Program Studi Teknik Informatika
Institut Teknologi Sumatera

Mata Kuliah: **Sistem / Teknologi Multimedia (IF40305)** Tugas: **Tugas Besar**

Nama Anggota:
1. **Elma Nurul Fatika (122140069)**
2. **Lois Novel E Gurning (122140098)**
3. **Dina Rahma Dita (122140184)**

Tanggal: 03 Desember 2025

# 1 Deskripsi Projek

Purrfect Pitch merupakan tugas besar pada mata kuliah Sistem/Teknologi Multimedia yang dikembangkan sebagai bentuk penerapan konsep-konsep teknologi multimedia. Implementasi tersebut diwujudkan melalui pembuatan sebuah filter interaktif yang memanfaatkan audio processing serta computer vision. Berbagai komponen seperti face tracking, pengolahan audio, antarmuka (GUI), dan logika aplikasi diintegrasikan dalam tugas besar ini.

Dalam penerapannya, suara kucing yang telah mengalami perubahan pitch melalui pustaka librosa diperdengarkan kepada pengguna. Berdasarkan suara tersebut, pengguna diminta mengidentifikasi pilihan gambar kucing yang benar. Pemilihan dilakukan bukan melalui tombol, melainkan dengan memiringkan kepala ke kiri atau ke kanan sesuai jawaban yang menurut pengguna benar, di mana arah kemiringan dibaca sebagai input oleh sistem melalui proses pendeteksian dan pelacakan wajah secara real-time menggunakan MediaPipe Face Mesh.

Desain kontrol pada filter ini juga disusun dengan kriteria khusus. Kemiringan kepala lebih dari 12° ke kiri akan diinterpretasikan sebagai pilihan kiri, sementara sudut lebih dari 12° ke kanan dianggap sebagai pilihan kanan. Visualisasi waveform setiap audio kucing juga turut ditampilkan. Untuk memastikan input tidak terbaca secara keliru, sistem menerapkan ambang batas roll angle 12 derajat serta hold time 0,6 detik. Seluruh proses pendeteksian dan pelacakan tersebut dijalankan menggunakan model MediaPipe Face Mesh.

# 2 Alat dan Bahan

Pengembangan Proyek Purrfect Pitch memanfaatkan berbagai perangkat lunak serta metode tertentu agar fungsionalitas yang ditargetkan dapat tercapai. Adapun alat dan bahan yang terlibat dalam proses pengembangannya disajikan pada daftar berikut.

## 2.1 Bahasa Pemrograman

Proyek ini dibuat menggunakan Python versi 3.11 ke atas. Pemilihan Python didasarkan pada ekosistem library yang sangat lengkap, mencakup pengolahan audio seperti *librosa* dan *soundfile*, pemrosesan citra melalui *OpenCV* dan *MediaPipe*, serta kebutuhan pengembangan media interaktif. Selain itu, Python mudah dipahami, memungkinkan penulisan kode yang ringkas, dan mendukung proses prototyping dengan cepat. Kemampuan Python dalam memproses sinyal audio, melakukan deteksi wajah secara *real-time*, serta mengintegrasikan berbagai elemen multimedia menjadikannya pilihan yang tepat untuk membangun filter interaktif yang memanfaatkan audio dan *face tracking* seperti Purrfect Pitch.

## 2.2 *Library*

Pengembangan proyek ini memanfaatkan sejumlah *library* Python yang memiliki peran krusial dalam proses audio, visual, dan logika permainan yang diterapkan pada filter.

- **Librosa (≥0.10.0)**: *Library* ini menjadi fondasi utama dalam pengolahan audio serta analisis sinyal. Melalui Librosa, sistem dapat melakukan *pitch shifting* tanpa memengaruhi tempo suara, mengekstraksi berbagai fitur audio, dan menghasilkan data *waveform* yang diperlukan untuk visualisasi secara *real-time.*
- **SoundFile (≥0.12.0)**: *SoundFile* digunakan untuk membaca dan menulis beragam format berkas audio, seperti WAV, FLAC, dan OGG. *Library* ini bekerja berdampingan dengan *Librosa* untuk menyimpan audio yang telah diproses sambil tetap menjaga kualitas suara dan dukungan multi-channel.
- **OpenCV (≥4.8.0)**: *OpenCV* berperan dalam merender *frame* video secara *real-time*, melakukan manipulasi gambar seperti *resize* dan *alpha blending*, serta menampilkan tampilan grafis filter. Selain menerima input dari webcam, *OpenCV* juga digunakan untuk menggambar elemen visual seperti *sprite, waveform*, dan komponen antarmuka filter.
- **MediaPipe (≥0.10.0)**: *Library* MediaPipe khususnya modul *Face Mesh* dipakai untuk mendeteksi serta melacak wajah pemain. Kemampuannya menyediakan koordinat *landmark* wajah berjumlah 478 titik secara presisi memungkinkan perhitungan *roll angle*, yaitu sudut kemiringan kepala yang dijadikan kontrol utama dalam filter.
- **Pygame (≥2.5.0)**: Pygame berfungsi mengatur pemutaran audio dan alur logika permainan yang diterapkan pada filter. Melalui Pygame Mixer, audio hasil pitch shifting diputar ulang, volume dikendalikan, dan event ketika audio selesai dapat dideteksi. Selain itu, library ini menangani game state dan *event loop* selama interaksi berlangsung.
- **NumPy (≥1.24.0)**: NumPy digunakan untuk operasi numerik intensif, terutama dalam memanipulasi array audio seperti waveform, melakukan normalisasi amplitudo, interpolasi sampel untuk visualisasi, serta proses alpha blending untuk komposisi gambar yang melibatkan transparansi.

## 2.3 Metode dan Algoritma

Proyek ini dibangun dengan beberapa metode dan algoritma utama sebagai berikut:

- **Pitch Shifting dengan Librosa**: Proses perubahan nada suara kucing dilakukan menggunakan fungsi `librosa.effects.pitch_shift()`. Algoritma ini memungkinkan perubahan *pitch* tanpa memengaruhi tempo suara. Semua file audio diproses secara batch dengan pengaturan standar shift sebesar -5 semitone untuk menghasilkan variasi suara yang nantinya harus dipilih oleh pemain.
- **Deteksi Wajah memakai MediaPipe Face Mesh**: MediaPipe Face Mesh digunakan sebagai sistem pendeteksi wajah *real-time*. Model ini menemukan wajah dalam setiap frame video dan mengekstrak 478 titik *landmark*, termasuk titik pada kedua mata (indeks 33 dan 263) yang dipakai untuk menghitung sudut kemiringan kepala pemain.
- **Interpretasi Head Tilt (Kemiringan Kepala)**: Sebuah algoritma dirancang untuk menerjemahkan sudut kemiringan wajah menjadi perintah permainan.
  - Perhitungan Sudut *(Roll Angle)* dilakukan menggunakan koordinat kedua mata dan rumus atan2(dy, dx) lalu dikonversi ke derajat.

  - Deteksi Arah Tilt memakai aturan: tilt LEFT jika sudut < -12°, tilt RIGHT jika sudut > 12°, dan posisi CENTER jika berada di antara ±12°.
  - *Hold Time & Cooldown* diterapkan agar input tidak salah terbaca: *hold time* 0,6 detik untuk memastikan kemiringan stabil dan *cooldown* 1 detik agar tidak terjadi input berulang yang tidak sengaja.

- **Pembuatan Waveform untuk Visualisasi**: Waveform audio dibuat dengan membaca audio menggunakan `librosa.load()`, mengambil nilai amplitudo absolut, lalu melakukan *downsampling* menjadi 512 sampel menggunakan interpolasi linear `(np.interp())`. Hasil akhirnya dinormalisasi ke rentang 0–1 sehingga dapat divisualisasikan secara *real-time.*

- **Logika Permainan**: Algoritma permainan mencakup beberapa komponen:
  - *Random Question Generation*: Pertanyaan atau audio diacak dengan `random.shuffle()`.
  - *Timer*: Waktu permainan dihitung memakai `time.monotonic()` dengan batas 45 detik.
  - *Answer Checking*: Jawaban pemain (LEFT/RIGHT) dibandingkan dengan nilai `correct_side`.
  - *Score Tracking*: Sistem otomatis menambah skor saat pemain menjawab benar dan mencatat jumlah pertanyaan yang sudah dilewati.

- **Pemrosesan Video *Real-Time***: Input dari webcam diambil secara terus-menerus melalui `cv2.VideoCapture`. Setiap frame diproses untuk mendeteksi wajah, menghitung *head tilt*, waveform, dan elemen UI. Proses ini berjalan di dalam game loop utama sehingga pengalaman bermain terasa lancar dan responsif.

- **Manajemen Pemutaran Audio**: Pygame Mixer digunakan untuk mengatur seluruh pemutaran audio, mulai dari memuat file suara, memulai dan menghentikan *playback*, hingga menjalankan *event callback* saat audio selesai. Semua proses ini disinkronkan dengan state permainan agar setiap pergantian soal berjalan halus.

# 3 Penjelasan

Program Purfect Pitch dibangun dari sejumlah modul Python yang bekerja sama untuk menggerakkan seluruh fitur. Setiap modul memiliki tugasnya masing-masing, mulai dari logika hingga pemrosesan input gerakan.

## 3.1 Instalasi *Library*

Agar program dapat berjalan dengan baik, langkah pertama yang perlu dilakukan adalah memasang seluruh *library* yang menjadi dependensi aplikasi. Semua kebutuhan tersebut sudah tercantum dalam file `requirements.txt` sehingga proses instalasi cukup dilakukan melalui satu perintah:

```
pip install -r requirements.txt
```

Kode 1: Instalasi dependensi melalui file requirements.txt

Dengan menjalankan perintah tersebut, seluruh paket yang diperlukan akan terpasang otomatis. Disarankan untuk membuat *virtual environment* terlebih dahulu supaya instalasi library tetap terpisah dan tidak mengganggu konfigurasi Python pada sistem utama.

## 3.2 Modul `main.py`

Modul `main.py` berfungsi sebagai entry point atau titik masuk utama untuk menjalankan aplikasi game Purrfect Pitch. File ini dirancang dengan prinsip kesederhanaan, di mana semua logika kompleks didelegasikan ke modul lain, khususnya modul `gui.py`. Dengan pendekatan ini, `main.py` hanya bertanggung jawab untuk menginisialisasi game dan menangani error handling pada level tertinggi.

- Import Modul GUI dan Inisialisasi Game

  Modul ini mengimpor class PurrfectPitchGame dari `gui.py`, yang merupakan orchestrator utama untuk seluruh komponen game. Fungsi `main()` sederhana menciptakan instance game dan memanggil method `run()` untuk memulai game loop.

```python
#!/usr/bin/env python3
"""
Entry point for the Purrfect Pitch game.

Fokus file ini hanya menjalankan permainan dengan memanggil modul GUI.
Semua logika tampilan/loop berada di gui.py agar main tetap sederhana.
"""

import sys

from gui import PurrfectPitchGame


def main() -> None:
    """Start the Purrfect Pitch game."""
    game = PurrfectPitchGame()
    game.run()
```

Kode 2: Import modul GUI dan inisialisasi game

- Error Handling dan Crash Protection

  Pada bagian ini, terdapat try-except block yang menangkap semua exception yang mungkin terjadi selama game berjalan. Jika terjadi error fatal, program akan mencetak pesan error, traceback lengkap untuk debugging, dan keluar dengan exit code 1.

```python
if __name__ == "__main__":
    try:
        main()
    except Exception as exc:  # pragma: no cover
        print(f"\n[FATAL ERROR] {exc}")
        import traceback
        import sys

        traceback.print_exc()
        sys.exit(1)
```

Kode 3: Error Handling dan Crash Protection

## 3.3 Modul `audio_processing.py`

Modul `audio_processing.py` berfungsi sebagai utilitas pemrosesan audio yang menyediakan dua fungsi utama: *pitch shifting* dan generasi data *waveform*. Modul ini menggunakan pustaka `librosa` untuk manipulasi audio dan `soundfile` untuk operasi I/O berkas audio. Modul ini bersifat independen dan dapat digunakan sebagai *CLI tool* atau dipanggil oleh modul lain.

- Pitch Shifting dengan Mempertahankan Channel Audio

Fungsi `pitch_shift_file()` melakukan perubahan pitch (nada) audio tanpa mengubah tempo/kecepatan. Fungsi ini menangani audio mono dan stereo dengan cara yang berbeda: audio mono diproses secara langsung, sedangkan audio multi-channel diproses per channel, kemudian digabungkan kembali. Padding diterapkan apabila terdapat perbedaan panjang antar channel setelah pemrosesan.

```
def pitch_shift_file(input_path: Path, output_path: Path, n_semitones: float, sr_target=None)
    :
    """
    Mengubah pitch audio tanpa mengubah kecepatan/tempo.

    Args:
        input_path (Path): Path file audio input
        output_path (Path): Path file audio output (format WAV)
        n_semitones (float): Jumlah semitone untuk shift
                             (negatif = pitch down, positif = pitch up)
        sr_target (int, optional): Target sample rate. None = gunakan sample rate asli

    Returns:
        tuple: (sample_rate, duration) dari file output
    """

    # Load audio dengan mempertahankan channel asli (mono/stereo)
    y, sr = librosa.load(str(input_path), sr=sr_target, mono=False)

    # Proses audio berdasarkan jumlah dimensi array
    if y.ndim == 1:
        # Audio mono: proses langsung dalam satu operasi
        y_shift = librosa.effects.pitch_shift(y, sr=sr, n_steps=n_semitones)
        sf.write(str(output_path), y_shift, sr)
    else:
        # Audio multi-channel: proses setiap channel secara terpisah
        channels = []
        for ch_idx in range(y.shape[0]):
            ch = y[ch_idx]
            ch_shift = librosa.effects.pitch_shift(ch, sr=sr, n_steps=n_semitones)
            channels.append(ch_shift)

        # Samakan panjang semua channel jika berbeda setelah pitch shift
        max_len = max(map(len, channels))
        channels = [np.pad(c, (0, max_len - len(c)), mode="constant") for c in channels]

        # Gabungkan semua channel menjadi satu array 2D
        y_shift = np.vstack(channels)
        sf.write(str(output_path), y_shift.T, sr)

    out_duration = librosa.get_duration(filename=str(output_path))
    return sr, out_duration
```

Kode 4: Pitch Shifting

- Generasi Data Waveform untuk Visualisasi

  Fungsi ini menghasilkan representasi visual dari audio dalam bentuk array amplitudo yang ternormalisasi (0.0 - 1.0). Fungsi ini menggunakan interpolasi linear untuk mengambil sejumlah sample dari audio secara merata, lalu menormalisasi nilainya agar dapat divisualisasikan dengan baik di GUI.

```
def generate_waveform_data(audio_path: Path, num_samples: int = 512):
    """
    Membuat data waveform ter-normalisasi dari file audio.

```

```
    Args:
        audio_path (Path): Path file audio sumber.
        num_samples (int): Jumlah sampel waveform yang diinginkan.

    Returns:
        list[float]: List amplitudo (0..1) sebanyak num_samples.
    """
    if num_samples <= 0:
        raise ValueError("num_samples harus lebih besar dari 0")

    y, _ = librosa.load(str(audio_path), sr=None, mono=True)
    if y.size == 0:
        return [0.0] * num_samples

    amplitudes = np.abs(y)
    positions = np.linspace(0, amplitudes.size - 1, num=num_samples)
    samples = np.interp(positions, np.arange(amplitudes.size), amplitudes)

    max_amp = samples.max()
    if max_amp > 0:
        samples = samples / max_amp

    return samples.tolist()
```

Kode 5: Generasi Data Waveform untuk Visualisasi

- Batch Processing dengan Command-Line Interface

  Fungsi ini menyediakan interface CLI lengkap untuk memproses banyak file audio sekaligus. User dapat menentukan folder input/output, jumlah semitone shift, dan target sample rate melalui argumen command-line. Hasil processing beserta metadata disimpan dalam file JSON untuk tracking dan debugging.

```
def main():
    """Fungsi utama untuk menjalankan batch processing pitch shift."""
    parser = argparse.ArgumentParser(description="Batch pitch-down audio files in a folder.")
    parser.add_argument("--in_folder", "-i", type=Path, default=Path("asset"))
    parser.add_argument("--out_folder", "-o", type=Path, default=Path("asset_output"))
    parser.add_argument("--semitones", "-s", type=float, default=-5.0)
    parser.add_argument("--sr", type=int, default=None)
    args = parser.parse_args()

    # Proses semua file dengan progress bar
    for f in tqdm(files, desc="Processing audio files"):
        try:
            # Generate nama output dan proses pitch shift
            out_name = f"{name}_pitch{int(semitones)}.wav"
            out_path = out_folder / out_name
            sr_out, out_duration = pitch_shift_file(f, out_path, n_semitones=semitones)

            # Simpan metadata processing
            metadata.append({
                "input": str(f.relative_to(Path.cwd())),
                "output": str(out_path.relative_to(Path.cwd())),
                "semitones": semitones,
                "orig_duration_s": round(orig_duration, 3),
                "out_duration_s": round(out_duration, 3),
                "sr_out": sr_out
            })
        except Exception as e:
            print(f"[WARN] Failed processing {f.name}: {e}")
```

Kode 6: Batch Processing dengan Command-Line Interface

### 3.4 Modul `face_tracker.py`

Modul `face_tracker.py` ini menyediakan sistem deteksi dan tracking wajah real-time menggunakan MediaPipe Face Mesh. Modul ini bertanggung jawab untuk mendeteksi keberadaan wajah, menghitung sudut kemiringan kepala (head tilt), dan melaporkan status konfirmasi berdasarkan durasi hold time. Modul ini dirancang modular agar dapat digunakan kembali tanpa ketergantungan pada game logic.

- Data Structure untuk State Face Tracking

  Class FaceTrackerState adalah dataclass yang menyimpan snapshot state deteksi wajah pada satu frame, termasuk apakah wajah terdeteksi, sudut kemiringan dalam derajat, status tilt (LEFT-/RIGHT/CENTER/NO FACE), konfirmasi tilt, timestamp, posisi center wajah, dan ukuran bounding box wajah.

```python
@dataclass
class FaceTrackerState:
    face_detected: bool
    roll_deg: float
    tilt_state: str  # LEFT, RIGHT, CENTER, NO_FACE
    tilt_confirmed: bool
    timestamp: float
    face_center: Optional[Tuple[int, int]] = None
    face_size: Optional[Tuple[int, int]] = None
```

Kode 7: Data Structure untuk State Face Tracking

- Inisialisasi Face Tracker dengan MediaPipe

  Class FaceTracker menginisialisasi MediaPipe Face Mesh dengan konfigurasi optimal untuk real-time tracking: `max_num_faces=1` untuk performa, `refine_landmarks=True` untuk akurasi tinggi, dan threshold detection/tracking yang seimbang. State internal meliputi threshold tilt, hold time, cooldown time, serta tracking untuk konfirmasi gesture.

```python
class FaceTracker:
    def __init__(
        self,
        tilt_threshold: float = TILT_THRESHOLD_DEG,
        hold_time: float = HOLD_TIME,
        cooldown_time: float = COOLDOWN_TIME,
    ) -> None:
        self.tilt_threshold = tilt_threshold
        self.hold_time = hold_time
        self.cooldown_time = cooldown_time
        self._face_mesh = mp.solutions.face_mesh.FaceMesh(
            static_image_mode=False,
            max_num_faces=1,
            refine_landmarks=True,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5,
        )
        self._cap: Optional[cv2.VideoCapture] = None
        self._tilt_state = "CENTER"
        self._tilt_start: Optional[float] = None
        self._last_confirm = 0.0
```

Kode 8: Inisialisasi Face Tracker dengan MediaPipe

- Kalkulasi Sudut Kemiringan Kepala (Roll Angle)

  Method `_compute_roll_deg()` menghitung sudut kemiringan kepala berdasarkan posisi landmark mata kiri (index 263) dan mata kanan (index 33). Dengan menghitung arctangent dari

delta Y dan delta X antara kedua mata, diperoleh sudut dalam radian yang kemudian dikonversi ke derajat. Nilai positif menandakan kepala miring ke kanan, negatif ke kiri.

```python
def _compute_roll_deg(self, landmarks, width: int, height: int) -> float:
    right_eye_idx = 33
    left_eye_idx = 263
    r_x = int(landmarks[right_eye_idx].x * width)
    r_y = int(landmarks[right_eye_idx].y * height)
    l_x = int(landmarks[left_eye_idx].x * width)
    l_y = int(landmarks[left_eye_idx].y * height)
    dx = l_x - r_x
    dy = l_y - r_y
    angle_rad = math.atan2(dy, dx)
    return math.degrees(angle_rad)
```

Kode 9: Kalkulasi Sudut Kemiringan Kepala (Roll Angle)

- State Machine untuk Konfirmasi Tilt dengan Hold Time

  Method `_evaluate_state()` mengimplementasikan state machine yang mendeteksi dan mengkonfirmasi gesture tilt kepala. Jika sudut melebihi threshold, state berubah menjadi LEFT atau RIGHT. Konfirmasi terjadi jika user mempertahankan posisi tilt selama durasi `hold_time` dan sudah melewati `cooldown_time` sejak konfirmasi terakhir. Informasi posisi dan ukuran wajah dihitung dari bounding box semua landmarks.

```python
# Evaluasi state tilt dan konfirmasi
prev_state = self._tilt_state
if roll_deg > self.tilt_threshold:
    self._tilt_state = "RIGHT"
elif roll_deg < -self.tilt_threshold:
    self._tilt_state = "LEFT"
else:
    self._tilt_state = "CENTER"

confirmed = False
if self._tilt_state in ("LEFT", "RIGHT"):
    if prev_state != self._tilt_state:
        self._tilt_start = now
    elif (
        self._tilt_start is not None
        and now - self._tilt_start >= self.hold_time
        and now - self._last_confirm >= self.cooldown_time
    ):
        confirmed = True
        self._last_confirm = now
        self._tilt_start = None
else:
    self._tilt_start = None

# Compute approximate face center from landmarks
xs = [int(p.x * width) for p in landmarks]
ys = [int(p.y * height) for p in landmarks]
if xs and ys:
    minx, maxx = min(xs), max(xs)
    miny, maxy = min(ys), max(ys)
    face_center = ((minx + maxx) // 2, (miny + maxy) // 2)
    face_size = (maxx - minx, maxy - miny)
```

Kode 10: State Machine untuk Konfirmasi Tilt dengan Hold Time

### 3.5 Modul `game_logic.py`

Modul `game_logic.py` mengatur logika inti permainan yang mencakup management soal, timer countdown, sistem scoring, dan state permainan. Modul ini dirancang independen dari GUI dan kamera sehingga mudah diuji secara unit testing. Semua perhitungan waktu menggunakan `time.monotonic()` untuk akurasi tinggi dan tidak terpengaruh perubahan system clock.

- Data Structure untuk Question dan Game State

  Dua dataclass utama adalah Question yang merepresentasikan satu soal lengkap dengan audio, waveform, gambar pilihan, dan jawaban benar; serta GameState yang menyimpan snapshot state game saat ini termasuk waktu tersisa, skor, index soal, dan status pause.

```python
@dataclass
class Question:
    id: str
    audio_path: Path
    waveform_data: Sequence[float]
    left_meme: Path
    right_meme: Path
    correct_side: str  # "LEFT" atau "RIGHT"

@dataclass
class GameState:
    is_running: bool
    remaining_time: float
    score: int
    current_index: int
    total_questions: int
    current_question: Optional[Question]
    is_paused: bool
```

Kode 11: Data Structure untuk Question dan Game State

- Inisialisasi Game Logic dengan Validasi

  Constructor `__init__()` menerima list Question dan durasi permainan dalam detik. Validasi dilakukan untuk memastikan minimal ada satu soal. Internal state mencakup tracking waktu mulai, index soal aktif, skor, status running, dan perhitungan akumulasi waktu pause untuk akurasi timing.

```python
class GameLogic:
    def __init__(self, questions: List[Question], duration_seconds: float = 30.0) -> None:
        if not questions:
            raise ValueError("Minimal butuh satu question untuk memulai game.")
        self._questions = questions
        self.duration_seconds = duration_seconds
        self._start_time: Optional[float] = None
        self._current_idx = 0
        self._score = 0
        self._is_running = False
        self._is_paused = False
        self._pause_start: Optional[float] = None
        self._pause_total: float = 0.0
```

Kode 12: Inisialisasi Game Logic dengan Validasi

- Sistem Timer dengan Pause Support

  Method `_remaining_time()` menghitung waktu tersisa dengan memperhitungkan waktu yang dihabiskan saat pause. Jika game di-pause, waktu berhenti di titik pause. Total waktu pause diakumulasi dan dikurangi dari elapsed time untuk mendapatkan waktu efektif yang telah berlalu. Game otomatis berhenti jika waktu habis.

```python
def _remaining_time(self) -> float:
    if not self._is_running or self._start_time is None:
        return self.duration_seconds
    now = time.monotonic()
    effective_now = (
        self._pause_start if self._is_paused and self._pause_start is not None else now
    )
    elapsed = effective_now - self._start_time - self._pause_total
    remaining = max(0.0, self.duration_seconds - elapsed)
    if remaining <= 0.0:
        self.stop_game()
    return remaining
```

Kode 13: Sistem Timer dengan Pause Support

- Submission Jawaban dan Scoring

  Method `submit_answer()` memvalidasi jawaban user dengan membandingkan side yang dipilih (LEFT/RIGHT) dengan `correct_side` dari soal aktif. Jika benar, skor bertambah 1. Setelah submission, index soal naik dan jika sudah mencapai akhir list, game otomatis berhenti. Return value boolean menandakan apakah jawaban benar atau salah.

```python
def submit_answer(self, side: str) -> bool:
    if not self._is_running or self._is_paused:
        return False
    question = self.current_question()
    if question is None:
        return False
    is_correct = side.upper() == question.correct_side.upper()
    if is_correct:
        self._score += 1
    self._current_idx += 1
    if self._current_idx >= len(self._questions):
        self.stop_game()
    return is_correct
```

Kode 14: Submission Jawaban dan Scoring

### 3.6 Modul `audio_manager.py`

Modul `audio_manager.py` adalah orchestrator untuk audio playback dalam game yang menghubungkan audio processing dengan pygame mixer. Modul ini bertanggung jawab untuk transformasi audio (pitch shifting), queue management clip audio, kontrol playback (play/stop/pause), dan menyediakan meta-data waveform ke GUI untuk visualisasi real-time.

- Data Structure untuk Audio Clip

  Dataclass AudioClip merepresentasikan satu clip audio lengkap dengan metadata yang diperlukan untuk playback dan visualisasi, termasuk path original dan processed, data waveform, durasi, dan sample rate.

```python
@dataclass
class AudioClip:
    """Data struktur untuk satu clip audio soal."""
    id: str
    original_path: Path
    processed_path: Path
    waveform_data: List[float]
    duration: float
    sample_rate: int
```

Kode 15: Data Structure untuk Audio Clip

- Inisialisasi Audio Manager dengan Pygame Mixer

  Constructor menginisialisasi pygame mixer dengan konfigurasi sample rate 44.1kHz, buffer size 512 untuk low latency, dan 2 channel stereo. Internal state mencakup queue management, current clip tracking, sound object untuk playback, callback notification, dan tracking background music.

```python
def __init__(
    self,
    sample_rate: int = 44100,
    buffer_size: int = 512,
    output_folder: Path = Path("asset_output")
) -> None:
    pygame.mixer.init(frequency=sample_rate, size=-16, channels=2, buffer=buffer_size)
    self._output_folder = output_folder
    self._output_folder.mkdir(parents=True, exist_ok=True)

    # Queue management
    self._audio_queue: List[AudioClip] = []
    self._current_clip: Optional[AudioClip] = None
    self._current_sound: Optional[pygame.mixer.Sound] = None

    # Callback untuk notifikasi
    self._on_finish_callback: Optional[Callable[[], None]] = None
    self._is_playing = False
    self._is_paused = False
    self._bgm_playing: bool = False
    self._bgm_path: Optional[Path] = None
```

Kode 16: Inisialisasi Audio Manager dengan Pygame Mixer

- Batch Audio Processing dengan Pitch Shifting

  Method `prepare_audio_clips()` melakukan batch processing file audio dengan memanggil `audio_processing.pitch_shift_file()` untuk transformasi pitch dan `generate_waveform_data()` untuk visualisasi. Hasil processing disimpan di output folder dengan naming convention yang konsisten. Method ini mengembalikan list AudioClip yang siap dimainkan.

```python
def prepare_audio_clips(
    self,
    audio_files: List[Path],
    semitones: float = -5.0,
    waveform_samples: int = 512
) -> List[AudioClip]:
    clips = []
    print(f"[AudioManager] Memproses {len(audio_files)} file audio...")

    for audio_file in audio_files:
        try:
            # Generate nama file output
            stem = audio_file.stem
            output_name = f"{stem}_pitch{int(semitones)}.wav"
            output_path = self._output_folder / output_name

            # Panggil audio_processing untuk pitch shift
            sr_out, duration = audio_processing.pitch_shift_file(
                input_path=audio_file,
                output_path=output_path,
                n_semitones=semitones,
                sr_target=None
            )

            # Generate waveform data untuk visualisasi
            waveform_data = audio_processing.generate_waveform_data(
```

```
                audio_path=output_path,
                num_samples=waveform_samples
            )

            # Buat AudioClip object
            clip = AudioClip(
                id=stem,
                original_path=audio_file,
                processed_path=output_path,
                waveform_data=waveform_data,
                duration=duration,
                sample_rate=sr_out
            )
            clips.append(clip)

        except Exception as e:
            print(f"[ERROR] Gagal memproses {audio_file.name}: {e}")

    return clips
```

Kode 17: Batch Audio Processing dengan Pitch Shifting

- Load dan Playback Management

  Method `load_clip()` memuat audio file ke memory sebagai pygame Sound object, siap untuk playback instant. Method `play()` memulai pemutaran dengan optional callback yang dipanggil ketika audio selesai. System juga menyediakan method `get_current_waveform()` dan `get_current_metadata()` untuk GUI mengakses data visualisasi dan informasi clip yang sedang dimainkan.

```
def load_clip(self, clip: AudioClip) -> bool:
    try:
        # Stop audio sebelumnya
        self.stop()

        # Load audio sebagai Sound object
        self._current_sound = pygame.mixer.Sound(str(clip.processed_path))
        self._current_clip = clip

        print(f"[AudioManager] Loaded: {clip.id}")
        return True

    except Exception as e:
        print(f"[ERROR] Gagal load clip {clip.id}: {e}")
        self._current_sound = None
        self._current_clip = None
        return False

def get_current_waveform(self) -> List[float]:
    if self._current_clip:
        return self._current_clip.waveform_data
    return []
```

Kode 18: Load dan Playback Management

## 3.7 Modul `meme_overlay.py`

Modul `meme_overlay.py` adalah komponen GUI untuk rendering sprite gambar kucing yang muncul di kiri dan kanan kepala pemain. Modul ini menangani animasi masuk/keluar (fade in/out dengan scale), highlight pilihan berdasarkan orientasi kepala, dan zoom effect saat sprite di-highlight. Semua rendering mendukung transparansi penuh (alpha channel) untuk hasil visual yang smooth.

- Class untuk Single Meme Sprite

  Class `MemeSprite` merepresentasikan satu sprite gambar dengan state animasi lengkap. Constructor memuat gambar dengan alpha channel (BGRA), resize dengan aspect ratio preservation, dan inisialisasi state animasi termasuk scale (0.0-1.0), highlight status, dan timing animation.

```python
class MemeSprite:
    def __init__(
        self,
        image_path: Path,
        side: str,
        target_size: Tuple[int, int] = (150, 150),
        highlight_zoom: float = 1.15,
    ):
        self.image_path = image_path
        self.side = side.upper()
        self.target_size = target_size
        self.highlight_zoom = float(highlight_zoom)

        # Load image dengan alpha channel
        self.original_image = cv2.imread(str(image_path), cv2.IMREAD_UNCHANGED)
        if self.original_image is None:
            raise FileNotFoundError(f"Tidak dapat load image: {image_path}")

        # Konversi ke BGRA jika belum
        if self.original_image.ndim == 2:
            self.original_image = cv2.cvtColor(self.original_image, cv2.COLOR_GRAY2BGRA)
        elif self.original_image.shape[2] == 3:
            bgr = self.original_image
            alpha = np.ones((bgr.shape[0], bgr.shape[1], 1), dtype=np.uint8) * 255
            self.original_image = np.concatenate([bgr, alpha], axis=2)

        # Resize dengan aspect ratio preservation
        self.image = self._resize_keep_aspect(self.original_image, target_size)

        # State animasi
        self.scale = 0.0
        self.is_highlighted = False
        self.animation_start_time: Optional[float] = None
        self.target_scale = 0.0
        self._start_scale: float = self.scale
```

Kode 19: Class untuk Single Meme Sprite

- Resize dengan Aspect Ratio Preservation

  Method `_resize_keep_aspect()` memastikan gambar ter-resize untuk fit dalam target size tanpa distorsi. Gambar di-scale proporsional berdasarkan dimensi yang lebih kecil, lalu ditempatkan di center canvas transparan berukuran target size. Ini menjaga kualitas visual dan proporsi original.

```python
def _resize_keep_aspect(self, img: np.ndarray, target_size: Tuple[int, int]) -> np.ndarray:
    tw, th = target_size
    h, w = img.shape[:2]

    # Hitung scale untuk fit ke dalam target (fit inside)
    scale = min(tw / w, th / h)
    new_w = max(1, int(w * scale))
    new_h = max(1, int(h * scale))

    resized = cv2.resize(img, (new_w, new_h), interpolation=cv2.INTER_AREA)

    # Buat canvas transparan BGRA
```

```
    canvas = np.zeros((th, tw, 4), dtype=np.uint8)
    # Tempatkan gambar di tengah canvas
    x_off = (tw - new_w) // 2
    y_off = (th - new_h) // 2
    canvas[y_off:y_off+new_h, x_off:x_off+new_w] = resized
    return canvas
```

Kode 20: Resize dengan Aspect Ratio Preservation

- Animasi Scale dengan Smooth Interpolation

  Method `update_animation()` mengupdate scale sprite berdasarkan elapsed time sejak animasi dimulai. Menggunakan smooth step easing function untuk transisi yang natural dari `_start_scale` ke `target_scale`. Duration default 0.3 detik memberikan feel yang responsive tanpa terlalu cepat.

```
def update_animation(self, current_time: float, duration: float = 0.3) -> None:
    if self.animation_start_time is None:
        return

    elapsed = current_time - self.animation_start_time
    if elapsed >= duration:
        # Animasi selesai
        self.scale = self.target_scale
        self.animation_start_time = None
        self._start_scale = self.scale
    else:
        # Interpolasi smooth
        t = elapsed / duration
        # Smooth step easing
        t = t * t * (3 - 2 * t)
        self.scale = self._start_scale + (self.target_scale - self._start_scale) * t
```

Kode 21: Animasi Scale dengan Smooth Interpolation

- Rendering Sprite dengan Alpha Blending

  Method `draw` merender sprite ke frame dengan posisi yang dihitung dari head position. Scale final adalah kombinasi dari animation scale dan highlight zoom. Alpha blending diterapkan untuk transparansi sempurna, dengan handling khusus untuk edge cases di boundary frame agar tidak terjadi crash atau artifact visual.

```
def draw(self, frame: np.ndarray, head_x: int, head_y: int, offset_x: int = 0, offset_y: int
    = 0) -> None:
    # Hitung posisi sprite berdasarkan head position dan offset
    final_scale = self.scale * (self.highlight_zoom if self.is_highlighted else 1.0)
    if final_scale <= 0:
        return

    # Scale sprite sesuai final_scale
    scaled_w = int(self.target_size[0] * final_scale)
    scaled_h = int(self.target_size[1] * final_scale)
    if scaled_w <= 0 or scaled_h <= 0:
        return

    scaled_img = cv2.resize(self.image, (scaled_w, scaled_h), interpolation=cv2.INTER_LINEAR)

    # Hitung posisi paste berdasarkan side
    if self.side == "LEFT":
        x = head_x - scaled_w + offset_x
    else:  # RIGHT
        x = head_x + offset_x
```

```
    y = head_y - scaled_h // 2 + offset_y
```

Kode 22: Rendering Sprite dengan Alpha Blending

## 3.8 Modul `waveform_view.py`

Modul `waveform_view.py` adalah komponen GUI untuk visualisasi waveform audio real-time. Modul ini menerima data waveform dari `audio_manager` dan merender visualisasi dalam dua style: bar chart atau line plot, dengan progress indicator animasi yang sinkron dengan playback audio. Semua elemen rendering dijamin berada dalam bounding box yang ditentukan untuk layout flexibility.

- Inisialisasi Waveform View dengan Customizable Style

  Constructor `__init__()` menerima parameter konfigurasi visual termasuk warna background, waveform, progress indicator, border, dan style visualisasi. Style "bars" menampilkan waveform sebagai bar chart vertikal, sedangkan "line" sebagai line plot kontinyu. Internal state mencakup data waveform, progress playback (0.0-1.0), dan status playing.

```
class WaveformView:
    def __init__(
        self,
        bg_color: Tuple[int, int, int] = (40, 40, 40),
        waveform_color: Tuple[int, int, int] = (100, 200, 255),
        progress_color: Tuple[int, int, int] = (0, 255, 255),
        border_color: Tuple[int, int, int] = (80, 80, 80),
        style: str = "bars"
    ):
        self.bg_color = bg_color
        self.waveform_color = waveform_color
        self.progress_color = progress_color
        self.border_color = border_color
        self.style = style

        # Waveform data
        self._waveform_data: List[float] = []
        self._playback_progress: float = 0.0
        self._is_playing: bool = False
```

Kode 23: Inisialisasi Waveform View dengan Customizable Style

- Data Management untuk Waveform dan Progress

  Method `set_waveform_data()` menerima list amplitudo ter-normalisasi dari audio manager dan menyimpannya untuk rendering. Method `set_playback_progress()` mengupdate progress animasi dengan nilai 0.0-1.0 yang di-clamp untuk keamanan. Method clear() mereset semua state untuk persiapan clip audio baru

```
def set_waveform_data(self, data: List[float]) -> None:
    self._waveform_data = data.copy() if data else []
    self._playback_progress = 0.0

def set_playback_progress(self, progress: float) -> None:
    self._playback_progress = max(0.0, min(1.0, progress))

def set_playing(self, is_playing: bool) -> None:
    self._is_playing = is_playing

def clear(self) -> None:
    self._waveform_data.clear()
    self._playback_progress = 0.0
```

```
    self._is_playing = False
```

Kode 24: Data Management untuk Waveform dan Progress

- Rendering Waveform dengan Bars Style

  Method internal untuk drawing bars style merender setiap sample waveform sebagai bar vertikal. Tinggi bar proporsional dengan amplitudo (0.0-1.0). Bar yang sudah di-play diberi warna berbeda (progress color) untuk visual feedback. Padding dan spacing antar bar dikonfigurasi untuk hasil yang aesthetic.

```
def _draw_bars_style(self, canvas, x, y, width, height, padding=10):
    if not self._waveform_data:
        return

    inner_width = width - 2 * padding
    inner_height = height - 2 * padding
    bar_area_height = inner_height - 20  # reserve space untuk border

    num_bars = len(self._waveform_data)
    bar_width = max(2, inner_width // num_bars)
    bar_spacing = 1

    progress_x = int(self._playback_progress * inner_width)

    for i, amp in enumerate(self._waveform_data):
        bar_x = padding + i * (bar_width + bar_spacing)
        bar_height = max(1, int(amp * bar_area_height))
        bar_y = padding + (bar_area_height - bar_height) // 2

        # Warna berubah jika sudah di-play
        color = self.progress_color if bar_x < progress_x else self.waveform_color

        cv2.rectangle(canvas, (bar_x, bar_y), (bar_x + bar_width, bar_y + bar_height), color,
 -1)
```

Kode 25: Rendering Waveform dengan Bars Style

- Main Draw Method dengan Boundary Protection

  Method `draw()` adalah entry point untuk rendering waveform ke frame OpenCV pada posisi (x, y) dengan ukuran (width, height). Method ini membuat canvas lokal, merender waveform sesuai style, menambahkan border jika diminta, lalu composite canvas ke frame target dengan boundary checking untuk mencegah out-of-bounds error.

```
def draw(
    self,
    frame: np.ndarray,
    x: int,
    y: int,
    width: int,
    height: int,
    show_border: bool = True
) -> None:
    # Validasi ukuran minimal
    if width < 20 or height < 20:
        return

    # Buat canvas lokal
    canvas = np.zeros((height, width, 3), dtype=np.uint8)
    canvas[:] = self.bg_color

```

```
    # Render waveform sesuai style
    if self.style == "bars":
        self._draw_bars_style(canvas, 0, 0, width, height)
    elif self.style == "line":
        self._draw_line_style(canvas, 0, 0, width, height)

    # Tambahkan border
    if show_border:
        cv2.rectangle(canvas, (0, 0), (width-1, height-1), self.border_color, 2)

    # Composite ke frame dengan boundary checking
    y1, y2 = max(0, y), min(frame.shape[0], y + height)
    x1, x2 = max(0, x), min(frame.shape[1], x + width)

    if y2 > y1 and x2 > x1:
        canvas_y1, canvas_y2 = y1 - y, y2 - y
        canvas_x1, canvas_x2 = x1 - x, x2 - x
        frame[y1:y2, x1:x2] = canvas[canvas_y1:canvas_y2, canvas_x1:canvas_x2]
```

Kode 26: Main Draw Method dengan Boundary Protection

## 3.9 Modul `generate_metadata.py`

Modul `generate_metadata.py` aadalah script utility untuk generasi metadata game yang membuat mapping antara file audio yang sudah di-pitch shift, gambar kucing yang sesuai, dan konfigurasi soal kuis (pilihan kiri/kanan dan jawaban benar). Output berupa file JSON yang digunakan oleh game untuk loading questions dan assets.

- Fungsi Utama Generate Game Metadata

  Fungsi `generate_game_metadata()` melakukan scanning folder asset untuk menemukan audio yang sudah diproses dan gambar kucing, lalu secara otomatis membuat soal dengan random pairing. Untuk setiap audio, script menentukan gambar correct answer berdasarkan ID matching, memilih satu gambar lain sebagai wrong answer, dan random placement di LEFT atau RIGHT.

```
def generate_game_metadata(
    asset_folder: Path = Path("asset"),
    output_folder: Path = Path("asset_output"),
    metadata_file: Path = Path("asset_output/game_metadata.json"),
    num_questions: int = 14
):
    print(f"[INFO] Generating game metadata...")

    # Cari semua file audio yang sudah diproses
    audio_files = sorted(output_folder.glob("audio-kucing*_pitch-*.wav"))
    image_files = sorted(asset_folder.glob("kucing*.png"))

    if not audio_files or not image_files:
        print(f"[ERROR] File tidak ditemukan")
        return

    print(f"[INFO] Ditemukan {len(audio_files)} audio, {len(image_files)} images")
```

Kode 27: Fungsi Utama Generate Game Metadata

- Parsing ID dan Matching Audio-Image

  Script melakukan parsing ID dari nama file audio (contoh: `"audio-kucing1_pitch-5.wav"` → ID: 1). ID ini digunakan untuk mencari gambar correct answer dengan pattern `"kucingID.png"`. Jika matching image tidak ditemukan, soal tersebut di-skip dengan warning message untuk debugging.

```python
# Extract ID dari nama file
audio_name = audio_path.stem
try:
    parts = audio_name.split("_")[0]  # audio-kucing1
    audio_id = int(parts.replace("audio-kucing", ""))
except:
    print(f"[WARN] Gagal parse ID dari {audio_name}, skip")
    continue

# Gambar yang sesuai (correct answer)
correct_image_name = f"kucing{audio_id}.png"
correct_image_path = asset_folder / correct_image_name

if not correct_image_path.exists():
    print(f"[WARN] Gambar {correct_image_name} tidak ditemukan, skip")
    continue
```

Kode 28: Parsing ID dan Matching Audio-Image

- Random Pairing dan Positioning

  Untuk setiap soal, script memilih satu gambar lain secara random sebagai wrong answer dari pool gambar yang tersedia. Position correct answer (LEFT atau RIGHT) juga dirandom untuk variasi gameplay. Durasi audio diambil menggunakan librosa dengan fallback handling untuk robustness.

```python
# Pilih gambar lain untuk wrong answer
other_images = [img for img in image_files if img.name != correct_image_name]
if not other_images:
    print(f"[WARN] Tidak ada gambar lain untuk soal {audio_id}, skip")
    continue

wrong_image_path = random.choice(other_images)

# Random posisi: correct di kiri atau kanan
correct_side = random.choice(["LEFT", "RIGHT"])

if correct_side == "LEFT":
    left_meme = str(correct_image_path).replace("\\", "/")
    right_meme = str(wrong_image_path).replace("\\", "/")
else:
    left_meme = str(wrong_image_path).replace("\\", "/")
    right_meme = str(correct_image_path).replace("\\", "/")

# Dapatkan durasi audio
try:
    duration = librosa.get_duration(path=str(audio_path))
except:
    duration = 3.0  # default fallback
```

Kode 29: Random Pairing dan Positioning

- Output Metadata JSON Structure

  Hasil akhir disimpan dalam struktur JSON terorganisir dengan metadata level atas (`version`, `total_questions`, `game_duration`) dan array questions berisi detail setiap soal. Setiap question entry mencakup ID, audio path, durasi, pasangan meme, correct side, dan correct image path untuk reference. File disimpan dengan encoding UTF-8 dan pretty-print indent untuk readability.

```python
# Buat question entry
question = {
```

```
    "id": f"q{i+1}",
    "audio_id": audio_id,
    "audio_path": str(audio_path).replace("\\", "/"),
    "duration": round(duration, 2),
    "left_meme": left_meme,
    "right_meme": right_meme,
    "correct_side": correct_side,
    "correct_image": str(correct_image_path).replace("\\", "/")
}

questions.append(question)

# Simpan metadata
metadata = {
    "version": "1.0",
    "total_questions": len(questions),
    "game_duration_seconds": 30,
    "questions": questions
}

with open(metadata_file, "w", encoding="utf-8") as f:
    json.dump(metadata, f, indent=2, ensure_ascii=False)

print(f"\n[DONE] Generated {len(questions)} questions")
```

Kode 30: Output Metadata JSON Structure

## 3.10 Modul `gui.py`

Modul `gui.py` adalah orchestrator utama game Purrfect Pitch yang mengintegrasikan semua komponen (face tracking, audio playback, game logic, meme overlay, waveform view) menjadi aplikasi interaktif lengkap. Modul ini menangani game loop, phase management, rendering visual, input handling, dan sinkronisasi antara deteksi wajah dengan gameplay.

- Game Phase Enum untuk State Management

  Enum `GamePhase` mendefinisikan semua state yang mungkin dalam game flow: `IDLE`, `PLAYING_AUDIO`, `WAITING_ANSWER`, `SHOW_FEEDBACK`, `START_POPUP`, `COUNTDOWN`, `GAME_OVER`, dan `PAUSED_NO_FACE`. Setiap phase memiliki behavior rendering dan logic yang berbeda, membuat state machine yang clear dan maintainable.

```
class GamePhase(Enum):
    """Phase/state game."""
    IDLE = "idle"
    PLAYING_AUDIO = "playing_audio"
    WAITING_ANSWER = "waiting_answer"
    SHOW_FEEDBACK = "show_feedback"
    START_POPUP = "start_popup"
    COUNTDOWN = "countdown"
    GAME_OVER = "game_over"
    PAUSED_NO_FACE = "paused_no_face"
```

Kode 31: Game Phase Enum untuk State Management

- Inisialisasi Comprehensive Game Components

  Constructor `__init__()` melakukan setup lengkap: load metadata dari JSON, inisialisasi face tracker dengan threshold dan timing yang sudah di-tune, setup audio manager dengan output folder, prepare questions dengan waveform data, inisialisasi game logic dengan duration 45 detik, setup meme overlay dan waveform view, serta load semua asset UI (start screen, game over, scoreboard, instruction board).

```python
def __init__(
    self,
    metadata_path: Path = Path("asset_output/game_metadata.json"),
    window_width: int = 1280,
    window_height: int = 720
):
    self.window_width = window_width
    self.window_height = window_height
    self.phase = GamePhase.START_POPUP

    # Load metadata
    self.metadata = self._load_metadata()
    if not self.metadata:
        raise RuntimeError(f"Gagal load metadata dari {metadata_path}")

    # Inisialisasi modul
    self.face_tracker = FaceTracker(
        tilt_threshold=12.0,
        hold_time=0.0,
        cooldown_time=0.5
    )

    self.audio_manager = AudioManager(output_folder=Path("asset_output"))
    self.questions = self._prepare_questions()
    self.game_logic = GameLogic(self.questions, duration_seconds=45.0)

    # Setup GUI components
    self.meme_overlay = MemeOverlay(offset_x=200, offset_y=-80, sprite_size=(240, 240))
    self.waveform_view = WaveformView(style="bars")

    # Load popup images
    self.start_img = self._load_image(Path("asset/start.png"))
    self.gameover_img = self._load_image(Path("asset/gameover.png"))
    self.scoreboard_img = self._load_image(Path("asset/score_board.png"))
    self.instruction_board_img = self._load_image(Path("asset/instruction_board.png"))
```

Kode 32: Inisialisasi Comprehensive Game Components

- Question Preparation dengan Waveform Generation

  Method `_prepare_questions()` mengiterasi metadata questions dan untuk setiap soal, memanggil `audio_processing.generate_waveform_data()` untuk membuat visualisasi audio. Hasil dikemas dalam object Question lengkap dengan audio path, waveform data, meme paths, dan correct side. List questions ini kemudian di-pass ke game logic.

```python
def _prepare_questions(self) -> List[Question]:
    qs = []
    for q in self.metadata.get("questions", []):
        wf = audio_processing.generate_waveform_data(
            Path(q["audio_path"]),
            num_samples=512
        )
        qs.append(Question(
            id=q["id"],
            audio_path=Path(q["audio_path"]),
            waveform_data=wf,
            left_meme=Path(q["left_meme"]),
            right_meme=Path(q["right_meme"]),
            correct_side=q["correct_side"]
        ))
    return qs
```

Kode 33: Question Preparation dengan Waveform Generation

- Game Flow Control dengan Phase Transitions

  Method-method seperti `start_game()`, `start_countdown_and_game()`, `_load_next_question()`, `_play_audio()`, dan `_on_audio_finished()` mengatur transisi antar phase. Countdown 3 detik sebelum game dimulai, load soal baru setelah jawaban dikonfirmasi, play audio dengan callback automatic transition ke `WAITING_ANSWER` phase, semuanya ter-orchestrate dengan smooth timing.

```python
def start_countdown_and_game(self):
    """Memulai countdown 3 detik sebelum soal pertama dimuat"""
    self.game_logic.start_game(shuffle=True)
    self.phase = GamePhase.COUNTDOWN
    self.countdown_start_time = time.time()

def _load_next_question(self):
    """Memuat soal berikutnya beserta audio dan gambar kucing"""
    self.answer_submitted = False
    s = self.game_logic.get_state()

    if not s.is_running or s.current_question is None:
        self.phase = GamePhase.GAME_OVER
        self.popup_start_time = time.time()
        return

    q = s.current_question
    clip = AudioClip(id=q.id, original_path=q.audio_path, ...)

    self.audio_manager.set_queue([clip])
    if self.audio_manager.load_clip(clip):
        self.waveform_view.set_waveform_data(q.waveform_data)

    self.meme_overlay.load_memes(q.left_meme, q.right_meme)
    self._play_audio()

def _on_audio_finished(self):
    """Callback dipanggil ketika audio selesai diputar"""
    self.phase = GamePhase.WAITING_ANSWER
```

Kode 34: Game Flow Control dengan Phase Transitions

- Face Tracking Callback dan Answer Submission

  Method `_on_face_tracking_update()` adalah callback yang dipanggil setiap frame oleh face tracker. Callback ini menerima `FaceTrackerState` dan frame kamera, lalu mentransformasi koordinat wajah dari camera space ke window space untuk positioning meme overlay. Jika tilt confirmed terdeteksi di phase `WAITING_ANSWER`, method `_submit_answer()` dipanggil untuk validasi dan feedback.

```python
def _on_face_tracking_update(self, st: FaceTrackerState, frame: np.ndarray):
    """Callback face tracking - update posisi overlay dan deteksi tilt"""
    # Scale koordinat wajah dari frame kamera ke window canvas
    fh, fw = frame.shape[:2]
    if st.face_detected and getattr(st, 'face_center', None):
        fx, fy = st.face_center
        sx, sy = self.window_width / fw, self.window_height / fh
        hx, hy = int(fx * sx), int(fy * sy)

        # Smooth position untuk mengurangi jitter
        if self._smoothed_head is None:
            self._smoothed_head = (hx, hy)
        else:
            alpha = 0.3
```

```
            shx, shy = self._smoothed_head
            self._smoothed_head = (int(shx * (1-alpha) + hx * alpha),
                                   int(shy * (1-alpha) + hy * alpha))

        # Update meme overlay position
        self.meme_overlay.set_head_position(*self._smoothed_head)

    # Handle tilt detection untuk answer submission
    if st.tilt_confirmed and self.phase == GamePhase.WAITING_ANSWER:
        if not self.answer_submitted:
            self._submit_answer(st.tilt_state)

def _submit_answer(self, side: str):
    """Submit jawaban dan tampilkan feedback"""
    self.answer_submitted = True
    is_correct = self.game_logic.submit_answer(side)

    self.feedback_message = "BENAR!" if is_correct else "SALAH!"
    self.feedback_start_time = time.time()
    self.phase = GamePhase.SHOW_FEEDBACK

    # Schedule next question atau end game
    self._scheduled_next_question_time = time.time() + self.feedback_duration
```

Kode 35: Face Tracking Callback dan Answer Submission

- Main Game Loop dengan Multi-Component Rendering

  Method `run()` adalah main loop yang menjalankan game sampai user quit. Loop ini: (1) memproses keyboard input, (2) menjalankan face tracker dengan callback, (3) membuat canvas kosong, (4) render camera feed dengan transformasi, (5) render meme overlays dengan animasi, (6) render waveform dengan progress, (7) render UI overlays (timer, skor, feedback, popup), (8) display ke window dengan `cv2.imshow()`, (9) handle scheduled transitions, (10) update background music. Semua berjalan smooth di 30+ FPS.

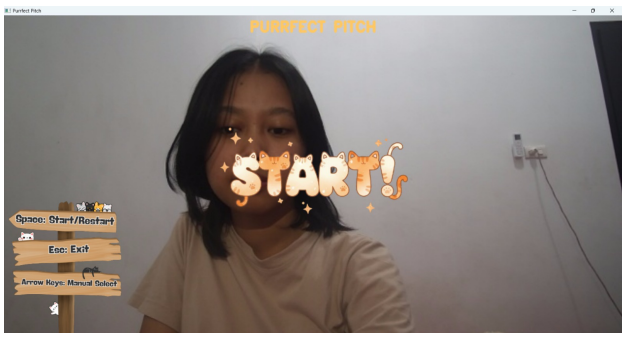
```
def run(self):
    """Main game loop"""
    print("[INFO] Starting game...")
    self.face_tracker.start()

    try:
        while True:
            # Baca frame dari kamera
            ret, frame = self.face_tracker._cap.read()
            if not ret:
                break

            frame = cv2.flip(frame, 1)

            # Evaluasi face state
            face_state = self.face_tracker._evaluate_state(frame)

            # Call callback untuk update game state
            self._on_face_tracking_update(face_state, frame)

            # Buat canvas untuk rendering
            canvas = np.zeros((self.window_height, self.window_width, 3), dtype=np.uint8)

            # Render camera feed
            self._render_camera_to_canvas(frame, canvas)

            # Render meme overlays
```

```
            self.meme_overlay.update()
            self.meme_overlay.draw(canvas)

            # Render waveform
            if self.phase in [GamePhase.PLAYING_AUDIO, GamePhase.WAITING_ANSWER]:
                self._render_waveform(canvas)

            # Render UI overlays (timer, score, feedback)
            self._render_ui_overlays(canvas)

            # Render popup screens (start, countdown, game over)
            self._render_popup_screens(canvas)

            # Display canvas
            cv2.imshow("Purrfect Pitch", canvas)

            # Handle keyboard input
            key = cv2.waitKey(1) & 0xFF
            if key == 27:  # ESC
                break
            elif key == ord(' '):  # SPACE
                self._handle_space_key()

            # Handle scheduled transitions
            self._update_scheduled_events()

            # Update BGM
            self._update_background_music()

    finally:
        self.face_tracker.stop()
        self.audio_manager.stop()
        cv2.destroyAllWindows()
```

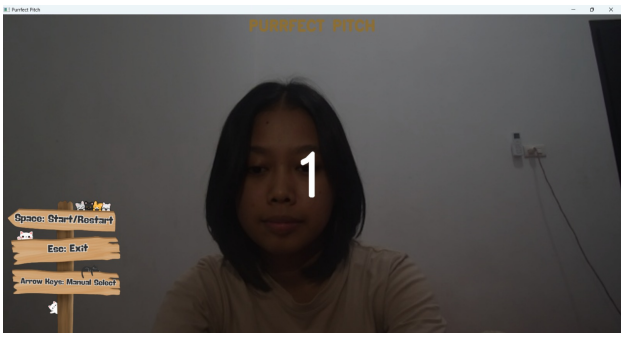
Kode 36: Main Game Loop dengan Multi-Component Rendering

# 4 Hasil

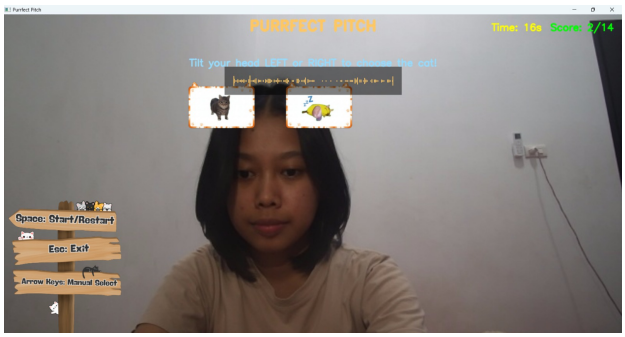
Hasil penerapan dari filter yang kami kembangkan ditunjukkan pada Gambar berikut.



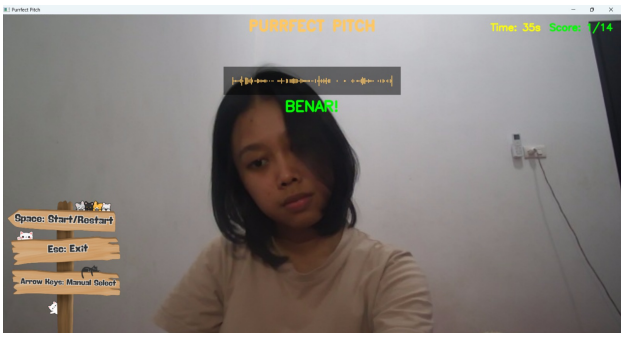

(a) Tampilan Start



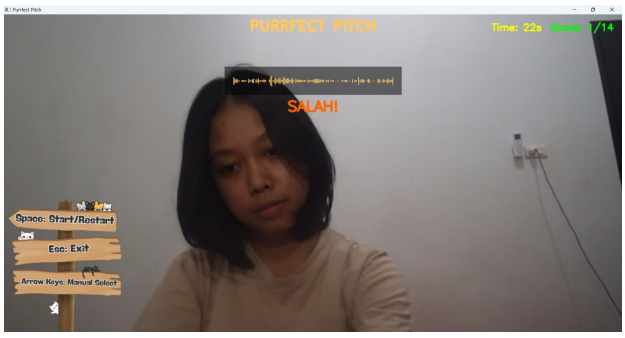

(b) Countdown filter




(c) Tampilan filter




(d) Jawaban benar




(e) Jawaban salah


(f) Tampilan Game Over

Gambar 1: Hasil tampilan aplikasi pada berbagai tahap penggunaan.

Purrfect Pitch yang dikembangkan mampu membaca kemiringan kepala dengan stabil sehingga proses pemilihan gambar kucing dapat dilakukan sesuai gesture yang ditetapkan. Pada tampilan awal (Gambar a), aplikasi menampilkan tampilan awal ketika game mulai dijalankan. Terdapat instruksi untuk menjalankan game di sebelah kiri. Ketika pengguna memilih Start (menekan spacebar) hitungan mundur sebelum filter dimulai akan muncul (Gambar b). Setelah hitung mundur selesai, sistem mulai memutar suara kucing yang akan ditebak dan menampilkan dua opsi gambar kucing di bagian sisi kanan dan kiri pengguna (Gambar c). Waveform dari setiap audio juga akan ditampilkan.

Pengguna akan memilih kucing yang benar dengan memiringkan kepala. Gesture ke kiri digunakan untuk memilih gambar di sisi kiri, sedangkan gesture ke kanan digunakan untuk memilih gambar di sisi kanan. Contoh deteksi kemiringan kepala saat pengguna memilih jawaban yang benar terlihat pada (Gambar d) dan apabila memilih jawaban salah akan terlihat seperti pada (Gambar e). Di akhir sesi, aplikasi menampilkan tampilan akhir berupa pesan "GAME OVER!" beserta akumulasi skor yang diperoleh pengguna (Gambar f). Tahap ini menandai bahwa waktu telah habis atau seluruh soal telah dijawab.

Implementasi *MediaPipe Face Mesh* memungkinkan filter membaca *roll angle* kepala dengan cukup akurat melalui analisis *landmark* wajah. Terdapat mekanisme *cooldown* untuk mencegah input terbaca dua kali secara berturut-turut, sehingga interaksi selama penggunaan filter terasa lebih stabil dan nyaman. Selain itu, Sistem juga memiliki kemampuan untuk mendeteksi keberadaan wajah pengguna. Apabila tidak terdapat wajah yang terdeteksi, maka seluruh proses yang sedang berjalan serta tampilan filter akan dihentikan dan tidak ditampilkan.

# 5 Kesimpulan

Pada tugas besar ini, sebuah filter interaktif bernama Purrfect Pitch berhasil dirancang dengan memanfaatkan sistem *face tracking* sebagai sumber input utama. Proses pengembangannya melibatkan integrasi berbagai elemen multimedia mulai dari pengolahan audio, teknik *computer vision*, hingga visualisasi yang ditampilkan secara *real-time*. Kombinasi elemen-elemen tersebut memungkinkan filter merespons gerakan kepala pengguna secara tepat dan menghasilkan interaksi yang sesuai.

Penerapan library seperti *MediaPipe*, *librosa*, dan *OpenCV* sangat berperan dalam membangun fungsi inti yang dibutuhkan. Setiap library menyediakan fitur yang mempermudah implementasi komponen teknis proyek sehingga pengerjaan dapat disusun lebih terarah. Kami juga menghaturkan terima kasih kepada Bapak Martin atas materi dan penjelasan beliau selama perkuliahan Multimedia, yang menjadi landasan dalam pengerjaan tugas ini. Selain itu, bantuan LLM turut dimanfaatkan untuk mendukung proses penulisan dan dokumentasi agar laporan dapat disusun dengan lebih rapi.

## References

https://drive.google.com/drive/folders/1-WDRP_IlDrk4atNv0IpnJ-WCOFbt6oAg?usp=drive_link