Too Many Eyes / Crafting System

System Design Document

# Changes

## V 1.0

Editor: Perrin Peterson

12/09/2022

* Document Created, and initial version

# Introduction

This document details the design and purpose of a locomotion system designed for Too Many Eyes. Designed for the Blight Brew Game, the document will cover;

* Usage for the designers.
* UML for the Programmers.

# Design Goals

The goal of this system is to be a bridge between the Potion, and the Resource systems;

* It gives the player a way of transforming the basic resources into another item, or potion.
* It gives the player a way of upgrading an existing potion to one of a higher potency, or a different usage.

# Behaviour

The crafting system handles is, of course, easily customised in the editor by the designers. The system is designed to handle the backend of crafting and is easy to hook into the UI system. The crafting currently works as such;

* A+B+C=output
* If any items are incorrect, the items are consumed and no output is given.
* If the recipe is correct but the items are in the wrong order, the items aren’t consumed, but the item isn’t given to the player.
* If the recipe is correct, and in order, the item output is rewarded and placed in a output slot to be grabbed by the player.

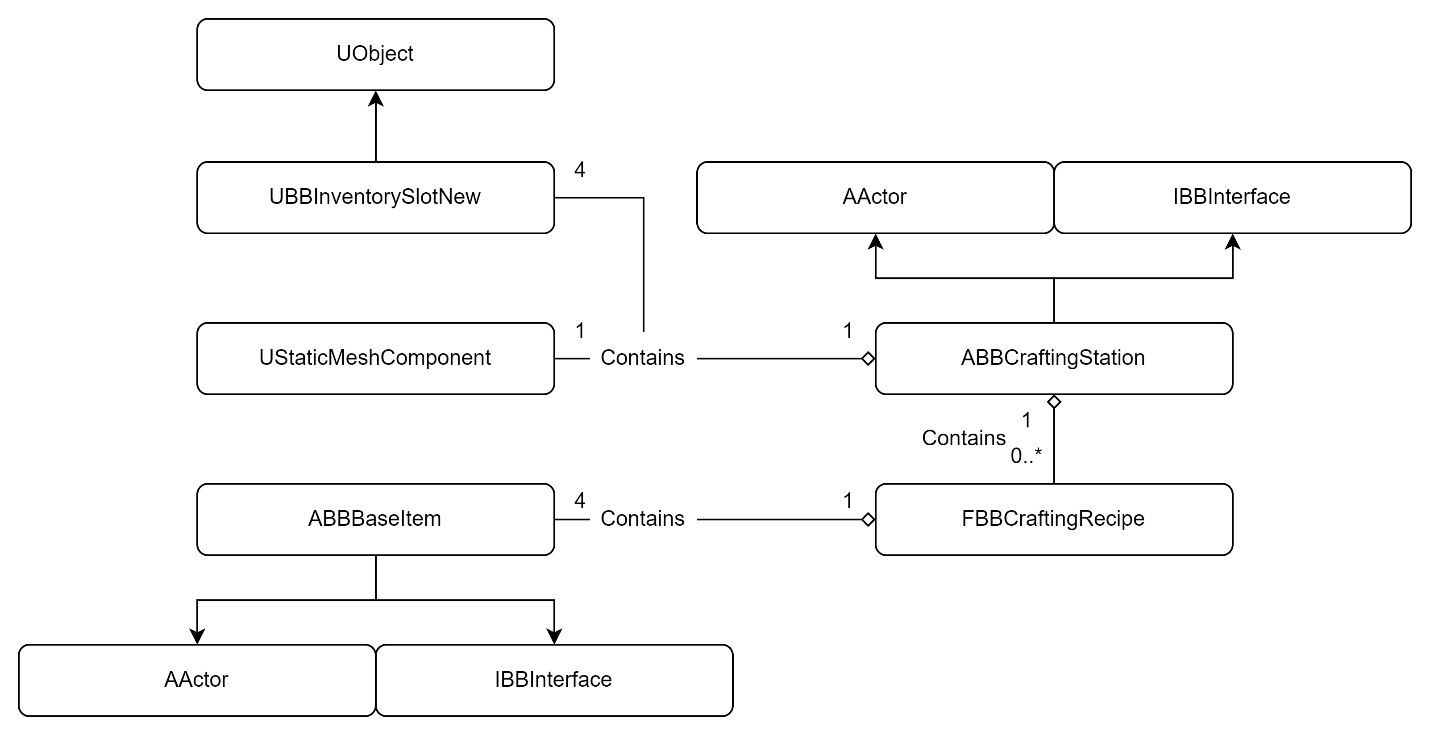
# High Level Design

Graphical user interface

Description automatically generated with medium confidence

* Player system - the system is our core system and is what the player interacts with to use all our other systems.
* Locomotion system - an advanced movement system we’re using to give us more control over the players movement. This gives us variables we can control, such as stamina, and additional functionality, such as climbing and vaulting.
* Potion system – a system meant to be easy to use for designers. The system allows for the designers as much freedom as possible, while giving the coders a minimal amount of updating to do. This is the main system the player will use to complete puzzles and generate income.
* Inventory system – a system meant to be easy to use for designers. The system allows for the designers as much freedom as possible, while giving the coders a minimal amount of updating to do. The system works as a container to hold items that the player collects. This includes the players backpack, as well as storage containers around the world.
* Resource system – a system meant to be the first of the steps for the player to generate income. This system gives the player ways to harvest materials and shows off behavioural logic for when the system is used.
* Crafting System – the system in between the Resource system, and the Shop/Potion systems, in terms of actual gameplay. This systems job is to refine the resources the player gathers into other items or potions for use in one of the other systems.
* Shop system (Not yet implemented) – a system to generate income, and allow for the player to improve. The system interfaces with AI, the player, and Storage objects, to give the player a way of selling items to NPC’s. This can have numerous effects on the AI and is the primary source of income, a necessary resource for improving the players arsenal, and serves to break the monotony of just grinding for resources.
* AI System - The system dealing with the NPC’s in the world, including fauna. The system interfaces with the shop system, allowing for NPC’s to be customers, as well as gives the AI their logic for movement, interaction, and anything else.

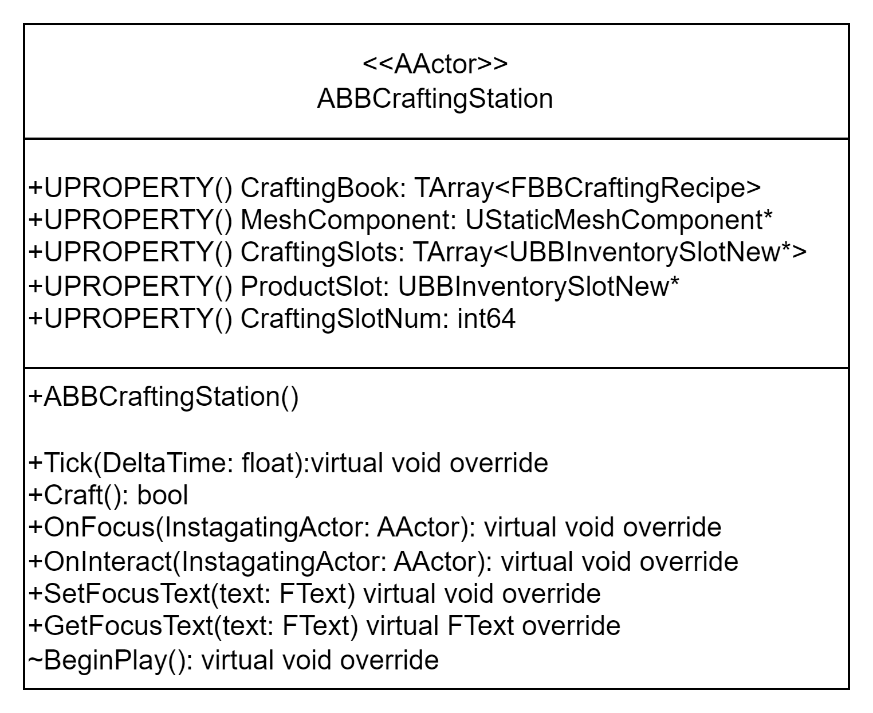
# Mid Level View

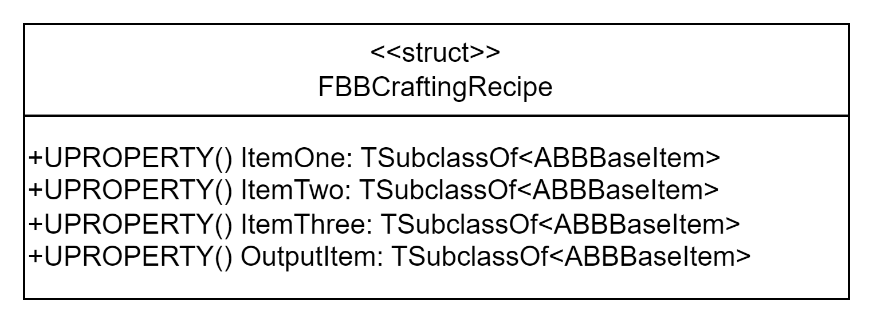


*For more detailed information about ABBBaseItem and UBBInventorySlotNew, please see the Inventory System MDD.*

* ABBCraftingStation is the interactable actor that the player can use to craft with. It holds the logic to check the internal InventorySlots and compare them against its internal list of CraftingRecipies, return to the player an appropriate action.
* FBBCraftingRecipe is the struct holding each individual recipe. It utilises TSubclassOf so the designers have basically drag-and-drop ease to set up recipes.

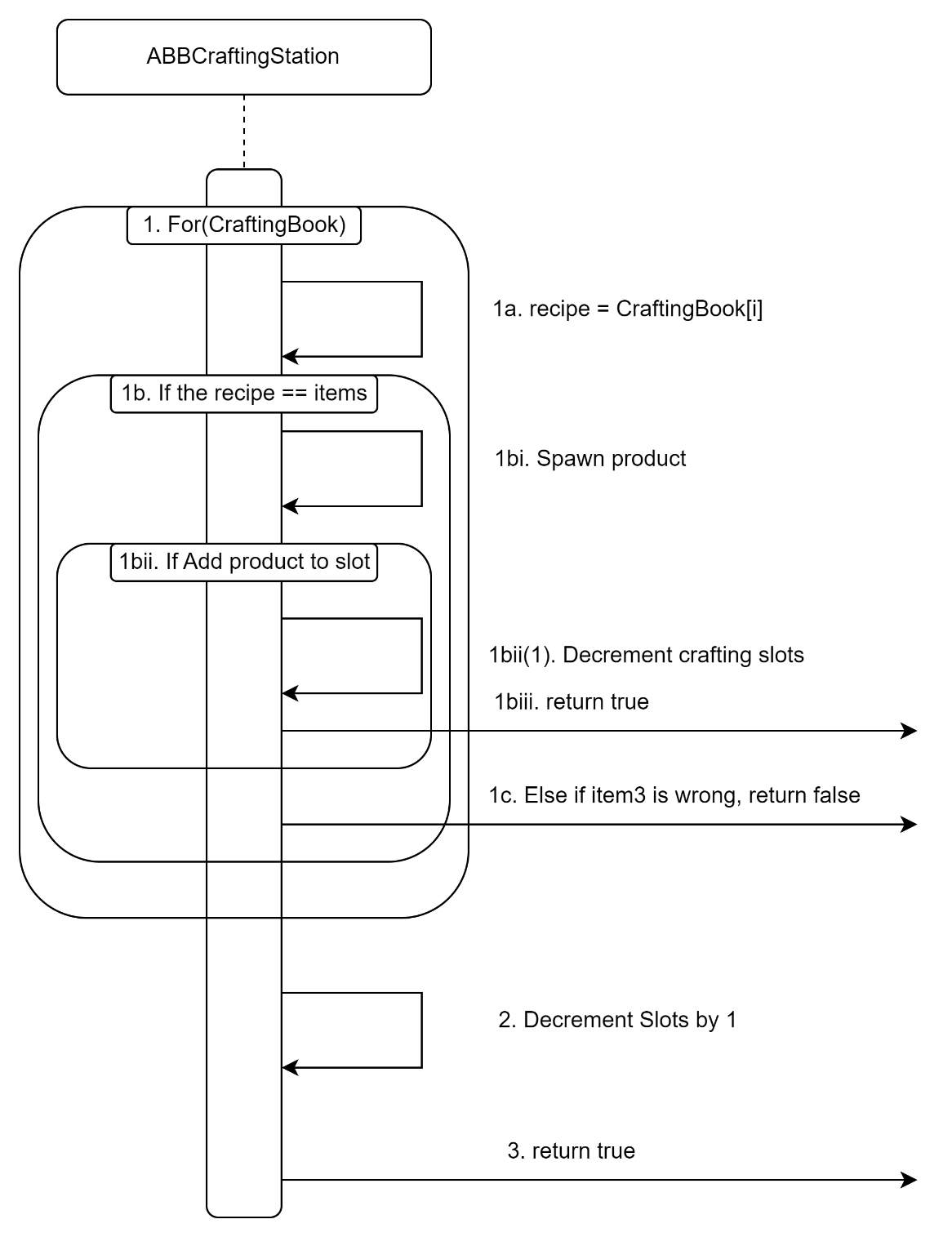
# Logical View





# Process View

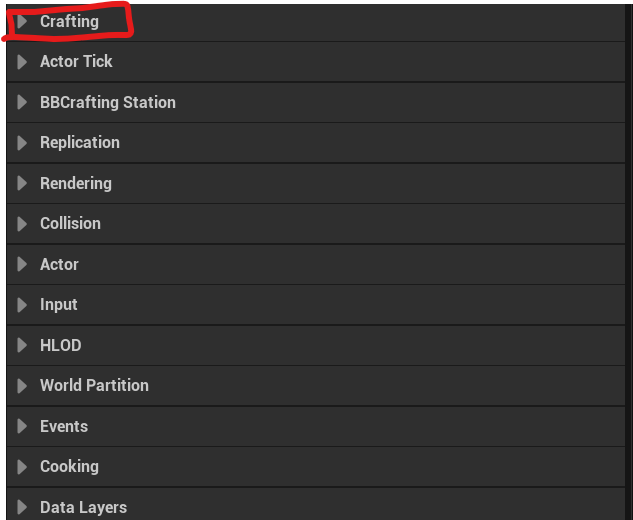
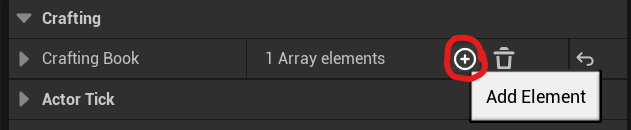
## Calling the Craft Function



1. Loop through the Recipes in the craftingbook.
   1. Copy recipe[i] to a local var recipe.
   2. if the correct items are in the correct order for the recipe;
      1. Spawn the item in the recipes output slot.
      2. Attempt to add the item to the ProductSlot.
         1. If success, decrement all the crafting slots inventories by 1.
      3. Return true.
   3. Otherwise, if the last slot (Bottle) is wrong, return false, not consuming items.
2. If the whole loop fails, decrement the crafting slots by one.
3. Return true.

# Use Case View

## Adding a recipe to a CraftingStation

1. Access the Crafting properties of the stations BP.  
   
2. Add an element to the array.  
   
3. Expand the new element and customise your new recipe.  
   