PCA 3D ADVENTURE STUDIO

Progress Bars, Hints and Journals

Build useful meters, offer a little help and keep track of the adventure

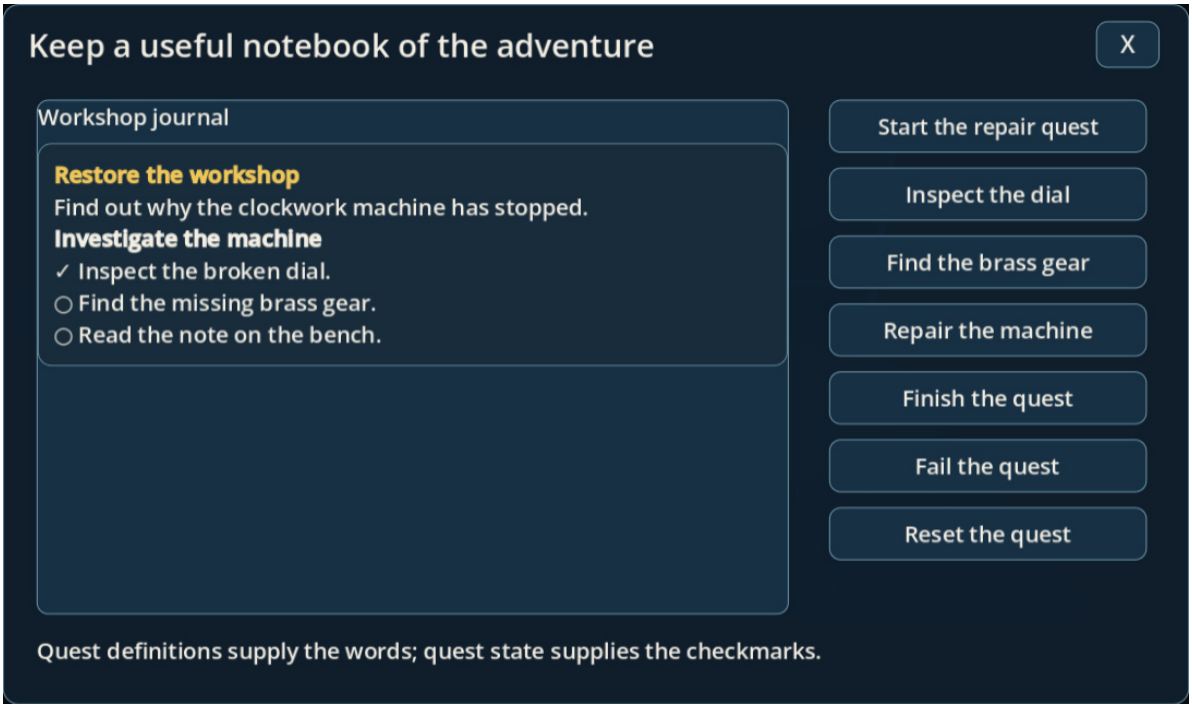

Figure 1. A working journal, with one objective checked off during gameplay.

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 three GUI elements: Progress Bar, Hint List and Journal List. The examples follow a small workshop puzzle. The player investigates a broken machine, finds a missing gear and repairs it. A meter shows the machine’s power, hints offer increasingly direct help, and a journal records the job.

The examples are setups you can build in your own project. No practice project or extra practice files are included. All screenshots were captured from the running editor or its game preview; the guide uses the same instructions across desktop platforms.

You should already know	What this guide adds
How to create a GUI and add elements	The particular fields, data sources and behavior of these three elements.
How a name differs from an ID	Which IDs connect the GUI, variables, hint topics, quests, stages and objectives.
How to assign a GUI event or action list	The actions that update the number, reveal a hint or complete an objective.
Basic script editing, if you want the script examples	Complete small GDScript records and explanations of what each one does.

GUI property changes normally preview live. Raw JSON needs its Validate and Apply button. Scripts need Apply Changes in their editor. Save Project after authoring. A running game is needed to test real actions, variable changes and quest state.

The behavior was checked in 0.86.3.4. Some controls are shared across widget types even when a particular widget does not use them. The relevant chapters call out those cases rather than asking you to hunt for a setting that is not there.

Contents

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

1. Give each element the right job 4
2. Add your first Progress Bar 5
3. Understand the number and the range 6
4. Put useful text on the meter 7
5. Choose the direction and fill 8
6. Change a bar with an action 11
7. Bind the bar to a Number variable 12
8. Run actions at a percentage 15
9. Avoid misleading progress events 17
10. Add a Hint List 18
11. Write the topics and answers 19
12. Unlock a hint with an action 21
13. Make a Give one more hint Button 23
14. Reset or remove hint progress 24
15. Hint appearance, text and limits 25
16. Add a Journal List 27
17. Create the quest definition 28
18. Build the first stage 30
19. Add objectives to the stage 31
20. Start the quest during gameplay 33
21. Complete an objective with an action 35
22. Protect progress from old interactions 37
23. Complete, fail and restart a quest 38
24. Style the journal surface and entries 39
25. Decide which journal entries to show 41
26. Connect all three to the puzzle 43
27. Use a guarded interaction script 44
28. Open and close the right window 45
29. Save the progress, not just the picture 46
30. Know what Preview is proving 47
31. Troubleshoot a Progress Bar 48
32. Progress Bar reference 50

1. Give each element the right job

Element	What it reads	What you supply
Progress Bar (progress)	A current number and a maximum. It can follow a runtime variable.	The number’s meaning, its updates and any threshold actions.
Hint List (hintList)	Authored topics plus the number of answers unlocked for each topic.	The hints themselves and the actions that reveal them.
Journal List (journalList)	Quest definitions and their current runtime state.	The quests, stages, objectives and actions that update them.

A bar does not count tasks or time by itself. A hint list does not work out a solution to a puzzle. A journal does not infer what the player has done by reading its description. Each element displays the information your game supplies.

For the workshop example, power is an ordinary Number variable. The hint topic is machine. The quest is restore_workshop. Those IDs are independent: changing power to 100 does not automatically complete the quest or reveal every hint.

> Use separate, readable names and stable IDs. A selector may display Restore the workshop(restore_workshop), but an action or script uses restore_workshop alone. Avoid giving several GUI elements the same ID, even if they live in different GUIs.

2. Add your first Progress Bar

1. Create a GUI named Machine Power with ID progress_lab.
2. Add a Progress Bar element. Set its ID to power_bar.
3. In General, give it enough room to read comfortably. The example uses X 40, Y 152, Width 1040 and Height 68 inside a 1120 × 660 window.
4. In Type Settings, set Maximum to 100 and Preview Value to 60.
5. In Text & Content, enter Power {percent}%. Use a centered alignment and a readable font size.

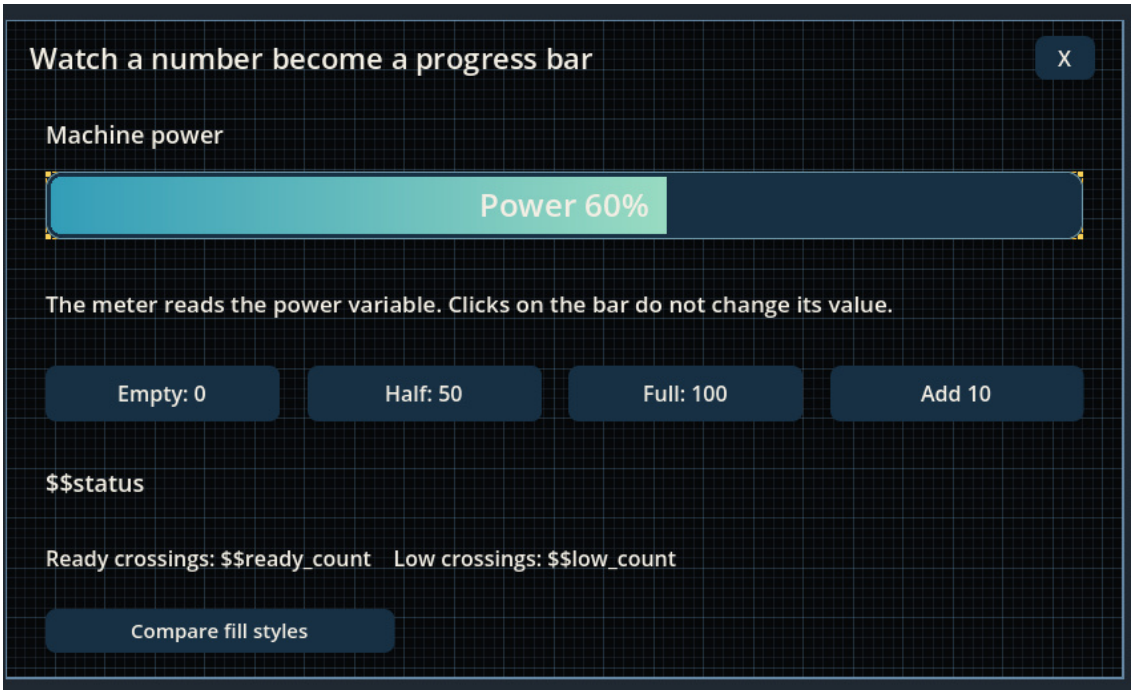

Figure 2. The designer shows a 60% sample before any gameplay logic is connected.

The large dimensions are only the example’s layout. A small HUD meter can be much narrower. Leave room for the longest caption and test at your game’s actual display size.

> Preview Value is a design sample. It is not a live link to a variable and does not make the game start at that value when another runtime value is supplied.

3. Understand the number and the range



Figure 3. Maximum sets the full amount; Preview Value chooses the sample shown in the designer.

Setup	Result
Current 30, Maximum 100	30% full.
Current 3, Maximum 5	60% full.
Current 2, Maximum 3	About 66.67% full; a percentage caption can show the decimals.
Current above Maximum	The displayed bar clamps to full.
Current below 0	The displayed bar clamps to empty.

The bar’s lower end is zero. Maximum must stay positive; this build enforces a small positive minimum. If a puzzle has five parts, Maximum 5 is usually clearer than converting every step to a percentage yourself.

Display clamping does not repair the source variable. In the runtime test, power = 150 displayed a full 100-point bar while the variable still contained 150. Clamp the number in the game logic too when later comparisons depend on it.

> A meter is a display, not a slider. Clicking or dragging across it does not choose a new value. Use separate Buttons, actions or a Slider when the player needs to change the amount.

4. Put useful text on the meter

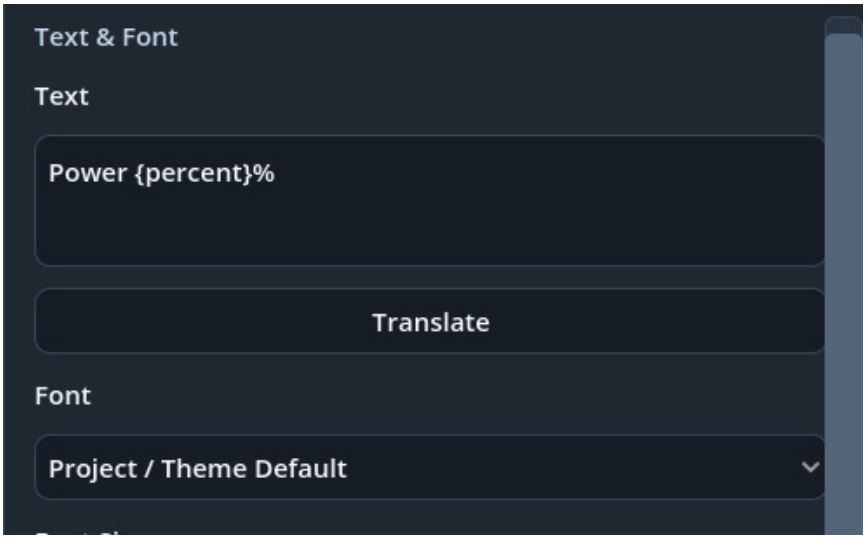

Figure 4. The caption is edited in Text & Content. The brace tokens belong to the Progress Bar.

Caption	At a current value of 60 and Maximum 100
{value}/{max}	60/100
Power {percent}%	Power 60%
Power {percentage}%	Power 60% - the longer token is an alias.
Gears fitted: {value} of {max}	Gears fitted: 60 of 100; use Maximum 5 for a five-gear puzzle.
An empty caption	The fill remains visible without text.

The percent sign is ordinary text. {percent} supplies the number, so include % yourself if you want it. Whole numbers are shown without a decimal part; non-whole numbers use up to two decimal places.

These are the bar’s local tokens. Do not assume that the normal $$variable or @PlayerName expansion used by Labels also runs inside this caption. Use a separate Label for a player name, a long description or other changing text.

> Keep the caption readable over both the filled and empty parts. Test 0%, a middle value and 100%; a color that looks good over one half may disappear over the other.

5. Choose the direction and fill

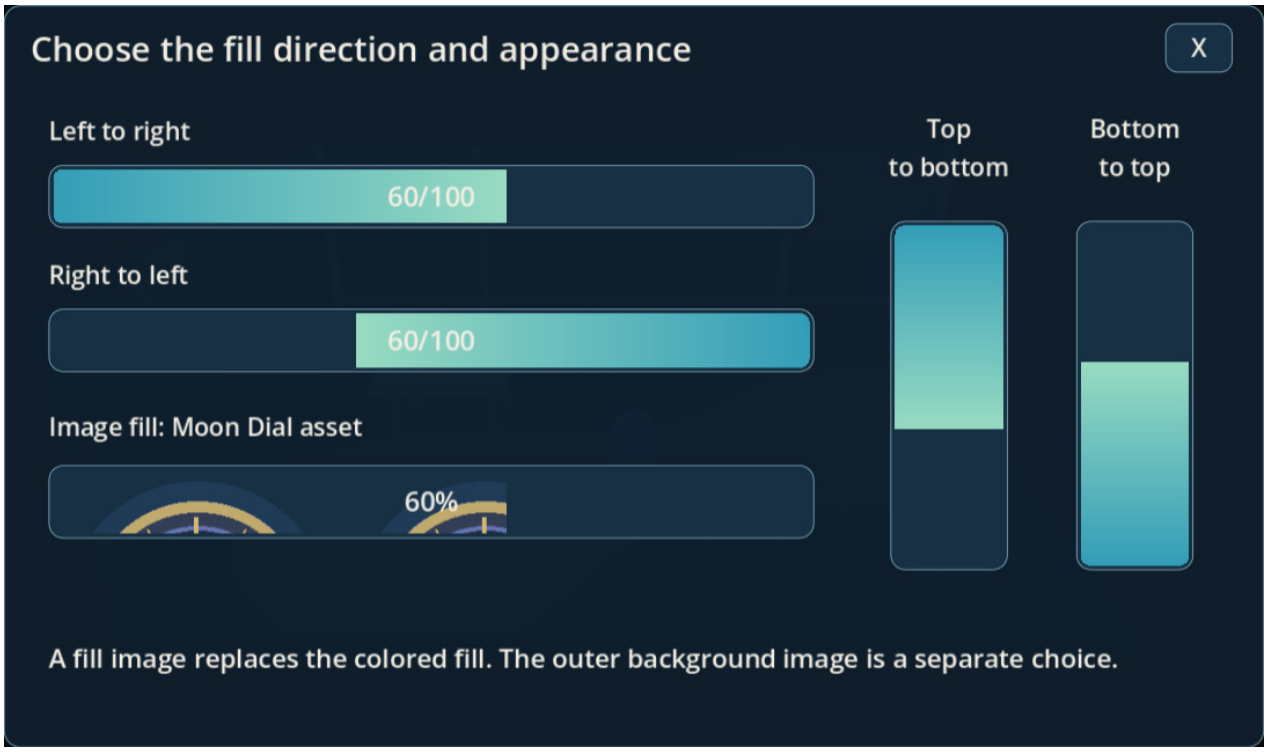

Figure 5. Horizontal and vertical meters use the same value; Fill Direction chooses where the fill starts.

Fill Direction	The full part grows from
left-to-right	The left edge.
right-to-left	The right edge.
top-to-bottom	The top edge.
bottom-to-top	The bottom edge.

Direction does not change the number’s meaning. A right-to-left bar at 60 still represents 60% of its Maximum. Resize the element into a tall shape for a tank, charge tube or vertical gauge; choosing a vertical direction does not resize it for you.

Colors, gradients and image fills

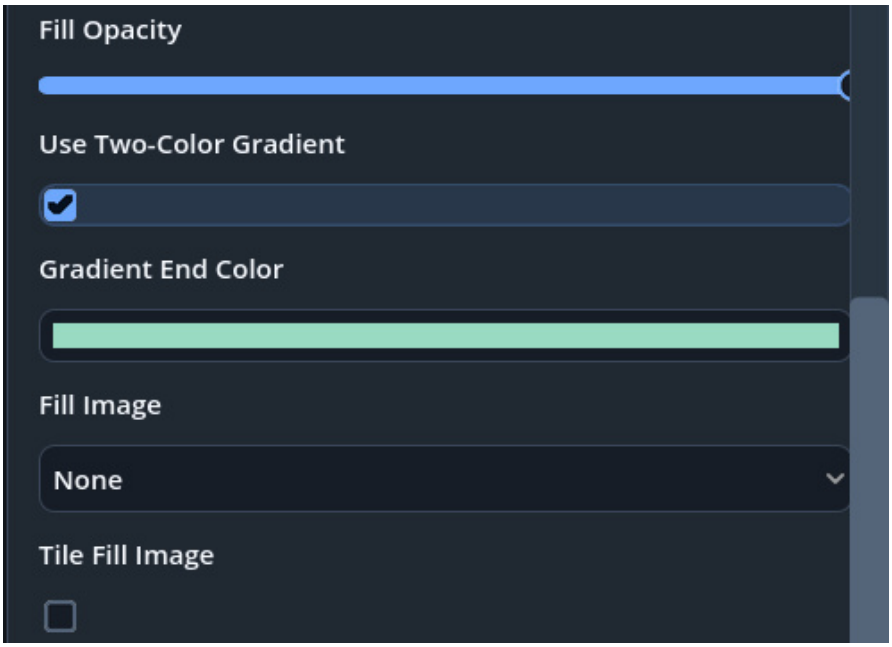

Figure 6. Fill Opacity, gradient controls and Fill Image describe the filled portion of the meter.

For a plain fill, choose Fill Color and leave Use Two-Color Gradient off. For a gradient, turn it on and choose Gradient End Color. Fill Opacity controls the filled part’s alpha. Neither choice changes the current value.

Fill Image loads the first texture from the selected image asset and uses it instead of the colored fill. The example uses the imported Moon Dial art to make the difference visible. Use a simple strip or repeatable pattern for a finished interface; a detailed picture can become hard to read as the meter shrinks.

With Tile Fill Image off, the image is mapped across the current filled rectangle. With it on, the texture repeats. A gradient does not remain underneath an image fill as a second visible paint layer.

> The background image in Images is the outer backdrop. Fill Image in Type Settings is the moving fill. Assigning one does not assign the other.

Borders, padding and corners

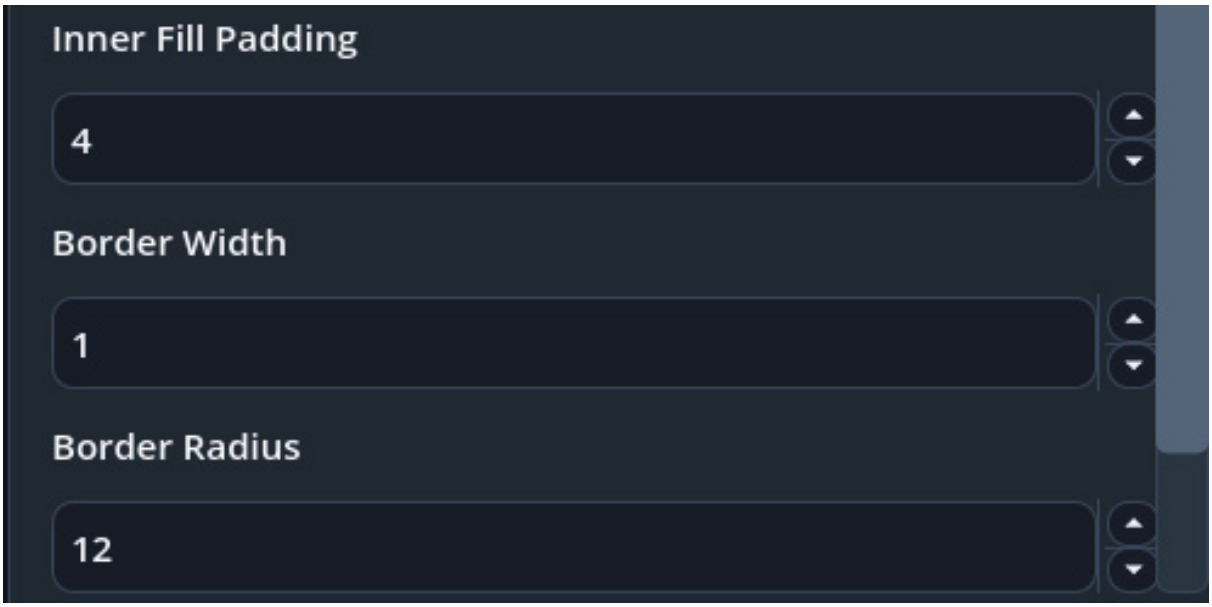

Figure 7. Inner Fill Padding keeps the fill away from the edge. Border Width and Radius describe the outer frame.

Use Inner Fill Padding to leave a small gap inside the meter. Start with 4 on the large example. Large padding on a short bar can leave no useful area for the fill, even though the value is correct.

Border Width and Border Radius here refer to the same authored borderWidth and radius properties used by Visual Style. They are convenient duplicates, not two independent frames. The per-corner fields in Visual Style can override the common radius: -1 inherits it, 0 makes a square corner and a positive number sets that corner’s curve.

The filled part is clipped to the padded rounded shape, and the outline is drawn again over a full fill. Give the empty background and border enough contrast to show where the meter ends.

> Hover and press appearance are separate from progress. A hover image or background color does not mean the player is changing the number. Keep ordinary click events empty unless the meter itself has a deliberate action.

6. Change a bar with an action

For an unbound meter, use Change GUI Element Property. This is the simplest first test: a Button changes one element’s current value.

1. Add a Button beside the Progress Bar.
2. Open its Events tab. Choose Change GUI Element Property for Click.
3. Set GUI to progress_lab, Element to power_bar, Property to value and To to 50.
4. Run the game and press the Button. The meter should become half full when Maximum is 100.

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

func run(_context: Dictionary = {}) -> Variant:
    await ChangeGUIElementProperty({
        "guiId": "progress_lab",
        "elementId": "power_bar",
        "property": "value",
        "value": 50
    })
    return true
```

This is a complete GDScript record for the same operation. Apply it, then assign the record to the intended event or script-running action. The shared action also exists in action lists and visual scripting.

Use this value-based method for an unbound meter. If variable is set on the element, update that variable instead; the next dynamic refresh reads it and can replace a direct value change.

7. Bind the bar to a Number variable

A variable is a better source for power, health, a timer or a puzzle counter. It gives the game one number to save, compare and display. The current typed Progress Bar inspector has no dedicated Variable picker, so this part uses Raw JSON.

1. In the Variable Designer, create a Number variable named Machine Power with ID power and a default of 0.
2. Select power_bar and open Raw JSON (Advanced). Use the tab arrows if it is off-screen.
3. Keep the existing element object. Add "variable": "power" as another property, with a comma separating it from the preceding property.
4. Use "value": 0 for the initial fallback and keep "previewValue": 60 for the designer sample.
5. Click Validate and Apply Element JSON (Advanced), then Save Project.

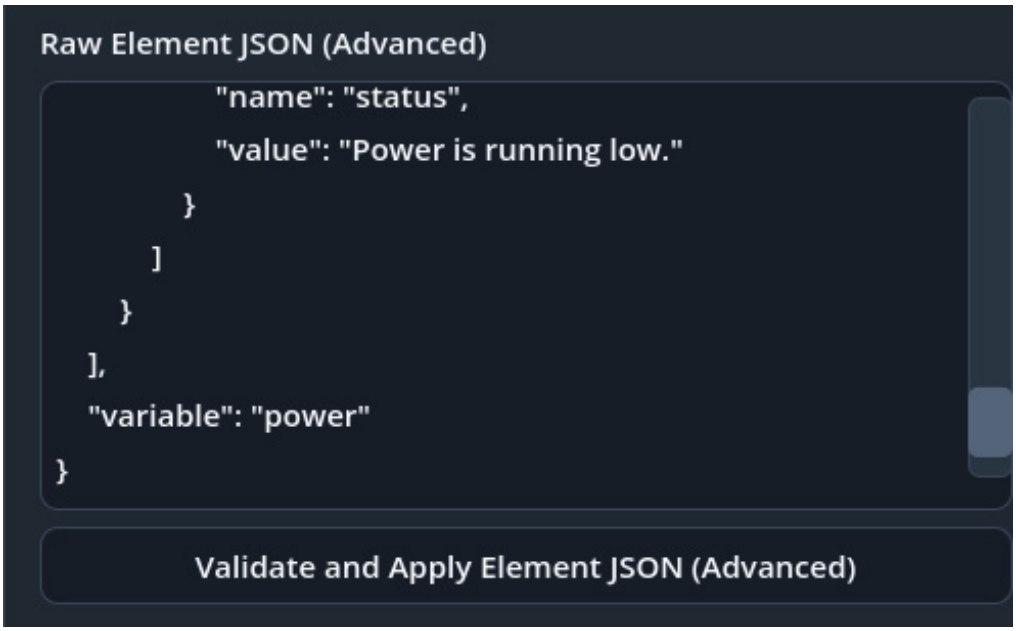

Figure 8. The variable property points to power. Keep the rest of the element JSON when making this edit.

> Do not replace the entire element with a one-line fragment. Its ID, type, layout and style must remain. JSON uses double quotes, commas between properties and no trailing comma after the final property.

Update the source, then test the result

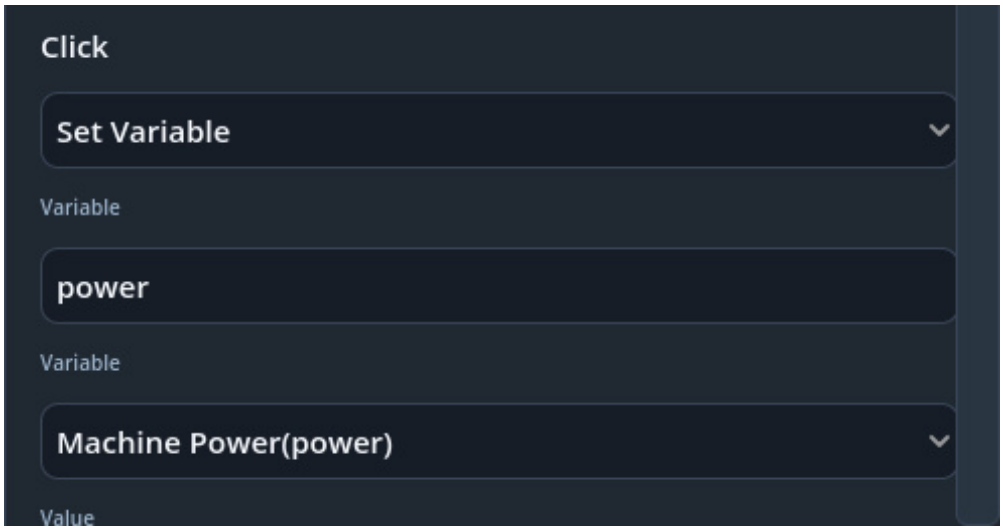

Figure 9. A normal Set Variable action changes the source number. The bound meter follows it.

1. Add an Empty Button that sets power to 0.
2. Add a Half Button that sets power to 50.
3. Add a Full Button that sets power to 100.
4. Run the game, open the GUI and try all three. Compare the actual fill and the caption.

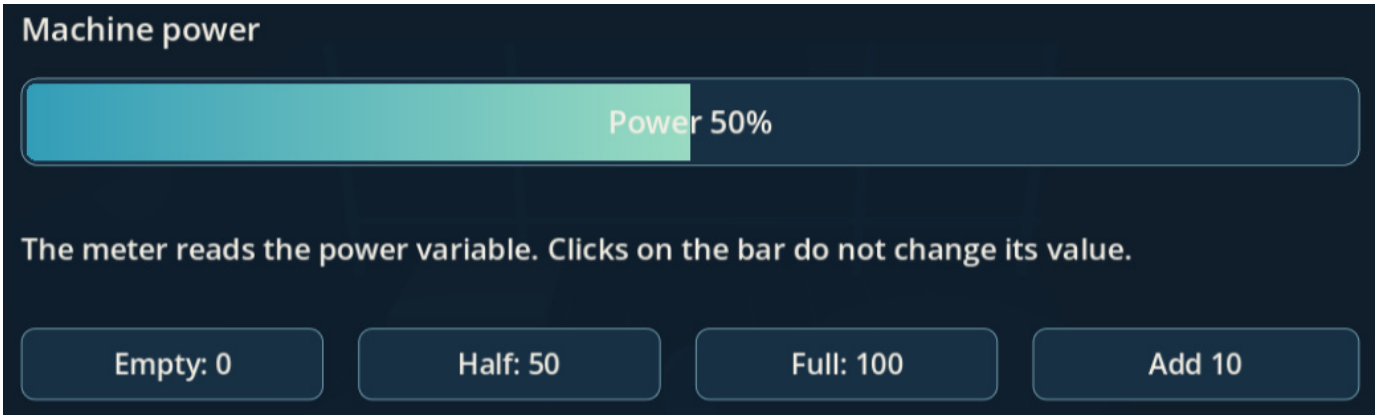

Figure 10. In gameplay, the bound meter reads 50 from power and displays Power 50%.

The designer can still show Preview Value 60 while gameplay shows 0 or 50. That is expected. A missing variable falls back to the element’s authored value, or Preview Value if no value is supplied; a spelling error can therefore look like a meter that is stuck at a plausible number.

Add an amount without overflowing

For a simple increase, Change Variable with operation add is available as an action. It does not automatically cap the number. This complete GDScript record adds ten and keeps the source between 0 and 100:

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

func run(_context: Dictionary = {}) -> Variant:
    var values: Dictionary = Runtime.get("variables") as Dictionary
    var current: float = float(values.get("power", 0.0))
    var updated: float = clampf(current + 10.0, 0.0, 100.0)
    await SetVariable({"name": "power", "value": updated})
    return true
```

1. Create and apply a script record with ID add_power.
2. Add an Add 10 Button. In Events, choose run-visual-script and target add_power. That behavior can run a GDScript record as well.
3. Press it several times in a running game. The meter and source number should stop at 100.
4. Use the same approach with a negative amount when the machine consumes power.

The Dictionary is the game’s current variables. The script reads power, calculates a new amount and uses SetVariable to notify the game of the change. If you choose Maximum 5, change the upper limit in this script to 5 too.

A progress bar is not a clock. For a countdown, your timer or repeated game logic must change the variable. Do not run an endless, no-wait loop from a GUI click to animate the meter.

8. Run actions at a percentage

Percentage Events JSON lets a meter react when its displayed percentage crosses a boundary. A 75% event means three quarters of Maximum, whether Maximum is 4, 100 or 500.

For a first test, create a Boolean variable called test_once with a default of false. Paste this array into Percentage Events JSON in Type Settings:

```
[
  {
    "id": "power_ready",
    "percentage": 75,
    "direction": "up",
    "once": true,
    "actions": [
      {"type": "SetVariable", "name": "test_once", "value": true}
    ]
  }
]
```

1. Begin below 75%, such as power = 0.
2. Set power to 80 or 100. The upward crossing sets test_once to true.
3. Drop below 75 and cross upward again. With once enabled, that event is suppressed for this live bar instance.

The guide’s screenshot fixture also counts ready and low-power crossings and puts the result in ordinary Labels. That makes repeated-event behavior easy to see without repeatedly opening a message box.

> Percentage Events is an array of event objects. A single object without the surrounding square brackets is rejected. The field reports JSON syntax errors; correct them before leaving the element.

Threshold directions and repeat rules

Field	Meaning
id	A unique name for this event on this bar. Give every once event its own ID.
percentage	The boundary from 0 to 100. It is a percentage, not a raw value.
direction: up	Previously below the boundary, now at or above it.
direction: down	Previously above the boundary, now at or below it.
direction: either	Either crossing. both is also accepted in this build.
once: true	Suppresses further crossings with this ID on the current live bar.
visualScriptId	Optional script record ID to run when the boundary is crossed.
actions	Optional array of shared actions to run for the event.

Remaining above a boundary does not keep firing an upward event. A change from 80 to 90 does not cross 75. A change from 20 to 90 can cross several configured boundaries in one update; records are checked in their authored array order.

A downward warning at 25% needs a move from above 25 to 25 or less. A 0% upward event cannot be reached from below a clamped zero, and a 100% downward event cannot be reached from above a clamped full bar. Choose direction to match the journey you intend.

The script context includes guiElementId, percentage and value. Use a unique script or actions list when different meters need different responses. Avoid writing actions that immediately bounce the same meter across its boundary.

> once is local to the widget instance. It is not a saved story flag. For a reward, cutscene or irreversible puzzle result, also check a saved Boolean variable in the game logic.

9. Avoid misleading progress events

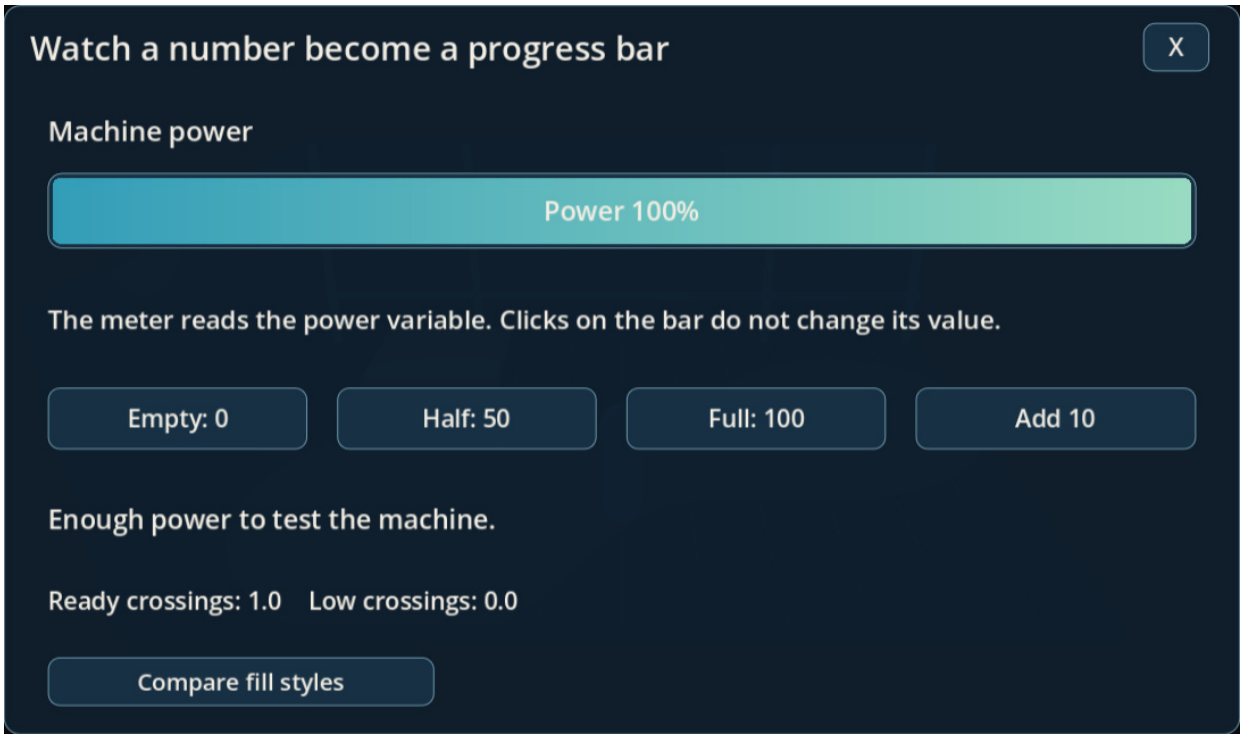

Figure 11. The full meter has crossed the ready boundary. The status and counters are separate Labels.

Changing a bar’s style through Change GUI Element Property reapplies its definition. In this build, that also reapplies the authored value or preview fallback. On a bound bar, a later refresh reads the variable again. Those movements can cross a threshold even though the story’s number did not change.

The runtime check confirmed this: changing Fill Color while power was high caused a down-crossing counter to increase. Set the appearance before active play where possible. Keep important puzzle decisions in the actions that change the source variable, with a saved guard flag, rather than trusting a visual threshold alone.

Construction is another reason to test carefully. Applying a definition initially does not emit thresholds, but a later variable refresh can. Save loading or reopening a screen must not accidentally award a second reward.

> Use thresholds for display feedback such as a warning, color choice or status message. Guard major story consequences with explicit game state, especially if you rebuild or restyle the GUI during play.

10. Add a Hint List

1. Create a GUI called Workshop Hints with ID hint_lab.
2. Add a Hint List element and set its ID to workshop_hints.
3. Give it a generous rectangle. The example uses X 32, Y 92, Width 700 and Height 448.
4. In Text & Content, change Displayed Title to Workshop hints.
5. Open Type Settings and find Topics JSON in the shared Journal Entries section.

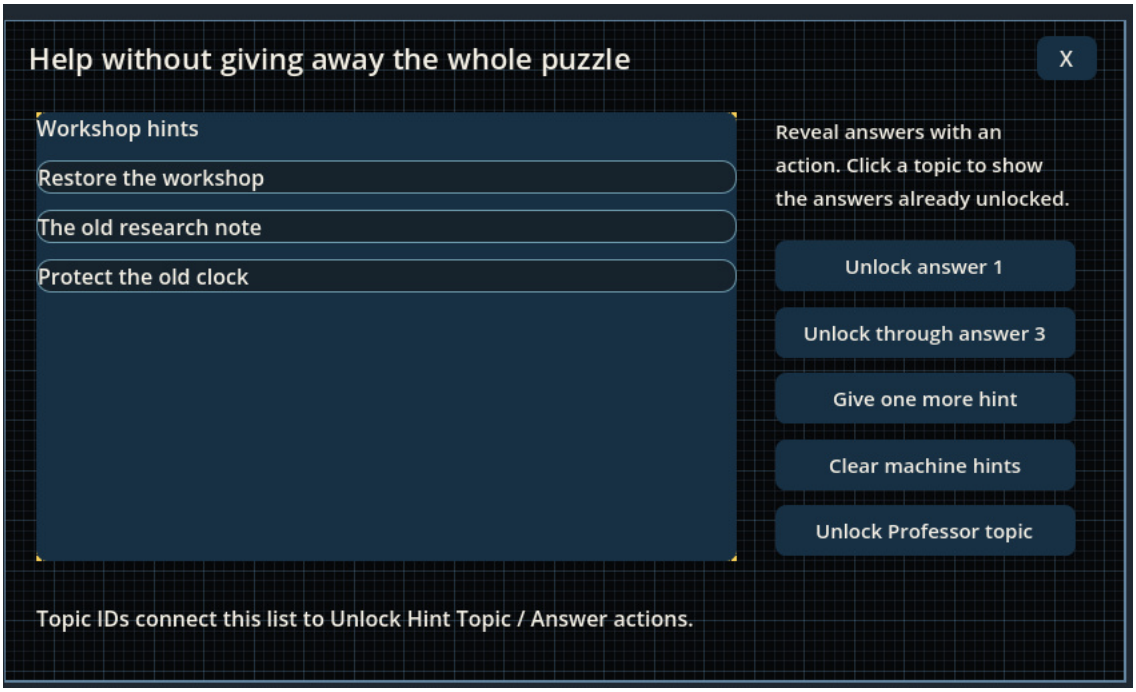

Figure 12. The designer uses a simplified widget preview. These quest titles are not the list of unlocked hints.

In this release, the designer’s Hint List preview uses the same simplified renderer as Journal List. It can show quest titles rather than your hint topics. Use it to place the rectangle, then run the game to test actual topic titles, answers and unlocking.

> The Hint List is supplied with example topic data when created. Replace that example with your own topic IDs and answers before wiring the puzzle.

11. Write the topics and answers

Topics JSON is a list of topics. Each topic needs an id, a displayed title and an answers array. Here is the workshop topic, spaced out so the structure is easy to read:

```
[
  {
    "id": "machine",
    "title": "How do I restart the machine?",
    "answers": [
      "Inspect the dial before touching the controls.",
      "The missing gear belongs in the open housing.",
      "Fit the brass gear, then use the winding handle."
    ]
  }
]
```

Part	Use it for
id: machine	The stable reference used by Unlock Hint actions.
title	The topic Button the player sees.
answers	Plain text strings, ordered from the gentlest hint to the solution.

The inspector field is a single line. Enter the same JSON on one line; line breaks outside quoted text are only formatting. Keep punctuation inside quotes when it belongs to an answer. Use \" for a quote inside a JSON string.

Use an array even for one topic. Give each topic a different ID. An answer should be a string, not an object containing unrelated fields. A misspelled property can leave you with a valid JSON file that displays no useful content.

Start small, then make the hints helpful

For a short first test, this entire example fits on one line and can be pasted directly into Topics JSON:

```
[{"id":"machine","title":"Machine","answers":["Inspect the dial."]}]
```

Once that works, add more answers and use the longer title from the previous page. Add another topic object after a comma, still inside the outer square brackets. The second topic in the screenshots is professor, with the title Who can help me?

Level	A useful workshop answer	Why it helps
1: point somewhere	Inspect the dial before touching the controls.	Gives the player a fresh place to look.
2: explain a connection	The missing gear belongs in the open housing.	Connects two discoveries without spelling out every action.
3: give the answer	Fit the brass gear, then use the winding handle.	Provides a definite solution when the player asks for it.

Keep each answer understandable on its own, but remember that the widget shows every unlocked answer up to the chosen number. The third answer does not replace the first two.

Topic IDs belong to the game’s unlock state, not just this one GUI. If two Hint Lists use machine, they read the same unlock count. Give unrelated puzzles different topic IDs, such as tower_door and workshop_machine.

> Back up the topic text before a large edit. This field does not provide a row-by-row topic editor or a separate Add Answer Button.

12. Unlock a hint with an action

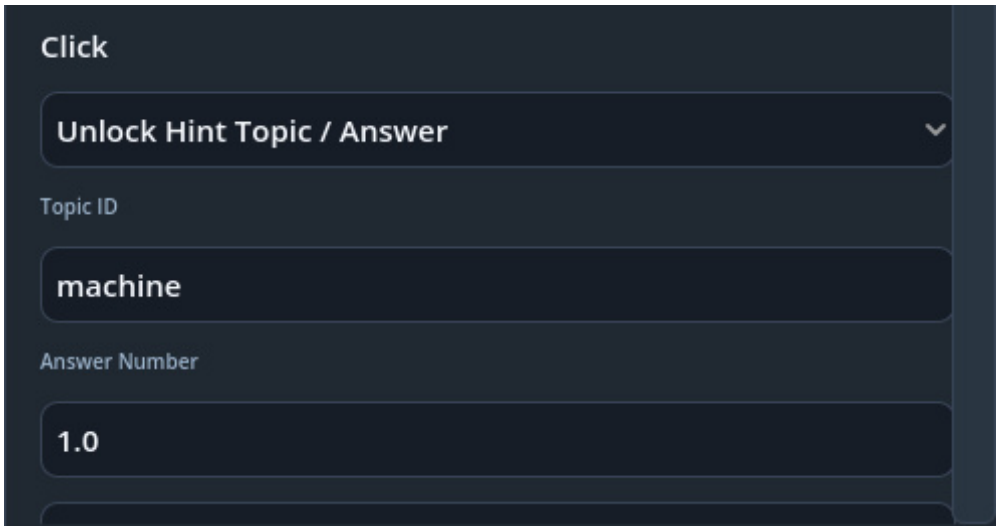

Figure 13. Unlock Hint Topic / Answer uses a topic ID and an answer number.

1. Add a Button with the caption Unlock answer 1.
2. In Events, choose Unlock Hint Topic / Answer for Click.
3. Set Topic ID to machine and Answer Number to 1.
4. Run the game and open the hints GUI. Press the Button, then click the machine topic.

Action request	What becomes available
machine, answer 1	The topic and its first answer.
machine, answer 3	Answers 1, 2 and 3 together.
machine, answer 1 after answer 3	Still 1 through 3; it does not reduce the unlocked count.
A number larger than the answer list	The widget can only display the answers that exist.

The same shared action can run from an object’s verb interaction, an action list or a visual-script node. It does not need to come from a Button inside the hints GUI.

> UnlockHint does not choose a GUI or element. It changes the global hintsUnlocked dictionary using Topic ID. The Hint List then matches that ID against its own Topics JSON.

What clicking a topic actually does

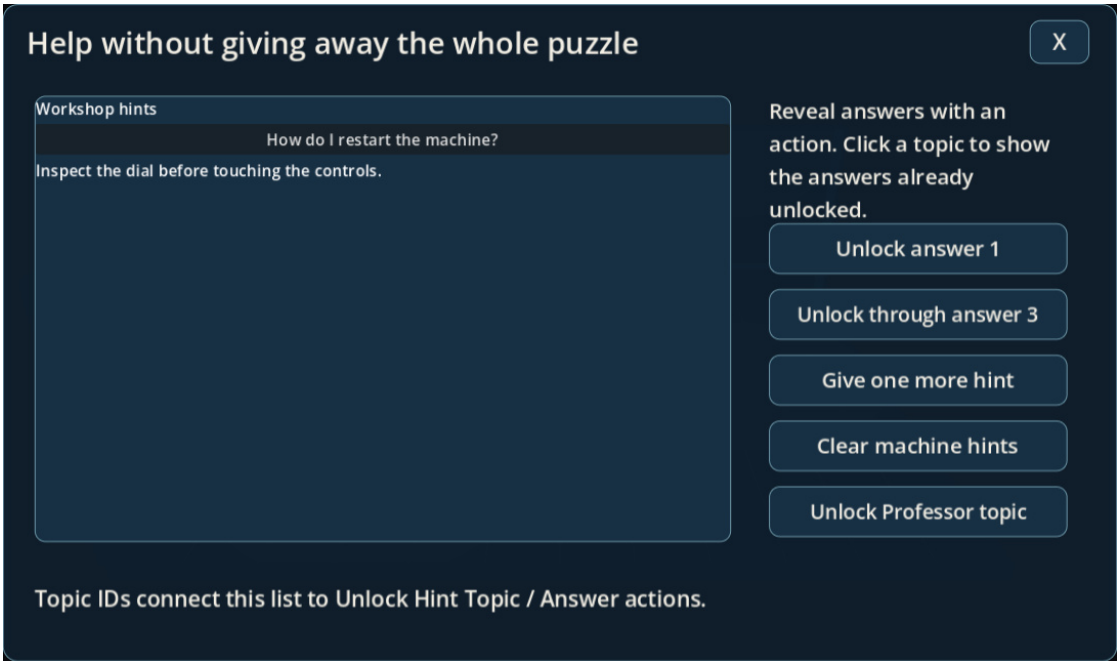

Figure 14. Answer 1 has been unlocked and expanded under its topic.

The topic Button expands or collapses the answers that are already available. Clicking it does not unlock the next answer. That separation lets you decide whether hints are earned through exploration, requested with a More help Button or made available from the start.

An unlock or another dynamic refresh rebuilds the topic rows in this version. Expanded answers can collapse again. The unlocked count remains; click the topic again to reopen it. The fold-open state is presentation, not puzzle progress.

An unlock count of zero is a special case: the topic can appear because its ID exists in the dictionary, but it has no answer text to show. Use 1 for the first useful answer. Remove the topic’s dictionary entry if you want it to disappear entirely.

> A Hint List can be empty for several reasons: no unlocks, a topic-ID mismatch, invalid JSON or an empty topic array. Check those in that order before changing colors or moving the element.

13. Make a Give one more hint Button

Create a GDScript record with ID next_hint. This version advances the machine topic by one answer each time and stops at three:

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

func run(_context: Dictionary = {}) -> Variant:
    var values: Dictionary = Runtime.get("variables") as Dictionary
    var unlocked: Dictionary = values.get("hintsUnlocked", {}) as Dictionary
    var next_answer: int = mini(3, int(unlocked.get("machine", 0)) + 1)
    await UnlockHint({
        "topicId": "machine",
        "answerNumber": next_answer
    })
    return true
```

1. Apply Changes in the script editor.
2. Add a Button called Give one more hint.
3. In Events, set Click to run-visual-script and choose next_hint as the target.
4. Test repeated clicks in gameplay. Reopen the topic after each refresh to read the available answers.

hintsUnlocked stores the highest unlocked answer number for each topic. If machine is absent, the script treats the count as zero. Change both machine references for another topic, and replace 3 with that topic’s actual number of answers.

> This Button deliberately gives help every time it is pressed. If you want a confirmation before revealing the final solution, add that choice to the surrounding GUI or script.

14. Reset or remove hint progress

The Unlock Hint action is designed to increase progress. A smaller answer number does not take hints away. To reset a particular puzzle, copy the dictionary, remove its topic ID and write it back with SetVariable:

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

func run(_context: Dictionary = {}) -> Variant:
    var values: Dictionary = Runtime.get("variables") as Dictionary
    var unlocked: Dictionary = (
        values.get("hintsUnlocked", {}) as Dictionary
    ).duplicate(true)
    unlocked.erase("machine")
    await SetVariable({"name": "hintsUnlocked", "value": unlocked})
    return true
```

Use a dictionary copy so that you make one deliberate state change. Other topics, such as professor, remain available. For a complete restart of the hint system, SetVariable can assign an empty dictionary to hintsUnlocked.

There is no dedicated Lock Hint action in this build. A reset belongs in a new-game or deliberate puzzle-reset path, not in the normal Close Button. Closing and reopening the GUI should not make the player ask for every hint again.

> hintsUnlocked is a Dictionary, not the string "{}". In an advanced value editor or script, pass an actual object. Do not replace it with a Number or text variable.

15. Hint appearance, text and limits

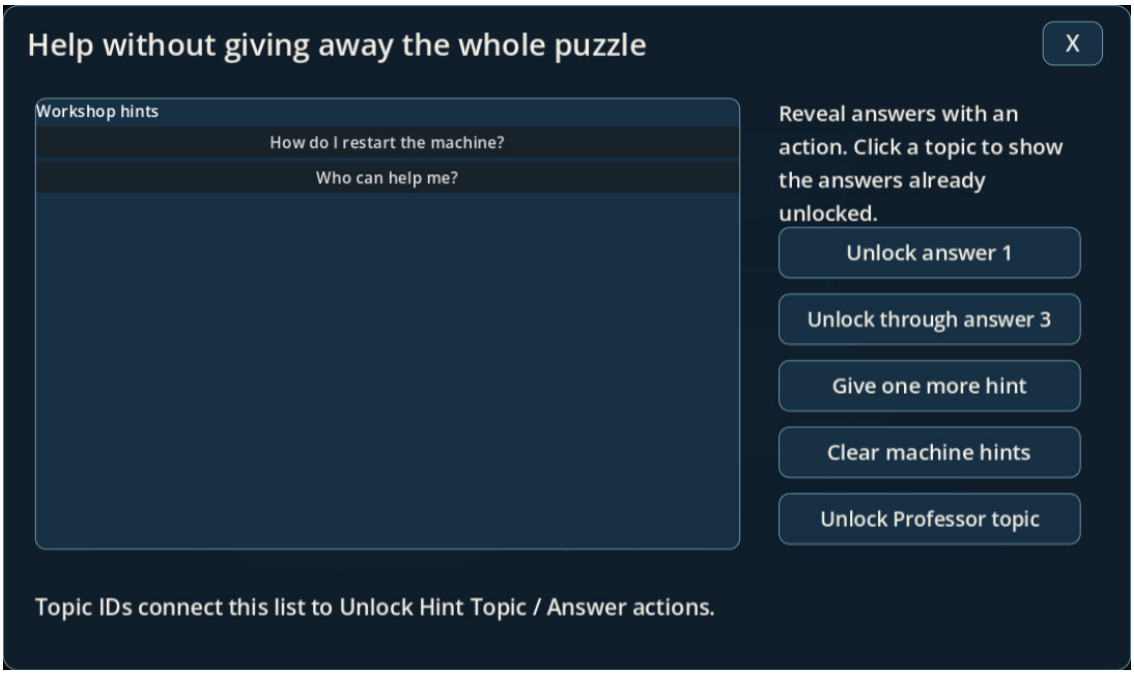

Figure 15. Two unlocked topics appear together. Each reads its own unlock count.

The outer surface uses the element’s background, image and border appearance. The runtime hint renderer builds ordinary Buttons and Labels inside a ScrollContainer. It does not provide the Inventory or Save Widget’s separate grid, slot and screenshot image regions.

The shared Journal Entries section also displays Entry Border Opacity. That is a Journal List entry setting; the runtime Hint List does not build those journal-entry panels. Likewise, this version does not apply all authored font settings to the internal hint Buttons and answer Labels. Check the running result rather than relying on the designer’s simplified preview.

Keep answers short enough to wrap comfortably and test the longest topic title. If you need fully art-directed hint cards, custom disclosure icons or exact per-answer typography, build that presentation with ordinary Buttons and Labels and keep the same saved unlock logic.

> Do not put an Unlock Hint action on every click of the entire Hint List. A click intended only to fold an answer could then reveal more help than the player asked for.

Hint wildcards and translated text

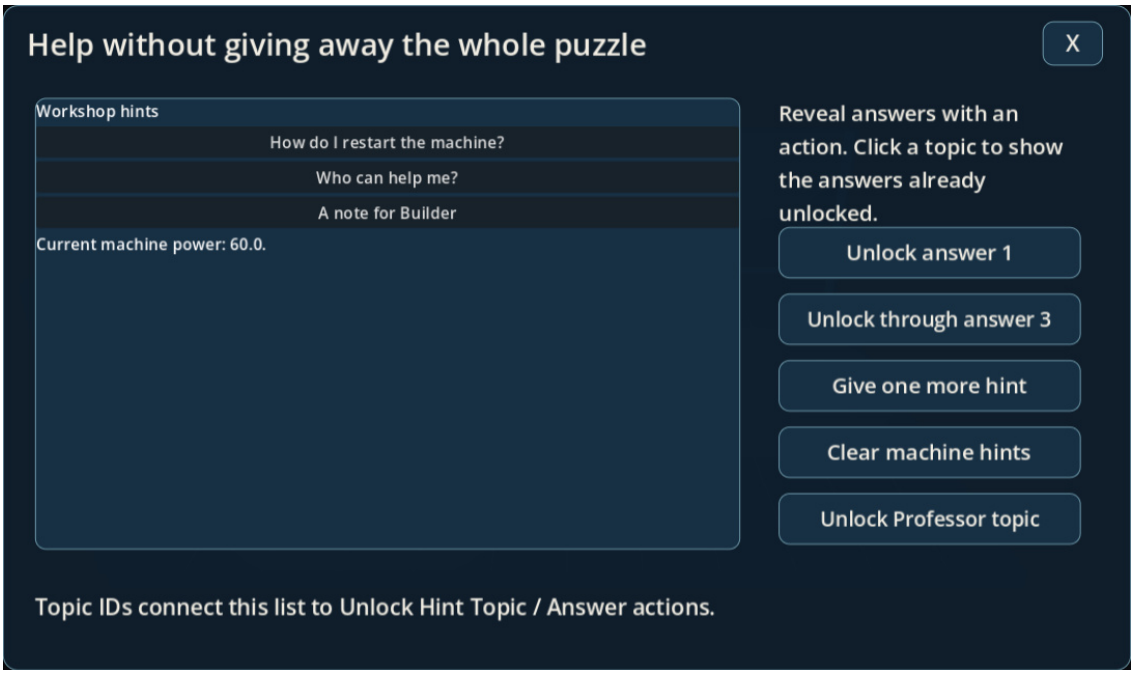

Figure 16. This topic title uses the player name; its answer reads the power variable.

The runtime expands the displayed hint title and answer text. For example, A note for @PlayerName can use the active player’s name, and Current machine power: $$power. can show the current source number. These are normal game-text expansions, unlike the Progress Bar’s local brace tokens.

A state refresh can rebuild the hint rows when a variable changes, so an expanded answer may close. The new text appears when it is opened again. Keep changing telemetry in a separate Label if the player needs to watch it continuously.

Topics JSON is stored as one string. The automatic translation scanner in this release does not unpack that field into individual topic and answer entries. Do not assume every embedded sentence will appear automatically in the translation export. Check the catalog and the target language in gameplay; keep topic IDs unchanged when supplying translated content.

> Translate player-facing words, not IDs, JSON property names, wildcard names or action names. Test actual accented text, wrapping and the longest translation in the real hint window.

16. Add a Journal List

1. Create a GUI called Workshop Journal with ID journal_lab.
2. Add a Journal List element. Set its ID to quest_journal.
3. Give it room for several paragraphs and objectives. The example uses X 32, Y 90, Width 710 and Height 485.
4. In Text & Content, set Displayed Title to Workshop journal and choose a readable font size.
5. Add a Close Button using Hide GUI for journal_lab. Opening behavior is covered later.

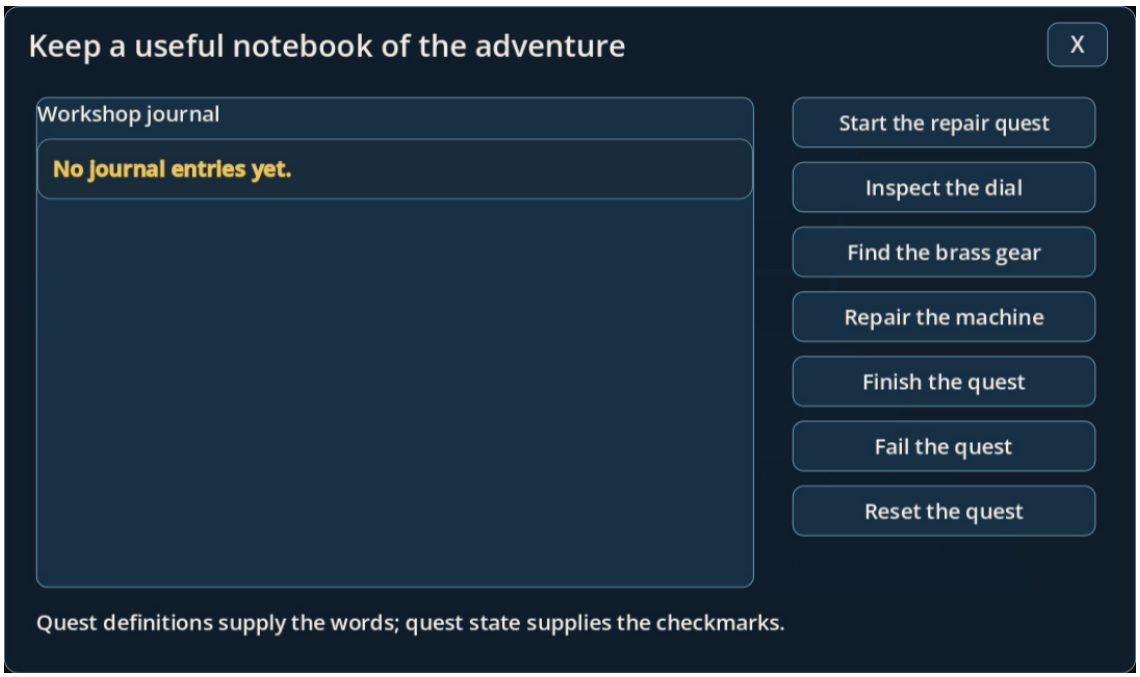

Figure 17. A running Journal List with no started quest states displays an empty entry.

The Journal List reads quests; you do not type its entries into a list of rows in the GUI Designer. In the designer, the simplified preview may show all authored quest titles. In a running game, a quest needs a runtime state before this widget can display it.

> The element ID is quest_journal. The GUI ID is journal_lab. The quest ID will be restore_workshop. Keep those three references distinct when choosing targets or writing scripts.

17. Create the quest definition

1. Open the Quest Designer and use Add New.
2. Set ID to restore_workshop and Title to Restore the workshop.
3. Set Description to Find out why the clockwork machine has stopped.
4. Set Category to Workshop if you want to organize the definition.
5. For this example, turn Hidden on and AutoTrack off so it does not automatically start when the project begins.
6. Leave Repeatable off, then Apply Changes.

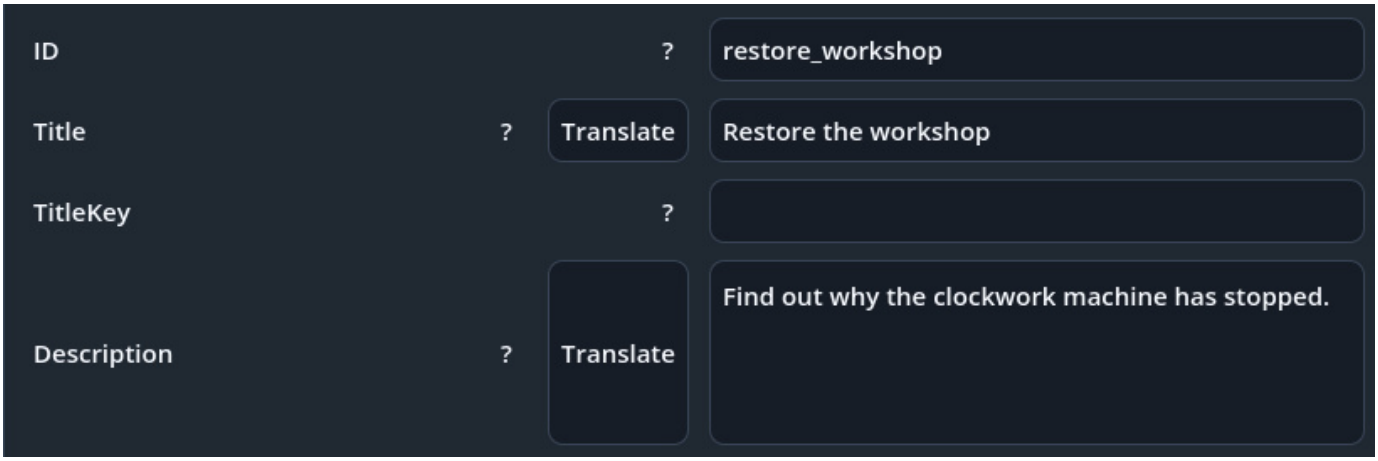

Figure 18. The quest’s General tab supplies the entry ID, title and description.

Title is player-facing text; ID is the action reference. The optional TitleKey and DescriptionKey fields are available, but writing a key is not a substitute for checking actual translated output. The journal-text limitations are explained later.

What the quest’s startup options mean

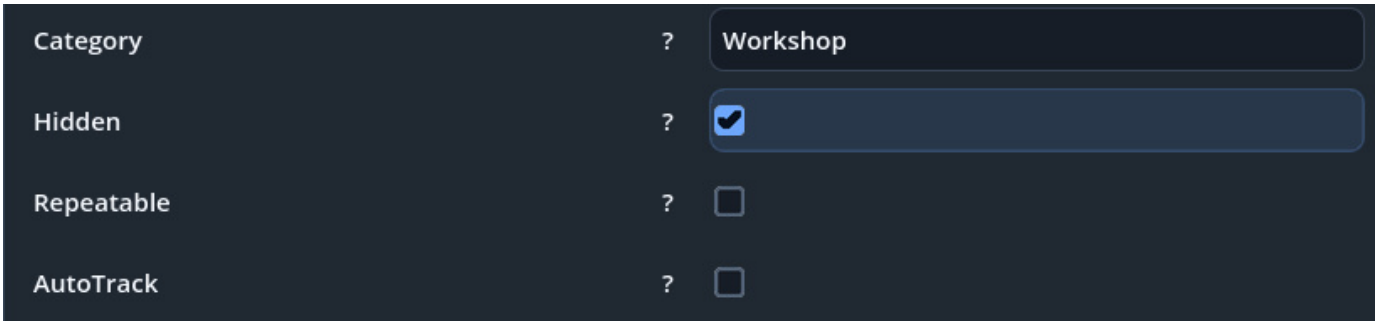

Figure 19. Hidden, Repeatable and AutoTrack have different jobs.

Option	Behavior in this build
Hidden off	The quest manager starts a state for the quest during setup.
AutoTrack on	Also causes a state to be started, even if Hidden is on.
Hidden on and AutoTrack off	No initial state is created. Use Start Quest at the intended story moment.
Repeatable on	Starting a completed or failed quest again resets it for another attempt.
Category	Organizes the definition. The GUI Journal List does not supply a category filter.

The combination matters. Hidden by itself does not defer a quest if AutoTrack is still on. For a job the player learns about later, use Hidden on and AutoTrack off, then explicitly start it in the discovery interaction.

The GUI Journal List currently differs from the built-in journal in how it handles Hidden and Tracked. Once a state exists, this custom widget does not use those flags to filter the entry. The filter chapter shows the supported controls and the limitation.

> Repeatable is about accepting a new attempt after a finished state. It does not repeat a quest continuously, and it does not automatically reset the related hint counts, power variable, inventory or scene objects.

18. Build the first stage

1. Open Stages & Objectives in the Quest Designer.
2. In Stage Actions, click Add.
3. Set Stage ID to inspect and Stage Title to Investigate the machine.
4. Use Stage Description for your authoring notes or other quest uses. This Journal List does not show that description as a separate paragraph.
5. Click Apply Stage Fields, then Apply Changes for the quest.

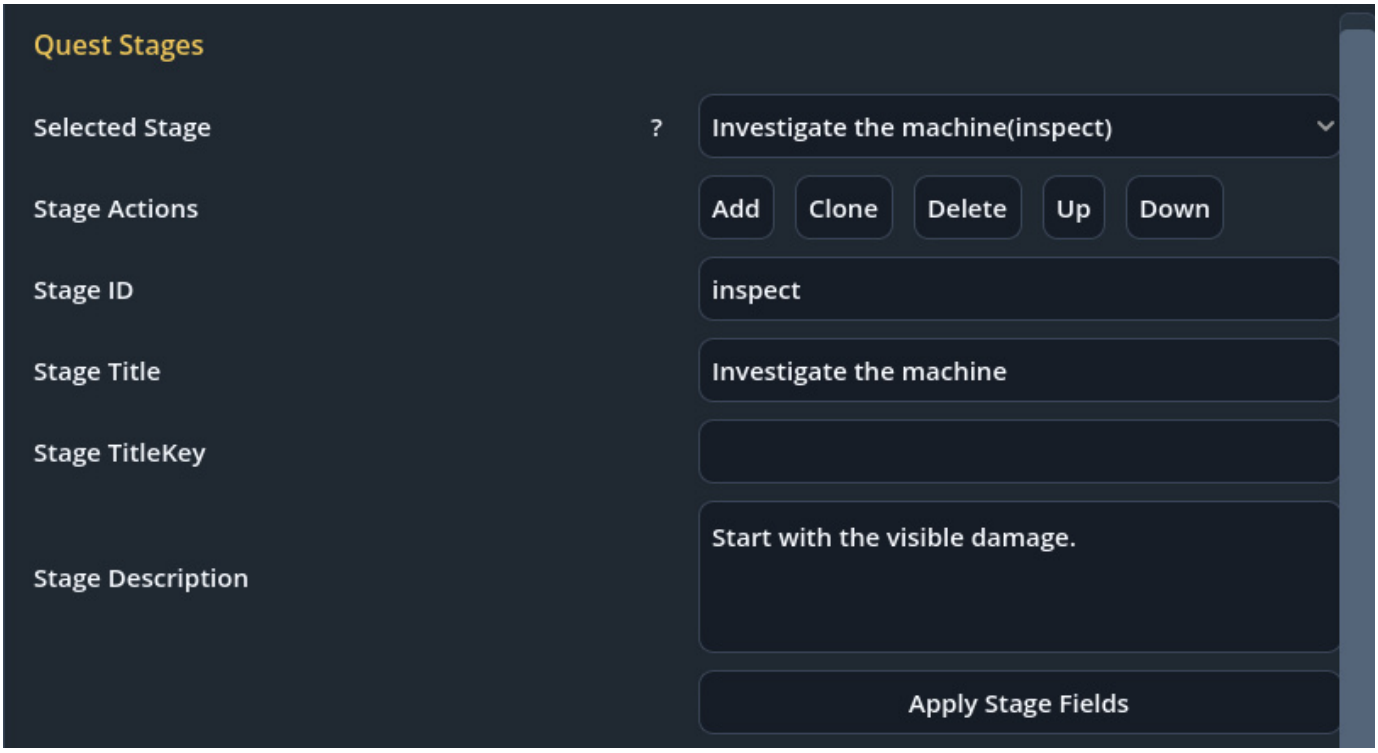

Figure 20. Stages are ordered steps in the quest. Apply Stage Fields stores the selected stage’s edits.

The first stage is index 0 internally, but actions use its ID, inspect. Add a second stage with ID repair and title Put the mechanism right. Use Up and Down to set their order: inspect first, repair second.

> Do not rename stage IDs casually after wiring actions. A title can change for readability; an action targeting inspect still needs a stage with the ID inspect.

19. Add objectives to the stage

1. Select the inspect stage.
2. In Objective Actions, click Add.
3. Set Objective ID to inspect_dial and Objective Text to Inspect the broken dial.
4. Leave Optional and Hidden off. Click Apply Objective Fields.
5. Add find_gear with the text Find the missing brass gear.
6. Optionally add read_note with the text Read the note on the bench. Turn Optional on for this one.
7. Click Apply Changes for the quest, then Save Project.

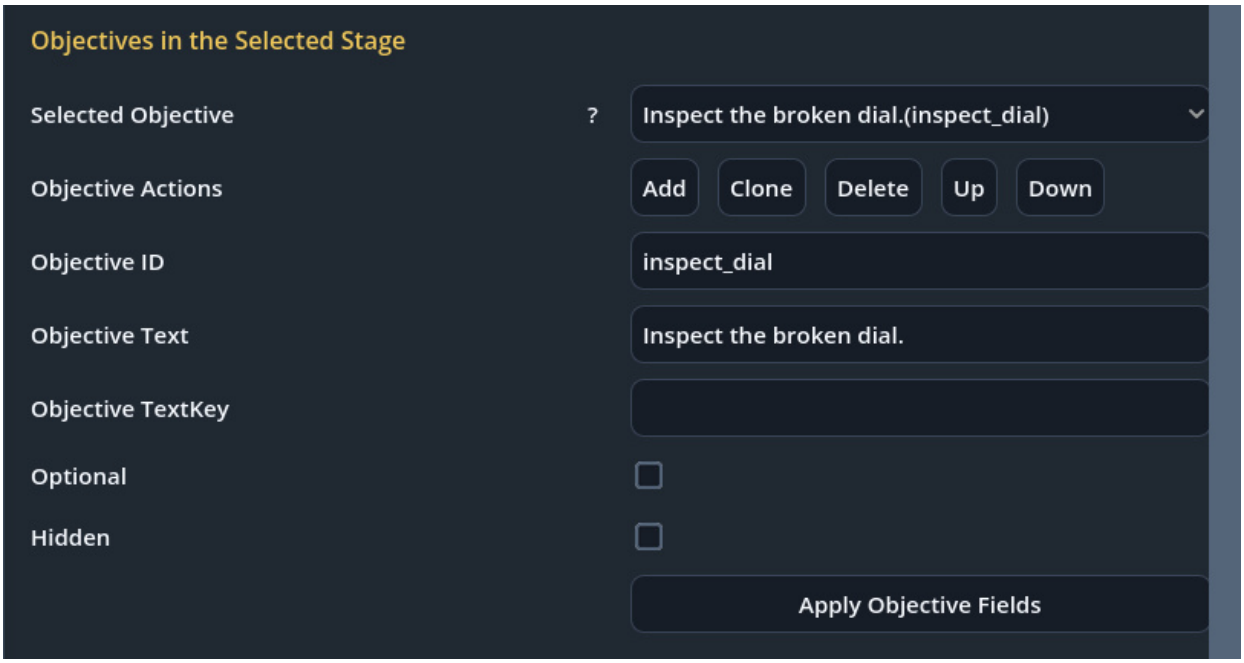

Figure 21. Objective Text is the line the player reads. Objective ID is the reference an action completes.

For repair, add repair_machine: Fit the brass gear and wind the machine. Add test_machine: Test the repaired mechanism. These two are required. The IDs only need to make sense within this quest, but using distinct IDs across stages makes troubleshooting much easier.

Save each layer of quest editing

Control	What it commits
Apply Stage Fields	Edits to the selected stage inside the quest editor draft.
Apply Objective Fields	Edits to the selected objective inside that draft.
Apply Changes	The quest record back into the project document.
Save Project	The updated project on disk.

When a title or objective seems to revert, check which layer was last applied. Edit one stage, apply it, select the next and confirm the first still looks right. Do the same for objectives before you begin wiring actions.

Optional means that completing the stage does not wait for that objective. Hidden means the line is not displayed in this widget. Those are separate choices: a hidden, required objective still prevents automatic advancement until game logic completes it.

The runtime Journal List shows the current stage’s objectives. It does not keep every earlier stage as a permanent checklist underneath the active one. Completed and failed quests become short status entries rather than a complete archive of all objective text.

> For a permanent clue notebook or a collectable-document library, a quest journal may not be the right display on its own. Keep that information in separate GUI text or another deliberate content list.

20. Start the quest during gameplay

Choose the moment that makes the job known to the player: talking to the Professor, inspecting the broken machine or accepting a task. Add Start Quest to that interaction’s action list and choose Restore the workshop(restore_workshop).

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

func run(_context: Dictionary = {}) -> Variant:
    await StartQuest({"questId": "restore_workshop"})
    await SetQuestHidden({
        "questId": "restore_workshop", "hidden": false
    })
    await SetQuestTracked({
        "questId": "restore_workshop", "tracked": true
    })
    return true
```

This complete script makes the job active and also sets the visibility/tracking flags used by the built-in journal. For the custom Journal List, Start Quest alone creates the state it needs; the additional flags keep the intended state clear across journal presentations.

Start Quest’s optional Initial Stage selects a stage deliberately. Leave it blank for the normal first-stage start. Repeatedly choosing an initial stage is not a safe way to ask whether a quest has already begun.

> Opening the journal should display progress, not start every quest. Put Start Quest on the story interaction. Use Show GUI or toggle-journal on the journal’s Open Button.

Check the first live entry

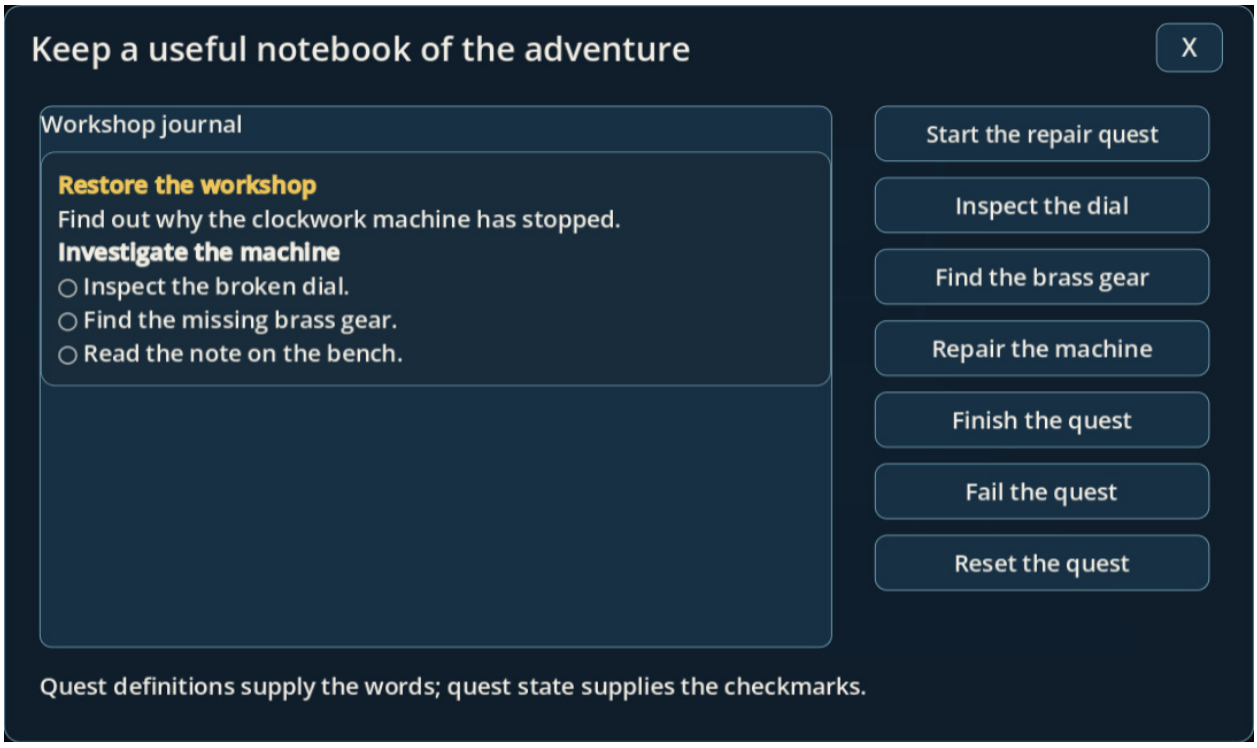

Figure 22. The started quest supplies its title, description, current stage and objective list.

Compare the running entry with the definition. You should see Restore the workshop, its description, Investigate the machine and the visible objectives. An empty circle marks an unfinished objective. A checkmark marks a completed one.

Entries are read-only: clicking an objective does not check it off. The list scrolls vertically when its contents exceed the available height. Give the player enough space to read an entry without constant scrolling.

The right-hand Buttons in this guide are ordinary test controls. In a finished game, those same actions usually run from objects, character conversations, inventory use or scene triggers. The player should see progress change because of an adventure action, not because they click a debugging Button.

If the entry is missing, first check that the quest has started and the action used the quest ID. If the stage is wrong, check the stage order and any Start Quest or Set Stage action that ran before opening the GUI.

21. Complete an objective with an action

1. On the dial’s Look interaction, add Quest: Complete Objective.
2. Choose quest restore_workshop, stage inspect and objective inspect_dial.
3. Run the game, start the quest and inspect the dial.
4. Open the journal. Inspect the broken dial should now have a checkmark.



Figure 23. The objective action identifies the quest, stage and objective.

The GUI refreshes from quest state; you do not replace the journal text manually. Use a matching objective action when the gear is actually found, rather than marking it complete merely because the journal has been opened.

> Choosing a Stage in this action actively selects that stage before completing the objective. It is useful for a first test, but repeated interactions need the guard described in the next chapters.

Let the quest advance in order

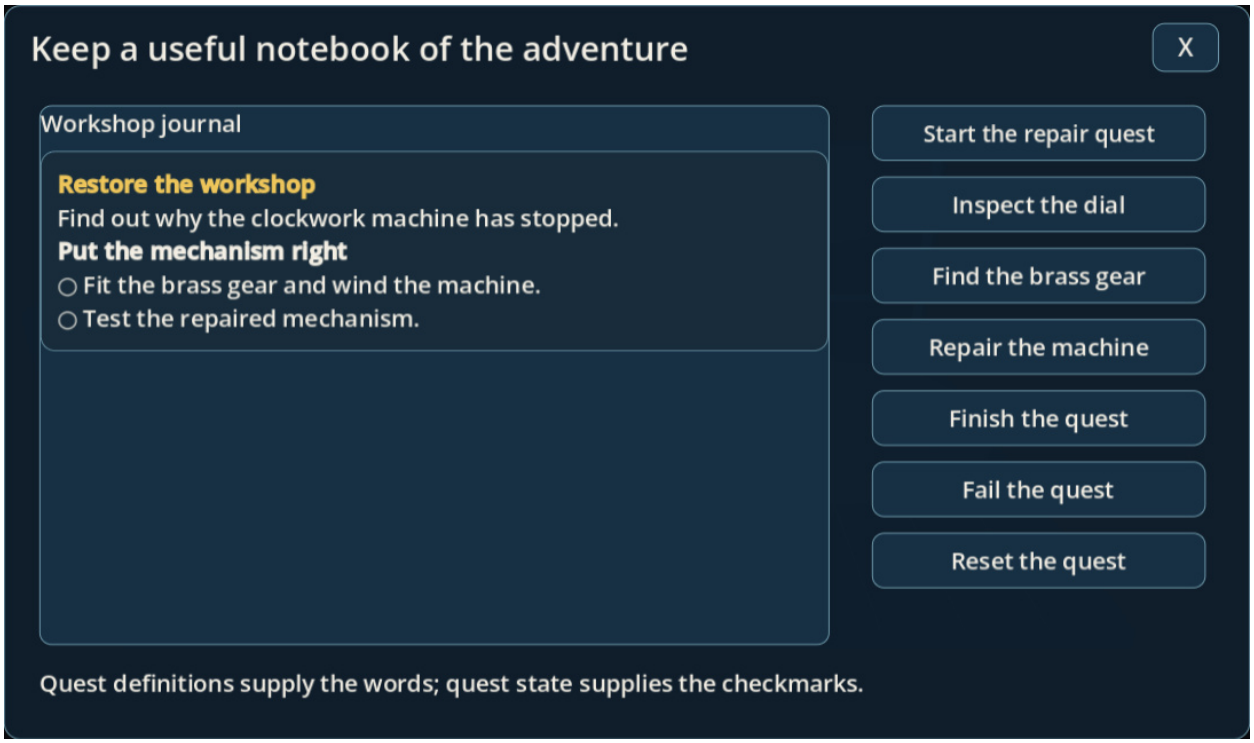

Figure 24. Both required inspect objectives are complete, so the journal has advanced to the repair stage.

Complete inspect_dial and find_gear. Because read_note is Optional, the quest can advance without it. The next stage becomes active automatically, and the current completed-objective list is cleared for that stage.

The quest finishes automatically when the final stage’s required objectives are complete. You can also use Quest: Complete Quest for a deliberate outcome, but doing so bypasses the unfinished objectives rather than proving that they were completed.

Fail Objective records an objective failure. It does not automatically fail the whole quest. This Journal List only draws a checkmark for completed objectives; a failed objective can still look like an unchecked circle. Use a clear message or another state display when the distinction matters.

> Some built-in quest behavior also reacts to inventory additions and completed dialogue. Objective IDs that resemble an item ID, or begin with speak_ or talk_, can be affected. For exact puzzle control, test those interactions and use deliberate objective actions with unambiguous IDs.

22. Protect progress from old interactions

There is a trap worth checking early. Complete Objective can select its authored Stage before it runs. After the player reaches repair, an old dial action still targeting inspect can select inspect again and reset that stage’s checklist.

That behavior was reproduced in the runtime test. A simple way to avoid it is to use a script that asks whether this objective is currently active, then completes it without forcing a stage selection:

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

func run(_context: Dictionary = {}) -> Variant:
    if ObjectiveState("restore_workshop", "inspect_dial") != "active":
        return false
    await CompleteObjective({
        "questId": "restore_workshop",
        "objectiveId": "inspect_dial"
    })
    return true
```

Use this record in the dial’s repeated interaction. While inspect_dial is active, it is completed. Once it has already been completed or belongs to an earlier stage, the script exits. A later Look can still show a normal description using your surrounding interaction logic.

> Set Quest Stage is a progression command, not a harmless selector. It can reactivate a finished quest and resets the objective lists when it changes stage. Use it only when that change is intended.

23. Complete, fail and restart a quest

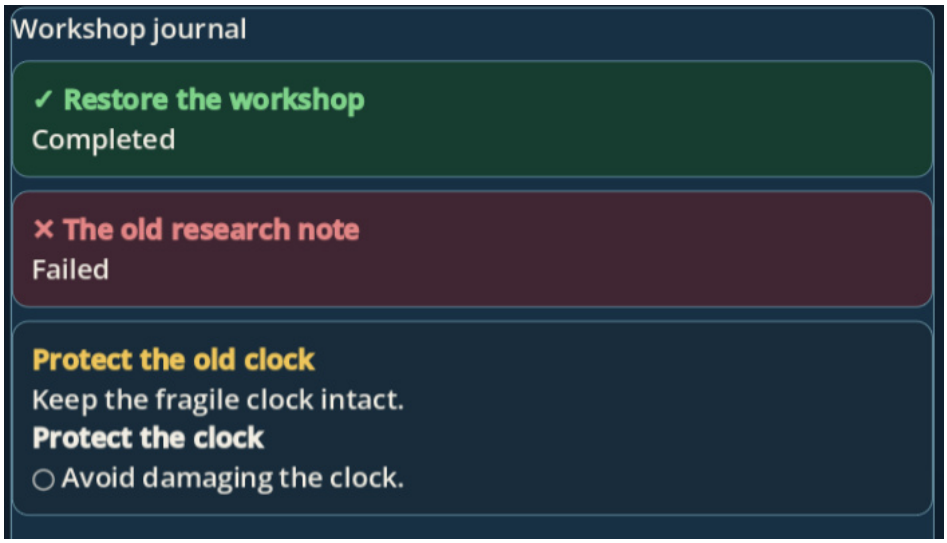

Figure 25. Completed, failed and active quests use different status presentations.

Shared action	Effect
Quest: Complete Quest	Marks the quest completed and runs its completion handling.
Quest: Fail Quest	Marks it failed. It does not invent a retry or undo scene changes.
Quest: Reset Quest	Creates a fresh active state at the first stage with cleared objective lists.
Quest: Reset Objective	Removes that objective’s completed/failed marking in the current stage.
Start Quest on a repeatable finished quest	Starts another attempt. Non-repeatable completed quests remain completed.

Reset Quest is not Remove Quest. In this build it creates an active state, so it can make a previously unstarted definition appear in the custom Journal List. Plan a reset together with any related variables, hints, inventory and scene-object changes.

24. Style the journal surface and entries

The journal has an outer surface and a separate panel for each quest entry. The common Visual Style fields affect the surface; Entry Border Opacity in Type Settings affects the entry borders. Additional entry fields are currently authored through Raw JSON.

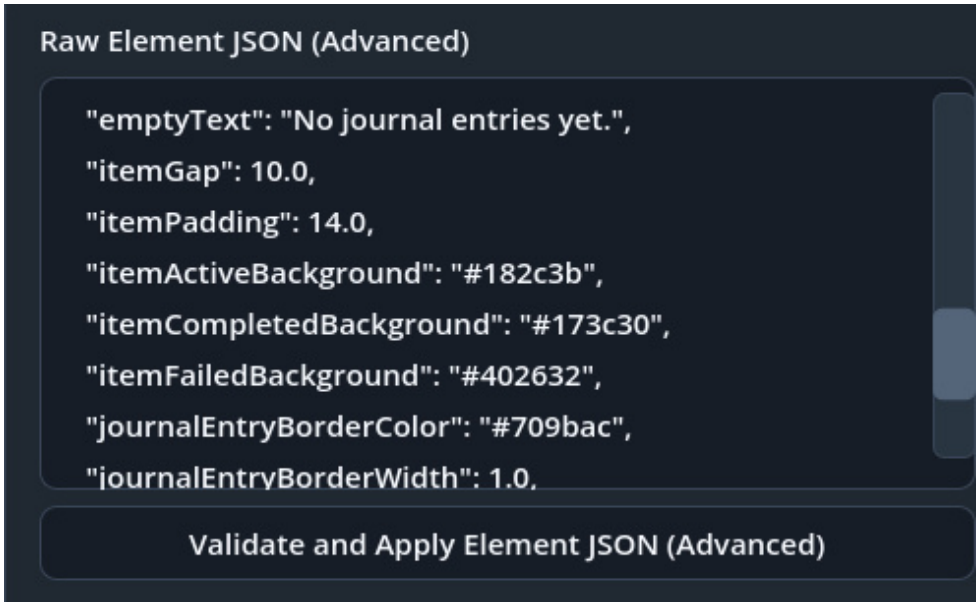

Figure 26. The advanced element data includes separate active, completed and failed entry colors.

Property	Use
itemActiveBackground	Background for an active quest entry.
itemCompletedBackground / itemFailedBackground	Backgrounds for the two finished states.
journalEntryBorderColor / Width / Opacity	Entry border appearance; use the full property names shown on the next page.
journalEntryRadius	The entry panel’s common radius.
itemPadding / itemGap	Space inside an entry / between entries.
showHeader	Show or hide the widget’s title above the list.

The running journal uses the element’s font and fontSize for entry text. Status titles also receive built-in colors and bold formatting. Changing ordinary Text Color does not replace every status color in the generated rich text.

Edit entry properties without losing the element

Open Raw JSON (Advanced) on quest_journal. Add or edit these properties inside the existing element object; this is a fragment to merge, not a replacement for the whole element:

```
"showHeader": true,
"emptyText": "No journal entries yet.",
"itemActiveBackground": "#182c3b",
"itemCompletedBackground": "#173c30",
"itemFailedBackground": "#402632",
"journalEntryBorderColor": "#709bac",
"journalEntryBorderWidth": 1,
"journalEntryBorderOpacity": 0.8,
"journalEntryRadius": 12,
"itemPadding": 14,
"itemGap": 10,
"showCompleted": true,
"showFailed": true
```

Use Validate and Apply Element JSON, then Save Project and test in gameplay. Empty Text is also useful here because the shared paged-widget field is not exposed for Journal List in the current typed layout.

The outer common radius and journalEntryRadius can differ, but this renderer copies the element’s individual corner overrides into the entry style too. If you set per-corner radii on the outer element, inspect the entries as well. Do not assume there are independent per-corner entry controls.

> Entry panels use flat styles in this version. They do not have dedicated background/hover/pressed image slots like inventory or save-widget regions. A background image on the Journal List belongs to its outer surface.

25. Decide which journal entries to show

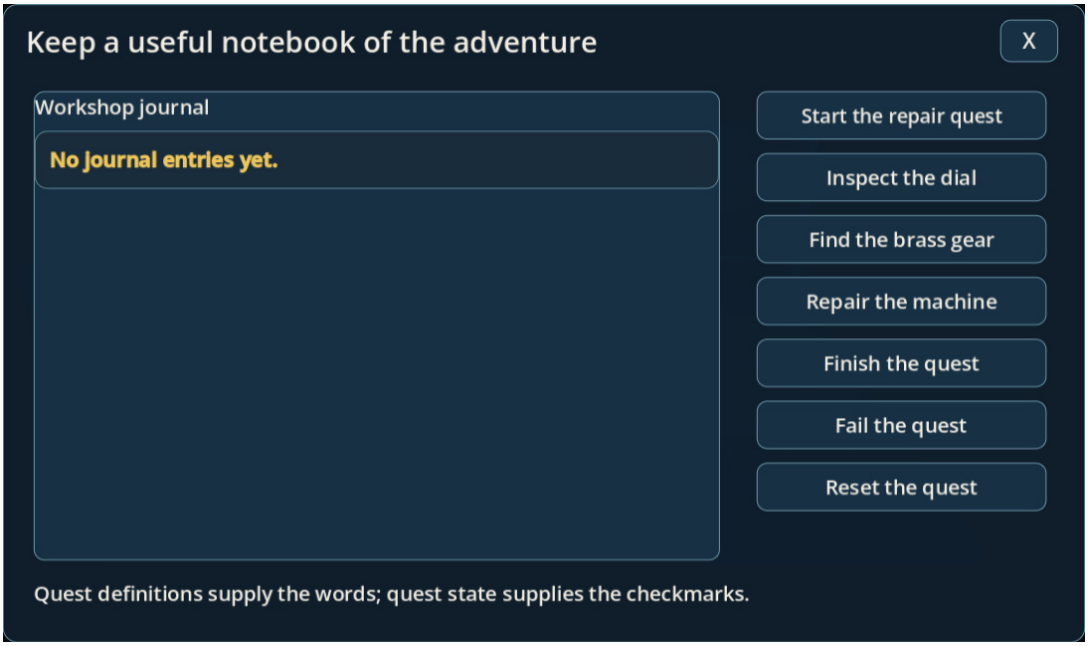

Figure 27. With showCompleted false, a completed-only journal falls back to its empty entry.

Set showCompleted to false on the Journal List to leave completed quests out of this widget. Set showFailed to false to omit failed quests. They are advanced element properties in this release, not visible checkboxes in the typed Journal Entries section.

The custom Journal List currently includes every defined quest with a runtime state that passes those two status filters. It does not filter by Category, Hidden or Tracked. The built-in journal does honor Hidden and Tracked, so swapping the presentation can change what appears.

This was checked directly: a started quest marked hidden and untracked remained in the GUI Journal List while being absent from the built-in journal text. For an undiscovered quest, prevent its initial state by using Hidden on and AutoTrack off. Once started, do not promise a per-quest Hide action for this particular widget.

> showCompleted and showFailed apply to that entire Journal List. They are not a per-entry filter or a way to split the list by quest category.

What the journal does with its text

Definition or setting	Custom Journal List result
Quest title and description	Shown for active quests.
Current stage title	Shown above its visible objective lines.
Stage description	Not emitted as a separate paragraph by this renderer.
Hidden objective	Not displayed; still required unless Optional is also on.
Completed / failed quest	A short title and status entry; earlier checklists are not retained on-screen.
Quest TitleKey / DescriptionKey / Objective TextKey	Do not assume these fields alone make this custom output translate.

The widget builds its own rich text from quest definitions. It does not send each quest line through the same game-text expansion used by a normal Label or a hint answer. A $$power token or @PlayerName inside a quest description can therefore remain literal in this custom list.

Quest titles, descriptions and objective text are included in the translation scanner’s normal text fields, but this renderer does not itself perform the usual expansion/translation pass for its body. Check the running journal in each language. For a deliberately localized custom journal, supply the relevant translated definition text through the quest property actions, or build the display with Labels that use normal game text.

> Changing a quest definition during play is available through Change Quest Property, Change Stage Property and Change Objective Property. Those actions update the runtime copy; they do not rewrite the editor’s authored quest.

26. Connect all three to the puzzle

Build the adventure interaction first, then update the displays from that result. The widgets should describe what happened; their visibility should not decide whether the puzzle was solved.

Player action	Update the game
Learn about the broken machine	Start restore_workshop. Reveal machine hint 1 if the first hint should be available now.
Look at the dial	Complete inspect_dial only while it is active.
Take the brass gear	Give the item and complete find_gear at the intended point.
Use the gear on the mechanism	Check the item and current objective, change the machine, then complete repair_machine.
Wind or test the repaired machine	Change power and complete test_machine when the test succeeds.
Ask for help	Run UnlockHint or next_hint; do not alter the quest merely for opening hints.

You can place shared actions directly in an action list or connect their visual-script nodes in sequence. For a small first test, On Interact → Start Quest → Complete Objective → Unlock Hint makes the connection visible. Add the state guard before using that interaction repeatedly in a finished game.

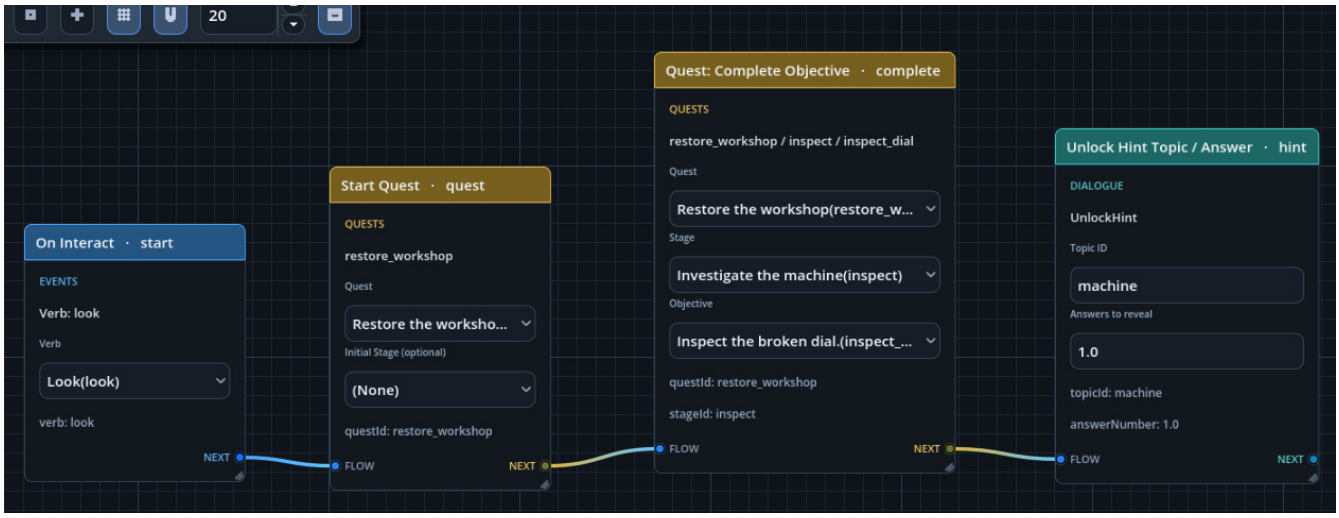

Figure 28. The same quest and hint actions are available as visual nodes. The repeated-interaction guard is still your game logic.

Quest completion does not automatically set power to 100. Hint unlocking does not complete an objective. Add every intended state change explicitly to the successful interaction.

27. Use a guarded interaction script

This complete record can run when the player inspects the dial. It starts the quest if necessary, completes only an active dial objective and makes the first hint available:

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

func run(_context: Dictionary = {}) -> Variant:
    if QuestState("restore_workshop") == "inactive":
        await StartQuest({"questId": "restore_workshop"})
        await SetQuestHidden({
            "questId": "restore_workshop", "hidden": false
        })
        await SetQuestTracked({
            "questId": "restore_workshop", "tracked": true
        })
    if ObjectiveState("restore_workshop", "inspect_dial") == "active":
        await CompleteObjective({
            "questId": "restore_workshop",
            "objectiveId": "inspect_dial"
        })
    await UnlockHint({"topicId": "machine", "answerNumber": 1})
    return true
```

Apply the record and run it from the dial’s intended verb interaction. The script does not force the inspect stage, so examining the dial again during repair cannot rewind the quest. Unlocking answer 1 again is harmless because the hint action preserves the highest count.

QuestState returns inactive when there is no state, then active, completed or failed as the quest changes. ObjectiveState reports inactive, active, completed or failed for the current objective state. Treat those results as decisions in the story logic, not as display text.

28. Open and close the right window

For an ordinary custom GUI, a HUD Button can use Show GUI, Toggle GUI or an action list containing ShowGUI. A Close Button uses Hide GUI. Those operations choose the GUI ID, not the widget’s element ID.

For the journal’s built-in open command, assign your custom GUI in Global Settings. In the General tab, scroll down to Custom System Interfaces and choose Workshop Journal(journal_lab) for Journal Screen. Apply Changes and Save Project.

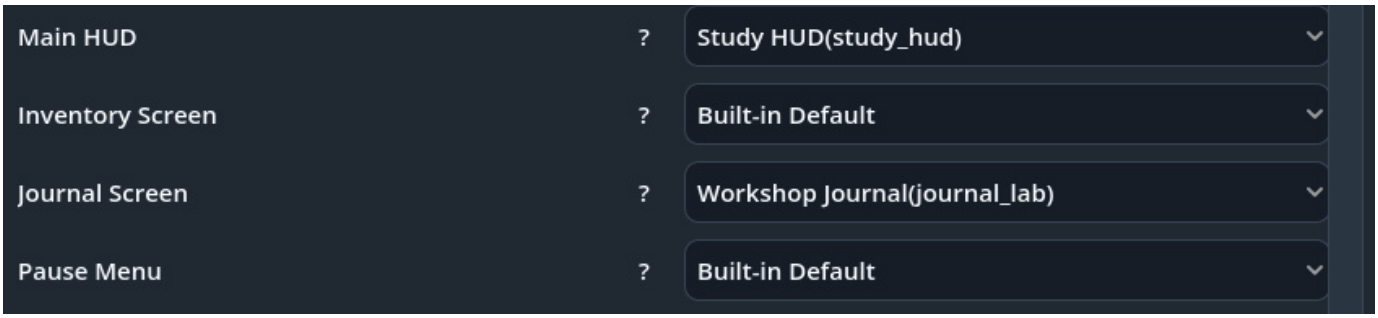

Figure 29. Journal Screen chooses which custom GUI the journal command opens.

A HUD Button using toggle-journal, or the configured journal keybinding, then targets that screen. The engine can raise it above ordinary HUD layers and coordinate it with the inventory screen. If no custom journal is assigned, the command can use the built-in panel instead.

For hints, use an ordinary Show/Toggle GUI Button unless your game has a more specific help flow. A modal GUI blocks normal world input while the window is open; close it deliberately before expecting another world interaction.

> Use one open path consistently. A generic ShowGUI call and the journal’s system command do not necessarily apply identical priority and coordination behavior. Test the exact Button and keybinding the player will use.

29. Save the progress, not just the picture

Information	How to treat it
power and other game variables	Use normal game state so the value saves and restores. A bound bar follows it.
hintsUnlocked	The unlock dictionary is part of runtime variables and saves with them.
Quest states	Current stage, quest status and objective states have their own saved quest data.
Runtime quest text overrides	Quest property actions store overrides with the saved quest state.
Expanded/collapsed hint rows	Temporary presentation; do not rely on it as saved progress.
A progress event’s once cache	Lives on the current widget, not as a saved story flag.

1. Start the quest, complete inspect_dial and set power to 60.
2. Unlock machine answers 1 and 2. Save the game to a fresh test slot.
3. Change power, clear the hints and fail or advance the quest.
4. Load the saved slot and reopen the windows.
5. Check power = 60, two available hint answers and the earlier quest checklist.
6. Repeat after stopping and restarting the game preview, then loading the same slot.

The first five steps were exercised in the running build for this guide: power, the two unlocked answers and the quest state restored. Restarting is a separate check because it creates new widget instances.

> A story reward needs its own saved flag. A visual once event can survive a load within the same live widget yet fire again after a fresh run or rebuild.

30. Know what Preview is proving

Check	Use
Bar size, fill shape, caption and sample value	GUI Designer, including local Preview.
Bound variable updates and threshold actions	A running game preview.
Hint topic titles, unlock counts and answer expansion	A running game; the designer uses a simplified journal-style preview.
Real quest checklist, stage advancement and outcomes	A running game; designer quest titles do not prove the state.
Modal behavior, HUD opening and journal keybinding	The player-facing game flow.
Saved data and a fresh widget instance	Save, restart the preview and load.
Text wrapping, input and scale on the target device	An exported game on that device.

Turn designer Preview off before selecting elements or editing their fields. It is a local visual test; it does not run the story events used by these examples.

Hint and journal contents are rebuilt when dynamic text refreshes. The list may reflow, a hint answer may collapse and a long scroll position can feel different after a large update. Avoid a constantly changing variable that forces unnecessary refreshes while the player reads.

> Use the real font size and the longest text early. A generous desktop rectangle can become cramped on a small display, especially when translations wrap to extra lines.

31. Troubleshoot a Progress Bar

Problem	Check this first
The designer says 60 but the game says 0	Preview Value and the runtime source are different. Check value and variable.
The bar stays at one plausible value	Check the exact variable ID and that it exists as a Number.
It reaches full too early	Compare the actual source value with Maximum.
The number grows but the fill does not	Check Fill Opacity, fill color/background contrast and Inner Fill Padding.
The bar is full but a source comparison is wrong	The displayed value clamps; the source variable may still exceed Maximum.
Clicking does not change the value	That is normal for a Progress Bar. Use a Button/action or a Slider.
A threshold never runs	Cross it in the configured direction. Staying on one side is not a crossing.
A threshold runs after a visual edit	A property update can reapply the authored value. Guard story actions independently.
An event only runs once during a live session	Check once and the event ID. Different once events need distinct IDs.
The image appears behind the whole meter	You assigned the outer background. Use Fill Image for the filled portion.

Test with a plain, high-contrast fill, Maximum 100 and an explicit Set Variable to 50. Once that works, restore the image, gradient and events one at a time.

Troubleshoot hints and the journal

Problem	Check this first
Hints show quest names in the designer	That is this version’s simplified preview. Test the running game.
No hint topics appear during play	Check Topics JSON, topic IDs and hintsUnlocked.
The topic opens but there is no answer	The unlock count may be 0, or answers may be empty/not a string array.
A smaller unlock does not hide later hints	UnlockHint preserves the highest count. Remove/reset the dictionary entry deliberately.
An answer closes after another update	The hint rows were rebuilt. The unlocked count remains.
The journal is empty	Start the quest and check its ID. Check showCompleted/showFailed if it has finished.
A quest appears before discovery	Hidden must be on and AutoTrack off to defer its initial state.
Hiding/untracking does not remove a custom entry	This GUI renderer does not filter those flags. The built-in journal does.
A stage goes backwards	An old Complete Objective or Set Stage action may be forcing the earlier stage.
A quest does not advance	Check every required objective, including any hidden required objective.
Edits revert when another item is selected	Apply Stage/Objective Fields, then Apply Changes and Save Project.
A quest sentence keeps a wildcard or source language	The custom body does not use the normal per-line text expansion. Use explicit definition text or a custom Label display.

32. Progress Bar reference

Property	Purpose
type: progress	The authored GUI element type.
max	The amount represented by a full bar.
previewValue	Sample value for designer rendering.
value	Authored current/fallback value used by the runtime definition.
variable	Optional Number variable ID read during dynamic refresh.
text	Caption; supports {value}, {max}, {percent}, {percentage}.
fillDirection	left-to-right, right-to-left, top-to-bottom or bottom-to-top.
fillColor / fillOpacity	Solid fill color and alpha.
fillGradientEnabled / fillColor2	Enable the two-color fill and choose its second color.
progressFillAssetId / tileFillImage	Image asset for the fill and whether it repeats.
progressPadding	Inset of the fill from the outer rectangle.
borderWidth / radius	Common outer border width and radius.
percentageEvents	Array of crossing-event objects.

Inspector labels can be friendlier than the stored property names. For example, the typed Border Radius field writes radius. Use the stored names for Raw JSON and Change GUI Element Property.

> Choose one source for the number. A variable-bound meter should be driven through its variable; an unbound meter can be driven through value.

Hint List reference

Property or action	Purpose
type: hintList	The authored GUI element type.
title	Displayed list heading. Edited as Displayed Title.
hintTopicsJson	String containing a JSON array of topic objects.
Topic id / title / answers	Stable topic ID, displayed question and ordered answer strings.
emptyText	Message when no unlocked topics can be displayed; use Raw JSON if needed.
hintsUnlocked	Runtime dictionary: topic ID → highest unlocked answer number.
UnlockHint	Shared action with topicId and answerNumber.
SetVariable	Can deliberately replace the unlock dictionary for a reset.
Topic Button	Expands/collapses the answers already unlocked.

Term	Plain meaning
Definition	The words and settings saved in the project.
State	What has happened during this playthrough.
Dictionary	A set of named values, such as machine: 2 and professor: 1.
Runtime	The game while it is actually running.
Widget instance	One live copy of an element; rebuilding it creates a new copy.

Journal and quest action reference

Property or action	Purpose
type: journalList	The authored GUI element type.
title / showHeader / emptyText	Heading, whether it is displayed and the empty-entry message.
showCompleted / showFailed	Status filters for the whole Journal List.
itemPadding / itemGap	Entry inset and spacing between entries.
journalEntryBorderColor / journalEntryBorderWidth	Entry border color and thickness.
journalEntryBorderOpacity / journalEntryRadius	Entry border alpha and common radius.
itemActiveBackground / itemCompletedBackground / itemFailedBackground	Backgrounds for the three quest status presentations.
StartQuest	questId, optional stageId.
SetQuestStage	questId and stageId; changes progression.
CompleteObjective / FailObjective / ResetObjective	questId, objectiveId, optional stageId.
CompleteQuest / FailQuest / ResetQuest	questId; changes the quest outcome or restarts it.
ChangeQuestProperty	questId, property, value.
ChangeQuestStageProperty	questId, stageId, property, value.
ChangeQuestObjectiveProperty	questId, stageId, objectiveId, property, value.
QuestState / ObjectiveState	GDScript query helpers used to protect repeated interactions.

Screenshots and behavior checks: PCA 3D Adventure Studio 0.86.3.4. The guide supplies instructions only; no practice files are included. Test the finished puzzle through the player’s actual interactions before packaging your game.