About the project

I was recently playing Valheim, so that was my main inspiration for what to do for this demo, for the basic resource gathering and UI.

I wanted the basic mechanics for collecting resources and equipping items, so I first looked for assets to match the most basic things I could think of: wood, axe and trees.

This was my first time using the terrain tool in Unity, I only used landscaping from Unreal, so I had some problems with it.

The primary focus was speed because of the time constraint of 48h, so code quality suffered, especially on the second day, so there's a lot of room for improvement.

I believe the basic inventory system works great, as well as the item assets creation and database. I made custom editors to help populate and manipulate them.

Item data is separated in what defines the item and is shared (icons, prefab, max stacks...) and what is the runtime data (quantity, instance id).

Shared data could have one more separation for assets that need loading like icons and prefabs, to support content versioning/delivery via addressables.

The worst part of the code is definitely input handling, it's really messy. There are parts using InputActions, there's one piece of code using the old Input api and the UI is using

the IPointer interfaces, so there are 3 different input sources. There is also no gamepad support.

The inventory UI works nicely with the inventory controller and inventory slots, but there is no View/ViewController separation, only the data layer is properly separated, in the form of ItemData.

Save system is really simple and crude but works well enough. The most obvious improvement would be to have proper save points so the code doesn't write to file every change. Since I have no save screens, every modification

must be saved for now. Chopped trees are not yet saved but the same method to save picked objects could be used: generate a tree instance id, save chopped instances and them destroy them (or better yet, don't even load them).

Also, all trees are loaded from start, which is bad, but at least they have proper LOD.

Movement has some jank, but it works, so I left as is, but this should definitely be very well polished for a final product. The jank is also worse on build compared to editor.

The player controller class started getting a bit too many responsibilities. There's a bunch os logic that could be moved to other places. The controlled should only listen to input and fire off events.

Generally I also write automated tests for most code and avoid writing everything in MonoBehaviours, I prefer pure C# classes and using dependency injection, but a setup like this requires the implementation of an orchestrator, so I went with the simplest way.

Everything is properly separated into sensible assemblies, and there is no cyclic dependencies.