PCA 3D ADVENTURE STUDIO

Images, Carousels and Choice Controls

Sliders, radio buttons and check boxes, step by step

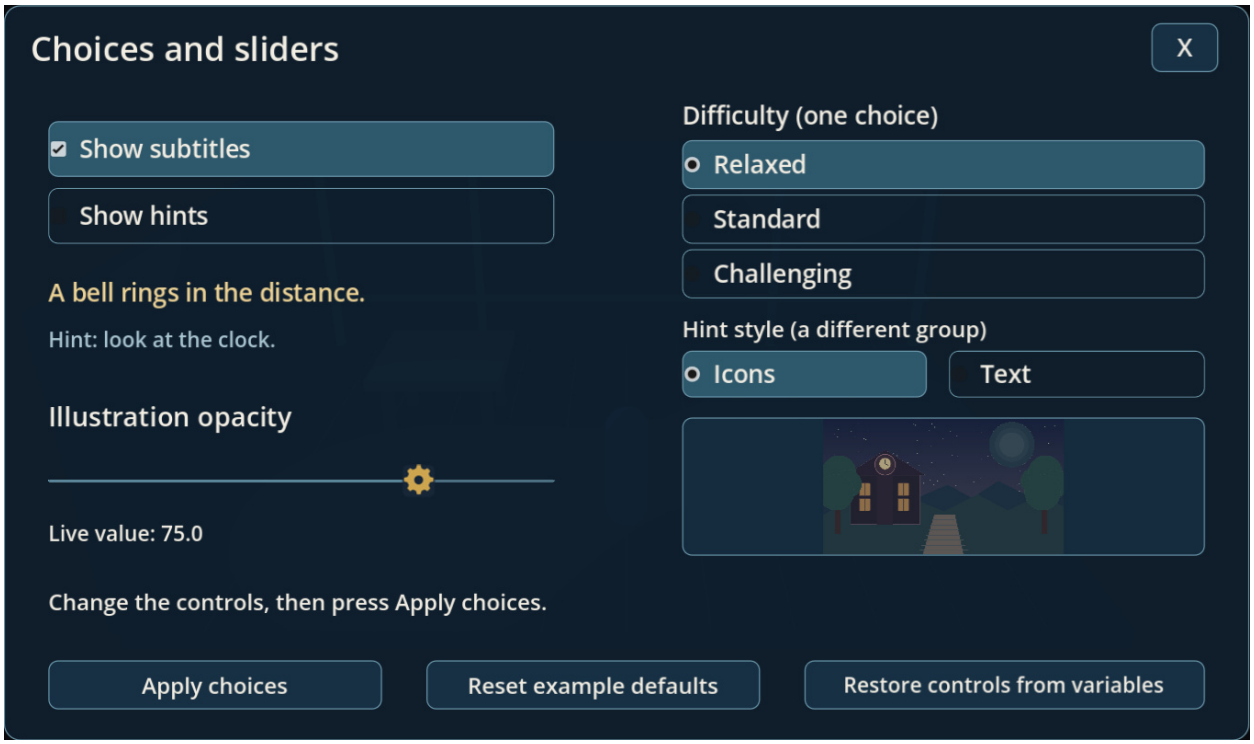

Figure 1. A running preferences screen made from ordinary GUI elements.

A practical guide for Windows, macOS and Linux.

Editor reference: 0.86.3.4 | Companion to the GUI Designer tutorials

Before you begin

This guide covers five kinds of GUI control: Images, Image Carousels, Sliders, Radio Buttons and Check Boxes. It also covers the Inventory Image Carousel, which uses carried items instead of an authored picture list.

You will build an image display, a reference gallery and a small preferences screen. The examples show how to make controls look right, how to try them in Preview and how to use their values during a running game.

The instructions assume you can open the GUI Designer, add an element and save a project. The earlier GUI tutorials cover those basics. Work in a copy of an existing project if you want to experiment. No separate practice files are needed or supplied.

Part of the interface	What belongs there
General	Identity, position, dimensions, visibility and enabled state.
Text & Content	Captions and font settings on controls that display text.
Type Settings	Image fitting, carousel behavior, checked state and slider range.
Visual Style / Images	Surface colors, corners, background artwork and supported state images.
Events	Actions or scripts for pointer interactions.
Raw JSON (Advanced)	Optional properties that do not have a dedicated typed field in this build.

> Element fields normally update live. Script editors and whole-GUI record settings have Apply controls. Use Save Project to keep authoring changes. A running game has its own temporary state.

Screenshots and behavior were checked in 0.86.3.4. Desktop window decorations can differ, but these editor controls are shared across platforms.

Contents

Click a chapter title or use the PDF bookmarks. Continued pages stay with their chapter.

1. Choose the right control 4
2. Set up a comfortable workspace 5
3. Prepare the image assets 6
4. Add an Image element 7
5. Understand image fitting 9
6. Add hover and pressed images 11
7. Animate an Image with frame assets 13
8. Give an image a gameplay action 14
9. Build an Image Carousel 16
10. Choose carousel navigation 18
11. Show several cards at once 19
12. Use an Inventory Image Carousel 20
13. Control a carousel with actions 23
14. Read the current carousel entry 26
15. Add a Slider 28
16. Style the track and handle 30
17. Read a slider’s live value 32
18. Add independent Check Boxes 34
19. Read or bind a Check Box 36
20. Build a group of Radio Buttons 37
21. Choose the right event timing 40
22. Apply choices to visible game behavior 41
23. Restore controls from game data 43
24. Captions and wildcards 45
25. Know what Preview can test 46
26. Save the right kind of state 47
27. Make controls comfortable to use 48
28. Fix image and carousel problems 49
29. Fix slider and choice problems 50
30. A compact reference for scripts 51

1. Choose the right control

Control	Use it for	Do not assume it does this
Image	One illustration, icon, portrait or frame sequence.	An image is not automatically an inventory item.
Image Carousel	Browsing an ordered set of imported images.	It does not automatically become a timed slideshow or write a selected-ID variable.
Inventory Image Carousel	Browsing the selected character’s carried item pictures.	Browsing does not use, equip or remove an item.
Slider	Choosing a number within a range.	A slider named Volume does not automatically change the sound system.
Radio Buttons	Choosing one option from a related group.	Different groups need different Radio Group names.
Check Boxes	Independent yes/no options.	Checking a box does not automatically apply the setting named in its caption.

Appearance, current control state and game data are separate. A useful preferences screen connects all three: it looks clear, lets the player make a choice, then applies that choice to a real game behavior.

The examples use a separate Apply choices Button. This makes the timing easy to understand and gives one place to read the completed choices. Optional variable bindings are covered after the basic controls.

2. Set up a comfortable workspace

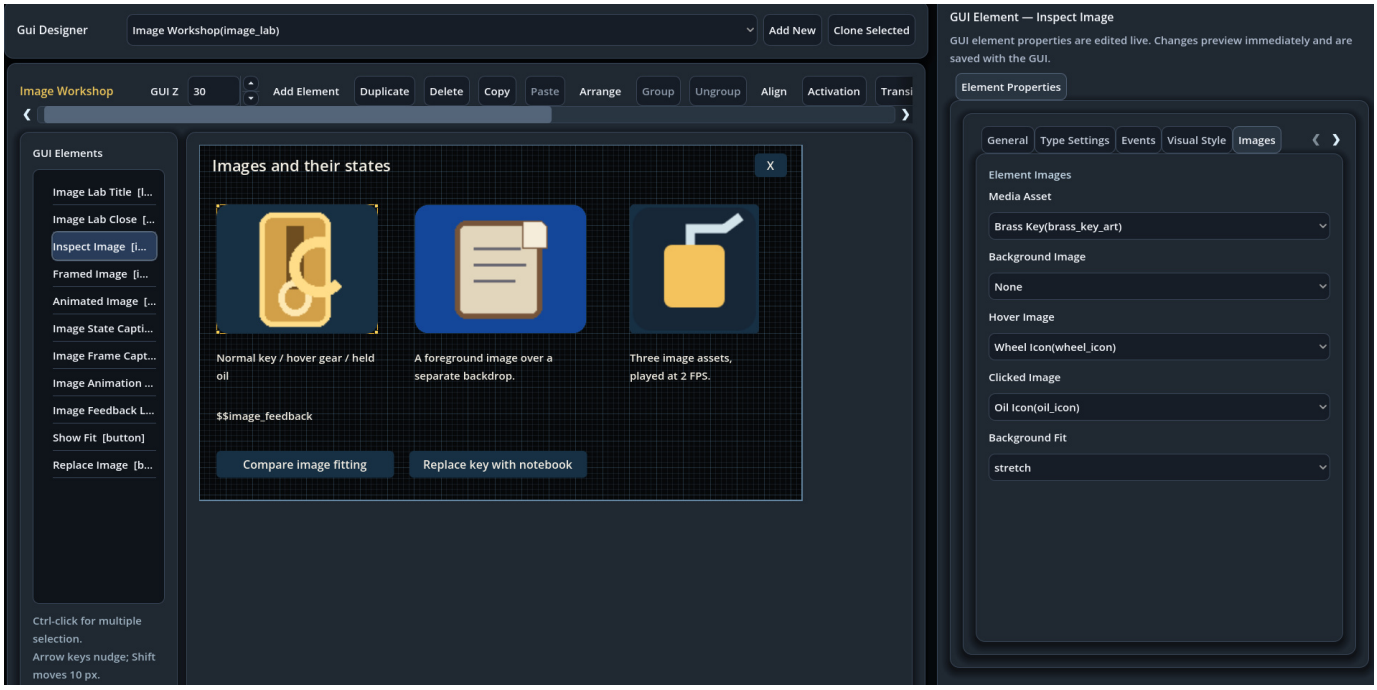

Figure 2. Choose the GUI at the top, select an element on the left and edit its properties in the Inspector.

1. Open the GUI Designer. Create or choose the GUI you want to work on.
2. Add an element, then give it a clear ID and Editor Name in General.
3. Set its dimensions and position before choosing detailed artwork or behavior.
4. Use Preview to try local interaction. Turn Preview off again before moving or resizing elements.
5. Save after each useful stage, then test gameplay behavior in Play / Restart or Test Scene.

The examples use a 1280 by 720 project canvas and centered 1120 by 660 GUI windows. A smaller design is fine. The important thing is to leave room for captions, navigation and a way to close the window.

> Editor zoom only changes the editing view. It does not resize the authored GUI. If the Inspector is crowded, widen it or scroll the property tabs rather than changing the game’s layout to fit your monitor.

3. Prepare the image assets

Import the images before assigning them to GUI controls. For these examples, any clear key, notebook and gear images will do. The screenshots use assets already present in the engine projects.

Example asset ID	Readable name	Where it is used
brass_key_art	Brass Key	The normal image and an optional gallery entry.
notebook_art	Notebook	A foreground picture over a separate backdrop.
wheel_icon / oil_icon	Wheel Icon / Oil Icon	Distinct hover, pressed and animation examples.
lab_view	Moonlit Workshop	A wide picture for fitting and gallery tests.
map_view / moon_dial	Map Room / Moon Dial	More gallery pictures.
state_blue	Blue Backdrop	An obvious background behind the notebook.

Use your asset importer to bring in the files, then check their IDs and names. A dropdown may show Brass Key(brass_key_art); scripts and ID lists use brass_key_art alone. The source filename, editor name and asset ID are not interchangeable.

Use transparent artwork for an icon that should sit over a background. Large empty margins inside a source image can make its visible drawing look smaller than the GUI rectangle suggests.

> Use a small number of appropriately sized images while learning. A full-screen background and a tiny slider handle do not need the same source resolution. This guide does not require any new art files.

4. Add an Image element

1. Create an Image Workshop GUI with ID image_lab.
2. Use Add Element to add an Image. Set ID to inspect_image.
3. In General, set X 32, Y 110, Width 300 and Height 240 for this example.
4. Open Images and choose Brass Key as Media Asset.
5. Open Type Settings and leave Content Fit on contain. Save the project.

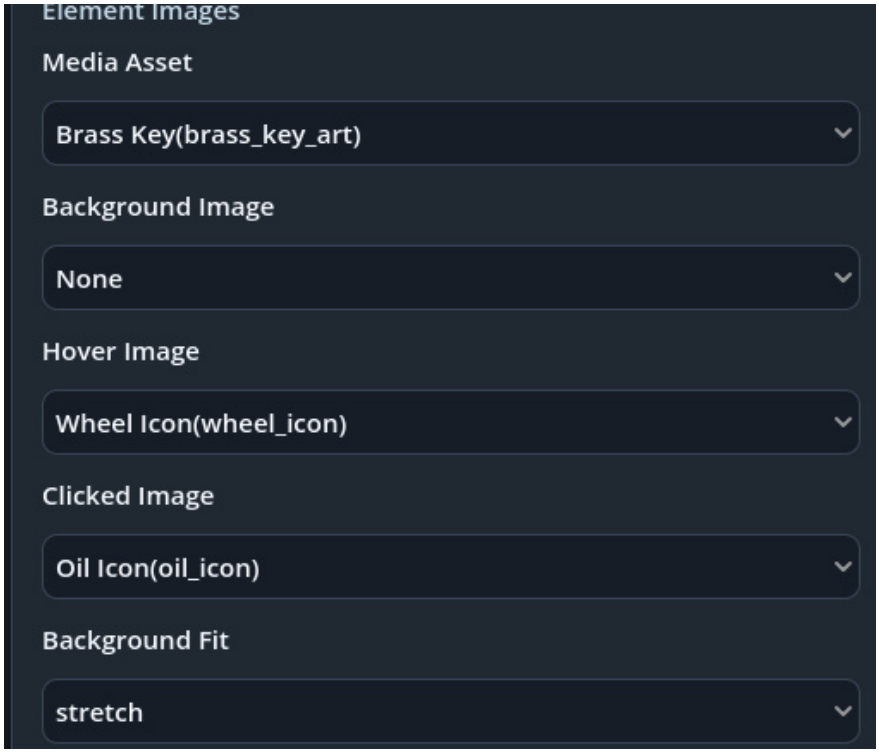

Figure 3. For an Image element, Media Asset supplies the foreground picture. Hover and Clicked Image supply alternate foreground states.

The selection rectangle is the element’s layout area. It is not necessarily the visible drawing’s outline. A square transparent icon fitted inside a wide rectangle will leave space on the sides.

An Image element does not draw a text caption from the Text field. Put a separate Label beside or below it when you need a description.

Foreground picture and background artwork

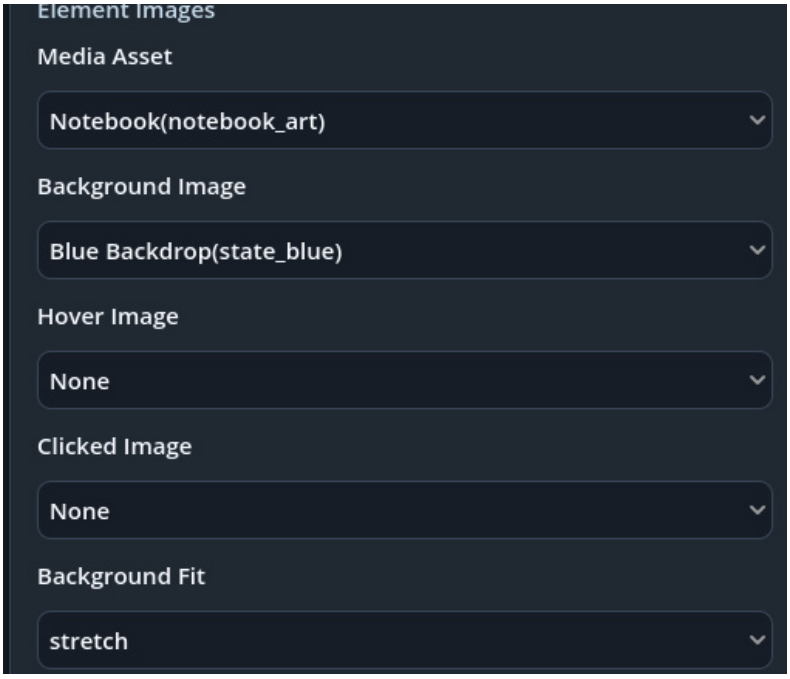

Figure 4. The notebook is the Media Asset. Blue Backdrop is a separate Background Image.

An Image element can have two visible layers. The foreground is the main picture selected as Media Asset. The background is artwork drawn behind that picture, selected as Background Image.

Setting	Controls
Media Asset	The main image, such as the notebook.
Content Fit and content margins	How that foreground image fits inside the element.
Background Image	Separate backing artwork.
Background Fit	How the backing artwork fills the element.
Background Color	A surface color where it is visible behind or around the images.

For the example, use a 320 by 240 Image, choose Notebook as Media Asset and Blue Backdrop as Background Image. Set each content margin to 24. The result is a notebook inset from a blue frame.

> An opaque foreground can hide its background completely. That does not mean the Background Image setting failed. Use transparency, margins or a fitting mode that leaves space.

5. Understand image fitting

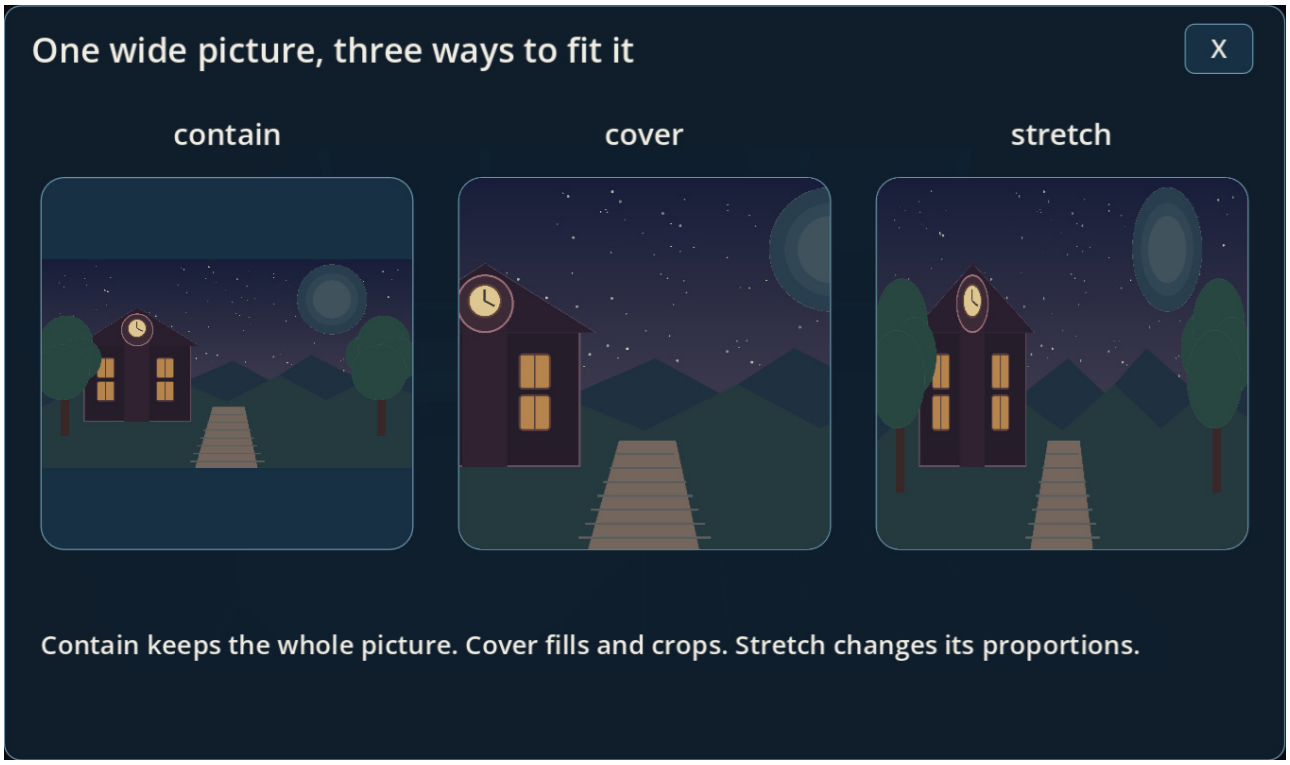

Figure 5. The same wide picture in three square Image elements. The source artwork is unchanged.

Content Fit	What happens	Good starting use
contain	Keeps the whole picture and its proportions. Empty space may remain.	Inventory art, portraits, diagrams and illustrations that must stay complete.
cover	Fills the rectangle while keeping proportions. Parts outside it are cropped.	A thumbnail or decorative panel that must fill its frame.
stretch	Fills the rectangle by changing proportions.	Artwork designed for that exact shape, or a deliberate distortion.
tile	Repeats the source texture.	A repeating pattern rather than a single portrait or photograph.

Choose the fitting mode before changing Width and Height repeatedly. If a face is being cropped, try contain. If a frame has unwanted empty bands, try cover and check that the crop is acceptable.

Margins, corners and transparency

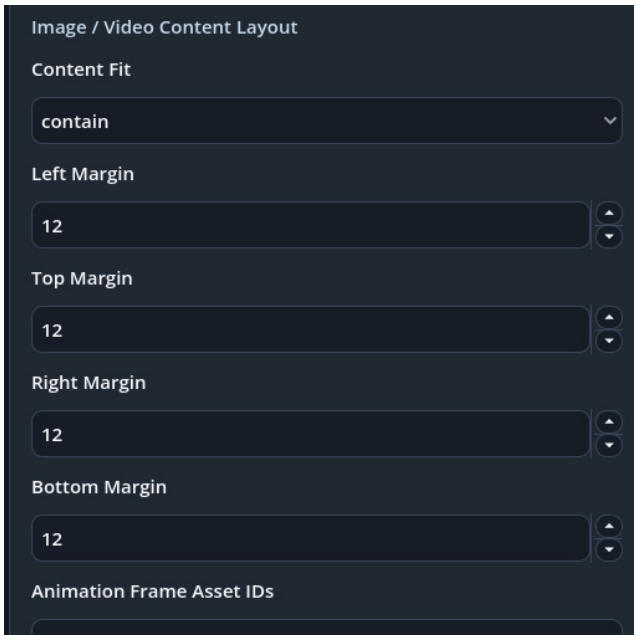

Figure 6. Content Fit and the four margins control the foreground picture’s usable area.

Field	Effect
Left / Top / Right / Bottom Margin	Inset the foreground independently from each edge. The background remains a separate layer.
Border Radius	Common corner rounding in Visual Style.
Four corner overrides	-1 inherits the common radius; 0 makes that corner square; a positive value sets its curve.
Surface Opacity	Controls visibility. Check both the image and the background in gameplay.

Start with equal margins, then change one side only when the design needs it. Very large margins can leave almost no room for the image. Corners clip the artwork to the element’s rounded shape; they do not edit the source file.

For an icon over a room, check its source transparency first. Do not rely on a black background becoming transparent automatically. A dark pixel in a normal image is still part of the image.

6. Add hover and pressed images

1. Select inspect_image and open Images.
2. Leave Media Asset set to Brass Key.
3. Set Hover Image to Wheel Icon.
4. Set Clicked Image to Oil Icon.
5. Enable designer Preview, or run the game. Point at the key, press and hold, then release.



Figure 7. Hover: the wheel image replaces the foreground key.

The deliberately different drawings make the state changes obvious. In a finished interface, you would normally use matching artwork with a highlight, glow or pressed treatment. Keep all three source images aligned so the drawing does not jump when the state changes.

> For an Image element, Hover and Clicked Image replace the foreground picture. Background Image remains a separate layer. The same state fields on a composite control can affect its overall backing instead of an individual item.

A pressed image is temporary

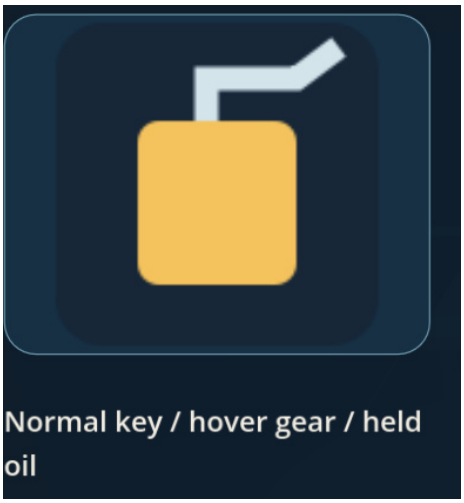

Figure 8. Held: the oil image is shown while the pointer press is active.

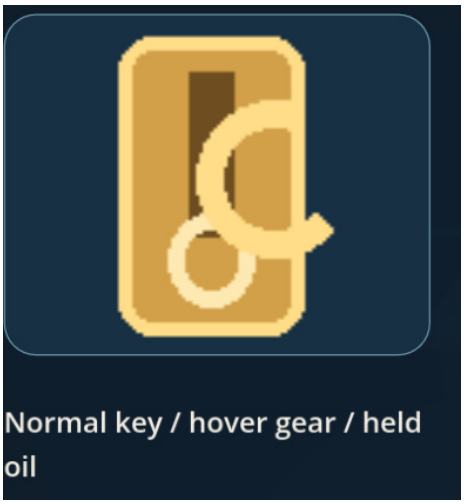

Figure 9. Released: the normal key returns. This is not a permanent selected-state image.

Clicked Image is momentary artwork. In this build, releasing a press that used it restores the normal image, even if the pointer is still over the element. Leaving and entering again can show the hover image.

Test a release outside the element too: press, drag away and release. The pressed image should not remain stuck. A selected radio option or checked box is a different kind of state; do not use a pressed picture as its saved on/off value.

7. Animate an Image with frame assets

An Image can play a sequence of imported image assets. This is useful for a blinking lamp, a small animated ornament or a changing icon. It is separate from carousel browsing.

1. Add another Image with ID animated_image.
2. In Type Settings, enter key_icon, wheel_icon, oil_icon in Animation Frame Asset IDs.
3. Set Image Animation FPS to 2. Start Preview and watch the sequence.
4. Replace those test icons with the actual frame assets for your animation when the timing feels right.

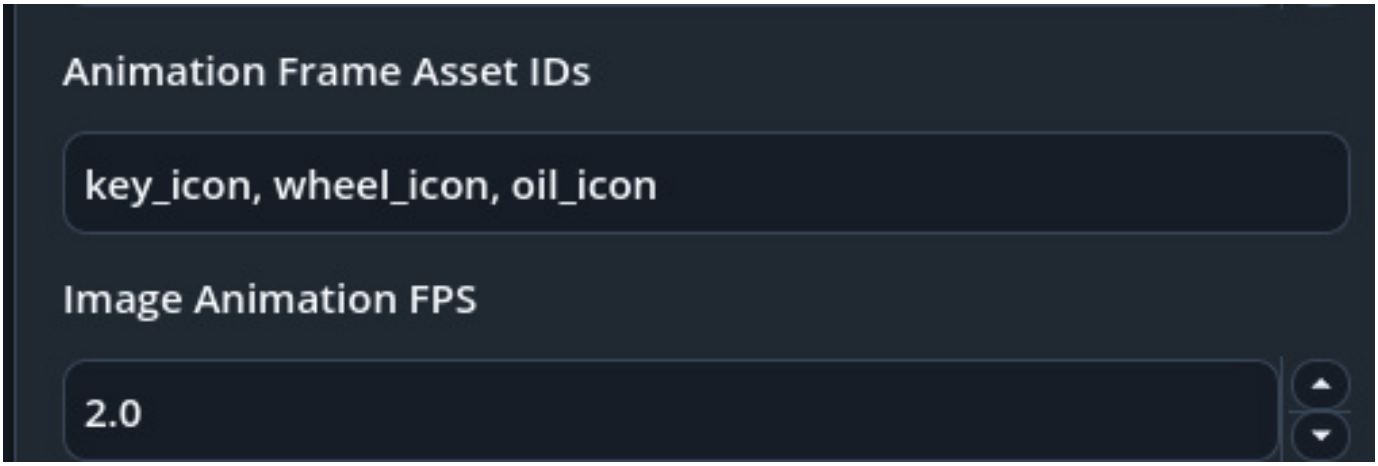

Figure 10. Animation Frame Asset IDs lists assets in order. FPS controls the playback rate.

The entries are image asset IDs, not cell numbers such as 0, 1 and 2. In this workflow, each entry resolves to a texture. A spritesheet file should not be assumed to turn into separately cropped animation cells here.

The normal image is the fallback when no listed frames can be used. Valid listed frames take precedence over that fallback. Keep the frames the same size and alignment so the animation stays steady.

> At 2 FPS, a three-frame sequence takes about 1.5 seconds to loop. A carousel’s Next control changes a selected picture; it does not use this FPS field to run a slideshow.

8. Give an image a gameplay action

State images work without a gameplay event. Add an event only when the picture should do something, such as inspect a clue, open a GUI or run a script.

1. Create a String variable named image_feedback.
2. Select inspect_image and open Events. Set Click to Set Variable.
3. Choose image_feedback and enter a short response as its value.
4. Add a normal Label whose Text is $$image_feedback.
5. Run the game and click the image. Check the Label changes.

The event list also includes Hover, Double Click, Right Click and Double Right Click, with action-list operations available alongside the basic behaviors. Use the event that matches the player’s intended interaction.

A Hover event can run whenever the pointer enters. It is a poor place for an unguarded one-time item reward or a long conversation. Keep hover feedback lightweight, or use a story variable to prevent repeated work.

Goal	Useful action
Open an inspection window	Show GUI or an Action List that opens the desired GUI.
Run custom logic	run-visual-script with the script ID in Click Target, or the shared script-running action.
Change the displayed picture	Change GUI Element Property → assetId.
Disable an interactive image	Disable GUI Element.

An image with click artwork can still be decorative. To make it consume a room click deliberately, assign the intended Click behavior and test it in the running game.

Change an image from a script

Use Change GUI Element Property when a story event should replace a picture. This complete GDScript example changes the key to a notebook:

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

func run(_context: Dictionary = {}) -> Variant:
    await ChangeGUIElementProperty({
        "guiId": "image_lab",
        "elementId": "inspect_image",
        "property": "assetId",
        "value": "notebook_art"
    })
    return true
```

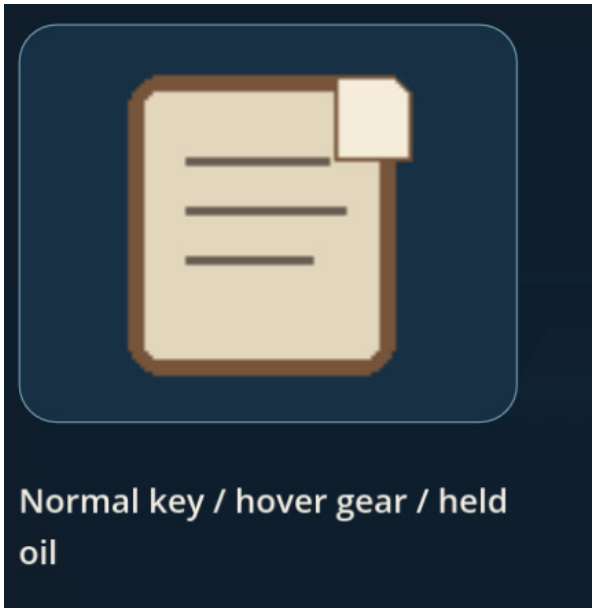

Figure 11. The same Image element now displays the notebook as its normal foreground.

Create a script record, Apply Changes, then assign it to a Button or story interaction. The GUI owns the element; the new value is an image asset ID. Use backgroundImageAssetId instead of assetId when you intend to change only the background layer.

Keep the element ID stable. Replacing its picture does not require deleting the element and creating a new one in the designer. Existing event and layout references can continue to point to inspect_image.

9. Build an Image Carousel

A carousel displays one current picture, with optional neighboring cards and navigation. Use it for reference art, illustrations, collected photos or a selection preview.

1. Create a Reference Gallery GUI with ID gallery_lab.
2. Add an Image Carousel with ID reference_carousel. Set X 32, Y 92, Width 1056 and Height 372.
3. Open Type Settings. Enter lab_view, map_view, moon_dial in Initial Image Assets (IDs).
4. Leave Images Visible at 1. Enable Show Arrow Buttons and Show Image / Item Name.
5. Run Preview and use the arrows to browse.

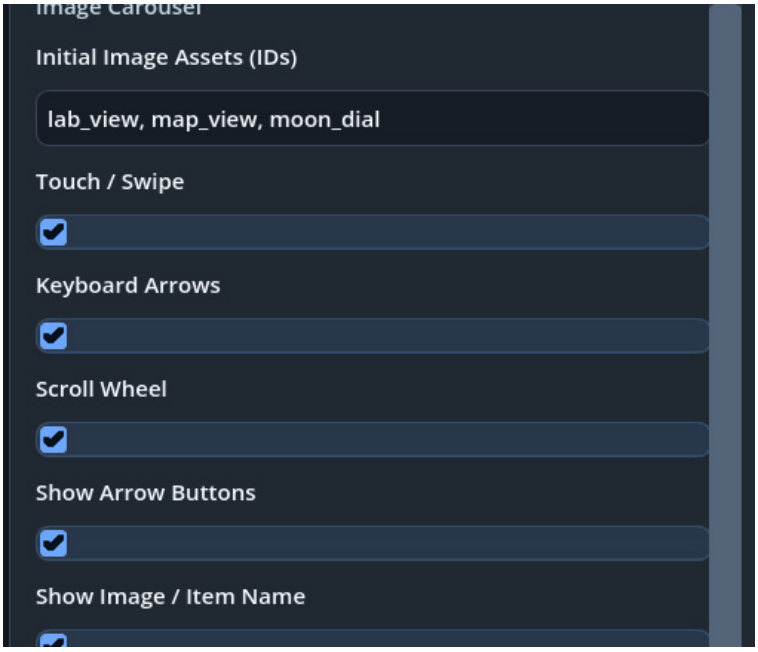

Figure 12. Initial Image Assets sets the picture order. The picker offers imported image assets.

The caption uses each asset’s assigned name. Choose readable asset names before filling a large gallery, so the player sees Moonlit Workshop rather than a technical filename.

Choose and order the gallery pictures

Use Choose Initial Images to add imported images, or edit the IDs directly. The sequence in Initial Image Assets is the browsing order.

```
lab_view, map_view, moon_dial
```

Position in this example	Asset ID	Caption
First	lab_view	Moonlit Workshop
Second	map_view	Map Room
Third	moon_dial	Moon Dial

Separate IDs with commas. The parser also accepts semicolons or newlines. Repeated IDs in the authored list are de-duplicated, so repeating the same ID is not a reliable way to make it appear twice at startup.

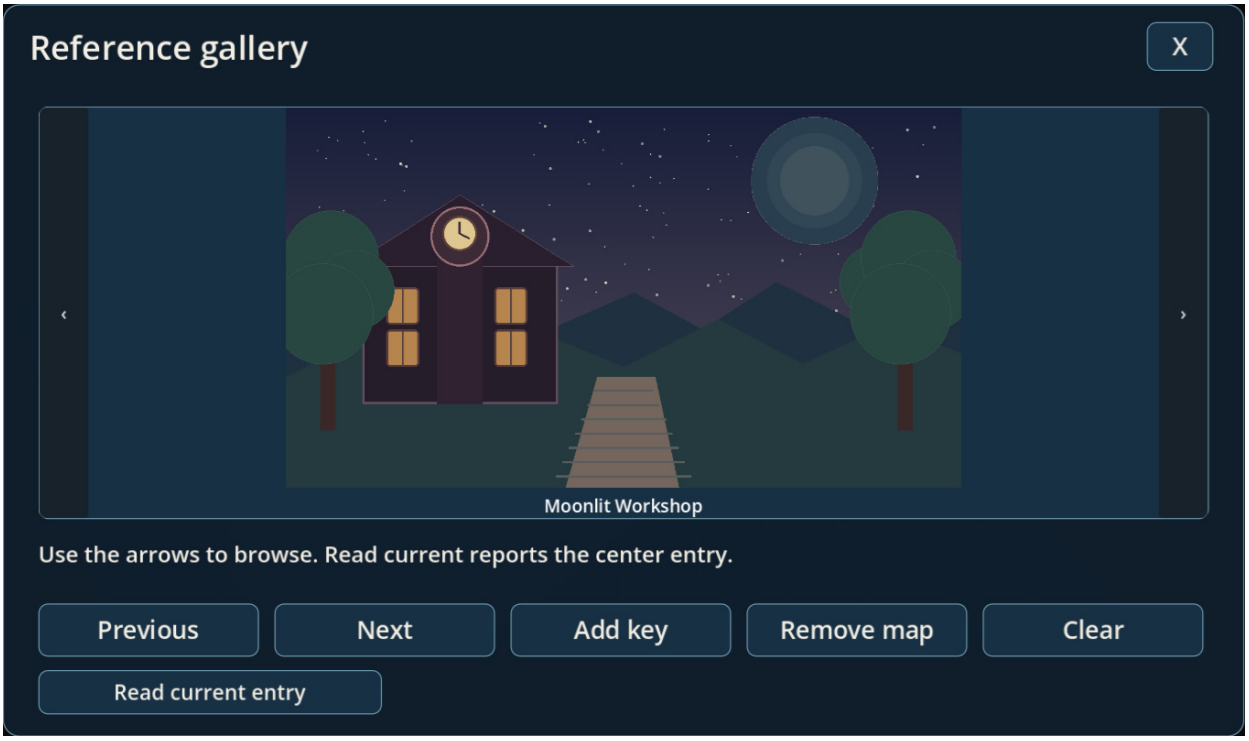

Figure 13. A single-picture gallery. Arrows and the caption are built into the carousel.

Carousels fit each card’s texture proportionally. The Image element’s Content Fit and animation-frame controls are not per-card carousel settings. Use prepared source artwork when cards need consistent borders or cropping.

10. Choose carousel navigation

Type Settings field	What it does
Touch / Swipe	Enables supported touch swipes and pan gestures.
Keyboard Arrows	Browses when the carousel has the relevant keyboard focus.
Scroll Wheel	Moves backward or forward over the carousel.
Show Arrow Buttons	Shows the built-in previous and next buttons.
Show Image / Item Name	Shows the current entry’s caption.
Wrap Around	Continues from the last entry to the first, and vice versa.
Empty Text	The caption shown when there are no entries.

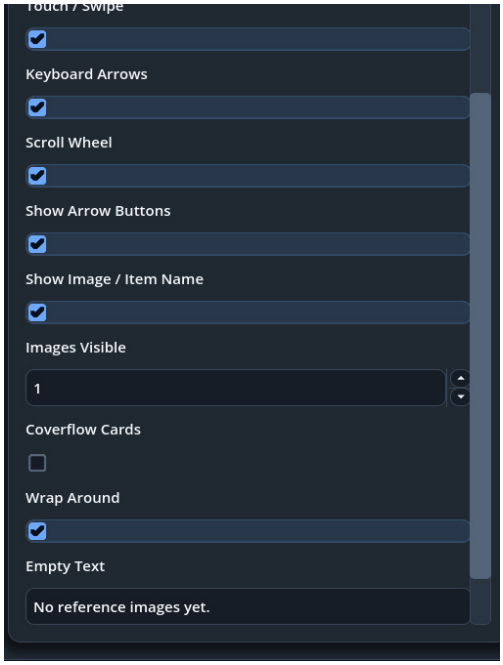

Figure 14. The navigation switches are independent. Keep at least one usable way to browse.

With Wrap Around off, navigation stops at the ends. Do not expect both arrows to disappear or become disabled at an end; with several entries, they can remain visible while the index is clamped.

> Clicking a neighboring picture is not the same as choosing it. In this build, browse with the arrows, wheel, supported gestures or external carousel actions.

11. Show several cards at once



Figure 15. Three visible cards with Coverflow Cards off.



Figure 16. The same gallery with Coverflow Cards on: neighboring cards are smaller, dimmer and overlap more.

Images Visible accepts 1 through 7. Start with 1 for a clear inspection view or 3 for a central picture with neighbors. Odd counts are usually easier to balance around the current entry.

Coverflow Cards changes the arrangement, scale and opacity of the visible cards. It does not turn the gallery into a 3D model viewer, and side cards do not gain their own click-selection behavior.

With Wrap Around off, some neighboring positions can be empty near the beginning or end. Increase the carousel’s width before trying to squeeze several large cards into a narrow control.

12. Use an Inventory Image Carousel

An Inventory Image Carousel builds its entries from carried inventory. It is useful for an illustrated bag or a large-item browser. The item definition and its owner decide what appears.

1. Create a Carried Item Carousel GUI with ID inventory_lab.
2. Add an Inventory Image Carousel with ID carried_carousel.
3. In Type Settings, choose Current Active Player as Inventory Owner.
4. Give the player a defined item with an Icon Asset assigned.
5. Run the game and check that its card appears.

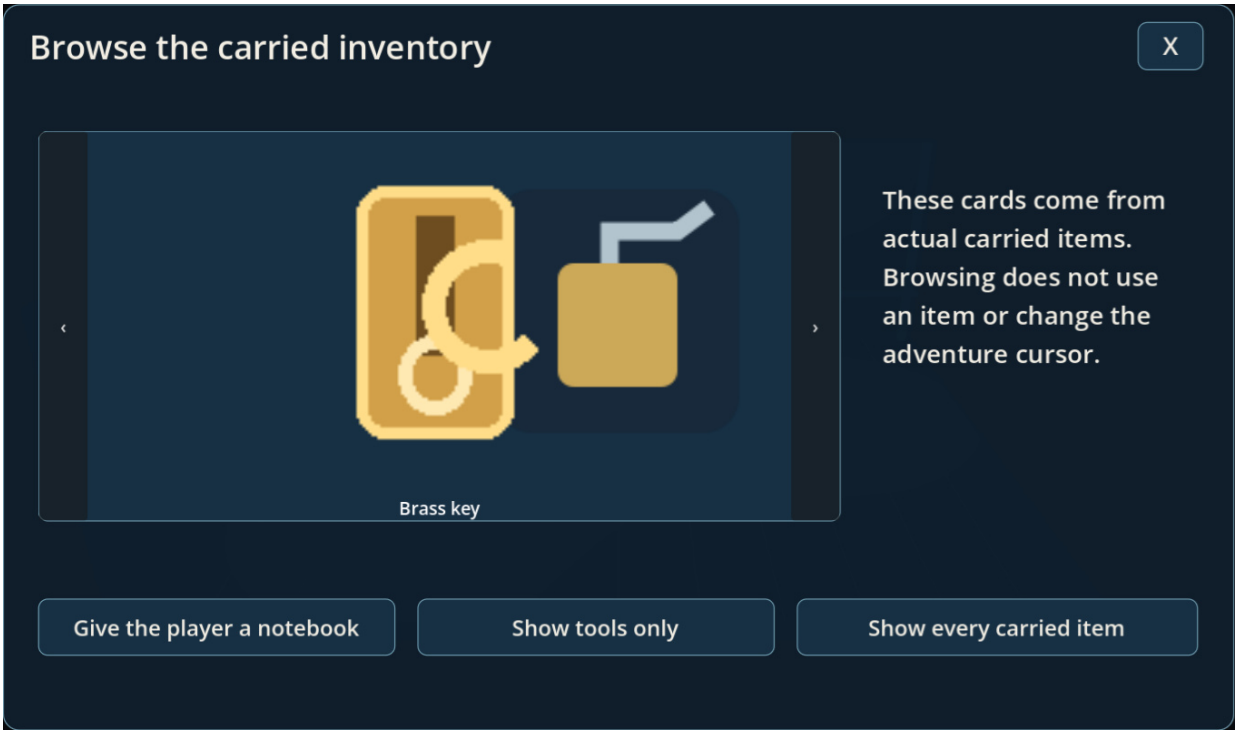

Figure 17. The starting inventory contains a key and oil. This is live carried-item data.

Use Specific Character and the Character selector when this interface should show one particular character’s inventory. Choose a character definition ID, not a scene object that happens to represent that character.

> The normal Image Carousel uses authored image IDs. The Inventory Image Carousel uses item IDs and their icon assets. These are different sources, even though the controls look similar.

Give, remove and filter carried items



Figure 18. Inventory Owner, Character and Inventory Categories / Tags belong to the inventory carousel.

Leave Inventory Categories / Tags empty to show every eligible carried item. Enter tools to show items that have tools in a supported category, type or tag field. Several comma-separated tokens work as alternatives: an item matching any requested token can appear.

1. Give Brass key the category tools. Give Notebook the category notes in their inventory definitions.
2. Use Add Inventory to give notebook_item to the player during gameplay.
3. Confirm that the unfiltered carousel gains a notebook card.
4. Set Inventory Categories / Tags to tools. The notebook should disappear from this view while remaining carried by the player.

If an item is missing	Check
No carried quantity	The player must actually own it; a definition alone is not carried inventory.
Wrong owner	Check Current Active Player versus Specific Character.
No compatible icon	Assign the item’s Icon Asset. Do not assume another image field is used by this carousel.
Filter excludes it	Clear the filter, then check the item’s category, types or tags.

Changing the filter changes the view. It does not remove items from inventory. Stack quantities are not automatically shown as a quantity badge in this carousel’s caption.

Browsing is separate from using an item

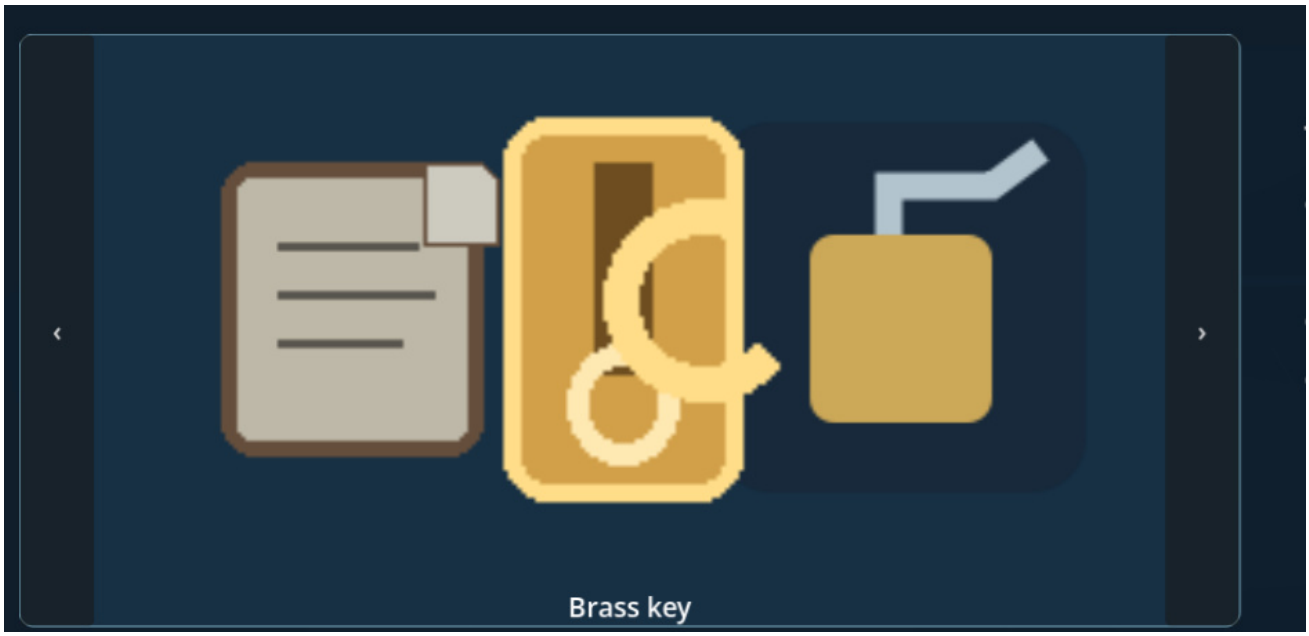

Figure 19. After an inventory award, the notebook joins the carried-item cards.

Moving the carousel changes its current entry. It does not automatically select an adventure inventory cursor, equip the item, combine it or call the item’s Use interaction. Add the gameplay action you actually intend.

Use inventory actions when you want to give, remove or change a carried quantity. The normal gallery’s Add Image, Remove Image and Clear Images actions are not a way to rewrite the player’s inventory.

The designer can show starting items from project and character setup. A running game may have different quantities, a different active player or a different character owner. Always test the intended game state.

> An empty filtered view is legitimate. Write an Empty Text message such as Nothing in this category so the player knows the interface is working.

13. Control a carousel with actions

Use external Buttons when you want navigation somewhere other than the built-in arrows. The same actions can be used in a GUI event, an action list or a visual script.

1. Add a Button beside the carousel and give it the caption Next.
2. In Events, choose Click → Carousel: Next Image.
3. Set GUI to Reference Gallery and Carousel to Reference Carousel.
4. Add a Previous Button using Carousel: Previous Image.
5. Run the game and confirm that both external Buttons browse the same central carousel.

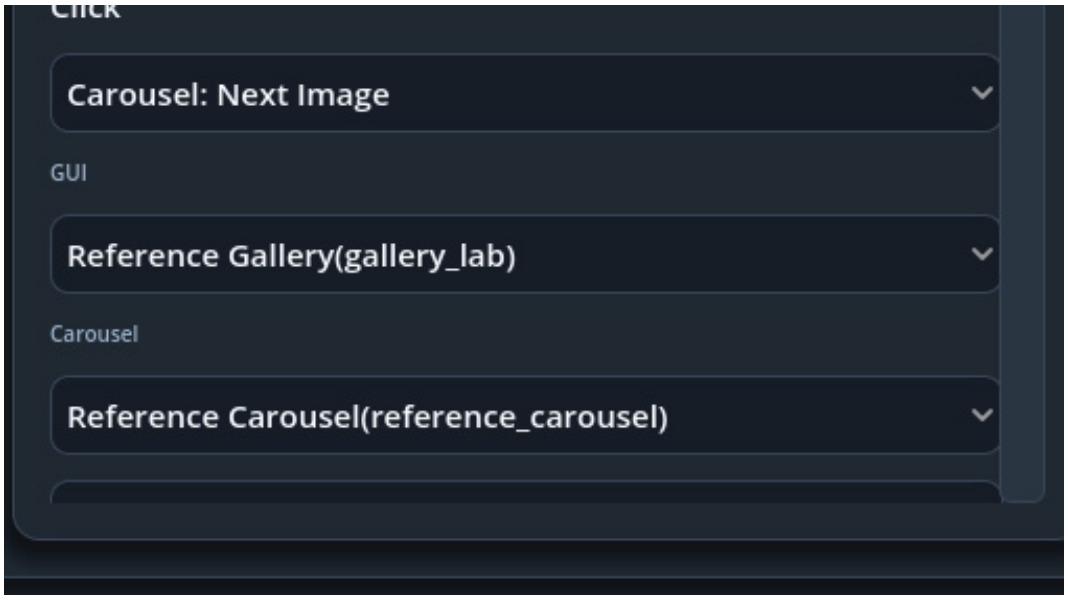

Figure 20. An external Next Button targets the gallery GUI and its carousel element.

Shared action	Purpose
Carousel: Previous Image	Step backward.
Carousel: Next Image	Step forward.
Carousel: Add Image	Add a picture to a normal Image Carousel.
Carousel: Remove Image	Remove one matching picture from a normal gallery.
Carousel: Clear Images	Clear the normal gallery’s current authored and added pictures.

Keep targets specific. Supplying the GUI and element ID makes it clear which gallery should respond when your game has several.

Add, remove and clear gallery entries

For an Add key Button, choose Carousel: Add Image, then use GUI gallery_lab, Image Carousel reference_carousel and Image Asset brass_key_art. The added entry goes after the existing entries.



Figure 21. The normal gallery now includes a Brass Key entry added during play.

Runtime additions can repeat an asset ID. Pressing Add key twice can create two matching added entries. This differs from the de-duplicated startup ID list. Use a story variable to guard a one-time discovery.

Remove Image removes one matching occurrence, looking in runtime additions before the original authored list. Clear Images empties both normal-gallery lists. Give a cleared gallery a useful Empty Text caption and a way to close it.

> These edits belong to the running widget. They do not rewrite the project’s Initial Image Assets field. A later rebuild of the carousel can restore authored entries and lose temporary additions.

Carousel actions in scripts and nodes

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

func run(_context: Dictionary = {}) -> Variant:
    await CarouselAddImage({
        "guiId": "gallery_lab",
        "elementId": "reference_carousel",
        "assetId": "brass_key_art"
    })
    await CarouselNext({
        "guiId": "gallery_lab",
        "elementId": "reference_carousel"
    })
    return true
```

Create a GDScript record with this source, Apply Changes and assign it to an interaction. Add Image appends the key; Next advances one position. Next does not automatically jump to the last added picture.

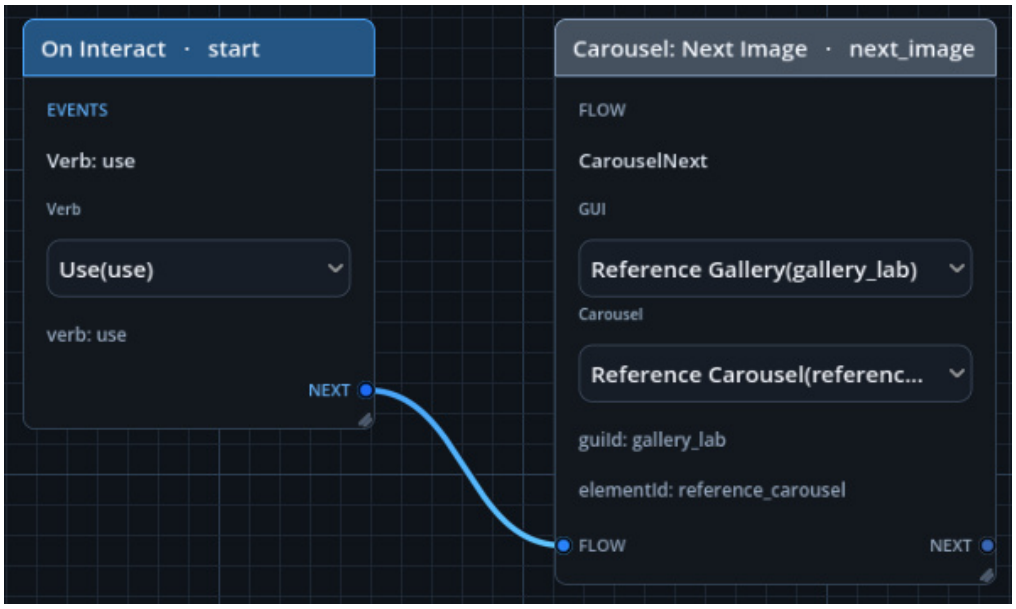

Figure 22. A visual graph can use Carousel: Next Image with the same two target IDs.

For visual scripting, connect the intended entry node to the carousel action node. An action list uses the same fields. You can place another story action before or after the navigation step without changing the carousel's source.

> Change GUI Element Property can rebuild a normal carousel. Set its layout and style before adding temporary runtime entries, or reconstruct the entries deliberately after a rebuild.

14. Read the current carousel entry

The current carousel entry is not automatically written to a Selected Item Variable. This build does not expose a typed output field like the List Box does. A separate confirmation Button can read it through the public GUI-property helpers.

Create a String variable gallery_feedback and a Label with Text $$gallery_feedback. Then create this GDScript record:

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

func run(_context: Dictionary = {}) -> Variant:
    var entries: Array = GetGUIElementProperty(
        "gallery_lab", "reference_carousel", "entries", []
    )
    var index: int = int(GetGUIElementProperty(
        "gallery_lab", "reference_carousel", "selected_index", 0
    ))
    var result: String = "The gallery is empty."
    if index >= 0 and index < entries.size():
        var entry: Dictionary = entries[index]
        result = str(entry.get("caption", ""))
        result += " | ID: " + str(entry.get("id", ""))
    await SetVariable({
        "name": "gallery_feedback", "value": result
    })
    return true
```

1. Apply Changes to the script.
2. Add a Read current entry Button. Set Click to run-visual-script and Click Target to the script ID.
3. Run the game, browse to Map Room and press the Button. Check the returned caption and ID.

entries and selected_index are widget properties checked in 0.86.3.4. This is an advanced property-reading workflow, rather than a dedicated carousel selection event.

Know which ID the carousel returns



Figure 23. After browsing to Map Room, Read current reports its caption, entry ID and image asset.

Entry field	Normal Image Carousel	Inventory Image Carousel
id	The image asset ID.	The inventory item ID.
assetId	The image asset ID used for the card.	The item’s icon image asset ID.
caption	The asset’s assigned name.	The inventory item’s assigned name.

For an inventory confirmation script, change the target GUI and carousel IDs to inventory_lab and carried_carousel. Use the returned id when looking up or acting on the inventory item; assetId identifies its picture.

Always handle an empty entries Array and an out-of-range index. After removing pictures or changing the inventory filter, the currently available entries can be different from the ones shown a moment earlier.

Do not store selected_index as a permanent story identity. Index 1 means the second current entry, and a changed source order can put another picture there. Store a stable entry ID when that identity matters.

15. Add a Slider

A slider lets the player choose a number by moving a handle along a track. This example chooses an illustration’s opacity from 0 to 100, then applies it to an Image.

1. Create a Preferences Example GUI with ID settings_lab.
2. Add a Label with the caption Illustration opacity.
3. Add a Slider with ID opacity_slider. Use X 40, Y 402, Width 455 and Height 50.
4. Open Type Settings. Set Minimum 0, Maximum 100, Value 75 and Step 5.
5. Enable Preview and move the handle. Check that it stays within the range.

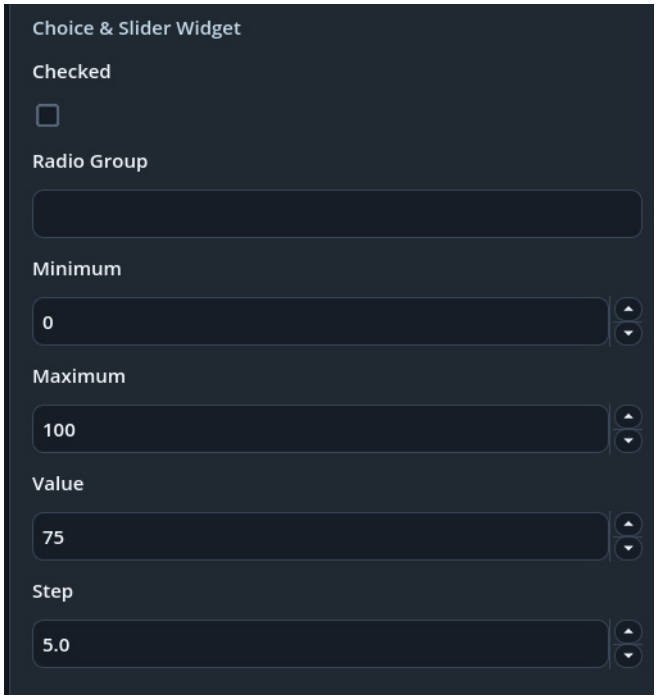

Figure 24. The range uses Minimum, Maximum, Value and Step. Checked and Radio Group are shared-section fields for the other control types.

Sliders are horizontal in this workflow. Increasing their Height gives the track and handle more vertical room; it does not turn the control into a vertical slider.

Choose a sensible range and step

Setting	Example	Meaning
Minimum	0	Lowest allowed value.
Maximum	100	Highest allowed value. Keep it greater than Minimum for a useful range.
Value	75	Authored starting value, clamped to the range during construction.
Step	5	The increment between allowed values: 0, 5, 10 and so on.

Step is a numeric increment, not a speed or a delay. Use Step 1 for whole-number choices. Use a fractional step, such as 0.05, when the range is 0 to 1 and small changes matter.

```
0 to 100 display scale:
Minimum 0, Maximum 100, Step 5

0 to 1 fractional scale:
Minimum 0, Maximum 1, Step 0.05
```

The control’s live value is numeric and can be a floating-point value. Read it as a float in scripts, even if this particular range uses whole steps. Convert to an integer only when the gameplay rule needs a whole number.

A slider’s caption does not bind it to an engine setting. A slider called Volume still needs an audio workflow; one called Opacity needs an action that applies the value to an Image.

> A zero-length range cannot provide a useful drag choice. Start with a small practical range, test both ends, then adjust the step for the amount of precision the player actually needs.

16. Style the track and handle

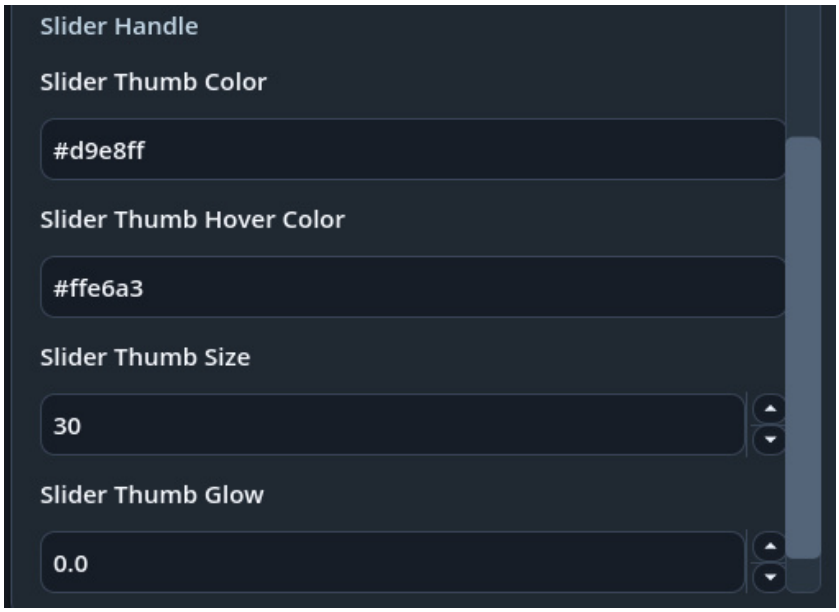

Figure 25. The Slider Handle controls appear in Visual Style. Colors are entered as color strings in this build.

Field	Use
Background and border colors	The ordinary track surface and outline.
Hover Background	The filled track area’s color in the runtime style.
Slider Thumb Color	Color of the generated normal handle when no handle image is used.
Slider Thumb Hover Color	Generated handle color for its highlighted state.
Slider Thumb Size	Handle size, from 4 to 128 in the typed control.
Slider Thumb Glow	Soft halo for the generated fallback handle.

Keep the handle easy to distinguish from the track. A tiny dark handle on a dark line may look elegant in a screenshot and be frustrating to drag. Give it enough room inside the element’s Height.

> Thumb color and glow generate the fallback handle. They are not a promise to recolor or add a halo to an imported handle image.

Use your own slider-handle artwork

1. Select opacity_slider and open Images.
2. Under Slider Handle, set Slider Thumb Image to Wheel Icon.
3. Set Slider Thumb Hover Image to Key Icon for an obvious test.
4. In Visual Style, set Slider Thumb Size to 30.
5. Run Preview or gameplay. Move over the handle, then drag it.

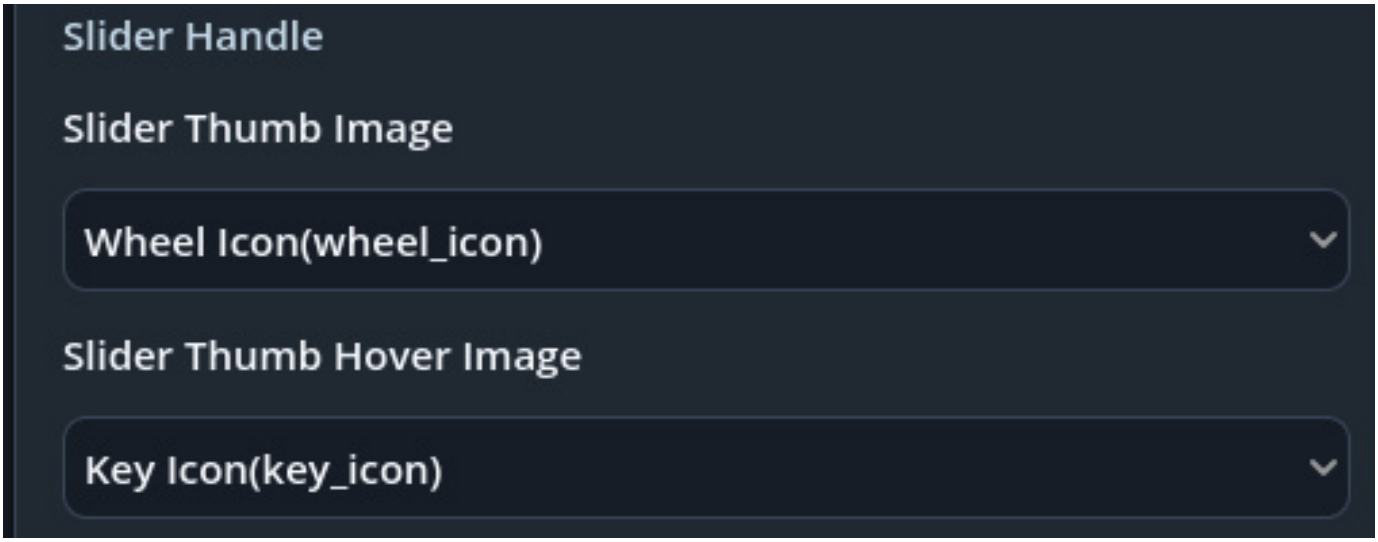

Figure 26. The handle’s two image fields are separate from the control’s general background images.

The imported handle is resized to the requested size while keeping its proportions. The first usable texture frame is used for this handle; it is not a general animated handle sequence. A transparent image with tight bounds is easier to align.

If the hover image is absent, the normal thumb image is the fallback. Keep their visible centers aligned and their apparent size similar unless the size change is deliberate.

Background Image and Hover Image above this section belong to the slider’s backing. They do not replace the draggable handle. There is no separate typed Clicked Thumb Image field in this build.

> Test the two end positions. A handle that extends far beyond the track can appear clipped or make the endpoint look wrong, even when the value itself is correct.

17. Read a slider’s live value

You can use a slider without a variable binding. A separate Button can read its current value, apply it and store it in a game variable.

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

func run(_context: Dictionary = {}) -> Variant:
    var percent: float = float(GetGUIElementProperty(
        "settings_lab", "opacity_slider", "value", 75.0
    ))
    await SetVariable({
        "name": "art_opacity", "value": percent
    })
    await ChangeGUIElementProperty({
        "guiId": "settings_lab",
        "elementId": "settings_art",
        "property": "opacity",
        "value": clampf(percent / 100.0, 0.0, 1.0)
    })
    return true
```

1. Add an Image with ID settings_art and choose a picture for it.
2. Create a Number variable named art_opacity, with initial value 75.
3. Create and apply this script. Add an Apply Button that runs it.
4. In gameplay, move the slider to about 25 and press Apply. The picture should become much more transparent.

The slider uses 0 to 100 because that is easy to read as a percentage. The Image’s opacity uses 0 to 1, so the script divides by 100. The slider changes the value; the script makes that value do something.

Optional: write slider values as they change

A slider can write its changing value to a variable. In 0.86.3.4, this binding property is supported at runtime but does not have a dedicated typed Variable field in the Slider form. Use Raw JSON (Advanced) for it.

1. Create a Number variable art_opacity with initial value 75.
2. Select opacity_slider and open Raw JSON (Advanced). Keep all its existing properties.
3. Immediately after the opening brace, insert the member shown below, with its comma. If variable already exists, edit that value instead of adding a duplicate key.
4. Click Validate and Apply Element JSON (Advanced), then Save Project.
5. Add a Label with Text Live value: $$art_opacity and test it in a running game.

```
"variable": "art_opacity",
```

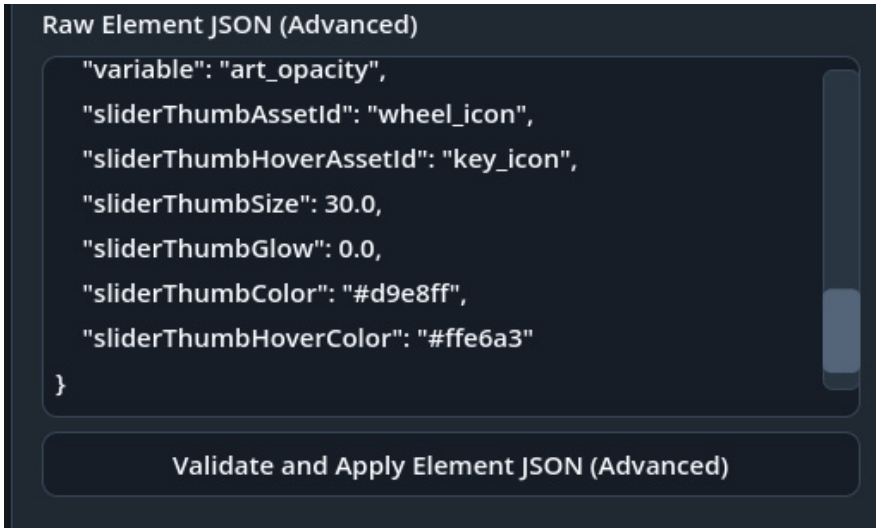

Figure 27. The existing element JSON includes a variable member. The Apply button commits the advanced edit.

During gameplay, moving the slider writes a number and refreshes game state. Designer Preview only moves the local slider; it does not update the real runtime variable.

> This is a write-out binding. Changing art_opacity with Set Variable does not automatically move an already running slider back to that value.

18. Add independent Check Boxes

Use check boxes when several options can be on together. Show subtitles and Show hints are independent choices; enabling one should not turn the other off.

1. Add a Check Box with ID subtitles_check.
2. In Text & Content, enter Show subtitles. Set a readable font size and left-centered text.
3. In Type Settings, turn Checked on.
4. Add another Check Box with ID hints_check and the caption Show hints. Leave Checked off.
5. Run Preview and click each box separately. Try all four on/off combinations.

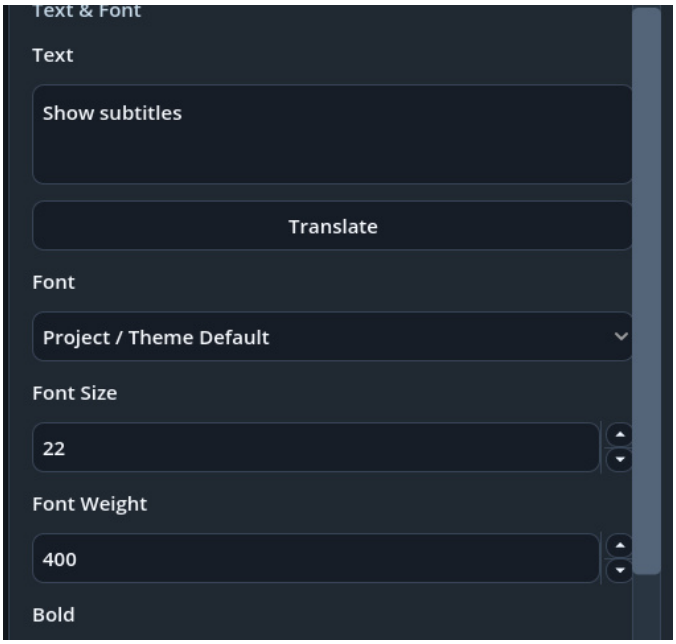

Figure 28. A Check Box’s Text & Content supplies its caption and font style.

A Boolean is a yes/no value: true means on, false means off. Checked is the authored starting state of this control. It does not automatically find a game variable whose name resembles the caption.

> Do not use a Radio Group to make these independent choices. Radio Group is for Radio Buttons that should exclude one another.

Checked state and check-box artwork

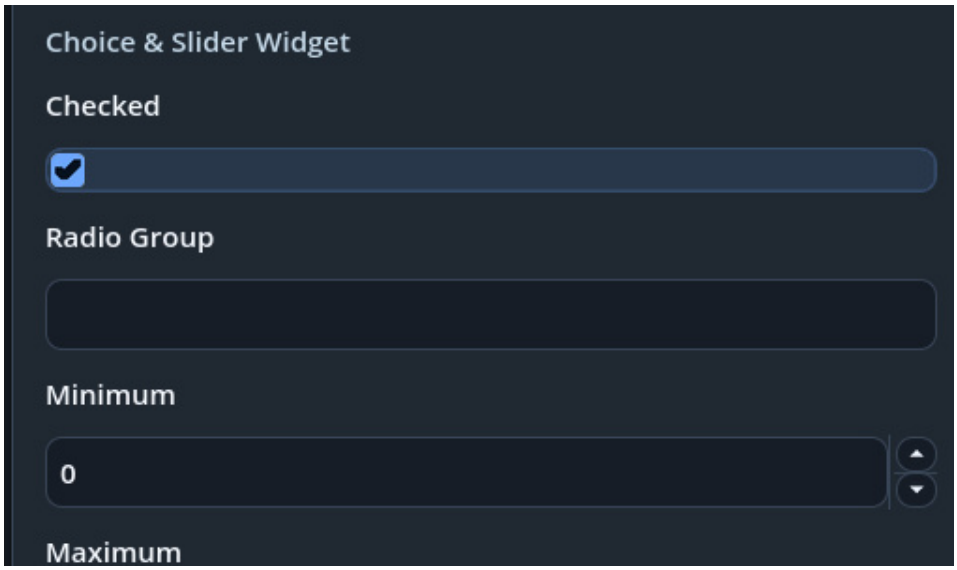

Figure 29. Checked sets the Check Box’s starting state. The shared Minimum, Maximum and Step fields do not turn it into a slider.

The normal Check Box mark shows whether it is on. Text & Content styles the caption. Visual Style controls the surrounding surface, colors and corners. Images can supply supported background and hover artwork for the whole control.

Those background pictures are not dedicated replacements for the native check mark. This build does not expose separate typed Checked Image and Unchecked Image selectors. Keep the check mark visible and use the caption to explain the option.

A Clicked Image, where an element type supports one, is temporary press feedback. It is not the same thing as a Check Box remaining checked after release.

Leave enough width for the mark and the longest caption. The label begins after the choice indicator, so a control that is only as wide as its words can still be too narrow.

> For a short confirmation such as I have read the note, use the checked value in your confirmation script. The box does not block another Button automatically just because its caption sounds like a requirement.

19. Read or bind a Check Box

For a separate Apply Button, read the live button_pressed property. The authored checked property describes construction data; it is not a dependable read of the player’s current choice.

```
var subtitles: bool = bool(GetGUIElementProperty(
    "settings_lab", "subtitles_check", "button_pressed", false
))
await SetVariable({
    "name": "show_subtitles", "value": subtitles
})
```

Put those lines inside a run function based on script_base.gd. If you prefer live variable updates whenever the box toggles, add a runtime binding instead.

1. Create a Boolean variable named show_subtitles with initial value true.
2. Select subtitles_check and open Raw JSON (Advanced). Keep the element’s existing properties.
3. Add the member below immediately after the opening brace, or edit the existing variable member.
4. Validate and Apply Element JSON, save, then test in gameplay.

```
"variable": "show_subtitles",
```

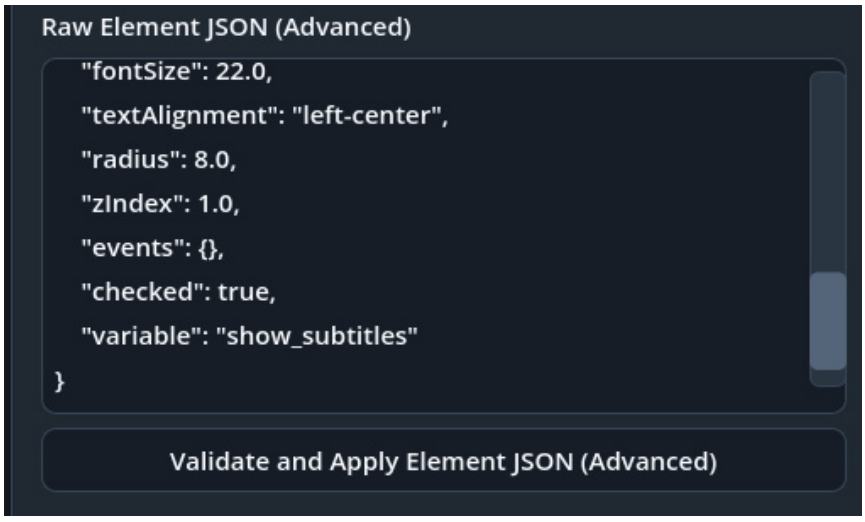

Figure 30. The Check Box’s checked and variable members are distinct. Keep the variable default and starting check mark consistent.

Give Show hints its own Boolean variable, show_hints. Two independent check boxes should not share one output variable unless your logic deliberately combines them.

20. Build a group of Radio Buttons

Use radio buttons for one-of-several choices. Difficulty can be Relaxed, Standard or Challenging; choosing one should clear the other two.

1. Add three Radio Buttons with IDs radio_relaxed, radio_standard and radio_challenging.
2. Set their captions in Text & Content.
3. In Type Settings, enter prefs_difficulty as Radio Group on all three.
4. Turn Checked on only for Relaxed. Leave the other two off.
5. Run Preview and choose Standard, then Challenging. Confirm that only one remains selected.

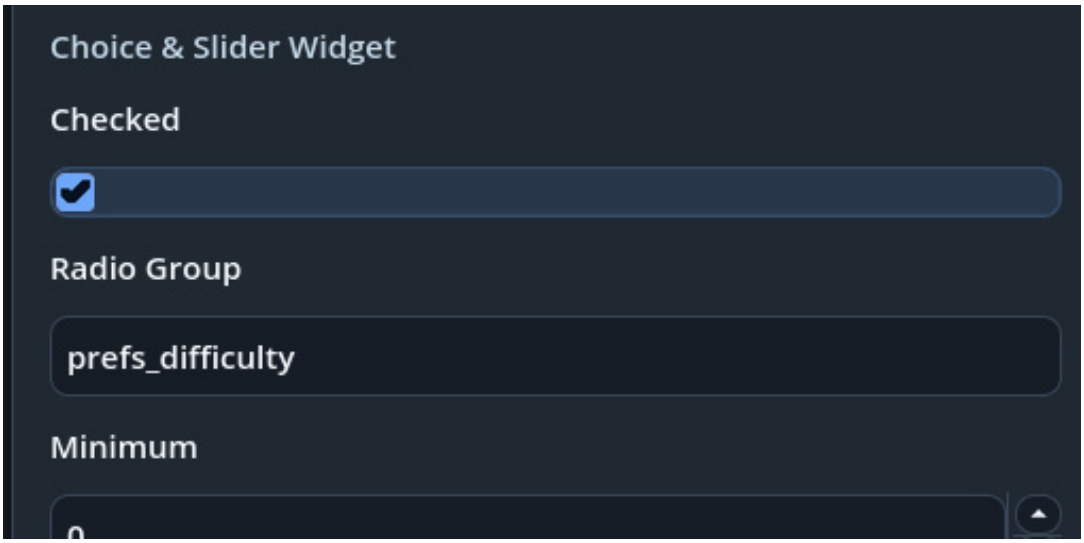

Figure 31. The Relaxed option starts checked. Its Radio Group string links it to the other difficulty options.

Radio Group is a logical relationship between choice controls. It is different from General → Group ID, which is used for GUI grouping and targeting. Giving two controls the same editing Group ID does not make them mutually exclusive.

> Set exactly one deliberate starting choice when the screen needs a default. Several checked entries in one group leave the result dependent on construction order.

Keep unrelated radio groups separate



Figure 32. The Icons option uses prefs_hint_style, a different group from difficulty.

Add two more Radio Buttons for a hint style: Icons and Text. Give both Radio Group prefs_hint_style. Set Icons checked initially. Changing this group should leave the chosen difficulty alone.

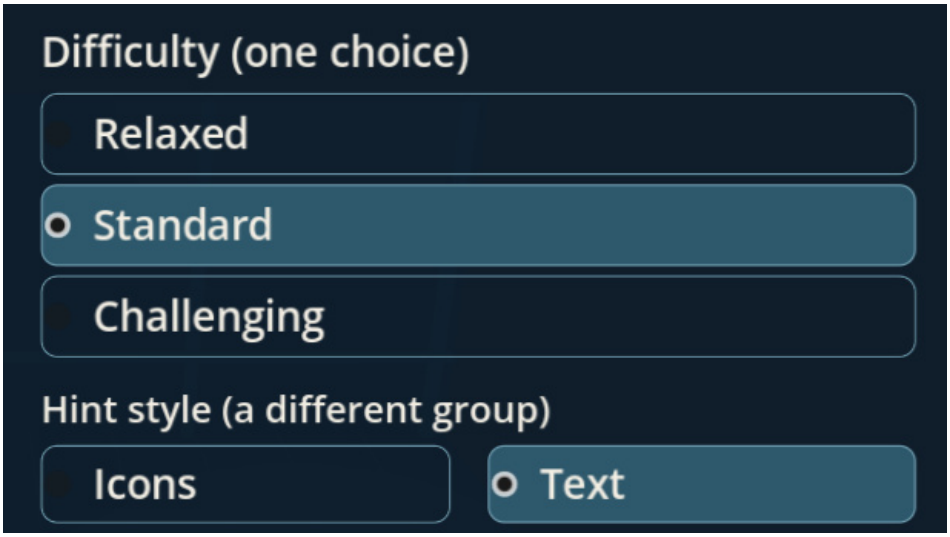

Figure 33. Standard difficulty and Text hint style can both be selected because they belong to different groups.

Group names are shared by the running GUI manager in this build. Reusing the same Radio Group name in another GUI can connect choices that you intended to keep separate. Use meaningful names such as prefs_difficulty, shop_sort_order and map_travel_mode.

> Avoid leaving every Radio Button in the default group. A hidden GUI with the same group name can make another screen’s choice appear to change mysteriously.

Store a radio choice as a stable value

For simple story logic, store one String such as relaxed, standard or challenging after reading the completed group. Do not store a translated caption as the identity of a difficulty mode.

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

func run(_context: Dictionary = {}) -> Variant:
    var mode: String = "relaxed"
    if bool(GetGUIElementProperty(
        "settings_lab", "radio_standard", "button_pressed", false
    )):
        mode = "standard"
    elif bool(GetGUIElementProperty(
        "settings_lab", "radio_challenging", "button_pressed", false
    )):
        mode = "challenging"
    await SetVariable({
        "name": "difficulty_mode", "value": mode
    })
    return true
```

Create a String variable difficulty_mode with initial value relaxed. Run this script from Apply or Continue after the player has finished choosing.

Optional variable bindings on Radio Buttons write Booleans, just like a Check Box. If you use them, give each option its own Boolean, such as mode_relaxed, mode_standard and mode_challenging. Selecting a new option turns the previous one off.

Do not bind all three radio buttons to the same String variable. Their toggle notifications write true and false; they do not automatically write the radio button’s ID or caption.

21. Choose the right event timing

A pointer Click event and a control’s completed value change are not the same moment. In this build, a general Click event can run on mouse press. A Check Box or Radio Button finishes its toggle later, when the normal button interaction completes.

Interaction	Reliable place to read the final choice
Check or uncheck a box	A separate Apply / Continue Button after the click finishes.
Choose a radio option	A separate confirmation Button, or its completed Boolean variable binding.
Drag a slider	Read its live value from Apply, or use its changing numeric variable binding.
Browse a carousel	Read its current entry from a separate confirmation Button.

In the test screen, the Hints box’s Click script saw false at the start of the press, while the finished click left its bound show_hints variable true. Both results describe different moments of the same interaction.

For a settings screen, use the simple pattern: change the controls, then press Apply. For a live setting, use the supported value binding and the game system that consumes that variable.

> The typed GUI Events tab does not provide a general On Value Changed or On Toggled action slot for these controls. Do not substitute the mouse Click event and assume it is a release or drag-finished event.

Keep Clicked Image feedback temporary too. It can acknowledge a press, but it should not be used as the data record of a selected mode or checked option.

22. Apply choices to visible game behavior

Here is a complete Apply script for two boxes and a sample subtitle Label. It works whether or not you added live variable bindings. Create the two Boolean variables before testing.

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

func run(_context: Dictionary = {}) -> Variant:
    var subtitles: bool = bool(GetGUIElementProperty(
        "settings_lab", "subtitles_check", "button_pressed", false
    ))
    var hints: bool = bool(GetGUIElementProperty(
        "settings_lab", "hints_check", "button_pressed", false
    ))
    await SetVariable({
        "name": "show_subtitles", "value": subtitles
    })
    await SetVariable({
        "name": "show_hints", "value": hints
    })
    if subtitles:
        await ShowGUIElement({
            "guiId": "settings_lab",
            "elementId": "sample_subtitle"
        })
    else:
        await HideGUIElement({
            "guiId": "settings_lab",
            "elementId": "sample_subtitle"
        })
    return true
```

1. Add a Label with ID sample_subtitle and some sample speech text.
2. Create this script record, Apply Changes and assign it to an Apply choices Button.
3. Run the game. Turn Show subtitles off and press Apply. The sample Label disappears.
4. Turn it on and apply again. The Label returns.

This demonstration controls one Label. A full game’s subtitle and hint systems must deliberately use the saved preferences; captions on check boxes do not connect those systems automatically.

Combine the small actions into one screen



Figure 34. After Apply: the sample subtitle is hidden, the hint is shown, Standard remains selected and the illustration uses the chosen opacity.

The full example combines the same operations you have already learned: read the two boxes, read the difficulty group, read the slider, store the values, then change the sample display.

Value	Stored as	Effect in this example
Show subtitles	show_subtitles, Boolean	Show or hide sample_subtitle.
Show hints	show_hints, Boolean	Show or hide sample_hint.
Difficulty	difficulty_mode, String	Report the stable mode ID in the result text.
Illustration opacity	art_opacity, Number	Set settings_art opacity after dividing by 100.

You can put the operations in one script, or run the smaller scripts in order from an Action List. For the latter, choose Action List on the Apply Button and add the shared script-running actions for each prepared script.

Test each piece before combining it. If the slider script works alone but the whole screen does not, check the IDs and event order instead of changing the artwork.

23. Restore controls from game data

A live variable binding writes out from a control. It does not automatically turn an existing control back into the value stored by a script or loaded game. Restore both the data and the visible controls deliberately.

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

func run(_context: Dictionary = {}) -> Variant:
    var subtitles: bool = bool(
        Runtime.variables.get("show_subtitles", true)
    )
    var percent: float = float(
        Runtime.variables.get("art_opacity", 75.0)
    )
    await ChangeGUIElementProperty({
        "guiId": "settings_lab",
        "elementId": "subtitles_check",
        "property": "checked",
        "value": subtitles
    })
    await ChangeGUIElementProperty({
        "guiId": "settings_lab",
        "elementId": "opacity_slider",
        "property": "value",
        "value": percent
    })
    return true
```

Notice the different property names. Read the current check mark as button_pressed. Set its authored/rebuilt state using checked. A Slider uses value for both the live read and its constructed value.

The setter can rebuild these controls. Do not restyle a radio button or slider repeatedly while the player is dragging or choosing; that can reset transient state and keyboard focus.

> Setting a control’s constructed checked or value property does not replace the need to store the intended preference. Keep the variable and control consistent rather than depending on a construction-time signal.

Restore a radio group and reset defaults

For a saved difficulty String, restore the group by clearing its choices and checking the intended one. Use a known set of mode IDs rather than accepting any text as an element ID.

```
var mode: String = str(
    Runtime.variables.get("difficulty_mode", "relaxed")
)
if mode not in ["relaxed", "standard", "challenging"]:
    mode = "relaxed"
for option: String in ["relaxed", "standard", "challenging"]:
    await ChangeGUIElementProperty({
        "guiId": "settings_lab",
        "elementId": "radio_" + option,
        "property": "checked",
        "value": false
    })
await ChangeGUIElementProperty({
    "guiId": "settings_lab",
    "elementId": "radio_" + mode,
    "property": "checked",
    "value": true
})
```

These lines belong inside a run function using script_base.gd. Put them after your Check Box and Slider restoration if one script restores the whole screen.

1. Choose Standard and a low opacity in gameplay, then apply them.
2. Change the saved variables deliberately. Observe that Set Variable alone does not redraw the existing choice controls.
3. Run the restore script. Check the visible controls now agree with the stored data.
4. For a Reset Defaults Button, set the variables to the chosen defaults first, restore the controls, then reapply their display effects.

Hiding and reopening a GUI is not a universal reset. Existing widget instances can retain state. A new game run starts again from authored settings and startup logic.

24. Captions and wildcards

Check Box and Radio Button captions use the normal runtime text-expansion path. A separate Label can also display the value of a bound slider or choice variable during testing.

Text or field	What to expect
Check Box / Radio Button Text: Show hints for @PlayerName	The player-name wildcard can expand in gameplay.
Label Text: Opacity: $$art_opacity	A normal Label displays the variable value.
Carousel caption	Comes from an asset or item name and is passed through runtime game-text expansion.
Carousel Empty Text	Also passes through runtime game-text expansion.
Media Asset, image ID lists and Radio Group	Use literal stable IDs or group names. These are not text-template fields.
Slider value / minimum / maximum	Numeric properties, not a place to type a Label wildcard.

A Boolean wildcard may display true or false, which is useful while testing but often too technical for a finished menu. Use a clear caption and the native check mark, or build a player-facing result sentence in a script.

Keep logic tied to IDs. If Standard becomes Normal in another language, difficulty_mode can remain standard. The visible caption can change without forcing you to rewrite story conditions.

> Designer Preview does not provide a running game’s variable state. A literal wildcard in the editing view is not proof that its gameplay expansion is broken.

25. Know what Preview can test

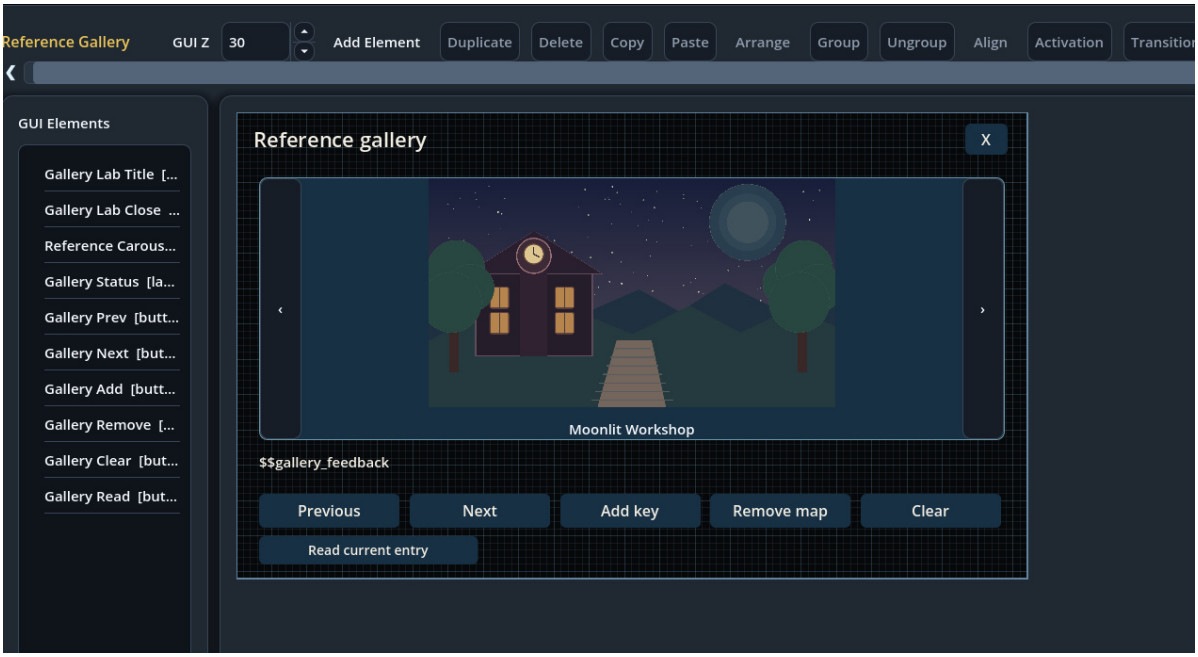

Figure 35. Designer Preview makes the carousel locally interactive while the design remains in the editor.

What you want to check	Where to test it
Image fitting, margins and corners	Designer view, then confirm in the running game.
Hover / pressed image feedback	Designer Preview or running gameplay.
Carousel browsing and card layout	Designer Preview, then test the intended input device in gameplay.
Slider movement; local check/radio changes	Designer Preview for local behavior.
Variable writes and scripts	A running game preview.
Actual carried inventory and filters	A running game with the intended owner and items.

Preview changes are local tests, not a replacement for authoring a starting Value or Checked state. Turn Preview off before selecting, moving or resizing controls.

> A gallery can look correct while its gameplay confirmation action is still unassigned. Test the entire path: open the GUI, interact, apply or confirm, close it and return to the room.

26. Save the right kind of state

Save Project stores the interface design: images, initial carousel IDs, layout, authored values and scripts. It does not turn every change in a running widget into permanent project data.

State	Where to keep the durable meaning
A discovered gallery picture	A story variable or data collection identifying the image asset. Rebuild the gallery from it when needed.
A carried item	The actual character inventory.
A settings choice	Boolean, Number or String game variables, plus the intended save/load workflow.
A slider handle position	Restore it from the stored numeric value.
A radio highlight	Restore the group from the stored stable mode ID.
A held/hover image	Temporary visual feedback; do not save it as a permanent choice.

Game variables and device-wide preferences are not automatically the same thing. Decide whether an option belongs to a particular save slot or should follow the player across games, then use the appropriate persistence workflow.

Test a real save/load sequence. Make choices, apply them, save, change them, load, reopen the menu and restore the controls. Check both the stored values and their visible effects.

> A loaded Boolean can be correct while an already constructed Check Box still displays its previous state. That is the reason for the explicit restore steps, not a reason to change the variable type.

27. Make controls comfortable to use

A good settings screen should be understandable before the player touches anything. Keep labels short, put related options together and make each selected state obvious.

Control	Useful checks
Image	State artwork stays aligned; important content is not cropped; decorative images do not unexpectedly block room clicks.
Carousel	At least one navigation method is visible or explained. Captions identify pictures. The empty state is readable.
Slider	The handle is easy to grab. The range and current value are understandable. Both endpoints work.
Radio group	Its heading explains the question. Exactly one intended default is selected. Unrelated groups stay independent.
Check Box	The caption describes what on means. The check mark is visible. The option does not depend on color alone.

Use more space for touch controls. Test long translated captions, high UI scaling and a small window. A Slider or carousel with keyboard options still needs a sensible focus path and clear instructions on the target device.

Changing a theme can alter native indicators, fonts and contrast. Recheck the whole screen after choosing your final theme, especially Check Box marks, radio dots and generated slider handles.

> Do not hide every navigation arrow just because wheel input works on your own mouse. The player may be using a touch screen or another input method.

28. Fix image and carousel problems

Problem	Check this first
The image is blank.	Confirm the asset ID resolves to an imported texture, opacity is visible and margins have not consumed the content area.
The picture is cut off.	Use contain instead of cover if the whole picture must be visible. Check the source’s own transparent bounds.
The background seems missing.	An opaque foreground may cover it. Check Background Image separately from Media Asset.
Hover changes the wrong layer.	For an Image, state images replace its foreground. Composite widgets use common backing artwork.
The pressed picture remains wrong after a click.	Test release inside and outside. Distinguish temporary pressed art from saved checked/selected state.
Animation does not play as expected.	Use image asset IDs as frames, not numeric cell indices. Check the FPS and valid frame assets.
The gallery is empty.	Check Initial Image Assets for a normal carousel; check carried inventory, owner, icon assets and filters for the inventory version.
A neighboring card cannot be clicked.	Browse with supported navigation or external carousel actions; side cards are not separate selection buttons here.
Runtime additions disappeared.	A property edit may have rebuilt the carousel from authored data. Reconstruct temporary entries deliberately.
Clearing inventory cards did not remove items.	Use inventory actions to change the actual carried inventory. Gallery image actions are not inventory removal.

29. Fix slider and choice problems

Problem	Check this first
The slider moves but nothing changes in the game.	Read or bind its value, then apply it to an actual system. A descriptive caption is not a connection.
A bound variable changes but the handle does not move back.	The binding writes out. Restore the Slider value explicitly from the stored variable.
The imported handle ignores its color/glow fields.	Those settings generate the fallback handle. They do not automatically modify imported artwork.
The Check Box looks on but its variable begins false.	Match Checked and the variable’s default, or restore the controls from saved data.
My Click script reads the old check state.	Click can run before the toggle completes. Read from a separate Apply Button.
Reading checked gives the wrong answer.	Read live button_pressed. Set constructed state with checked.
Radio options in separate screens affect each other.	Use distinct Radio Group names. Groups are shared by the running GUI manager.
All radio bindings seem to fight over one variable.	Use one Boolean per radio binding, or store one String mode after confirmation.
A restored variable leaves the old GUI highlight.	Run the explicit control restoration, then reapply display effects.
Nothing runs in designer Preview.	Use a running game for events, scripts, inventory and real variable writes.
The GUI stops working after a style change.	Some property changes rebuild controls. Recheck transient values, keyboard focus and carousel runtime entries.

30. A compact reference for scripts

What you need	Read this	Set or change this
Current Check Box / Radio Button state	GetGUIElementProperty(..., "button_pressed", false)	ChangeGUIElementProperty, property "checked", Boolean value.
Current Slider value	GetGUIElementProperty(..., "value", 75.0)	ChangeGUIElementProperty, property "value", numeric value.
Carousel current entry	Read "entries" and "selected_index"; check bounds.	Browse with CarouselPrevious / CarouselNext.
Normal Image foreground	Asset identity belongs to the element.	property "assetId", value = image asset ID.
Image background	Separate from the foreground.	property "backgroundImageAssetId".
Game variable	Runtime.variables.get("id", default)	SetVariable with name and value.

The identifiers used in this guide

GUI	Important element IDs
image_lab	inspect_image, framed_image, animated_image
gallery_lab	reference_carousel
inventory_lab	carried_carousel
settings_lab	subtitles_check, hints_check, opacity_slider, settings_art
settings_lab radio groups	prefs_difficulty; prefs_hint_style

Keep element IDs distinct across these examples. Supply the owning GUI ID as well as the element ID when using GUI actions and property helpers.

Screenshots and runtime checks: PCA 3D Adventure Studio 0.86.3.4. Artwork shown is from existing engine project assets. No practice files are included.