Bag Inventory System – Phase Roadmap

This document outlines what has already been completed and what remains for each phase of the Bag Inventory project. It is intended as a reference so the project can be continued or resumed in a new session without losing context.

Phase 1 – Core Inventory Functionality (COMPLETED)

Goal: Establish a functional, console-based inventory system with safe core operations.

Completed

- Defined `Bag` class with internal `std::vector<Item>` storage
- Created `Item` data structure (name, quantity, id/index)
- Implemented:
- Add item
- Remove item by index
- View inventory list
- Input validation for menu selection
- Basic loop-driven menu system
- Correct handling of invalid indices
- Git commit with clear separation of `bag.h` and `bag.cpp`

Outcome

A stable, usable inventory system that supports basic CRUD-style operations.

Phase 2 – Safety, Validation, and API Cleanup (COMPLETED)

Goal: Make the core system safer, cleaner, and reusable.

Completed

- Refactored logic out of `main` into `Bag` methods
- Added bounds checking before vector operations
- Used `const` correctness where applicable
- Ensured removal logic deletes the correct element (no off-by-one bugs)
- Improved function naming and intent clarity
- Reduced duplicate logic in switch cases

Outcome

A clean internal API suitable for reuse outside a console application.

Phase 3 – Persistence (Save / Load) (COMPLETED)

Goal: Allow inventory to persist between runs using a text file.

Completed

- Implemented `SaveToFile(const std::string& filename)`
- Implemented `LoadFromFile(const std::string& filename)`
- Chose and enforced file format:
- `itemName|quantity`
- Correct parsing of item names containing spaces
- Defensive handling of:
- Missing files
- Empty lines
- Malformed data
- Inventory reset before load (no duplication)
- Verified save/load round-trip correctness
- Git commit marking Phase 3 completion

Outcome

Inventory state can be safely saved, restored, and verified.

Phase 4 – Edge Cases & Data Robustness (NEXT)

Goal: Harden the system against misuse and unexpected input.

Steps 1) Quantity rules - Prevent negative quantities - Define behavior for zero quantity (remove item vs keep at 0) - Decide max quantity per stack

2) Duplicate item behavior - Add by name: merge into existing stack OR create a new entry (pick one) - Case-insensitive name matching option

3) Item type + stack rules per type (v1) - Add an ItemType enum (Armor, Consumable, Material, Quest, etc.) - Define stackability rules per type (e.g., Armor non-stackable, Consumables stackable)

4) Item metadata (v1) - Add fields such as: type, rarity, value, weight (optional), description (optional) - Ensure save/load supports these fields safely

5) Stronger return/status reporting - Add return status enums (Success / Failure / InvalidInput / NotFound)

Outcome A production-safe core that behaves predictably under stress, with stack rules and metadata needed for a real v1.

Phase 5 – Engine-Ready Structure & UI Demo (PLANNED)

Goal: Make the system easy to integrate into engines and demonstrate it with a simple GUI.

Steps 1) Separate concerns - Remove console I/O from core logic (no cin/cout inside Bag) - Keep core as pure functions/methods returning statuses/data

2) Expose a clean API - Methods like: AddItem, RemoveById, RemoveByIndex, UpdateQuantity, FindByName, GetItems - Optional callbacks/events (OnItemAdded/Removed) as a pattern (not required)

3) GUI-driven inventory window (v1) - Provide a wxWidgets demo app: - Inventory list/grid view - Drag-and-drop reordering within the inventory - Search box (filter by name) - Sorting options (name, type, quantity) - Buttons: Add, Remove, Save, Load - Visual feedback for invalid actions - Keep it as a demo layer on top of the core (core remains engine-agnostic)

4) Integration notes - Document how to drop the core into Unreal/other engines

Outcome An engine-agnostic inventory module plus a working GUI demo that proves usability.

Phase 6 – Productization & Release (PLANNED)

Goal: Prepare the system for sale and public use.

To Do

- Final code cleanup and formatting
- README documentation:
- Features
- File format
- Integration steps
- Example usage code
- License file
- Folder structure for distribution
- Final testing pass

Outcome

A sellable v1 inventory core suitable for marketplaces (Fiverr, itch.io, Gumroad).

Optional Future Add-ons (POST v1)

- Equipment slots (armor, weapons, accessories)
- Weight / encumbrance system
- Crafting or durability hooks

- Advanced UI polish (icons, tooltips, rarity colors)
- Save format migration (v1 → v2)

Current Status

- **Completed:** Phases 1–3
- **Active Next Phase:** Phase 4
- **Target Product:** Core Inventory v1 ($30) + optional add-ons

---

End of document.