



-
1. **Lucky Immanuel S [122140179]**

Nama: 2. **Rachel Olivia Manullang [122140181]**
2. **Eric Arwido Damanik [122140157]**

Tugas Ke: **Tugas Besar**

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

Tanggal: 09 Desember 2025

1 Deskripsi Proyek

Proyek ini dirancang untuk menunjukkan bagaimana teknologi pemrosesan visual dapat dikombinasikan dengan elemen permainan (game mechanics) untuk menghasilkan bentuk interaksi yang bersifat natural, intuitif, dan real-time. Dalam konteks multimedia, aplikasi ini tidak hanya mengolah gambar secara statis, melainkan memanfaatkan aliran data visual langsung dari kamera, kemudian memprosesnya menjadi informasi bermakna yang digunakan sebagai input permainan.

Permainan ini menggunakan teknologi *Computer Vision* untuk mendeteksi wajah dan kedipan mata sebagai metode interaksi utama. Pemain harus mengedipkan mata pada waktu yang tepat untuk menghentikan bagian-bagian wajah yang jatuh dari sisi atas layar dan menempelkannya pada posisi landmark wajah mereka sendiri. Dengan demikian, FaceFit Blink Challenge mengintegrasikan tiga elemen multimedia inti secara bersamaan: citra real-time, interaksi berbasis ekspresi wajah, dan grafik permainan yang bergerak dinamis.

Pengembangan proyek ini juga menjadi wadah penerapan materi teoretis seperti pengolahan citra digital, deteksi fitur wajah, analisis visual berbasis landmark, serta integrasi multimodal input pada sistem interaktif. Selain itu, proyek ini menunjukkan bagaimana sinkronisasi antara deteksi kamera dan tampilan grafis dapat membentuk pengalaman multimedia yang responsif dan menarik.

2 Teknologi

Dalam membangun FaceFit Blink Challenge, digunakan beberapa teknologi inti yang mendukung pengolahan visual dan interaksi real-time. Python dipilih sebagai bahasa utama karena fleksibilitasnya serta ketersediaan pustaka yang kuat dalam pengolahan citra maupun pengembangan aplikasi multimedia.

OpenCV digunakan untuk menangani pengambilan frame dari kamera dan menyediakan berbagai fungsi dasar pengolahan citra. MediaPipe Face Mesh merupakan komponen utama dalam deteksi wajah, menghasilkan 468 landmark wajah berpresisi tinggi yang kemudian dianalisis untuk menentukan posisi fitur wajah serta mendeteksi kedipan melalui perhitungan rasio mata.

Pada sisi grafis, Pygame digunakan untuk membangun lingkungan permainan, merender objek bagian wajah yang jatuh, menampilkan antarmuka visual, dan mengelola game loop. Integrasi antara hasil deteksi wajah dan Pygame memungkinkan sistem menampilkan dinamika permainan secara sinkron dengan pergerakan dan ekspresi wajah pemain.

Teknologi-teknologi tersebut menunjukkan bagaimana konsep dalam Teknologi Multimedia seperti pemrosesan multimodal input, sinkronisasi media, dan real-time rendering dapat diterapkan dalam satu aplikasi yang koheren. Proyek ini menjadi contoh konkret penerapan teknologi multimedia modern berbasis visi komputer.

Tabel 1: Stack Teknologi Aplikasi

Komponen	Teknologi	Fungsi
Bahasa Pemrograman	Python 3.11	Bahasa utama dengan ekosistem library yang kaya
Computer Vision	OpenCV 4.10	Akses kamera, manipulasi gambar, rendering
Face Detection	MediaPipe 0.10.21	Deteksi wajah dan 468 landmark presisi tinggi
Komputasi Numerik	NumPy 1.26.4	Operasi array dan transformasi geometri
Audio	Pygame 2.6.1	Background music dan sound effects

3 Cara Kerja

Bagian ini menjelaskan alur kerja FaceFit Blink Challenge secara sistematis dan runut, menggambarkan bagaimana data kamera diproses dan bagaimana interaksi permainan terjadi berdasarkan kedipan mata pengguna.

- **Pengambilan Frame Kamera**

Sistem dimulai dengan mengaktifkan kamera untuk mengambil frame secara berkelanjutan. Frame mentah ini menjadi sumber data utama bagi proses deteksi wajah dan interaksi permainan.

- **Konversi Frame dan Pemrosesan Awal**

Setiap frame dikonversi dari format BGR ke RGB agar kompatibel dengan pipeline MediaPipe. Konversi ini memastikan proses deteksi wajah bekerja optimal.

- **Deteksi Landmark Menggunakan MediaPipe Face Mesh**

Frame yang telah diproses masuk ke Face Mesh untuk menghasilkan ratusan landmark wajah. Landmark-landmark ini digunakan untuk menentukan posisi mata, hidung, alis, dan mulut.

- **Perhitungan Rasio Kedipan Mata**

Sistem menghitung jarak antar kelopak mata dibandingkan dengan lebar mata. Jika rasio turun di bawah ambang tertentu, sistem mengenali kondisi tersebut sebagai kedipan.

- **Render Bagian Wajah yang Jatuh**

Pada sisi permainan, Pygame merender bagian wajah yang jatuh secara terus-menerus. Posisi setiap objek diperbarui sesuai kecepatan jatuh yang ditentukan.

- **Pengecekan Overlap antara Objek dan Landmark**

Saat bagian wajah jatuh mendekati area landmark yang sesuai, sistem mengecek apakah terjadi overlap. Overlap menandakan bahwa objek berada pada posisi siap ditempelkan.

- **Konfirmasi melalui Deteksi Kedipan**

Sistem menunggu kedipan pengguna sebagai bentuk persetujuan. Jika kedipan terjadi pada waktu yang tepat, objek dihentikan dan ditempelkan pada posisi landmark.

- **Penempelan Objek dan Pembaruan Status Permainan**

Setelah objek menempel, statusnya berubah menjadi terpasang. Sistem kemudian memeriksa apakah seluruh bagian wajah telah lengkap untuk menentukan akhir permainan.

- **Siklus Game Loop**

Semua proses di atas berjalan berulang dalam game loop, memungkinkan permainan berlangsung secara interaktif dan responsif terhadap ekspresi wajah pemain.

4 Penjelasan Kode Program

Pada bagian ini dijelaskan modul-modul utama yang menyusun FaceFit Blink Challenge beserta contoh potongan kode dan penjelasan baris demi baris.

4.1 Struktur Berkas Utama

- main.py – menjalankan game loop dan menghubungkan kamera serta modul deteksi wajah.
- face_processing.py – bertanggung jawab terhadap deteksi wajah, landmark, dan kedipan.
- game_controller.py – mengatur logika permainan dan interaksi dengan hasil deteksi wajah.
- falling_face_part.py – mendefinisikan objek bagian wajah yang jatuh.
- constants.py – menyimpan nilai konfigurasi permainan.

4.2 File main.py

File main.py merupakan entry point atau titik masuk utama aplikasi. File ini bertanggung jawab untuk inisialisasi kamera dan menjalankan game loop utama.

```
1 import cv2
2 from game_controller import FaceFilterGame
3
4 def main() -> None:
5     preferred_index = 1 # Kamera eksternal
6     fallback_index = 0 # Kamera default
7     selected_index = preferred_index
8
9     # Deteksi kamera tersedia
10    cap = cv2.VideoCapture(preferred_index)
11    if not cap.isOpened():
12        print("Kamera index 1 tidak tersedia")
13        selected_index = fallback_index
14        cap.release()
15        cap = cv2.VideoCapture(fallback_index)
16        if not cap.isOpened():
17            raise RuntimeError("Tidak ada kamera tersedia")
18    cap.release()
19
20    # Inisialisasi dan jalankan game
21    game = FaceFilterGame(camera_index=selected_index)
22    game.run()
23
24 if __name__ == "__main__":
25     main()
```

Kode 1: Struktur main.py

Penjelasan Fungsi:

- **Deteksi Kamera Otomatis:** Sistem mencoba membuka kamera eksternal (index 1) terlebih dahulu, jika gagal akan fallback ke kamera default (index 0). Ini memastikan aplikasi dapat berjalan pada berbagai konfigurasi hardware.
- **Error Handling:** Menggunakan try-except pattern untuk menangani kasus ketika tidak ada kamera yang tersedia, memberikan pesan error yang jelas.
- **Single Responsibility:** File ini hanya fokus pada inisialisasi hardware, delegasi logika kompleks ke class GameController.

4.3 File constants.py

File ini menyimpan semua konstanta dan parameter konfigurasi sistem dalam satu lokasi terpusat, memudahkan maintenance dan tuning parameter.

```
1 # Ambang Eye Aspect Ratio untuk deteksi kedipan
2 EAR_THRESHOLD = 0.21
3
4 # Konfigurasi Kalibrasi Dinamis EAR
5 EAR_DYNAMIC_ENABLED = True
6 EAR_DYNAMIC_SAMPLES = 30
7 EAR_DYNAMIC_FACTOR = 0.78
8 EAR_MIN_THRESHOLD = 0.17
9 EAR_HYSTERESIS = 0.02
10
11 # State machine aplikasi
12 STATE_MENU = 2
13 STATE_CAPTURE = 0
14 STATE_PLAYING = 1
15
16 # Countdown capture
17 CAPTURE_COUNTDOWN = 3
18
19 # Indeks landmark mata MediaPipe
20 LEFT_EYE_INDICES = [362, 385, 387, 263, 373, 380]
21 RIGHT_EYE_INDICES = [33, 160, 158, 133, 153, 144]
22
23 # Urutan bagian wajah
24 FACE_PART_SEQUENCE = [
25     "left_eyebrow", "right_eyebrow",
26     "left_eye", "right_eye",
27     "nose", "mouth"
28 ]
```

Kode 2: Konstanta dalam constants.py

Penjelasan Parameter Penting:

- **EAR_THRESHOLD (0.21):** Nilai ambang batas Eye Aspect Ratio. Ketika EAR turun di bawah nilai ini, sistem mendeteksi mata tertutup (kedipan).
- **EAR_DYNAMIC_ENABLED:** Mengaktifkan kalibrasi otomatis threshold EAR berdasarkan kondisi mata terbuka pengguna. Sistem mengumpulkan sampel EAR saat countdown dan menghitung threshold optimal.
- **EAR_HYSTERESIS (0.02):** Jarak aman antara threshold tutup dan buka mata untuk mencegah flicker/kedipan ganda karena noise.
- **State Machine:** Aplikasi menggunakan tiga state: MENU (tampilan awal), CAPTURE (mengambil foto wajah), dan PLAYING (mode permainan aktif).
- **Landmark Indices:** Indeks spesifik dari 468 landmark MediaPipe Face Mesh yang digunakan untuk menghitung EAR pada mata kiri dan kanan.

4.4 File face_processing.py

File ini merupakan inti dari pemrosesan visual, menangani deteksi landmark, perhitungan EAR, masking wajah, dan cropping bagian wajah.

4.4.1 Fungsi calculate_eye_aspect_ratio

```
1 def calculate_eye_aspect_ratio(
2     landmarks: List[List[float]],
3     eye_indices: Iterable[int]
4 ) -> float:
5     idx = list(eye_indices)
6     p = [np.array(landmarks[i], dtype=np.float32)
7           for i in idx]
8
9     v1 = np.linalg.norm(p[1] - p[5])
10    v2 = np.linalg.norm(p[2] - p[4])
11    h = np.linalg.norm(p[0] - p[3])
12
13    if h == 0.0:
14        return 0.0
15    return (v1 + v2) / (2.0 * h)
```

Kode 3: Perhitungan Eye Aspect Ratio

Penjelasan Algoritma: EAR dihitung menggunakan rumus: $EAR = \frac{||p_2-p_6||+||p_3-p_5||}{2 \times ||p_1-p_4||}$
Di mana:

- p_1, p_4 = sudut kiri dan kanan mata (jarak horizontal)
- p_2, p_3, p_5, p_6 = titik kelopak atas dan bawah (jarak vertikal)
- Ketika mata terbuka, jarak vertikal besar \rightarrow EAR tinggi
- Ketika mata tertutup, jarak vertikal kecil \rightarrow EAR rendah

4.4.2 Fungsi apply_face_mask

```
1 def apply_face_mask(
2     frame: np.ndarray,
3     landmarks: List[List[float]],
4     frame_width: int,
5     frame_height: int
6 ) -> np.ndarray:
7     # Estimasi warna kulit dari pipi
8     skin_color = _sample_skin_color(
9         frame, landmarks, frame_width, frame_height
10    )
11
12    # Area yang akan di-mask
13    masked_areas = [
14        # Mata kiri, mata kanan, hidung, alis
15    ]
16
17    # Tutup setiap area dengan warna kulit
18    for area_indices in masked_areas:
19        pts = _px_points(landmarks, area_indices,
20                         frame_width, frame_height)
21        hull = cv2.convexHull(pts)
22        cv2.fillPoly(frame, [hull], skin_color)
23
24    return frame
```

Kode 4: Masking Area Wajah

Penjelasan Proses:

- **Sampling Warna Kulit:** Mengambil sampel warna dari beberapa titik pipi, dahi, dan dagu, kemudian menghitung median BGR untuk mendapat warna kulit yang robust terhadap pencat-hayaan.
- **Convex Hull Masking:** Menggunakan convex hull untuk membuat polygon yang menutupi area mata, hidung, dan alis dengan warna kulit estimasi.
- **Feathering:** Gaussian blur diterapkan pada mask untuk membuat transisi tepi yang halus dan natural.

4.4.3 Fungsi crop_face_part

```
1 def crop_face_part(
2     frame: np.ndarray,
3     landmarks: List[List[float]],
4     frame_width: int, frame_height: int,
5     part_type: str,
6     mask_style: str = "polygon",
7     with_alpha: bool = False
8 ) -> Optional[FacePartData]:
9     # Dapatkan indeks landmark sesuai part_type
10    indices = indices_map.get(part_type)
11    points = _px_points(landmarks, indices,
12                         frame_width, frame_height)
13
14    # Buat convex hull
15    hull = cv2.convexHull(points)
16    x, y, w, h = cv2.boundingRect(hull)
17
18    # Crop ROI dengan padding
19    padding = 25
20    roi = frame[y-padding:y+h+padding,
21                 x-padding:x+w+padding]
22
23    # Buat mask polygon
24    mask = np.zeros((h, w), dtype=np.uint8)
25    cv2.fillPoly(mask, [hull_local], 255)
26
27    # Apply alpha channel
28    bgra = cv2.cvtColor(roi, cv2.COLOR_BGR2BGRA)
29    bgra[:, :, 3] = mask
30
31    return FacePartData(image=bgra, center=center, ...)
```

Kode 5: Crop Bagian Wajah

Penjelasan Teknik:

- **Polygon Masking:** Menggunakan convex hull dari landmark untuk membuat bentuk mask yang presisi mengikuti kontur bagian wajah.
- **Alpha Channel:** Menambahkan channel transparansi agar bagian wajah dapat di-blend dengan smooth pada background.
- **Anchor System:** Menyimpan data anchor (dua titik referensi) untuk transformasi skala dan rotasi saat tracking wajah.

4.5 File falling_face_part.py

File ini mendefinisikan class untuk objek bagian wajah yang jatuh dan tracking-nya terhadap wajah pengguna.

```
1 class FallingFacePart:
2     def __init__(self, part_data: FacePartData,
3                  part_type: str, screen_width: int,
4                  screen_height: int, spawn_x: int,
5                  fall_speed: float = 10.0):
6         self.part_data = part_data
7         self.x = float(spawn_x)
8         self.y = -50.0 # Mulai dari atas
9         self.is_falling = True
10        self.current_scale = 1.0
11        self.original_image = part_data.image.copy()
12
13    def update(self) -> None:
14        if self.is_falling:
15            self.y += self.fall_speed
16
17    def stop(self, target_center: Tuple[int, int]) -> None:
18        self.x = float(target_center[0])
19        self.y = float(target_center[1])
20        self.is_falling = False
21        self.part_data.center = target_center
22
23    def update_position_from_landmarks(
24        self, landmarks, frame_width, frame_height
25    ):
26        if self.is_falling:
27            return
28        result = compute_aligned_center(
29            self.part_data, landmarks,
30            frame_width, frame_height
31        )
32        if result:
33            new_center, scale, _, _ = result
34            self.x = float(new_center[0])
35            self.y = float(new_center[1])
36            self.apply_scale(scale)
```

Kode 6: Class FallingFacePart

Penjelasan Method Penting:

- **update():** Memperbarui posisi Y objek saat jatuh dengan menambahkan fall_speed setiap frame.
- **stop():** Menghentikan animasi jatuh dan menetapkan posisi final objek saat kedipan terdeteksi.
- **update_position_from_landmarks():** Setelah objek ditempel, method ini melakukan tracking wajah dengan menghitung transformasi anchor (skala dan rotasi) berdasarkan pergerakan landmark wajah.
- **apply_scale():** Mengubah ukuran gambar bagian wajah secara dinamis mengikuti jarak wajah ke kamera, menggunakan interpolasi linear untuk hasil smooth.

4.6 File game_controller.py

File terbesar dan paling kompleks, mengatur seluruh game loop, state machine, dan integrasi semua komponen.

4.6.1 Class SVGImageLoader

```
1 class SVGImageLoader:
2     def __init__(self):
3         self._cache: Dict[str, Optional[np.ndarray]] = {}
4         self.assets_dir = Path(__file__).parent / "Assets"
5
6     def load_svg_as_png(self, filename: str,
7                         width: int, height: int):
8         cache_key = f"{filename}-{width}-{height}"
9         if cache_key in self._cache:
10             return self._cache[cache_key]
11
12         png_path = self.assets_dir / filename
13         img = cv2.imread(str(png_path),
14                          cv2.IMREAD_UNCHANGED)
15         if img is not None:
16             img = cv2.resize(img, (width, height))
17             self._cache[cache_key] = img
18
19     return img
```

Kode 7: Loader Asset Image

Penjelasan: Loader ini mengimplementasikan caching pattern untuk menghindari loading berulang file image yang sama, meningkatkan performa aplikasi.

4.6.2 State Machine Implementation

```
1 def run(self) -> None:
2     while capture.isOpened():
3         frame_available, frame = capture.read()
4         frame = cv2.flip(frame, 1)
5
6         # Proses berdasarkan state
7         if self.current_state == STATE_MENU:
8             self._process_menu_state(display_frame, ...)
9         elif self.current_state == STATE_CAPTURE:
10            self._process_capture_state(frame, results, ...)
11        else: # STATE_PLAYING
12            self._process_play_state(frame, results, ...)
13
14         cv2.imshow("FaceFit Blink Challenge", display_frame)
15         key = cv2.waitKey(1) & 0xFF
16         if key == ord("q"):
17             break
```

Kode 8: Pengelolaan State

State Flow:

1. **STATE_MENU:** Menampilkan menu dengan tombol START, background gradien, dan instruksi.
2. **STATE_CAPTURE:** Countdown 3 detik sambil mengumpulkan sampel EAR untuk kalibrasi threshold dinamis, lalu capture bagian wajah.
3. **STATE_PLAYING:** Mode permainan aktif dengan deteksi kedipan dan penempelan objek.

4.6.3 Deteksi Kedipan dengan Hysteresis

```
1 avg_ear = calculate_average_ear(landmarks)
2 threshold = self._ear_threshold
3 close_level = threshold
4 open_level = threshold + EAR_HYSTERESIS
5
6 if not self._ear_state_closed:
7     if avg_ear < close_level:
8         self._ear_state_closed = True
9         blink_display = True
10    if not self._blink_active:
11        self._blink_active = True
12        self.blink_count += 1
13        blink_event = True
14 else:
15     if avg_ear > open_level:
16         self._ear_state_closed = False
17         self._blink_active = False
18     else:
19         blink_display = True
```

Kode 9: Deteksi Kedipan Anti-Flicker

Penjelasan Hysteresis: Sistem menggunakan dua threshold berbeda untuk tutup (close_level) dan buka (open_level) mata. Ini mencegah kedipan ganda karena nilai EAR yang berfluktuasi di sekitar threshold tunggal. Mata dianggap menutup ketika $\text{EAR} < \text{close_level}$, dan baru dianggap membuka kembali ketika $\text{EAR} > \text{open_level}$.

4.6.4 Kalibrasi EAR Dinamis

```
1 # Kumpulkan sampel saat countdown
2 if EAR_DYNAMIC_ENABLED and results:
3     ear_sample = calculate_average_ear(landmarks)
4     if len(self._ear_samples) < EAR_DYNAMIC_SAMPLES:
5         self._ear_samples.append(ear_sample)
6
7 # Setelah countdown, kalibrasi threshold
8 if EAR_DYNAMIC_ENABLED and self._ear_samples:
9     baseline = float(np.median(self._ear_samples))
10    calibrated = max(EAR_MIN_THRESHOLD,
11                      baseline * EAR_DYNAMIC_FACTOR)
12    self._ear_threshold = calibrated
```

Kode 10: Kalibrasi Threshold Otomatis

Keunggulan Kalibrasi Dinamis:

- Adaptif terhadap variasi bentuk mata antar individu
- Menyesuaikan dengan kondisi pencahayaan
- Menggunakan median untuk robust terhadap outlier
- Memiliki batas minimum untuk mencegah threshold terlalu rendah

4.6.5 Audio System

```
1 def __init_audio(self) -> None:
2     pygame.mixer.init(frequency=44100, size=-16,
3                         channels=2, buffer=512)
4     audio_path = Path(__file__).parent / "Assets" /
5                         "Cartoon Bounce.mp3"
```

```
6     if audio_path.exists():
7         pygame.mixer.music.load(str(audio_path))
8         self._audio_loaded = True
9
10    def _play_background_music(self) -> None:
11        if self._audio_loaded:
12            pygame.mixer.music.play(-1) # Loop infinite
```

Kode 11: Background Music

Penjelasan: Pygame mixer digunakan untuk memutar background music secara loop. Parameter -1 pada play() membuat music berulang tanpa henti hingga di-stop.

4.7 File requirements.txt

```
1 mediapipe==0.10.21
2 opencv-python==4.10.0.84
3 numpy==1.26.4
4 protobuf==4.25.3
5 pygame==2.6.1
```

Kode 12: Dependencies Python

Penjelasan Dependencies:

- **mediapipe:** Library Google untuk deteksi wajah dan 468 landmark presisi tinggi.
- **opencv-python:** Library computer vision untuk akses kamera, manipulasi gambar, dan rendering.
- **numpy:** Komputasi numerik untuk operasi array dan transformasi geometri.
- **protobuf:** Di-pin ke versi 4.25.3 untuk menghindari konflik dengan MediaPipe yang tidak support protobuf 5.x.
- **pygame:** Audio playback untuk background music.

5 Integrasi Antar File

5.1 Flow Eksekusi Lengkap

1. **main.py** mendeteksi kamera dan membuat instance FaceFilterGame
2. **game_controller.py** menginisialisasi:
 - MediaPipe Face Mesh
 - SVGImageLoader untuk asset UI
 - Pygame mixer untuk audio
 - State machine dimulai di STATE_MENU
3. **State MENU:** User klik tombol START
4. **State CAPTURE:**
 - Countdown 3 detik
 - face_processing.calculate_average_ear() mengumpulkan sampel
 - Kalibrasi threshold EAR

- `face_processing.crop_face_part()` untuk setiap bagian wajah
- Simpan hasil crop ke dictionary `face_parts`

5. State PLAYING:

- Spawn `FallingFacePart` dari queue
- Update posisi objek jatuh setiap frame
- Deteksi kedipan dengan hysteresis
- Saat kedipan: stop objek, apply scale, reanchor
- Tracking objek yang sudah ditempel mengikuti wajah
- Repeat untuk bagian wajah berikutnya

6. Selesai: Semua bagian wajah terpasang, tampilkan pesan selesai

5.2 Data Flow Diagram



Gambar 1: Diagram Alur Data FaceFit Blink Challenge

5.3 Key Design Patterns

- **State Pattern:** Tiga state (MENU, CAPTURE, PLAYING) dengan behavior berbeda per state
- **Caching Pattern:** `SVGImageLoader` cache image yang sudah di-load
- **Observer Pattern:** Face tracking observer untuk update posisi objek terpasang
- **Factory Pattern:** Spawn objek `FallingFacePart` dari `FacePartData`

6 Hasil Analisis

6.1 Program Utama

Program utama dibangun dengan pendekatan modular, di mana setiap komponen menjalankan tugasnya masing-masing namun terhubung melalui game loop di file `main.py`. Secara umum, alur utama aplikasi melibatkan: Inisialisasi modul deteksi wajah, Inisialisasi permainan, Pembacaan input kamera secara real-time, Pemrosesan landmark dan deteksi kedipan, Pembaruan objek permainan, Render tampilan ke layar, dan Sinkronisasi antar modul hingga permainan selesai. Dengan arsitektur ini, sistem dapat memproses input visual secara responsif tanpa mengganggu performa permainan. Berikut adalah isi kode program utama:

```
1 from game_controller import GameController
2
3 def main():
4     game = GameController()      # Inisialisasi game controller
5     game.run()                  # Memulai loop permainan
6
7 if __name__ == "__main__":
8     main()
```

Kode 13: Program Utama Permainan

6.2 Deskripsi Kode

Kode program utama sengaja dibuat ringkas agar fokus pengendalian berada pada kelas GameController. Fungsi main() bertujuan:

1. **Membuat instance GameController:**

Objek ini berisi seluruh komponen permainan seperti loader sprite, kamera, detektor wajah, dan daftar objek jatuh.

2. **Menjalankan game loop:**

Metode run() memulai siklus pembacaan frame kamera, pemrosesan kedipan, serta penggambaran tampilan pada layar.

Dengan desain ini, file utama menjadi titik masuk eksekusi yang bersih dan mudah dipelihara.

6.3 Alur Eksekusi Program Utama

Secara detail, alur eksekusi saat game.run() dipanggil adalah sebagai berikut:

1. **Inisialisasi kamera**

Sistem membuka akses webcam dan memulai pipeline Face Mesh.

2. **Loop permainan dimulai**

Program terus berjalan selama window permainan masih aktif.

3. **Frame kamera dibaca dan diolah**

Setiap iterasi, gambar yang masuk dikonversi menjadi RGB untuk diproses oleh MediaPipe.

4. **Deteksi wajah dan landmark dilakukan**

Model Face Mesh menghasilkan koordinat 468 titik wajah.

5. **Deteksi kedipan dihitung**

EAR (Eye Aspect Ratio) dihitung. Jika $\text{EAR} < \text{threshold} \rightarrow$ kedipan terdeteksi.

6. **Update posisi objek jatuh**

Setiap objek diperbarui posisinya sesuai kecepatan gravitasi.

7. **Cek overlap antara objek dengan wajah**

Sistem menentukan apakah objek berada tepat di posisi landmark yang sesuai.

8. **Jika overlap + kedipan \rightarrow objek ditempelkan**

Objek tidak lagi jatuh dan menyatu dengan wajah sesuai aturan permainan.

9. **Frame ditampilkan ke layar**

Kamera dan objek digabungkan menjadi satu tampilan permainan.

10. **Loop berulang hingga kondisi selesai**

Pemain dapat menekan tombol keluar atau menyelesaikan semua bagian wajah.

6.4 Integrasi Antar Modul

Modul-modul sistem bekerja saling terhubung sebagai berikut:

- face_processing.py menyediakan data landmark serta status kedipan.
- falling_face_part.py mengatur posisi bagian wajah yang jatuh.

- game_controller.py membaca input dari face_processing.py, memperbarui objek jatuh, lalu merender gambar.
- main.py memanggil seluruh proses secara terstruktur.
- Integrasi ini mendukung pipeline multimedia: capture → process → update → render.

6.5 Output dan Tampilan Program

Program menampilkan:

1. **Video kamera real-time**
Wajah pengguna terlihat sebagai latar belakang permainan.
2. **Objek bagian wajah yang jatuh**
Seperti mata, hidung, alis, dan mulut yang jatuh dari atas layar.
3. **Penempelan objek saat kedipan terdeteksi**
Ketika pemain berkedip tepat waktu, objek otomatis menempel pada landmark yang sesuai.
4. **Indikator Status EAR**
Menampilkan nilai Eye Aspect Ratio dan threshold kalibrasi untuk monitoring deteksi kedipan.
5. **Jumlah Kedipan**
Counter yang menunjukkan berapa kali pemain berhasil mengedipkan mata.
6. **Tracking Objek Terpasang**
Bagian wajah yang sudah ditempel akan mengikuti pergerakan kepala secara real-time.
7. **Background Music**
Audio loop yang meningkatkan pengalaman bermain.
8. **Tampilan Menu Interaktif**
Tombol START dengan hover effect dan instruksi permainan.
9. **Countdown Capture**
Hitungan mundur 3 detik sebelum capture wajah dimulai.
10. **Pesan Selesai**
Notifikasi ketika semua bagian wajah berhasil ditempelkan dengan opsi restart.

6.6 Analisis Implementasi

Hasil implementasi menunjukkan bahwa:

1. **Sistem mampu memproses input real-time dengan stabil.**
2. **Deteksi kedipan bekerja dengan cepat dan tingkat akurasi baik.**
3. **Integrasi antara deteksi wajah dan logika permainan berjalan sinkron.**
4. **Proses rendering tidak mengalami penurunan FPS yang berarti.**

5. Pengguna dapat berinteraksi secara natural tanpa perangkat tambahan.
6. Kalibrasi EAR dinamis meningkatkan adaptabilitas sistem.
7. Tracking objek terpasang mengikuti pergerakan wajah dengan presisi.
8. Masking warna kulit menghasilkan tampilan natural.
9. Arsitektur modular memudahkan pengembangan lanjutan.
10. Caching asset meningkatkan performa loading.

7 Implementasi Program

7.1 Persiapan Lingkungan

Program dijalankan pada Python 3.11 dengan pustaka OpenCV, MediaPipe, dan Pygame. Semua dependensi tersedia di file requirements.txt.

7.2 Instalasi

```
1 git clone https://github.com/Youngstg/FaceFit-Blink-Challenge.git
2 cd FaceFit-Blink-Challenge
3 py -3.11 -m venv .venv
4 .\venv\Scripts\activate
5 pip install -r requirements.txt
```

7.3 Menjalankan Program

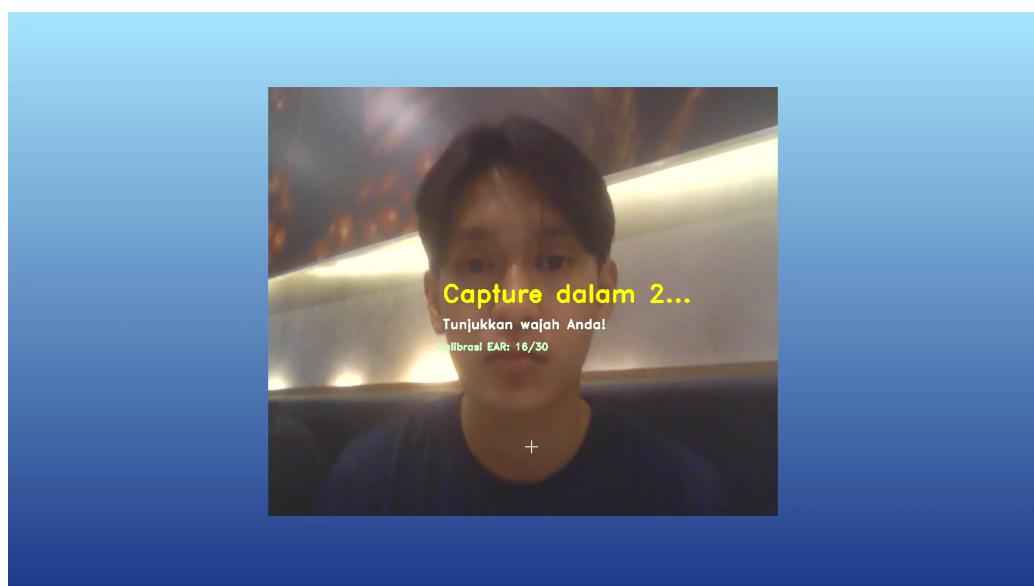
```
1 python main.py
```

7.4 Jalankan Program

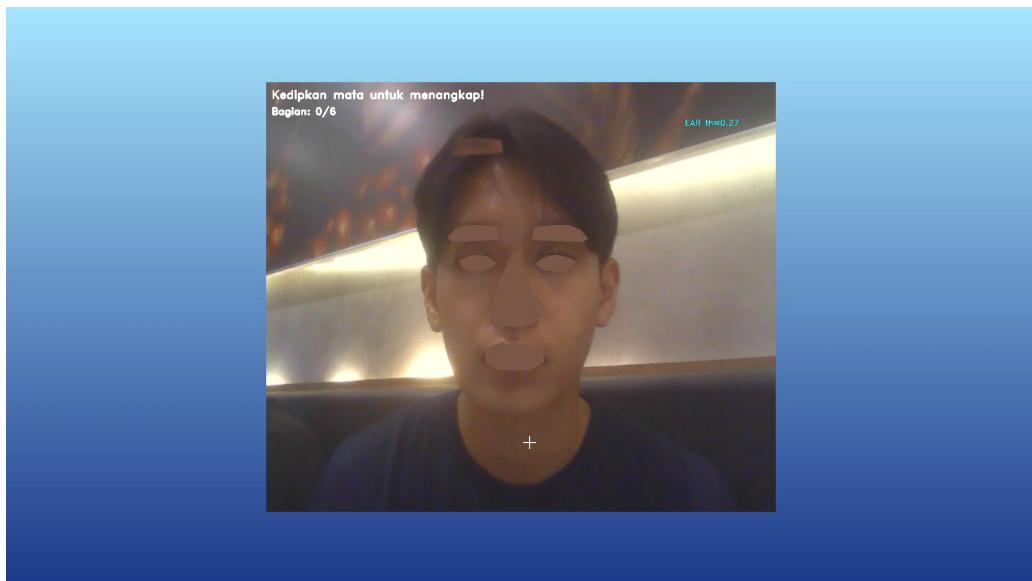
File main.py akan menjalankan program secara real-time. Rekan tombol "Start" untuk memulai program



Gambar 2: Tampilan awal antarmuka



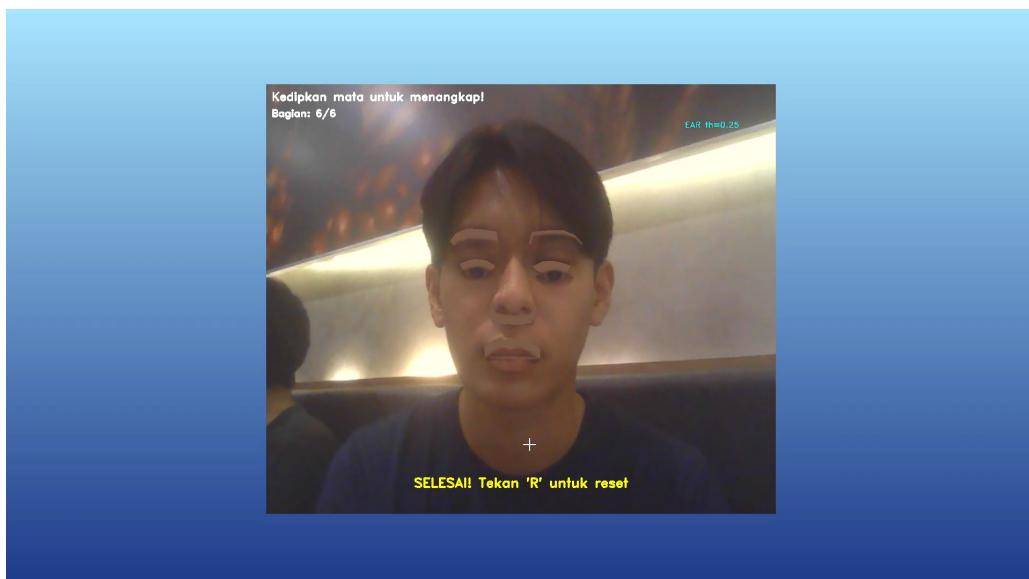
Gambar 3: konfigurasi EAR awal



Gambar 4: crop dan masking muka



Gambar 5: Pemain harus mengedipkan mata saat bagian wajah jatuh



Gambar 6: Game Selesai

- layar akan menampilkan keterangan jika 'Start' untuk memulai.
- Game akan mendeteksi kondisi mata pemain ketika terbuka menggunakan EAR.
- Ketika game dimulai permainan akan mendeteksi kedipan mata jika berkedip bagian wajah yang jatuh akan berhenti dan akan diberi anchor agar mengikuti posisi kepala.
- setelah pemain menyelesaikan game, akan muncul pilihan Tekan "R" untuk reset.