Technical Task – System Overview & Self Assessment

System Overview

For this technical task, I focused on building a modular and scalable gameplay architecture while fully meeting the required inventory and persistence features.

The player is driven by a state-based system using ScriptableObjects. Each `State` defines a set of `Skill` assets, which are dynamically instantiated at runtime. Skills are decoupled from input and player logic through interfaces (`ISkillComponent`, `IInputSkillComponent`, `IMove`), allowing new abilities to be added or swapped without modifying existing systems. Input is handled using Unity’s new Input System and mapped dynamically to skills depending on the active state.

Movement is processed frame-by-frame through a dedicated movement component, ensuring responsive controls while keeping input, movement, and animation responsibilities separated. Events are used extensively to decouple systems such as health, state changes, UI updates, and player death.

The inventory system is slot-based and fully interactive. Items can be added, removed, dragged, swapped, equipped, or used directly within the UI. Each slot has a fixed index, ensuring deterministic behavior and reliable persistence. Item details are displayed dynamically through a dedicated UI panel when hovering or selecting an item.

A save and load system was implemented to persist the complete inventory state. All slots are serialized and restored on game start, preserving item positions and equipped items. Special care was taken to avoid save conflicts during the loading process.

Personal Assessment

I am satisfied with the final result, especially the system architecture and code organization. While the task was open-ended, I prioritized clean separation of responsibilities, extensibility, and robustness. Given more time, I would expand world interactions and polish animations further, but I believe this prototype accurately reflects my approach to scalable gameplay systems and production-ready code.