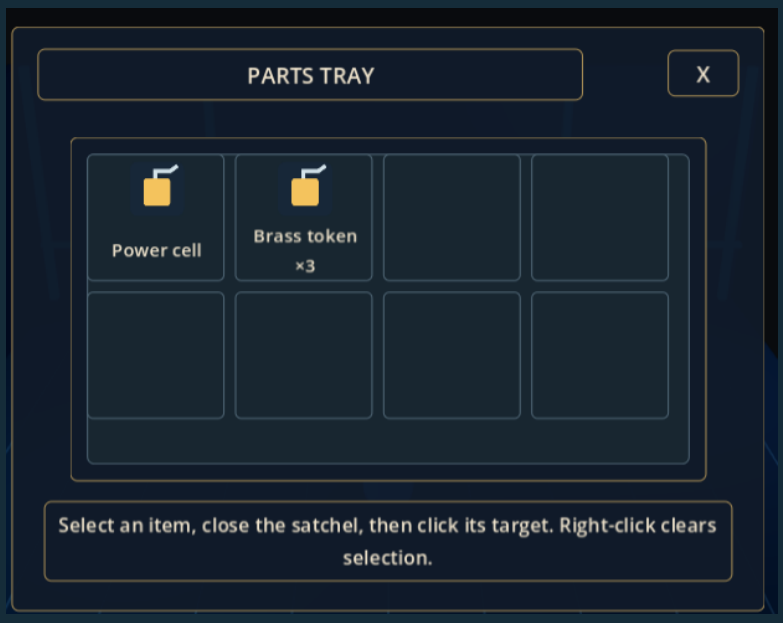
PCA 3D ADVENTURE STUDIO

Inventory Designer

Create items your characters can find, carry, examine and use together.



Follow a power cell from a workbench into a parts tray, then into the relay it was made to power.

Illustrated with PCA3D 0.86.3.4 on Linux

CONTENTS

Find a topic

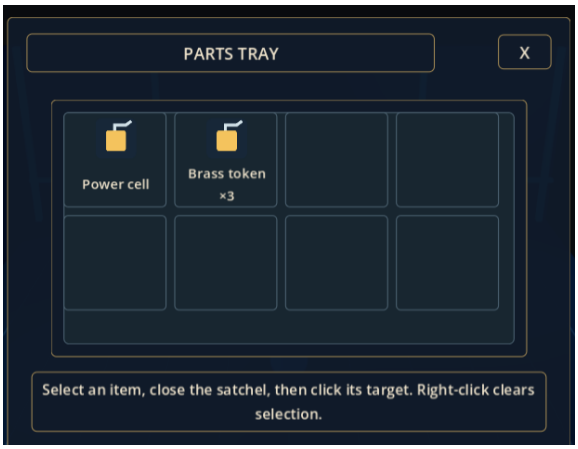
Start with the power cell if you’re new to inventory. If you already have an item, choose the question closest to what you want it to do.

One power cell, three places to look	3
Create or edit an item record	4
Set quantities and first-found behavior	5
Choose an icon for each state	6
Give the player an item they can pick up	7
Give an item something to say or do	8
Use a carried item on a world target	9
Combine two carried items	10
Decide who carries what	11
Use types, categories and custom values	12
Show the inventory in your own GUI	13
Size the grid and reach the next item	14
Make selection and dragging feel right	15
Check the bag after saving and loading	16
Find the cause of an inventory problem	17

HOW INVENTORY WORKS

One power cell, three places to look

On the workbench, the power cell is a clickable object. In the bag, it is a quantity of a named inventory item. The bag itself is a GUI that shows what the character carries.



The running game shows one Power cell and three Brass tokens in its Parts Tray.

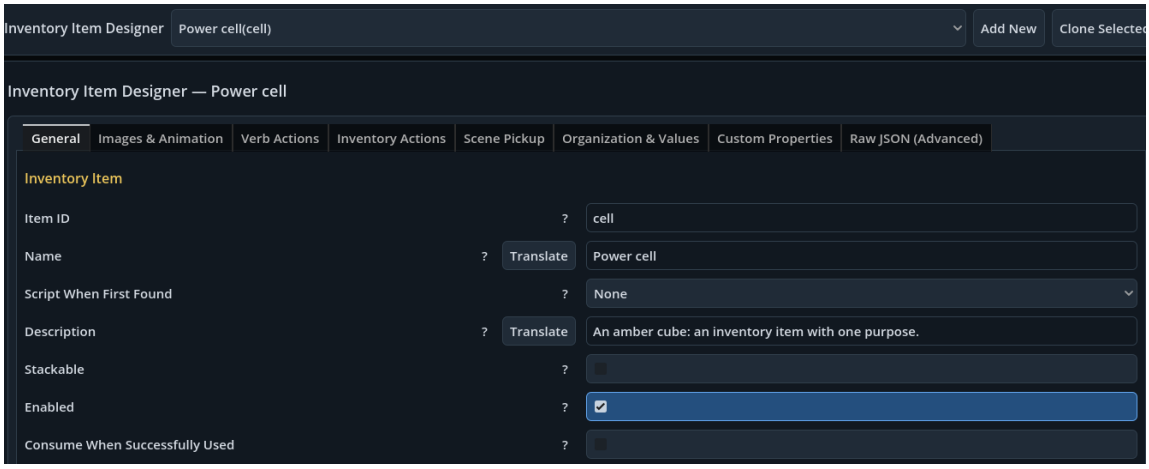
Inventory Designer	Defines the carried item. The example uses the ID cell, the name Power cell, an icon and a rule for using it on the relay.
Scene Designer	Defines the physical object the player sees. The pickup object has the separate ID power_cell. Its Take action gives the player the cell item.
GUI Designer	Defines the Parts Tray and its inventory grid. When a character gains an item, the running grid reads that character’s inventory.

> If the cube is visible on the table but the bag is empty, check the Take action. If the game says the character owns it but the slot is empty, check the GUI owner, type filter and display settings.

INVENTORY ITEM DESIGNER

Create or edit an item record

Begin with the name the player should see and a short ID you can recognize when selecting the item from an action.



The Power cell record is selected. General holds the ID, display name, description and basic behavior.

01 Open your game and select **Inventory** in the workspace strip, or select an item under Inventory in the Adventure Database. The workspace heading reads **Inventory Item Designer**.

02 Choose **Add New** to create an item. To change one that already exists, select it in the record picker. Check the heading before typing so you know which item you are editing.

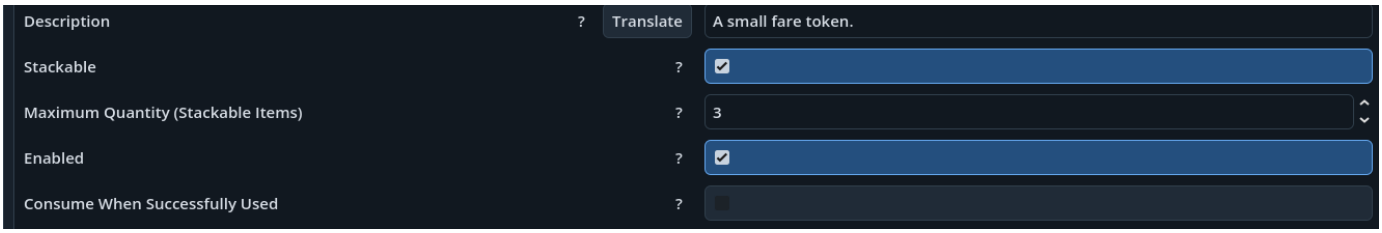
03 On **General**, set **Item ID** to a stable name such as **cell**. Set **Name** to “Power cell.” Write a description the player might actually want to read. Leave the record enabled for normal use.

04 Choose **Apply Changes**, then **Save Project**. Select this ID whenever an action asks which inventory item to give or remove. Changing a visible name later is usually easier than repairing references to a renamed ID.

GENERAL

Set quantities and first-found behavior

A unique key and a handful of coins should behave differently when the player finds a second one. Decide what one inventory slot represents before you build the pickup.



Brass token has Stackable turned on. Its maximum is three. Turning Stackable off hides the maximum field.

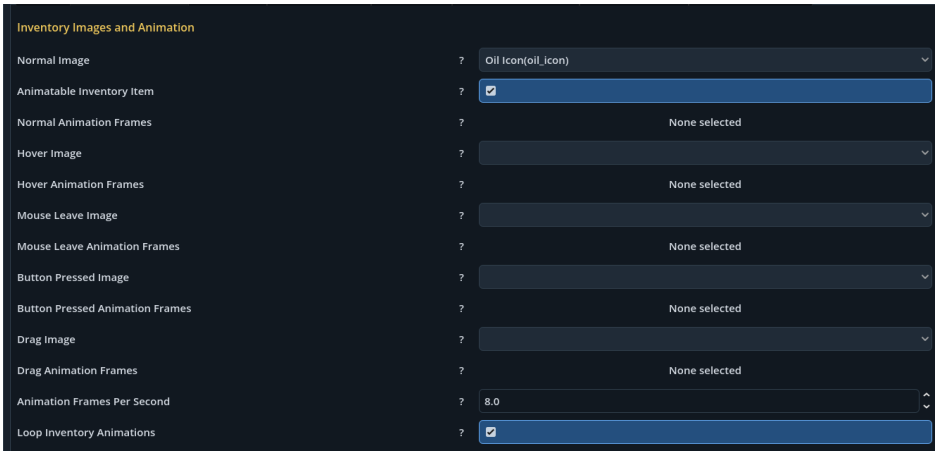
A unique item	Leave Stackable off for something like a particular key. The current runtime caps a nonstackable item at one.
Multiple copies	Enable Stackable and enter Maximum Quantity. With a cap of three, adding four tokens to an empty bag produced three. A full stack does not start a second stack of that item.
Script When First Found	Choose a script if acquiring the first copy should reveal a clue or play a line. In the active-player test, going from zero to two ran it once. Going from two to three added quantity without running it again.

> After all tokens were removed, acquiring one again ran the first-found script a second time. For a once-per-story clue, guard it with a saved game flag such as token_clue_seen. A stack limit also means a pickup can add fewer items than requested.

IMAGES & ANIMATION

Choose an icon for each state

Start with an image that reads clearly at the size of a real inventory slot. A sharp silhouette matters more here than detail that disappears at small sizes.



Enabling Animatable Inventory Item reveals the frame lists, playback speed and loop switch alongside the image pickers.

Normal Image	The resting image in the bag. Import the picture as an image asset, then select it here. Test it against the slot background in the running game.
Hover / Mouse Leave	A bright outline on hover can show which item the pointer is over. A leave image can ease it back to rest. Touch players still need to recognize the normal image.
Button Pressed / Drag	A pressed image acknowledges the press. A drag image helps the player see which item they are carrying across the screen.
Animatable Inventory Item	Enable it to reveal normal and state-specific frame lists, frames per second and looping. Ten frames at 10 FPS take about one second. Check Animate Item Icons on the GUI grid as well.

A static item needs only a normal image for an ordinary slot. If your supplied frames appear in the wrong sequence, check the order of the chosen frames and test the actual hover or drag gesture.

ROOM ACTIONS AND SCENE PICKUP

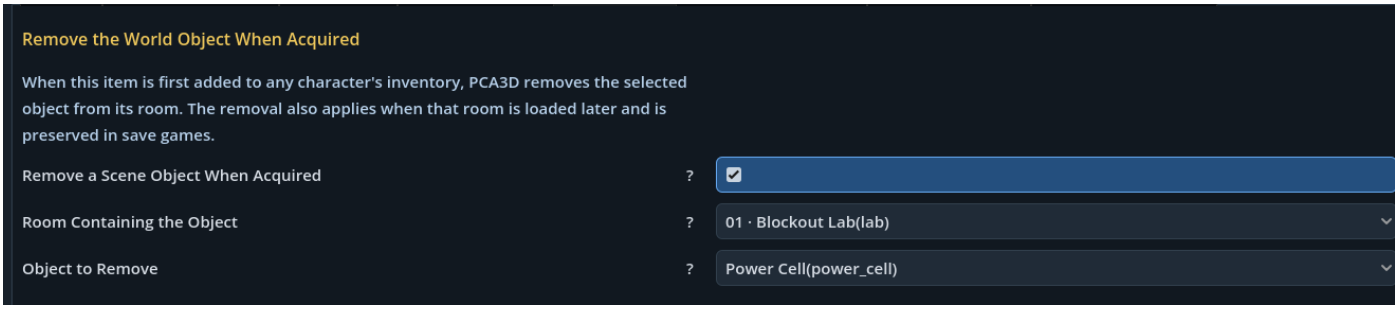
Give the player an item they can pick up

A player clicks the object in the room. That object’s response gives the inventory item. Scene Pickup tells PCA3D which room object should disappear after acquisition.

01 In Scene Designer, select the clickable world object. Confirm its hotspot and intended verb. In that object’s **Verb Actions**, choose the Take verb used by your project, or the interaction that should perform the pickup.

02 Add the **Add Inventory** action. Select the inventory item **cell** and a count of 1. If you also give points or play a one-time line, keep those effects on the successful pickup path.

03 In Inventory Designer, open **cell → Scene Pickup**. Turn on **Remove a Scene Object When Acquired**. Set the room first, then choose the object inside that room. Apply and save.



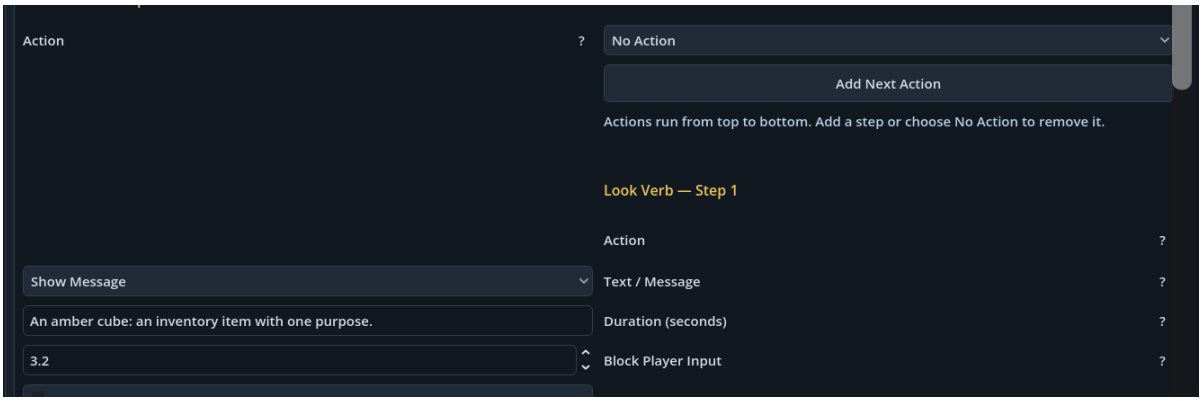
Scene Pickup points to the room 01 · Blockout Lab and the world object Power Cell [power_cell].

> In the running preview, Take changed the carried count from zero to one and removed power_cell from the room. Loading a save made after the pickup restored the carried item and kept the world object absent.

VERB ACTIONS

Give an item something to say or do

“Look at the cell” needs only one item. “Use the cell on the relay” needs a second target. Those two sentences lead you to different tabs.



The Look verb has a Show Message action. The message is attached to examining the carried cell.

01 Open the inventory item and choose **Verb Actions**. Locate the verb your game actually uses, such as Look. Available verb rows follow the project’s verbs.

02 Select a result action and fill in its settings. For Look, choose **Show Message** and enter “An amber cube: an inventory item with one purpose.” Give the player time to read it.

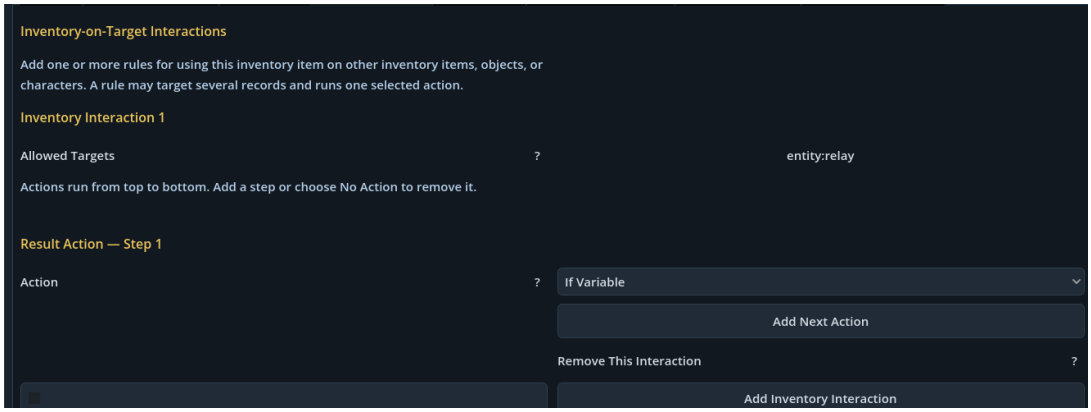
03 For a longer response, use **Add Next Action** and put the steps in the order the player should experience them. Apply and test the verb on the item in the bag.

If no Look response is authored, the runtime can display the item description. Use a dedicated action when examining the item should change a flag, give a different line later, or start another piece of gameplay.

INVENTORY ACTIONS

Use a carried item on a world target

Powering the relay requires both ends of an interaction: the carried power cell and the blue relay already placed in the room.



The item-side rule lists the allowed target and its result action. In this example, If Variable chooses the response based on powered.

01 Open **Inventory Actions** on the item. Choose **Add Inventory Interaction**. Under **Allowed Targets**, select the placed relay. One rule can accept several targets that genuinely share the same result.

02 Choose the **Result Action**. For a reusable puzzle, check a shared flag first. On the first success, set **powered** to true, remove one cell and show the relay’s response. A later attempt can explain that the relay is already powered.

03 Alternatively, author the rule on the placed target under **Inventory Actions → Allowed Inventory Items**. Pick the cell there. In this build a matching item-side rule runs before the target-side rule. Keep the main effect in one place.

> Observed result: using the carried cell on the relay set powered to true, removed the cell, and left the three unrelated tokens in the bag.

INVENTORY ACTIONS

Combine two carried items

A battery and a torch can form one working torch. Decide four things up front: which two items match, which result appears, which ingredients remain and what a second attempt should do.

Pick the pair	On the battery record, add an inventory interaction. Under Allowed Targets, choose the torch inventory item. An inventory target and a similarly named room object are separate choices.
Write the result	For a recipe, check the inputs before changing the bag. Remove the quantities your design consumes, then add the working torch. Check capacity beforehand if the result already exists or its stack is full.
Try both directions	The runtime checks for a matching rule on the dragged item first, then on the other item. In the test, one token-to-cell rule matched either dragging order. Two different rules on the same pair can make order matter.
Check the widget	In GUI Designer, turn on Drag Items Together for the grid. If a drag only moves the item to another slot, check whether the pair has a matching rule and whether Player Drag Sorting is enabled.

> A successful rule does not guarantee a quantity change. In 0.86.3.4, a token rule that only set a flag left all three tokens in place even with Consume When Successfully Used checked. Use an explicit Remove Inventory action when consumption matters, and retest the count in your edition.

OWNERSHIP AND STARTING ITEMS

Decide who carries what

A shared bag suits companions who travel together. Separate inventories matter when handing a tool to the right character is part of the puzzle.

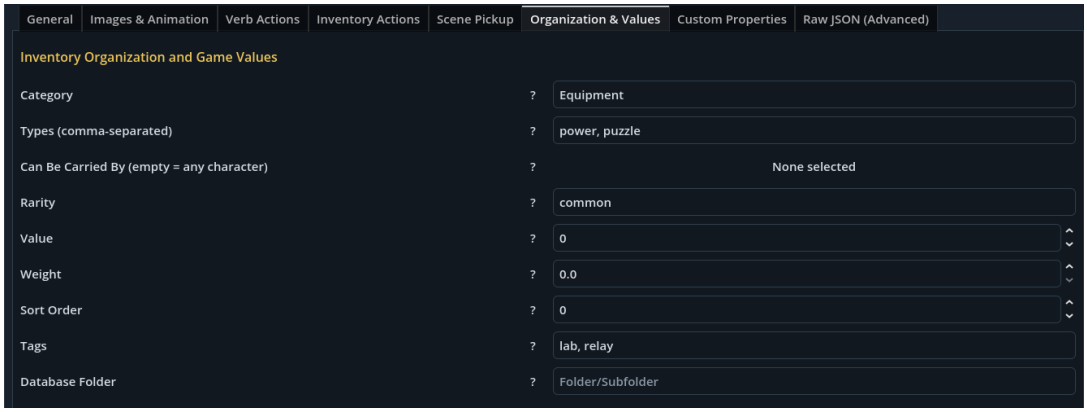
Project Inventory Mode	Choose shared or per-character in the project settings. Shared mode lets the active cast use a common inventory. Per-character mode keeps distinct supplies for playable characters.
Character starting items	In the character’s Player Settings, find Starting Inventory → Initial Inventory Item IDs. Enter existing item IDs separated by commas. The starting active character can also receive the project’s initial items.
Can Be Carried By	On the inventory item, open Organization & Values. An empty carrier list permits any character. Select specific characters only when you want that restriction.
Transfers	Use Transfer Inventory when handing an item from one character to another. Set the giving character, receiving character, item and count. Confirm the recipient is allowed to carry it and check both balances after the transfer.

If an item vanishes when switching characters, switch back once. Seeing it again is a strong clue that ownership or the GUI’s selected owner explains the result. A grid can be pinned to one character even when another is active.

ORGANIZATION & VALUES

Use types, categories and custom values

“Power cell” is the item’s name. “Equipment” can be its category. “Power” and “puzzle” can both be types. Keep those jobs distinct when designing filtered bags.



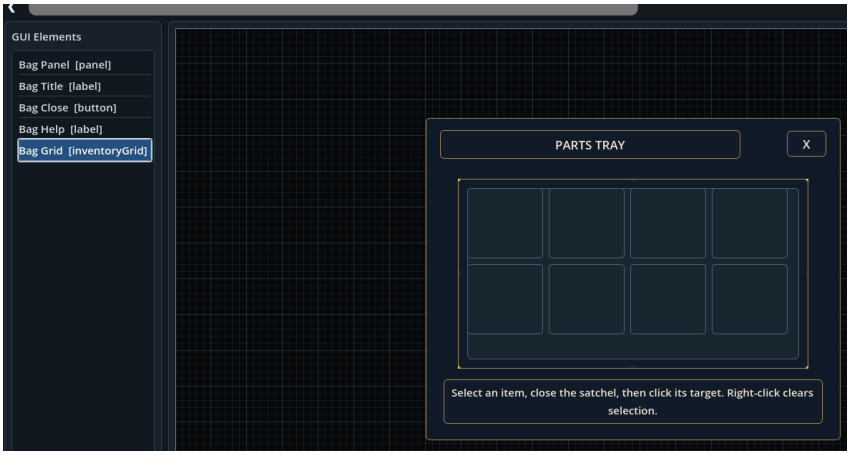
The cell is categorized as Equipment and has the types power and puzzle. The same tab holds carrier restrictions, rarity, value, weight, sorting and tags.

Types	Enter comma-separated types. A grid filtering on puzzle can display this item because at least one type matches. A grid filtering on fare would hide it.
Category, tags and folder	Use a category and tags that help you organize or query items. Database Folder groups records for the author. A folder name does not supply a GUI type filter.
Rarity, value and weight	Use these as authored data for your game’s rules. If a shop must charge the item value or a bag must reject excess weight, add and test that rule explicitly.
Custom Properties	Use Add Custom Property for information specific to your game, such as voltage. Decide whether a value describes the item definition or a particular player’s progress before storing it.

GUI DESIGNER

Show the inventory in your own GUI

The grid is a window onto an inventory. If the designer canvas looks empty, run the game and give the displayed character an item before deciding the grid is broken.



The Parts Tray layout in GUI Designer. The selected element is bag_grid, an inventoryGrid inside the inventory GUI.

01 Open or create the GUI that should contain the bag. Select an existing **Inventory Grid** or add one. Choose its size, rows and columns so a player can read the item names and comfortably select a slot.

02 In the grid’s properties, set **Inventory Owner** to the current player or a specific character. Leave **Inventory Types** empty for all items. To build a keyring, enter a type such as key and give each eligible item that type.

03 Choose **Show Item Text**, **Show Item Icon** and **Animate Item Icons** as appropriate. Give the player a way to open and close the GUI. Use page or arrow navigation if the bag can outgrow the visible slots.

A carousel offers a different presentation for a small selection of items. Keep ownership and type filters consistent with the grid if both controls represent the same bag.

GUI DESIGNER · TYPE SETTINGS

Size the grid and reach the next item

A bag can hold more items than it can show at once. In the pictured GUI, four columns and two rows give the player eight visible places.



Type Settings for Bag Grid. Visible Columns is 4, Visible Rows is 2 and Visible Slots is 8. Slot width is a separate setting.

01 Select the inventoryGrid element, then **Element Properties → Type Settings**. In Paged Widget & Grid set the visible columns, rows and slot count to match the layout you want. Four by two displays eight positions. It does not limit the character to eight kinds of item.

02 Adjust slot width and height, the gaps between slots and the padding around the widget. Leave room for names, quantities and arrows. Check the GUI at the game’s own resolution as well as on a small screen.

03 Choose a scrolling and arrow arrangement that fits your GUI. To use separate GUI buttons for navigation, connect them to the inventory navigation actions and check that they move the intended widget.

> To diagnose a missing ninth item, check the count, owner and filter before increasing the grid size. If the first eight positions are filled and the next page shows the item, the inventory is working. The player needs a clearer way to change pages.

INTERACTION STYLES

Make selection and dragging feel right

“It won’t use the item” can mean the game selected it successfully and is waiting for the next click. Try the gesture that matches the game’s interaction mode.

Pointer and verbs	In a verb-based game, select the verb and item or target in the expected order. Keep the Use wording and any preposition clear so “Use cell on relay” reads naturally.
Verb Direct	An item can be selected first and then applied to a world target, or the world target can be selected before choosing the item. With the double-click variant, confirm with the configured double-click or double-tap.
Touch	Drag a carried item out of the inventory and release it over a valid world target. Releasing on empty ground can leave the item selected for the next target tap. Canceling the drag should leave the item unused.
Inside the grid	Drag Items Together enables combination. When the pair has no rule, Player Drag Sorting can instead insert the item before the destination slot. Disable sorting when a fixed arrangement matters.
Other characters	An inventory for a character who is not active can serve as a read-only display. Test actions with the intended character active before diagnosing a failed drag.

> Test a short tap, a deliberate drag and a canceled drag separately. On touch devices also check real slot size and the edge of the panel. Desktop hover feedback alone cannot teach a touch gesture.

TESTING A PUZZLE

Check the bag after saving and loading

For a puzzle that moves something out of the room and into a bag, the save file has to remember both halves. A correct quantity beside a second copy on the floor is still a bug.

Before pickup	Start in the intended room. Confirm the world object exists. Confirm the active character owns zero copies.
After pickup	Use the room object’s real Take interaction. Read the count and inspect the original location. In the cell example, the count became one and the world object disappeared.
After reloading	Save to a disposable slot, change the live inventory, then reload the slot. In the checked run the cell returned to the bag and the world object stayed absent.
After correct use	Use the cell on the relay. Check the puzzle flag and the item count. Trigger the interaction again and confirm the reward or quantity change cannot repeat by accident.

For an exact count in a BASIC script, **InventoryCount("cell")** reads the active character’s quantity. A second character ID can be supplied when checking someone else. A shared variable called cell is not the inventory quantity.

> Repeat the test from a fresh game when you change starting inventory. Loading an existing slot restores earlier progress and can hide the effect of a new starting item.

TROUBLESHOOTING

Find the cause of an inventory problem

Start with the first point where the result differs from what you intended. That tells you whether to reopen the item, the room object, the character or the GUI.

The item never appears	Check the actual pickup action, item ID, stack cap and allowed carrier. Query the count before changing the icon.
Count is correct but the bag looks empty	Check the displayed owner, Inventory Types filter, current grid page and visibility. Enable text temporarily to distinguish an empty slot from missing artwork.
The object returns on re-entry	Check the room and placed object under Scene Pickup. Merely hiding the visible model may leave other parts of the puzzle active.
Using the wrong item works	Inspect both item-side and target-side rules. Narrow Allowed Targets or Allowed Inventory Items, then test one accepted and one rejected pair.
The first-found line repeats	Check whether the quantity fell to zero and the item was picked up again. Save a separate one-time story flag if that line belongs only to the first discovery.
The picture never animates	Check the item’s animation switch and frames, the grid’s Animate Item Icons setting, and whether you are looking at the normal or hover state.

Screenshots and behavior checks use PCA3D 0.86.3.4 on Linux. Use tab and field names when following along on another platform. Compare interaction and touch behavior against your installed release, particularly on builds preceding the 0.75.7 inventory changes.