PCA 3D ADVENTURE STUDIO

Variable Designer

Remember choices, count progress and give your puzzles a memory



Figure 1. A running workshop scene showing a puzzle flag, a score, an attempt counter and a list of clues.

Start with one question: has the player wound the clock? Build from that small yes-or-no answer into counters, objectives, lists, visual branches and saved progress.

Version 0.86.3.4 · Windows, macOS and Linux

A step-by-step guide with screenshots from the editor.
No practice project or extra downloads required.

How to use this guide

Open a project you already know, or use one of the bundled demos. You need an object the player can interact with and a scene you can test. A clock is used throughout these pages, but a door, a lamp or a desk drawer works just as well.

The screenshots show a small workshop made with the bundled Clockmaker artwork. It is an illustration of the steps, not a separate project you need to install. The labels and test buttons shown in play are example GUI elements, not built-in Variable Designer controls.

Choose a route through the guide

You want to…	Start here
Create and understand your first variables	Chapters 1–8
Change them through object actions	Chapters 9–10
Check values while the game runs	Chapter 11
Build conditions with visual nodes	Chapters 12–13
Display values or use an array in a GUI	Chapters 14–15
Read and change values in code	Chapters 16–18
Understand room changes, saves and resets	Chapter 19
Track down a problem and finish testing	Chapters 20–21

Follow the steps in order for the first Boolean, number and object-action examples. The scripting chapters are alternatives: you do not need to implement the same puzzle in every language.

Use the exact IDs in the examples, or replace them consistently with your own. A displayed name such as Clock Wound helps you find a record; the ID clock_wound is what actions and code use.

> Apply Changes accepts edits in the current designer. Save Project writes the project. Save Game records a player’s progress. Those three jobs are different.

Screenshots and the described behavior were checked in PCA 3D 0.86.3.4. The desktop workflow is the same across Windows, macOS and Linux. Use the clickable contents and PDF bookmarks to return to a particular topic.

Contents

Click a chapter or its page number. PDF bookmarks also include the smaller section headings.

How to use this guide . . . 2
1. What a variable remembers . . . 4
2. Find your way around the window . . . 5
3. Make your first yes-or-no variable . . . 7
4. Use numbers for scores and counters . . . 8
5. Use text for objectives and answers . . . 9
6. Blank, empty, False and zero . . . 10
7. Store a list in an Array . . . 11
8. Organise, copy and remove variables . . . 13
9. Change a variable from an object . . . 15
10. Pick the right change operation . . . 17
11. Watch values during a test . . . 18
12. Award points only once . . . 20
13. Compare amounts with If Compare . . . 23
14. Put live values in your GUI . . . 24
15. Feed an Array into a list box . . . 25
16. Use the same values in GDScript . . . 27
17. Share a value with PCA Script . . . 29
18. Create a variable during play . . . 31
19. Room changes, restarts and saves . . . 32
20. Find the cause when a value is wrong . . . 34
21. Finish with one complete test pass . . . 36

1. What a variable remembers

An adventure game needs to remember what has happened. Has the player opened the drawer? How many coins do they have? What is the current objective? A variable stores one of those pieces of information under an ID you can refer to later.

For this walkthrough, the player uses a workshop clock. The game remembers whether it has been wound, adds points and changes an objective. Later, a condition checks that memory so the same clock cannot keep handing out points.

ID	Type and starting value	What it remembers
clock_wound	Boolean · False	Whether the clock puzzle has been completed.
score	Number · 0	The player’s points.
attempts	Number · 0	How often the first test action has run.
objective	String · Wind the workshop clock.	A short instruction that a GUI label can display.
clues	Array · ["clock diagram"]	A list of discovered clue labels.
last_choice	String · blank/null	No clue has been chosen yet.

Starting value and current value

Variable Designer sets the value a fresh game starts with. During play, the game has its own current value. Changing clock_wound to True in play does not tick its starting-value box in the designer.

These designer variables are shared game values. A scene, a character interaction, a GUI and a script can all read the same score. Putting a variable in a database folder does not give that folder its own private copy.

> A variable records a fact. It does not automatically play a clock sound, give an inventory item, open a door or change an animation. Add the actions that make those things happen, and use variables to remember the result.

2. Find your way around the window

Open Variable Designer from the designer tabs. In the icon strip, hover over a tab to see its name; use the strip’s arrows if the tab is off-screen. You can also open an existing variable from the Database panel’s Variables category.

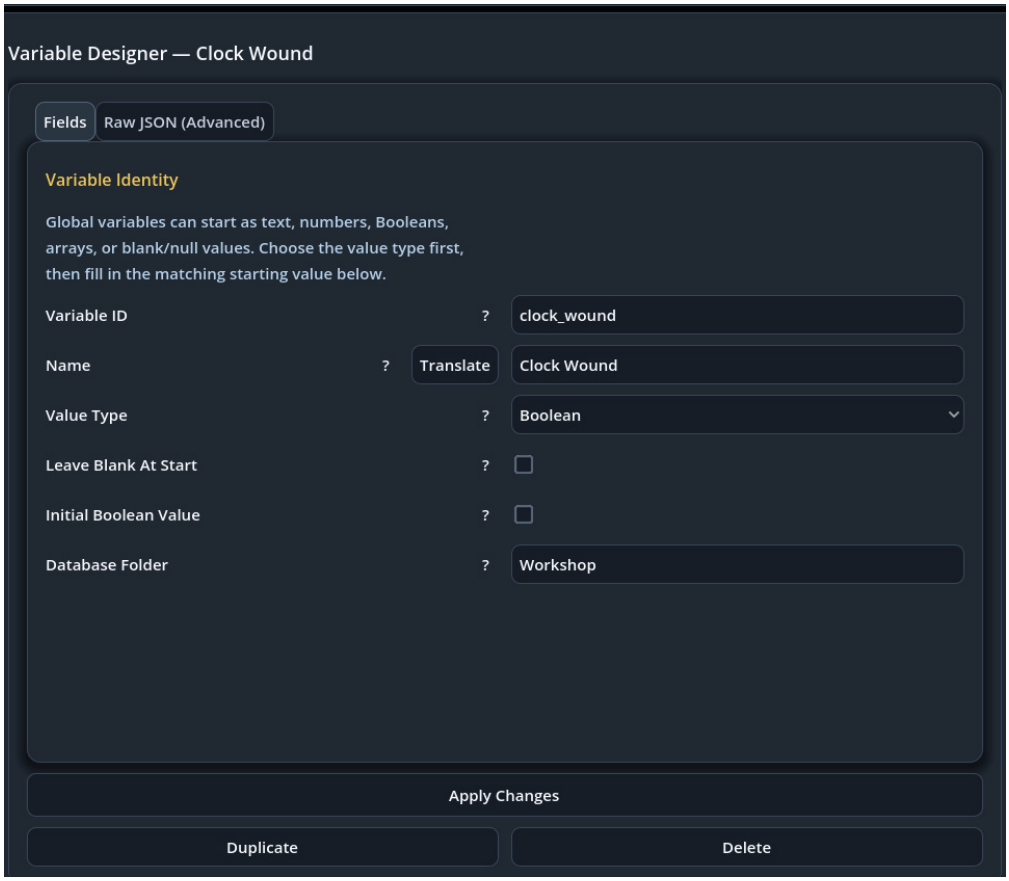

Figure 2. Variable Designer with Clock Wound selected. The selector above the form shows the name and ID together.

Control	What it does
Record selector	Chooses the variable you want to edit. For example, Clock Wound(clock_wound).
Add New	Creates a new variable record.
Clone Selected / Duplicate	Makes an independent copy of the selected record. The available button depends on the layout.
Fields	Shows the ordinary, labelled editing controls.
Raw JSON (Advanced)	Shows the record as structured text. You can leave this tab alone for normal work.

The fields and the two kinds of save

Field or button	How to use it
Variable ID	The stable key used by actions, scripts and text tokens. Use a short ID such as clock_wound.
Name	A readable editor label, such as Clock Wound. The Translate control beside it concerns that name; it does not translate the variable’s stored story text for you.
Value Type	Choose String, Boolean, Array or Number. The matching initial-value control appears below.
Leave Blank At Start	Starts the value as null: no value has been supplied yet. It hides the initial-value control.
Initial … Value	The starting text, checkbox, list or number for the chosen type.
Database Folder	Organises the record in the Database. A slash can create a subfolder, for example Puzzles/Workshop.
Apply Changes	Accepts what you entered into the current record.
Save Project	Saves the project definitions, including the accepted starting values.
Delete	Requests deletion of the selected record. Read the confirmation before accepting.

Keep IDs predictable

Use lower-case letters and underscores: clock_wound, workshop_attempts and current_objective. The designer trims surrounding spaces, converts the ID to lower case and replaces spaces with underscores when applying this form. Start with a letter; avoid punctuation.

IDs are exact keys. score, Score and total_score should not be treated as interchangeable in hand-written code. Select the existing record from a reference dropdown where one is offered.

> Choose the type before entering the starting value. Switching the type changes the visible editor; it is not a dependable conversion of the value you already entered. Check the new field and enter the value you intend.

3. Make your first yes-or-no variable

A Boolean has two ordinary values: True and False. Use it for a fact with two states, such as a solved puzzle, an unlocked door or a conversation the player has already heard.

1. Check that `clock_wound` does not already exist. Reuse that record if it does; otherwise click **Add New** in Variable Designer.
2. Set **Variable ID** to `clock_wound` and **Name** to Clock Wound.
3. Choose **Boolean** for Value Type.
4. Leave **Leave Blank At Start** off. Leave **Initial Boolean Value** unticked, which means False.
5. Set **Database Folder** to Workshop, click **Apply Changes**, then **Save Project**.

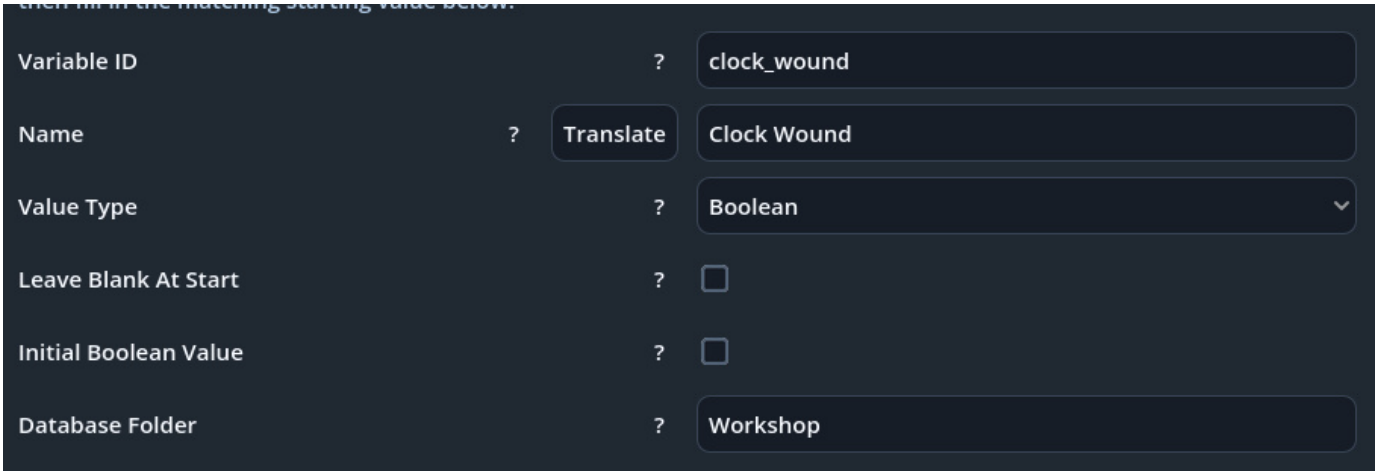

Figure 3. The new record filled in before Apply Changes. An unticked initial Boolean is False; it is still a real value.

Read the question the same way every time

The ID clock_wound asks “has the clock been wound?” False means it has not; True means it has. Keep that meaning consistent throughout the project. A second variable named clock_not_wound would make every condition harder to follow.

After applying, select a different record and come back. Clock Wound should still have type Boolean and an unticked starting value. The selector should now show the new name and ID.

> False is not blank. It answers the question with “no”. Blank/null means no answer has been stored yet. Chapter 6 shows when that distinction matters.

4. Use numbers for scores and counters

Number stores an amount. Use it for points, attempts, money, time remaining or a puzzle dial. Whole numbers and decimal numbers use the same Value Type.

1. Add a variable with ID `score`, Name Score and Value Type **Number**.
2. Keep Leave Blank At Start off and set **Initial Number Value** to `0`.
3. Put it in a Progress folder and apply the record.
4. Add `attempts`, Name Wind Attempts, type Number, initial value `0`, in Workshop. Apply and save the project.

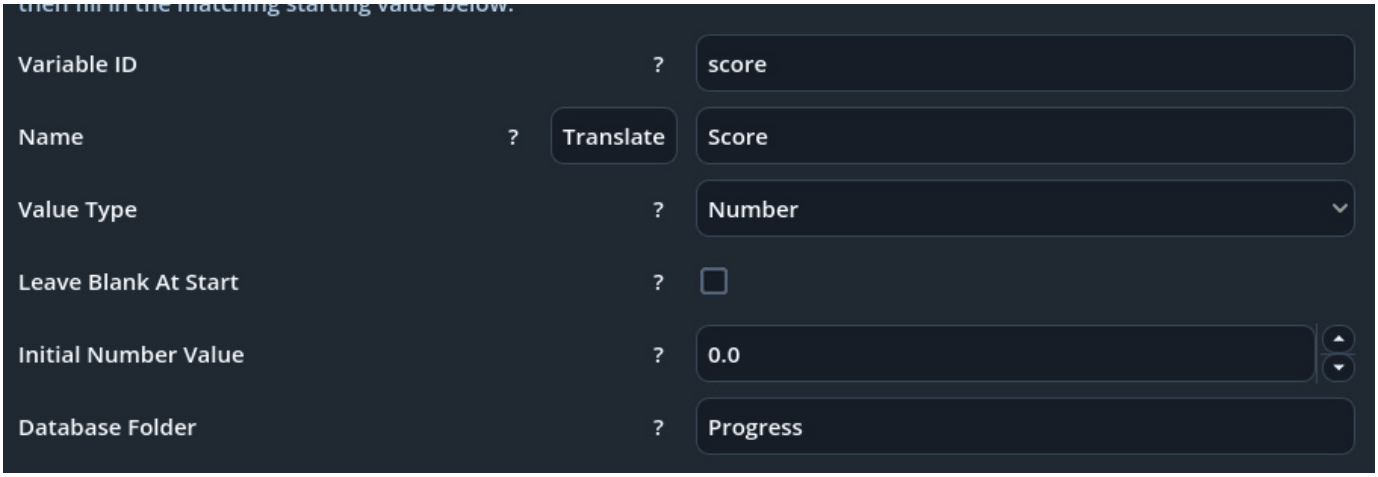

Figure 4. A Number starting at zero. Zero is a useful starting amount; it does not mean the value is missing.

Set an amount or change an amount?

Set Variable with a value of 10 replaces the current score with 10. Change Variable with Operation add and Value 10 adds ten to whatever the score is now. For a reward, you usually want the second one.

Before	Action	After
score = 7	Set Variable → 10	score = 10
score = 7	Change Variable → add 10	score = 17
attempts = 0	Change Variable → add 1	attempts = 1

Arithmetic actions may display a whole amount as 10.0. It is still the number ten. A display-formatting choice is separate from the value’s type. Typing the quoted text "10" in a JSON value field makes a String instead.

5. Use text for objectives and answers

String is the name for a piece of text. It can hold a short objective, a name the player entered or an answer to a riddle. It is one value even when the text contains several words.

1. Add a variable with ID `objective` and Name Current Objective.
2. Choose **String** and leave Leave Blank At Start off.
3. In **Initial String Value**, enter `Wind the workshop clock.`
4. Choose the Workshop folder, apply the record and save.

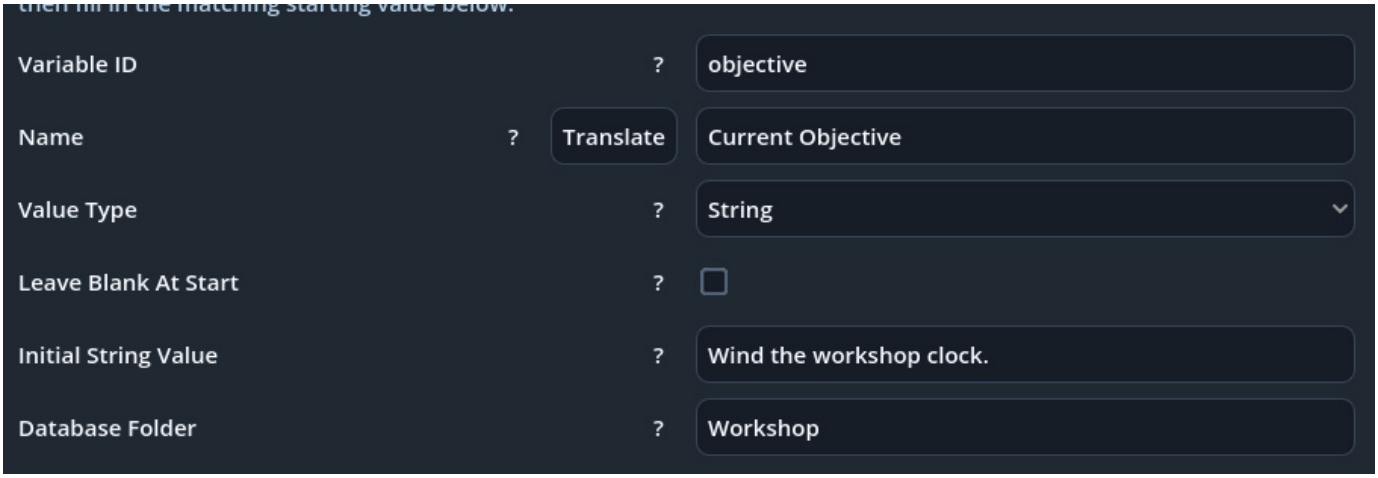

Figure 5. The ordinary Initial String Value field contains the text itself. Do not put JSON quotation marks around it here.

The field decides whether quotes are needed

In this designer’s Initial String Value, type the words normally. The field already knows it is editing text. If you type quotation marks around the sentence here, those marks become part of the stored sentence.

An action list’s general Value field has a different job: it accepts several types. In that field, a quoted JSON string such as "The clock is running." makes the intended type explicit. Chapter 9 shows the two fields side by side in context.

> Changing a text variable is not the same as showing a message. Set objective to update the stored instruction. Use a GUI label containing $$objective, or a Show Message action containing that token, to display it.

A story sentence stored in a variable is not automatically translated by the Name field’s Translate control. Use your game’s translation workflow for player-facing text and test each language in play.

6. Blank, empty, False and zero

Sometimes a question has not been answered yet. A clue-selection variable is a good example: the player has not picked a clue, so there is no selected ID to store. Leave Blank At Start represents that state with null.

1. Add `last_choice`, Name Last Choice, and choose **String**.
2. Tick **Leave Blank At Start**. The initial String field disappears.
3. Apply and save. The record still has type String, but its starting value is null.

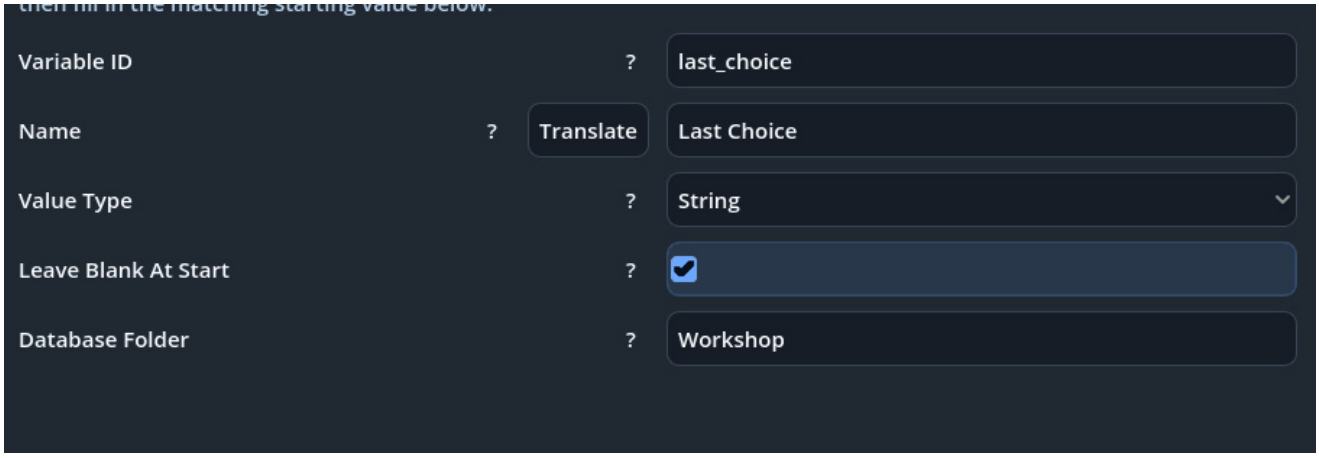

Figure 6. Last Choice starts blank. The declared type stays String while the starting value is null.

Value	Meaning in your design
False	A Boolean answer: no, this has not happened.
0	A Number: none of this amount.
""	An empty String: text exists, but contains no characters.
[]	An empty Array: a list with no members.
null	No value has been supplied yet.

Untick Leave Blank At Start and leave Initial String Value empty if you want an empty String instead. Do not use the word null in the String starting field to mean blank; that stores four letters.

> During play, visual Set Variable nodes have a Blank / null option, and GDScript can supply null directly. In this release, typing null into an ordinary action-list Value text field does not reliably create null. Use the explicit blank option or code for that case.

7. Store a list in an Array

An Array is an ordered list stored in one variable. It is useful for discovered clues, a sequence entered into a puzzle or choices collected during a conversation. Each entry is a separate member of the list.

1. Add `clues`, Name Clue List, with Value Type **Array**.
2. Keep Leave Blank At Start off.
3. Enter `["clock diagram"]` in **Initial Array Value**.
4. Apply and save. Use `[]` instead if the player should begin with no clues.

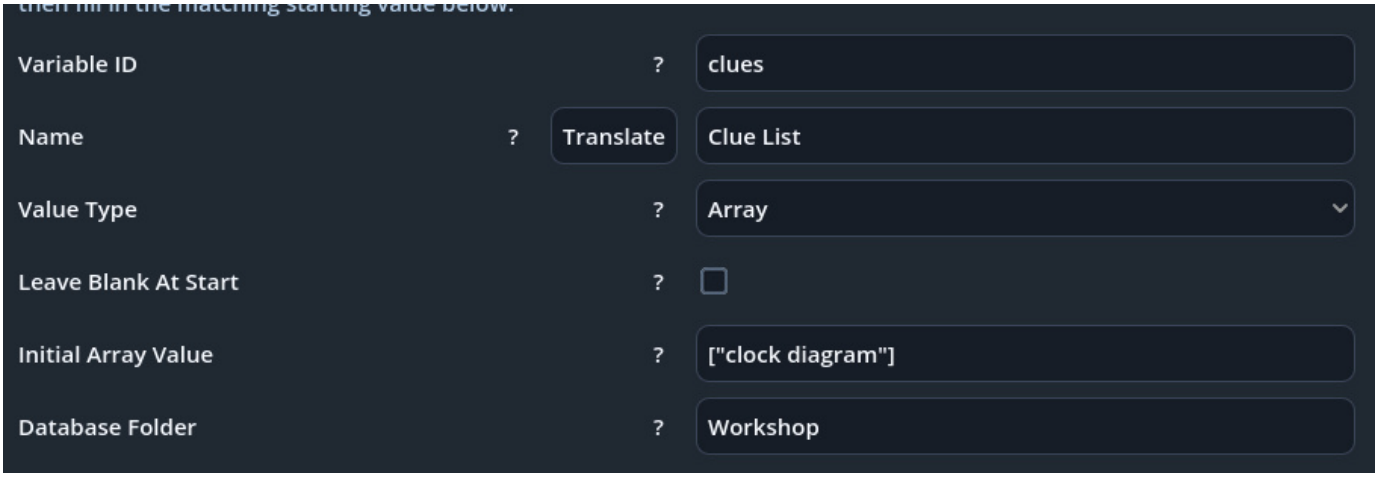

Figure 7. The list begins with one String member: clock diagram. Square brackets identify a JSON array.

Add one clue during play

Use Change Variable, choose clues, set Operation to append, and supply "winding socket" as the Value in the action-list editor. The resulting list is ["clock diagram", "winding socket"]. You are adding one new member, rather than replacing the whole list.

Operation	Result
Set to []	Clears the list.
Append "winding socket"	Adds that one String to the end.
Append the same clue again	Adds another copy; duplicates are allowed.
Append ["note", "key"]	Adds one nested Array, not two separate members.

A clue name in this list is just data. It does not create an inventory item. Use the inventory actions when the player should actually receive or lose something.

Arrays: exact JSON or a quick text list?

Initial Array Value accepts either a JSON array or a simple separated list. The quick form is convenient for text labels, but it does not preserve numeric and Boolean types.

What you enter	What is stored
gear, 2, true	Three Strings: "gear", "2", "true".
["gear", 2, true]	A String, a Number and a Boolean.
[]	An empty Array.
An empty Array input field	An empty Array. Leave Blank At Start would be null instead.

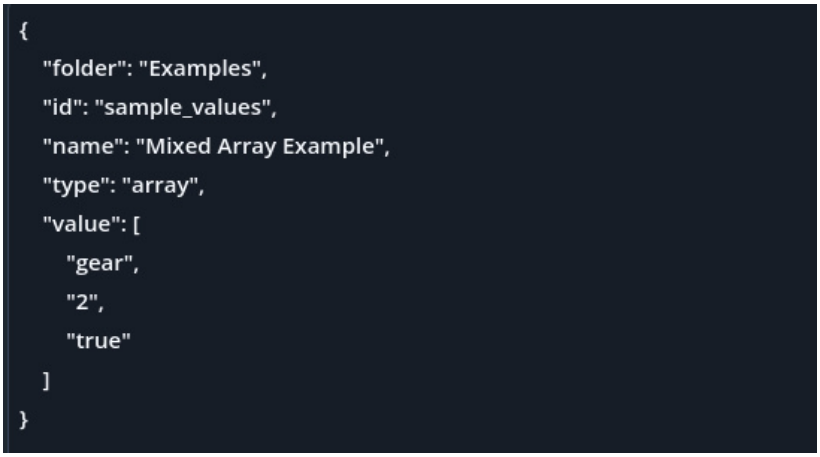

Figure 8. Raw JSON after entering a comma-separated list. Both 2 and true are quoted: they are text.

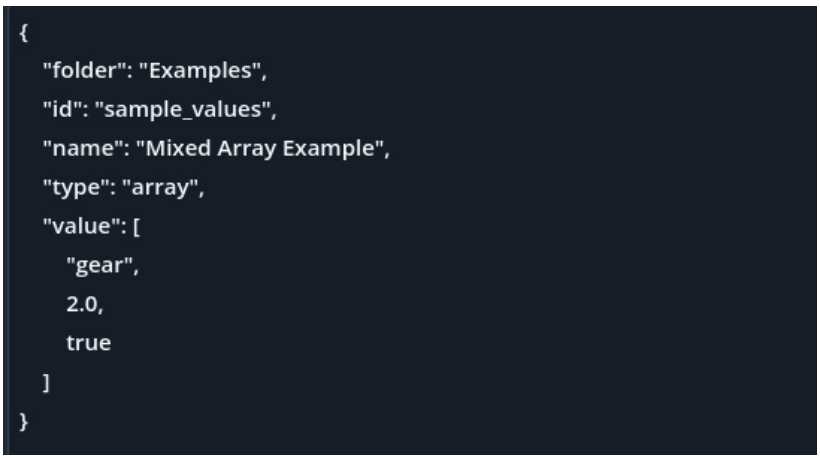

Figure 9. The same example entered as valid JSON. The unquoted 2 and true retain their Number and Boolean types.

Commas, semicolons and line breaks can separate the quick text form. For mixed data, use valid JSON. A malformed JSON array can fall back to text splitting, so inspect the saved record if the result looks wrong. Validation checks the outer Array type, not the meaning of every member.

8. Organise, copy and remove variables

Good names save time once a game has dozens of puzzles. Keep the ID stable, give it a useful display name and place related records in a database folder. Workshop, Progress and Puzzles/Clockwork are ordinary organisational choices.

Make a separate copy

1. Select Clock Wound in the record selector.
2. Click **Clone Selected**, or **Duplicate** at the bottom of the record form.
3. Inspect the new ID and name. A copy receives its own unique ID, such as clock_wound_copy.
4. Rename and configure the copy for its actual purpose, then apply and save.

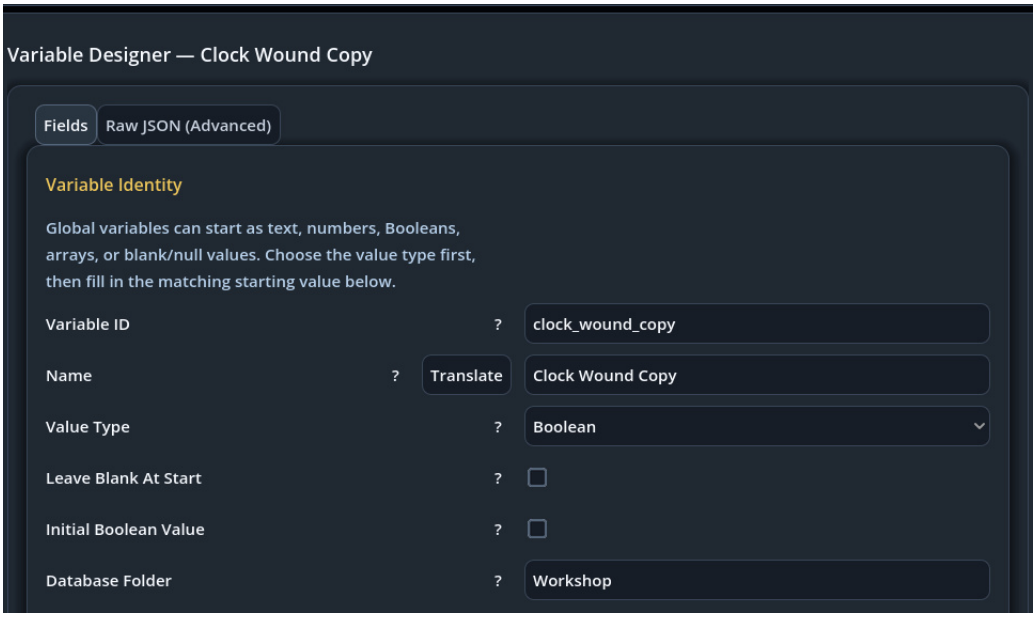

Figure 10. A copied variable is a separate record. Changing it does not change clock_wound.

If you intended the new puzzle to share the original state, keep using the original ID instead. Copying the record creates a second value; it does not make an alias for the first one.

Delete with the references in mind

Select the unwanted record, click Delete and read the confirmation. Deletion can be undone through the editor’s Undo history. Before keeping the deletion, remove or replace actions, GUI tokens and script references that still use that ID.

> Renaming or deleting a record does not guarantee that every hand-written script string or text token is repaired. Run Validate, then test the affected interactions. A display-name change is usually simpler than an ID change.

Read the record in Raw JSON

The Fields tab is enough for ordinary editing. Raw JSON (Advanced) is useful when you want to inspect the exact stored value, especially the difference between null, "", a number and quoted text.



Figure 11. Last Choice in Raw JSON. Its value is null, without quotation marks.

```
{
  "id": "last_choice",
  "name": "Last Choice",
  "type": "string",
  "value": null,
  "folder": "Workshop"
}
```

The important property is value. That is the starting value used by a fresh runtime. Type is one of string, boolean, array or number. The folder helps you organise the record; it does not change its lifetime or who can read it.

1. Make a small edit in Raw JSON only if you need one. Keep the surrounding braces, commas and quotation marks valid.
2. Click **Validate and Apply JSON (Advanced)**.
3. Return to Fields, check the result and save the project.

JSON uses straight double quotes, lower-case true, false and null, and no trailing comma after the final property. There is no Dictionary entry in Value Type. The structured list example in Chapter 15 uses dictionaries as members of an Array.

9. Change a variable from an object

Now give the clock something to do. This first version deliberately runs every time the player uses it. That makes the change easy to see. Chapter 12 adds the check that turns it into a one-time puzzle reward.

1. Open **Scene Designer**, select the clock or your chosen object, and open its Inspector.
2. Open **Verb Actions** and find **Use Verb — Step 1**. This walkthrough uses an interaction mode that executes verb actions.
3. Choose **Set Variable** as the Action.
4. Select **Clock Wound(clock_wound)** in Variable and enter `true` in Value. Use lower-case true without quotation marks.
5. Click **Apply Changes**. Save the project before testing.

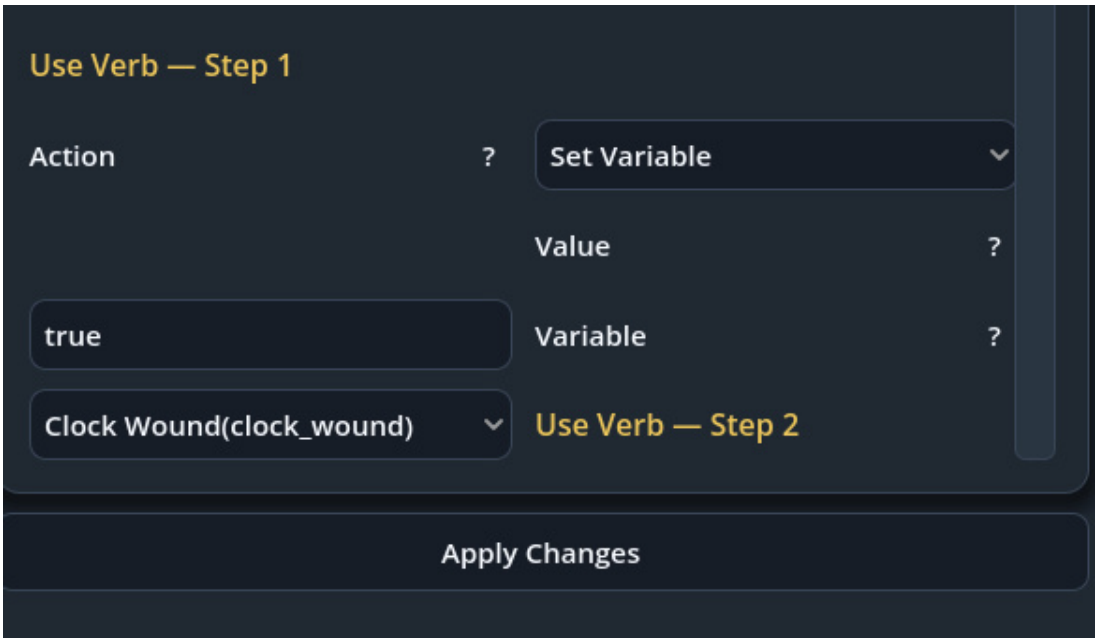

Figure 12. The clock’s first Use action. The Value is the Boolean true and the selected variable is clock_wound.

Keep one route to the interaction

Use the existing verb setup from your game. In a verb-based mode, choose Use and click the object; a context interaction can use the object’s Default Verb. If a pointer event is set to override the usual verb route, check that event rather than adding the same reward in two places.

Start a fresh preview and use the object. The stored value should now be True. Nothing visible has to change yet: the Debug Console can confirm the result, and a GUI label can display it later.

> In the screenshot, controls wrap to fit the Inspector. Follow the field labels rather than assuming Value and Variable will always sit on the same row.

Add points and update the objective

Under the Use action group, Add Next Action appends another step. Actions run from top to bottom. Build the sequence below, apply the Inspector and save the project.

Step	Action	Settings
1	Set Variable	Variable clock_wound; Value true.
2	Change Variable	Variable score; Operation add; Value 10.
3	Change Variable	Variable attempts; Operation add; Value 1.
4	Set Variable	Variable objective; Value "The clock is running."
5	Show Message	Text: You wind the clock. Score: $$score. Attempts: $$attempts.

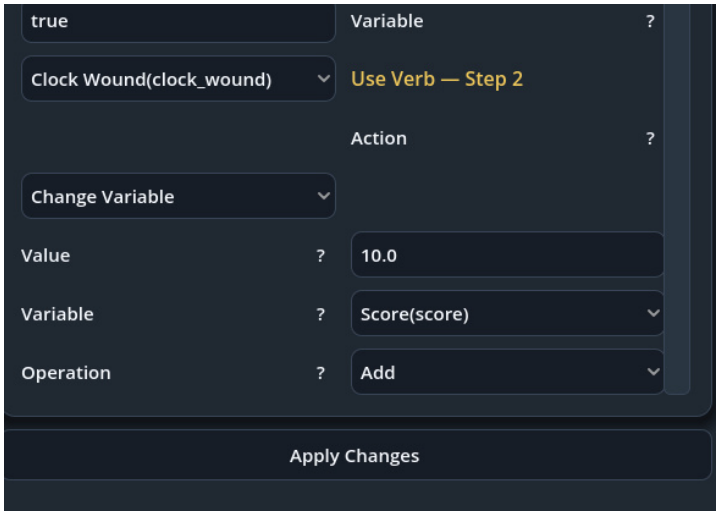

Figure 13. Change Variable adds to the score. It does not replace it with the reward amount.

Values in these general action fields

Intended value	Enter in an action-list Value field
Boolean	true or false
Number	10 or 2.5
String	"The clock is running."
Array	["clock diagram", "winding socket"]

The table uses JSON value notation. Ordinary unquoted words can fall back to text, but quotes make String values unambiguous here. For Show Message’s dedicated Text / Message field, type the sentence normally. For null, use the explicit options described in Chapter 6.

10. Pick the right change operation

Set Variable replaces the complete value. Change Variable offers a choice of operations, so it is the useful action for counters, toggles and list additions. Choose the operation according to the kind of data you are keeping.

Operation	Example	Result or rule
set	score 7; set 10	Replaces 7 with 10. For an Array, replaces the whole list.
add	score 7; add 10	17. Adds the numeric amount.
subtract	score 7; subtract 2	5. No automatic minimum of zero.
multiply	score 7; multiply 2	14. Numeric multiplication.
divide	score 7; divide 2	3.5. A divisor of zero leaves the value unchanged.
append	clues ["note"]; append "key"	["note", "key"]. Adds one member to an Array.
toggle	clock_wound False; toggle	True. Use this with an intended Boolean.

Choose actions that match the meaning

To mark a puzzle solved, Set Variable → true is usually clearer than toggle. If the player uses the solved object again, a toggle would turn the flag back off. Toggle fits a switch that is meant to alternate between on and off.

Append is an Array operation. It is not a text-joining command, and it does not prevent duplicate members. If you append an Array, the new member is that Array, nested inside the first one. To replace an entire list, choose set.

Add Score is also available as a separate action. It targets the score value and uses an integer amount. Change Variable → add lets you work with any chosen numeric variable.

> The declared designer type is guidance for authoring and validation. It is not a hard lock on every runtime assignment. An action can still write text into a variable you intended to be a Number. Consistent inputs and play tests matter.

11. Watch values during a test

Use the runtime preview to check what the game is doing. Variable Designer keeps showing the starting definition; it is not a live values window.

1. Apply the object actions and save the project.
2. Start a fresh preview with **Play / Restart**, or **Test Scene** for the room you are working on.
3. Click **Show Debug Console** below the preview.
4. Enter `get clock_wound` and submit the command. Before using the clock, the result should be false.
5. Use the clock, then enter `get clock_wound`, `get score` and `get attempts` one at a time.

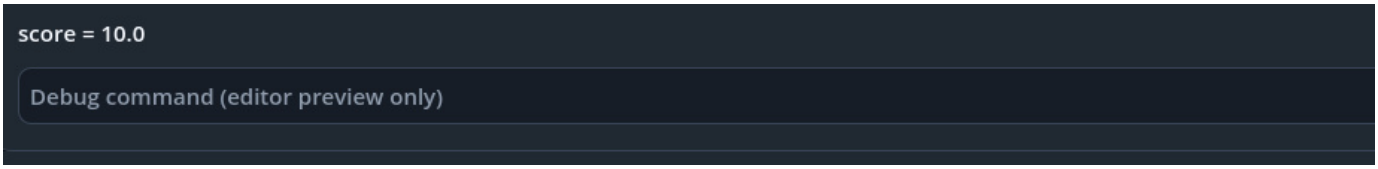

Figure 14. The console reports score = 10 after the first test. A get command prints a result at the time you submit it.

Test in the unguarded version	clock_wound	score	attempts
Fresh preview	false	0	0
Use the clock once	true	10	1
Use it a second time	true	20	2

That second reward is expected in this early version: there is no condition yet. It shows why a stored flag must actually be checked before awarding points. Chapter 12 makes that change.

> A console result is not a live watch. If you change a value after asking for it, the old printed result does not update itself. Run get again before deciding that the value is wrong.

Try a value without changing the project

The console’s set command changes the current test session. It is useful for jumping to a puzzle state without playing the whole opening again. These edits do not change the variable’s starting record.

```
get score
set score 42
set clock_wound true
set objective "Find the key."
set clues ["note", "key"]
set last_choice null
get last_choice
```

Command value	Stored type
true	Boolean
"true"	String containing the word true
42	Number
"42"	String containing two digits
["note", "key"]	Array with two String members
null	Blank/null in this Debug Console command

Make deliberate tests, then restart

Try setting clock_wound to the quoted String "true" and inspect it with get. It can look similar to a Boolean when displayed as text, but a condition expecting Boolean True should not be built around that String. Restore the correct value or start a fresh preview afterward.

A new preview should return clock_wound to False and score to 0, provided those are still their starting values and you have not loaded a saved game. Save Project does not copy test-session values back into Variable Designer.

A misspelt ID can create a different runtime key when you set it. If get returns <undefined>, compare the spelling with the designer record. Do not keep compensating by setting several slightly different names.

The Debug Console shown here belongs to the editor preview. It is a development aid, not a command-entry interface you need to ship with your game.

12. Award points only once

The flag now needs to control the reward. The rule is straightforward: if clock_wound is True, say that the clock is already running. Otherwise, set it to True and award ten points.

1. Open **Visual Script Designer** and create a script named Wind Clock Once, ID `wind_once`.
2. Use an **On Interact** entry node and set its verb to Use.
3. Add **If Variable**, **Set Variable**, **Change Variable** and two **Show Message** nodes. Use Add Node or find the node in the palette.
4. Give each message its appropriate text: “The clock is already running.” or “The clock starts ticking. Score: $$score.”

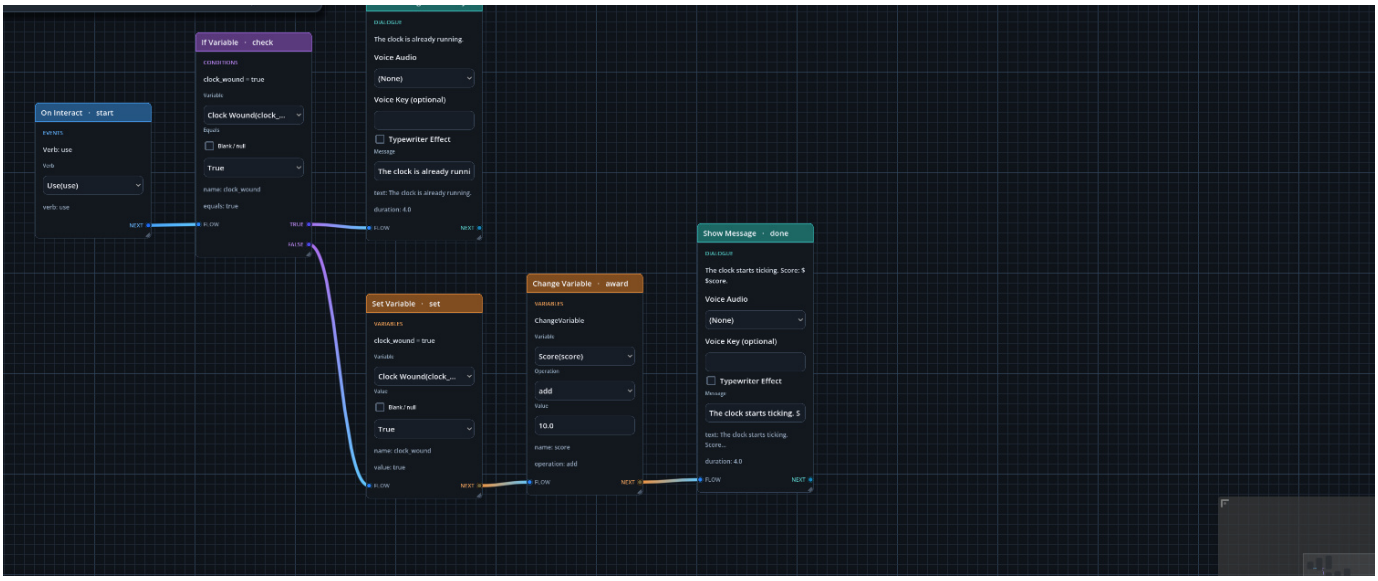

Figure 15. The complete graph. TRUE takes the short “already running” branch; FALSE takes the lower reward branch. Close-ups follow.

This compact version checks the flag and adds points. It deliberately leaves out the attempt counter and objective update from the earlier test sequence. Keep those extra actions on the appropriate branch if your game needs them.

> A line between nodes controls the order. A node sitting nearby without a connecting flow line will not run as part of this interaction.

Set the condition and connect both branches

Node	Settings
If Variable	Variable Clock Wound(clock_wound); Equals True; Blank / null off.
Set Variable	Variable Clock Wound(clock_wound); Value True; Blank / null off.
Change Variable	Variable Score(score); Operation add; Value 10.
First Show Message	The clock is already running.
Second Show Message	The clock starts ticking. Score: $$score.

1. Connect On Interact’s NEXT output to If Variable’s FLOW input.
2. Connect If Variable’s TRUE output to the “already running” message.
3. Connect its FALSE output to Set Variable.
4. Connect Set Variable → Change Variable → the “starts ticking” message.
5. Use Auto Arrange or move the nodes until you can follow both paths. Save the project.

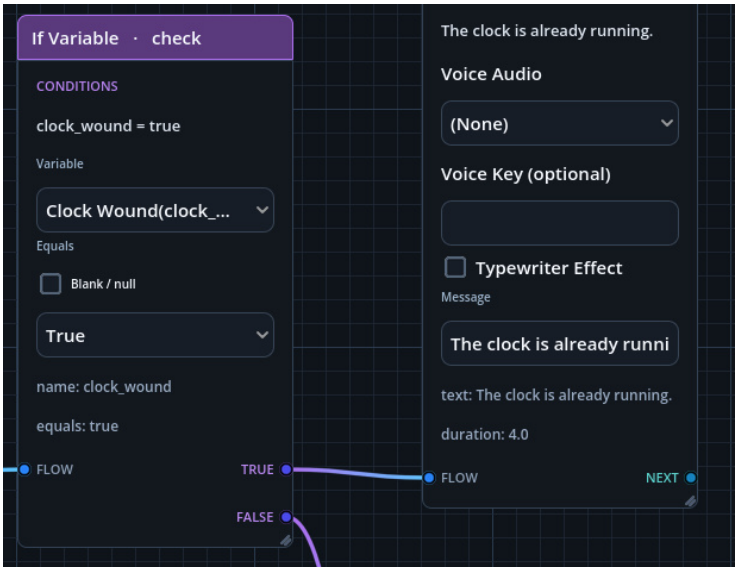

Figure 16. The condition compares clock_wound with Boolean True. The TRUE and FALSE sockets lead to different actions.

If the variable is False, the equality test is False, so the game follows FALSE and performs the reward. After Set Variable changes the flag to True, later calls follow TRUE instead.

Connect the graph to the clock and test

1. Return to the clock’s **Verb Actions** in Scene Designer.
2. Replace the old unconditional Use sequence with one **Run Visual Script** action. Select Wind Clock Once(wind_once) as Script.
3. Remove the old reward steps from this Use group, or choose No Action for those steps. Do not leave an extra add-10 action after the graph call.
4. Apply, save and start a fresh preview. Use the clock, let the response finish, then use it again.

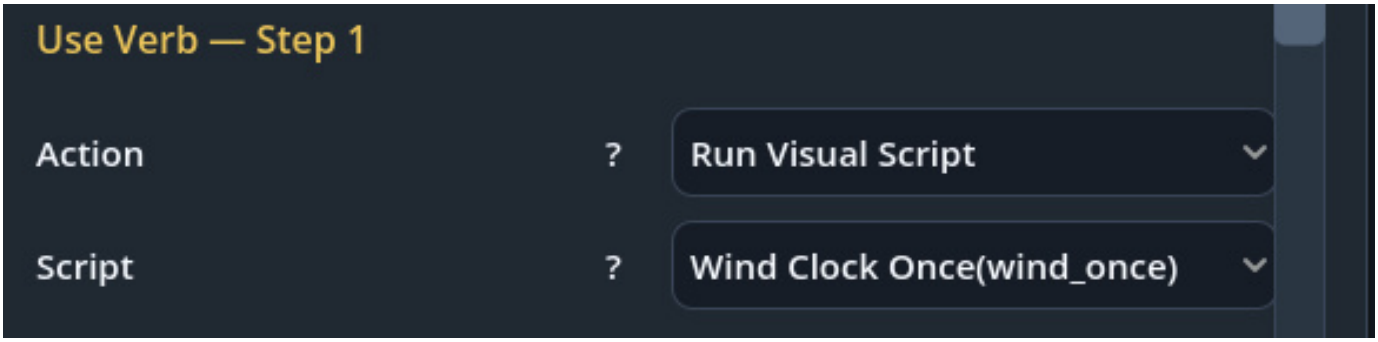

Figure 17. The object calls Wind Clock Once. The native action-list label in this release is Run Visual Script.

When you use the clock	Expected result
First time	clock_wound becomes True; score becomes 10; the success message appears.
After that response finishes	The “already running” message appears; score stays at 10.
After a fresh restart	The starting False and 0 values return. The reward is available again.

Do not test the second click while the first interaction is still busy. That click may be ignored by the interaction lock; it does not prove that your condition worked. Wait for the response to finish and check the second message as well as the score.

This same graph can be called from another supported action list. Keep one shared check for a one-time reward so a GUI button and an object click cannot accidentally use different rules.

13. Compare amounts with If Compare

If Variable asks whether a value equals a chosen value. For “at least ten points”, use If Compare. The left-hand side must read the score variable, not compare the literal word score.

1. Create a visual script called Score Check, ID `score_gate`, with an On Interact entry.
2. Add **If Compare**. In **Left**, enter `{"$var":"score"}`.
3. Choose >= in **Compare** and enter `10` in **Right**.
4. Connect TRUE to a Show Message saying “You have enough points.” Connect FALSE to “You need ten points.”
5. Call the script from an interaction. Test once at score 0 and again at score 10.

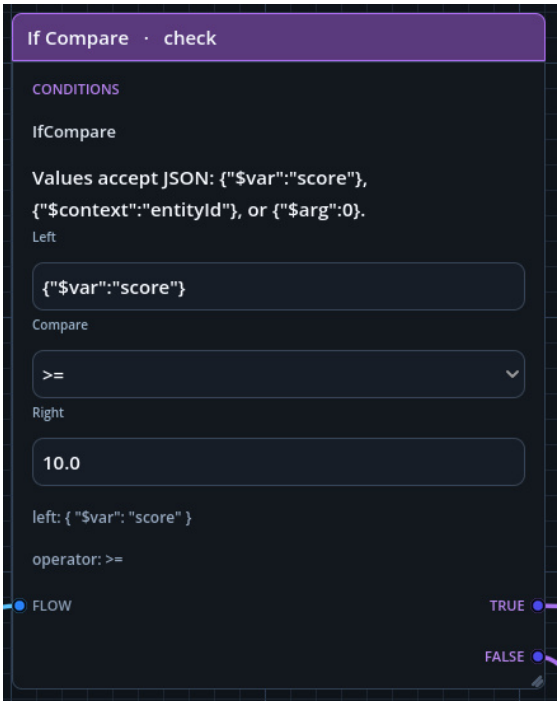

Figure 18. A numeric comparison reads the live score through a $var reference. The two outputs handle the success and failure cases.

Entry	Meaning
{"$var":"score"}	Resolve the current value of the global variable score.
"score"	The literal String score. It does not fetch a variable.
10	A numeric threshold.

Available comparisons include ==, !=, <, <=, > and >=. Use numbers on both sides of an amount comparison. The reference object uses one key, $var, with the exact variable ID as its value.

14. Put live values in your GUI

A variable becomes much easier to test when you can see it in the game. A normal GUI label can display a current value by using two dollar signs followed by its ID.

1. Open **GUI Designer** and choose a GUI that is shown during gameplay.
2. Add or select a **Label** element.
3. Open **Text & Content** in its properties. In Text, enter `Score: $$score`.
4. Make the label wide enough for the result. Repeat for `Objective: $$objective` if you want an objective line.
5. Save the project, start a gameplay preview and run an action that changes the score or objective.

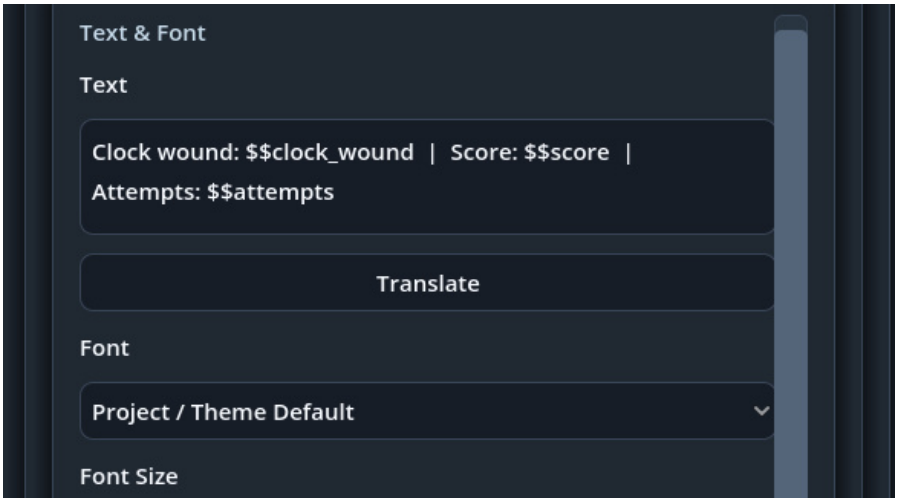

Figure 19. The label’s Text field contains variable tokens. This example combines a Boolean and two counters.

Label text	Example result during play
Score: $$score	Score: 10.0
Clock wound: $$clock_wound	Clock wound: true
Objective: $$objective	Objective: The clock is running.
Clues: $$clues	Clues: ["clock diagram", "winding socket"]

The usual variable actions notify the interface when state changes, so these labels can refresh. The same $$ tokens can be used in a Show Message text. A message shows the value when it is produced; a gameplay label can keep reflecting later changes.

GUI Designer can show the token while you are authoring. Check resolved values in a running game preview. Use the exact ID after $$; parser tokens such as {command} have a different purpose. If an unknown $$ token remains visible, check that the variable exists and the spelling matches.

15. Feed an Array into a list box

A list box can read its contents from an Array and write the player’s selection to another variable. That is useful for a clue notebook, a list of known topics or a puzzle that asks the player to choose an answer.

Keep the contents and the selection separate. The contents variable holds every available item. The selection variable holds the ID of the one the player picked.

1. Create an **Array** variable, ID `clue_options`, Name Clue Options. Leave Blank At Start off.
2. Enter the JSON below as its Initial Array Value. Apply and save.
3. Use the String variable `last_choice` from Chapter 6 for the selected ID. It can start blank/null.

```
[
  {"id":"diagram", "text":"Clock diagram"},
  {"id":"socket", "text":"Winding socket"}
]
```

Connect the GUI element

1. In GUI Designer, add or select a **List Box** element.
2. Open **Type Settings** and find **List Box Content & Behavior**.
3. Set **Item Source** to `array`. Keep **Allow Multiple Selection** off for this example.
4. Set **Selected Item Variable** to `last_choice`.
5. Under **Array Source**, set **Array Variable ID** to `clue_options`. Save the project.

> The small {...} entries are dictionaries inside an Array. You still choose Array in Variable Designer; there is no separate Dictionary Value Type in this release.

The entries can also carry supported optional fields such as iconAssetId and disabled. Use a stable item ID that makes sense to your conditions. The display text can be more descriptive than that ID.

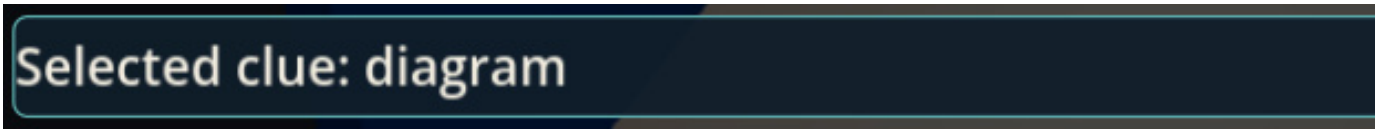

Figure 20. During play, the label resolves last_choice to diagram after Clock diagram is selected.

Read the selected ID, not its display text

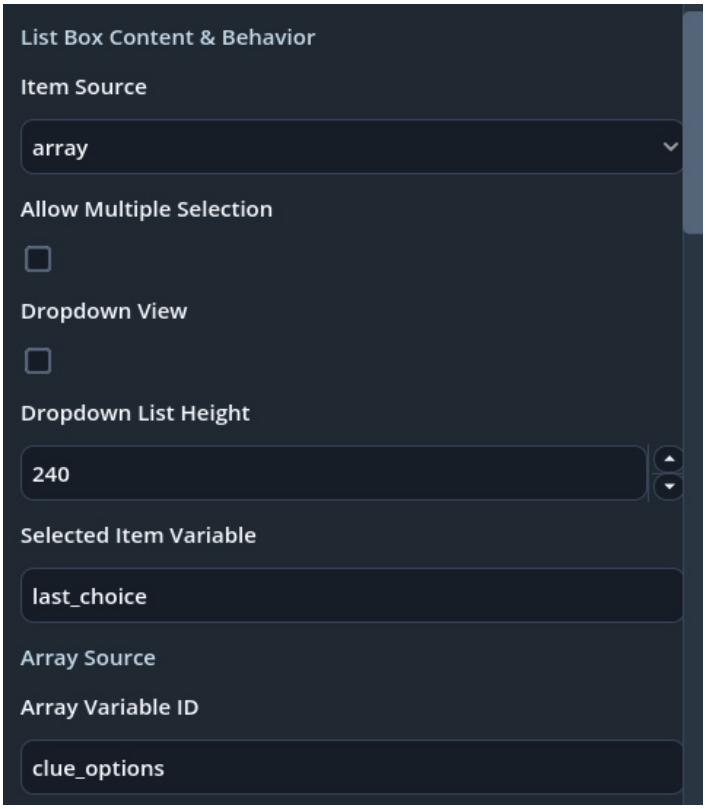

Figure 21. The list reads clue_options and writes its selection to last_choice. These are two different variable IDs.

1. Add a Label with text `Selected clue: $$last_choice`.
2. Start the game preview and select Clock diagram in the list.
3. The label should show `Selected clue: diagram`. Use `get last_choice` in the Debug Console to confirm the stored ID.
4. Select Winding socket. The stored ID should become `socket`.

The list writes the item ID, not the display sentence. That means your condition can compare last_choice with "diagram" even if you later change the displayed wording.

List setup	Selection value to expect
Single selection	One item ID, stored as a String.
Allow Multiple Selection on	An Array of selected item IDs. Use an Array selection variable.
Source is a plain String array	Generated item IDs such as item_1 and item_2, rather than the displayed text.

For a puzzle that depends on particular answers, explicit id/text records are easier to maintain than generated item IDs. A single-selection result should not share the same variable as the source Array.

16. Use the same values in GDScript

Actions, visual nodes and native scripts can all use the same runtime variable store. You do not need a separate score for each language. The GDScript example on the next page performs the same one-time reward as the visual graph.

1. Open **GDScript Designer**, create a record with ID `wind_script` and give it a readable name.
2. Use the source on the next page. Keep the extends line: it supplies the PCA action helpers.
3. Click **Validate**, then **Apply Changes**, and save the project.
4. In the object's Use actions, choose **Run Visual Script** and select this GDScript record in Script. That selector includes the supported script kinds despite the action's label.
5. Replace the old implementation for this test. Start a fresh preview, use the object once, wait, and use it again.

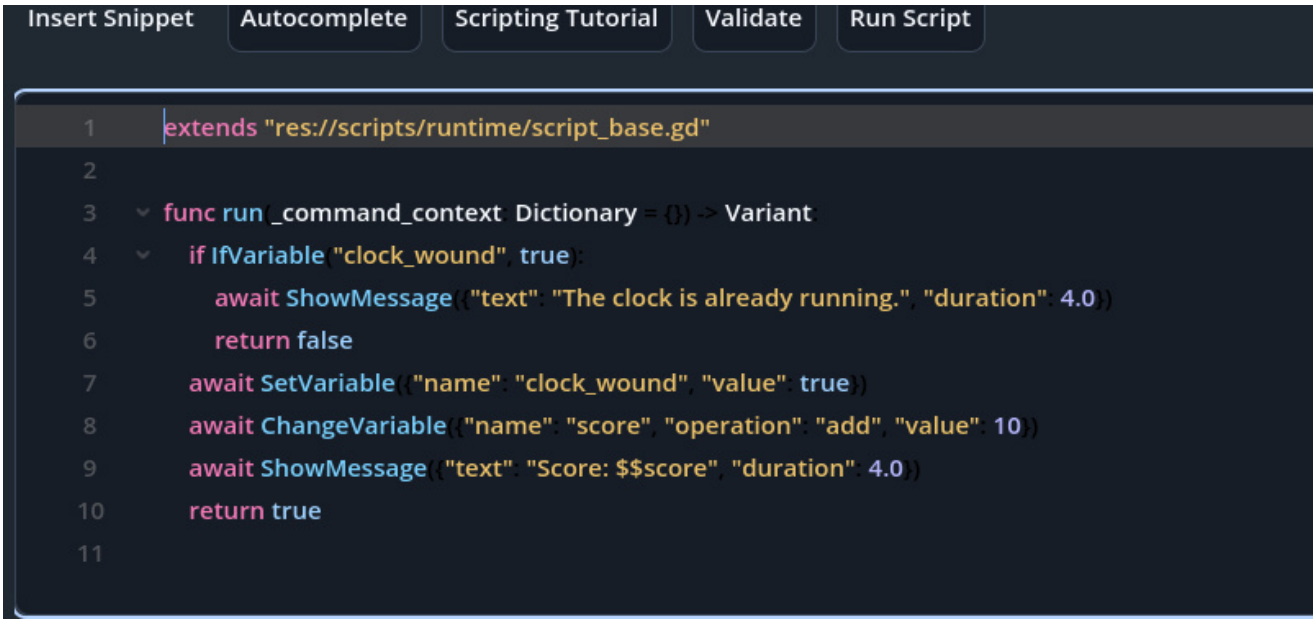

Figure 22. The guard implemented in GDScript Designer. The full, readable source follows on the next page.

Shared variables and local variables

A GDScript declaration such as var score = 0 inside a function creates a local value. It is not the global score record from Variable Designer. Use the PCA variable helpers, the Variable registry, or an explicit runtime read when you intend to use the shared value.

Keep global instance alive is about the lifetime of a script instance. It does not turn every script member into a designer variable or automatically save arbitrary member fields in a game slot.

> Choose one implementation for a single interaction while learning. Running the visual reward and the GDScript reward as extra sequential steps makes it harder to tell which action changed the state.

A complete GDScript puzzle check

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

func run(_command_context: Dictionary = {}) -> Variant:
    if IfVariable("clock_wound", true):
        await ShowMessage({
            "text": "The clock is already running.",
            "duration": 4.0
        })
        return false

    await SetVariable({
        "name": "clock_wound", "value": true
    })
    await ChangeVariable({
        "name": "score", "operation": "add", "value": 10
    })
    await ShowMessage({
        "text": "Score: $$score", "duration": 4.0
    })
    return true
```

IfVariable checks the current Boolean. The early return ends this invocation after the “already running” message. It does not undo the flag. On the first call, the code continues, sets the flag and adds ten points.

Read a value when the script needs it

```
var current_score = Variable.score.value
var safe_score = Runtime.variables.get("score", 0)
Variable.score.value = current_score + 5
```

Variable.score.value reads or writes the live shared value. The runtime .get form lets you supply a fallback if the ID is absent. Prefer the helpers or registry for writes: they notify the game so displays can refresh.

If you write directly to Runtime.variables in advanced native code, call Runtime.state_changed() afterward when the interface needs to refresh. The parameter name _command_context avoids confusing the run argument with the base script’s context member.

17. Share a value with PCA Script

PCA Script has its own language for shared variables. A global declaration connects to the runtime store, so the score below is the score from Variable Designer. Its initial expression is used when that global value is missing; it does not reset an existing score every time run is called.

1. Open **PCA Script Designer** and add a script with ID `award_pca`.
2. Enter the source below, validate it and apply the changes.
3. Call the script from a suitable action list, then inspect score in the gameplay preview.

```
language 1.0

global score = 0

function run(context)
    add 5 to score
    return true
end
```

What the five-point test proves

If score is 30 before this script runs, it should be 35 afterward. If it runs again, it should be 40. The global declaration is not an instruction to start over at zero on each call.

Use let for a temporary local value inside the script. A local calculation and a shared game value have different jobs; give them different names if that makes the code easier to follow.

PCA Script choice	Use it for
global	Shared game state that other actions or scripts need to read.
let	Temporary work in the script, such as a calculation or a loop value.

> PCA Script 1.0 does not use the native GDScript Variable.score.value syntax for these designer records. Use the language’s global declaration and commands, as shown here.

The next page shows the same five-point change in Lua and BASIC. Those are alternatives, not three rewards you must chain together for one puzzle.

The same change in Lua and BASIC

Lua

Create a Lua script record, validate and apply it, then call it from an action list. This version uses the shared ChangeVariable action helper.

```
local Script = {}

function Script.run(context)
    await(ChangeVariable({
        name = "score", operation = "add", value = 5
    }))
    return true
end

return Script
```

BASIC

Create a BASIC script record with this source. The wrapper identifies PCA 3D BASIC 1.0, and AWAIT calls the same variable-changing helper.

```
OPTION PCA3D BASIC 1.0
FUNCTION Run(context AS Dictionary)
    AWAIT ChangeVariable({"name": "score", "operation": "add", "value": 5})
    RETURN TRUE
END FUNCTION
```

Test run	Score before	Score after
Lua helper	20	25
BASIC helper	25	30
PCA Script global	30	35

These changes all reached the same shared score in the runtime test. Lua locals and ordinary BASIC local/private values remain script data; they do not become designer records just because they use the same name.

BASIC also provides GLOBAL / SHARED declarations for runtime state. For the small example here, using ChangeVariable keeps the operation explicit and avoids confusing local declarations with the global designer value.

18. Create a variable during play

Most puzzle facts are easiest to define in Variable Designer before the game starts. That gives them a known type, a starting value and an entry in reference selectors. Create Variable is useful when a script or action must introduce a key during play.

Create Variable field	Example
Variable ID	runtime_note
Type	string
Initial Value	"First value" in the action-list value field

Create Variable only creates the key if it does not already exist. Running it again with another initial value leaves the existing current value alone. Use Set Variable when you intend to replace that value.

```
await CreateVariable({
    "name": "runtime_note",
    "variableType": "string",
    "value": "First value"
})

await SetVariable({
    "name": "runtime_note", "value": "Replacement"
})
```

What it does and does not add

The new key belongs to the running game. It does not create a new authoring record in Variable Designer, so do not expect it to appear there after the test. It can be included in a saved game because saves keep the current runtime variables.

The Type choice is not a universal conversion step. Supply the correct kind of Initial Value yourself. In code, use true for a Boolean, a number for a Number and an actual Array for an Array.

During the test	Result
Create runtime_note = "First value"	The missing key is created.
Create it again = "Second value"	It remains "First value".
Set it = "Replacement"	It becomes "Replacement".

If a variable is central to the game and other designers should select it, define it in Variable Designer. Runtime-created keys are better kept for deliberate dynamic data than for working around a misspelt ID.

19. Room changes, restarts and saves

Designer variables belong to the current game state, rather than to the room in which they were first changed. If the player earns points in the workshop and walks into the hall, the same score is still available there.

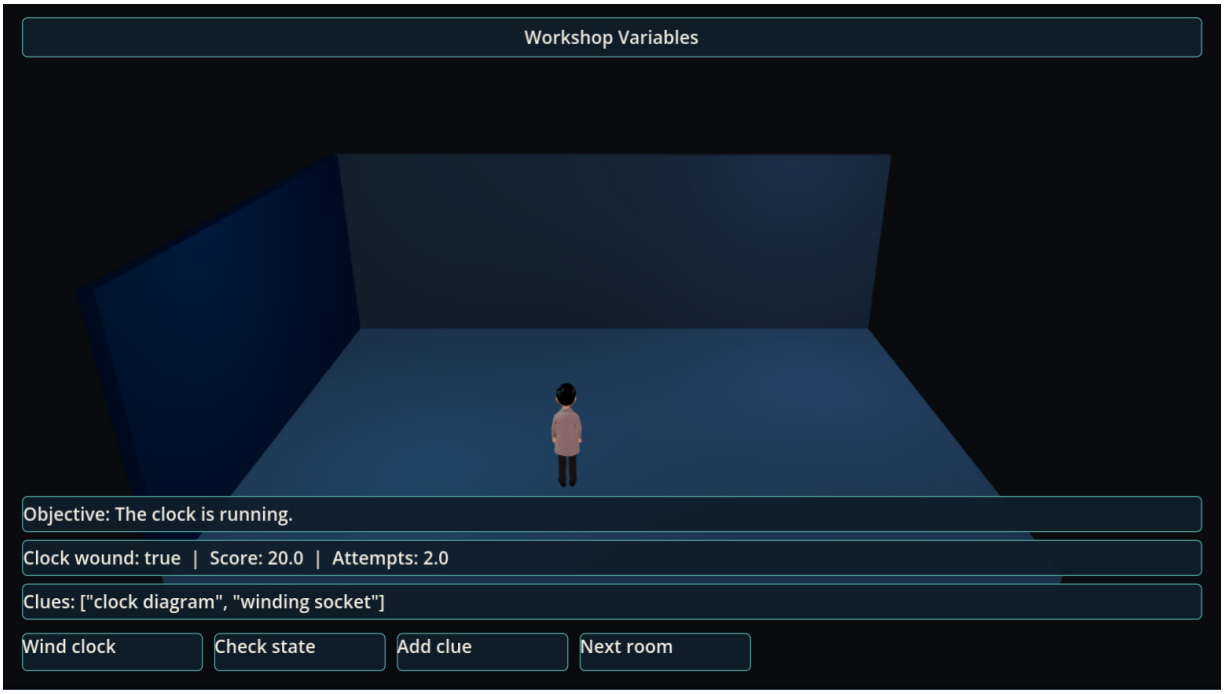

Figure 23. The preview has moved to the adjoining hall. The scene geometry changes while the shared puzzle values remain available.

Action	What happens to variables
Change to another room	The current runtime values continue with the game.
Stop, then start a fresh preview	A new runtime copies the authoring starting values. Startup actions can then change them.
Save Project	Saves designer definitions. It does not capture the current play session.
Save Game / a game slot action	Stores the current runtime variables with the saved game state.
Load Game / a game slot action	Restores the saved variables. It does not start each one again from its designer value.

The hall shown here is only a test of shared state; you do not need to build another room to use Variable Designer. You can check the same behavior with any two connected scenes in your own project.

A variable folder does not alter these rules. Neither does the fact that the action changing it belongs to a particular character or object. To keep separate facts, create separate IDs.

Test a saved value deliberately

1. Use a spare save slot in a test session. Get score to a known value, such as 20.
2. Run **Save Game Slot**, choose that slot and give the save a useful name if wanted.
3. In the Debug Console, enter `set score 999`, then `get score`. Confirm that the current value has changed.
4. Run **Load Game Slot** for the same slot. Allow loading to finish.
5. Submit `get score` again. It should report the saved 20, not the temporary 999.

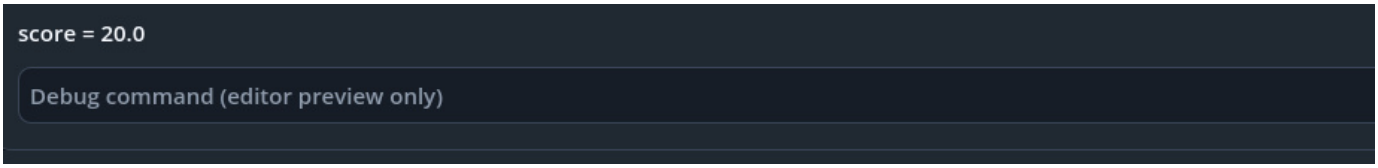

Figure 24. A fresh console query after loading reports the restored score. Re-query after load; old console output may still show the earlier test value.

Why an old save can confuse a new test

Suppose you change the starting score to 5 in Variable Designer, then load a slot saved when the score was 20. The restored score should be 20. The save is preserving progress, which is what the player expects.

A save made before you added a new variable can also lack that key. Loading restores the saved variable dictionary; it does not necessarily fill every new designer definition back into that older state. Use a fresh game when checking new starting values.

For a released game that must support old saves, plan an update step that supplies missing keys without overwriting existing progress. Create Variable’s “only if missing” behavior can help with that job when it is called at the appropriate point in your loading flow.

> Keep development save slots separate from any progress you care about. For the tutorial test, use a spare slot so the known-value experiment does not replace a save you wanted to keep.

20. Find the cause when a value is wrong

Start with the exact ID, the current value and the action that writes it. That is usually faster than changing several settings at once.

What you see	Check this first
The designer still says False after play	It shows the starting definition. Use get clock_wound in the running preview.
Points rise every time the object is used	The reward is unconditional, or an old add action remains after the guarded script call.
The second click appears to do nothing	Wait for the first interaction to finish. A busy interaction can ignore another click.
A condition fails despite the word true showing	Check for the String "true" rather than Boolean true. Displayed text can hide that difference.
A numeric comparison gives an odd result	Check that both values are numbers. If Compare needs {"$var":"score"}, not the literal String "score".
A GUI shows $$score literally	Check the ID, spelling and that the record exists. Test in gameplay, not only the authoring canvas.
Array entries have the wrong types	Use valid JSON. The quick separated-list form creates String members.
Appending makes a list inside the list	Append adds one member. An Array operand becomes one nested Array.
A new initial value does not appear	Apply and save, then test a fresh game. An old save can restore earlier data.
get returns <undefined>	Check the ID and whether a loaded save predates that variable.

Change one thing, then run one known test. A label, a fresh get query and a single interaction are enough to narrow down most variable problems.

Validate the project and trace the writer

1. Apply the edited records and click the main **Validate** button.
2. Read both errors and warnings. Check duplicate IDs, missing variable references and starting values that do not fit their declared type.
3. Open the action or record named in the result and correct the specific field.
4. Start a fresh preview. Read the value before the interaction, run the interaction once, then read it again.
5. If it changes twice, inspect every route that can run: verb actions, pointer events, GUI events, room events and script calls.

Know what validation can tell you

The validator can catch useful authoring problems, including duplicate variable IDs, missing references in supported actions and mismatches between a declared type and its initial value. Some of these findings are warnings rather than a complete block on play.

It cannot decide whether your puzzle story is correct. Two valid actions can still award the same ten points twice. An Array can be correctly typed while containing the wrong item IDs. Hand-written source and text tokens also need their own checks.

Question	A practical check
Who last changed this value?	Inspect the object or event that ran immediately before the result changed.
Is this a reset or a restore?	Check for a new runtime, startup actions or a Load Game action.
Is the variable intentionally shared?	Search for the same ID in the relevant actions, GUI text and scripts.
Is the input actually typed correctly?	Inspect Raw JSON for a starting value; inspect the writer for a runtime value.

If you rename an ID, work through its consumers deliberately. Keep the display name free to improve readability, but treat the ID like a connection point other parts of the project depend on.

The examples in this guide were checked through the ordinary designer, native runtime actions and script execution. Their purpose is to make the changes observable: initial values, repeated use, cross-language updates, list-box selection, room changes and save restoration.

21. Finish with one complete test pass

Before moving on to the next designer, run through a small test with known results. Do the unconditional-action test first, then replace it with the guarded version. Mixing both implementations makes the expected score unclear.

Check	Expected result
Fresh game	clock_wound False; score 0; attempts 0; objective starts with the workshop instruction.
First unconditional Use	Flag True, score 10, attempts 1 and the updated objective.
Second unconditional Use	Score 20 and attempts 2. This is the early test, not the final reward rule.
Fresh game with the visual guard	First Use awards 10; a later completed Use shows “already running” and leaves score at 10.
If Compare at 0, then 10	The failure message at 0; the success message at 10.
Append one clue	The Array gains one member. It does not become an inventory award.
Select Clock diagram in the list box	last_choice is diagram; the source Array remains the list of available clues.
Change to another room	The same current values remain available.
Save at 20, change to 999, reload	A fresh get query reports the restored 20.
Restart without loading	The designer’s initial values return.

A useful habit for the next puzzle

Write down the question first: “Has the player done this?” Choose the type that answers it, create a stable ID, and decide which action is allowed to change it. Then add one check that proves the rule works both before and after the change.

For the workshop clock, that is a False starting flag, one successful reward and a True branch for every later visit. The same pattern works for a drawer already searched, a clue already examined or a conversation already completed.

Apply the designer edits, save the project and keep a fresh-game test separate from a saved-game test. Once those agree with your intended rules, the variable has a clear job in the game.