Project Explanation

PROJECT OVERVIEW - LIP-SYNC ANIMATION SYSTEM

==========

Hi! Thanks for checking out my Unity assignment. I wanted to explain the systems I built and the reasoning behind the architecture. This project focuses on creating a flexible lip-sync system that works with blend shapes on 3D character models.

The Model I used is has only blendshapes not a face rig so I designed the whole facial expression based around the blendshapes.

CORE SYSTEMS

==========

1. LOGIN PAGE (LoginManager.cs)

This is a simple pseudo-login screen. There's a bool toggle that determines success/failure.

Why fake it?

- Real authentication was out of scope for this assignment

- I focused on making the UI feel polished with DOTween animations (shake on failure, scale bounce on success)

- The system still has proper event callbacks (OnLoginSuccess, OnLoginFailed) so it could be connected to a real backend later

Key features:

- Smooth animations using DOTween (shake, scale, fade effects)

- Clean state management with bool flag

- Event-driven architecture for extensibility

- Scene transition after successful login

2. FACIAL EXPRESSION SYSTEM (FacialExpressionSystem.cs)

This is the foundation of everything. I built it as a centralized controller for all facial animations on the character.

Why this approach?

- The character model has multiple SkinnedMeshRenderers (Body, Eyes, Teeth, Tongue, etc.), and I needed a single system to manage blend shapes across all of them

- I implemented a caching system because searching through blend shapes every frame would be too expensive

- There's actually TWO cache dictionaries: one for unique blend shape names, and another for mesh-specific targeting (since some blend shapes like "jawOpen" exist on both Body and Teeth meshes)

Key features:

- Automatically finds all SkinnedMeshRenderers on the character at startup

- Supports both immediate and smooth transitions between expressions

- Has an auto-blink system that runs independently (eyes blink naturally while other animations play)

- All blend shape values are tracked in a dictionary so we always know the current state

2a. FACIAL EXPRESSION DATA (FacialExpressionData.cs)

This is a ScriptableObject that stores what an expression looks like.

Why ScriptableObjects?

- I wanted expressions to be reusable assets. You can create a "Happy" expression once and use it on multiple characters

- Artists or designers can create new expressions without touching code

- Easy to version control and share between team members

Each expression is just a list of blend shape names and their target weights, plus a transition duration. Simple but effective.

3. LIP-SYNC SYSTEM (SimpleLipSync.cs + LipSyncData.cs)

This was the main challenge. I needed fake lip-sync that looks convincing without doing complex audio analysis.

My approach:

- Instead of analyzing audio frequencies (which is complex and CPU-intensive), I went with a duration-based system

- The system plays an audio clip and animates blend shapes for a specified duration

- Each blend shape can be either "static" (stays at one value) or "animated" (randomly oscillates between min/max values)

Why this design?

- Real lip-sync would require FFT analysis, phoneme detection, etc. - overkill for this project

- Random oscillation between ranges actually looks pretty natural when tuned right

- I used DOTween for smooth easing between values - this was crucial because direct Lerp looked too choppy

LipSyncData is also a ScriptableObject so you can have different "talking styles" (whisper, shout, etc.) as separate assets.

The blend shapes update at configurable intervals (default 0.3 seconds) which mimics the rhythm of speech without being frame-perfect.

4. ANIMATION HANDLER (AnimationHandler.cs)

This coordinates everything - animator triggers, facial expressions, and lip-sync.

It's essentially the "director" that says:

- "Play the Sad expression and trigger the Sad animation"
- "Wait 2 seconds, then transition to Happy"
- "Start talking with lip-sync"

I built it to handle a reaction sequence (Sad → Happy → Sad → Idle) which demonstrates how all the systems work together.

5. EDITOR TOOLS

==========

I spent a lot of time making the editor tools user-friendly because I wanted to avoid manual string entry (super error-prone).

LipSyncDataEditor.cs:

- Scans the scene for all blend shapes automatically

- Shows them in a categorized, scrollable list

- Displays the mesh name for duplicates (so you can tell "jawOpen (Body)" from "jawOpen (Teeth)")

- Click a blend shape to add it, then configure if it's static or animated

FacialExpressionDataEditor.cs:

- Similar scanning system for expressions

- Organized by categories (Mouth, Eye, Teeth, etc.)

- Search filter to quickly find specific shapes

These editors save so much time compared to typing blend shape names manually.

TECHNICAL DECISIONS

==========

1. DOTween Integration

I chose DOTween for all animations because:

- Much smoother than manual Lerp

- Configurable easing curves (InOutSine, OutBack, etc.) make animations feel more natural

- Easy to kill/restart tweens without managing coroutines manually

2. Dual Cache System

The two dictionary approach for blend shapes might seem over-engineered, but it solved a real problem:

- Some blend shapes appear on multiple meshes with the same name

- We need to target them individually for lip-sync (jaw on both body and teeth)

- First cache handles simple cases, second handles specific mesh targeting

3. Coroutine-Based Animation

Using coroutines for the lip-sync timing made the most sense:

- Easy to pause/stop

- Can check gameObject.activeInHierarchy before starting (avoids errors)

- Simple to add delays between animation steps

4. ScriptableObject Architecture

Everything that's "data" (expressions, lip-sync configs) is a ScriptableObject:

- Clean separation of data and logic

- Easy to create variants

- Supports Unity's asset workflow

CHALLENGES & SOLUTIONS

==========

Challenge 1: Choppy Animations

Initially used Lerp for blend shape transitions - looked robotic.

Solution: Switched to DOTween with easing curves. Game changer.

Challenge 2: Duplicate Blend Shape Names

jawOpen existed on both Body and Teeth meshes, causing conflicts.

Solution: Added mesh name tracking and a second cache dictionary with (name, mesh) keys.

Challenge 3: Coroutine Errors on Inactive Objects

Tried to start coroutines when GameObject was disabled, crashed.

Solution: Check gameObject.activeInHierarchy before starting, with immediate fallback.

Challenge 4: Error-Prone Manual Entry

Typing blend shape names led to constant typos and bugs.

Solution: Built custom editors that scan and display all available options.

WHAT I'D IMPROVE WITH MORE TIME

==========

1. Real audio analysis - proper FFT with phoneme mapping
2. Preview system in the editor (see expression changes without entering Play Mode)
3. Expression blending (mix two expressions at different intensities)
4. Timeline integration for sequencing complex animations
5. Optimization - object pooling for tweens, better caching strategies

CLOSING THOUGHTS

==========

The goal was to create a flexible, artist-friendly system for facial animation that doesn't require frame-by-frame animation or complex audio processing. I think the architecture achieves that - it's easy to create new expressions and lip-sync configurations without touching code.

The editor tools were crucial to making this usable. Nobody wants to type "jawOpen_Body_Mesh" fifty times.

Everything is built with Unity best practices - component-based design, ScriptableObjects for data, proper lifecycle management, and a clean separation of concerns.

Hope this gives you a good understanding of how everything fits together!