INVENTORY SYSTEM

Medieval Tax Evasion Simulator | Technical Design Reference | 2026

Overview

The inventory system in Tax Evasion Simulator is a **slot-based, array-driven architecture** that handles everything from item pickup and storage to gold stacking, item spawning back into the world, and pet taming validation. It is built as a single Actor Component called **AC_Inventory** attached to **BP_FirstPersonCharacter**. All inventory state, item identity, and holding context live in this one component. No other system needs to reach outside it to read or modify inventory data.

The key architectural decision is that every slot in the **InventorySlots** array always exists, whether it is occupied or not. An empty slot is not a missing entry. It is a struct with **bIsOccupied = false**. This means all loop logic operates on a fixed-size array with no index gaps, which makes slot finding, validation, and UI refresh straightforward and predictable.

Data Structures

ST_InventorySlot (Struct)

Every element in the InventorySlots array is an ST_InventorySlot struct. This struct carries all data needed to identify, display, and respawn an item from the inventory without any additional lookups.

Field	Type	Purpose
`bIsOccupied`	Boolean	The primary occupancy flag. Every slot exists in the array at all times. This flag determines whether the slot is active or empty.
`ItemClass`	Actor Class	The class reference used by SpawnItemFromSlot to call SpawnActor. Without this, the system cannot recreate the item in the world.
`ItemName`	Text	Display name shown in WB_Inventory and WB_ItemSlot widgets.
`ItemIcon`	Texture2D	Icon reference passed to WB_ItemSlot via UpdateSlot. Rendered using Set Brush from Texture on Image_0.
`ItemType`	E_ItemType	Enum value (None, GoldPouch, Potion, Fish, Bone). Used by GetTotalGold to identify gold slots and by pet taming logic to validate taming items.

Field	Type	Purpose
`GoldAmount`	Integer	Only meaningful when ItemType is GoldPouch. Tracks the stacked gold value in this slot. Capped at MaxGoldPerSlot by AddGoldToInventory.
`MaxStack`	Integer	Reserved stack limit field. Currently enforced only on GoldPouch items via MaxGoldPerSlot. Designed for future item stacking expansion.

E_ItemType (Enumeration)

ItemType is the primary switch value used across the inventory, pet, and interaction systems. It determines how a slot is processed by gold functions, how the UI renders it, and whether pet taming conditions are met.

E_ItemType Value	Usage in System
`None`	Default value on empty or uninitialized slots. Acts as a null state for ItemType comparisons.
`GoldPouch`	Identifies gold-stacking slots. Targeted by GetTotalGold and AddGoldToInventory. SpawnItemFromSlot branches on this type to cast and set the Gold value on BP_GoldPouch.
`Potion`	Consumable item type. Reserved for future use in healing or buff systems.
`Fish`	Required taming item for BP_Pet_Cat. AC_PetSystem checks HeldItemType == Fish before allowing cat taming via AttemptTamePet.
`Bone`	Required taming item for BP_Pet_Dog. Same validation pattern as Fish. Both are checked using the CONTAINS node against the TamingItems array in AC_PetSystem.

AC_Inventory Component Variables

The following variables are defined directly on AC_Inventory. They represent the full live state of the player's inventory at any given moment. Most are read and written exclusively through the component's own functions rather than accessed directly from outside.

Variable Name	Type	Default	Purpose
`InventorySlots`	Array of ST_InventorySlot	Empty	The main storage array. Every slot is an ST_InventorySlot struct

Variable Name	Type	Default	Purpose
			regardless of whether it is occupied.
`MaxSlots`	Integer	5	Maximum number of slots available. Drives loop ranges throughout the component.
`CurrentSlotIndex`	Integer	0	Currently selected slot. Updated by CycleInventorySlot and reflected in WB_Inventory.
`HeldItemActor`	Actor	None	Reference to the actor currently held in the world. Null when no item is held.
`bIsHoldingItem`	Boolean	false	Guard flag checked before any held-item operation to prevent acting on a null actor.
`bHeldItemFromInventory`	Boolean	false	Tracks whether the currently held item came from an inventory slot or was picked up directly from the world. Determines whether RemoveItemFromSlot must be called on drop.
`HeldItemOriginSlot`	Integer	-1	The slot index the held item came from. Set to -1 when item originated from the world, not inventory.
`MaxGoldPerSlot`	Integer	50	Stack limit for gold pouches. AddGoldToInventory fills existing pouches to this cap before creating new slots.
`HeldItemType`	E_ItemType	None	Enum value of the currently held item. Used by pet systems and interaction logic to validate what the player is carrying.
`OwnerCharacter`	BP_FirstPersonCharacter	None	Cached reference to the owning player character. Used to calculate spawn locations for SpawnItemFromSlot.

Core Functions

All inventory operations go through one of the functions listed below. External systems such as **AC_PetSystem**, **BP_TaxEvasionGameMode**, and the interaction system in **BP_FirstPersonCharacter** call these functions rather than reading or writing to the array directly. This keeps the inventory's internal state consistent and makes debugging straightforward: if something is wrong with an item, the fault will always be inside one of these functions.

Function	Inputs / Outputs	Logic Summary
`GetFirstEmptySlotIndex`	Out: Integer (index or -1)	For Each Loop with Break over InventorySlots. Breaks ST_InventorySlot, checks bIsOccupied == false. Returns ArrayIndex on first empty slot. Returns -1 via Sequence if none found.
`IsInventoryFull`	Out: Boolean	Calls GetFirstEmptySlotIndex. Compares result to -1 using Equal node. Returns true if equal (no empty slots), false otherwise.
`GetSlotData`	In: SlotIndex (Int)Out: ST_InventorySlot, IsValid (Bool)	Validates index is >= 0 AND < MaxSlots via AND node. On valid branch, gets array element and returns IsValid = true. On invalid branch, returns empty struct and IsValid = false.
`AddItemToInventory`	In: ItemClass, GoldAmount, ItemName, ItemIconOut: Success (Bool), OutSlotIndex (Int)	Calls GetFirstEmptySlotIndex. Validates result >= 0. Constructs new ST_InventorySlot with bIsOccupied = true and provided data. Uses Set Array Elem at found index. Returns Success = true and index on success, false and -1 on failure.
`RemoveItemFromSlot`	In: SlotIndexToRemove (Int)Out: Success (Bool)	Validates index range. Constructs empty ST_InventorySlot with bIsOccupied = false. Uses Set Array Elem to overwrite at index. Returns Success = true. Returns false on invalid index.
`CycleInventorySlot`	In: Direction (Int, +1 or -1)Out: NewSlotIndex (Int)	Calculates new index: (CurrentSlotIndex + Direction + MaxSlots) % MaxSlots. This formula prevents negative modulo when Direction = -1 at index 0. Sets CurrentSlotIndex, calls

Function	Inputs / Outputs	Logic Summary
		UpdateSlotSelector on WB_Inventory, returns new index.
`SpawnItemFromSlot`	In: SlotToSpawnFrom (Int)Out: SpawnedActor, Success (Bool)	Calls GetSlotData. Validates IsValid AND bIsOccupied via AND node. Gets ItemClass. Calculates spawn transform: OwnerCharacter location + (ForwardVector * 100.0). Calls SpawnActor. For GoldPouch items, casts spawned actor to BP_GoldPouch and sets Gold value from slot's GoldAmount. Sets HeldItemActor, bIsHoldingItem = true, bHeldItemFromInventory = true, HeldItemOriginSlot.
`StoreCurrentHeldItem`	In: NoneOut: Success (Bool)	Validates: bIsHoldingItem == true AND NOT bHeldItemFromInventory AND HeldItemActor IsValid. Casts to BP_GoldPouch. If gold pouch, calls AddGoldToInventory with Gold value, destroys actor, clears state. If not gold pouch, gets class, calls AddItemToInventory, destroys actor on success, clears state.
`DropHeldItem`	In: NoneOut: None	Validates HeldItemActor via IsValid. Checks bHeldItemFromInventory. If true, calls RemoveItemFromSlot with HeldItemOriginSlot. Clears HeldItemActor to null, sets bIsHoldingItem = false, bHeldItemFromInventory = false, HeldItemOriginSlot = -1.
`ThrowHeldItem`	In: NoneOut: None	Validates HeldItemActor. Gets PrimitiveComponent via GetComponentByClass. Calculates impulse: OwnerCharacter ForwardVector * ThrowForce. Calls AddImpulse with VelChange = true. If bHeldItemFromInventory, calls RemoveItemFromSlot. Clears all holding state variables.
`DestroyHeldItem`	In: NoneOut: None	Validates HeldItemActor via IsValid. Calls DestroyActor. Clears HeldItemActor to null. Sets bIsHoldingItem = false,

Function	Inputs / Outputs	Logic Summary
		bHeldItemFromInventory = false, HeldItemOriginSlot = -1.

Gold Management Functions

Gold is handled differently from all other item types because it stacks. A gold pouch in the world is a physical **BP_GoldPouch** actor with a Gold float variable. When stored in inventory, that gold is represented as an **ST_InventorySlot** with ItemType = GoldPouch and a GoldAmount integer. Multiple pouches can stack into the same slot up to MaxGoldPerSlot (50). When a slot is full, the next pouch creates a new slot. Any gold that exceeds the available slot space is returned as OverflowGold.

Function	Inputs / Outputs	Logic Summary
`GetTotalGold`	Out: TotalGold (Integer)	For Each Loop with Break over InventorySlots. Breaks each slot, checks ItemType == GoldPouch. If true, adds GoldAmount to RunningTotal integer. Returns accumulated TotalGold after loop completes.
`AddGoldToInventory`	In: GoldToAdd (Integer)Out: OverflowGold (Integer)	Two-pass loop. Pass 1: fills existing GoldPouch slots. Gets slot, checks ItemType == GoldPouch, calculates available space (MaxGoldPerSlot minus GoldAmount), uses MIN node to determine fill amount, constructs updated ST_InventorySlot and sets array element. Subtracts filled amount from RemainingGold. Pass 2: if RemainingGold > 0, uses For Loop to find unoccupied slots, creates new GoldPouch slot with MIN(RemainingGold, MaxGoldPerSlot). Returns any remaining gold as OverflowGold.

Gold Flow: From World to Inventory

The full path a gold pouch takes from the world into the inventory is:

- Player picks up BP_GoldPouch from world via the line trace and grab system in BP_FPC.
- Player presses Store Item (I key), triggering StoreCurrentHeldItem on AC_Inventory.

- StoreCurrentHeldItem validates the held actor, casts to BP_GoldPouch, reads the Gold float value.
- Calls AddGoldToInventory with the gold amount. Two-pass loop fills existing pouch slots first, then creates new ones.
- BP_GoldPouch actor is destroyed. Holding state variables are cleared. WB_Inventory reflects the updated slot on the next 1-second timer tick.

Gold Flow: From Inventory to World

When the player needs to physically carry gold, such as to a hiding spot, the reverse path is:

- Player scrolls to a GoldPouch slot and presses Grab (right mouse button), triggering SpawnItemFromSlot.
- SpawnItemFromSlot reads the slot's ItemClass and calculates spawn transform 100 units in front of the player.
- Calls SpawnActor. Detects ItemType == GoldPouch via branch, casts spawned actor to BP_GoldPouch, sets its Gold variable from slot's GoldAmount.
- Sets HeldItemActor, bIsHoldingItem = true, bHeldItemFromInventory = true, HeldItemOriginSlot to the source index.
- The original slot is NOT cleared yet. It is only cleared via RemoveItemFromSlot when DropHeldItem, ThrowHeldItem, or a successful StoreCurrentHeldItem call confirms the item left the player's possession.

UI Layer: WB_Inventory and WB_ItemSlot

The inventory UI is intentionally thin. It does not own any data. It reads from AC_Inventory on a timer and reflects what it finds. This separation means the UI can never cause a desync with the actual inventory state.

Widget	Key Event or Function	What It Does
`WB_Inventory`	**Event Construct**	Gets PlayerCharacter, gets AC_Inventory by class, caches MyInventoryRef. Creates a looping timer calling RefreshInventoryUI every 1.0 second.
`WB_Inventory`	**RefreshInventoryUI**	For Loop 0 to 4. Uses Select node with Item1 to Item5 texture references to get icon by index. Gets slot from MyInventoryRef.InventorySlots. Breaks ST_InventorySlot. If bIsOccupied, sets visibility Visible and calls UpdateSlot on WB_ItemSlot with ItemIcon. Else sets visibility Hidden.

Widget	Key Event or Function	What It Does
`WB_Inventory`	**UpdateSlotSelector**	Takes CurrentSlot integer. Makes Array of Slot1 to Slot5 widget references. For Each Loop sets all to Hidden. Gets element at CurrentSlot index, sets that one to Visible.
`WB_ItemSlot`	**UpdateSlot**	Takes NewIcon (Texture2D) as input. Calls Set Brush from Texture on Image_0 component. This is the only function on WB_ItemSlot. Kept minimal by design.

Important: WB_Inventory sets up a 1.0 second repeating timer in Event Construct to call RefreshInventoryUI. This timer must be explicitly cleared and invalidated in Event Destruct to prevent Blueprint runtime errors on game shutdown. This is a known technical debt item flagged in the project's issues list.

Known Issues and Technical Notes

Negative Modulo in CycleInventorySlot

In languages and engines where modulo on negative numbers returns a negative result, the naive formula **(CurrentSlotIndex + Direction) % MaxSlots** would produce -1 when Direction = -1 and CurrentSlotIndex = 0. The implemented formula adds MaxSlots before taking the modulo:

```
(CurrentSlotIndex + Direction + MaxSlots) % MaxSlots
```

This ensures the result is always a valid non-negative index regardless of scroll direction. This is an explicit safety measure, not a workaround.

Timer Cleanup

Both WB_Inventory (the 1-second RefreshInventoryUI timer) and the line trace timer in BP_FPC require explicit cleanup in their respective Event Destruct nodes. Without this, the timers continue firing after the widget or character is destroyed, which can cause Blueprint runtime access errors on level transitions or game shutdown. Adding Clear Timer by Handle and Invalidate calls in Event Destruct for both is the required fix.

Slot Count Fixed at 5

MaxSlots is currently set to 5 and the UI is built with five explicit slot widget references (Item1 to Item5). Increasing MaxSlots without adding corresponding widget entries to WB_Inventory and updating the Select node in RefreshInventoryUI will cause out-of-bounds access. Any future inventory expansion must update both the variable and the widget hierarchy together.