PCA 3D ADVENTURE STUDIO

Save Games and Inventory

Build the menus your players will use throughout the adventure

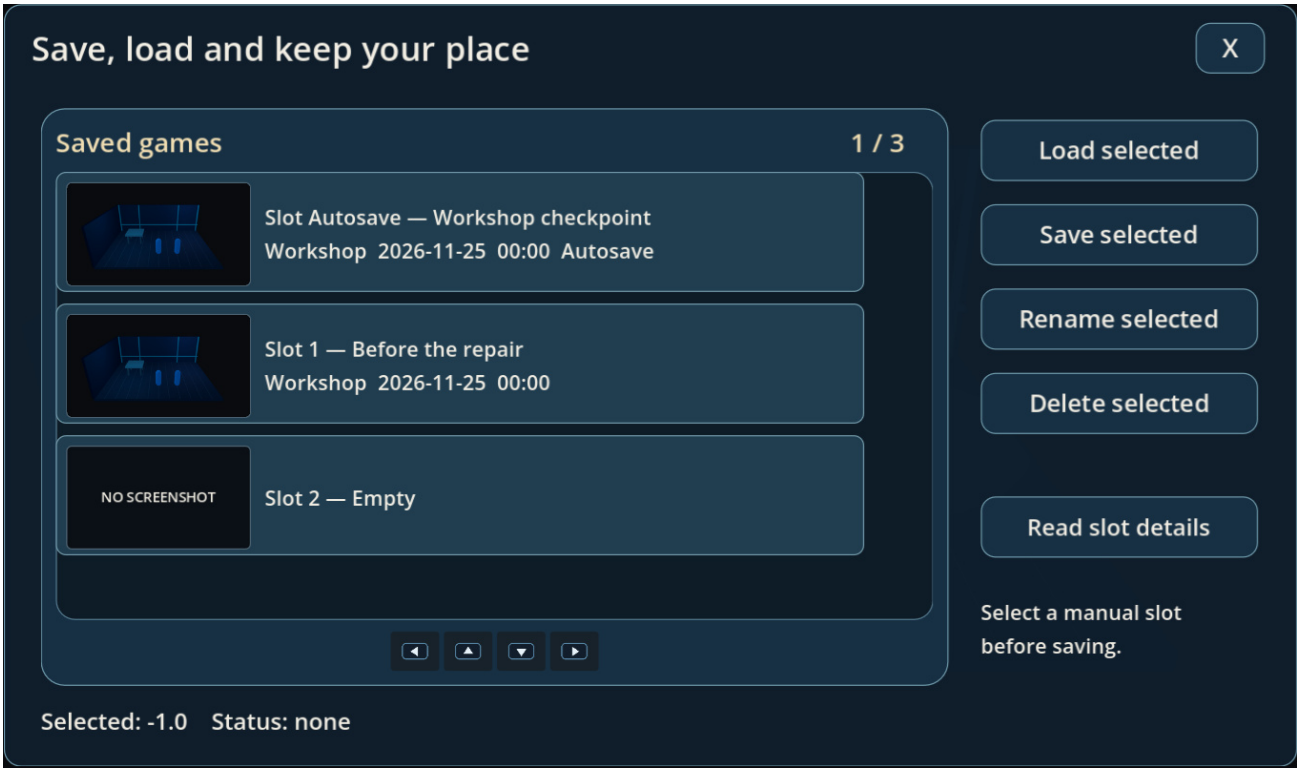

Figure 1. A working save menu with real saved games, empty slots and separate command buttons.

An in-depth GUI workshop for Windows, macOS and Linux.

Editor reference: 0.86.3.4 | Companion to the GUI Designer tutorials

Before you begin

This book is about two GUI elements: Inventory Grid and Save Game Widget. Both look like grids, but they display different information. One shows the items a character carries. The other shows the saved games stored for the current project.

We will build a workshop satchel, give its items useful interactions, and make a save-and-load window with pictures, names and separate buttons. Then we will take the widgets apart: layout, paging, ownership, images, corners, gestures, commands and the details that matter when you release a game.

All screenshots come from PCA 3D Adventure Studio itself, either in the editor or in Play mode. The running examples use the engine’s own icons and workshop scene. The instructions apply across desktop platforms; the screenshots happen to use the clean dark editor theme.

No practice project or extra practice files are included. You can follow along in a copy of your own game. The examples use readable IDs so you can replace them with yours.

First, be comfortable with	Then this book will teach you
Adding a GUI and an element	Making a complete inventory and save menu.
Names and IDs	Connecting items, characters, GUIs, elements and save slots.
A Button’s Events tab	Using action lists and visual nodes for item and save commands.
Applying a script record, if you use scripts	Small complete GDScript examples, with every important choice explained.

> The reference build is 0.86.3.4. A setting shown in the inspector is not always a promise that every widget uses it in the same way. Where a limitation matters, the instructions give a working approach and explain the boundary.

Contents

Click a chapter title, or use the PDF bookmarks to open a particular continued page.

1. Know what each piece does 4
2. Build your first inventory menu 6
3. Define the items the grid will show 8
4. Choose who owns the contents 13
5. Give the grid enough room 15
6. Page through the carried items 19
7. Select an item and use a verb 22
8. Combine two inventory items 25
9. Add, remove and transfer items 29
10. Style the independent parts 32
11. Assign images to the right layer 35
12. Build useful inventory variations 38
13. Prepare the game’s save system 40
14. Make a first save-and-load window 42
15. Choose mode, range and sort order 45
16. Arrange rows, cards or text lists 47
17. Capture and display save pictures 49
18. Write clear save captions 54
19. Assign clicks and commands deliberately 57
20. Names, overwrite and deletion 60
21. Track selection and slot events 63
22. Use the same operations in scripts 67
23. Save state, change it, then restore it 69
24. Build separate Save and Load screens 71
25. Test the menus before release 74
26. Inventory troubleshooting 76
27. Save-menu troubleshooting 79
28. Shared layout property reference 82
29. Action and script reference 87
30. Build the complete workshop flow 91

1. Know what each piece does

Piece	Its job	It does not do this for you
Inventory item definition	Names the item, chooses its icon and stores its authored behavior.	Put the item in the player’s bag.
Character inventory	Stores which item IDs a character owns and how many of each.	Choose the GUI layout.
Inventory Grid	Displays a character’s carried items and provides built-in selection and item interactions.	Create items simply because an empty cell is visible.
Save Game Widget	Lists saved-game slots and can select, save, load, rename or delete them.	Save the editor project.
A GUI window	Holds the widget, headings, buttons and other elements.	Automatically decide what every button should do.

Think of an inventory definition as an entry in your game’s catalogue. The carried inventory is the player’s actual collection. A perfectly good item definition can exist without appearing in the bag, because the player has not been given it yet.

A save slot is a numbered place on disk. The Save Game Widget is a view of those places. Changing its number of visible rows changes the menu, not the number of game saves your project allows.

> There are two separate kinds of saving. Save Project preserves your work as a game author. Save Game preserves a player’s progress while the game runs. An editor recovery draft is a third thing; it is not a player save slot.

Names, IDs and the example’s vocabulary

A name is what you want a person to read. An ID is the stable reference actions and scripts use. A dropdown may show Player Inventory(inventory_lab). The reference you type into a script is inventory_lab.

Example	Name	ID or number
Inventory GUI	Player Inventory	inventory_lab
Inventory element	Bag Grid	bag_grid
Save GUI	Save and Load	save_lab
Save element	Save Slots	save_slots
Player character	Builder	tester
Second character	Assistant	assistant
Inventory item	Brass key	brass_key
Stackable item	Bearing oil	oil
Manual save	Before the repair	Slot 1
Reserved autosave	Workshop checkpoint	Slot 0 in this example

Keep element IDs unique across your project. Several operations can address an element by ID, and reusing bag_grid for a different menu makes the intended target harder to identify.

The example inventory starts with eight different item IDs. It displays six at a time. That gives us a second page immediately, which is much easier to test than an inventory with only one item.

> Numbers are not item IDs. Save slot 4 means the game’s fourth manual slot; it does not mean the fourth row currently on screen. A sorted save list can put slot 4 at the top.

2. Build your first inventory menu

1. Open the GUI Designer. Add a GUI named Player Inventory with ID inventory_lab.
2. Keep it hidden at game start and open it from a Button or other deliberate action. Give the window a readable background and a Close Button.
3. Choose Add Element, then Inventory Grid. Name it Bag Grid and give it ID bag_grid.
4. In General, position and size it inside the window. The workshop uses X 32, Y 90, Width 1056 and Height 400 in a 1120 × 660 window.
5. Open Type Settings. Set Visible Columns to 3, Visible Rows to 2 and Visible Slots to 6. Set Slot Width to 310 and Slot Height to 116.

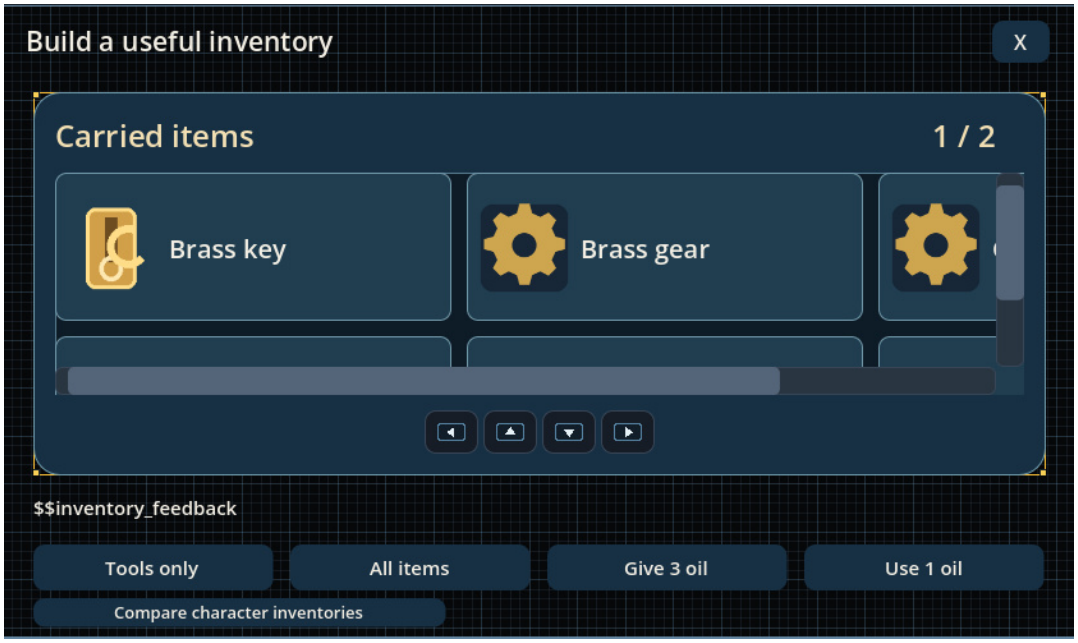

Figure 2. The inventory widget is one element. The title, Close Button and utility buttons belong to its containing GUI.

Start with plain colors and two or three items. Get ownership and clicking right before spending time on artwork. A beautiful empty menu is still an empty menu.

Open, close and map the inventory

The GUI must be reachable during gameplay. A Button called Inventory can run an action list that hides the HUD if needed and shows inventory_lab. Its Close Button does the reverse.

Button	Click actions, in order
Inventory on the HUD	Hide GUI: your HUD, if it would overlap. Show GUI: inventory_lab.
Close inside the inventory	Hide GUI: inventory_lab. Show GUI: your HUD.

In Global Settings, go to the custom system interface assignments near the bottom of General. Set Inventory Screen to Player Inventory(inventory_lab). Apply Changes, then Save Project. This assignment lets the inventory system recognize your custom inventory window.

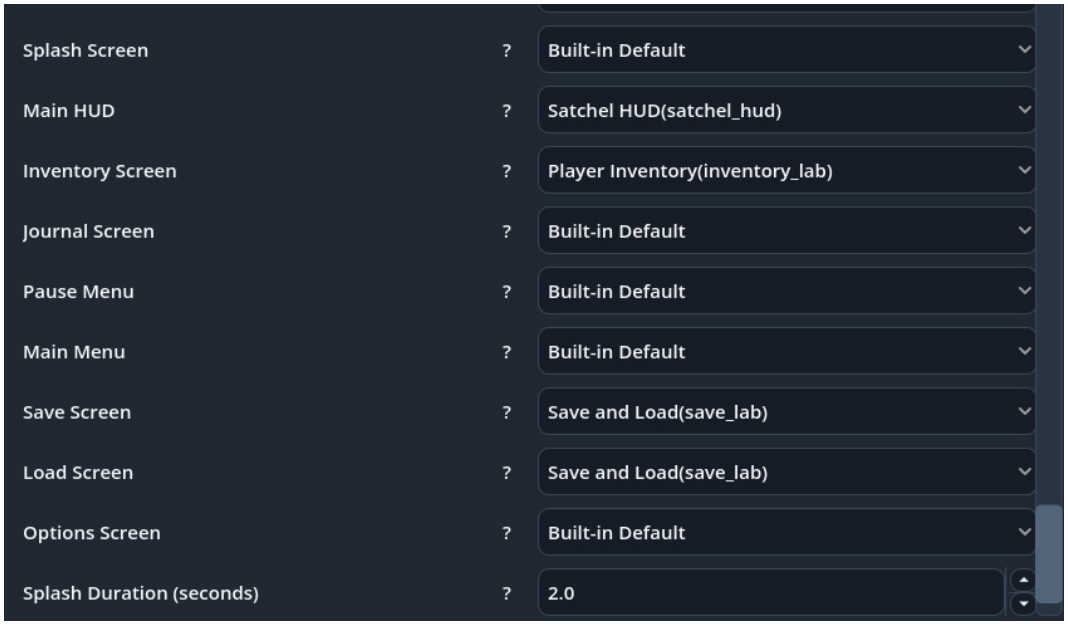

Figure 3. Inventory Screen, Save Screen and Load Screen point to authored GUIs. Their IDs connect the game systems to your menus.

> Mapping matters for a modal inventory. A modal window blocks ordinary world input. The inventory system makes an exception for the GUI assigned as Inventory Screen; another modal GUI can prevent ordinary item selection. Test the window after you map it.

Use modal behavior for a full inventory screen when you want to block clicks into the room. Use a non-modal strip for a bag that remains part of the HUD. Avoid leaving HUD buttons over the inventory’s own controls.

3. Define the items the grid will show

1. Open the Inventory Designer and add an item.
2. Give it a stable ID such as brass_key and a readable name such as Brass key.
3. Enter a description for Look and other explanatory uses.
4. Choose its main icon from an imported image asset.
5. Apply Changes. Repeat for the other items you want the player to collect.

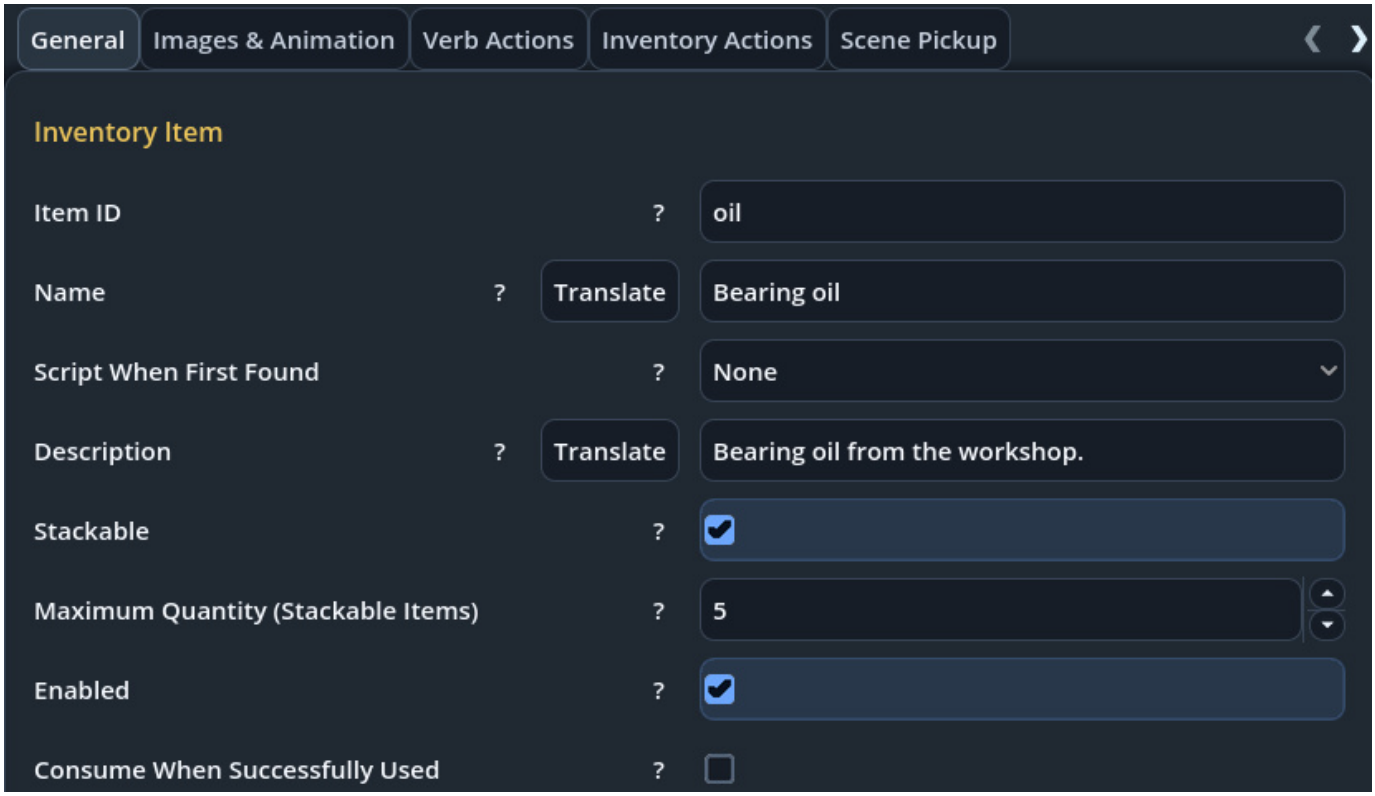

Figure 4. Bearing oil is an inventory definition. The grid reads its name, icon and carried quantity.

A 3D key sitting in the room and an inventory key are separate records. The room object handles being seen or clicked in the scene. Its Take action gives the inventory item and usually removes or hides the room object.

> Do not use the object’s scene ID as the inventory ID unless that is deliberately how you named both records. Give Inventory expects an inventory definition, not an arbitrary 3D object.

Stackable items and maximum amounts

Setting	Example	What the player gets
Stackable off	Brass key, Max Stack 1	At most one copy of that item ID.
Stackable on; Max Stack 5	Bearing oil	One visible slot with a quantity from 1 to 5.
Give 3 oil when the player has 1	Maximum 5	Four oil.
Give 20 oil when the player has 4	Maximum 5	Five oil, not 24.
Remove 1 oil from a stack of 1	No amount remains	The item disappears from the carried list.

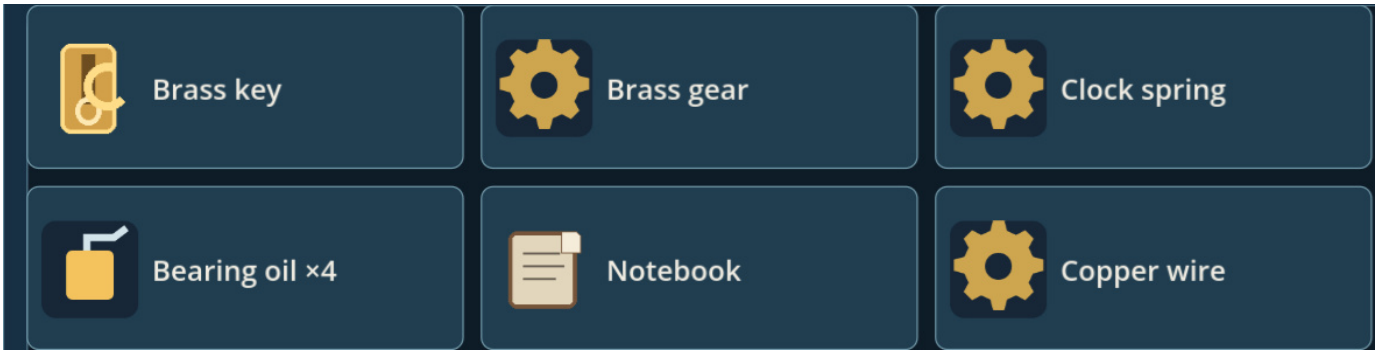

Figure 5. The oil occupies one cell. Its quantity is part of the caption; four copies do not make four cells.

The widget’s capacity counts different item IDs, not the total number of objects. Eight keys represented by one stackable item still use one slot. Eight different key IDs use eight slots.

Show Quantity is supported as showQuantity in the element JSON. It is true in this example. The quantity suffix appears when the amount is greater than one. If you want a custom counter in a separate Label, update it from game logic.

> Use Remove Inventory to take items away. Giving an amount of zero is not a reliable way to mean “do nothing”; the add action normalizes its count. Check your amount before calling the action.

The item icon has its own states



Figure 6. Item artwork belongs in Images & Animation in the Inventory Designer.

Item image	When it is used
Normal	The ordinary carried item icon.
Hover	The pointer enters this item’s cell.
Leave hover	A short departure state, then the normal icon.
Pressed	The item cell is being pressed.
Drag	The drag preview; otherwise the normal icon is used.

These images belong to the item. The blue surface behind the key is the slot background, which belongs to the GUI element. Changing the key’s Hover Image does not change the slot’s Hover Image.

> Use transparent PNGs for cut-out icons. Keep the object roughly the same size and in the same position in every state. Otherwise a hover can look like the item jumps across its cell.

Animated icons without surprises

An animated icon needs an image asset or authored frame list that supplies several frames. Turning on animation cannot create extra frames from one ordinary picture.

1. Import the frames using the Asset Importer’s appropriate image or animation workflow.
2. Assign the normal and optional state images to the item.
3. Enable the item’s animated-icon setting, choose its frames per second and decide whether it loops.
4. Use the GUI widget’s Animate Icons and icon FPS settings as defaults where the item does not supply its own override.
5. Run the game. Check the normal icon, hover, press and release, not just the designer’s still picture.

Situation	Behavior to expect
The item explicitly disables icon animation	That item remains still even when the widget allows animation.
The item supplies its own icon FPS	Its rate takes precedence over the widget’s default.
Animation is off	The first frame of each state is used.
The player drags the item	The drag preview uses a first-frame image rather than a full animated preview.

Use animation sparingly in an inventory. A softly moving potion is easy to read. Twenty flashing item icons make it harder to find the key the player actually wants.

If all items animate except one, inspect that item’s explicit animation setting before changing the entire GUI.

Types, categories and carrying rules

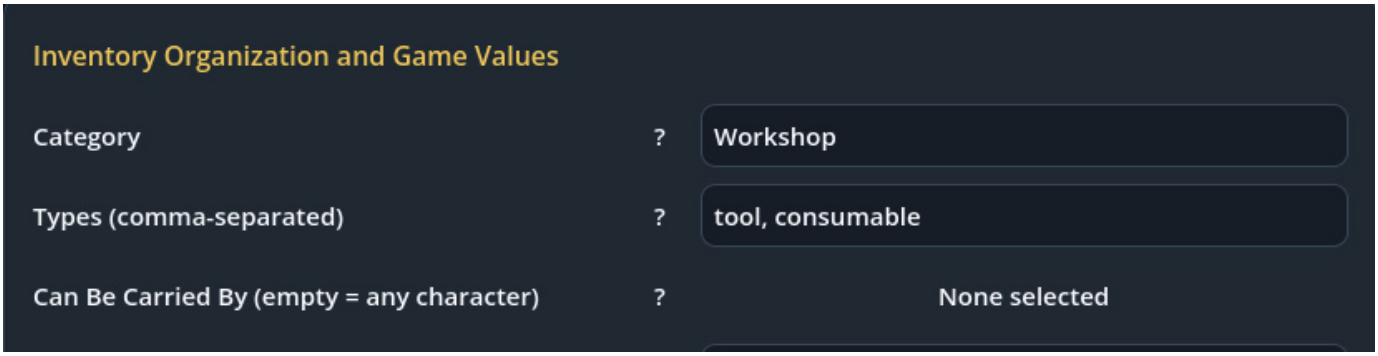

Figure 7. Types and Can Be Carried By serve different purposes. Category is not the widget’s type filter.

Types are practical labels used by the inventory widget’s filter. The workshop’s key has key and tool. Oil has tool and consumable. The notebook has note. One item can belong to several types.

Category, value and other organizational fields can help you arrange or describe item definitions. In this build, the grid’s Inventory Types filter reads the item’s Types list, not its Category or tag text. Setting Category to Tools alone does not make it match tool.

Can Be Carried By restricts ownership. Leave it empty to allow any eligible player character. Choose the Assistant for the Field radio if only that character may carry it. The runtime rejects giving it to Builder; the restriction is more than a visual filter.

> Do not treat Sort Order in an item definition as an automatic sort command for this grid. The carried display follows the runtime inventory order, including player reordering when that is allowed.

4. Choose who owns the contents

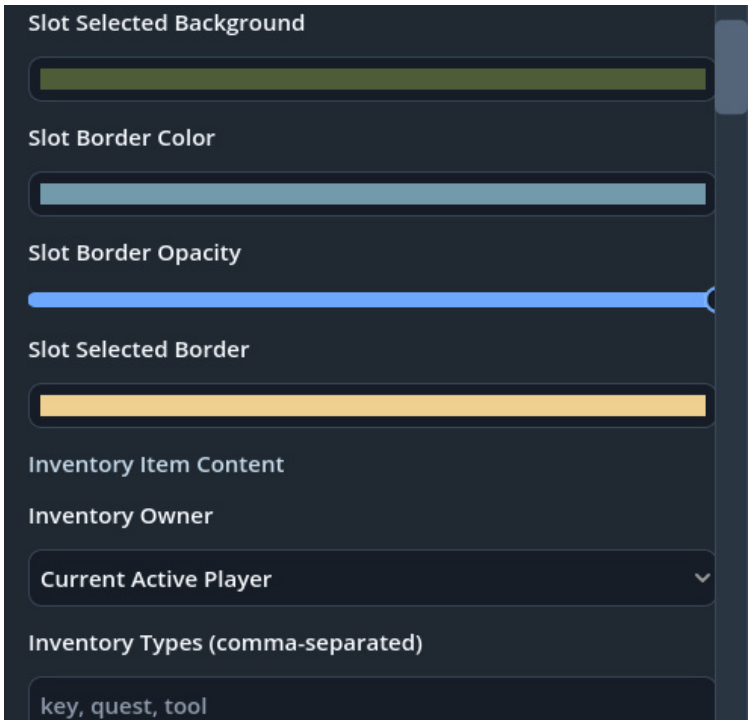

Figure 8. Inventory Owner chooses the source of the contents. Inventory Types narrows that source.

Owner setting	Use it for
Current player	The normal bag. It follows the active playable character.
Specific character	A view of one named character’s inventory, even while another character is active.

In a per-character inventory game, Builder and Assistant can carry different things. In a shared-inventory game, the game’s common collection is the source instead. Choose that policy in the project’s character/inventory settings before building puzzles around separate bags.

Initial Inventory gives characters their starting items. In per-character mode, each playable character receives its own authored starting list. The project-level initial inventory is also added to the active starting character. Avoid duplicating a stackable item in both places unless you intend to give more than one.

> Adding an item definition to the database does not place it in Initial Inventory. If the designer shows no items, check ownership and the starting list before changing the grid size.

Compare two characters’ bags

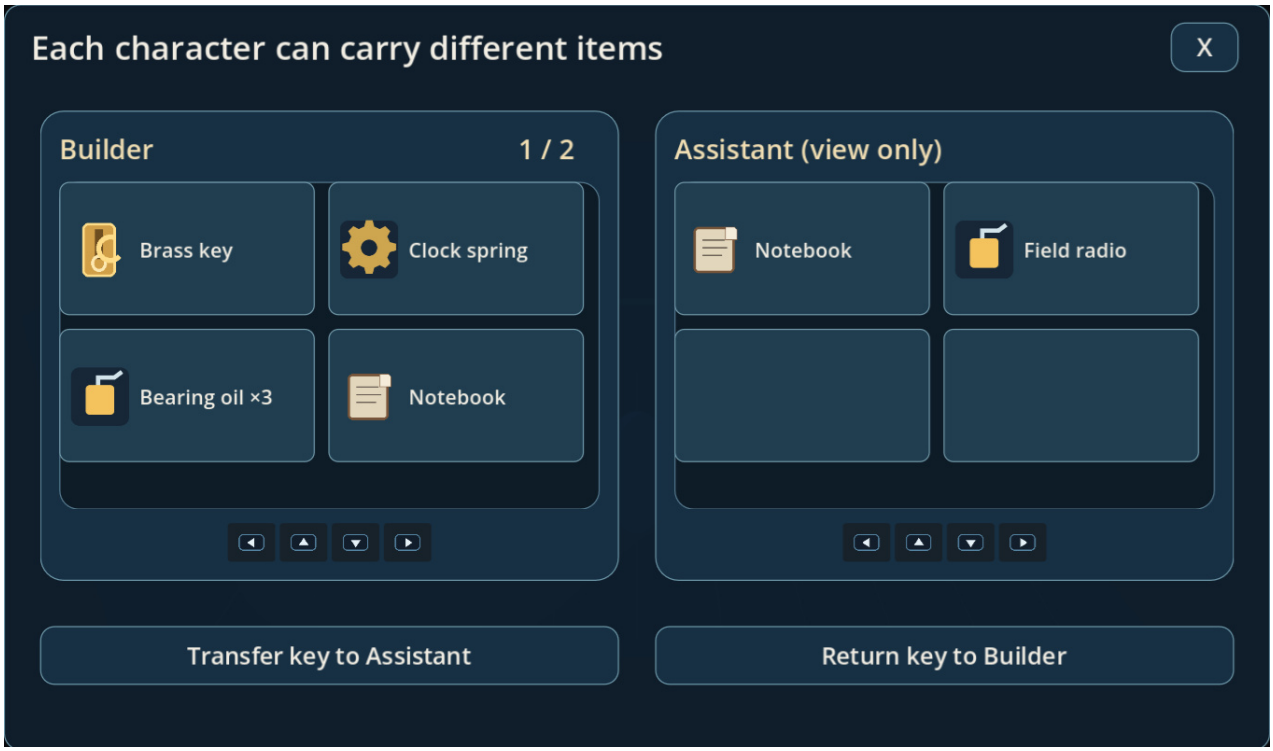

Figure 9. Two Inventory Grids show separate characters. Only the active player’s built-in item controls operate on that player’s inventory.

1. Create two grids with different element IDs, such as hero_bag and assistant_bag.
2. Set the first to Current player.
3. Set the second to Specific character and choose Assistant.
4. Give the Assistant a notebook and field radio in Initial Inventory.
5. Run the game as Builder. Verify that the radio appears only in the Assistant’s view.

The view for another character is read-only for built-in selection, verb use and item dragging. Clicking their radio is not permission to use it as Builder’s selected item. For an exchange screen, create explicit transfer buttons or actions.

> In a shared-inventory project, two character views are not a reliable way to show two independent bags. Use per-character ownership when the story depends on who has the item.

5. Give the grid enough room

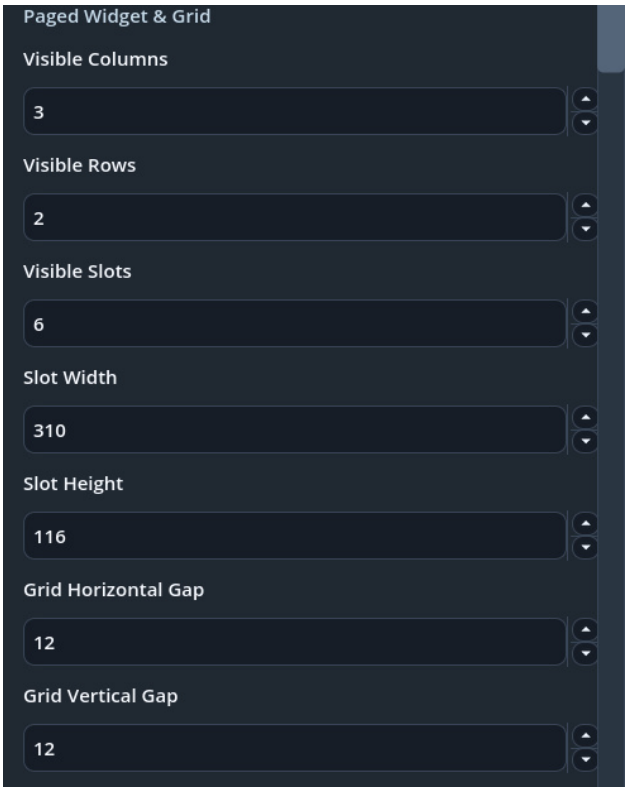

Figure 10. Visible Columns, Visible Rows and Visible Slots describe capacity. Slot Width and Height describe each cell.

For the first menu, use 3 columns, 2 rows and 6 visible slots. The effective page capacity cannot exceed columns × rows. Asking for 12 visible slots in a 3 × 2 layout does not create two extra rows.

Slot Width and Height do not automatically shrink to fit the widget. If the cells need more room than the element provides, internal scrollbars or clipping appear. Enlarging the outer element also does not automatically enlarge every cell.

> Begin with a simple rectangle. Fit the cells, header and arrows. Add decorative padding and artwork after the layout works at the game’s intended resolution.

Work out the width and height

The cell area needs at least the total cell sizes plus the gaps between them. The widget also needs padding, its optional header and navigation controls.

Part	Workshop calculation
Cell area width	3 × 310 + 2 × 12 = 954
Cell area height	2 × 116 + 1 × 12 = 244
Widget padding	16 on each side, adding 32 horizontally and vertically
Header and arrows	Extra vertical room; they are not included in the cell-area calculation.
Chosen outer size	1056 × 400, leaving useful space around the 954 × 244 cells.

Border styles and the scrollbar’s own width take a little space too. Do not fit the arithmetic to the last pixel and assume every theme will behave identically. Leave some breathing room.

The Save Game Widget uses the same layout machinery. A thumbnail can make a save card’s content taller than its authored slot height, so budget for the image, caption, gap and padding together.

> A scrollbar appearing inside a paged grid usually means the current page’s content does not fit. Paging and scrolling are different: the page changes which items are supplied; the scrollbar moves within an oversized supplied page.

Padding, grid offsets and slot offsets

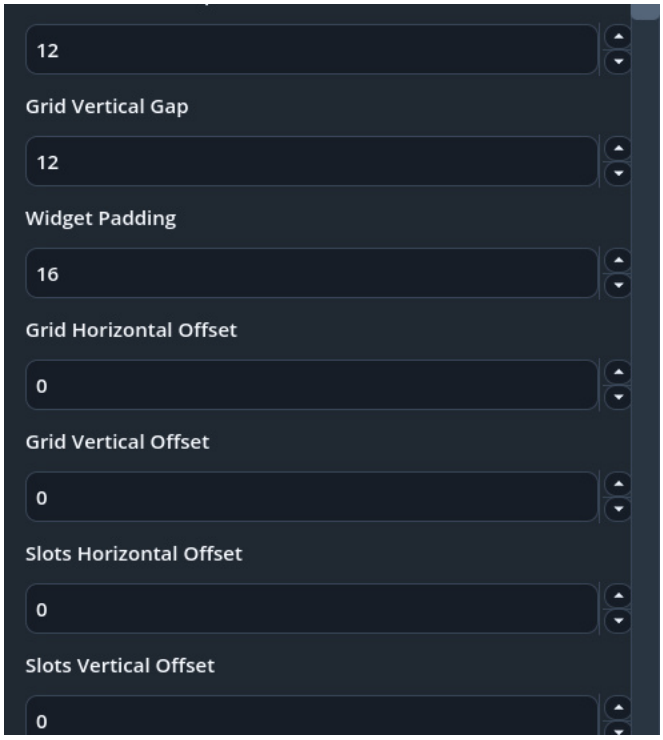

Figure 11. Offsets move different parts of the widget. Keep them at zero until the basic layout fits.

Control	What it moves or spaces
Widget Padding	Space inside the outer frame, around the widget’s content.
Grid Horizontal/Vertical Gap	Space between neighboring cells.
Grid Offset X/Y	The whole grid holder, including its background and scroller.
Slot Offset X/Y	The cells relative to the grid panel.
slotPadding, advanced JSON	Space inside an individual cell, around its icon and text.

Use grid offsets to align the cell panel with a decorative frame. Use slot offsets when the panel itself is in the right place but the cells need to sit differently inside it. These are layout adjustments, not extra invisible storage cells.

> Large positive slot offsets add content space. Negative offsets can push cells beyond the clipped area. If an item seems cut in half, reset both sets of offsets to zero before changing its image.

Headers, item captions and icons

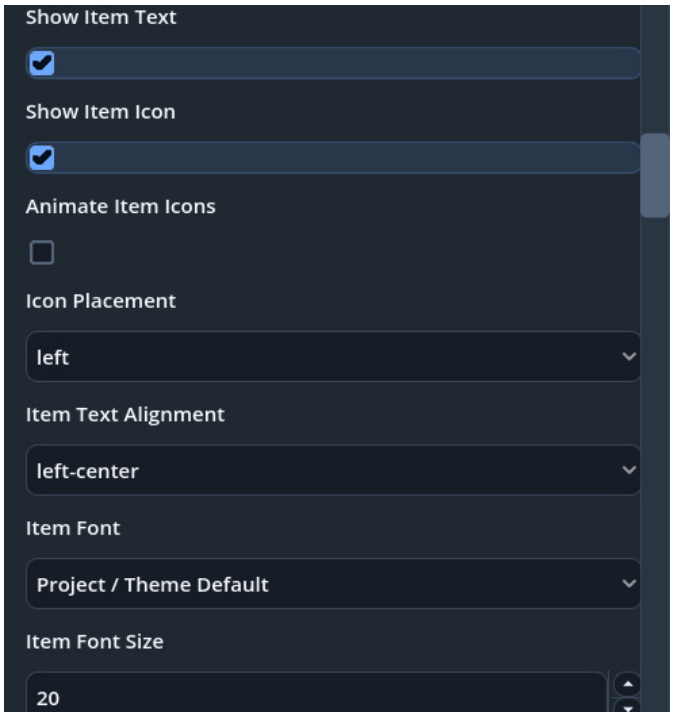

Figure 12. The item content controls choose text, icon placement and text style.

Show Header controls the title and its page indicator. Use a short title such as Carried items. The page indicator is hidden when there is only one page; hiding the header also hides that indicator.

Show Item Text and Show Item Icon can be used independently. An icon-only bag is compact; a text-only list can suit a parser game. A new adventure is easier to understand when the player can see both the object and its name.

Icon Placement can put the icon to the left, right, above or below the caption. A top placement needs a taller cell. The example uses a left icon with a 72-pixel iconSize and 8-pixel slotPadding, both supported as advanced element properties.

Title font controls style the header. Item font controls style names and quantities inside cells. Hover, click and right-click text styles are separate state overrides. Start with the same size in every state so the text does not jump.

> Inventory captions use the item’s name. Do not put normal Label wildcards such as $$score in an item name and expect the inventory grid to expand them. Use a separate Label for changing descriptions or counts.

6. Page through the carried items

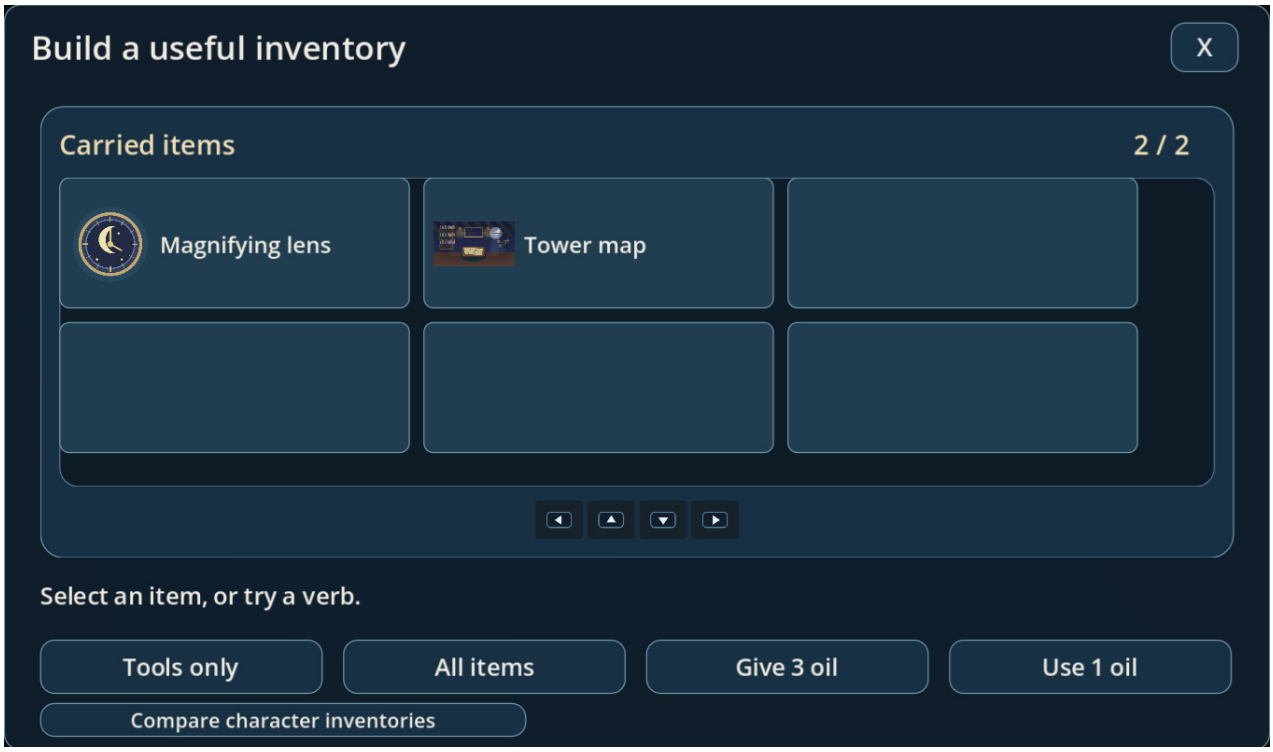

Figure 13. Page 2 contains the remaining item IDs. Empty cells are display space, not additional possessions.

Control	Movement
Left / previous	Back by a full page of visible slots.
Right / next	Forward by a full page.
Up	Back by one row: the number of columns.
Down	Forward by one row.

A 3 × 2 grid pages six IDs at a time. A row step moves three. The final page can contain fewer items. The page-start position is clamped so navigation cannot run endlessly past the end.

Show Empty Slots is supported as showEmptySlots in advanced JSON. With it true, blank cells fill the page. With it false and no carried items, the widget can show its Empty Text instead. This build does not give the inventory grid a dedicated typed checkbox for every advanced option.

Arrows, wheel, touch and external controls

Scroll Mode	What to arrange
arrows	Use the widget’s navigation buttons.
wheel-touch	Use wheel or touch paging where the widget receives that input.
both	Allow the built-in arrow and wheel/touch paths.
external	Provide your own controls outside the widget.
none	No built-in paging route; only suitable when the list fits or you deliberately control it another way.

Navigation Placement puts the arrow controls above, below, at the sides or split around the grid. It does not change their meanings. Turn off directions you do not need rather than showing four mysterious arrows in a one-row bag.

Wheel and touch behavior depends on where the input lands. A nested scroller can take a wheel event, while a draggable inventory item can take a touch gesture as an item drag. Test over the blank panel, a filled cell and an empty cell.

For an external Button, the legacy inventory-next, inventory-previous, inventory-up and inventory-down event behaviors target an inventory element ID. The GDScript runtime also exposes inventory_gui_command(element_id, command). Use a unique ID and test the controls again after changing filters or rebuilding the GUI.

> In 0.86.3.4, rebuilding a widget through runtime property changes can leave the external inventory-navigation path with an old widget reference. The built-in arrows address their live widget. If an external paging button stops after a filter change, keep the built-in arrows or re-test with a freshly opened menu.

Give each arrow a clear appearance

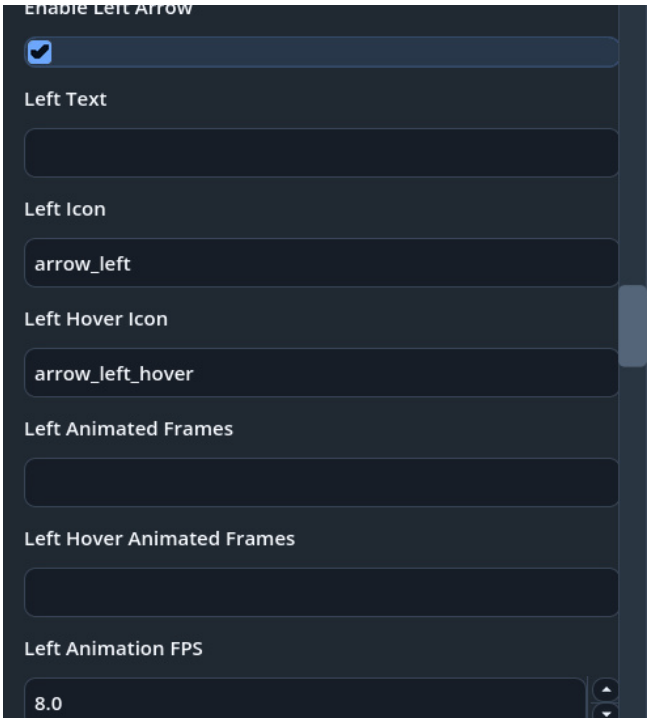

Figure 14. Each direction can have text, an image, a hover image, animation frames and its own FPS.

Arrow Enabled decides whether that direction is available. Arrow Text is useful for Previous and Next, or you can leave it empty when the image already explains the direction.

Arrow Image and Hover Image are navigation artwork. They are unrelated to the items in the bag. Frame lists and FPS allow animated arrows when you have supplied actual frames.

The built-in arrow control has its own minimum button and icon dimensions. Extremely large arrow art is not a way to increase the grid’s slot size. Use a conventional Button outside the widget when you need a fully custom large navigation control.

> There is no separate typed pressed-image picker for every built-in arrow in this build. An external ordinary Button can provide its normal, hover and clicked images, subject to the navigation limitation on the preceding page.

7. Select an item and use a verb

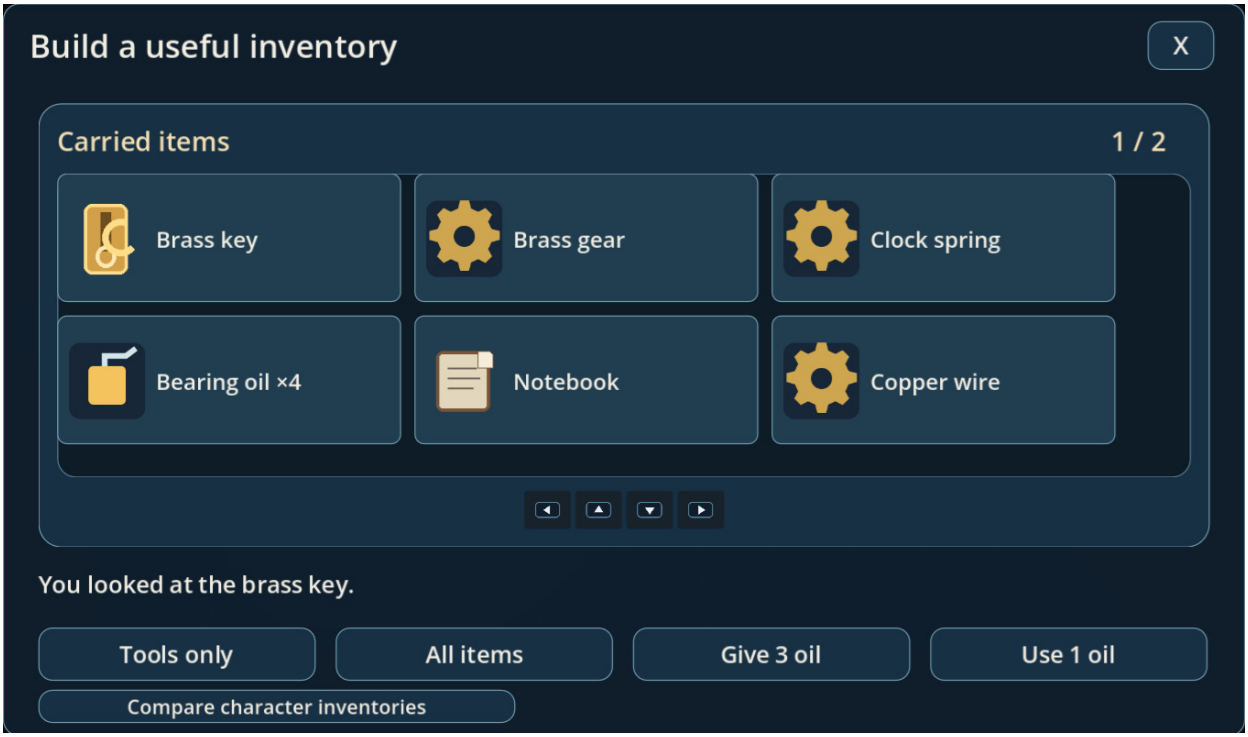

Figure 15. The Look action on Brass key updates a separate feedback Label during gameplay.

The workshop uses Verb Single Click as its interaction mode. In that mode, Walk selects a carried item. Look runs the item's Look actions, or falls back to its description. Other verbs run the actions you authored for that item.

1. In the Inventory Designer, select Brass key and open Verb Interactions.
2. Add a Look action. A Show Message action can display a short description; the workshop uses Set Variable to update inventory_feedback.
3. Put a Label below the grid with text $$inventory_feedback.
4. Run the game, choose Look and click the key. Confirm the feedback changes.
5. Choose Walk and click an item. Verify that it becomes the selected inventory item.

> Use Selected Verb in the widget controls whether it forwards clicks to the verb interaction path. Turning it off uses the inventory-selection path instead. It is not a master switch for all clicking or dragging.

Interaction mode changes the click rules

Inventory behavior is part of the project’s interaction system. Do not assume that left click, right click and double click mean the same thing in every game.

Project choice	What to check in the inventory
Verb Single Click / Verb Double Click	Which click confirms the selected verb and how selecting an inventory item works.
Verb Direct / Verb Direct Double Click	The item-selection path can take priority over a simple Look click. Test the double-click rule too.
Context or Direct	The game decides interactions differently from the workshop’s explicit verb example.
Cycling cursors or verb coin	Make sure the chosen verb and item selection agree with your cursor/menu presentation.

A selected item also changes the meaning of the next inventory click. If oil is selected and the player clicks a different item, the runtime attempts to use oil on that item before treating the click as an ordinary verb action.

If you are testing Look but keep getting “Those items do not interact,” clear the selected inventory item first. You may still be in an item-on-item attempt from the previous click.

> The guide’s click sequences assume Verb Single Click unless a page says otherwise. Set that mode for the first tests, then repeat them in your finished game’s chosen mode.

Use an inventory item on the room

An inventory key normally needs a target in the scene: a door, a lock or a character. Give the key to the player, then author the target’s matching inventory interaction. The target is where you decide what that particular key does to that particular door.

1. Create the inventory item brass_key and a scene object called workshop_door.
2. Give the player brass_key using an action or Initial Inventory.
3. In the door’s inventory interaction settings, add a rule for brass_key.
4. Run an action list or script that checks any puzzle conditions, opens the door and removes the key only if the story calls for it.
5. In Play mode, select the key, close the inventory if the modal window blocks the room, then click the door. Test an unrelated object as well.

The inventory GUI can remain visible as a non-modal strip, or it can be a full modal window that closes before the room click. Pick one clear flow. An item selected successfully inside a bag does not mean the player can click through that bag into the world.

If a verb uses a preposition, such as Use key on door, the verb’s preposition settings and the target’s matching setup must agree. The grid does not add a preposition rule merely because an icon looks like a key.

> Carryable inventory items and physically carried 3D attachments are different systems. Giving a key to the bag does not attach a 3D key to the character’s hand.

8. Combine two inventory items

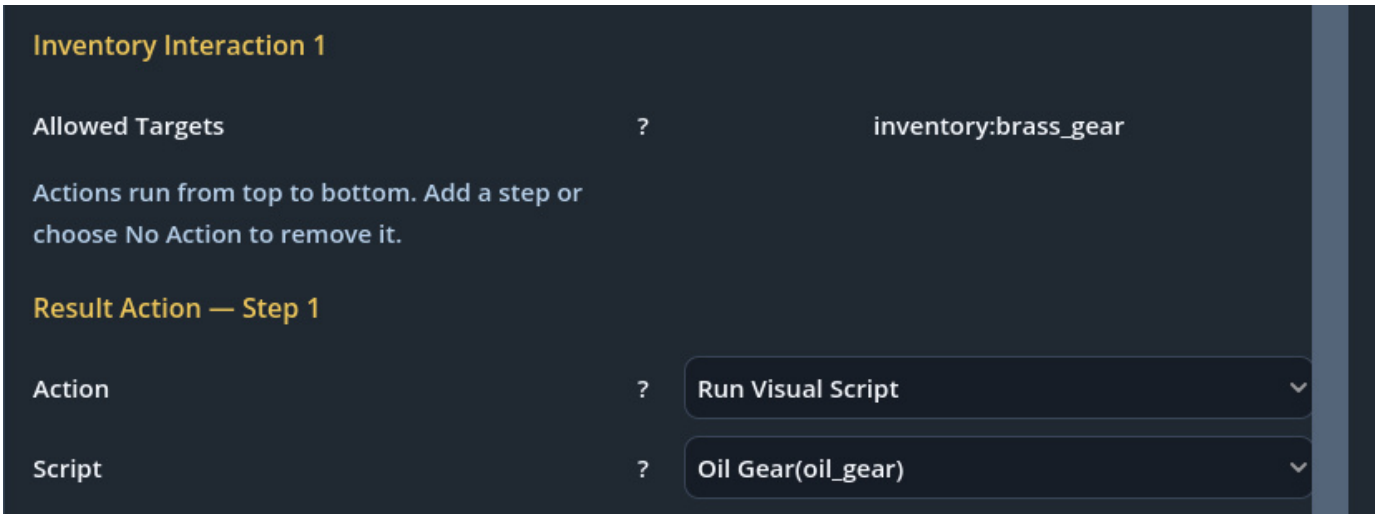

Figure 16. Bearing oil has an Inventory Actions rule targeting inventory:brass_gear.

The workshop combines Bearing oil with Brass gear to create Lubricated gear. A combination rule identifies the target; its action list or script supplies the result. Allow Combine controls drag-to-combine. The selected-item click route can still use a global recipe; it is not a master switch for every combination. Neither route invents recipes.

1. Create oil, brass_gear and lubricated_gear as inventory definitions.
2. Select Bearing oil and open Inventory Actions. Add a rule targeting Brass gear as an inventory item.
3. Choose an action that runs the guarded recipe script on the next page.
4. In the grid’s Type Settings, enable Allow Combine.
5. Run the game with oil and gear in the active player’s bag. Select oil, then click the gear, or test dragging one onto the other.

> The target reference is inventory:brass_gear when you use a typed reference. An entity: reference means a scene target instead. A plain display name is not the reference.

A complete, guarded combination script

Create a GDScript record named Lubricate the gear with ID oil_gear. Apply this complete source:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    if not HasInventory("oil"):
        return false
    if not HasInventory("brass_gear"):
        return false
    await RemoveInventory({"itemId": "oil", "count": 1})
    await RemoveInventory({"itemId": "brass_gear", "count": 1})
    await AddInventory({"itemId": "lubricated_gear", "count": 1})
    await SetVariable({
        "name": "inventory_feedback",
        "value": "The gear is now lubricated."
    })
    return true
```

The checks prevent a free reward if the recipe is called without its ingredients. The removals consume one oil and the old gear. The add gives the replacement item. A second call fails once the old gear has gone.

The result must already have a valid inventory definition, and the active player must be allowed to carry it. If the result is stackable, allow enough capacity for it before consuming ingredients in a more complicated recipe.

> Use explicit Remove Inventory actions on the successful path. Although Consume On Use is exposed in the item settings, do not rely on it as automatic recipe consumption in 0.86.3.4.

Test both the success and failure paths

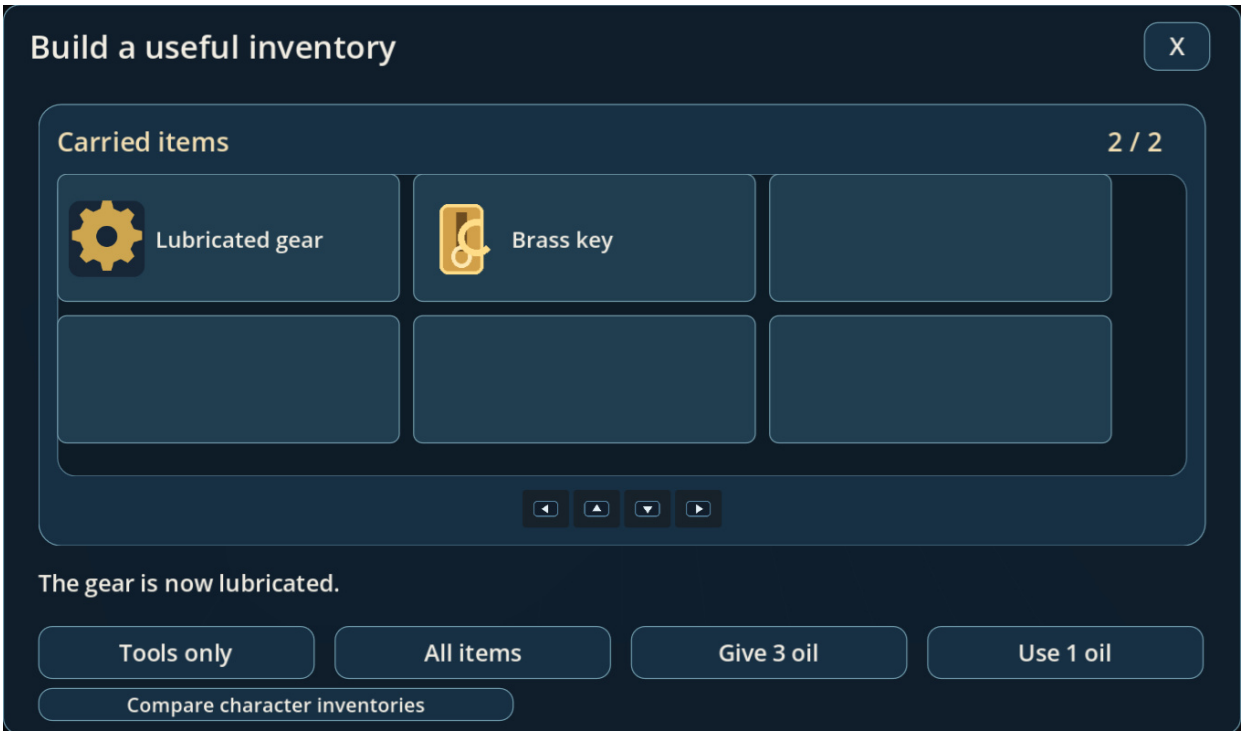

Figure 17. Lubricated gear replaces Brass gear. One oil has been consumed; the rest of the stack remains.

Test	Expected result
Oil and gear are both carried	One recipe result is added; the intended ingredients are removed.
Only oil is carried	No result is added by the guarded script.
An unrelated item is selected	No accidental recipe reward.
The result cannot be carried or is already at its limit	Handle that condition before consumption in your recipe logic.
The rule is reachable from both directions	A single player attempt should run the recipe once.

The runtime can find a rule on the source item or on the target item, with legacy combination data also supported. Keep one obvious place for the recipe rather than maintaining two competing copies.

> A combination’s result action can run a script or action list. Give it the same condition checks you would put in an ordinary puzzle. The GUI is the input surface; the recipe logic decides what is allowed.

Dragging: reorder or combine?

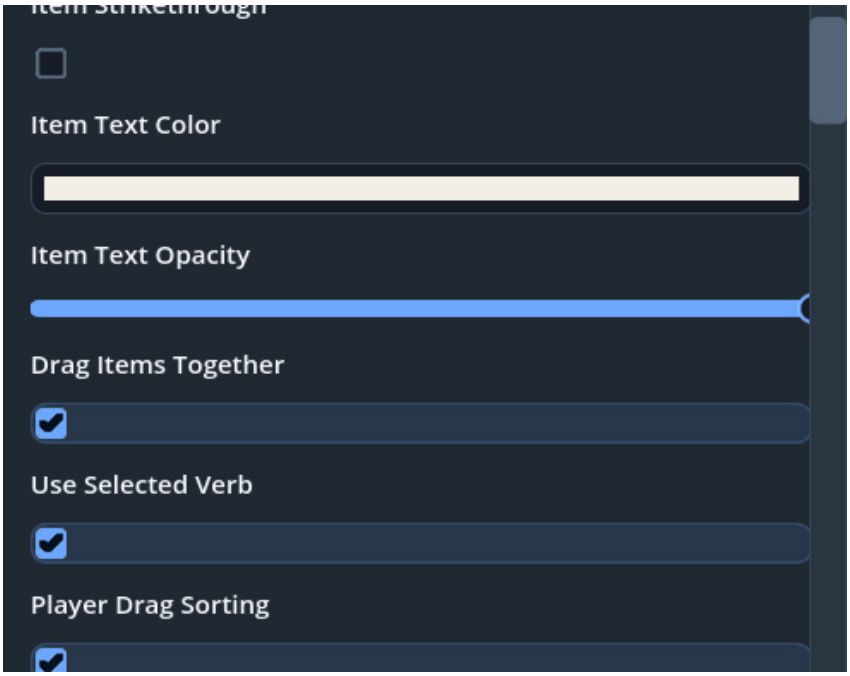

Figure 18. Allow Combine, Use Selected Verb and Allow Player Drag Sorting govern different behavior.

With Allow Player Drag Sorting on, dropping an item over another item can reorder the carried list. The operation inserts the source before the target; it is not a simple exchange of two cells.

If combination is enabled and a valid recipe exists for the pair, the recipe takes precedence. If no recipe applies and sorting is allowed, the drop can become a reorder instead. A blank cell can also be a destination when that path is allowed.

Turning off sorting does not necessarily disable all item dragging. A drag can still be useful for a combination or for choosing an item to use outside the bag. Decide whether you want to allow recipes, reorder behavior, or both, and test each path.

Be especially careful with a filtered inventory. Visible cell positions are relative to the filtered view, while the carried order belongs to the character’s full inventory. Test a reorder, switch to All items and check the result before enabling that combination of features in a released menu.

> Touch dragging and wheel paging share space with item interactions. A mouse test is not a substitute for a touch-device test when your game targets tablets.

9. Add, remove and transfer items

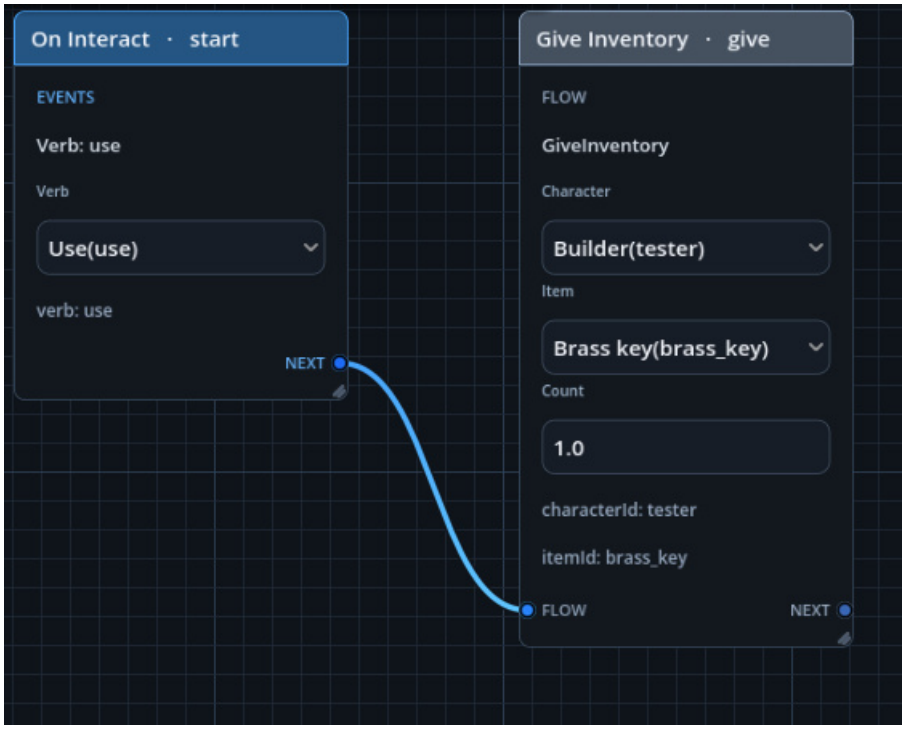

Figure 19. A visual script can give an item using the same shared action system as a Button or action list.

Action	Useful fields	Typical use
Give Inventory	Character, Item, Count	Give a named playable character an item.
Add Inventory	Item, Count; optional characterId	Give the active player an item through the runtime action API.
Remove Inventory	Item, Count; optional characterId	Consume, trade or take away a carried amount.
Change Inventory Quantity	Character, Item, Change By	Increase or decrease a stack.
Change Inventory from Player to Player	From Player, To Player, Inventory Item, Quantity	Transfer an item between playable characters.

Select the exact action in the action editor and fill its displayed fields. Shared action labels can differ from their script names; Change Inventory from Player to Player is TransferInventory in source.

> Give a stack of oil, then inspect it in a running game. The designer’s sample list is not a live readout of what your last test run collected.

Transfer to another character safely

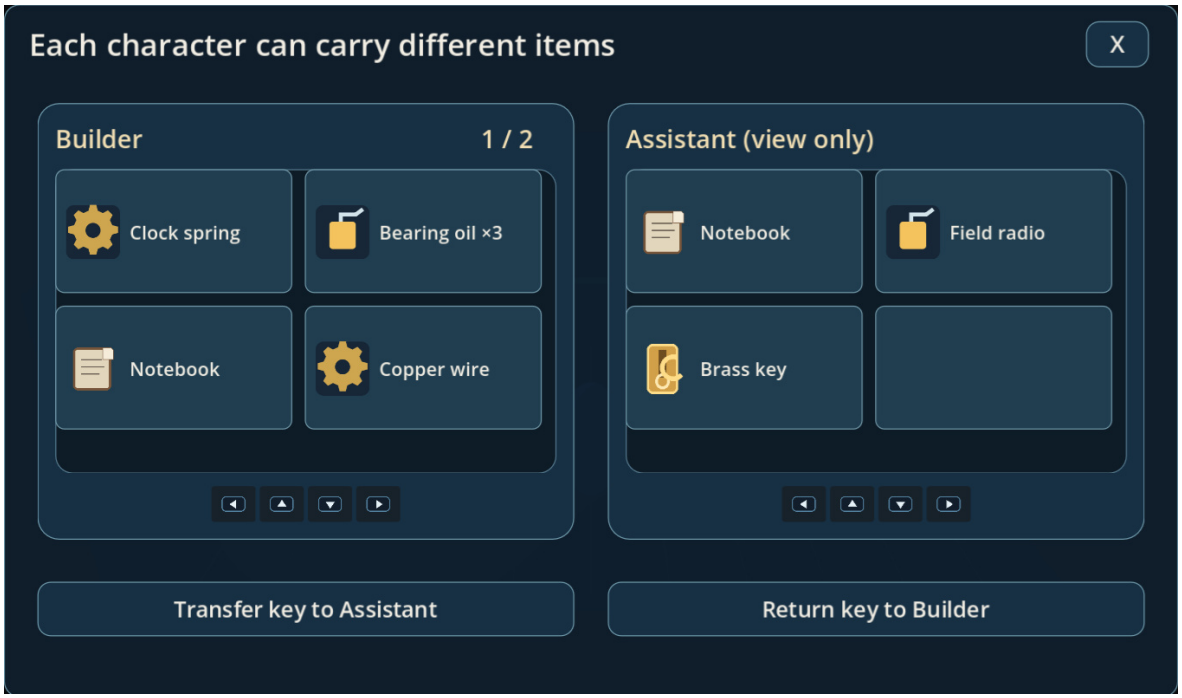

Figure 20. After transfer, the key leaves Builder’s bag and appears in Assistant’s bag.

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    if not HasInventory("brass_key", "tester"):
        return false
    await TransferInventory({
        "fromCharacterId": "tester",
        "toCharacterId": "assistant",
        "itemId": "brass_key",
        "count": 1
    })
    return true
```

The source and destination are character IDs. This is an explicit game operation, not a drag between two read-only owner views. Check the recipient’s carrying restrictions and stack capacity for important exchanges.

For a trade, do not remove payment and announce success before checking that the other character can receive the item. Design a clear failure path.

First-found scripts and world pickups

On Found Script is useful for a response when an item enters an empty carried entry: a journal update, a remark or a sound. It can run again if the item is lost completely and later acquired again. It is not, by itself, an “only once in the entire story” flag.

For a once-only reward, create a Boolean variable such as key_reward_given. Check it in the script, perform the reward when false, and set it true. That variable becomes part of the saved game state.

Scene Pickup / acquisition-removal settings can link an inventory item to a specific scene object. The acquisition path can then remove that world pickup and persist the removal state. Use the exact scene and entity references.

1. Make the scene pickup visible and interactive at the start.
2. Give the inventory item only after the Take conditions succeed.
3. Hide or remove the scene pickup through your chosen action/removal setup.
4. Leave the room and return. Confirm it does not appear again.
5. Save after pickup, restart and load. Confirm the bag and world agree.

> Starting with a linked item in Initial Inventory can also trigger acquisition behavior. Do not place the same supposedly unique key on the floor unless your opening state accounts for it.

10. Style the independent parts

Layer	Inventory Grid	Save Game Widget
Outer widget frame	The whole bag’s surface and border.	The whole save browser’s surface and border.
Grid panel	The surface around the collection of cells.	The surface around the save cards.
Slot frame	One inventory item cell, repeated.	One save row or card, repeated.
Screenshot frame	Not used for ordinary item icons.	The thumbnail area inside a save card.

These layers have separate styling. The key’s icon is content inside the slot; it is not a fourth rounded inventory panel. A Save Game Widget adds a true screenshot area, with its own corner and image controls.

Select the widget and open Visual Style. Work through Widget Frame, Grid Panel and Slot Frames. On saves, also work through Save Screenshot. Change one conspicuous corner first so you can see which layer you are editing.

> The containing GUI Window has another independent frame outside the widget. Editing the GUI Window’s border radius should not be mistaken for editing the inventory element’s root frame.

Set each corner without changing the others



Figure 21. Slot corner controls apply to the repeated cell frames. The grid and root frame have separate sections.

Corner value	Meaning
-1	Inherit the common radius for this same layer.
0	Make this corner square.
A positive number	Give this corner that particular curve.

For example, use outer Radius 24 and outer Top Left 40. Give the grid Radius 12 and grid Top Left 0. Give the slots Radius 8 and slot Top Left 20. Those are three different shapes; changing the outer value does not mean all three should become identical.

The four corners are Top Left, Top Right, Bottom Right and Bottom Left. Keep that order in mind when comparing the inspector to the visible rectangle.

> Old projects may have their previous inherited appearance preserved when normalized. Inspect each layer’s stored values after opening an older project; do not assume every -1 reaches all the way up to the GUI Window.

Colors, selection and surface effects

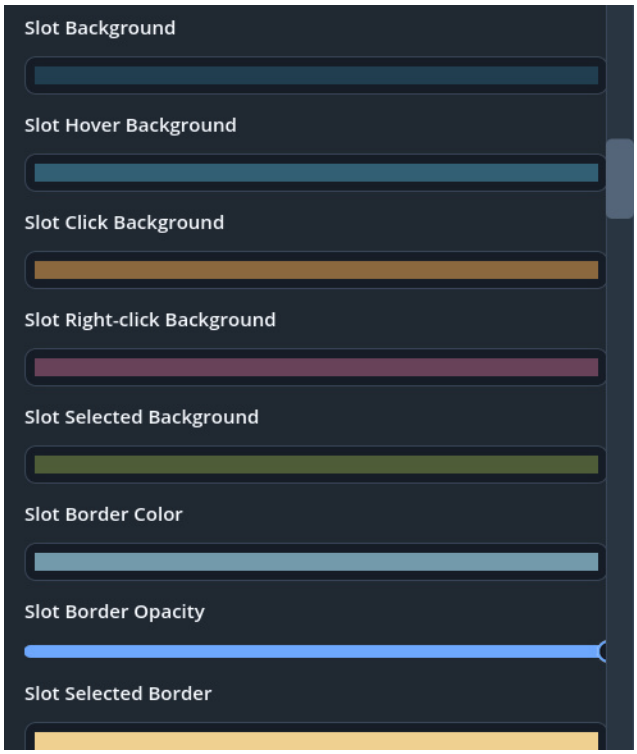

Figure 22. Grid and slot colors have different normal, hover, click and selected roles.

Use the grid background as a quiet backdrop. Use Slot Background for the ordinary cells. Hover, Click and Right Click backgrounds provide temporary feedback. Selected Background and Selected Border Color identify the currently selected item.

Selection is not the same thing as a held press. A selected key may remain highlighted after release while its pressed artwork has returned to normal. This is useful feedback, not necessarily a stuck clicked image.

Widget Opacity and Grid Opacity describe their respective surfaces. Border controls describe outlines. Surface/reflection/shader settings are appearance options; they do not change which items a character carries or whether an inventory recipe is valid.

Apply shaders gently to a menu containing text. Strong blur or glow can make the slot captions difficult to read. Rounded frame art also does not guarantee every child control is clipped to the same curve; keep the contents inside its safe area.

> If a color change appears to do nothing, check whether an opaque background image is covering it. Remove the image briefly or use artwork with transparency to reveal the color below.

11. Assign images to the right layer



Figure 23. The widget uses separate root, grid and slot artwork. Item icons remain independent content.

Open the Images tab, using the tab arrows if necessary. Each supported region has Background Image, Hover Image and Clicked Image. The clicked image is the pressed state; it is not a permanent picture after the click has finished.

Image section	What its artwork sits behind
Widget / outer background	The complete widget, including the space around the grid.
Grid	The group of cells, as one backdrop.
Slot	Every repeated cell.
Save Screenshot	The picture area inside each save card; discussed later.

Do not put a detailed key picture into Slot Background when you mean the carried key’s icon. Every item cell will receive the same slot background art.

Fit, transparency and background animation

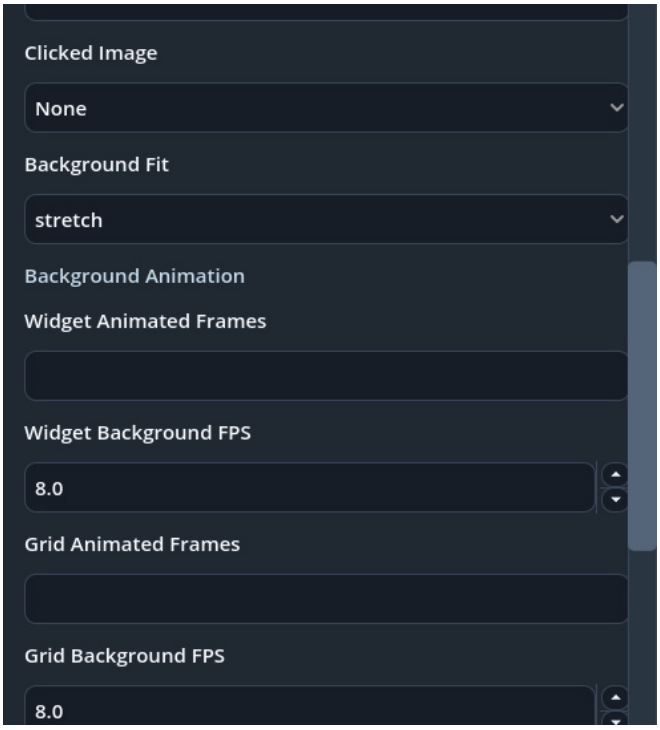

Figure 24. Widget and grid background frames can be animated independently of the item icons.

Fit	Result
stretch	Fills the region exactly, possibly changing the picture’s proportions.
contain	Keeps the whole picture visible, with unused space when the shape differs.
cover	Fills the region while preserving proportions, cropping the excess.

Use a neutral stretchable texture for a slot background. Use contain for an illustration whose full edges matter. Use cover for a decorative panel that can safely lose some of its edges.

Widget Background Frames and Grid Background Frames can supply animated backdrops; their FPS settings control those animations. A frame list is a sequence of imported image references, not a time in seconds. Start with a still image before adding animation.

> Normal image fallbacks can include the widget’s Media Asset or older background fields. When replacing old artwork, inspect the Images tab and advanced values so an older fallback does not keep appearing.

Hover, press, release: the expected result



Figure 25. Pressed artwork is a temporary state while the pointer is held down.

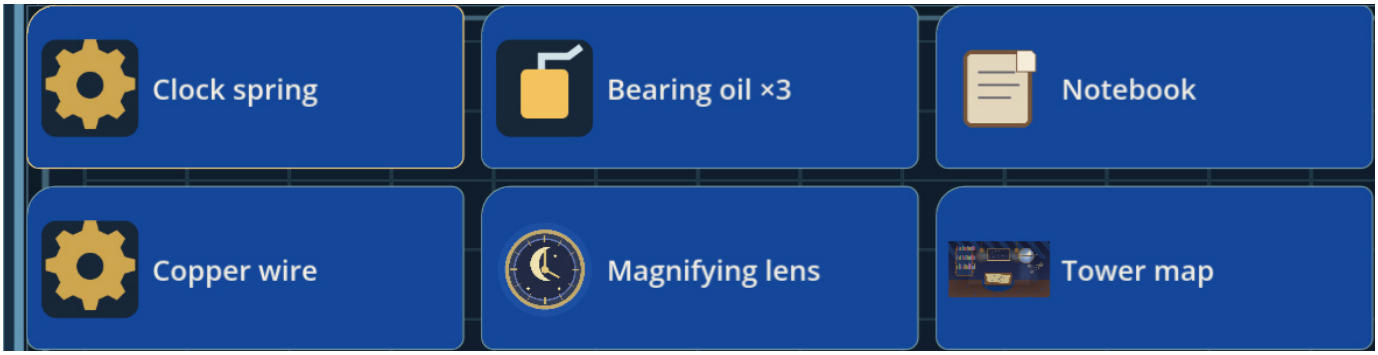

Figure 26. After release, the normal artwork returns. A selection highlight may remain separately.

The widget’s image regions and item cells observe the same basic release rule: press for pressed art, release for normal art. Hover is suppressed under the stationary pointer after release and returns when the pointer leaves and enters again.

Test a press released over the item, a drag released outside it, and a window losing focus during a press. Do this in both GUI Preview and a running game. The intended appearance is temporary press feedback, with deliberate selection handled by its own style.

> GUI Preview is for appearance and local visual behavior. It does not run your item rewards, puzzle scripts or save operations. Use Play for those tests.

12. Build useful inventory variations

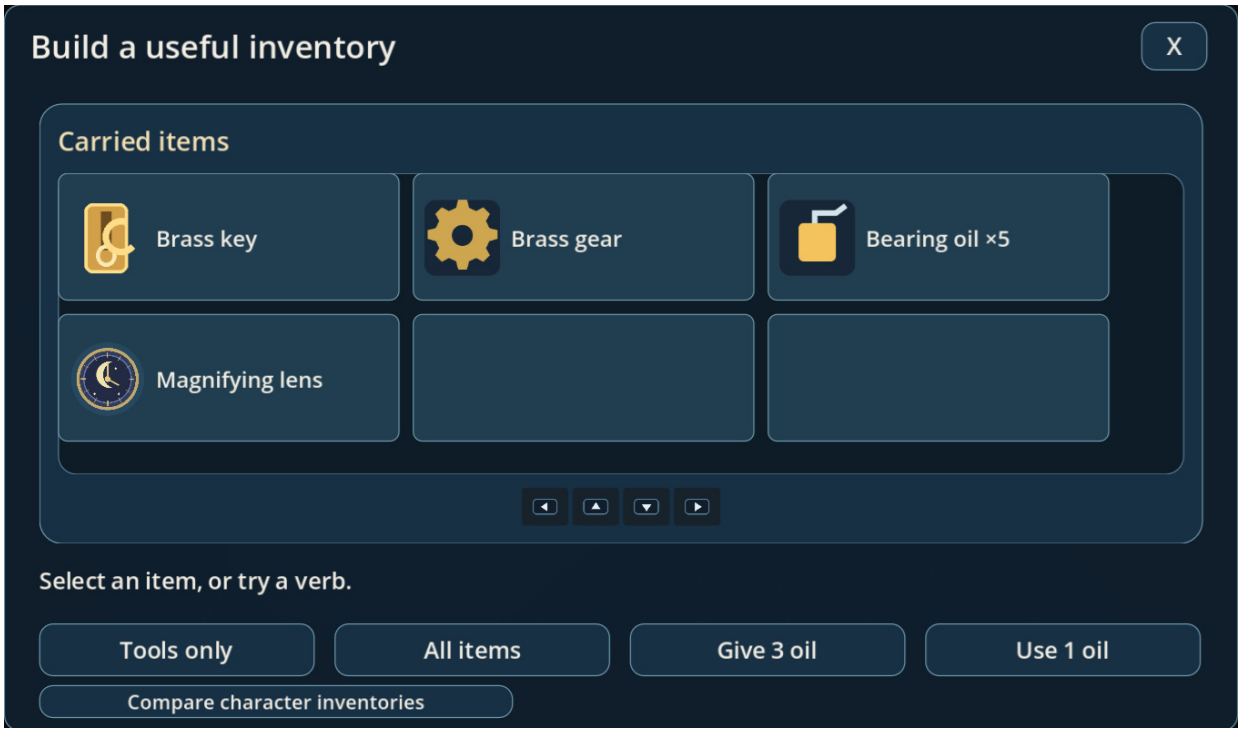

Figure 27. A tool-only view hides the notebook and map while they remain in the carried inventory.

1. Give item definitions Types such as tool, note and key.
2. Create a second authored inventory view with its own element ID. Set its Inventory Types field to tool in Type Settings.
3. Keep your first view’s filter empty so it shows all carried items.
4. Use ordinary Buttons to show the intended GUI/view and hide the other. Author the filters before running the game.
5. Run the game, switch views, then return to All items. Confirm no inventory was removed.

The filter accepts a list separated by commas, semicolons or new lines. Matching ignores case and trims spaces. Multiple types are an OR filter: tool,key means items with either type, not only items with both.

> A filter changes a view, not possession. HasInventory still sees a hidden notebook. Runtime Change GUI Element Property can change inventoryTypes too, but the rebuild/refresh limitation in chapter 26 makes separately authored views the safer starting point in 0.86.3.4.

A compact HUD, tabs and a detail area

Menu idea	Practical setup
One-row inventory strip	Use several columns, one row, a short slot height and a non-modal GUI. Hide the header if a page number is unnecessary.
All / Tools / Notes tabs	Use ordinary Buttons to switch between separately authored filtered views. Give the selected tab its own visual feedback.
Icon-only bag with a detail area	Hide Item Text in the grid. Use item actions or explicit game logic to update a separate Label and Image.
Trade or companion screen	Use owner-specific views and explicit Transfer Inventory actions. Check carrier rules and quantities.
A shop	Use your own stock, prices and purchase conditions. The inventory grid displays owned items; it is not a complete shop system.

Use a separate Label for dynamic text such as $$inventory_feedback. Ordinary Labels expand supported game wildcards; the inventory cell’s own item name does not behave like a general-purpose Label template.

For per-item responses, prefer the item’s Verb Interactions and Inventory Actions. In 0.86.3.4, do not rely on the grid element’s general Events tab to deliver a dependable clicked-item callback for every internal cell. Those internal events are not wired like Save Game Widget slot events.

> Keep the first version simple: one bag, one close button, clear item names and a visible selected state. Add tabs and detail panes after the item flow works.

13. Prepare the game’s save system

Set the game-wide save policy before arranging its menu. Global Settings controls the total manual slots, autosave, quick save and screenshot capture. The widget controls which of those slots it displays and what the player can do with them.

1. Open Global Settings. In 0.86.3.4, Save Slots is near the bottom of the Screen Overlay tab. Set it to 8 for this workshop.
2. Open Player Options. Find Automatic Game Saving and Save Slot Presentation.
3. For the first tests, leave automatic saving off. Use Autosave Slot 0 and Quick Save Slot 1.
4. Enable Capture Save-Game Screenshots. Use Width 320 and Height 180.
5. Apply Changes and Save Project.

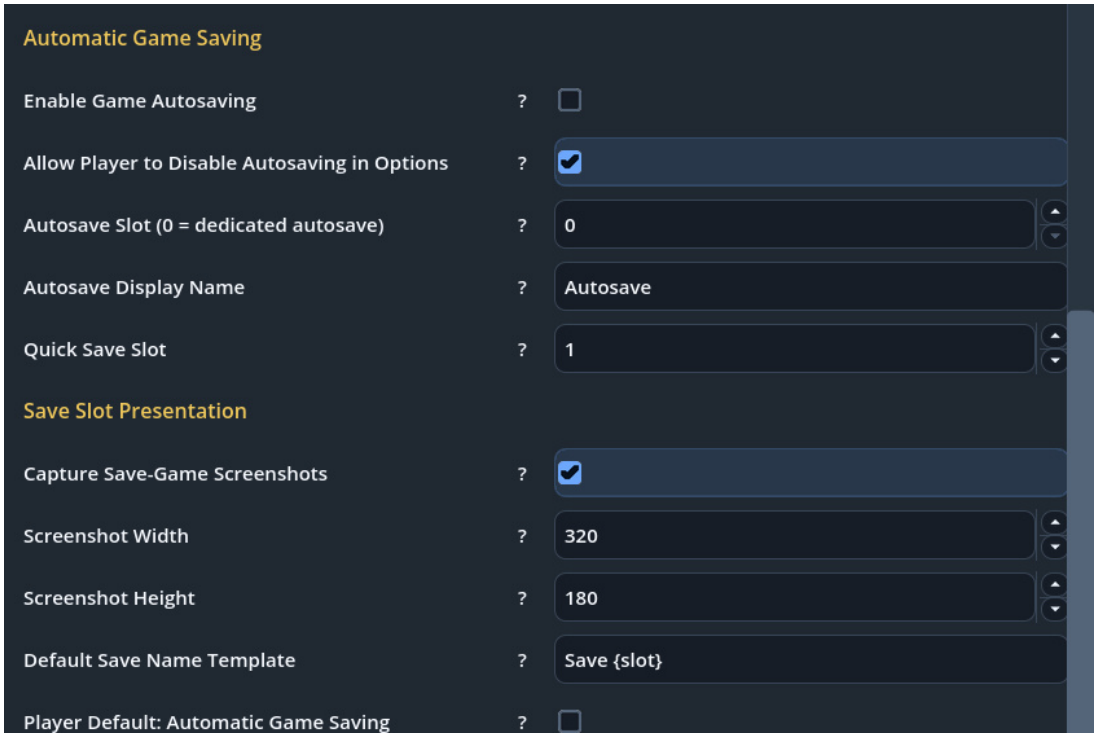

Figure 28. Autosave, quick save and screenshot capture are game-wide settings, separate from the widget’s appearance.

> The Save Slots limit accepts 1–100 in this build. A widget’s End Slot cannot create additional valid slots beyond that project limit.

Autosave and quick save need separate decisions

Setting	Meaning	Practical choice
Enable Game Autosaving	Allows the game’s automatic-save behavior.	Decide where a checkpoint may safely occur.
Player may disable autosaving	Exposes the choice through game options.	Honor it if you offer it.
Autosave Slot 0	Uses a dedicated autosave file.	Keeps autosave separate from manual slots 1–8.
A positive Autosave Slot	Uses that ordinary numbered slot for autosaves.	Reserve it in the menu; do not accidentally offer it as an ordinary manual destination.
Quick Save Slot	The fixed destination used by the quick-save route.	Choose it deliberately; quick save can replace that slot.

The widget’s Include Autosave option is only a listing choice. It does not enable or disable automatic saving. Allow Manual Autosave controls whether the widget lets the player manually save to the configured autosave slot.

If the autosave slot is a positive number already included in your manual range, it can still appear through that range. For the cleanest separation, use the dedicated slot 0 and manual slots beginning at 1.

> Do not assign quick save to slot 0 unless overwriting that autosave is intentional. These are real file destinations, not decorative labels.

14. Make a first save-and-load window

1. Add a GUI named Save and Load with ID save_lab. Keep it hidden at start.
2. Add a Save Game Widget with ID save_slots. The example uses X 32, Y 90, Width 786 and Height 500 inside a 1120 × 660 window.
3. Set 1 visible column, 3 visible rows and 3 visible slots. Use Slot Width 700, Slot Height 104, Widget Padding 12 and Vertical Gap 10.
4. Set Mode to save-load, Start Slot 1, End Slot 8, Include Autosave on and Show Empty Slots on.
5. Set Click to select and the other gestures to none for the first test. Add separate command Buttons on the right.

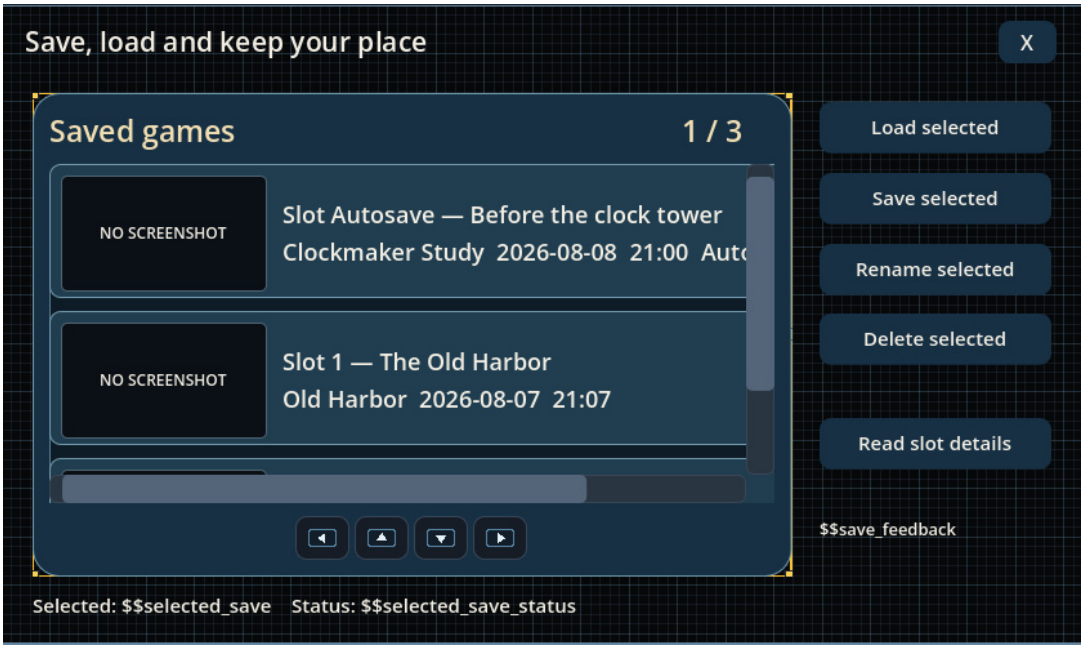

Figure 29. The widget and the command buttons are separate elements. Three visible cards do not limit the game to three saves.

The designer uses sample save entries, not your live save files. “The Old Harbor” and other sample names are placeholders for judging a layout. Play mode reads the actual saved games.

Map the menus and open them deliberately

In Global Settings → General, scroll to the custom system interface assignments. Set Save Screen to Save and Load(save_lab). Set Load Screen to the same GUI if you want a combined browser. Apply Changes and save the project.

A HUD Button can Show GUI save_lab. Hide overlapping HUD elements first if necessary. A Close Button can Hide GUI save_lab and show the HUD again.

You want	Recommended starting setup
One combined menu	One save-load widget, clearly labeled Save selected and Load selected buttons.
Separate save and load windows	Two GUIs with distinct IDs. Use save mode in one and load mode in the other. Map each system role to the matching GUI.
A display-only save gallery	Browse mode, with rename and delete permissions also disabled.

Save Game and Load Game actions can open the assigned system menus. Save Game Slot and Load Game Slot address a fixed numbered slot directly. Their names are similar, but their purpose and protection rules differ.

A combined browser is easiest to learn when a row only selects. Keep saving and loading on separate, clearly labeled buttons until the player flow is settled.

Add the four command buttons

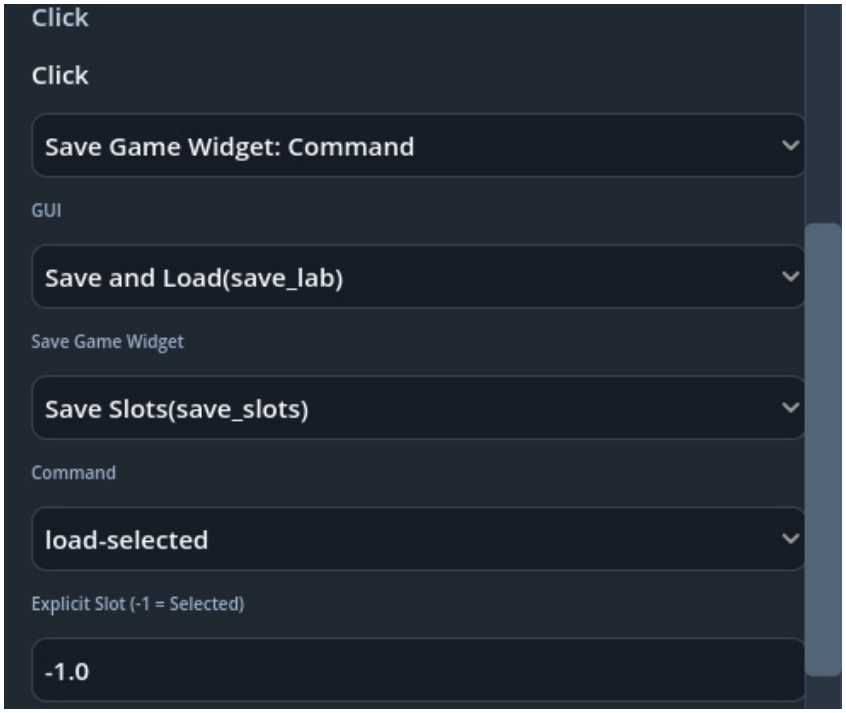

Figure 30. The Load selected Button uses Save Game Widget: Command, targeting one GUI and one element.

Button caption	Command	GUI / element	Explicit Slot
Load selected	load-selected	save_lab / save_slots	-1
Save selected	save-selected	save_lab / save_slots	-1
Rename selected	rename-selected	save_lab / save_slots	-1
Delete selected	delete-selected	save_lab / save_slots	-1

1. Select the Button and open Events.
2. For Click, choose Save Game Widget: Command.
3. Set GUI to save_lab and Save Game Widget to save_slots.
4. Choose the command from the table. Keep Explicit Slot at -1 for the selected-slot versions.
5. Repeat for the other buttons. Give Delete a clear destructive label rather than an ambiguous icon.

> Do not add a second Save Game Slot action to the same button. The widget command already performs the operation, including the widget’s prompts and permissions.

15. Choose mode, range and sort order

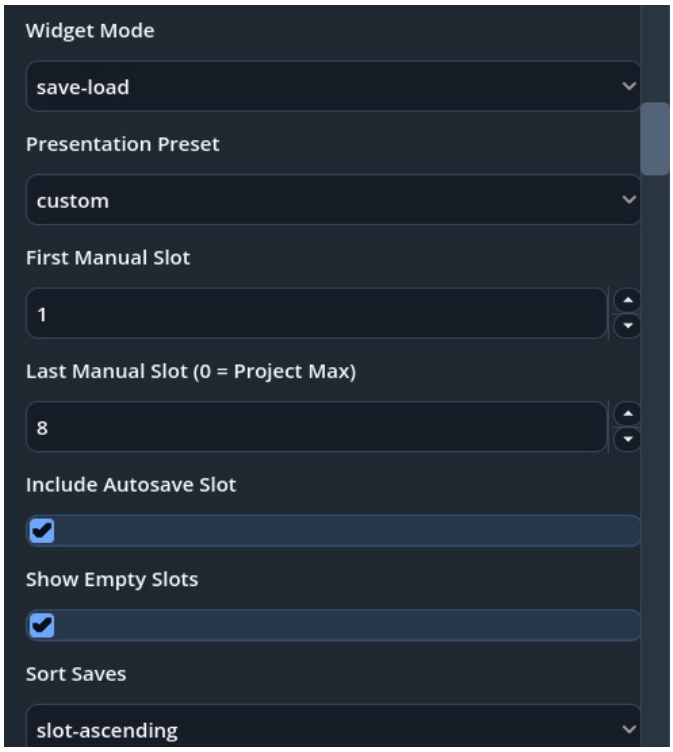

Figure 31. Start/End Slot, Include Autosave, empty-slot visibility and sorting change the browser’s contents.

Mode	Save allowed	Load allowed
load	No	Yes
save	Yes	No
save-load	Yes	Yes
browse	No	No

Mode gates saving and loading through this widget. Rename and delete have separate permissions, so browse does not automatically mean a fully read-only menu. Turn those permissions off too if that is the intention.

Start Slot begins at a manual slot of at least 1. End Slot 0 means the project’s maximum. Include Autosave can add the configured autosave separately. The final range is clamped to the game’s valid slot count.

> Two widgets may show different ranges of the same save files. They do not create independent save profiles merely because they have different element IDs.

Sorting does not renumber saved games

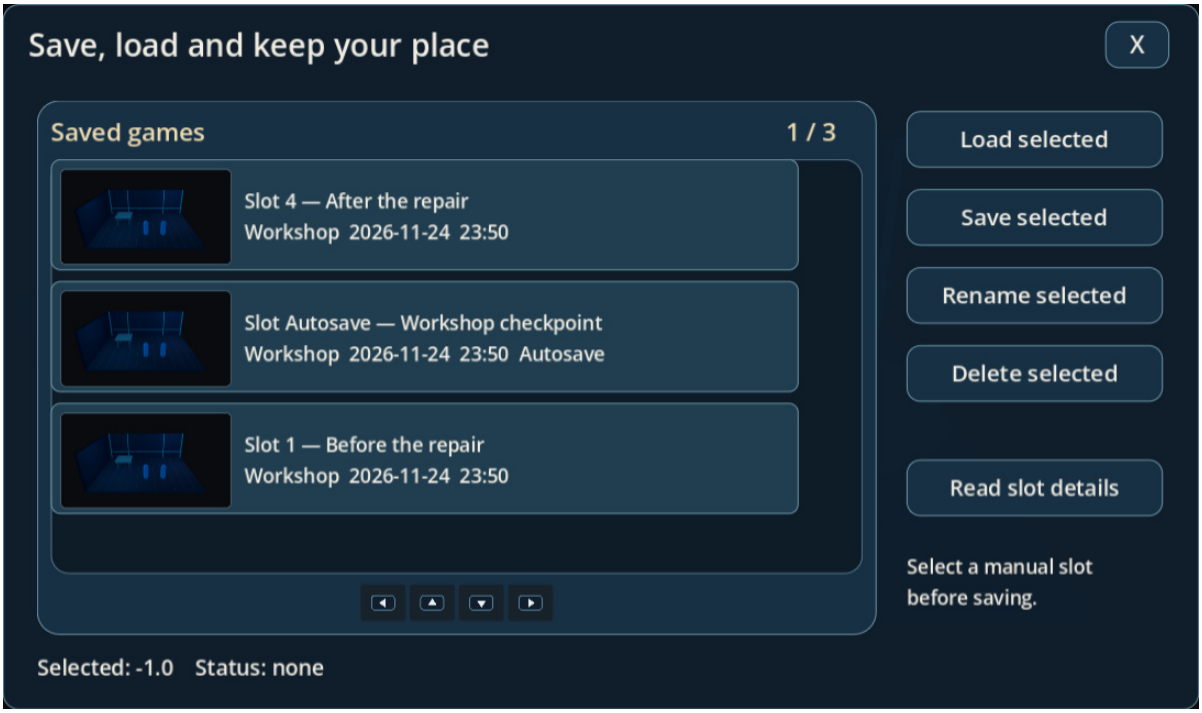

Figure 32. Newest-first changes the order of cards; the slot numbers remain the file destinations.

Sort option	What it prioritizes
slot-ascending	Lower numbered slots first.
slot-descending	Higher numbered slots first.
newest-first	Occupied saves with the newest stored save timestamp first.
oldest-first	Occupied saves with the oldest stored save timestamp first.

Date sorting places occupied saves ahead of empty destinations. Ties use slot order. Renaming a file does not create a new gameplay save timestamp, so a rename alone should not move a card to the top of a newest-first list.

Show Empty Slots is useful in a Save menu: it provides unused destinations. Hide empty slots in a Load menu if you want to show only games that can be loaded. An entirely empty list can be correct when no saved games exist.

> A selected slot keeps its number when sorting changes. Never send “row 1” to a fixed-slot save action on the assumption that the first displayed card means slot 1.

16. Arrange rows, cards or text lists

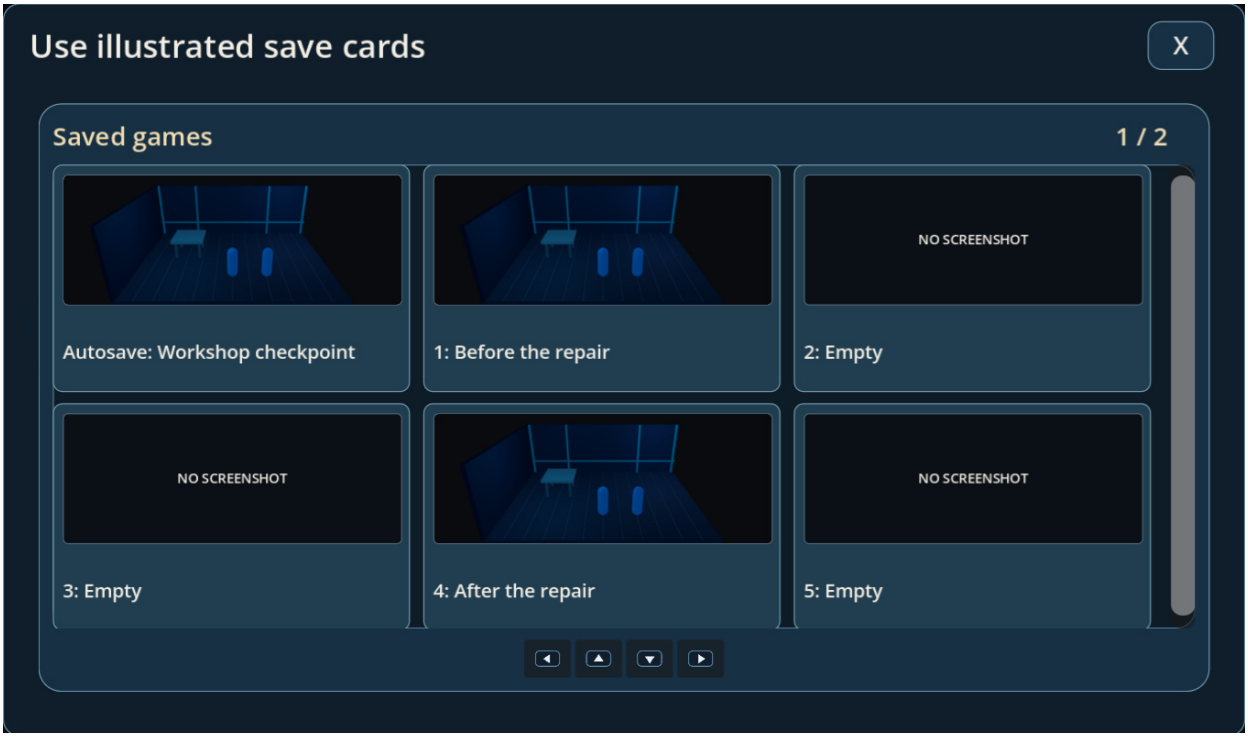

Figure 33. An illustrated grid uses three columns and two rows instead of one vertical list.

Save Game Widget inherits the inventory widget’s columns, rows, visible slots, slot dimensions, gaps, header, arrows, offsets, frame styling and image layers. The content inside each slot is different: screenshot plus save caption.

For the illustrated example, use 3 columns, 2 rows, 6 visible slots and a 1056 × 530 widget. Its cards are 322 × 205 with the screenshot above the text. A short template such as {slot}: {name} leaves room for six readable cards.

For a classic text menu, turn off screenshots, use one column and shallow rows, and choose a concise caption. Turning off screenshots alone does not reduce the authored slot height; resize the rows too.

> The save widget does not use the inventory item-icon switch as a substitute for Show Save Screenshots. Change the save-specific screenshot and text controls.

Presentation presets are starting layouts



Figure 34. The SCI-style preset creates a compact text list.

Preset	Starting arrangement
SCI text list	1 × 8, 8 visible; 540 × 38 rows; no screenshots.
AGI text list	1 × 10, 10 visible; 500 × 30 rows; no screenshots.
Zak-style list	1 × 10, 10 visible; 560 × 34 rows; no screenshots.
Lucas-style list	1 × 8, 8 visible; 580 × 42 rows; no screenshots.
Modern thumbnail grid	2 × 3, 6 visible; 290 × 170 cards; screenshot on top.
Curse-era illustrated	1 × 5, 5 visible; 620 × 92 rows; screenshot on the left.

A preset applies a collection of authored settings. It can change the widget size, cell size, screenshot layout, text template and click behavior. Review all of those after choosing one.

> Clone the GUI or use Undo before experimenting. Choosing Custom afterwards does not restore your previous values. Changing only the preset-name property at runtime does not perform the editor’s full layout merge.

17. Capture and display save pictures

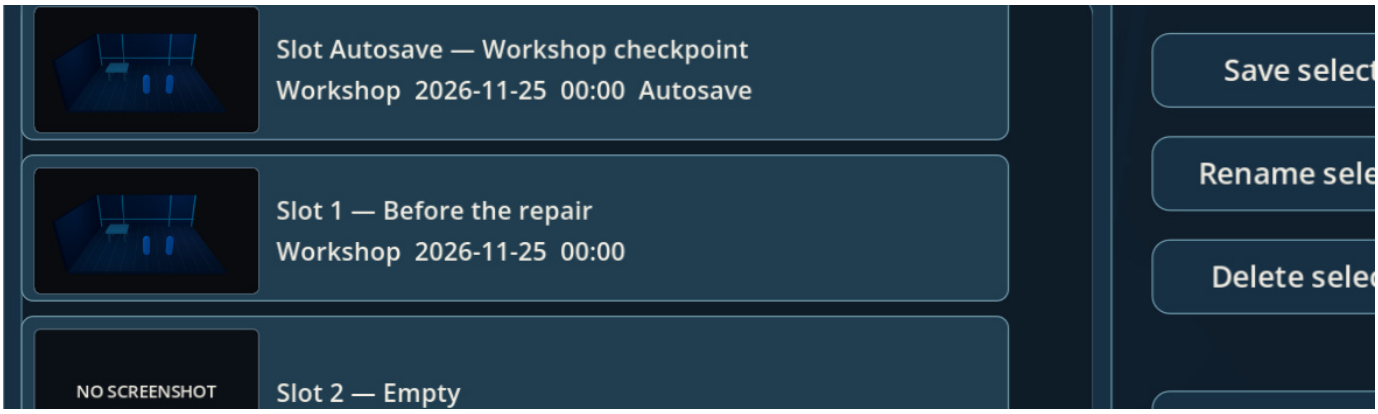

Figure 35. The occupied cards have captured workshop views. The empty card uses the placeholder area.

Setting	Where it lives	What it controls
Capture Save-Game Screenshots	Global Settings → Player Options	Whether a save captures a real image.
Screenshot Width / Height	Global Settings → Player Options	The resolution of that saved image.
Show Save Screenshots	Save widget → Type Settings	Whether this particular menu displays pictures.
Screenshot Width / Height	Save widget → Type Settings	The size of the picture area inside a card.

The same labels can refer to different sizes. Capture at 320 × 180 and display at 160 × 90 to keep a small, clear 16:9 thumbnail. Displaying it larger cannot add detail that was never captured.

> Changing capture settings does not repair earlier saves retroactively. Make a new save to test them. A headless run cannot provide the same rendered capture as the visible game.

Place and fit the picture inside the card

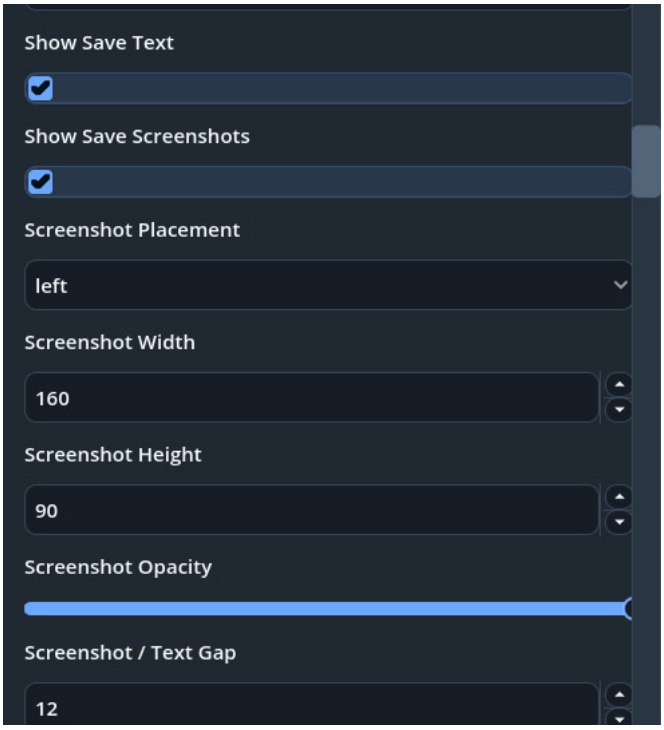

Figure 36. Placement, width, height, opacity and gap define the screenshot area inside each card.

Placement	Design implication
left / right	The caption needs enough remaining horizontal space.
top / bottom	The card must fit image height, caption height, gap and padding.
background	The picture sits behind the caption; text needs contrast.
none	No screenshot area, even if screenshot display is otherwise enabled.

Screenshot Gap separates the picture from the caption. Screenshot Opacity affects the picture region. Fit cover fills the area by cropping; contain preserves the complete picture; stretch changes its proportions to fill the exact rectangle.

The picture dimensions are constrained by the card in some placements, but the widget does not invent a larger card to suit every image and caption. Increase Slot Height yourself when moving a thumbnail above a long caption.

> The screenshot normally comes from the rendered room viewport. It is not a guarantee that every GUI overlay is included. Check the captured result in your actual game, especially for FMV or unusual presentation setups.

Empty slots, missing pictures and placeholders

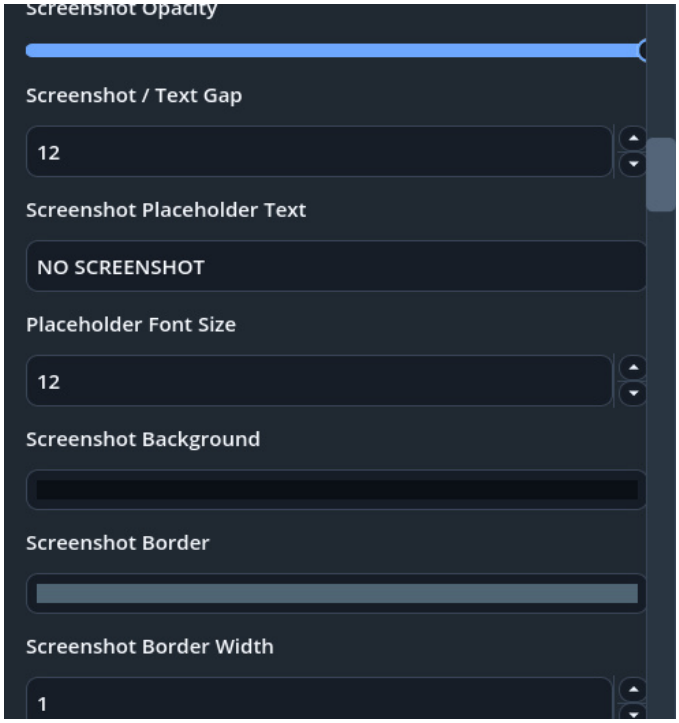

Figure 37. A placeholder can use text, a chosen image and its own border settings.

An empty slot has no saved screenshot. An occupied save can also lack a picture if capture was disabled, unavailable, or its thumbnail file is missing. These situations should not make the menu look broken.

Choose a Screenshot Placeholder Image in Images when you want a picture fallback. Otherwise, use Screenshot Placeholder Text such as NO SCREENSHOT. Give that text its own readable font size, color and background.

The placeholder is presentation, not proof that the saved game exists. Use save information or the selected-slot status for a Load button’s decision. A missing picture does not automatically mean the gameplay save is empty.

For background placement, the thumbnail is a backdrop layer; do not expect the same separate placeholder-text box used by a left or top picture holder. Use a deliberate fallback image when the background must remain illustrated.

> Keep a copy of both the saved state file and its PNG if you back up saves outside the game. Losing the PNG can leave a loadable save with a missing picture.

Screenshot artwork versus a real thumbnail

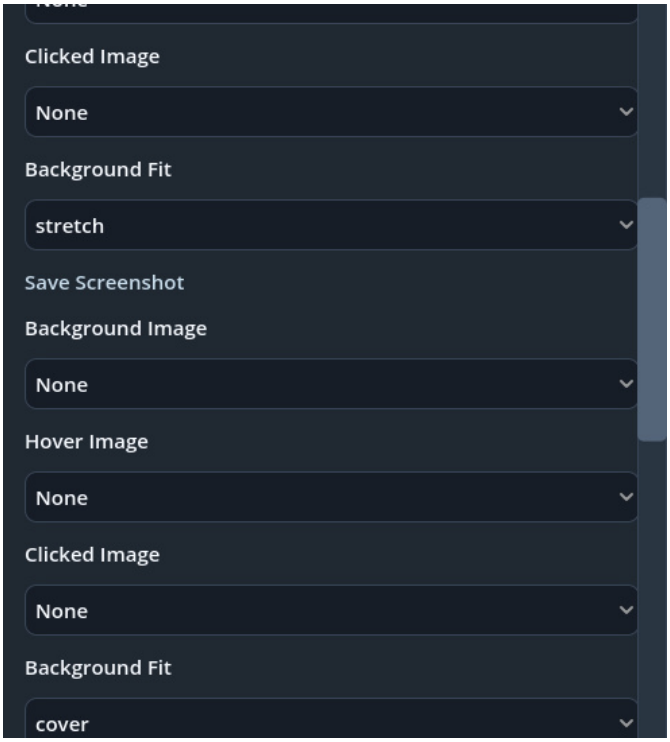

Figure 38. The screenshot region has its own normal, hover and clicked image choices.

The Save Screenshot normal background image is an override. If you assign it, that image can replace the real per-slot thumbnail. Leave it empty when each slot should show its own captured view.

Hover and Clicked images can still be assigned to the screenshot region while its normal state uses the actual thumbnail. That gives temporary feedback without making every save look identical at rest.

Screenshot radius, four corner overrides, border and opacity are independent of the slot’s frame. A square thumbnail can sit inside a rounded row, or a rounded thumbnail can sit inside a square row.

> If every slot suddenly shows the same map, inspect Save Screenshot → Background Image first. Do not immediately blame screenshot capture; you may have deliberately overridden its display.

Use a deliberate picture for the next save

Set Save Slot Image is a shared action for an authored save picture. It chooses an image asset for the save-capture path. This is useful for chapter artwork, a portrait-style save menu or a room where a live capture is unhelpful.

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    await SetSaveSlotImage({"assetId": "map_art"})
    await SaveGameSlot({
        "slot": 4,
        "saveName": "The map room"
    })
    await SetSaveSlotImage({"assetId": ""})
    return true
```

This complete example sets the picture, saves to fixed slot 4, then clears the override so later saves can use normal capture again. Replace map_art with your imported image asset ID.

This is different from the widget’s screenshot background override. Set Save Slot Image supplies the picture stored for a save. The widget’s Background Image controls how that menu displays its picture region.

> Save Game Slot writes directly to its fixed destination. This example deliberately uses that behavior; it does not ask for a name or an overwrite confirmation. Use a widget command instead when you want the player’s usual save-dialog flow.

18. Write clear save captions

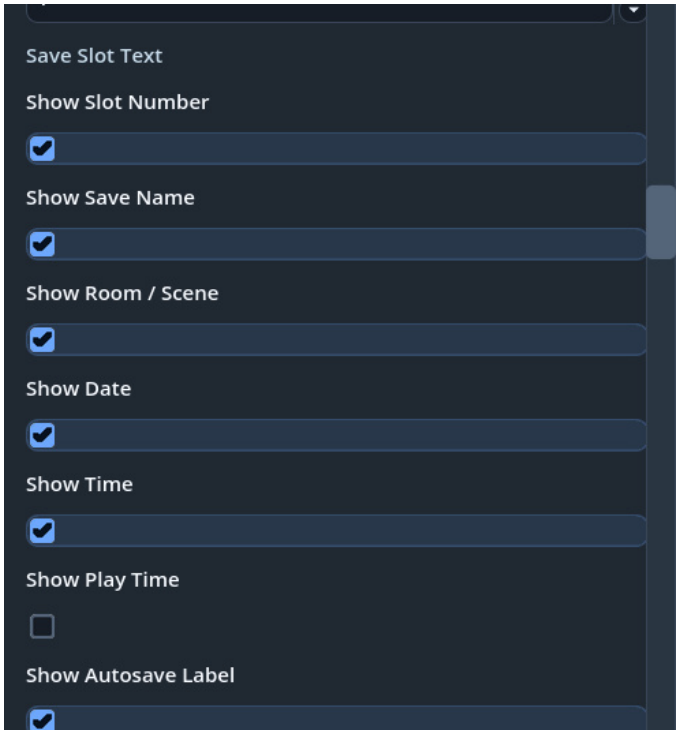

Figure 39. Generated captions can include the slot, save name, scene, date, time, play time and autosave label.

With the custom text templates left empty, the widget assembles a caption from the Show controls. A practical row uses a name on the first line and scene/date/time on the next. A tiny thumbnail card needs less text.

Show Save Text controls the caption as a whole. Show Name, Show Scene and the other flags control the generated pieces. These are save-specific settings; the inventory’s Item Text option is not the main caption switch here.

Slot Label, Autosave Label and Empty Label supply words used by the generated caption and template tokens. They are useful places to choose your game’s terminology or translated wording.

> A custom occupied or empty text template replaces the automatic assembly. Once you use a template, the template—not the separate Show Name or Show Date flags—decides whether those tokens are present.

Every save-caption token

Token	What it supplies
{slot}	The formatted slot value; this can be the autosave label.
{slotNumber}	The numerical slot, such as 0, 1 or 4.
{slotLabel}	The configured ordinary slot label, such as Slot.
{name}	The saved game’s name.
{scene}	The stored scene name.
{date}	The stored date in the selected date presentation.
{time}	The stored time in the selected time presentation.
{datetime}	Date and time together.
{playtime}	The recorded play-time text.
{autosave}	The autosave label when the slot is the autosave.
{empty}	The configured empty label.
{status}	Saved or Empty.

For occupied slots, try {slot}: {name}. For empty slots, try {slot}: {empty}. A two-line detailed caption can use {name} on the first line and {scene} | {date} | {playtime} on the next.

These are local save-caption tokens, not the full wildcard system used by ordinary Labels. Do not expect $$chapter, @PlayerName, BBCode or arbitrary script expressions to be evaluated inside a save template.

> If you need a line break, store an actual newline in the template. In advanced JSON, the escape \n inside a quoted string represents that newline. Typing a backslash and n into an ordinary one-line field may leave those two characters as text.

Date, time and default save names

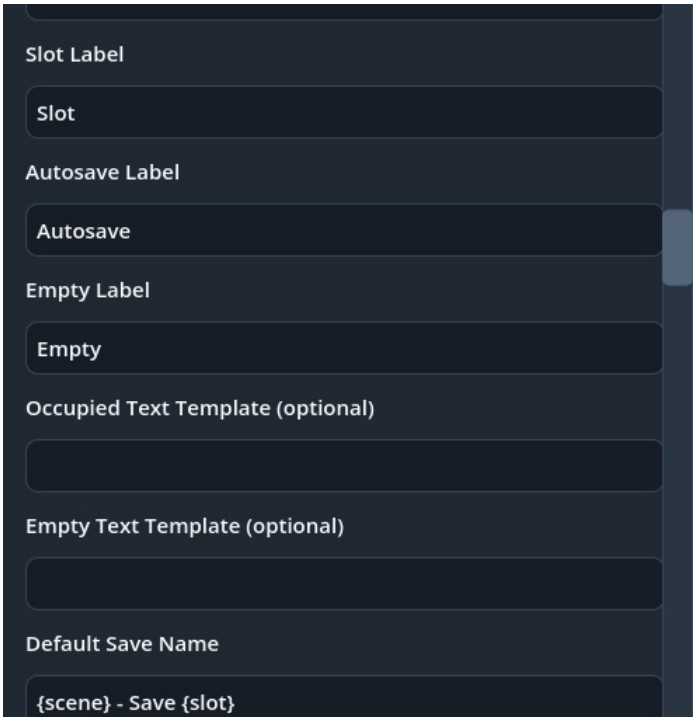

Figure 40. Text Template, Empty Text Template and Default Save Name Template have different jobs.

Field	Purpose
Save Text Template	Caption for an occupied slot.
Empty Text Template	Caption for an unused slot.
Default Save Name Template	Suggested name when the widget saves without a supplied name. Supports {slot} and {scene}.
Global Default Save Name Template	Fallback for direct/global save paths, separate from this widget’s naming preference.

Date choices include year-month-day, month/day/year and day/month/year. Time can use 24-hour or 12-hour presentation. The system option in this build should not be taken as a promise of the operating system’s full regional formatting; inspect the resulting caption.

The screenshot’s timestamp is a saved piece of information. The menu is not a live clock. A rename changes the save name, not the moment when gameplay state was captured.

> The default-name template does not support the full caption token list. Use {scene} - Save {slot}, rather than expecting {playtime} to become a suggested name.

19. Assign clicks and commands deliberately

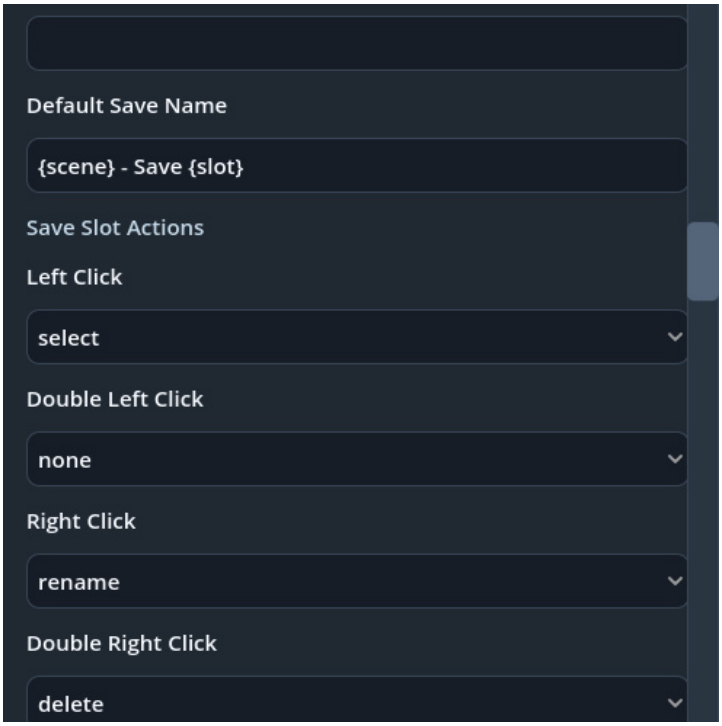

Figure 41. Click, Double Click, Right Click and Double Right Click each choose a slot action.

Slot action	Result
none	No further operation; a clicked row can still become selected.
select	Choose the row without saving or loading.
load	Load that slot if the widget allows it and a save exists.
save	Save to that slot, subject to widget rules and prompts.
rename	Name an existing save, if allowed.
delete	Delete an existing save, if allowed, with confirmation when enabled.

Use Click select and Double Click none for an easy first menu. If you later add Double Click load, check the sequence of clicks on your target platform. Do not assign a destructive first-click action and a contradictory double-click action.

> Right click should not be the only way to rename or delete when the game supports touch or controllers. Visible buttons make these operations discoverable.

The widget-command reference

Command	What it addresses
previous / next	Move by a page.
up / down	Move by a row.
refresh	Re-read the slot list and pictures.
select-first	Select the first slot in the full ordered list.
select-previous / select-next	Move selection through the ordered list, wrapping at its ends.
load-selected / save-selected	Operate on the selected slot; see the fallback rule below.
rename-selected / delete-selected	Rename or delete the selected slot.
select-slot	Select Explicit Slot.
load-slot / save-slot	Operate on Explicit Slot.
rename-slot / delete-slot	Rename or delete Explicit Slot.

For the -slot commands, supply a real numerical slot. For selected commands, leave Explicit Slot at -1. The command’s GUI and element identify which widget supplies the range, permissions, prompts and mode.

If no slot is selected, an operation on the selected slot can fall back to the first ordered slot. That may be autosave 0. It is not necessarily the first empty manual slot or the first card on the currently visible page.

> Select-first selects from the complete ordered list. Select-next does not automatically scroll the page to bring a distant selection into view in this build. Coordinate selection and paging explicitly for a controller-driven menu.

Widget commands and direct saves differ

Operation route	Uses widget mode and range?	Uses widget name/overwrite prompts?
Save Game Widget: Command	Yes	Yes, according to its settings.
Save Game Slot / Load Game Slot	No; uses the valid game-wide slot.	No. These are direct operations.
Runtime.save_game / load_game	No; direct runtime calls.	No. Your script owns the surrounding player flow.

Use widget commands for player-facing menu buttons. Use direct slots for a deliberate checkpoint, quick-save feature or scripted save operation where your game has already chosen the destination.

A load-mode widget can refuse a save-selected command while a direct Save Game Slot action still writes to a valid slot. That is expected: the widget’s mode is not a global prohibition on all saving.

Keep the target GUI and element explicit when several save widgets exist. A command with an empty target can affect more than one matching widget; that is rarely useful for a Save selected button.

> A command being dispatched does not prove an operation succeeded. A forbidden slot, an empty load slot or a canceled prompt can leave the saved state unchanged. Inspect the actual slot information when later logic needs to know the result.

20. Names, overwrite and deletion

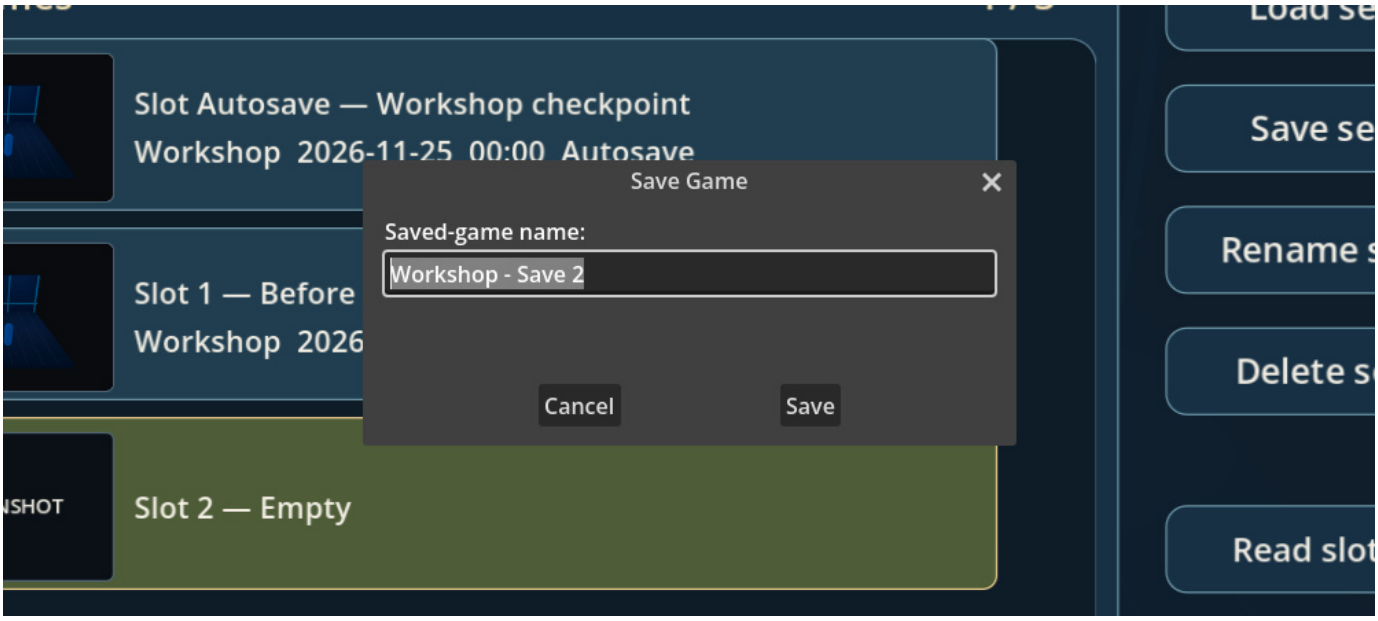

Figure 42. Saving a new slot can open a name dialog before anything is written.

Prompt for Name lets the player name the save. An empty entered name falls back to the widget's default-name template. Without a prompt, an occupied slot can preserve its existing name; a new slot receives the fallback name.

Confirm Overwrite asks before replacing an occupied save. When both options are enabled, the name step can come before the overwrite step. Canceling either step should leave the previous saved gameplay state alone.

Keep both checks on while learning. Once the flow works, decide whether a streamlined quick-save button belongs elsewhere. Do not quietly turn the main Save button into an instant overwrite.

> The name is a label inside the saved payload. Renaming does not create another slot, move the player, or save the current unsaved state.

Rename and delete an existing save

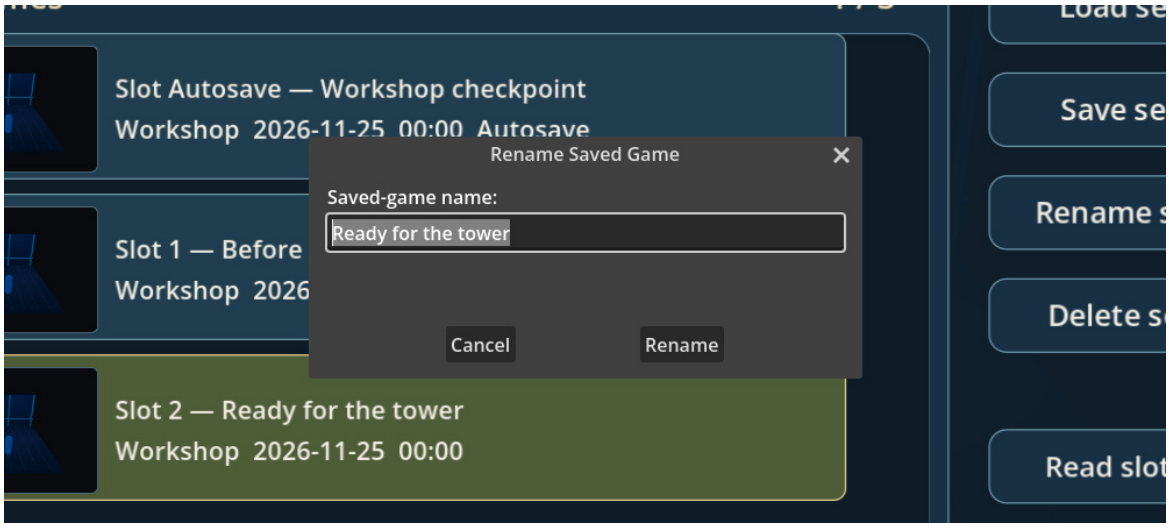

Figure 43. Rename changes the selected saved game’s name.

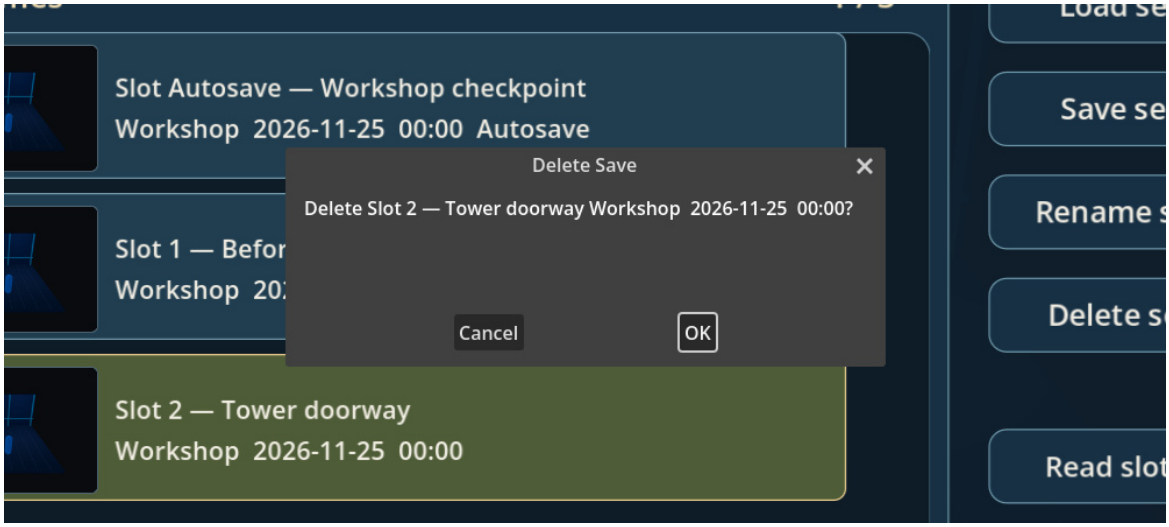

Figure 44. Delete removes the chosen saved state and its screenshot after confirmation.

Allow Rename and Allow Delete are separate permission switches. An empty slot has nothing to rename or delete. Rename preserves the stored gameplay state and timestamp. Delete removes the state file and its thumbnail; it is not an inventory-style “hide this card” operation.

> There is no automatic undo for player save deletion. Use Confirm Delete and a clear slot/name in your menu. Test deletion on throwaway saves while authoring, not the only copy of a long playthrough.

Choose when the menu closes

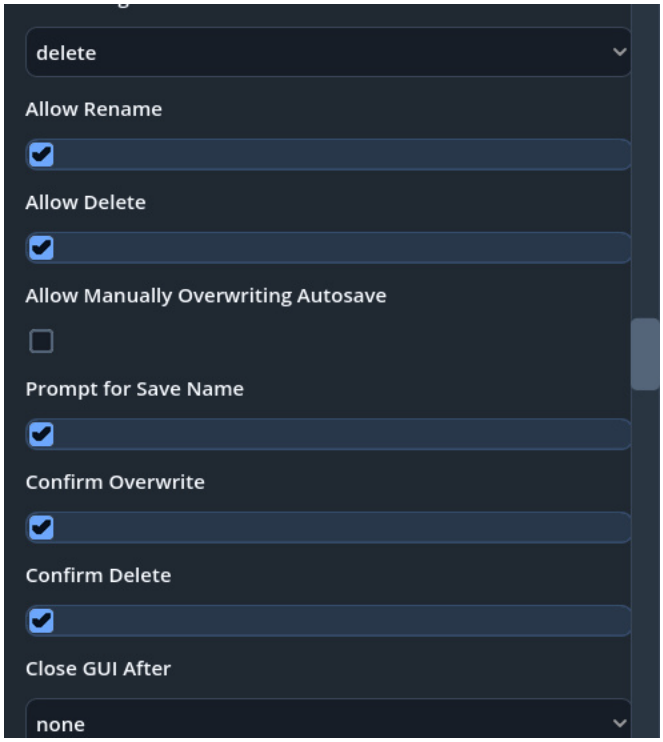

Figure 45. Prompt, confirmation and Close After settings describe the player’s save/load flow.

Close After	Menu closes after a successful
none	No automatic close.
load	Load.
save	Save.
load-save	Either load or save.
any	Load or save in this implementation; not every rename, delete or selection gesture.

A canceled name prompt or failed empty-slot load should not be mistaken for a successful operation. Test the close setting on both successful and canceled paths.

The widget closes its own GUI. If you hid a separate HUD when opening the menu, check how the HUD returns. An explicit Close button can hide the menu and show the HUD; an automatic widget close needs a coherent surrounding UI plan too.

> If you need detailed post-save behavior, verify the save result explicitly. A generic slot Click event is an input event, not a “save completed” signal.

21. Track selection and slot events

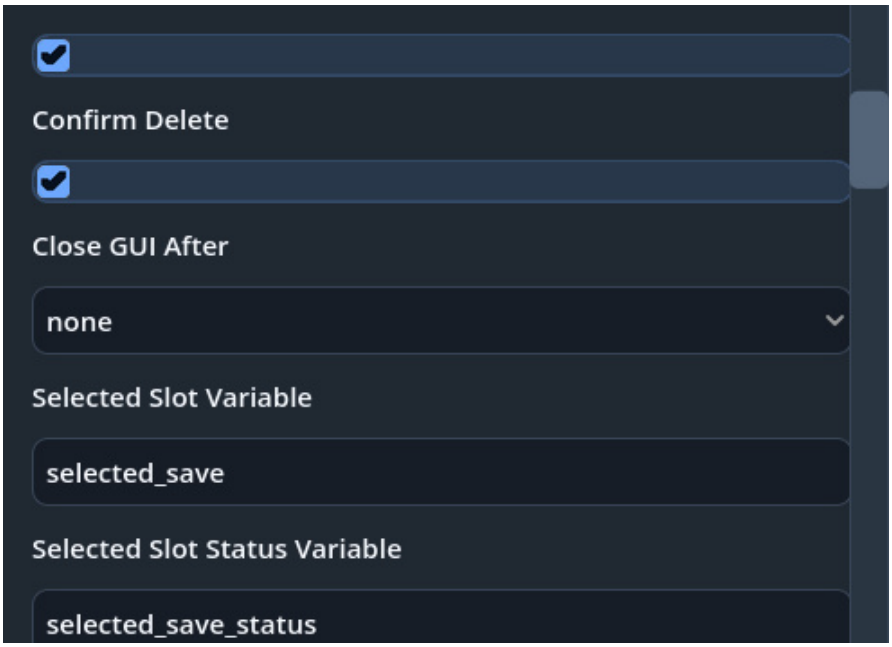

Figure 46. Selection and status variables can drive separate labels or authored checks.

1. In the Variable Designer, create Number selected_save with default -1.
2. Create String selected_save_status with default none.
3. Set the widget’s Selection Variable to selected_save and Status Variable to selected_save_status.
4. Add a Label with Selected: $$selected_save Status: $$selected_save_status.
5. Run the game and click an empty manual slot, an occupied manual slot and the autosave. Compare the displayed values.

The status variable uses empty, saved or autosave. The slot number is the actual saved-game number. A Number variable can display 1.0 in an ordinary Label; that still represents slot 1.

> These variables update on explicit selection. In this build, saving, renaming or deleting does not necessarily rewrite them to the new file status. Re-select or query current save information before a later decision.

Read live slot information after an operation

A separate Read details button in the workshop runs this GDScript record. It reads the widget’s actual selected slot, then checks the current saved file information:

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var slot_number: int = int(GetGUIElementProperty(
        "save_lab", "save_slots", "selected_slot", -1))
    if slot_number < 0:
        return false
    var info: Dictionary = Runtime.call(
        "get_save_slot_info", slot_number)
    var exists: bool = bool(info.get("exists", false))
    await SetVariable({
        "name": "selected_save_status",
        "value": "saved" if exists else "empty"
    })
    var caption: String = "This slot is empty."
    if exists:
        caption = "Saved: " + str(info.get("name", ""))
    await SetVariable({
        "name": "save_feedback", "value": caption
    })
    return exists
```

Assign this record to a Button and show $$save_feedback in a Label. This version deliberately distinguishes file existence from the earlier selection status. Add an autosave-specific caption if your menu needs one.

selected_slot is a readable widget property in this build. The fallback -1 means there is no selected slot. A command can still apply its first-slot fallback, so use your own guard when selection must be mandatory.

What a save-slot event actually reports

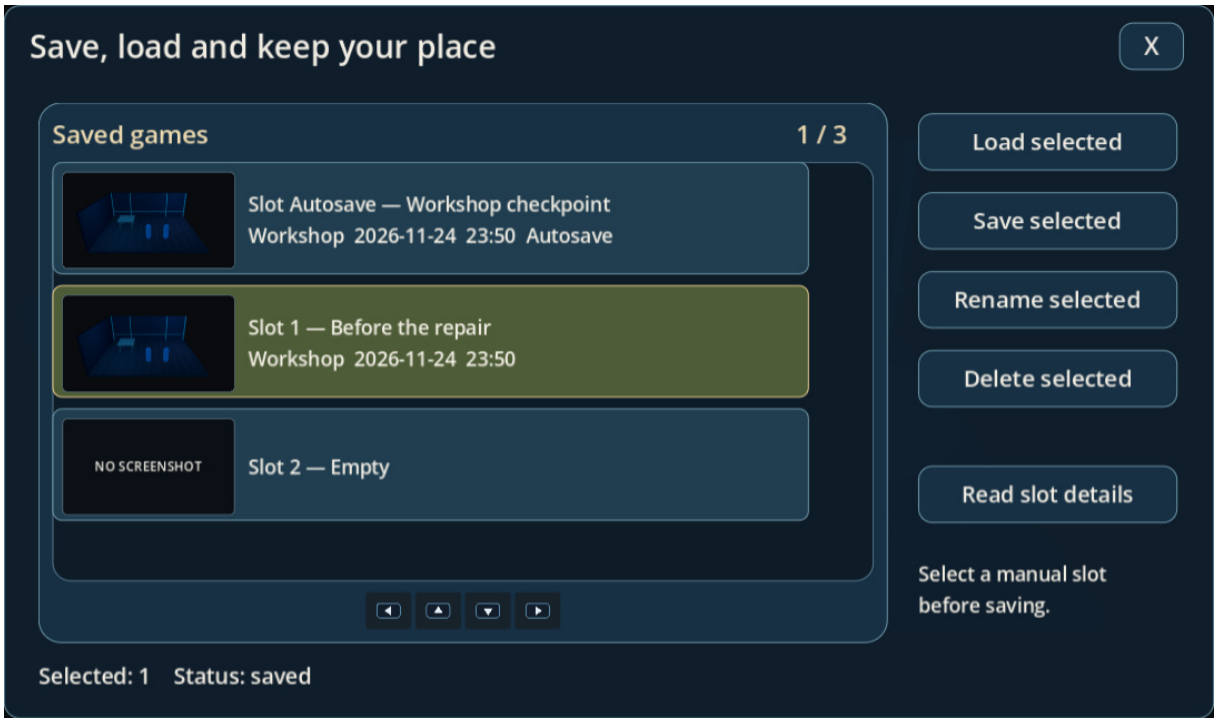

Figure 47. The save widget exposes the clicked slot and its current information to its event action.

A Save Game Widget connects its internal slot events to the GUI event system. The event context includes the slot number and information about the save. This is different from relying on generic Inventory Grid events for individual item cells.

The event is forwarded before the widget performs its built-in selection or save/load/rename/delete action. It reports the slot as it was when the input arrived. It is not an operation-completed callback.

Hover can report information without selecting. Click can report the pre-action data and then select or act. Avoid giving the same input both a custom destructive event action and a conflicting built-in slot action.

> The event’s saveStatus is saved or empty. Autosave is reported separately through saveIsAutosave. The Selection Variable’s status vocabulary also has autosave; do not mix the two without checking which source you are reading.

Use the event context in a small script

```
extends "res://scripts/runtime/script_base.gd"

func run(event_context: Dictionary = {}) -> Variant:
    var slot_number: int = int(
        event_context.get("saveSlot", -1))
    if slot_number < 0:
        return false
    var caption: String = "Slot " + str(slot_number)
    if bool(event_context.get("saveExists", false)):
        caption += ": " + str(event_context.get("saveName", ""))
    else:
        caption += ": empty"
    await SetVariable({
        "name": "save_feedback", "value": caption
    })
    return true
```

1. Create a GDScript record with this source and apply it.
2. Select the Save Game Widget, open Events and assign the record to Click using the script-running behavior.
3. Keep the built-in Click slot action at select while testing.
4. Show $$save_feedback in a separate Label.
5. Click several rows. The Label should describe each clicked slot without performing a save.

The context belongs to this input. If the subsequent built-in action creates a save, this script has already run. Use a later explicit file query when you need the state after completion.

22. Use the same operations in scripts

```
extends "res://scripts/runtime/script_base.gd"

func run(_context: Dictionary = {}) -> Variant:
    var slot_number: int = int(GetGUIElementProperty(
        "save_lab", "save_slots", "selected_slot", -1))
    if slot_number < 1:
        await SetVariable({
            "name": "save_feedback",
            "value": "Choose a manual slot first."
        })
        return false
    await SaveGameWidgetCommand({
        "guiId": "save_lab",
        "elementId": "save_slots",
        "command": "save-selected",
        "slot": -1
    })
    return true
```

This Button script adds a guard before dispatching the normal widget command. With dedicated autosave 0, a manual choice must be at least 1. The widget still owns naming, overwrite confirmation and its mode/range rules.

The final true means the script dispatched the command. It does not mean the player finished the name dialog or accepted an overwrite. Do not award an achievement simply because this record returned true.

> If you reserve a positive slot for autosave, adjust the guard to exclude that number too. The simple less-than-1 check only matches this workshop’s slot policy.

Build the equivalent visual node flow



Figure 48. A visual script calls Save Game Widget: Command with the same GUI, element, command and slot fields.

1. Create a visual script named Load the Selected Save.
2. Keep its interaction start node and add Save Game Widget: Command.
3. Connect the start flow to the new node.
4. Set GUI save_lab, Save Game Widget save_slots, Command load-selected and Explicit Slot -1.
5. Apply the script. Assign it to a Button with the script-running event behavior, then test a real saved slot.

An action list can contain the same shared action without a separate visual script. Use the form that keeps the behavior easiest to read. A short one-command Button rarely needs a large visual graph.

> A visual node does not change the command’s meaning. Loading an empty slot still cannot restore a game, and load mode still governs widget-command loading.

23. Save state, change it, then restore it

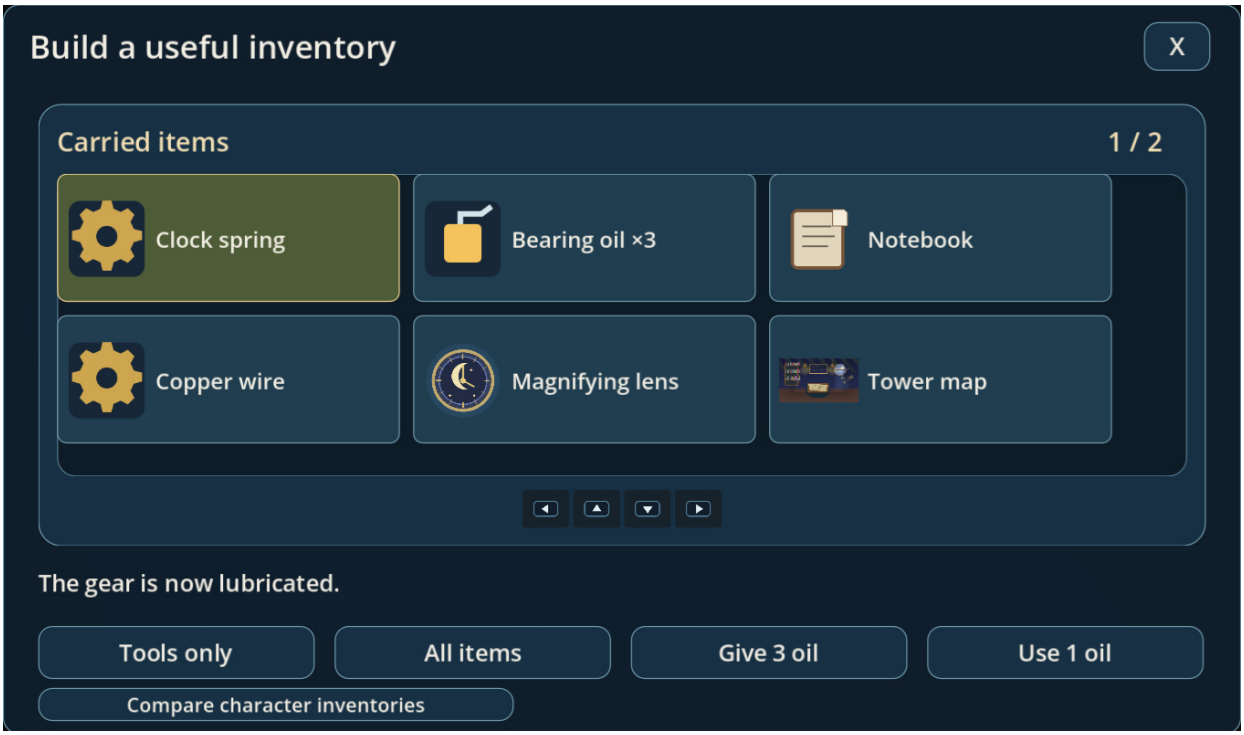

Figure 49. Loading the earlier save restores the clock spring that was removed later in the test.

1. Start a fresh game with a clock spring in the inventory and chapter set to 1.
2. Save to a known manual slot with a descriptive name such as Before the repair.
3. Remove the spring, change chapter to 2 and move the character.
4. Load the earlier slot.
5. Check the inventory, chapter value, scene and character position. Do not judge success only by the save row’s thumbnail.

The save payload contains much more than an image: variables, carried inventories and order, character states, scene information, quest state and other supported runtime state. A screenshot is only the menu’s visual reminder.

> Arbitrary custom Nodes, a temporary open dialog, or every detail of GUI selection are not automatically a general serialization system. Store important story state in supported variables and runtime systems, then test it after a fresh launch.

Where saved games belong

Saved games are separate from the authoring project. In this build, the runtime uses a project-specific folder under user://pca3d/. Godot resolves user:// for the platform and application. Do not assume the same literal operating-system folder on Windows, macOS and Linux.

The project storage identity comes from a sanitized project ID, with the title as a fallback. Keep the project ID stable across releases when the same game should find its existing saves. A new ID can make the menu look empty because it is looking in a different project folder.

Change	What to check
Rename only the visible game title	A stable explicit project ID should remain the storage identity.
Clone the game into a different project	Give the clone its own ID if it should not share saves.
Run in the editor, then an exported game	Check the actual exported application profile; do not assume it shares every editor test save.
Move a save manually	Keep its state file and thumbnail together, and test compatibility.

The dedicated slot 0 uses autosave.json and its associated picture. Manual slots use their numbered paths. Use the runtime’s slot information instead of constructing filenames inside ordinary menu logic.

> Local saving is not cloud synchronization. A save does not automatically appear on another computer or in a store cloud service merely because both copies have the same game title.

24. Build separate Save and Load screens

Choice	Save screen	Load screen
Mode	save	load
Show empty slots	On	Usually off
Single click	select	select
Double click	none, initially	load, if desired
Main button	save-selected	load-selected
Close after	save or none	load
Delete / rename	Your deliberate permission choice	Your deliberate permission choice

1. Clone the working combined GUI rather than rebuilding its styling from scratch.
2. Give the GUI and every element distinct IDs. Update every Button’s target GUI and element.
3. Apply the Save and Load settings from the table.
4. Map Save Screen and Load Screen in Global Settings to the correct GUIs.
5. Test both menus after creating, renaming and deleting a slot in the other one.

Both screens read the same project’s save files. A rename on the Save screen should be visible when the Load screen refreshes. Do not maintain a separate hard-coded array of save names that can fall out of sync.

> Cloned Buttons can still target the old element if you do not update them. A correct-looking Load menu with commands aimed at the Save widget is a particularly confusing error.

A restrained checkpoint and Continue flow

A checkpoint should preserve a safe, intentional state. Put it after the puzzle’s important variables, inventory changes and scene transition have settled, rather than halfway through an action sequence.

Part	One practical design
Checkpoint destination	Dedicated slot 0, clearly labeled Autosave.
Manual save menu	Slots 1–8; Allow Manual Autosave off.
Continue button	Check that slot 0 exists, then Load Game Slot 0.
No checkpoint yet	Disable Continue or explain that no saved game exists.
Manual loading	A separate browser lets the player choose another occupied slot.

A direct Save Game Slot 0 action can be used for an authored checkpoint even when you are not depending on automatic room-save behavior. Decide whether your own checkpoint logic should respect the player’s autosave option; do not silently contradict the choice your options menu advertises.

When loading from a title menu or another gameplay state, test the whole transition into the saved room. A Continue label alone is not a complete load flow.

> A direct checkpoint save has no widget overwrite prompt. That is usually appropriate for a clearly identified checkpoint slot, but it should never accidentally target the player’s manual archive slot.

Match the menu’s artwork to its purpose



Figure 50. Root, grid, slot and screenshot regions can each have their own appearance.

Use a quiet outer frame, a slightly different grid surface and stronger card outlines. Save pictures already contain visual detail, so bright patterned card artwork can compete with them.

For a text-only classic menu, turn screenshots off, use a shallow row and a short name template. For a chapter gallery, use a few larger cards and deliberate saved artwork. For a utility load browser, favor readable names, dates and a clear selected border.

Change corners per layer. The screenshot might be square, the card softly rounded and the entire browser more rounded. There is no reason to force the same radius on all three.

> Use temporary hover/pressed art to signal input. Use Selected Background and Selected Border for the current choice. A permanently selected card should not depend on an image intended only for a held press.

25. Test the menus before release

A menu is ready when it works with awkward states, not just the tidy state in its first screenshot. Keep a short checklist and use disposable saves while you develop it.

Inventory test	Check
Nothing carried	Empty text or intentional blank cells; no stale selected icon.
Exactly one page	No misleading extra page indicator.
One item more than capacity	The remaining item can be reached.
Long names / large quantities	No unreadable clipping or overlapping icons.
Maximum stack	Adding more obeys the cap; removing the final copy removes its slot.
Combination success and failure	No free result, duplicate reward or lost ingredients on a forbidden path.
Two playable characters	Correct bag, restrictions and transfer destination.
Press, release and drag outside	Pressed art resets; deliberate selection remains clear.

Repeat the inventory tests after a save and load. Reorder a few items before saving and check the restored order too. If a puzzle removes a world pickup, verify the pickup remains gone after loading.

> Run these checks in the exported game as well as the editor. The player’s application profile, window size and input device are part of the test.

A save-menu test you can repeat

Save test	Expected behavior
Fresh profile; no files	Save shows empty destinations. Load shows an honest empty list or disables impossible operations.
New manual save	Name accepted; correct slot occupied; picture and metadata appear.
Cancel new-save name	No new file.
Overwrite then cancel	Earlier game state remains loadable.
Rename	Only the save name changes; stored game state and timestamp remain.
Delete then cancel	The save remains.
Delete then confirm	State and thumbnail are removed.
Empty-slot load	No pretend success or unwanted scene reset.
Autosave selected for manual save	Refused when Allow Manual Autosave is off.
Load an earlier state	Inventory, variables, position and supported puzzle state are restored.

After the ordinary tests, close the game application completely and launch it again. Load a saved game from the new session. That catches mistakes hidden by variables left in memory during one test run.

> Keep a small set of representative saves from earlier public versions. Before releasing an update, test them with the new build rather than assuming all saved-state changes are backward compatible.

26. Inventory troubleshooting

Symptom	Check in this order
The bag is empty	Was the item actually given? Correct owner? Per-character or shared policy? Type filter? Carrier restriction?
The designer shows an item but Play does not	Designer starting-state sample versus the runtime’s actual carried inventory.
A new item definition never appears	Definitions are catalogue entries. Add it to Initial Inventory or give it during gameplay.
All cells show the same key picture	You assigned the key art as Slot Background rather than the item’s icon.
The item appears twice in the intended starting amount	Check whether a stackable item is listed in both character and project initial inventories.
A tool filter hides everything	Use Types on item definitions. Category and tags are not the widget’s type source.
The character cannot receive an item	Can Be Carried By, valid item ID, stack limit and destination character.

Reset one thing at a time. For a missing item, first remove the filter and use Current player. Then give one known valid item explicitly. Once it appears, restore the intended filter and restrictions.

> Do not solve a logic problem by making the GUI enormous. An item that is not owned will still be absent in a larger grid.

When clicks, recipes or selection seem wrong

Symptom	Likely cause and useful check
Clicking an item does not run Look	Interaction mode, current verb, Use Selected Verb, or an already selected different item.
“Those items do not interact” during a Look test	A different inventory item is still selected; the next click is an item-on-item attempt.
Modal inventory will not select normally	Map this GUI as Inventory Screen; other modal windows can block the selection path.
Assistant’s bag does not let Builder use its radio	The non-active owner view is read-only for built-in item interaction. Transfer explicitly if appropriate.
A combination does nothing	For dragging: Allow Combine. For either route: target reference, recipe action, ingredients and result definition.
Ingredients vanish but no result appears	Result missing, restricted or already at its maximum; guard those conditions before removal.
Consume On Use does not remove an ingredient	Use explicit Remove Inventory on the successful action path in this build.

Put a simple Set Variable or Show Message action on one item’s Look interaction before testing a large puzzle script. That tells you whether input reached the item behavior at all.

> Do not attach the same reward to the item’s recipe and a generic GUI click. One player attempt should have one clear owner for the game-changing action.

Layout and runtime-refresh limits

Symptom	Useful correction
Unexpected scrollbar inside the page	Reduce slot size/gaps/padding, hide unneeded header or arrows, or enlarge the outer element.
A corner change affects the wrong rectangle	Identify GUI Window, widget root, grid, slot and screenshot separately.
An icon jumps on hover	Match image canvas sizes, alignment and transparent margins.
Pressed picture seems stuck	Release, move out and back in; distinguish normal art from selected background. Test actual input, focus loss and event overrides.
External paging fails after a runtime filter/style change	The widget may have been rebuilt while the external navigation cache still refers to its older instance.
Errors or a crash follow a property change plus an item action	In 0.86.3.4, a stale inventory widget reference can also affect refresh. Avoid that runtime-rebuild sequence in a shipped menu until corrected.

For stable filter tabs in this reference build, author separate filtered grids or GUIs and switch their visibility. Each grid keeps its filter as authored data. Test that complete arrangement with item actions and saving; do not use a runtime property-changing experiment as your release checklist.

The guide’s item, transfer, recipe and save/load tests were also run with a fresh, unchanged widget layout. This isolates gameplay behavior from the dynamic-rebuild limitation.

> A guide cannot turn an engine limitation into a feature. If your menu depends on rebuilding live inventory widgets, verify that path in the exact build you distribute.

27. Save-menu troubleshooting

Symptom	Check
No slots appear	Mode, Start/End range, project Save Slots limit and Show Empty Slots. A load-only browser can correctly be empty.
Sample demo saves appear in the designer	That is a layout sample. Enter Play to see your actual files.
Saving refuses the selected row	Load/browse mode, autosave protection, out-of-range number or an unfinished dialog.
Save selected seems to target autosave	No explicit selection; first-slot fallback can choose slot 0. Guard selection first.
Load selected does nothing	No file, wrong target GUI/element, save-only/browse mode, or the wrong selected slot.
A direct save ignores the menu’s restrictions	Direct Save Game Slot is not the widget-command route.
Two save prompts appear	Check for duplicated event actions or a built-in save gesture plus a custom save action on the same input.

Give the Load, Save, Rename and Delete buttons one job each. Set row Click to select while debugging. Once that simple arrangement works, add one gesture shortcut at a time.

> If a dialog or modal-window error appears, close the extra prompt and simplify the event chain. Do not stack two independent confirmation paths on the same click.

Pictures, captions and stale selection

Symptom	Check
An occupied save says NO SCREENSHOT	Capture setting when it was created, available renderer, missing PNG, display setting and placeholder behavior.
Every save displays the same picture	Save Screenshot Background Image overrides the actual thumbnail.
The thumbnail is cropped	Use contain instead of cover, or match capture and display aspect ratios.
The caption ignores Show Date or Show Scene	A custom text template overrides automatic caption assembly.
{playtime} stays literal in a default save name	The default-name template supports only {slot} and {scene}.
$$chapter stays literal in a save card	Save templates have their own local brace tokens. Use a separate ordinary Label for normal game wildcards.
Status still says empty after saving	Selection variables reflect the earlier selection; re-select or query get_save_slot_info.
A selected card is off-screen	Selection navigation does not automatically adjust paging. Move the page deliberately.

Create a new test save after changing screenshot capture settings. Reusing an old pictureless save to test a new capture option tells you very little.

> A save row’s screenshot is a reminder of the state that was stored. It is not a live preview camera into that saved game.

Missing saves and release compatibility

Symptom	Check
All saves seem gone after a project rename or clone	Project storage ID and application profile. You may be reading a different save directory.
Editor saves do not appear in the exported app	The exported application’s user profile can differ. Test with real exported saves.
A renamed save is still in the same date order	Rename preserves the stored save timestamp.
Deleted save does not return after a refresh	Deletion is a real file operation, not a temporary view filter.
An older save loads strangely after an update	Changed IDs, removed content or altered saved-state structure; test compatibility deliberately.
Only a screenshot was copied to another machine	The image is not the gameplay save. Preserve the actual state file too.

Keep record IDs stable when saved games depend on them. Renaming the display label of a door is easier to support than deleting its old ID and creating a new object with a similar name.

Use a release test on each desktop platform you support. The guide’s instructions are platform-neutral, but file permissions, profiles, rendering and input still need testing in the application your player receives.

> Do not delete a player’s entire save folder as the first troubleshooting step. Check the selected project identity and make a backup before any manual file cleanup.

28. Shared layout property reference

These are authored element property keys. Use the typed fields where available. Advanced keys belong inside the existing element object in Raw JSON, not in a replacement one-line object.

Property	Purpose
columns / rows	Grid dimensions, each at least one.
pageSize	Visible slots, capped by columns × rows.
slotWidth / slotHeight	Fixed authored cell size.
gridGapX / gridGapY	Horizontal and vertical space between cells.
padding	Outer widget content padding.
slotPadding	Inner cell padding; useful as an advanced property.
gridOffsetX / gridOffsetY	Move the grid holder.
slotOffsetX / slotOffsetY	Move the cells inside their panel.
showHeader / title	Show and name the widget header.
scrollMode / navPlacement	Input route and position of navigation controls.
emptyText	Inventory empty-state wording.
showEmptySlots	Inventory page fillers; save widgets force this internal filler off and use saveShowEmptySlots instead.

When editing Raw JSON: keep the ID, type, layout and all other needed properties; use double quotes; separate properties with commas; then choose Validate and Apply Element JSON (Advanced). Save Project afterwards.

Text, ownership and item behavior keys

Property or family	Purpose
inventoryOwnerMode	current-player or specific-character.
inventoryCharacterId	Character for the specific-owner view.
inventoryTypes	OR filter over item Types; empty means all.
allowCombine	Permit drag-to-combine; selected-item clicks can still use global recipes.
allowVerbActions	Forward item clicks to the selected-verb path.
allowDragReorder	Allow player item reordering.
gridShowText / gridShowIcon	Inventory caption and icon display.
gridIconPlacement / iconSize	Item icon layout; iconSize can be advanced JSON.
animateIcons / iconFps	Widget defaults, subject to item overrides.
showQuantity	Append quantity for a carried stack larger than one.
title... style properties	Header font, color, opacity and alignment.
grid... style properties	Normal item/caption font, color, opacity and alignment.
gridHover... / gridClick... / gridRightClick...	Text-state overrides. Double clicks use the corresponding click/right-click family.

The style families include font selection, system font, size, weight/bold, italic, underline, strikethrough, text color, opacity and alignment where exposed. Keep normal and interaction states consistent before trying dramatic differences.

> Do not infer a new property by attaching any word to a prefix. Use the inspector, autocomplete/property catalog or the exact supported keys shown in this guide.

Independent corner and image keys

Layer	Common radius	Normal image
Widget	radius	backgroundImageAssetId
Grid	gridRadius	gridBackgroundAssetId
Slot	slotRadius	slotBackgroundImageAssetId
Save screenshot	saveScreenshotRadius	saveScreenshotBackgroundImageAssetId

Each radius family adds TopLeft, TopRight, BottomRight and BottomLeft. For example, gridRadiusBottomRight changes only the grid’s lower-right curve. A corner value of -1 inherits that family’s common radius.

Layer	Hover image	Clicked / pressed image
Widget	hoverImageAssetId	pressedImageAssetId
Grid	gridHoverImageAssetId	gridPressedImageAssetId
Slot	slotHoverImageAssetId	slotPressedImageAssetId
Screenshot	saveScreenshotHoverImageAssetId	saveScreenshotPressedImageAssetId

Fit keys are backgroundFit, gridBackgroundFit, slotBackgroundFit and saveScreenshotFit. Use stretch, contain or cover. The item’s own iconAssetId and icon-state images belong to its inventory definition, outside these repeated GUI background layers.

> The screenshot normal-image override hides the actual thumbnail at rest. Leave it empty for live per-slot save pictures.

Save filtering and input keys

Property	Values or purpose
saveWidgetMode	load, save, save-load, browse.
saveStartSlot / saveEndSlot	Manual range; End 0 uses the project maximum.
saveIncludeAutosave	Include the configured autosave through its dedicated listing route.
saveShowEmptySlots	Show unused save destinations.
saveSortMode	slot-ascending, slot-descending, newest-first, oldest-first.
saveClickAction / saveDoubleClickAction	none, select, load, save, rename or delete.
saveRightClickAction / saveDoubleRightClickAction	The corresponding right-button slot operation.
saveAllowRename / saveAllowDelete	Separate permissions.
saveAllowManualAutosave	Allow widget manual saves to the configured autosave slot.
savePromptForName	Ask for a name when saving.
saveConfirmOverwrite / saveConfirmDelete	Confirm replacement and removal.
saveCloseAfterAction	none, load, save, load-save, any; successful load/save only.
saveSelectionVariable / saveStatusVariable	Runtime variables written on selection.

saveSelectedSlot is the authored selection field; selected_slot is the live property read by the script examples. A default of -1 means no explicit selection.

> The shared command action calls this widget’s rules. Direct slot save/load actions use the game-wide file path instead.

Save picture and caption keys

Property or family	Purpose
saveShowScreenshots	Display picture areas.
saveScreenshotPlacement	left, right, top, bottom, background or none.
saveScreenshotWidth / saveScreenshotHeight	Display area dimensions.
saveScreenshotGap / saveScreenshotOpacity	Image/text spacing and picture alpha.
saveScreenshotPlaceholderAssetId	Fallback image when no thumbnail exists.
saveScreenshotPlaceholderText / FontSize	Placeholder wording and text size; the font-size key has the same prefix.
saveScreenshotBackground / BorderColor / BorderWidth	Screenshot-region frame appearance, with the same saveScreenshot prefix.
saveShowText	Display the caption.
saveShowSlotNumber / Name / Scene	Generated-caption pieces, each with the saveShow prefix.
saveShowDate / Time / PlayTime / AutosaveLabel	Other generated pieces, each with the saveShow prefix.
saveTextTemplate / saveEmptyTextTemplate	Occupied and empty custom captions.
saveDefaultNameTemplate	Suggested save name; {slot} and {scene}.
saveDateFormat / saveTimeFormat	Stored date/time presentation.

The shortened family entries in this table are reading aids. For example, saveShowName is a full key; Name alone is not. Prefer the typed inspector when you are unsure of an exact spelling.

29. Action and script reference

You need to…	Use
Give or consume an item	GiveInventory / AddInventory / RemoveInventory.
Change an amount	ChangeInventoryQuantity.
Pass an item between characters	TransferInventory.
Check ownership	HasInventory(item_id, character_id, amount).
Read a quantity	InventoryCount(item_id, character_id).
Open / close a menu	ShowGUI / HideGUI.
Enable / disable a control	EnableGUIElement / DisableGUIElement.
Read a live GUI value	GetGUIElementProperty(gui, element, property, fallback).
Operate through widget rules	SaveGameWidgetCommand.
Write / restore a fixed save	SaveGameSlot / LoadGameSlot.
Open system Save / Load flow	SaveGame / LoadGame.
Choose authored picture for saves	SetSaveSlotImage.

These are API names. Action-list labels use readable names and can add spaces. Use the action’s displayed form to fill character, GUI, element and slot references; do not paste a whole script into a single ID field.

GDScript records in this book extend script_base.gd. Create a normal script record, apply it and assign that record to the intended event. Do not paste the run function into the raw JSON for a GUI element.

PCA Script: the same item and save actions

PCA Script uses readable statements for common inventory work and snake-case shared action functions for detailed operations. This record gives one oil and opens the inventory:

```
language 1.0

give oil to player
show gui inventory_lab
```

Use the current game’s IDs. A second record can dispatch the same Save selected command used by the Button:

```
language 1.0

let options = {"guiId": "save_lab", "elementId": "save_slots", "command": "save-selected"}
save_game_widget_command(options)
```

Shared action functions wait through the action system. They use the same property names as the GDScript dictionaries. This command still obeys the widget’s mode, range and prompt rules; the language does not bypass them.

> The small command example has no mandatory-selection guard. Add your own selection check or use the guarded GDScript version when that is part of your menu’s design.

Lua and BASIC use the shared actions too

A Lua record returns a module table. This complete record dispatches Load selected:

```
local Script = {}

function Script.run(context)
    await(SaveGameWidgetCommand({
        guiId = "save_lab",
        elementId = "save_slots",
        command = "load-selected",
        slot = -1
    }))
    return true
end

return Script
```

The equivalent BASIC record uses a Run function and a dictionary:

```
OPTION PCA3D BASIC 1.0

FUNCTION Run(context AS Dictionary)
    DIM options = {"guiId": "save_lab", "elementId": "save_slots"}
    options["command"] = "load-selected"
    options["slot"] = -1
    AWAIT SaveGameWidgetCommand(options)
    RETURN TRUE
END FUNCTION
```

These are command-dispatch examples. Add a selected-slot/file-existence check when your surrounding UI needs one. Keep slot values numerical; a name such as checkpoint_1 is not a numbered slot.

Save-slot event context reference

Context key	Meaning at input time
guiElementId	The Save Game Widget element ID.
saveSlot	Actual numerical save slot.
saveExists	Whether the saved state exists.
saveName	Stored save name.
saveSceneId / saveSceneName	Saved scene references.
saveTimestamp	Stored timestamp text.
saveDate / saveTime	Stored date and time text.
savePlayTime	Formatted recorded play time.
savePlayTimeSeconds	Recorded duration as a number.
saveThumbnailPath	Thumbnail path, if supplied.
saveIsAutosave	Whether this slot is the configured autosave.
saveStatus	saved or empty.

The normal GUI event context also identifies the GUI and event. The script example names its parameter event_context so the context is supplied to run. Use Dictionary.get with a sensible fallback when a script can be called from more than one kind of event.

The context describes the slot before the built-in action. Query the live slot later for a completed save or delete. Status-variable values written on selection have their own autosave distinction.

> Do not use saveThumbnailPath as proof that the gameplay save can load. Check saveExists and the actual load result when a later action depends on success.

30. Build the complete workshop flow

This final walk-through ties the book’s pieces together. Use your own project and assets. The IDs below match the screenshots, but the same sequence works for a museum, a spaceship or a village adventure.

1. Define Brass key, Brass gear, Clock spring, Bearing oil, Notebook, Copper wire, Magnifying lens, Tower map and Lubricated gear.
2. Make oil stackable with Max Stack 5. Give the items appropriate Types. Add the eight starting items to the active character’s initial collection.
3. Create inventory_lab with bag_grid. Use 3 × 2 cells, six visible slots, left icons and clear names.
4. Add Look actions to at least the key and notebook. Add the guarded oil-on-gear recipe.
5. Map Inventory Screen, and add open/close actions. Test selection, Look, combination and the second inventory page.

At this point the player can collect, inspect and combine items. Keep the first layout unchanged while proving these interactions. Add artwork next, then re-test the same sequence.

> Use separately authored filtered views if you need inventory tabs in this reference build. The dynamic runtime-rebuild limitation is covered in chapter 26.

Finish the save browser

1. Set eight manual slots, dedicated autosave 0 and screenshot capture 320 × 180 in Global Settings.
2. Create save_lab with save_slots. Use one column, three visible rows and manual range 1–8, with autosave listed.
3. Use save-load mode. Make row Click select. Add Load selected, Save selected, Rename selected and Delete selected Buttons with widget commands.
4. Keep name, overwrite and delete prompts enabled. Leave Close After none during your first tests.
5. Map Save Screen and Load Screen. Add a HUD open Button and a Close Button.
6. Save before the workshop repair. Repair the gear, remove a spring and change a test variable. Load the earlier slot.
7. Verify the restored bag and variable, then test rename, canceled overwrite, canceled delete and confirmed delete.
8. Close the application, launch it again and load your remaining test save.

Only after that sequence works should you shorten the player flow with double-click loading, automatic menu close or a custom checkpoint button. Each shortcut should preserve the same clear destination and protection rules.

> Save Project when authoring is finished. That preserves the menus and actions you built. Make a separate player save in Play mode when testing the actual save system.

A release checklist worth keeping

Area	Ready when…
Inventory ownership	Every intended starting item appears for the correct character.
Item actions	Look, selected use and recipes each run once on the right target.
Quantities	Caps, consumption and transfers agree with the puzzle rules.
Layout	The longest names and final partial pages remain readable.
Artwork	Root, grid, slots and screenshots have deliberate independent styling.
Input	Press/release, mouse, touch and controller flows are checked where supported.
Save destinations	Manual, quick and autosave slots cannot be confused.
Protection	Names, overwrite, delete and cancel paths behave as advertised.
Restore	A fresh application session loads representative inventory and story state correctly.
Distribution	The exported application passes the same tests on each supported platform.

Keep one uncomplicated version of the menus available while experimenting. It gives you a known comparison when a new effect, layout change or shortcut behaves differently.

Players usually meet these menus when they are trying to remember an item, protect their progress or return to a puzzle. Clear names, honest feedback and predictable buttons matter more than squeezing another decoration into the corner.

Glossary and quick lookup

Term	Meaning in this guide
Definition	An authored record describing an item, GUI or other game resource.
Carried inventory	The items and quantities a character actually owns during play.
Capacity	Different item IDs or save cards supplied on one page.
Slot	An inventory display cell, or a numbered save destination; context matters.
Widget	A composite GUI element that creates its own repeated internal controls.
Modal	A window that blocks ordinary interaction behind it.
Type filter	A view of carried items matching at least one authored item type.
Selection	The currently chosen item or save slot.
Pressed state	Temporary feedback while an input is held.
Thumbnail	The small stored picture associated with a saved game.
Template	A caption or name containing supported replacement tokens.
Project ID	The stable project identity also used to distinguish local save storage.

For setup, start with chapters 2 and 14. For separate corner and image layers, use chapters 10–11 and 17. For saving rules, use chapters 19–21. For problems, use chapters 26–27. Exact keys and context values are in chapters 28–29.

Reference edition: PCA 3D Adventure Studio 0.86.3.4. Screenshots and behavior were checked against the available editor and runtime for that build. No practice files are required.