Technical Test

Technical Explanation of the System

I developed a prototype in Unity (version 6000.0.35f1) that includes character movement, a slot-based inventory, interactions, and a save/load system. Movement was implemented using Unity's standard input system, with animations reflecting the character's actions, such as walking. The inventory consists of UI slots that allow adding, removing, moving, and swapping items via drag & drop, as well as using or equipping them.

To ensure data persistence across scenes and manage dependency injection, I chose to use Extenject. The save system uses binary serialization to store the inventory state and player attributes in local files, ensuring player progress is maintained between sessions.

In the prototype, the player can interact with an NPC disguised as a statue by approaching it and triggering an event. Additionally, the player's lantern automatically attacks when near a zombie located in the cemetery, creating dynamic environmental interaction.

Development Reasoning

My priority was to keep the code organized and modular, clearly separating responsibilities between models and views to facilitate future improvements. At the start of the project, I focused on building the gameplay environment and UI screens before fully implementing the programming logic.

Game actions communicate through Unity Events, and thanks to dependency injection, I limited event usage only to classes involved in communication, keeping the code clean. For data persistence, I selected binary serialization due to its efficiency handling complex structures, especially since I used generic lists for inventory and equipment, which helped manage data dynamically.

The interface was designed to be clear and responsive, providing immediate visual feedback and organizing essential information cohesively.

Throughout development, I made frequent GitHub commits to document progress. The most time-consuming part was the save and load system because the inventory is not fully populated and many Unity assets are not serializable, which required creating helper classes and loading resources from the Resources folder.

Self-Assessment

The prototype is stable and meets the requirements, with a functional inventory, efficient save system, and well-implemented event communication. The code structure allows easy expansion, though there are still opportunities for organization and refactoring.

Areas for improvement include adding more elaborate sound and visual effects, making the game design more intuitive, and optimizing graphics. The UI could also be enhanced to provide a more appealing visual experience.