Unreal Developer Task - System Architecture & Assessment

Candidate: Anderson

The core of this project, developed in Unreal Engine 5.5.4, is built upon a modular C++ architecture. The main gameplay logic is centralized in **ASkaterCharacterBase**, an extension of **ACharacter**. I implemented a custom state machine (**ESkaterMovementState**) to manage key behaviors: Coasting, Accelerating, and Braking. This involves manually calculating physics forces, like friction and deceleration, within the **Tick** function to accurately simulate momentum.

Scoring System and Data-Driven Design

The scoring mechanism is designed using an Interface-based approach (**ICollectable**, **IArtifact**, **IPointSystem**). This crucial design choice ensures decoupling, allowing any actor to be defined as collectable without creating hard dependencies on the "Collector". Furthermore, artifacts are Data-Driven using **UArtifactData** (Data Assets). This empowers designers to modify point values and display names external to the code, eliminating the need for recompilation. The UI (**USkaterHUD**) updates asynchronously via an event-driven observer pattern, binding to delegates in **ASkaterPlayerState**.

Architectural Philosophy (Thought Process)

My primary objective was to uphold strict C++ standards and maintain a clean, organized codebase. I consistently favored creating reusable components (such as **UCollectionFeedbackComponent**) and Interfaces over direct type casting. Blueprints were intentionally limited to visual composition (Mesh configuration) and Input Mapping contexts (**IMC_Default**), with all core logic being implemented in C++.

Personal Assessment & Performance Review

I am confident that the code's technical quality is high. The class hierarchy is scalable, and the system adheres to Unreal best practices, particularly in its separation of concerns (e.g., managing points in **PlayerState** versus movement in **Character**).

However, I must acknowledge a significant personal challenge this week that impacted my workflow. In my intense focus to meet the functional requirements within the remaining time, I entered a "sprint" mode and consequently failed to commit my changes with the desired granularity. While the code quality remains unaffected, I recognize that the resulting version control history does not ideally reflect the iterative nature of the development process, showing fewer, larger commits than my typical standard.

Time Investment

- Core Character Movement (C++): ~6 Hours
- Collection System & Interfaces: ~3 Hours
- UI & Game Mode Setup: ~2 Hours
- Asset Integration (Animation/Map): ~2 Hours
- Debugging & Polish: ~2 Hours
- Total Time: ~15 Hours