Unity Programmer Task

David Guillermo Gonzalez Breton – Unity Developer

davidguibregon@gmail.com

11/12/2025

Resume

This project implements multiple interconnected systems for a 2D video game developed in Unity 6.2.1f1. It includes systems such as inventory, dialogue, save/load, and NPC interactions. All these systems follow the MVC (Model–View–Controller) architecture to ensure a clear separation of responsibilities, simplify maintenance, and support scalable code development. Dialogue runs through ScriptableObject nodes that allow branching conversations, and NPC interactions use a simple interface-based approach. Player Stats update through events and support regeneration and UI feedback.

The system layers are divided into three:

Cape	Responsability	Example
Model	Business logic and data.	InventoryModel, DialogueModel and StatModel
View	Visual representation in the UI	InventoryView, DialogueView and StatBarView
Controller	Brain of the application	InventoryController, DialogueController and StatController

Core InventorySystem:

The inventory uses two separate containers, one for general storage and another for quick access during gameplay. The main inventory has configurable slots where players keep their stuff, and the hotbar gives them nine slots they can switch between with keys 1 through 9.

On the data side, Inventory Model and HotbarModel each manage their own list of InventorySlot. They handle the usual operations: adding, removing, moving items around, swapping slots. Whenever something changes, they fire off events so the rest of the code knows to update.

Items themselves are split into two parts, ItemInstace is what actually sits in a slot it points to and ItemDefinition, which is a ScriptableObject holding things like the item's name, icon, how many can stack, and what happens when you use it. All definitions live in ItemDatabase, so looking up an item by ID is fast and everything stays consistent.

For the UI, the views create slot elements based on how big the inventory is set to be, and they refresh automatically when the models change. Each slot view deals with clicks, hovering, drag and drop, basically translating Unity's UI events into something the controllers understand. There's also a tooltip that pops up with item info, plus world-space views for items sitting in the scene (these have unique IDs so the save system can track them).

Controllers sit in the middle and tie everything together. They handle input, opening/closing the inventory, picking which hotbar slot is active, using items. When the player picks up or drops something, specialized controllers manage that and pull from an Object Pool instead of spawning new objects every time. Recycling world items this way keeps things running smoothly.

Personal assessment:

I'm satisfied with the result. Things connect how I wanted them to, the code doesn't feel like spaghetti, and adding new features shouldn't require rewriting half the project. This is pretty much how I like to work, get the foundation right first, then iterate.