glow_thickness)
        cv2.addWeighted(img, 1-alpha, overlay, alpha, 0, img) # Blend glow ke gambar utama

    # Gambar garis barrier utama
    cv2.line(img, (0, top_y), (width, top_y), barrier_color, thickness)
    cv2.line(img, (0, bottom_y), (width, bottom_y), barrier_color, thickness)
# ...existing code...
```

Kode 26: Fungsi draw_modern_barriers

– Fungsi `add_ball_trail` dan `draw_ball_trail`
add_ball_trail menyimpan histori posisi bola dalam sebuah list. draw_ball_trail kemudian

menggunakan histori ini untuk menggambar jejak di belakang bola. Jejak digambar sebagai serangkaian garis yang menghubungkan posisi-posisi sebelumnya, dengan alpha (transparansi) dan ketebalan yang berkurang untuk segmen jejak yang lebih tua, menciptakan efek memudar.

```
# ...existing code...
def add_ball_trail(x, y):
    """Menambahkan posisi bola saat ini ke list jejak bola (config.ball_trail)."""
    config.ball_trail.append((x, y))
    # Jika jejak terlalu panjang, hapus elemen tertua
    if len(config.ball_trail) > config.max_trail_length:
        config.ball_trail.pop(0)

def draw_ball_trail(img):
    """Menggambar jejak bola berdasarkan posisi yang tersimpan di config.ball_trail."""
    if len(config.ball_trail) < 2: # Perlu minimal 2 titik untuk menggambar garis
        return

    for i in range(1, len(config.ball_trail)):
        # Alpha dan ketebalan berkurang untuk segmen jejak yang lebih tua
        alpha_trail = i / len(config.ball_trail)
        thickness_trail = int(3 * alpha_trail) # Ketebalan maksimal 3px

        if thickness_trail > 0:
            # Warna jejak dengan alpha yang disesuaikan
            trail_color_base = config.color_schemes["accent"]
            # Membuat warna dengan alpha (jika gambar mendukung RGBA, ini bisa lebih
    baik)
            # Untuk BGR, kita bisa mencoba memudarkan warnanya
            current_trail_color = tuple(int(c * alpha_trail) for c in trail_color_base)

            pt1 = config.ball_trail[i-1]
            pt2 = config.ball_trail[i]

            start_point = (int(pt1[0]), int(pt1[1]))
            end_point = (int(pt2[0]), int(pt2[1]))

            cv2.line(img, start_point, end_point, current_trail_color, thickness_trail)
# ...existing code...
```

Kode 27: Fungsi add_ball$_d$*andraw_ball_trail*

– Fungsi `draw_modern_game_over`
Fungsi ini merender layar yang muncul ketika permainan berakhir. Layar ini memiliki latar belakang gradien dan panel utama yang juga bergradien dengan border membulat. Teks "GAME OVER" ditampilkan dengan efek glow. Skor akhir pemain dan instruksi untuk keluar juga ditampilkan.

```
# ...existing code...
def draw_modern_game_over(width, height, final_score, max_score_achieved):
    """Membuat gambar untuk layar Game Over dengan skor akhir."""
    game_over_img = np.zeros((height, width, 3), dtype=np.uint8) # Latar belakang hitam

    # Latar belakang gradien
    visuals_utils.draw_gradient_panel(game_over_img, (0, 0), (width, height), (20, 20,
    30), (40, 40, 60), 1.0)

    # Panel utama Game Over
    panel_width_go = 400 # Hindari konflik nama
    panel_height_go = 300
    panel_x_go = (width - panel_width_go) // 2
    panel_y_go = (height - panel_height_go) // 2

```

```
    visuals_utils.draw_gradient_panel(game_over_img, (panel_x_go, panel_y_go), (panel_x_go + panel_width_go, panel_y_go + panel_height_go),
                                      (60, 60, 80), (40, 40, 60), 0.95)
    visuals_utils.draw_rounded_rectangle(game_over_img, (panel_x_go, panel_y_go), (panel_x_go + panel_width_go, panel_y_go + panel_height_go),
                                         config.color_schemes["text_secondary"], 3, 15)

    # Teks "GAME OVER"
    text_center_x_go = panel_x_go + panel_width_go // 2
    text_y_go = panel_y_go + 80

    # Efek glow untuk judul
    title_text_go = "GAME OVER"
    (w_text, h_text), _ = cv2.getTextSize(title_text_go, cv2.FONT_HERSHEY_SIMPLEX, 2, 3)
    for i in range(3): # Beberapa layer glow
        glow_alpha = 0.4 - i*0.1
        overlay = game_over_img.copy()
        cv2.putText(overlay, title_text_go, (text_center_x_go - w_text//2, text_y_go),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, config.color_schemes["danger"], 5 + i*2) # Glow lebih tebal
        cv2.addWeighted(game_over_img, 1-glow_alpha, overlay, glow_alpha, 0, game_over_img)

    cv2.putText(game_over_img, title_text_go, (text_center_x_go - w_text//2, text_y_go),
                cv2.FONT_HERSHEY_SIMPLEX, 2, config.color_schemes["text_primary"], 3)

    # Tampilan Skor Akhir
    score_y_go = text_y_go + 80
    score_text = f"Final Score: {final_score}"
    (w_score_text, _), _ = cv2.getTextSize(score_text, cv2.FONT_HERSHEY_SIMPLEX, 1, 2)
    cv2.putText(game_over_img, score_text, (text_center_x_go - w_score_text//2, score_y_go),
                cv2.FONT_HERSHEY_SIMPLEX, 1, config.color_schemes["success"], 2)

    # Instruksi keluar
    instruction_y_go = score_y_go + 60
    instr_text = "Press any key to exit"
    (w_instr_text, _), _ = cv2.getTextSize(instr_text, cv2.FONT_HERSHEY_SIMPLEX, 0.8, 2)
    cv2.putText(game_over_img, instr_text, (text_center_x_go - w_instr_text//2, instruction_y_go),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, config.color_schemes["text_secondary"], 2)

    return game_over_img
# ...existing code...
```

Kode 28: Fungsi draw_modern_game_over

– Fungsi `draw_game_view_overlay`
Fungsi ini digunakan untuk menggambar informasi tambahan atau elemen UI minimalis langsung di atas area tampilan game. Contohnya adalah menampilkan nomor level saat ini di pojok kiri atas dan instruksi singkat cara bermain di bagian bawah area game. Teks ini diberi latar belakang semi-transparan agar lebih mudah terbaca.

```
# ...existing code...
def draw_game_view_overlay(img, game_view_width, height, current_level_val):
    """Menggambar overlay minimal pada tampilan game (indikator level, instruksi)."""
    # Indikator Level (kiri atas area game)
    level_text_overlay = f"LEVEL {current_level_val}"
    (w_level_text, h_level_text), _ = cv2.getTextSize(level_text_overlay, cv2.FONT_HERSHEY_SIMPLEX, 0.8, 2)

    # Latar belakang semi-transparan untuk teks level
```

```
    overlay_level_bg = img.copy() # Salin bagian gambar yang akan ditimpa
    cv2.rectangle(overlay_level_bg, (10, 5), (10 + w_level_text + 20, 5 + h_level_text +
     10), (0, 0, 0), -1) # Background hitam
    cv2.addWeighted(img, 0.7, overlay_level_bg, 0.3, 0, img) # Blend dengan alpha
    cv2.putText(img, level_text_overlay, (20, 5 + h_level_text + 5), cv2.
    FONT_HERSHEY_SIMPLEX, 0.8, config.color_schemes["text_primary"], 2)

    # Instruksi (bawah area game)
    instruction_y_overlay = height - 60 # Posisi Y instruksi
    instructions_list = [
        "Use your voice to control the ball",
        "High pitch = UP, Low pitch = DOWN"
    ]

    # Hitung lebar maksimum teks instruksi untuk background
    max_instr_width = 0
    instr_line_height = 0
    for instr in instructions_list:
        (w_instr, h_instr), _ = cv2.getTextSize(instr, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
        max_instr_width = max(max_instr_width, w_instr)
        if instr_line_height == 0: instr_line_height = h_instr + 5 # Ambil tinggi satu
    baris + padding

    # Latar belakang semi-transparan untuk instruksi
    overlay_instr_bg = img.copy()
    bg_instr_y_start = instruction_y_overlay - instr_line_height // 2 # Sesuaikan agar
    teks di tengah background
    bg_instr_y_end = bg_instr_y_start + len(instructions_list) * instr_line_height + 10
    cv2.rectangle(overlay_instr_bg, (10, bg_instr_y_start), (10 + max_instr_width + 20,
    bg_instr_y_end), (0, 0, 0), -1)
    cv2.addWeighted(img, 0.7, overlay_instr_bg, 0.3, 0, img)

    # Gambar teks instruksi
    for i, instr in enumerate(instructions_list):
        cv2.putText(img, instr, (20, instruction_y_overlay + i * instr_line_height), cv2
    .FONT_HERSHEY_SIMPLEX, 0.5, config.color_schemes["text_secondary"], 1)
# ...existing code...
```

Kode 29: Fungsi draw_game_view_overlay

3.3.6 Sistem Partikel `particles.py`

Sistem partikel dirancang untuk menciptakan efek visual dinamis. Ini dicapai dengan mengelola objek-objek individual yang disebut partikel.

- Representasi Partikel (`Class Particle`)
  Setiap partikel dalam sistem direpresentasikan sebagai sebuah instance dari Class Particle. Class ini mendefinisikan atribut-atribut yang menentukan keadaan dan penampilan setiap partikel:
  - x, y: Menyatakan koordinat posisi partikel pada layar.
  - vx, vy: Menyatakan komponen kecepatan horizontal dan vertikal partikel, yang menentukan arah dan laju pergerakannya.
  - color: Menyimpan tuple BGR yang merepresentasikan warna partikel.
  - size: Menyatakan ukuran awal partikel.
  - life: Menyatakan sisa masa hidup partikel dalam satuan frame. Partikel akan hilang setelah life mencapai nol.
  - max_life: Menyimpan nilai masa hidup awal partikel, yang digunakan untuk perhitungan efek visual seperti perubahan ukuran atau transparansi seiring waktu.

```python
# ...existing code...
class Particle:
    """Kelas untuk merepresentasikan satu partikel dengan posisi, kecepatan, warna, ukuran, dan masa hidup."""
    def __init__(self, x, y, vx, vy, color, size, life):
        self.x = x          # Posisi x
        self.y = y          # Posisi y
        self.vx = vx        # Kecepatan horizontal
        self.vy = vy        # Kecepatan vertikal
        self.color = color  # Warna partikel (BGR tuple)
        self.size = size    # Ukuran awal partikel
        self.life = life    # Masa hidup partikel (dalam frame)
        self.max_life = life # Masa hidup maksimum untuk perhitungan alpha
# ...existing code...
```

Kode 30: Class Particle

- Pembaruan Status Partikel (Method `update` dalam `Class Particle`)
  Method update dipanggil untuk setiap partikel pada setiap frame. Metode ini melakukan:
  - Memperbarui posisi partikel (self.x, self.y) berdasarkan kecepatannya (self.vx, self.vy).
  - Menerapkan efek gravitasi sederhana dengan menambahkan nilai konstan (0.1) ke kecepatan vertikal (self.vy), membuat partikel cenderung jatuh ke bawah.
  - Mengurangi masa hidup partikel (self.life) sebanyak satu unit.

```python
# ...existing code...
    def update(self):
        """Memperbarui posisi dan masa hidup partikel."""
        self.x += self.vx
        self.y += self.vy
        self.vy += 0.1  # Efek gravitasi sederhana
        self.life -= 1  # Kurangi masa hidup
# ...existing code...
```

Kode 31: Method update

- Pengecekan Status Hidup Partikel (Method is_alive dalam `Class Particle`)
  Method is_alive mengembalikan nilai boolean yang menunjukkan apakah partikel masih aktif. Partikel dianggap hidup jika nilai self.life lebih besar dari nol.

```python
# ...existing code...
    def is_alive(self):
        """Mengembalikan True jika partikel masih hidup, False jika tidak."""
        return self.life > 0
# ...existing code...
```

Kode 32: Method is_alive

- Penggambaran Partikel (Method draw dalam `Class Particle`)
  Method draw bertanggung jawab untuk menggambar partikel pada gambar (img) yang diberikan, tetapi hanya jika partikel tersebut masih hidup (self.is_alive() mengembalikan True).
  - Ukuran partikel saat digambar (current_size) dihitung secara proporsional terhadap sisa masa hidupnya relatif terhadap masa hidup awalnya (self.life / self.max_life), sehingga partikel akan tampak mengecil seiring waktu.
  - Partikel digambar sebagai lingkaran yang diisi (cv2.circle dengan parameter ketebalan -1) menggunakan warna dan ukuran yang telah dihitung.

```
# ...existing code...
    def draw(self, img):
        """Menggambar partikel pada gambar 'img' jika masih hidup."""
        if self.is_alive():
            # Ukuran dan alpha partikel berkurang seiring berkurangnya masa hidup
            alpha = self.life / self.max_life
            current_size = int(self.size * alpha) # Ukuran mengecil

            if current_size > 0:
                # Untuk menggambar dengan transparansi alpha, perlu blending manual jika tidak menggunakan RGBA image
                # Di sini, kita hanya menggambar lingkaran solid dengan ukuran yang disesuaikan
                # Jika 'img' adalah RGBA, blending bisa lebih canggih
                cv2.circle(img, (int(self.x), int(self.y)), current_size, self.color, -1)
# ...existing code...
```

Kode 33: Method draw

- Menambahkan Sekumpulan Partikel Baru (Fungsi `add_particles`)
  Fungsi add_particles digunakan untuk memunculkan sejumlah (count) partikel baru pada posisi (x, y) dengan warna (color) tertentu.
  Untuk setiap partikel yang dibuat:
  - Kecepatan horizontal (vx) diinisialisasi secara acak dalam rentang -3 hingga 3.
  - Kecepatan vertikal (vy) diinisialisasi secara acak dalam rentang -5 hingga -1, memberikan efek semburan awal ke atas.
  - Ukuran (size) diinisialisasi secara acak antara 2 hingga 5 piksel.
  - Masa hidup (life) diinisialisasi secara acak antara 20 hingga 40 frame.

  Partikel-partikel baru ini kemudian ditambahkan ke list global config.particles.

```
# ...existing code...
def add_particles(x, y, color, count=10):
    """Menambahkan sejumlah 'count' partikel baru ke list global config.particles."""
    for _ in range(count):
        # Kecepatan acak
        vx = np.random.uniform(-3, 3)
        vy = np.random.uniform(-5, -1) # Awalnya bergerak ke atas
        # Ukuran dan masa hidup acak
        size = np.random.randint(2, 6)
        life = np.random.randint(20, 40) # Masa hidup dalam frame
        config.particles.append(Particle(x, y, vx, vy, color, size, life))
# ...existing code...
```

Kode 34: Fungsi add_particles

- Memperbarui Semua Partikel Aktif (Fungsi `update_particles`)
  Fungsi update_particles dipanggil pada setiap frame untuk mengelola seluruh kumpulan partikel. Pertama, ia membuat list baru config.particles yang hanya berisi partikel-partikel yang masih hidup (menggunakan list comprehension dan metode is_alive). Ini secara efektif menghapus partikel yang sudah mati. Kemudian, ia mengiterasi melalui partikel-partikel yang masih hidup tersebut dan memanggil metode update() pada masing-masing partikel.

```
# ...existing code...
def update_particles():
    """Memperbarui semua partikel dalam list global dan menghapus yang sudah mati."""
    # Buat list baru yang hanya berisi partikel yang masih hidup
    config.particles = [p for p in config.particles if p.is_alive()]
```

```
    # Update setiap partikel yang masih hidup
    for particle in config.particles:
        particle.update()
# ...existing code...
```

Kode 35: Fungsi update_particles

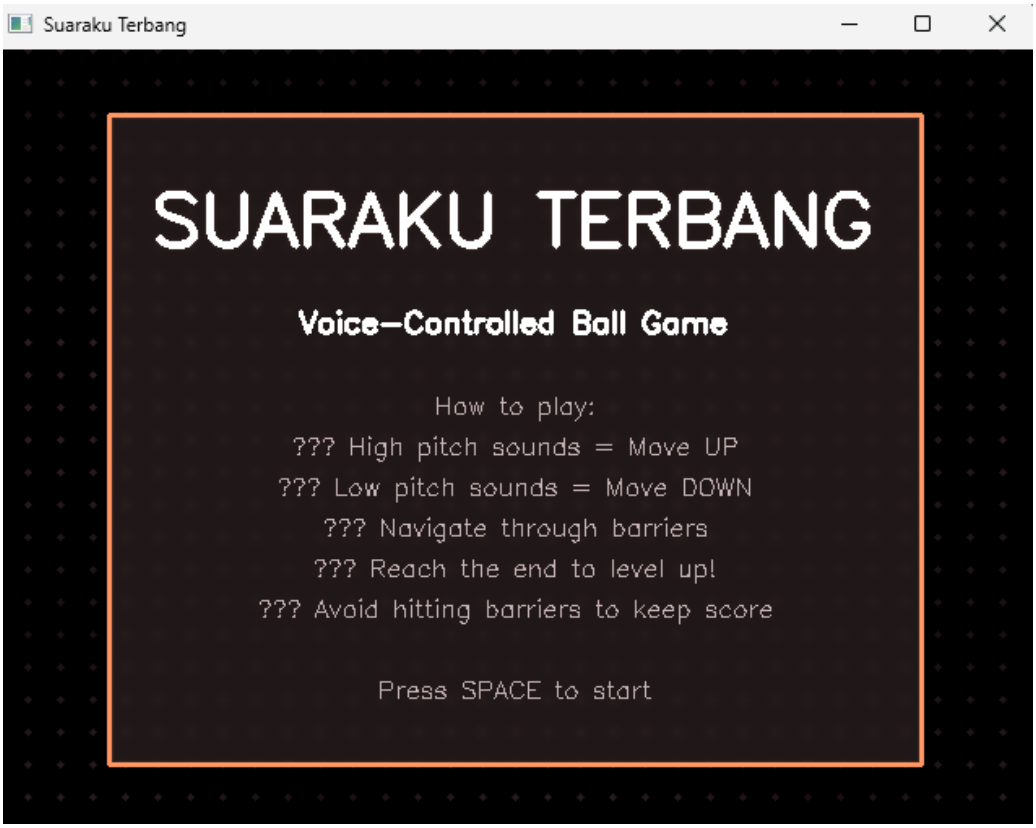
- Menggambar Semua Partikel Aktif (Fungsi `draw_particles`)
  Fungsi draw_particles dipanggil pada setiap frame untuk merender semua partikel yang aktif ke gambar (img) yang diberikan. Ia mengiterasi melalui list config.particles dan memanggil metode draw(img) pada setiap partikel.

```
# ...existing code...
def draw_particles(img):
    """Menggambar semua partikel aktif pada gambar 'img'."""
    for particle in config.particles:
        particle.draw(img)
# ...existing code...
```

Kode 36: Fungsi draw_particles

4 Hasil Program

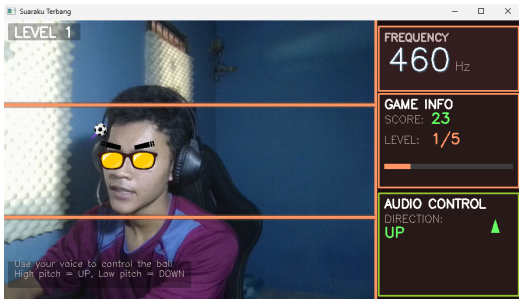
Berdasarkan hasil implementasi, program berhasil mendeteksi suara bass dan treble pengguna serta menggerakkan bola berdasarkan suara tersebut.



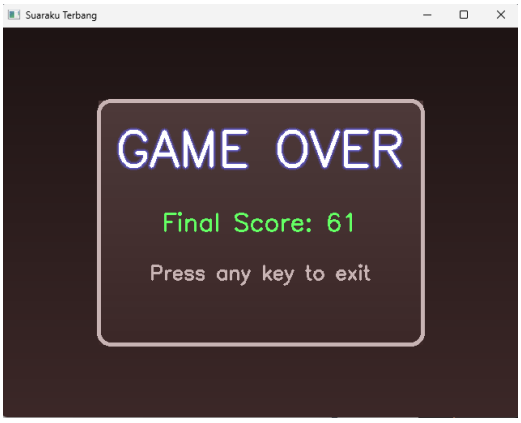
Gambar 1: Layar awal

Ketika pengguna membuka filter untuk pertama kali maka akan ditampilkan judul atau nama dari filter yaitu "Suaraku Terbang". Selain itu, ditampilkan juga informasi singkat untuk menjalankan filter, yaitu

jika pengguna mengeluarkan suara bernada tinggi (treble) maka bola akan bergerak ke atas, sedangkan jika pengguna mengeluarkan suara bernada rendah (bass) maka bola akan bergerak ke bawah.



(a) Antarmuka utama



(b) Game over

Gambar 2: Demonstrasi filter suaraku terbang

Antarmuka utama selama menjalankan filter dibagi menjadi 2 kolom. Kolom di sisi kiri layar terdapat video real-time dari webcam serta permainan bola dan penghalangnya yang menjadi fitur utama filter serta di wajah pengguna terdapat efek kacamata. Sedangkan di sisi kanan layar terdapat panel informasi yang menampilkan data frekuensi suara pengguna, jumlah skor, level saat ini, serta arah bola saat ini. Ketika pengguna berhasil menggerakkan bola dari ujung kiri ke ujung kanan, maka pengguna akan mendapatkan level baru dengan jarak kedua penghalang yang lebih kecil. Jaraknya akan terus mengecil hingga maksimal pengguna mencapai level 5.
Ketika bola bergerak dan menyentuh penghalang atas atau bawah, maka permainan berakhir. Pengguna diberikan layar akhir berupa judul "Game Over" dan skor yang telah diraih oleh pengguna.
Untuk demonstrasi lebih lengkapnya dapat dilihat di video youtube melalui link berikut: Suaraku-Terbang

5 Kesimpulan

Proyek "Suaraku Terbang" berhasil dikembangkan dengan memanfaatkan frekuensi suara sebagai mekanisme kontrol utama yang terinspirasi dari konsep dasar permainan Flappy Bird. Dengan menggunakan bahasa pemrograman Python dan library seperti `sounddevice` untuk akuisisi audio secara real-time, `numpy` untuk analisis frekuensi, `OpenCV` untuk visualisasi dan pemrosesan gambar, serta `time` dan `threading` untuk manajemen waktu. Tujuan dari proyek ini adalah menggerakkan objek berbentuk bola secara vertikal (naik untuk frekuensi tinggi dan turun untuk frekuensi rendah) sebagai respons terhadap frekuensi suara, terdapat implementasi pembatas atas dan bawah yang menyempit seiring peningkatan level, serta kondisi kalah ketika bola menyentuh pembatas.

5.1 Saran Pengembangan

1. Menambahkan fitur kalibrasi agar sistem dapat menyesuaikan sensitivitasnya terhadap karakteristik vokal pengguna yang berbeda, dan kondisi kebisingan lingkungan. Ini akan meningkatkan akurasi kontrol dan aksesibilitas.

2. Penambahan variasi rintangan seperti rintangan bergerak, rintangan yang muncul secara acak, atau zona khusus yang memberikan efek tertentu pada bola.

3. Menambahkan kontrol berbasis amplitudo suara, sensitivitas gerakan bola bisa dipengaruhi oleh amplitudo (kenyaringan) suara. Misalnya, suara frekuensi tinggi yang lebih nyaring membuat bola naik lebih cepat.

4. Meningkatkan eksplorasi untuk mediapipe untuk visualisasi spektrum suara secara real-time atau fitur interaktif lainnya yang relevan dengan multimedia.

References

6 Lampiran

Selama pengerjaan proyek "Suaraku Terbang" ini, kelompok kami juga memanfaatkan teknologi AI seperti Copilot. AI membantu dalam mengentahui dan memperbaiki error yang terjadi dalam penyusunan code, serta dalam penyusunana laporan agar lebih tersusun dengan baik. Beberapa link percakapan AI dalam bentuk Screenshot pada pengembangan proyek ini :

1. Screenshot

Link Github : GitHub