■ Fighting Game Project Coding Standard

1. Project Structure

Use a consistent directory layout to keep systems modular and maintainable.
Assets/
■■■ Characters/
■■■ Systems/
■ ■■■ Combat/
■ ■■■ Input/
■ ■■■ AI/
■ ■■■ Physics/
■ ■■■ UI/
■ ■■■ VFX/
■■■ ScriptableObjects/
■■■ Resources/
■■■ Editor/

2. Naming Conventions

Classes: PascalCase → FighterController
Methods: PascalCase → ApplyHitstun()
Variables: camelCase → currentMove
Constants: ALL_CAPS → MAX_HEALTH
Enums: PascalCase → FighterState.Attack
Events: On + Verb → OnHitLanded

3. Code Style Rules

• Always use braces even for one-liners.
• Each script should have a clear single purpose.
• Functions < 30 lines whenever possible.
• Comment only non-obvious logic.
• Use FixedUpdate() for frame-precise systems.

4. Script Guidelines by System

Input: Wrap player actions in Command objects.
Combat: Centralize hitbox logic, frame-precise updates.
Physics: Handle pushbox and collision resolution manually.
AI: Execute commands same as player for deterministic replay.
Animation: Trigger via parameters, not clip names.

5. Command Pattern Framework

```
interface ICommand {
void Execute(FighterController fighter);
void Undo(FighterController fighter);
}
```

Example Commands:
• LightPunchCommand : Performs a light punch.
• JumpCommand : Performs a jump action.

CommandInvoker processes queued commands during FixedUpdate().

6. Coding Practice Standards Summary

■ Frame-based determinism (FixedUpdate).
■ Data-driven moves (MoveData, not hardcoded).
■ One responsibility per class.
■ No scene hard references.
■ Training tools for hitbox/frame debugging.