PCA 3D ADVENTURE STUDIO

Buttons, Labels & Text Boxes

Live text, useful buttons and input the game can read

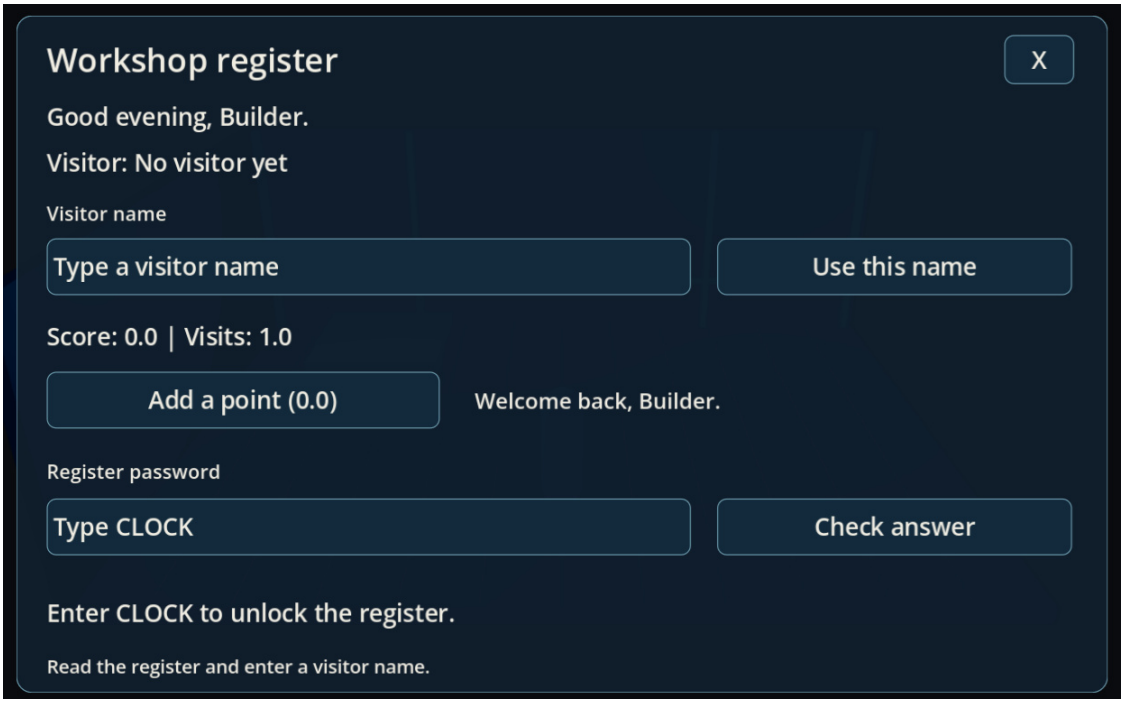

Figure 1. The workshop register during play. Labels display game information; the text boxes collect answers; the buttons decide what happens next.

Build a working register, display live values, read a player's answer and connect the interface to an adventure. Includes a reference to the supported text tokens.

Version 0.86.3.4 | Windows, macOS and Linux

Actual editor and gameplay screenshots.
No practice project or additional download required.

How to use this tutorial

Use an existing project or one of the bundled demos. The pictures use a small interface over a primitive workshop scene. You can build the same controls in your own room; you do not need the pictured project.

The main example is a Workshop Register. A label greets the active player. A text box accepts a visitor name. A button copies that name into a variable. A second text box and button make a small password puzzle. Other pages show notes, text-parser entry and display masks.

For this job...	Start here
Place and style the three controls	Chapters 1-3
Show live values and use wildcards	Chapters 4-6
Give buttons useful actions	Chapters 7-9
Set up editable text and Enter behavior	Chapters 10-11
Read a name or check a puzzle answer	Chapters 12-13
Keep notes, mask text or use a cipher	Chapters 14-15
Connect a command box to the parser	Chapter 16
Runtime properties, dialogue and final checks	Chapters 17-20

Save Project writes your authoring work. A gameplay test has its own running values. Stop and restart to check the intended starting state, rather than treating the last test's score or typed text as the new project default.

The examples use fixed IDs so the instructions remain easy to follow. Substitute your own IDs consistently. The version number matters: later releases may move a field or add another option.

Contents

Click a chapter or page number, or use the PDF bookmarks for the smaller sections.

How to use this tutorial . . . 2

1. Give each control one clear job . . . 4

2. Prepare a small working interface . . . 5

3. Make a readable label . . . 7

4. Display a live variable . . . 10

5. The built-in @ tokens . . . 12

6. Cycle or randomise alternate wording . . . 15

7. Make a button do one useful thing . . . 17

8. Give the button normal and held artwork . . . 19

9. Open a panel or choose a verb . . . 21

10. Add an ordinary text box . . . 22

11. Decide what Enter should do . . . 24

12. Read and save the entered name . . . 26

13. Turn an answer into a small puzzle . . . 29

14. Use a multiline box for notes . . . 32

15. Mask a single-line answer . . . 34

16. Make a parser command field . . . 37

17. Change a control during gameplay . . . 39

18. Use labels for dialogue and commands . . . 41

19. Diagnose text and input problems . . . 42

20. Finish with a complete player test . . . 44

1. Give each control one clear job

Control	Its ordinary job	Workshop example
Label	Show information the player reads.	Good evening, @PlayerName.
Button	Offer an action the player can choose.	Use this name or Check answer.
Text Box	Let the player enter or edit text.	A visitor name, a command or a puzzle answer.

A label can display a variable without changing it. A text box can collect a name without automatically saving it. A button can look clickable without having an action. The useful connection is the one you deliberately make between them.

Keep names, IDs and visible words separate

Field or value	Example	Why it matters
GUI ID	register_panel	Names the whole interface.
Element ID	name_box	Names one control inside it.
Editor Name	Name Box	Helps you find that control in the editor.
Displayed text	Use this name	The words the player sees on a button.
Variable ID	visitor_name	The game value the label or script will use.

Use stable IDs, such as name_box, and readable editor names, such as Visitor Name Entry. Giving the text box the same name as a variable does not bind the two together.

> An event can also make a label clickable. That is useful for a small help link or a dialogue line, but use a recognisable Button for a main command so the player can tell it is interactive.

2. Prepare a small working interface

1. Open **GUI Designer**. Click **Add New** beside the GUI selector.
2. Give the GUI ID `register_panel` and Name Workshop Register. Apply the whole-GUI form.
3. Choose **GUI Window** and set GUI Width to `1000` and GUI Height to `620`.
4. Enable a window background and border so the room does not compete with the text.
5. In **Layout & Appearance**, set **Anchor Entire GUI** to Center, with both offsets at 0.
6. Set Activation to **Called by Trigger Action / Visual Script**. Apply and save.

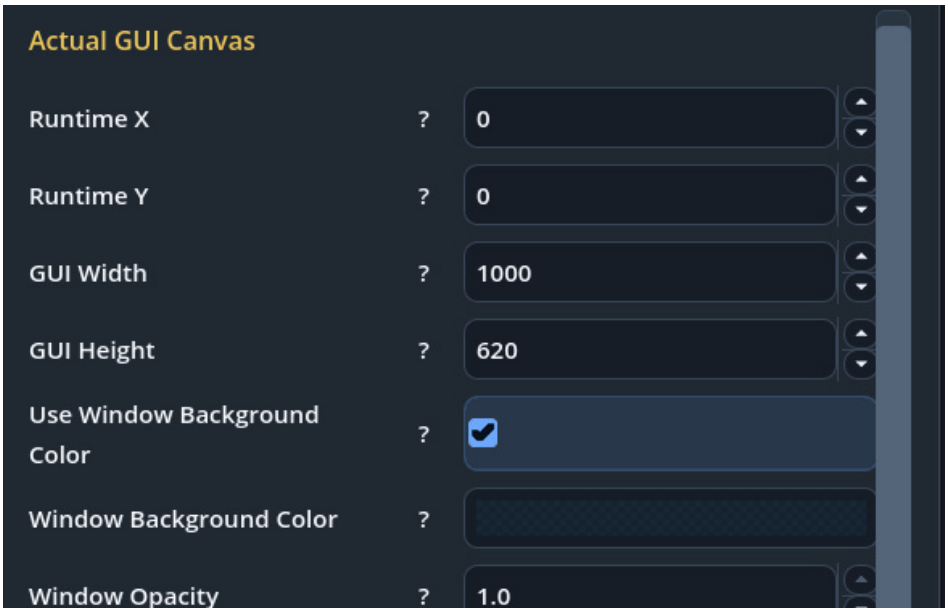

Figure 2. The GUI window owns the available space. A large background Panel would be a child element, not a replacement for the window dimensions.

For the example, a separate gameplay HUD has a Register button whose Click action is Show GUI, targeting register_panel. You can also show the panel from an existing scene action while you test it.

> If your HUD has not been assigned yet, use Global Settings > General > Custom System Interfaces > Main HUD. Designer Preview is for appearance and control states; run the game to execute Show GUI or any other event.

Place the register controls

Add the following controls with Add Element. These positions use top-left anchors inside the 1000 by 620 window. The empty text boxes will show their placeholders until the player types.

ID	Type	X, Y	Width, Height
register_title	Label	28, 18	840, 45
register_greeting	Label	28, 74	930, 38
register_visitor	Label	28, 116	930, 38
name_box	Text Box	28, 204	590, 52
name_submit	Button	642, 204	325, 52
register_status	Label	28, 275	930, 38
point_button	Button	28, 326	360, 52
answer_box	Text Box	28, 442	590, 52
answer_submit	Button	642, 442	325, 52
register_feedback	Label	28, 512	930, 70

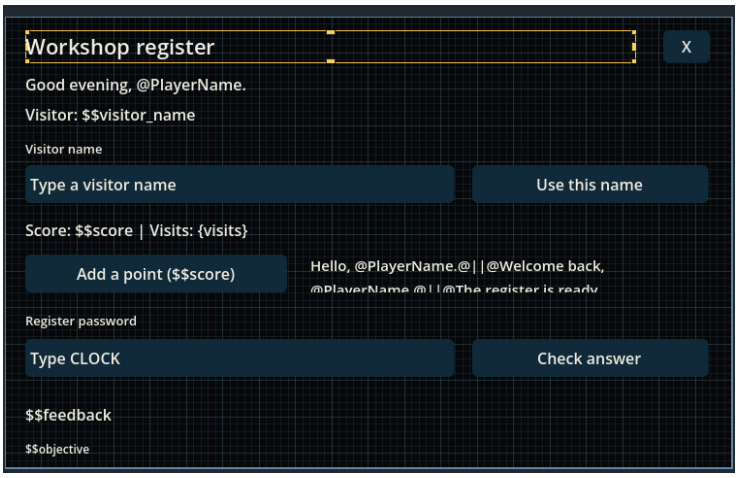

Figure 3. The register layout in the designer. The raw wildcard strings can be longer than their eventual displayed results.

Use the element list to select a control hidden under another. Enter exact measurements in General, or drag and resize it with Preview off. Give the sample elements distinct IDs rather than reusing one ID on several interfaces.

Editor zoom changes your view, not the saved dimensions. Snap controls placement increments. Lower the zoom to see more of the panel.

3. Make a readable label

1. Choose **Add Element > Label** and select the new element.
2. In **General**, set its ID, Editor Name, position and dimensions.
3. Open **Text & Content**. Enter `Good evening, @PlayerName.` in Text.
4. Choose a readable Font and Font Size. The example uses 22 for ordinary text and 30 for the heading.
5. Set Text Alignment to suit the frame. Save and inspect the result during gameplay.

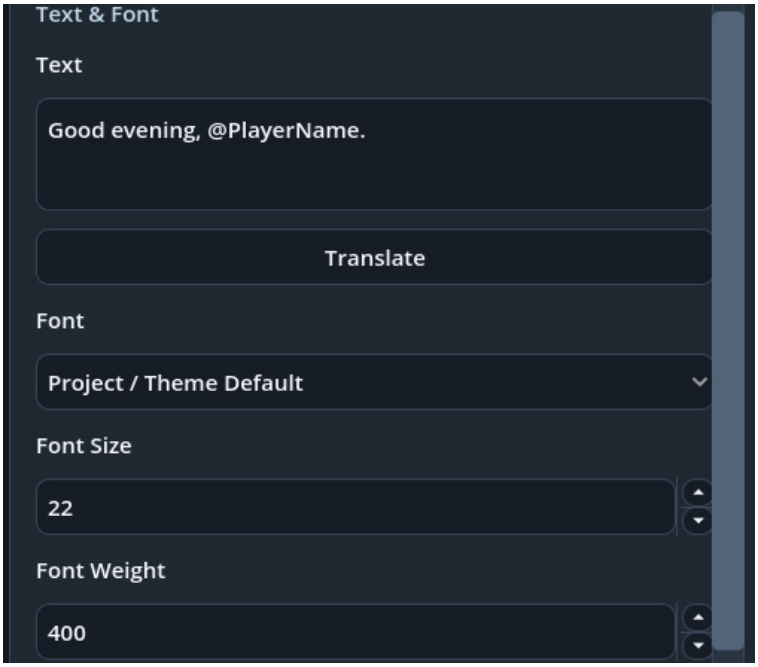

Figure 4. The greeting label's Text & Content settings. The Text field contains the authored sentence, including its token.

Give a label space before shrinking it

Labels wrap words within their available area. Make the frame wide and tall enough for the longest likely message. A fixed 38-pixel greeting may suit a short line, while puzzle feedback needs room for a second line.

Try a long player name and a longer translation. A layout that fits Builder may fail when a character has a much longer name. Read it at the game's actual size, not only at 200% designer zoom.

> A label is an output. If the player should type a replacement name or answer, use a Text Box rather than trying to make the label editable.

Set font, alignment and decoration

Setting	What it changes
Font	The project font, a system font or the default. Imported project fonts travel more predictably with a game.
Font Size / Weight	Letter size and the requested font weight. The chosen font must support the appearance you expect.
Bold / Italic	Emphasis for the text.
Underline / Strikethrough	Text decoration, independent of the control's border.
Text Color / Text Opacity	The lettering, separately from the surface behind it.
Text Alignment	The position of the text within the element frame.

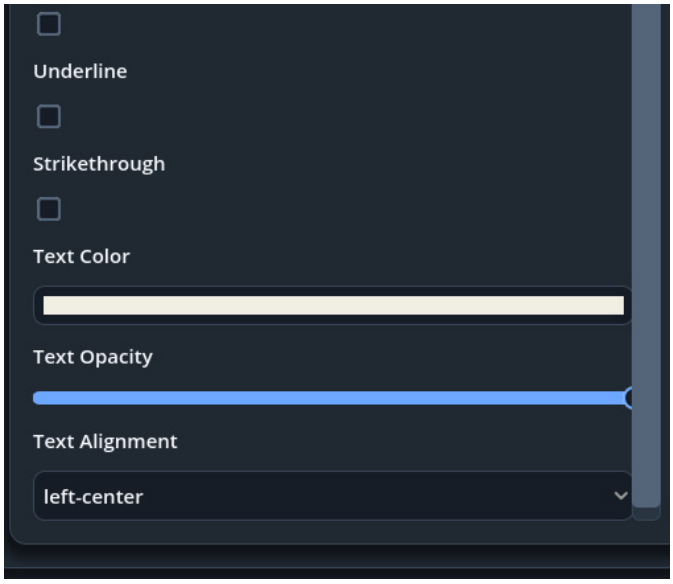

Figure 5. The lower text controls include color, opacity and alignment. Use them for the lettering, rather than changing the background opacity.

Use centered text for a short command button. Use left or left-center for information the player scans from the same edge. Keep button captions short; give longer explanations their own label.

A font installed on your machine may be absent on another system. Import the intended font through the asset workflow and choose that project font when the finished game needs a specific look.

Test accents and symbols with the actual font. A replacement square is usually a missing glyph, not a broken wildcard. Plain words and ordinary punctuation make a useful first test.

Style the frame without hiding the text

Visual Style controls the surface, outline, state colors and corners. Text & Content controls the lettering. Images supplies the artwork used behind the control or for its states.

Control	Useful starting approach
Background Color	Choose enough contrast against both the text and the room.
Surface Opacity	Make the surface quieter without assuming the text should fade by the same amount.
Border Color / Border Opacity	Keep the field or button edge visible over the scene.
Hover / Pressed Background	Use related colors that visibly acknowledge the pointer and a held press.
Border Radius	The default curve for the element's corners.
Four corner overrides	-1 inherits the overall radius; 0 is square; a positive number supplies that corner's curve.



Figure 6. Corner settings belong to this element. An explicit corner override can differ from the overall Border Radius.

The example uses dark blue surfaces and light lettering. The particular colors are optional. What matters is that the player can see the frame and read its contents over the room.

> Metallic Look and Reflection Strength are GUI surface finishes. They do not turn a text field into a mirror showing the 3D character. Use Element Shader sparingly when text still needs to be read.

4. Display a live variable

A variable holds a game value. A wildcard in a label or button caption displays that value. Use two dollar signs followed immediately by the exact variable ID.

Create this variable	Type	Starting value
score	Number	0
visitor_name	String	No visitor yet
feedback	String	Enter CLOCK to unlock the register.
objective	String	Read the register and enter a visitor name.
register_open	Boolean	false
notes_saved	String	Nothing saved yet.
visits	Number	1

1. Create the variables in Variable Designer and save.
2. Set Register Visitor's Text to `Visitor: $$visitor_name`.
3. Set Register Status's Text to `Score: $$score | Visits: {visits}`.
4. Set Register Feedback's Text to `$$feedback`.
5. Run the game. The labels should display the starting values, not the raw tokens.

A button caption can use the same text expansion. Add a point ($$score) shows the current score inside the button. Changing the caption does not itself award a point; the event supplies that action.

> Use the variable ID, not its display name. $$Visitor Name is not the same expression as $$visitor_name. The token ends at a character outside letters, numbers and underscore.

Understand what updates and what stays fixed

A live GUI label is refreshed as the game values change. The score label and the score inside Add a point can therefore stay in step. A Show Message sentence is expanded when that message is shown; it is not a permanent score display.

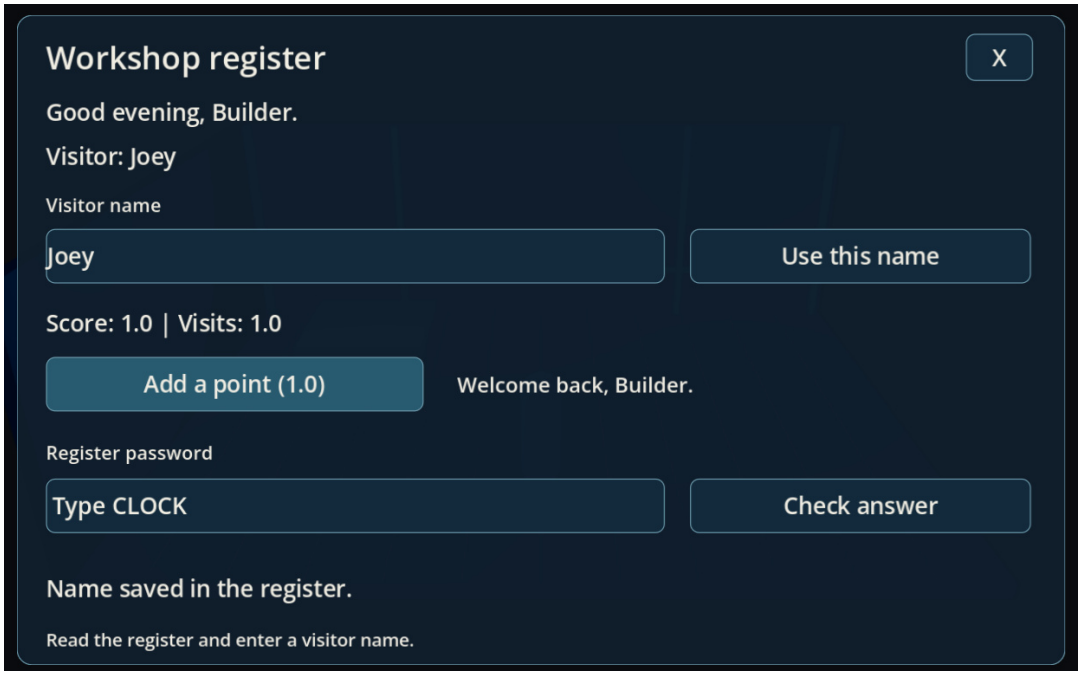

Figure 7. After one click, both the label and the button caption show the new score. The input and puzzle controls are separate from that display.

Authored text	What to expect
Score: $$score	Displays the current score variable.
Visitor: $$visitor_name	Displays the current visitor_name String.
Open: $$register_open	Shows the Boolean as its text representation. It does not choose a friendly sentence automatically.
Unknown: $$missing	The unknown variable token remains visible so you can find the missing ID.

Arithmetic can turn a whole amount into a numeric value displayed with .0. If presentation needs “6” instead of “6.0”, format a separate display String in a script. The wildcard does not contain a number-formatting command.

For a friendly Boolean result, set feedback to a sentence such as The register is open. A raw true/false value is useful during testing but is rarely the best finished message.

5. The built-in @ tokens

These four built-in text tokens are available through the game's text expansion. Their capitalization is part of the spelling.

Token	Meaning	Example in this tutorial
@PlayerName	The active playable character's display name.	Builder
@PlayerId	The active playable character's ID.	tester
@SceneName	The current room's name.	Workshop
@Score	The score variable; 0 when that key is absent.	6.0 after the example actions



Figure 8. The wildcard board during gameplay. The headings are ordinary labels; their neighbouring labels contain the tokens.

Use @PlayerName for the adventure character. It does not become the visitor name typed into name_box. That typed name is a different value, stored in visitor_name and displayed with $$visitor_name.

> @Score is a specific shorthand for score. @Coins is not a general variable syntax. Define a coins variable and use $$coins instead.

The GUI-only {brace} tokens

Labels and button captions have an additional GUI expansion step. These brace tokens are useful for a command line that follows the selected verb and hovered object.

Token	What a GUI label or caption displays
{command}	The current verb display name and hovered target name, combined and trimmed.
{verb}	The current verb's display name, such as Look.
{target}	The current hovered target label. It can be empty over blank space.
{player}	The active player's display name.
{playerId}	The active player's ID.
{value}	The element's runtimeValue, or its named variable's value; 0 is the fallback.
{max}	The element's max, or maxValue; 100 is the fallback.
{visitor_name}	A runtime variable whose exact ID is visitor_name. Other variable IDs can use the same brace form.

For a standard label or button with no value/max setup, {value} / {max} can simply read 0 / 100. These tokens do not mean “the words in a nearby text box”. Use the input-reading script in Chapter 12 for that.

Reserved GUI tokens are replaced before the general variable brace lookup. If a variable is called target, {target} still refers to the hovered object. Use $$target to display that variable without the reserved-token collision.

> For variable text you also want in Show Message, prefer $$variable. A GUI {command} means the current verb-and-hover display; it is not a general record of the last sentence typed into the parser.

Write tokens that cannot be misread

Write this	Avoid assuming this
$$visitor_name	A display name with spaces can replace the variable ID.
@PlayerName	@playername is the same spelling. It is not.
$$score points	$$scorepoints means score followed by points. It asks for a different variable.
Score: $$score.	A punctuation mark after the variable is part of its ID. The period is separate.
A script computes a value first.	$$score + 5 performs arithmetic. The + 5 is ordinary displayed text.
The exact variable key	The wildcard automatically reads a property such as $$player.health.
An output label with a token	An editable text box expands the same token. Its contents stay literal.

Choose clear variable IDs

The dollar form recognises a letter or underscore first, then letters, numbers or underscores. Use IDs such as visitor_name, score and clue_count. Avoid a leading digit, spaces or dots for values you intend to display this way.

Keep inserted values straightforward

A wildcard converts the stored value to text. It does not loop through an Array and create a polished list, format a date or choose singular/plural wording. Build a display String in a script when you need those choices.

Do not rely on nested token expansion. Keep your source sentence in the label and ordinary player-facing words in its variables. That is easier to translate and much easier to diagnose.

The runtime also accepts a legacy variable key beginning with one dollar sign as a fallback for the $$ form. For new work, use ordinary IDs such as score and let $$ be the display marker.

6. Cycle or randomise alternate wording

Two special separators choose an alternative before the other text tokens are expanded. Put a complete alternative on each side of the separator.

Separator	Result
@||@	Cycles through the alternatives in order, then returns to the first.
@//@	Chooses one alternative randomly. A later choice can repeat the same alternative.

```
Hello, @PlayerName.@||@Welcome back, @PlayerName.@||@The register is ready.
```

1. Put that complete string in a label on the register panel.
2. Start a fresh gameplay preview and open the register.
3. Read the greeting, close the panel and open it again.
4. Repeat once more. The sequential greeting should advance without flickering while the panel stays open.



Figure 9. The second opening uses the next greeting. The chosen line stays put while that GUI display remains open.

The GUI keeps the selected alternative for that display; its $$ variables can still update. Hidden GUI text can be prepared during startup, so the first visible line may already be a later choice. Use a variable for a real visit counter. Showing an already visible GUI is not another reopening.

> Use one separator type in a single source string. Do not mix cycling and random separators and expect nested choices. Other separators such as {one|two} are not substitutes for the supported syntax.

Translate the sentence, preserve its markers

Use Translate beside the authored Text field for the player-facing wording, or use the Game Translations workflow for the project. Keep the tokens exactly as written inside each translation.

English source	A valid French translation
Good evening, @PlayerName.	Bonsoir, @PlayerName.
Visitor: $$visitor_name	Visiteur : $$visitor_name
Score: @Score	Score : @Score

The game looks up the translated source text before replacing its tokens. Translate the sentence around the marker. Do not translate visitor_name into another variable ID, or change the case of @PlayerName.

Variables are not a translation system

A visitor name entered by the player should remain that person's name. A whole instruction stored in objective is different: do not assume that replacing $$objective also performs a new translation lookup on the inserted String. For reusable game prose, plan where the translated sentence will live.

A fixed button such as Return to game should normally use the authored Text translation. Use a variable for genuinely changing information, such as a score, entered name or changing status.

Allow room for the result

Keep buttons wide enough for longer translations, and give multiline feedback enough height. Test accented characters with the imported font. Include any @||@ or @//@ separator in the complete translated source when you use alternate wording.

> The Text Box placeholder is ordinary authored guidance in this release. Do not assume that a variable or wildcard inside it will expand. Test the actual placeholder separately from a translated output label.

7. Make a button do one useful thing

1. Add a Button to the register panel. Set ID to `point_button` and Text to `Add a point ($$score)`.
2. Open **Events** and find **Click**.
3. Choose **Action List** as the primary behavior.
4. Expand **Additional Event Actions** and click **Add Next Action**.
5. Choose **Change Variable**, select score, set Operation to add and Value to 1.
6. Save, run the game, open the register and click once.

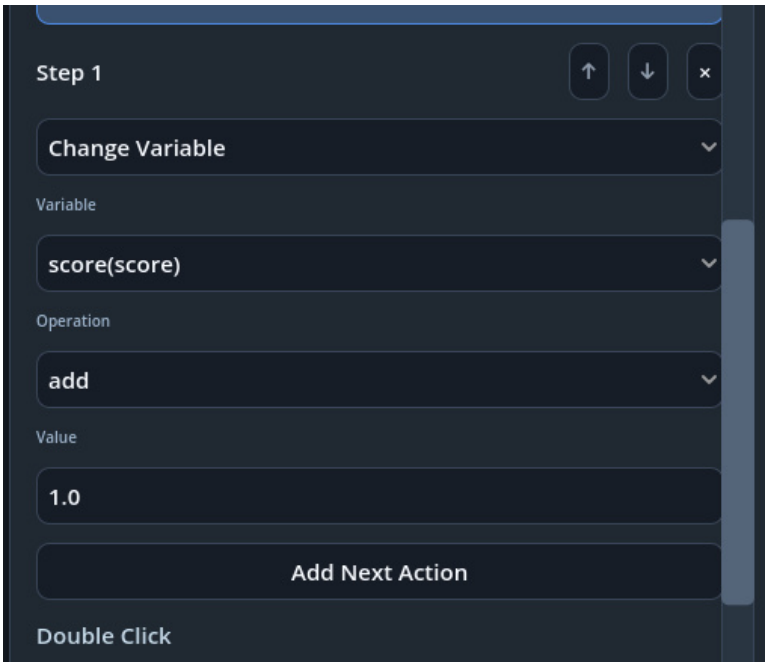

Figure 10. The score-changing step is in the button's event list. The arrows reorder steps and the x removes a step.

The live score should rise once. Both the label and the button caption then display the new value. You do not need a second action to rewrite those two captions when they already contain $$score.

> An Action List on a GUI event uses the same shared actions as other action-list editors. Choose a single primary action for one task, or a list when the command needs several ordered steps.

Choose the right mouse event

Event	What to use it for
Hover	A small pointer-enter response. It can repeat when the pointer leaves and re-enters.
Click	The ordinary left-click command.
Double Click	A separate double-left-click response.
Right Click	An alternate/context command.
Double Right Click	A separate double-right-click response.

These are mouse-event choices, not five different image slots. Hover and held-click images are appearance settings. An event action might instead open a GUI, change a variable or call a script.

The generic Click event is dispatched on the press. Releasing clears a configured pressed picture; it does not undo a variable change that the press already made.

Avoid accidentally running two commands

A double-click normally includes an earlier single click. Do not award unrelated points on both events and assume only one will run. Keep the primary action and its additional steps deliberate, and check for a leftover Legacy Click Action when converting an old button.

Labels and text boxes can have mouse events too

The same event areas are available on these elements. A clickable label can show help. A text box's Click event still means clicking the box; it does not mean its text changed or the player submitted an answer. Use the dedicated Enter options or a separate submit button for that.

> For keyboard, controller or touch support, test the actual input route in gameplay. A focused button or a working hover image does not by itself prove that every authored mouse event is mapped to every other device.

8. Give the button normal and held artwork

Select the button and open **Images**. Import the artwork first, then select its project asset in the matching field.

Image field	Use
Background Image	The explicit normal backdrop.
Media Asset	Can supply a normal background on an ordinary button when Background Image is not set.
Hover Image	The artwork shown when the pointer enters.
Clicked Image	The artwork shown while a left press is held. It should clear on release.
Background Fit	Contain, stretch, cover or tile, according to the kind of artwork.

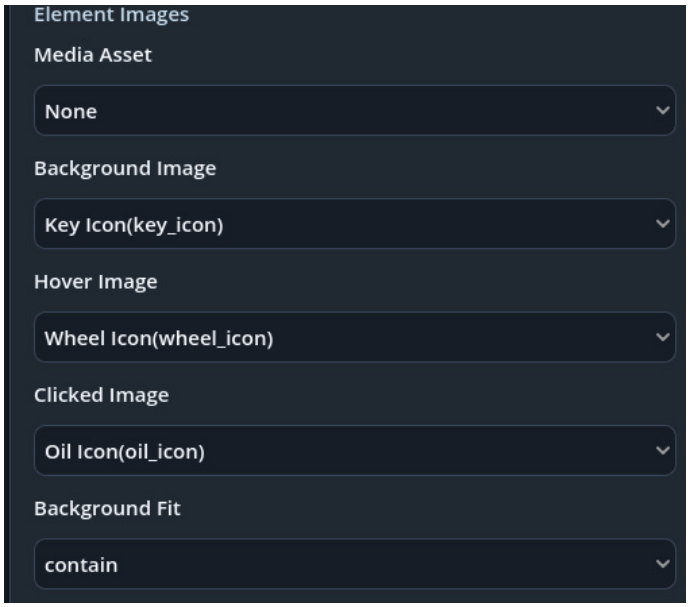

Figure 11. The image choices for the test button. Different icons make a temporary state easy to spot; finished artwork would usually use matching variants.

Keep the caption readable over all three pictures. The picture changes the appearance, not the rectangular clickable area or the command stored on the event. Surface hover/focus coloring can remain even after the held picture has cleared.

> Do not change Background Image permanently in the Click event merely to make a pressed appearance. Use Clicked Image for a temporary held state. Use a variable and an explicit property change when you actually want a lasting “selected” appearance.

Test hover, hold and release separately

1. Turn the designer's **Preview** switch on.
2. Move onto the test button, hold the left button, then let go.
3. Check the hover artwork, held artwork and return to the normal artwork.
4. Move off and back on when you want to test hover again.
5. Turn Preview off before moving or resizing controls.
6. Save and repeat in gameplay to test both the state and its actual action.



Figure 12. Normal artwork before the pointer enters.



Figure 13. Hover artwork after entering.

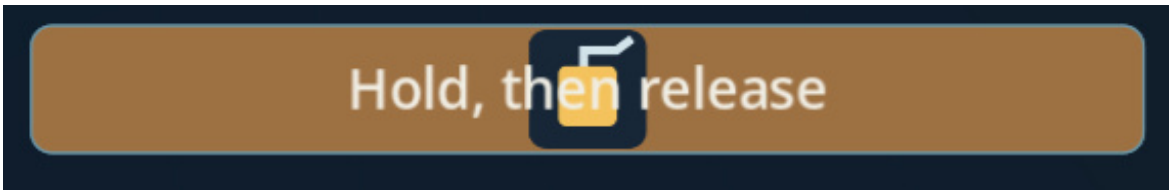

Figure 14. Clicked Image while the press is held.

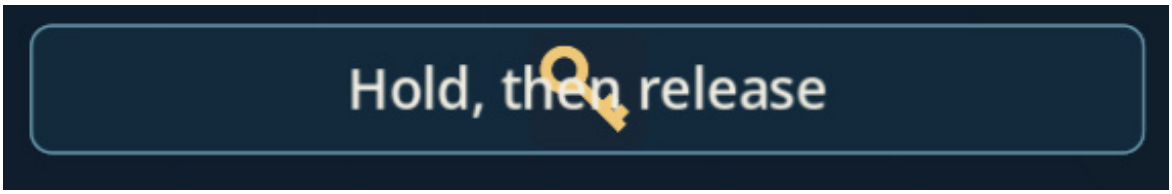

Figure 15. The normal picture returns after release. Focus or hover color can still be visible.

Designer Preview deliberately does not run gameplay events. The Add a point button should be tested in Play / Restart or Test Scene when you want to prove the score action.

9. Open a panel or choose a verb

A button that opens the register

1. On a visible HUD, add a Register button.
2. In its Click event, choose the catalog **Show GUI** action.
3. Select Workshop Register(register_panel) in the GUI parameter.
4. Give the panel a close/return Button using **Hide GUI**, targeting the same GUI.
5. Keep the panel trigger-controlled, save and test opening and closing it.

A modal GUI blocks interaction with the room while it is visible. It does not automatically stop every animation or timer. If modal is not exposed on a new record's General form, it is the whole-GUI JSON property "modal": true; use that advanced option only when you need this behavior.

A button that chooses Look or Use

1. Choose the built-in **set-verb** behavior for Click.
2. In **Click Verb**, select the existing Look or Use verb.
3. Add a HUD Label containing `{command}`.
4. Run the game, choose the verb and hover over an interactable room object.

The command label combines the verb display name with the hovered target. Choosing Use prepares that interaction; the object still needs its own verb actions. For the uncluttered panel screenshots, the open sequence also hides workshop_hud; the close sequence shows it again with the matching Show GUI action.

> A GUI ID in an action is the stored ID, not the full Name(ID) display label. Choose the reference selector when offered. For built-in show-gui/hide-gui behaviors with a plain Target field, enter register_panel exactly.

10. Add an ordinary text box

1. Choose **Add Element > Text Box**. Set its ID to `name_box`.
2. In General, use X 28, Y 204, Width 590 and Height 52 for the register example.
3. In **Text & Content**, leave the starting Text empty. Choose a readable font, size and color.
4. In **Type Settings**, set Placeholder to Type a visitor name.
5. Keep Multiline, all three Enter options and Password / Cipher Mode off for this first form.
6. Save and test typing in the box during gameplay.

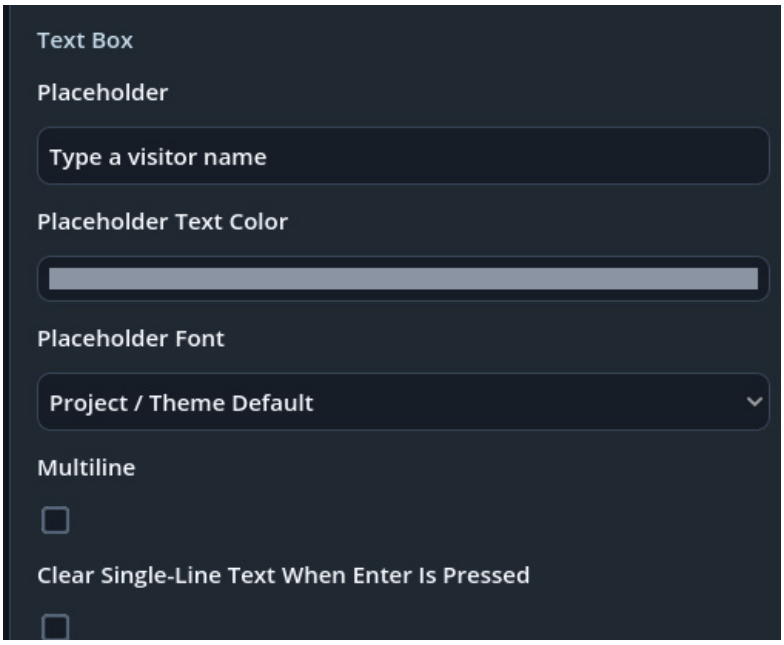

Figure 16. The first input uses a placeholder and ordinary single-line editing. Submission is handled by the nearby button.

Clicking the box gives it keyboard focus. The player can type, select and replace its contents. A separate submit button lets you decide when those contents should become game data.

> Text & Content edits the starting contents, stored as value on a text-box record. Once the game is running, the live editable control has its current text. These are not automatically kept in step with a global variable.

Use a placeholder as a short instruction

Setting	Purpose
Placeholder	Guidance displayed only while the box is empty.
Placeholder Text Color	The guidance color, separate from the player's entered text.
Placeholder Font	The font used for that empty-field guidance.
Text / starting value	Real initial contents that the player can edit; not a placeholder.

Type a visitor name is useful guidance. Joey as a starting value is actual editable input. If you put Joey in Text & Content, the placeholder will not appear until the player clears those contents.

Keep the instruction visible elsewhere when needed

A placeholder disappears as soon as the player types. For an important instruction such as Register password, put a normal Label above the text box. The player can then still see what the field means after entering the first letter.



Figure 17. A separate Visitor name label remains above the field after the name is entered. The output label displays the saved value.

The placeholder is not submitted as the player's answer. It is not a default value for a variable. It also does not run the label/button wildcard expansion, so keep it as plain guidance rather than a live score display.

Do not rely on a faint placeholder as the only sign that a text box exists. Give the field enough border contrast and keep its entered text readable against the normal and focused surfaces.

11. Decide what Enter should do

The single-line Enter options are independent. Enable only the behavior that suits the field. For the visitor-name form, the button submits the name, so these options can remain off.

Type Settings option	What happens on Enter
Clear Single-Line Text When Enter Is Pressed	Clears the current field after submission. This can be enabled without running the parser.
Run Text Parser Command When Enter Is Pressed	Sends the submitted sentence to the game's text parser. The parser must be enabled and configured.
Return Focus To Text Box After Enter	Keeps or restores typing focus so the player can enter another line.
Multiline	Switches to a notes-style editor. The single-line Enter/parser and password controls are hidden. Enter is then for a new line.

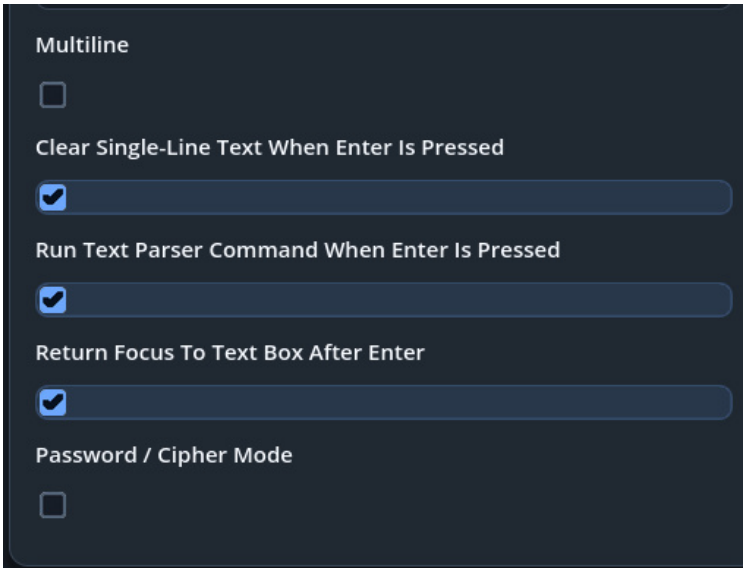

Figure 18. The command-entry example has all three Enter options enabled and Multiline off.

The parser receives the submitted sentence before a configured clear empties the box. A separate button script that reads the text box after it has been cleared will naturally see an empty field.

> Enter is not a general “run any script on this text” switch here. Use a submit button for the name/password recipes. Use the parser option when you want a real parser command.

Keep editable text literal

A text box is not a live output label. Its starting Text and Placeholder are not passed through the label/button wildcard expansion. This prevents the editor from continually replacing words the player is trying to edit.

Put this text here	Result
Label Text: @PlayerName	The active character's display name.
Button Text: Add a point ($$score)	A caption containing the live score.
Text Box starting Text: @PlayerName / $$score	Those literal characters remain editable.
Text Box Placeholder: Hello @PlayerName	Literal guidance while empty; not the expanded character name.

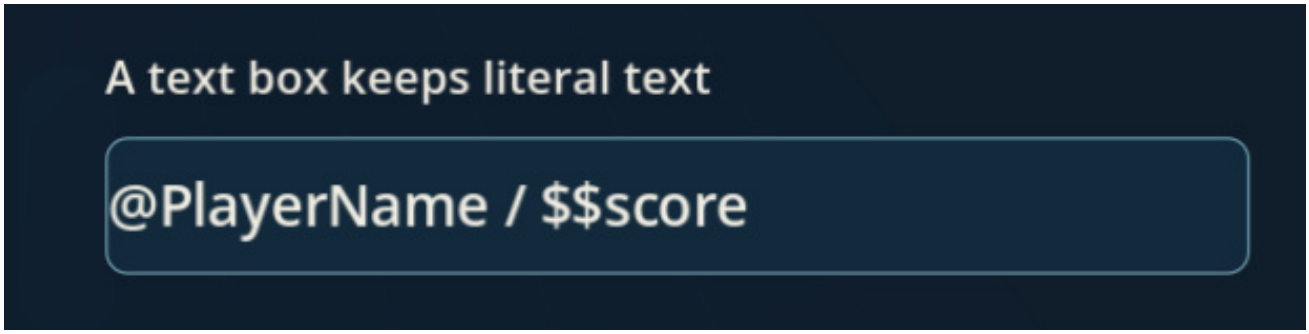

Figure 19. The literal-input example keeps @PlayerName and $$score in the text box. The neighbouring password/cipher controls have different display rules.

Fill a field deliberately when you want a value there

To prefill a text box with the saved visitor name, read visitor_name in a script and write that String to the text box's value property. Do this when the form opens or when the player chooses Restore, rather than refreshing it every frame.

Constantly rebuilding or refilling a focused input can move its caret, remove focus or replace an unfinished answer. Use a separate label for live output, and let the input remain under the player's control until submission.

The visitor-name and note recipes below show the explicit read/write connection. The same idea works for an answer form, a notebook title or a rename dialog.

12. Read and save the entered name

The Use this name button will read name_box, trim the outside spaces and store the result in visitor_name. The visitor label already contains $$visitor_name, so it will display the new value.

Step	Job
Read	Get the current text from register_panel / name_box.
Check	Reject an empty or all-space answer.
Store	Write the cleaned String into visitor_name.
Report	Put a short confirmation in feedback.

1. Create a native **GDScript** record with ID `save_name` and Name Save Name.
2. Replace its source with the complete script on the next page.
3. Validate/apply the script and save the project.
4. Set Use this name's Click behavior to the built-in **run-visual-script** and Click Target to `save_name`.
5. Run the game, enter a name and press Use this name.

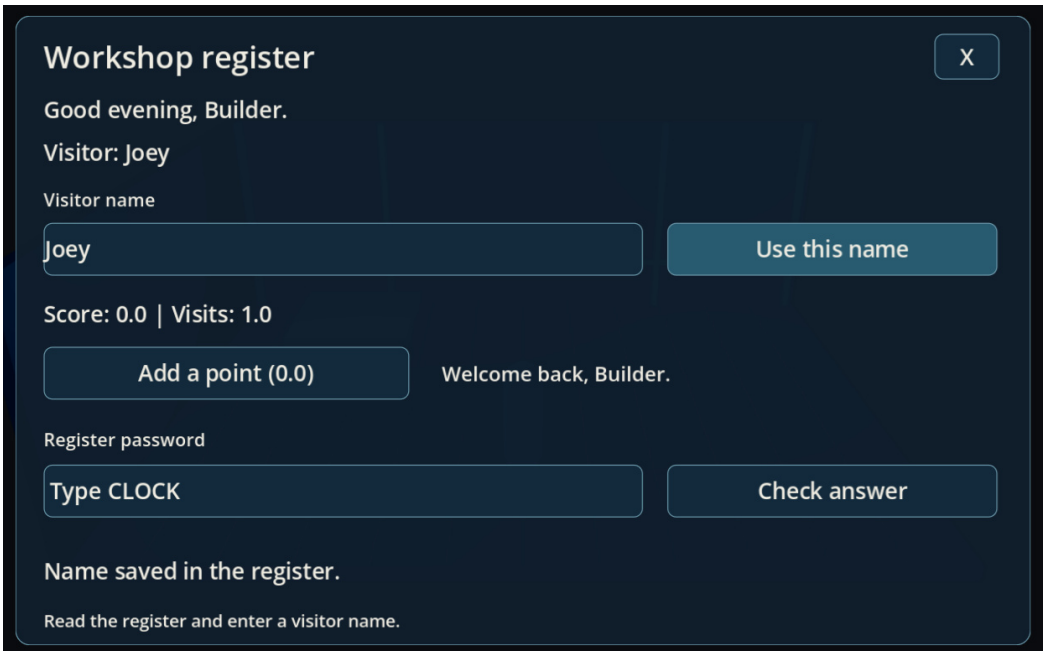

Figure 20. The saved visitor is Joey, while the active adventure character is still Builder. They are separate values.

The built-in behavior has an older name, run-visual-script, but this release can dispatch the named GDScript record too. The catalog Run Visual Script action also offers a Script selector for the shared script types.

The complete Save Name script

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

func run(_context: Dictionary = {}) -> Variant:
    var entered: String = str(GetGUIElementProperty(
        "register_panel", "name_box", "text", ""
    )).strip_edges()

    if entered.is_empty():
        await SetVariable({
            "name": "feedback",
            "value": "Please enter a visitor name."
        })
        return false

    await SetVariable({
        "name": "visitor_name", "value": entered
    })
    await SetVariable({
        "name": "feedback",
        "value": "Name saved in the register."
    })
    return true
```

Read the parts that you may want to change

Part	Meaning
register_panel / name_box	The GUI and the specific input control.
text	Read the current editable contents, not the original definition's value.
The final empty String	Fallback if the field cannot be found.
strip_edges()	Remove spaces at the beginning and end. It keeps the name's internal spaces and letter case.

This saves the name in a runtime variable. It does not rename the active Character definition. Use @PlayerName for the character and $$visitor_name for the separate entered name.

Connect and test the name button

Select the actual Use this name button, not its neighbouring label or text box. Open Events, find Click, choose run-visual-script and enter save_name in Click Target.

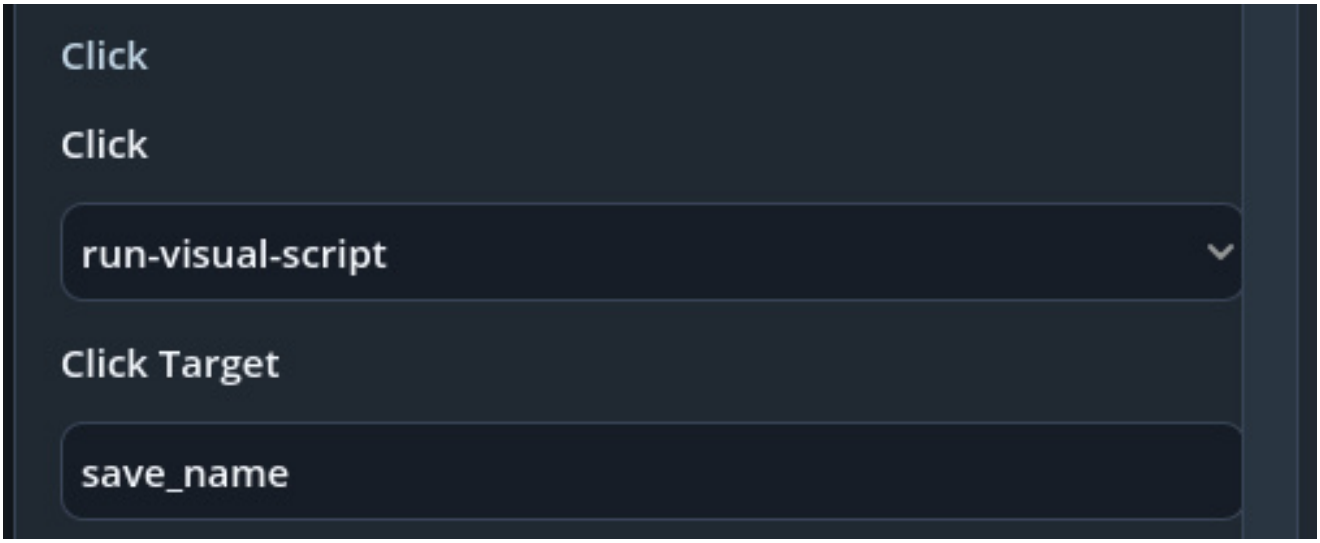

Figure 21. The button names the helper script in its Click event. It does not read the input merely because its caption says Use this name.

Try this	Expected result
An ordinary name	Visitor changes and the confirmation appears.
Outside spaces around a name	The stored name omits those outside spaces.
An empty field or only spaces	The script asks for a name and keeps the previous stored value.
A long name	The value can be stored; check that the output label has enough room.
Typing without pressing the button	The stored variable does not change yet.

Keep input collection and game-state changes separate. If you later use the same field to rename a save slot or title a journal note, change the receiving action deliberately rather than assuming the textbox ID decides the destination.

> If the script runs but reads an empty String, check the GUI ID, element ID, that the field is still present, and whether Clear on Enter emptied it before the button read it.

13. Turn an answer into a small puzzle

Add answer_box and a Check answer button. Use the same ordinary single-line setup as name_box, with Placeholder set to Type CLOCK. The button will run a second GDScript record called check_answer.

1. Define register_open as Boolean false and score as Number 0.
2. Give the feedback label the text `$$feedback`.
3. Create the check_answer script from the next page and validate/apply it.
4. Set Check answer's Click behavior to run-visual-script, Target `check_answer`.
5. Try a wrong word, then CLOCK. Check the feedback, score and disabled button.

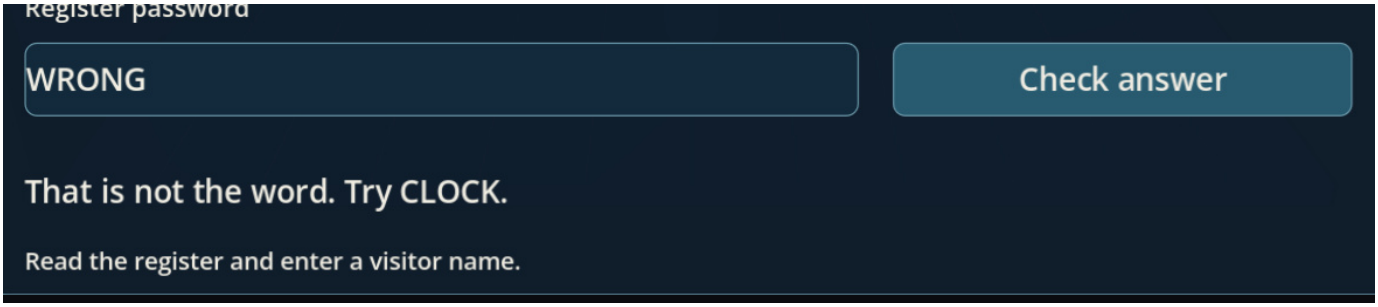

Figure 22. A wrong answer produces feedback without awarding the reward. The entered word remains in the box for editing.

Choice in this recipe	Reason
Trim outside spaces	A stray space at the end should not spoil the answer.
Convert to uppercase	clock, Clock and CLOCK can mean the same answer.
Check a Boolean first	Prevents the reward being given repeatedly.
Disable the solved button	Makes the completed state clear to the player.

The comparison rule belongs to the script. Change it if your puzzle needs exact capitalization, several acceptable answers or a different response to a blank field.

The complete Check Answer script

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

func run(_context: Dictionary = {}) -> Variant:
    var entered: String = str(GetGUIElementProperty(
        "register_panel", "answer_box", "text", ""
    )).strip_edges().to_upper()

    if entered != "CLOCK":
        await SetVariable({
            "name": "feedback",
            "value": "That is not the word. Try CLOCK."
        })
        return false

    if bool(Runtime.variables.get("register_open", false)):
        return true
    await SetVariable({"name": "register_open", "value": true})
    await ChangeVariable({
        "name": "score", "operation": "add", "value": 5
    })
    await SetVariable({
        "name": "feedback",
        "value": "The register unlocks. Five points earned."
    })
    await SetVariable({
        "name": "objective",
        "value": "Read the newly opened register."
    })
    await DisableGUIElement({
        "guiId": "register_panel", "elementId": "answer_submit"
    })
    return true
```

Use answer_submit as the Check answer button's ID. The script disables that specific button after a correct answer. Replace these explicit IDs if your controls have different ones.

The flag is the reward guard. The disabled button is a useful presentation choice, but the flag still protects the reward if the script is called from somewhere else.

Check the solved state and the reward

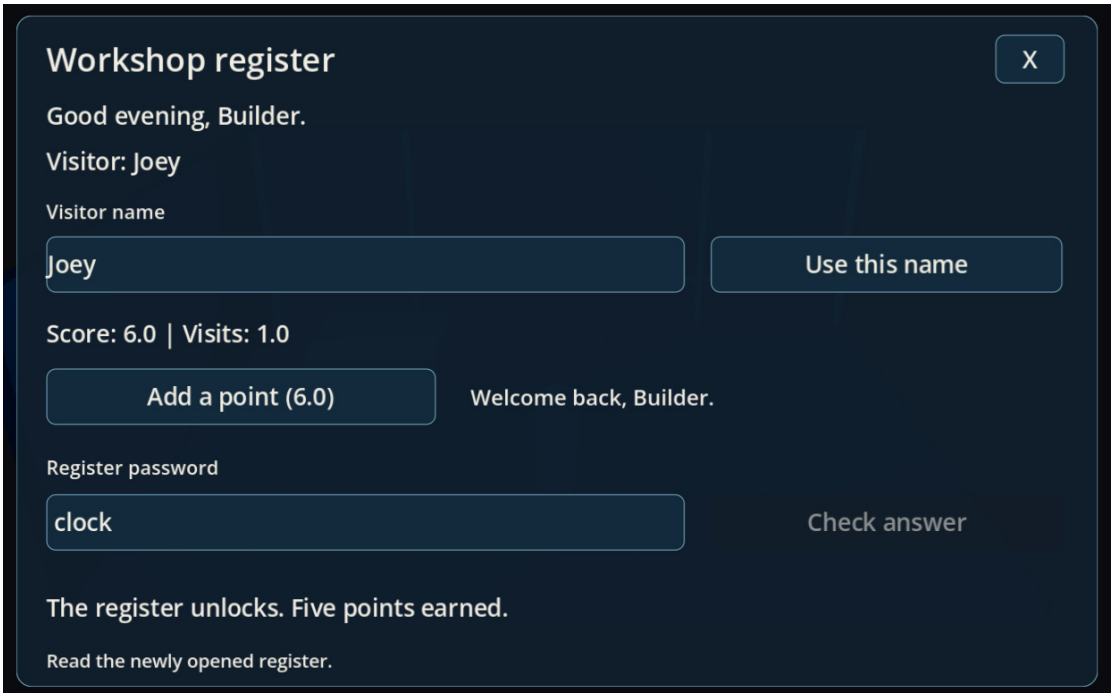

Figure 23. The correct answer updates the feedback and objective. One earlier test point plus the five-point reward gives a score of 6.0.

Test	Expected result
WRONG	No reward; feedback asks for the correct word.
clock with outside spaces	Accepted after trimming and uppercasing.
Correct answer once	register_open becomes true and five points are added.
Call check_answer again	The existing true flag prevents another reward.
Start a fresh game	Starting variables return to the authored defaults.

For an actual room puzzle, replace or extend the successful actions with the change you need: show a clue GUI, change a room object, play a sound or start a conversation. Keep the flag check before the reward.

When loading a saved game, restore presentation from the saved flag as part of opening the panel. A Boolean saved in game state is more dependable than treating the last control's enabled/disabled appearance as the only record of a solved puzzle.

Use the same condition in a visual action list when its data is already in variables. Reading free text into that state is the small script bridge demonstrated here.

14. Use a multiline box for notes

1. Add a Text Box on a separate GUI, such as Text-box Examples(text_examples).
2. Set its ID to `notes_box`. Give it enough height for several lines; the example uses 450 by 214.
3. Open Type Settings and enable **Multiline**.
4. Enter Placeholder guidance such as Write a clue or a short note...
5. Add Save note and Restore note buttons beside or below the box.
6. Create a String variable called `notes_saved` for the stored note.

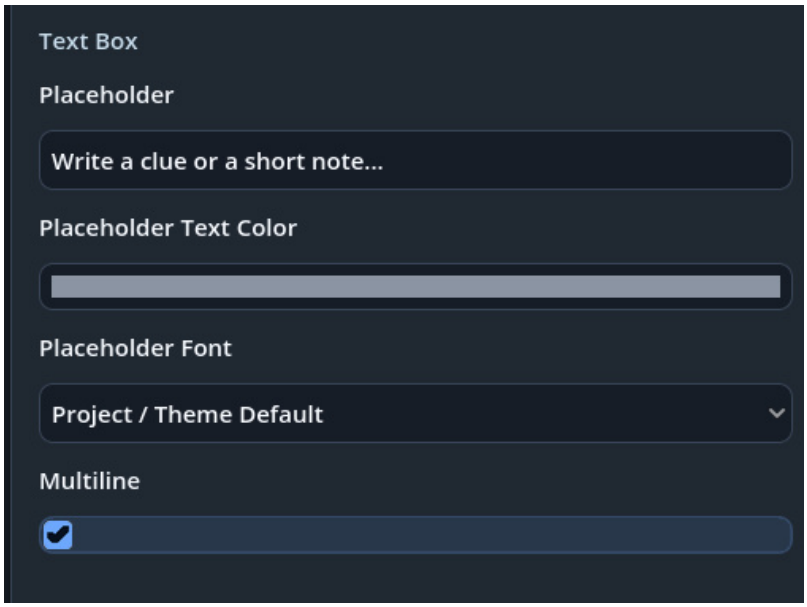

Figure 24. Multiline changes the kind of input. The single-line Enter and password options are not shown in this mode.

Enter now creates a new line. The box wraps text inside its available width. It is suitable for notes, a longer answer or a draft message, rather than a single parser command.

Hiding a GUI and saving its input are different operations. The current control may keep its text during the same run, but a fresh game, a rebuilt control or a different UI flow can replace it. Copy important input into a variable deliberately.

> Use Save note to accept a draft, and Restore note to refill the field from the stored value. Do not automatically overwrite a focused draft every time another label refreshes.

Save and restore the note explicitly

Create a GDScript record called save_note. Connect the Save note button to it with the same run-visual-script event setup as Save Name.

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

func run(_context: Dictionary = {}) -> Variant:
    var note: String = str(GetGUIElementProperty(
        "text_examples", "notes_box", "text", ""
    ))
    await SetVariable({"name": "notes_saved", "value": note})
    return true
```

Create a second GDScript record called restore_note and connect Restore note to it.

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

func run(_context: Dictionary = {}) -> Variant:
    var note: String = str(Runtime.variables.get("notes_saved", ""))
    SetGUIElementProperty(
        "text_examples", "notes_box", "value", note
    )
    return true
```

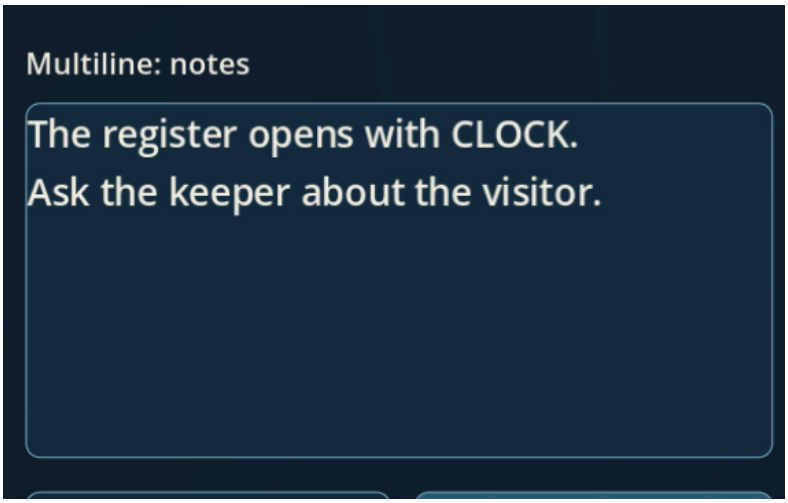

Figure 25. The restored multiline text contains both saved lines. The script reads live text and writes the text-box value when refilling it.

A label containing Saved: $$notes_saved can show the stored note separately. Saving this variable is not the same as writing a text file. Game saves can preserve runtime variables; the draft control still needs an intentional restore step when your UI is rebuilt or reopened after loading.

15. Mask a single-line answer

Password / Cipher Mode changes how a single-line text box displays its contents. The underlying text is still the answer the player typed, so the same input-reading script can check it.

1. Select a single-line Text Box and open **Type Settings**.
2. Enable **Password / Cipher Mode**.
3. Set **Password Mask Character** to *, or another single character supported by the font.
4. Leave Password Mask Image as None and Use Per-Character Cipher off for a simple mask.
5. Run the game and type CLOCK. The field displays five mask characters.

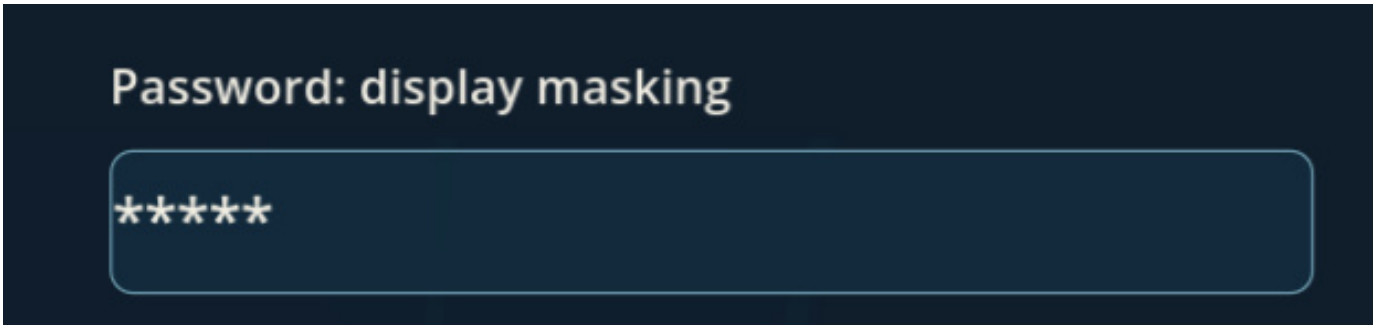

Figure 26. CLOCK is displayed as five asterisks. The masking changes the view, not the String that the answer script reads.

Setting	Meaning
Password Mask Character	The ordinary one-character mask. If you enter a longer String, the simple mask uses its first character.
Password Mask Image	An imported image repeated for the entered characters.
Mask Image Size	The image size; shown when a mask image is selected. The field height also limits its visible size.
Mask Character Spacing	The space between custom mask/cipher marks.

This is a display effect for game input. It does not encrypt the answer or make a local game's stored data secret. Use it for a fictional terminal, combination or puzzle presentation.

Replace characters with a small cipher

For a substitution display, enable **Use Per-Character Cipher** while Password / Cipher Mode is on. Enter a valid JSON object in Cipher Map JSON.

```
{"A":"1", "B":"2", "C":"3"}
```

With that map, entering ABC displays 1 2 3, with the chosen custom spacing. The saved/read input remains ABC. A source letter not listed in the map uses the normal mask fallback.

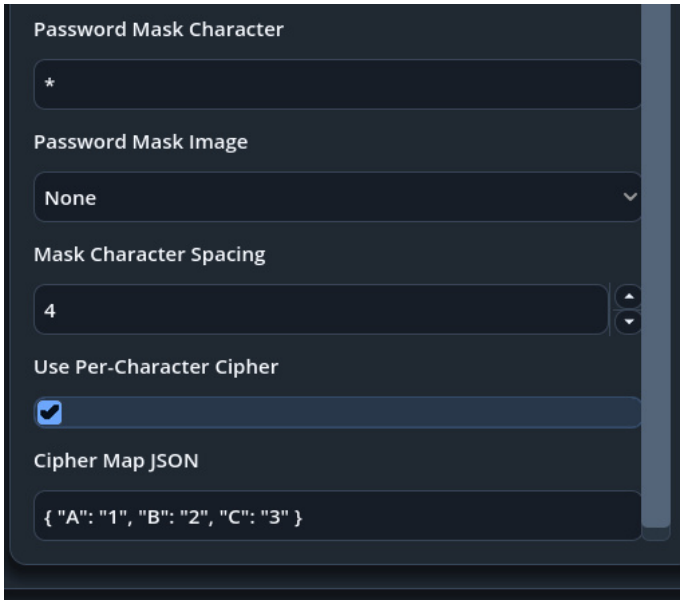

Figure 27. The cipher has its own switch and map. Its keys are source characters, with case-sensitive matching.

Map entry	Display replacement
"A":"1"	Ordinary replacement text.
"B":{"text":"!"}	Text supplied by an object entry.
"C":"{ID:key_icon}"	The imported image asset key_icon.
"D":{"assetId":"wheel_icon"}	An image supplied by an object entry.

Use the exact imported image ID. The {ID:...} form belongs to the cipher map; it is not a general wildcard that inserts an image into every ordinary Label.

A lowercase a and uppercase A are different source keys. Add both if you want both to use the same mark. Use wide enough fields for custom symbols/images, and keep an ordinary readable test before adding elaborate artwork.

Check the cipher map and its fallback

Input or setup	Expected result
ABC with A/B/C mapped to 1/2/3	The display uses the three replacements.
An unmapped X	Uses the default mask character, or the configured general mask image.
a when only A is mapped	Falls back; the map does not silently uppercase input.
Read the field's text in a script	The original input, such as ABC, not the drawn 1 2 3.
Multiline enabled	Single-line password/cipher settings are hidden.
Invalid JSON	Not a reliable cipher map. Correct the JSON syntax and test again.

```
{
    "A": "1",
    "B": {"text": "!"},
    "C": "{ID:key_icon}"
}
```

If the map field is a single-line editor, enter the same object on one line. Keep double quotes around JSON keys and text values, commas between entries and no trailing comma after the last entry.

Keep the puzzle answer rule separate. The check_answer script can compare CLOCK whether the field shows CLOCK, * or custom pictures. The display cipher does not automatically decode a different answer into the expected word.

> Change the game logic only when the puzzle requires it. A decorative cipher is not a substitute for accepting the correct input, normalising case deliberately and protecting the one-time reward.

16. Make a parser command field

A parser field submits a command to the adventure's vocabulary and actions. It is different from a password field that checks one word in its own script.

1. Enable and configure **Text Parser Designer** for the project.
2. Add a single-line Text Box to a visible HUD. The example ID is `hud_parser` on workshop_hud.
3. Set a short Placeholder, such as Type a command, then press Enter.
4. Enable Run Text Parser Command When Enter Is Pressed.
5. Enable Clear Single-Line Text When Enter Is Pressed and Return Focus To Text Box After Enter if you want repeated command entry.
6. Leave Password / Cipher Mode off so the command remains readable.

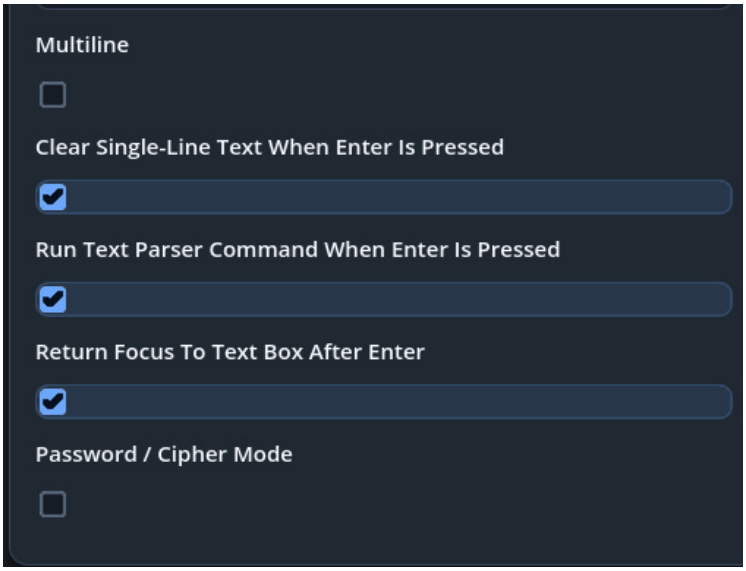

Figure 28. These three Enter settings submit the sentence, clear the field and return typing focus.

Test an ordinary command that your room already supports, such as look at the clock. For the pictured status test, Text Parser Designer has a custom command status pointing to a script that displays the current visitor and score.

> The parser uses the commands, verbs and target names you author. Adding a text box does not create that vocabulary. A GUI {command} label is the current verb/hover display, not a transcript of the last submitted parser sentence.

Offer a Submit button as another route

A visible Submit button is useful when you also want a tap/click route. Create a GDScript record called submit_command, then call it from that button's Click event.

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

func run(_context: Dictionary = {}) -> Variant:
    var command: String = str(GetGUIElementProperty(
        "workshop_hud", "hud_parser", "text", ""
    )).strip_edges()
    if command.is_empty():
        return false
    SetGUIElementProperty(
        "workshop_hud", "hud_parser", "value", ""
    )
    await Runtime.run_text_parser_command(command)
    return true
```

This native GDScript helper reads the field before clearing it and passes that saved sentence to the runtime parser. It does not need a new parser rule for the Submit button itself.

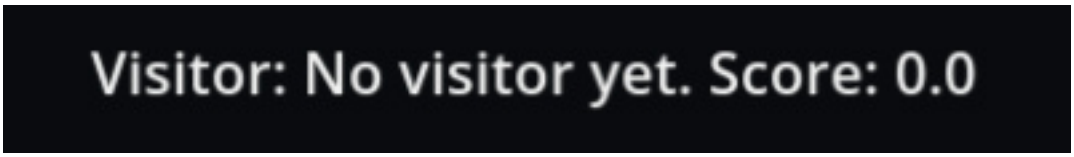

Figure 29. The status response in the fresh preview uses the starting visitor and score.

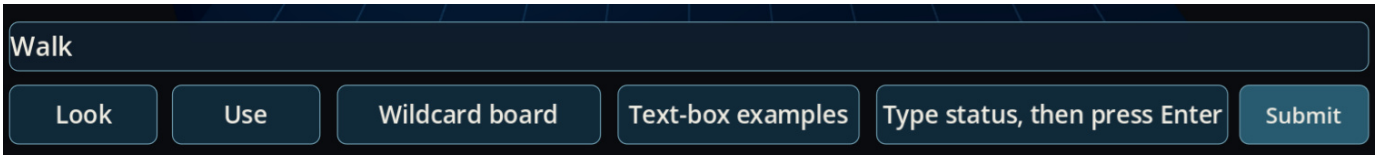

Figure 30. The HUD has both the Enter field and a separate Submit button. The field is empty again after submission.

The button helper does not add an extra focus-restoration routine after it rebuilds the field. The player can click the field again. Use the built-in Enter route when its return-focus behavior is the one you want for repeated typing.

17. Change a control during gameplay

Use the shared GUI actions when an element needs to change during play. Give an explicit GUI ID and element ID for these examples.

Task	Action or property
Show/hide the whole panel	Show GUI / Hide GUI
Show/hide a particular control	Show GUI Element / Hide GUI Element
Keep a button visible but unavailable	Disable GUI Element; Enable GUI Element restores it.
Change a label or button caption	Change GUI Element Property: text
Fill or clear a text box	Change GUI Element Property: value
Change temporary hover artwork	Change GUI Element Property: hoverImageAssetId
Change one corner	Change GUI Element Property: radiusTopLeft, or the corresponding corner key.

```
await ChangeGUIElementProperty({
    "guiId": "register_panel",
    "elementId": "register_feedback",
    "property": "text",
    "value": "The keeper has closed the register."
})
```

If the feedback label originally contained $$feedback, replacing its text with this fixed sentence removes that token from the running element. Change the feedback variable instead when you want to preserve the live binding.

Runtime changes do not rewrite the saved authoring layout. Rebuilding a text box to change one of its definition properties can also replace its current draft. Store important entered text before an operation that recreates that control.

Use a visual node, action list or small helper



Figure 31. The same Change Variable action can be a visual node. Its parameters still identify the score variable, operation and amount.

1. Create a visual script with an **On Interact** entry.
2. Add **Change Variable**, choose score, Operation add and Value 1.
3. Connect the entry's NEXT output to the action's FLOW input.
4. Call that visual script from the button, then test the live score.

For a longer button command, put the same shared actions in Additional Event Actions and reorder the steps. For reading free text, call the tested Save Name or Check Answer GDScript helper, then continue with actions that use its variables.

Direct GDScript helper	Use
GetGUIElementProperty(gui, element, "text", "")	Read the live text of these text boxes.
SetGUIElementProperty(gui, element, "value", text)	Refill a text box from a String.
Runtime.variables.get("visitor_name", "")	Read the accepted game value directly.

Other elements can return stored definition properties rather than the final painted child-label text. Use the variable itself for game decisions. This guide uses explicit IDs; script caller references such as this are a separate feature, not another text wildcard.

18. Use labels for dialogue and commands

An ordinary label displays its authored text and tokens. A dialogue label can instead be filled by the game's dialogue/speech system when you assign the expected Group ID and system GUI role.

Use	Connection
A live command line	A normal Label with Text {command}.
A fixed help sentence	A normal Label with its own Text and optional Translate entry.
A spoken/narrator line	A suitable Label or Button with Group ID @talkdialogue on the assigned speech/dialogue GUI.
Dialogue choices	Clickable controls using the expected @choice1 through @choice20 Group IDs.

Assign the authored GUI in Global Settings > General > Custom System Interfaces to the appropriate Dialogue, Speech Bubble / Character Say, General Narrator Speech or Dialogue Choices role. Then test an actual conversation.

Do not confuse three different uses of symbols

Where it appears	Example	Meaning
Text field	@PlayerName	A text token expanded for display.
Group ID field	@talkdialogue	A special connection used by the dialogue system.
Cipher Map JSON	{ID:key_icon}	An image replacement inside that cipher display.

Putting @talkdialogue into the ordinary Text field does not make a dialogue connection. Likewise, a caption containing @PlayerName does not change the speaker or attach a portrait. Portraits and lip sync use their character/dialogue setup.

> A normal live label does not automatically become a typewriter message. Use the message/dialogue typewriter options for that workflow rather than assuming a wildcard controls reveal speed.

19. Diagnose text and input problems

Symptom	Check first
$$visitor_name is printed literally	The exact variable ID, its existence in the running game and whether this is an output label/button.
@playername is unchanged	Use the exact built-in spelling @PlayerName.
A text box keeps @PlayerName	That is expected: its editable contents are literal. Fill value explicitly if required.
{target} is empty	The pointer may not be over an interactable target.
{value}/{max} shows 0/100	Those are element-value defaults, not the contents of another text box.
The score has .0	Numeric arithmetic produces a number. Format a display String if needed.
A corner ignores Border Radius	It may have an explicit per-corner override. -1 restores inheritance.
A long sentence clips	Check frame Width/Height, font size, line breaks and the translated wording.
A symbol becomes a square	Check that the actual font contains that glyph.
Formatting tags appear as text	These are plain text controls. Do not assume RichTextLabel/BBCode parsing.

Separate the source from the result. Inspect the authored Text in the designer, the variable value in the running game and the resulting label on screen. A display token does not compute a puzzle outcome or save a typed answer by itself.

> GUI Preview can show raw tokens or local editing states. Use a running-game test for values, current player/scene information, action results and parser commands.

Trace a button or submission problem

Symptom	Focused check
The button does nothing in Designer Preview	Gameplay events are intentionally suppressed there. Test in Play / Restart or Test Scene.
It does nothing in the game	Inspect the right element's Click event, target ID, enabled state and any overlapping GUI.
A click awards twice	Check primary action, additional list, legacy action and double-click behavior.
The script reads the initial value	Read text for the current input. value belongs to the text-box definition used for filling it.
Restoring text has no effect	For these text-box recipes, write value rather than changing a second, unused text field in the definition.
Enter clears but does not run a script	Clear on Enter is independent. Use Run Parser on Enter for a parser sentence, or a submit button for the helper script.
Parser commands fail	Enable/configure the parser and check the room's actual verbs, targets or custom command script.
The note is lost in a fresh run	Copy accepted input into a game variable and restore the field deliberately when needed.
A password comparison fails	Check case, outside spaces and the original input, not the drawn mask symbols.
A held image stays selected	Use Clicked Image for the temporary state and check for a permanent event-driven image change.

Reduce a failing setup to one field, one button and one visible result. Prove that connection before adding animation, several GUIs or a longer action list.

The examples were checked in the running editor: name entry, a real score-button click, wrong/correct answers, the one-time reward guard, multiline note restore, literal text, display masks, image states and parser submission.

20. Finish with a complete player test

1. Start a fresh game. Check the default score and the active-character greeting.
2. Open the register, enter a name and press Use this name. Check the stored visitor output.
3. Click Add a point once and verify both score displays.
4. Try a wrong answer, then clock with outside spaces. Check feedback, reward and the completed state.
5. Hide and reopen the panel. Check its appearance and the sequential greeting.
6. Enter a two-line note, save it, change the draft and restore the accepted note.
7. Try the password and cipher controls. Read the original input through the helper when testing their logic.
8. Test status or another configured parser command through Enter and, if added, Submit.
9. Test normal, hover, held and released button artwork in preview and gameplay.
10. Try a longer name, a longer translation and the target devices' actual keyboard/touch/controller routes.

Keep the connections easy to follow

A Label shows the result. A Text Box collects the draft. A Button or Enter route accepts an instruction. Variables hold the game state. Once those jobs are clear, you can build a name form, notebook, terminal or puzzle without making every control do everything.

Keep this for reference	Meaning
$$variable	A game variable displayed as text.
@PlayerName / @PlayerId / @SceneName / @Score	The four built-in @ text tokens.
{command}, {verb}, {target} and other GUI braces	Extra GUI label/button display tokens.
@||@ / @//@	Sequential / random alternatives.
Read text / write value	The text-box read/refill distinction used in the scripts.

Save the accepted authoring changes, stop the preview and check the starting records once more. Keep the finished interface simple enough that its next change is easy to understand.