PCA 3D ADVENTURE STUDIO

The Dialogue Designer

Conversations that carry the story forward.

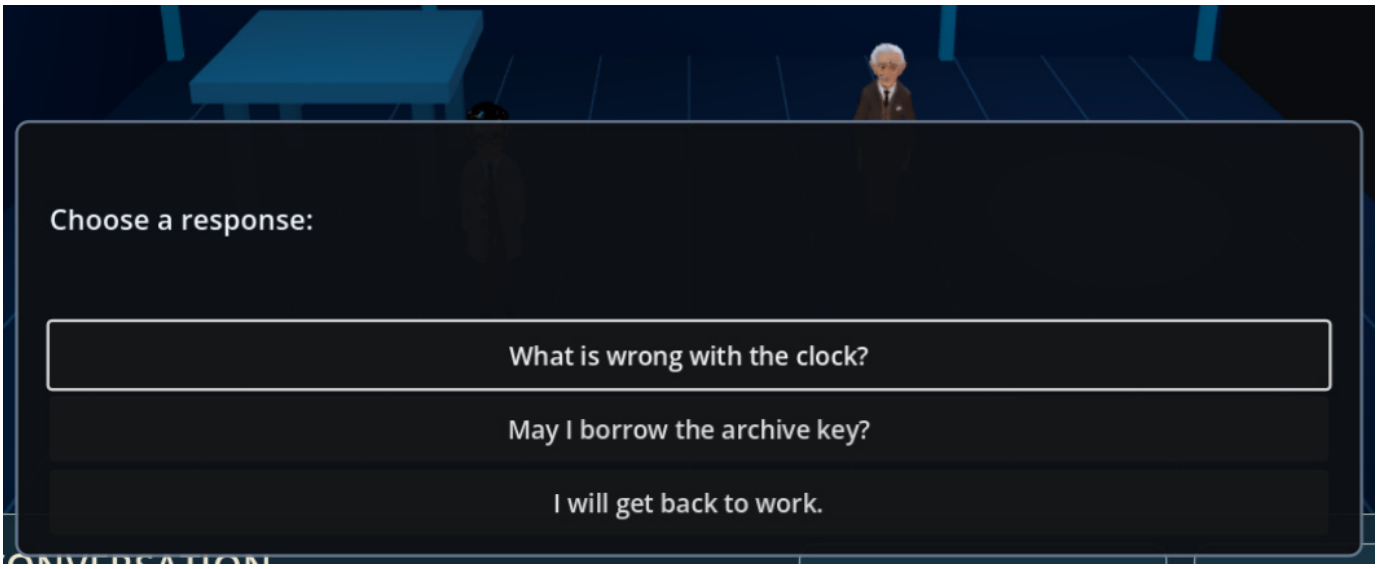

Figure 1. Elias’s conversation running in the workshop. The player’s questions lead to answers, a borrowed key and a new clue.

An in-depth, illustrated guide to lines, choices, actions, voices, portraits, lip sync and custom dialogue windows.

For PCA 3D Adventure Studio 0.86.3.4

How to use this guide

Start with a short, working conversation. Add its game rules next, and give it its finished appearance last. That order lets you fix one problem at a time.

Lessons 1-18 build the graph: an opening line, a topic menu, answers and a clean ending. Lessons 19-38 add rewards, conditions, read history, timers and the ways a game starts or ends a conversation.

Lessons 39-50 cover presentation: recorded voices, typewriter text, portraits, lip sync, custom windows and translations. Lessons 51-59 cover runtime edits, events, staging and a thorough test pass.

The appendices contain the complete example dialogue record, short script recipes and field checklists. You can read the whole guide in order or use the contents and PDF bookmarks to return to one job.

Names and IDs are shown separately where they matter. Use your own record IDs if you build the example with existing characters. In the screenshots, Professor Ivo is the active player and Elias is the clockmaker.

This is a reference PDF. It does not include a practice project or require you to download one. You can follow the steps in a project of your own. Save a copy before changing a conversation you want to keep.

Screenshot note

The images are actual editor and Play captures. Close-ups intentionally show just the controls discussed on that page. Layout and panel placement can vary with your theme, UI size and screen.

Code note

GDScript recipes use PCA’s PCAScriptBase and action API. Preserve indentation and ordinary quotation marks when copying from the PDF. The same named actions are available through the visual action controls; you do not need a script for the basic conversation.

Contents

1. What the Dialogue Designer does . . . 5
2. The example we will build . . . 6
3. Open a dialogue and find its controls . . . 7
4. The four node types . . . 8
5. Move around the graph without losing your place . . . 9
6. Write the first spoken line . . . 10
7. Choose the right speaker . . . 11
8. Set the greeting as the start . . . 12
9. Connect and replace flow wires . . . 13
10. Test the tiny conversation first . . . 14
11. Add a topic menu . . . 15
12. Add and remove response options . . . 16
13. Give every response a stable Choice ID . . . 17
14. Return to the menu without a second wire . . . 18
15. Chain several nodes with explicit flow . . . 19
16. Reuse an answer from two menus . . . 20
17. Duplicate, delete and tidy safely . . . 21
18. Build the complete archive conversation . . . 22
19. Put game logic in Action nodes . . . 23
20. Give the key and record the loan . . . 24
21. Conditions hide options; Enabled disables them . . . 25
22. Add a simple variable condition . . . 26
23. Reveal the archive question after the reward . . . 27
24. Test conditions with real starting values . . . 28
25. Change availability from an action or script . . . 29
26. Protect important rewards with a guard . . . 30
27. Mark previously chosen responses . . . 31
28. Keep Goodbye and useful help repeatable . . . 32
29. Save history without confusing it with position . . . 33
30. Reset history at the scope you intend . . . 34
31. Put a time limit on a decision . . . 35
32. Understand preselected response numbers . . . 36
33. Put consequences after the timed choice . . . 37
34. Decide whether the player repeats a choice . . . 38
35. Start the conversation from the NPC . . . 39
36. Call dialogue from scripts and visual nodes . . . 40
37. Call another dialogue and return . . . 41

38. End normally or interrupt deliberately . . . 42
39. Add a recorded voice to a Line . . . 43
40. Use Voice Key for language-specific recordings . . . 44
41. Reveal text with the typewriter effect . . . 45
42. Assign a default portrait and a Line override . . . 46
43. Choose the right portrait source . . . 47
44. Make scene characters and portraits lip-sync . . . 48
45. Prepare and correct the voice cue track . . . 49
46. Replace the spoken-line window . . . 50
47. Build a response menu with paging . . . 51
48. Test custom windows and personal overrides . . . 52
49. Translate dialogue and keep voices aligned . . . 53
50. Set the style without changing the dialogue . . . 54
51. Add a new topic while the game is running . . . 55
52. Remove or update runtime topics deliberately . . . 56
53. Use dialogue events and quest completion . . . 57
54. Add camera and character direction . . . 58
55. Write choices that make sense to the player . . . 59
56. Run a complete conversation test . . . 60
57. Fix a conversation that closes or loops . . . 61
58. Fix text, voices, portraits and custom GUIs . . . 62
59. Make the finished conversation ready to ship . . . 63
Appendix A. The complete archive dialogue . . . 64
Appendix B. Script recipes . . . 68
Appendix C. Line and Choice field reference . . . 73
Appendix D. A quick diagnosis by reference . . . 74
Appendix E. Put names and values in a sentence . . . 75
Version, illustrations and verification . . . 77

1. What the Dialogue Designer does

The Dialogue Designer decides who speaks, what the player can ask and what happens after each reply. It can also give an item, change a story variable, run a script or start another conversation.

This guide builds a conversation with Elias, the clockmaker. The player can ask about the broken clock, borrow the archive key and unlock a question about the archive. We will build the simple version first, then add conditions, recorded voices, portraits and a custom interface.

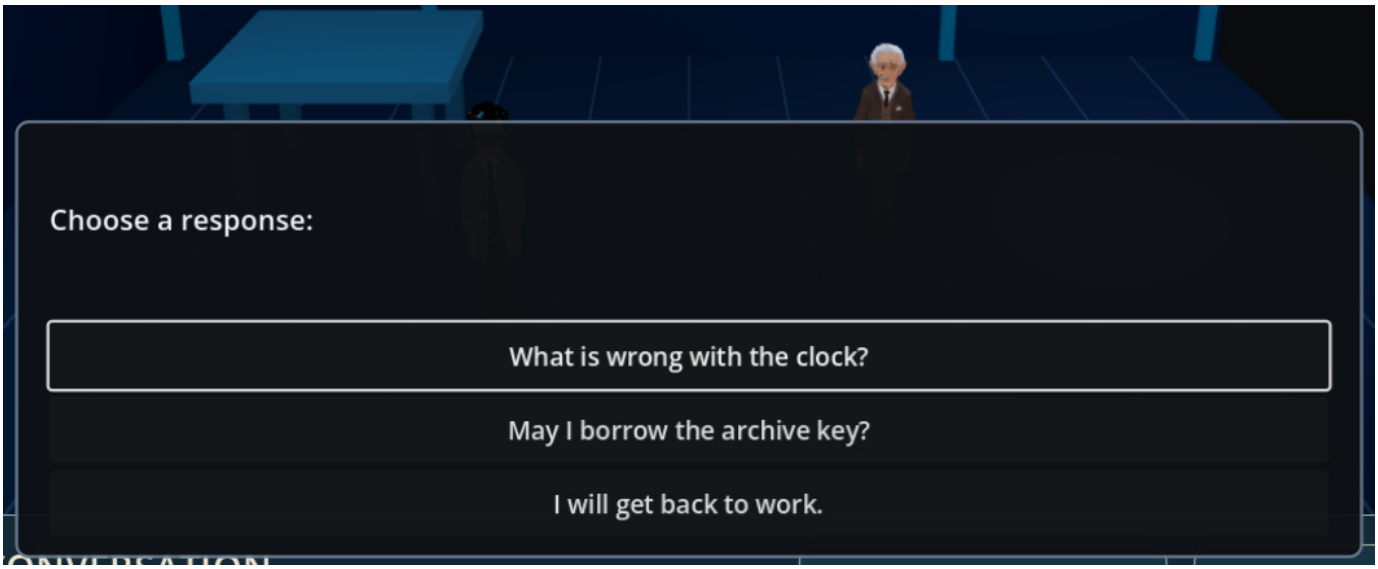

Figure 2. The example conversation running in PCA. A conversation is both a flow of dialogue and a set of game rules.

Part of the job	Where it belongs
Dialogue flow	Dialogue Designer: lines, choices, actions and endings.
Character identity and portraits	Character Designer.
Voice files and mouth timing	Asset Designer and Lip Sync tools.
Window, text and choice appearance	Global Settings and GUI Designer.
Starting a conversation in the game	A verb response, action list, script, GUI event or cutscene.

The instructions and screenshots use PCA 3D Adventure Studio 0.86.3.4. The guide is platform-neutral. All illustrations of the editor and Play view are actual captures; no practice project is required.

2. The example we will build

Elias has an archive key. Professor Ivo needs the key to find a missing brass wheel. This is enough of a story to exercise a real conversation without making you manage twenty characters at once.

Player asks	Elias / the game responds
What is wrong with the clock?	Elias describes the missing wheel, then returns to the topic menu.
May I borrow the archive key?	The game gives the key, records that it was given and plays Elias’s reply.
What should I look for in the archive?	This question appears after the key has been given.
I will get back to work.	The conversation ends.

The screenshots use Professor Ivo and Elias from the bundled Clockmaker content in a simple workshop room. You may use your own characters. The dialogue structure is the same in 2D, 2.5D and 3D; your character art and presentation are separate choices.

Prepare these records

Record	Example ID	Purpose
Starting player	ada	Professor Ivo; the special speaker player resolves to the active player.
NPC	elias	The person being questioned.
Inventory item	archive_key	The item Elias can give.
Boolean variable	archive_key_given	Starts false; becomes true after the reward.
Dialogue	archive_chat	The complete conversation.

Names can change without changing IDs. The sample uses readable node IDs such as greeting and topics. Your editor may generate line_1 or choice_1 instead. Record your actual IDs, or use the complete dialogue record in Appendix A when you want the exact sample names.

3. Open a dialogue and find its controls

1. Open Adventure Database → Dialogues, or select the Dialogue Designer from the designer toolbar.
2. Choose an existing dialogue in the selector, or use Add New.
3. Give the conversation a useful display name and a stable ID.
4. Find the canvas, the node toolbar and the Start indicator.
5. Save the project before experimenting with a large existing conversation.

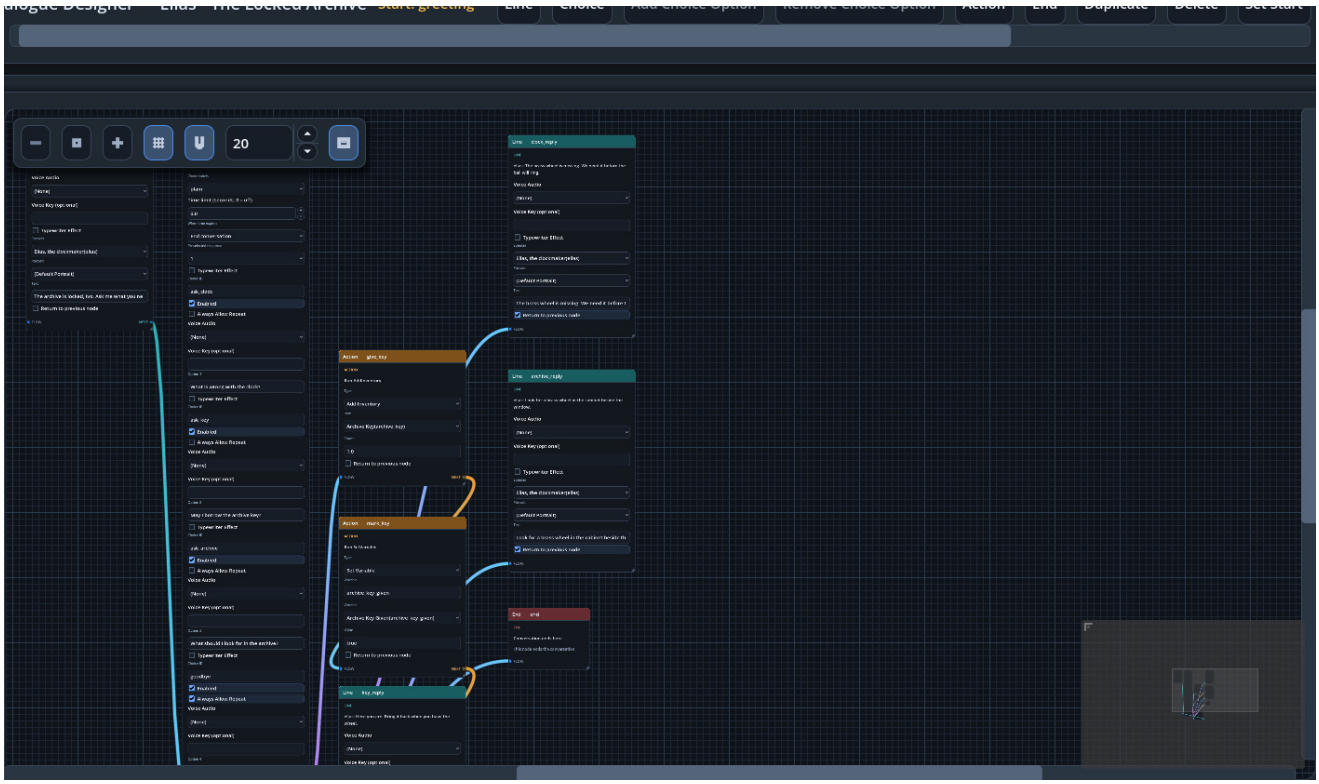

Figure 3. The graph provides a map of the conversation. Close-up figures later show the fields at a readable editing scale.

The outer selector chooses the dialogue record. The graph toolbar works on nodes inside that selected record. Add New creates another conversation; Line or Choice creates a node in the current conversation.

An ID identifies a record. A title describes it to you. For example, Elias - The Locked Archive is the display name, while archive_chat is the reference used by Start Dialogue actions.

> A new dialogue in this version starts with a single End node named start. It is a valid empty conversation, so testing it immediately ends without showing text. Add a Line and make that Line the start before expecting a greeting.

4. The four node types

Node	What it does	Typical use
Line	Shows a speaker’s line, with optional voice, portrait override and typewriter effect.	A greeting or answer.
Choice	Presents response options. Each option has its own destination.	A topic menu or decision.
Action	Runs one action from the shared action catalog.	Give an item, change a flag or call a script.
End	Finishes this conversation normally.	Goodbye or the end of a branch.

The flow wires connect a node’s output to another node’s FLOW input. A Line or Action has one NEXT output when automatic return is off. A Choice has one output for each response. An End has no outgoing flow.

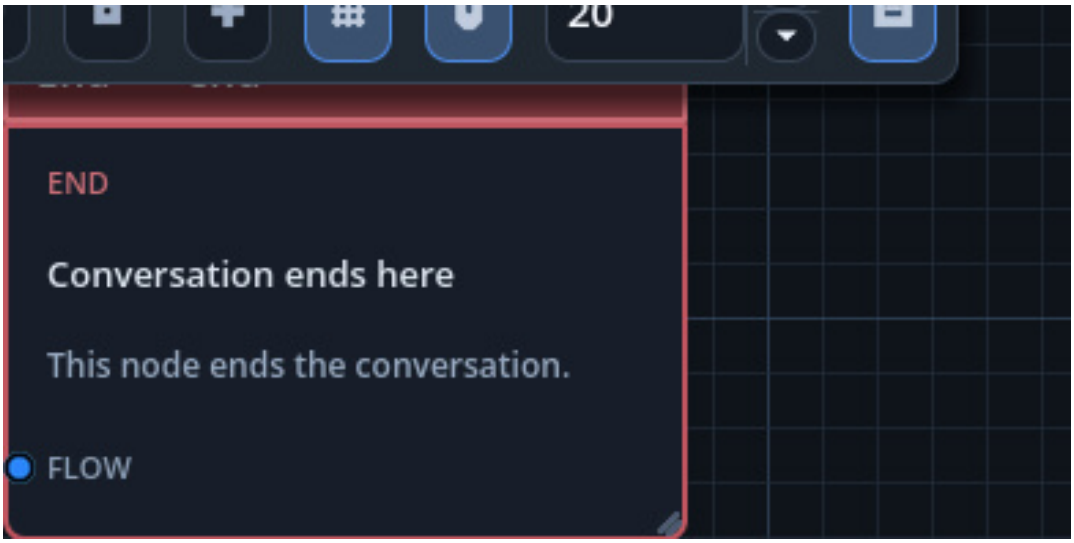

Figure 4. An End node has a simple job: finish this conversation. It is different from an End Dialogue action that interrupts active conversations.

There is no separate Condition node in this designer. Conditions can control individual options, and Action nodes can call scripts for more involved decisions. We will cover both routes.

> A wire means “continue here.” It does not mean “this object belongs to that character,” and it does not carry a variable value as a data cable would in some visual-script systems.

5. Move around the graph without losing your place

Start by zooming out until you can see the shape of the conversation. Zoom in before editing a line, reading a Choice ID or connecting a small port. The graph’s zoom buttons and percentage control are more useful than trying to edit everything at one scale.

Control	Use
Canvas scrollbars / panning	Move across a conversation larger than the visible area.
Zoom controls	Inspect details or see the wider layout.
Minimap	Find the part of a large graph that is outside the current view.
Auto Arrange	Reposition nodes using their connections and actual sizes.
Reset View	Return the graph to its default zoom and scroll origin.
Node border / resize handle	Give a node more editing room.
Toolbar arrows / horizontal scrolling	Reach buttons that do not fit in a narrow workspace.

Choice nodes can become tall because every option includes its own settings. Pan down inside the graph to reach the later options and output ports. A small overview is a map; it is not a comfortable place to edit a voice key.

Use the Panels controls to make more room when needed. The Inspector is useful for record settings and Raw JSON, while the graph fields are edited on the nodes themselves.

> Auto Arrange changes layout, not story logic. It cannot decide which branch should follow an answer, and Reset View does not repair an incorrect start node.

6. Write the first spoken line

1. Choose Line in the graph toolbar.
2. Select the new node and set Speaker to Elias.
3. Leave Voice Audio as None for the first test.
4. Leave Portrait on Default Portrait unless the character already has an alternate portrait you want.
5. Enter: The archive is locked, Ivo. Ask me what you need.
6. Press Enter in the text field, or click out of it, to commit the text.

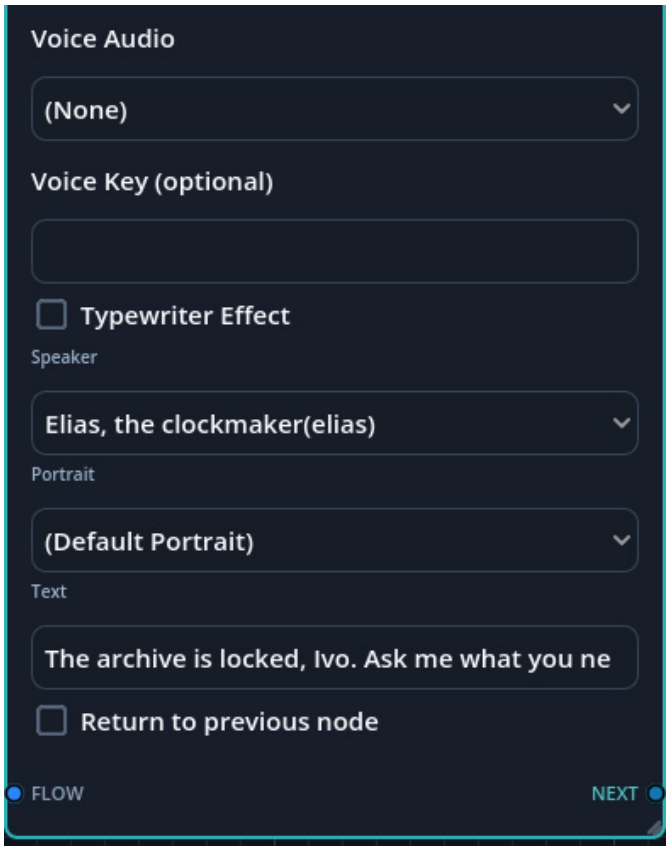

Figure 5. A Line node contains its speaker, text, portrait and optional voice settings.

The text field commits on Enter or when it loses focus. Dropdown and checkbox changes are applied as you change them. Save Project still writes the project to disk; editing a node is not a substitute for saving the project.

> Keep the first line short while testing. You can improve the wording after the player can reach the choices and leave the conversation correctly.

7. Choose the right speaker

The Speaker field identifies the character who says the line. It controls the displayed name, character appearance settings, voice speaker binding and the portrait that can be selected.

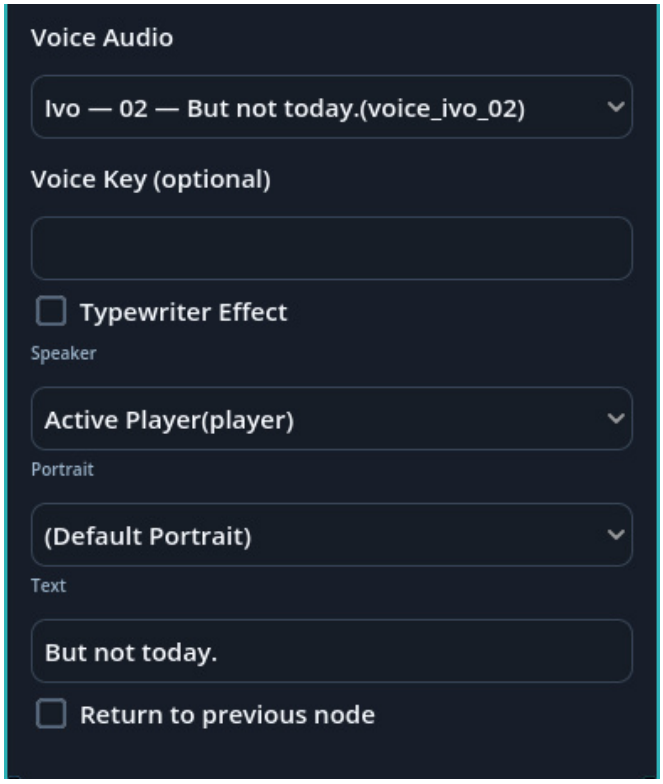

Figure 6. The special speaker player follows the active player character. A fixed character ID always refers to that specific character.

Speaker selection	Use it when…
player	The line belongs to whoever the player currently controls.
Professor Ivo (ada)	Ivo must speak even if another character is controlled.
Elias (elias)	Elias is the speaker.

A line’s speaker is not a command to make that character the player. If the wrong name or portrait appears, check the Speaker first, then the character’s display name and default portrait.

For narration, a straightforward route is an Action node with Show Message. That uses the narrator/message presentation rather than pretending a non-character is a normal dialogue speaker.

> Selecting a character does not create a voice recording. With no voice asset, the line is still valid text dialogue, and any text-based mouth fallback depends on the character’s lip-sync settings.

8. Set the greeting as the start

1. Select the greeting Line.
2. Choose Set Start in the toolbar.
3. Check the Start indicator and the star on that node’s title.
4. Turn off Return to previous node on the greeting.
5. Connect its NEXT output to the existing End node for the first test.

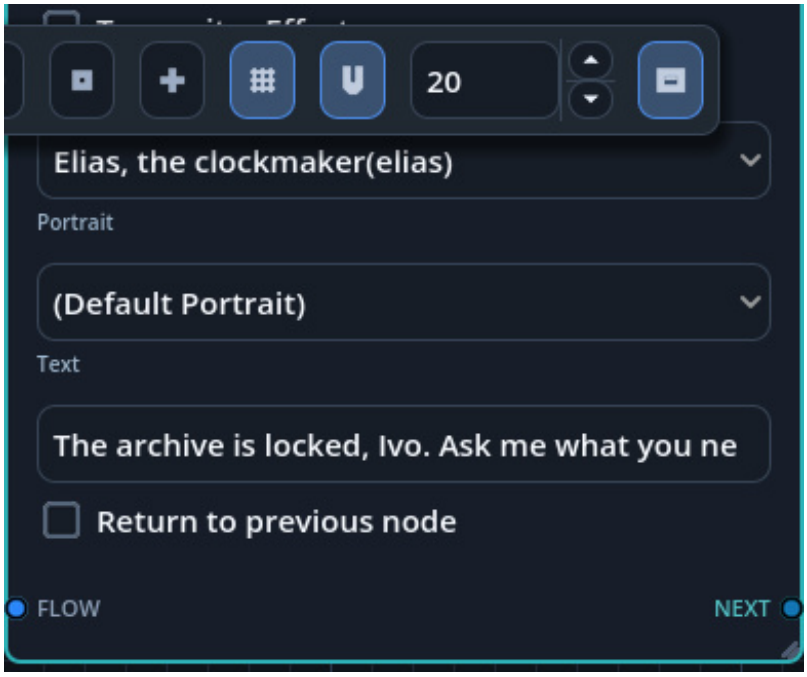

Figure 7. An opening line needs explicit forward flow. Its NEXT port is available when Return to previous node is off.

Starting a dialogue begins at the record’s start node. Selecting another node in the editor does not make Test Dialogue start at that selected node.

Automatic return needs a previous node from this run. A greeting used as the start has no caller. If its Return to previous node option is left on, it can display its line and then finish instead of following a NEXT wire.

> New Line and Action nodes default to automatic return. That is convenient for short answers reached from a menu, but an opening line or an action sequence normally needs it turned off.

9. Connect and replace flow wires

Find the output dot beside NEXT near the bottom of the greeting. Drag it to the FLOW input on the End node. If you cannot see the bottom of a tall node, pan the canvas or zoom out until its ports are visible.

Source	Output	Destination
Greeting Line	NEXT	End node FLOW
Choice node	One named option	That option’s answer or action
Action node	NEXT	The next action or spoken line

Each output has one destination. Connecting the same output to a different node replaces that destination; it does not create two simultaneous branches. Separate choices need separate option outputs.

The graph supports disconnection from connected ports. After disconnecting or deleting a node, inspect the remaining outputs. An empty destination can finish a branch without saying anything.

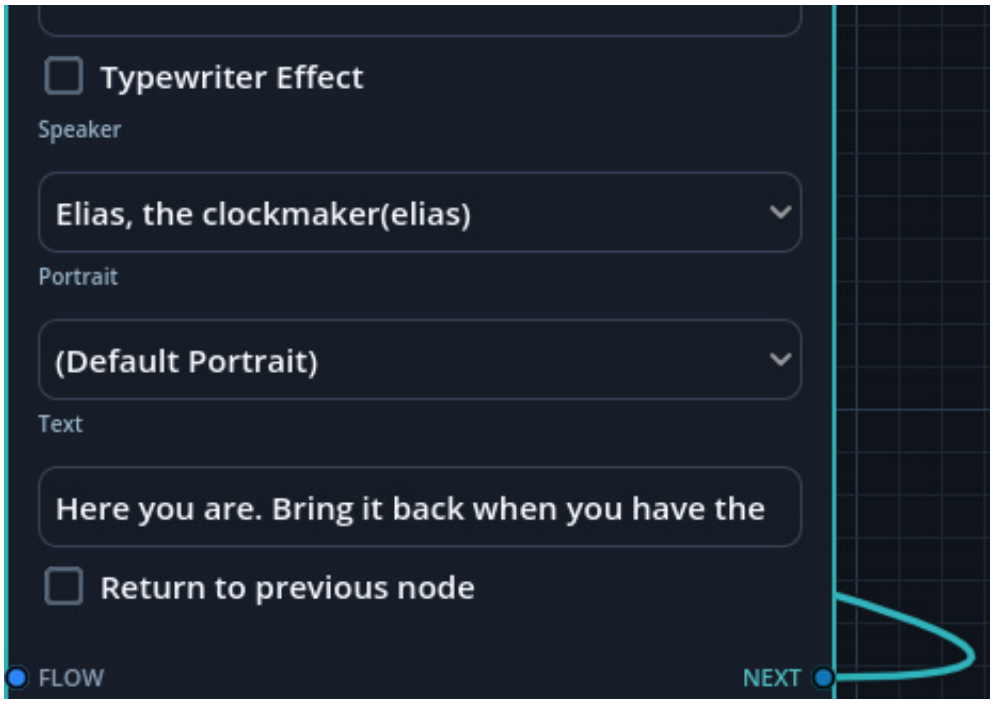

Figure 8. An explicit NEXT link is visible on a node whose automatic return is disabled.

A node can retain an old NEXT value while automatic return is on, but that NEXT route is not the active route. Turn automatic return off and verify the resulting wire before relying on it.

10. Test the tiny conversation first

1. Choose Test Dialogue.
2. Check that the greeting appears under Elias’s name.
3. Use Continue or your configured dialogue-continue input.
4. Check that the conversation closes and gameplay is available again.
5. Return to the designer and save the project.



Figure 9. The first successful test is one greeting followed by a clean ending.

Test Dialogue launches a fresh preview with the current editor data and runs the selected dialogue from its start. It is useful for a quick conversation check, but it also resets the preview’s test state.

Later, test the real Talk interaction in Play / Restart as well. A graph can work perfectly through Test Dialogue while the NPC’s Talk response still points to an older dialogue.

> This build includes a 30-second watchdog for an idle spoken Line. The timer is held back while typewriter text is revealing. Do not rely on leaving a Line open forever while inspecting another screen.

11. Add a topic menu

1. Add a Choice node.
2. Replace its first option text with What is wrong with the clock?
3. Replace its second option text with I will get back to work.
4. Connect the greeting’s NEXT output to the Choice node.
5. Add an answer Line for the first option and use the End node for the goodbye option.

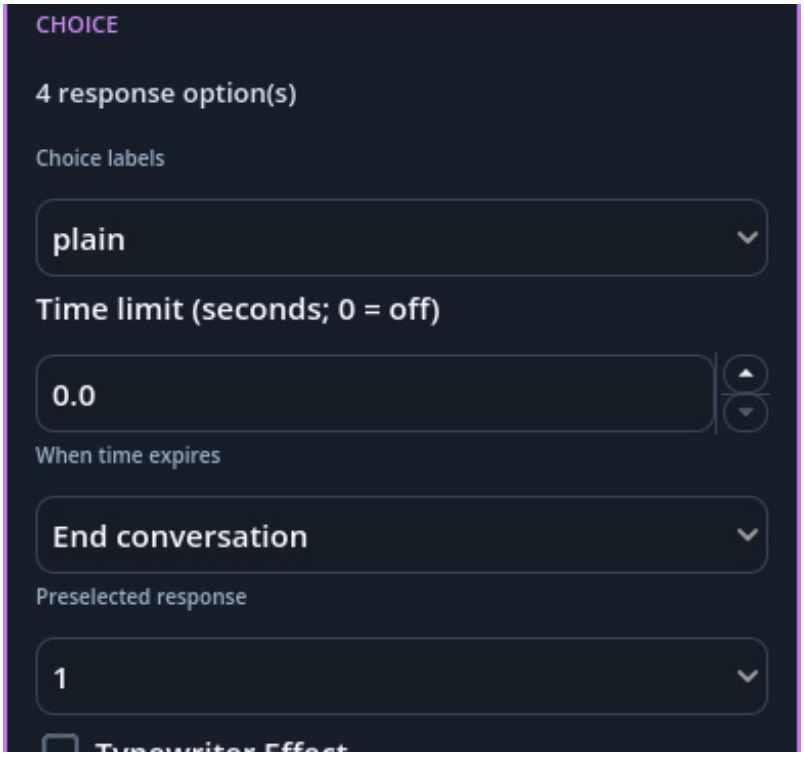

Figure 10. A Choice node starts with two options. Timing and label settings apply to that menu.

The menu text is what the player chooses. The destination is what happens after the choice. The chosen option does not automatically write an NPC answer; you must connect it to one.

For the first answer, use Elias as Speaker and enter: The brass wheel is missing. We need it before the bell will ring. Leave Return to previous node on for this short answer.

> The runtime does not infer which response should follow an option just because their wording is related. Every intended option needs a destination.

12. Add and remove response options

Select a Choice node before using Add Choice Option or Remove Choice Option. Those buttons operate on the selected menu, not on whichever text field you last clicked.

Button	What it changes
Add Choice Option	Appends one response to the selected Choice node.
Remove Choice Option	Removes the last response in that node.
Delete	Deletes selected graph nodes, not one response inside a menu.
Duplicate	Copies selected nodes, with their outgoing targets cleared.

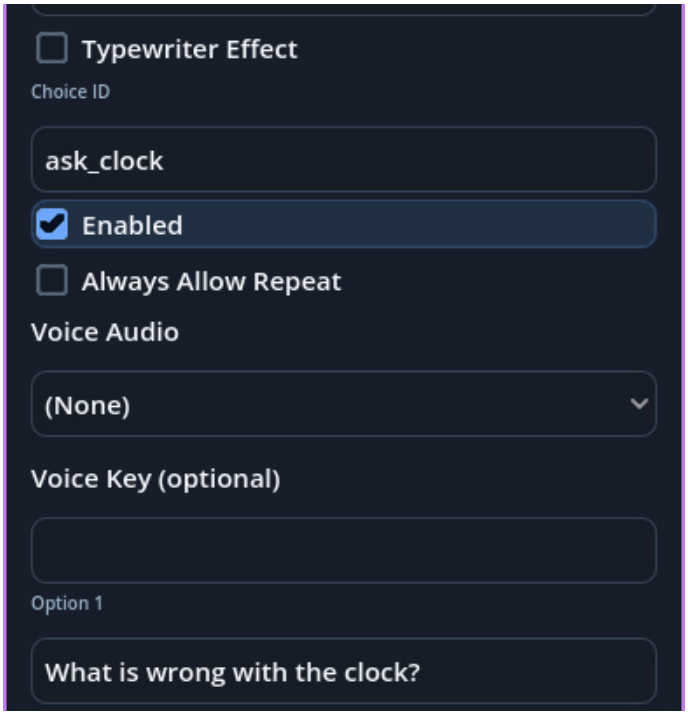

Figure 11. Each response has an ID, Enabled and Always Allow Repeat switches, voice fields and its displayed text.

The current Remove Choice Option button removes the last option. It is not a “delete the option whose text field has focus” command. To remove a specific middle response, carefully edit the record’s options array in Raw JSON, or reorganize the menu first.

The editor keeps at least one option when using its normal removal button. A menu also needs at least one available response at runtime; a conversation with no selectable responses ends that branch.

After adding an option, remember its output port near the bottom of the Choice node. Writing its text alone does not connect it to an answer.

13. Give every response a stable Choice ID

A Choice ID identifies one response inside a Choice node. It is different from the dialogue ID and the node ID. Those three references together identify the response’s read history and runtime edits.

Reference	Example	What it identifies
Dialogue ID	archive_chat	The whole conversation.
Node ID	topics	One Choice node in the graph.
Choice ID	ask_clock	One option inside topics.

PCA supplies generated Choice IDs where needed. You may replace them with meaningful names such as ask_key and goodbye. Keep them unique within their menu and stable once saves or scripts refer to them.

Read history is attached to these IDs, not to the sentence shown on the button. Rewording “May I borrow the key?” does not have to erase its history. Changing its Choice ID intentionally gives it a different history identity.

The same Choice ID can exist in two different menu nodes without sharing history. For example, menu_a/help and menu_b/help remain separate response histories.

> Do not use option position as a permanent identity. A timed preselected response uses an index, but runtime choice-editing actions use the stable Choice ID.

14. Return to the menu without a second wire

A short answer is a natural use for Return to previous node. The topic menu sends the flow to the answer; when that Line finishes, it returns to the node that entered it during this conversation.

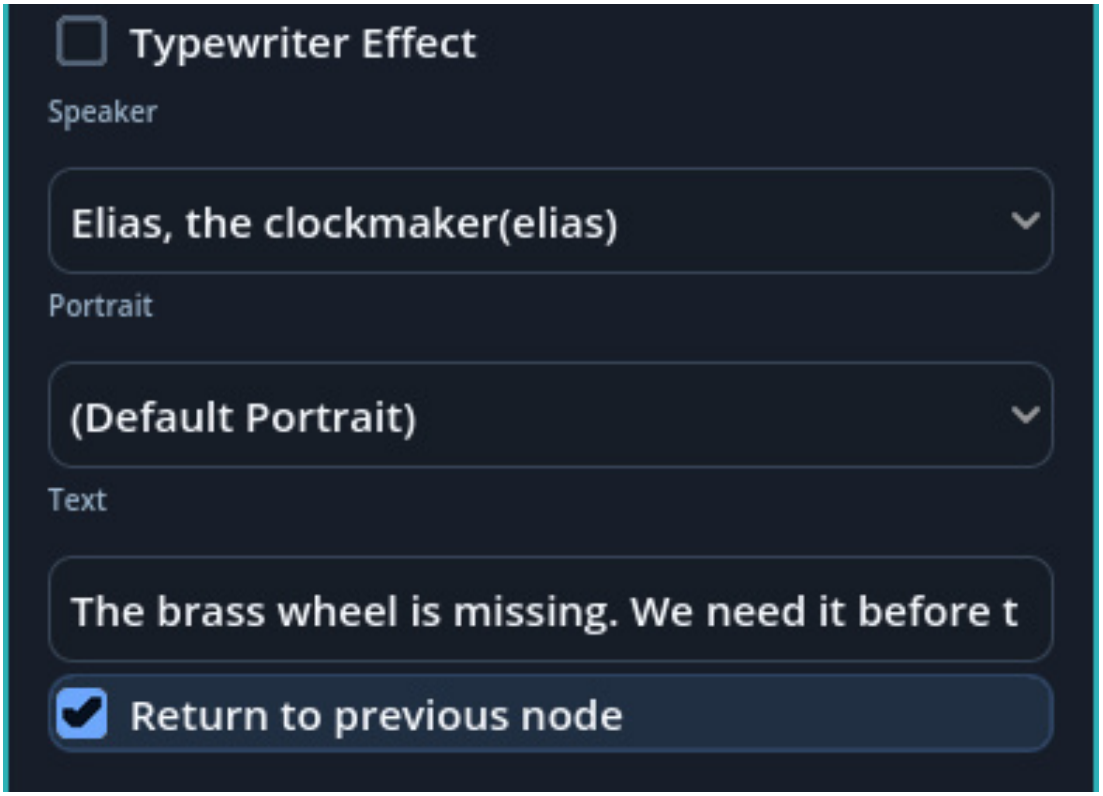

Figure 12. With automatic return on, the answer has no NEXT output to connect.

Node	Return to previous node	Result
Greeting	Off	Follows NEXT into the topics menu.
Clock answer	On	Returns to the menu that called it.
Goodbye End	Not applicable	Finishes the conversation.

You can reuse that same answer from another menu. It will return to whichever menu the player used, rather than always jumping to a hardcoded topics node.

> “Previous” means the immediate caller in this run. It does not mean “always return to the most recent Choice node,” and it is not a permanent wire stored on the answer.

15. Chain several nodes with explicit flow

Suppose borrowing the key requires three steps: give the item, mark the story flag and show Elias’s reply. If every node returns automatically, the chain will not run as intended.

Node	Action / line	NEXT	Automatic return
give_key	Add Inventory: archive_key, 1	mark_key	Off
mark_key	Set Variable: archive_key_given = true	key_reply	Off
key_reply	Here you are. Bring it back...	topics	Off

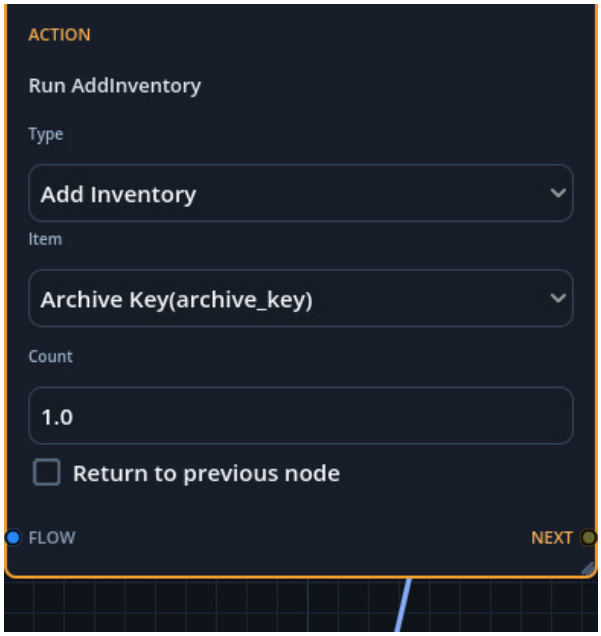

Figure 13. The first action follows explicit NEXT flow to the flag-setting action.

The final Line explicitly returns to topics. That makes the whole sequence unambiguous. An automatic return on the final Line would return to mark_key, its immediate caller, rather than skipping back to the menu.

> Automatic return is ideal for a single called answer. For a sequence of lines or actions, turn it off through the sequence and connect the intended final destination explicitly.

16. Reuse an answer from two menus

A reusable answer can keep a large conversation easier to maintain. For example, both a general topic menu and a workshop-details submenu might ask where Elias left a key.

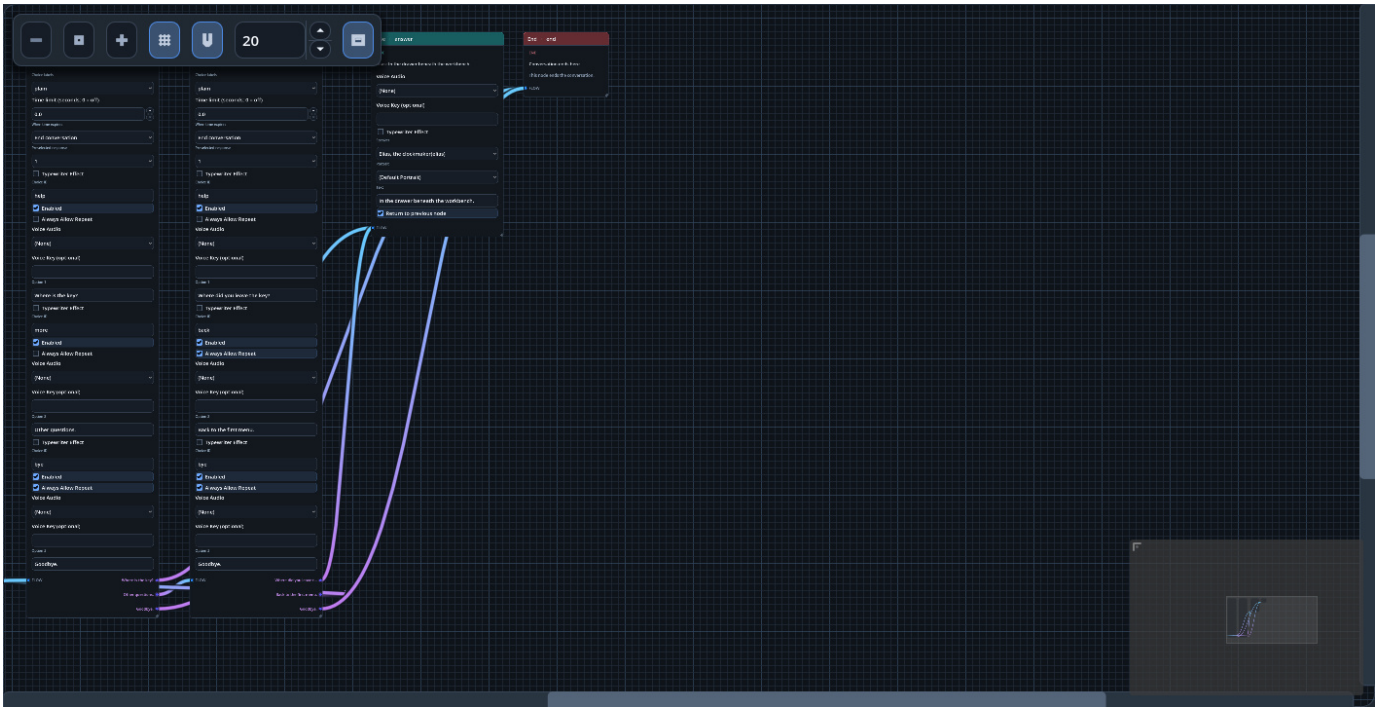

Figure 14. Two menus can call one automatically returning answer. The current conversation path determines where it returns.

Starting menu	Choice destination	After answer
menu_a	answer	Back to menu_a.
menu_b	answer	Back to menu_b.

Use explicit connections for the menu-to-menu choices such as Other questions and Back. Leave automatic return on only for the shared one-Line answer.

This works because the runtime tracks the current path. It also recognizes returning to an ancestor through an explicit link, which prevents an ordinary topic loop from adding another copy of the same menu to the path forever.

> Do not reuse a reward branch merely because its words are similar. Shared wording is easy; shared state changes need their own ownership and “already done” guards.

17. Duplicate, delete and tidy safely

Duplicate is useful for similar lines or action settings. After duplicating, reconnect the copy’s outputs. PCA clears outgoing targets on copied nodes so the new branch does not silently inherit an old route.

Operation	Check afterward
Duplicate a Line	Speaker, voice, portrait, text, return mode and NEXT.
Duplicate a Choice	All option text and IDs, then every output destination.
Delete a node	Incoming links that may now be empty; the start node if it was deleted.
Change an Action Type	The newly displayed fields; the previous action’s fields are replaced.
Auto Arrange	The whole layout, including long loops and tall Choice nodes.

Deleting through the editor clears references to the deleted destination. Removing a node through the runtime Remove Dialogue Node action has different bypass rules, explained later. Do not assume the two operations are identical.

Save before a major rearrangement. Use Undo if you delete the wrong node. Keep a named End node visible in the graph so the exit route is easy to check.

> A tidy layout makes mistakes easier to notice. It does not prove that conditions, read history or inventory rewards are correct; those need Play tests.

18. Build the complete archive conversation

Node ID	Type	Purpose / continuation
greeting	Line	Elias’s introduction; NEXT topics; return off.
topics	Choice	Four authored options; only the available ones are shown.
clock_reply	Line	Explains the missing wheel; return on.
give_key	Action	Add Inventory, then mark_key; return off.
mark_key	Action	Set archive_key_given true, then key_reply; return off.
key_reply	Line	Acknowledges the loan; NEXT topics; return off.
archive_reply	Line	Describes the cabinet; return on.
end	End	Normal exit.

Connect the topics outputs to clock_reply, give_key, archive_reply and end. The next chapters add the state conditions that hide the key request after it succeeds and reveal the archive question.

Create archive_key in the Inventory Designer and archive_key_given as a Boolean variable starting false. Without those records, a correct-looking action node can still refer to a missing item or the wrong variable.

> Appendix A contains the complete example dialogue record if you want its exact IDs. It is optional: the normal graph controls can build the same conversation with generated node IDs.

19. Put game logic in Action nodes

An Action node runs one action from PCA’s shared catalog, then follows its flow rule. You can use the same kinds of actions that appear in action lists and visual scripts.

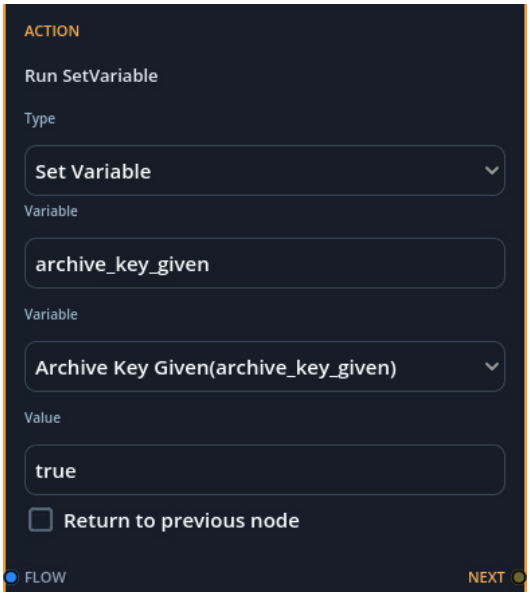

Figure 15. This Action changes a Boolean story variable before the next spoken reply.

You want to…	Action
Record a story change	Set Variable.
Give the active player an item	Add Inventory.
Give a specific character an item	Give Inventory.
Remove a carried item	Remove Inventory.
Run project code	Run Script, or Call Script Function for exposed functions.
Start another conversation	Start Dialogue.
Play a cinematic	Start Cutscene.
Change a camera or animation	The relevant camera or animation action.

Choose Type first, then fill in the fields it displays. Changing Type replaces the old action configuration with defaults for the new type. Review every field after switching types.

> An Action node is one step. To perform several steps in order, connect several Action nodes, or call a script/action list that owns the sequence.

20. Give the key and record the loan

1. Create the Archive Key item in the Inventory Designer with ID archive_key.
2. On give_key, choose Add Inventory and select Archive Key. Set Count to 1.
3. Turn automatic return off and connect NEXT to mark_key.
4. On mark_key, choose Set Variable. Select archive_key_given and set its value to the Boolean true.
5. Connect mark_key to key_reply, then key_reply back to topics.

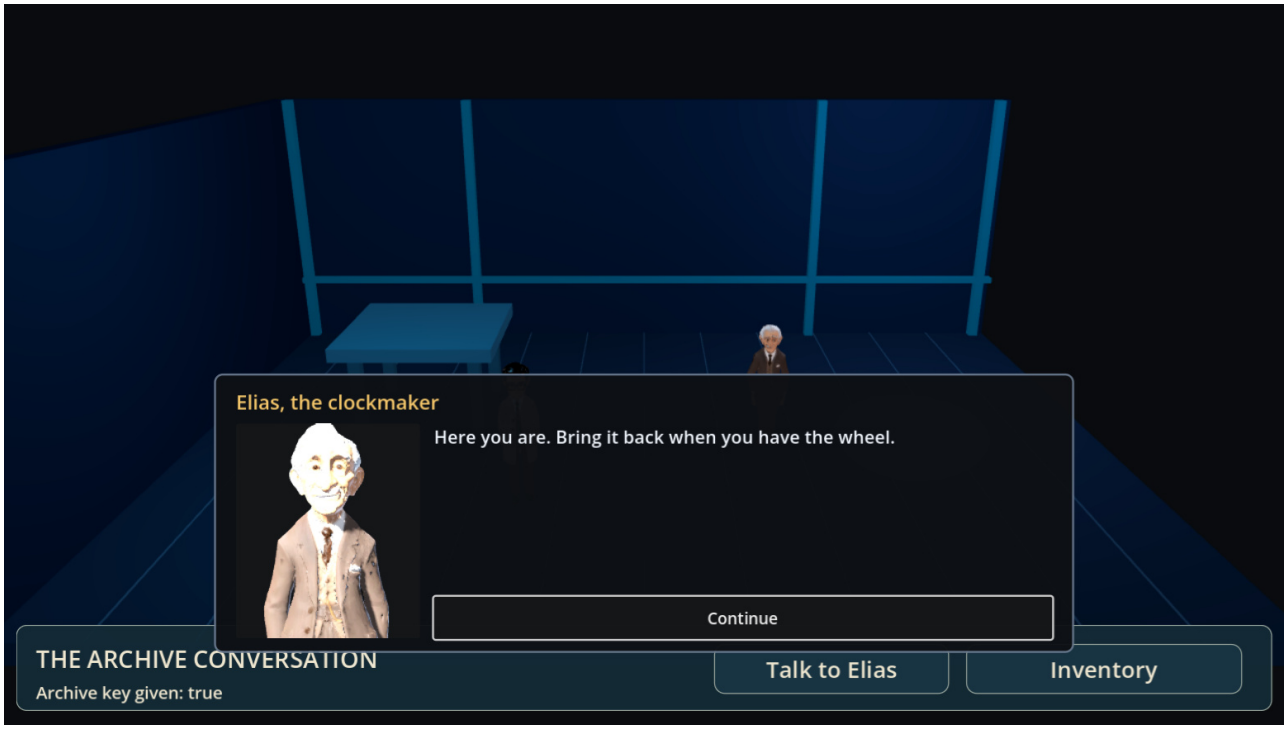

Figure 16. Elias acknowledges the loan after the inventory and variable steps have completed.

Add Inventory means the active player’s inventory. Give Inventory lets you choose a particular character. That distinction matters in a game with several playable characters and separate bags.

The story flag means “Elias has already given this key.” It is not exactly the same as “the player is carrying the key right now.” A later puzzle may remove, drop or transfer the item while the loan remains part of the story.

> A non-stackable item definition is useful, but it is not a complete one-time reward rule. Prevent repeat rewards through conditions or a guarded script as well.

21. Conditions hide options; Enabled disables them

Two controls answer different questions: should this response appear at all, and can the player currently select it? Decide which behavior you want before adding a condition.

Control	When it fails / is off
Option condition	The response is omitted from the displayed menu.
Enabled	The response can remain visible but is unavailable.
Read-history disabling	A previously chosen response can remain visible but disabled.
Always Allow Repeat	Exempts that response from read-history disabling; it does not override a failed condition or Enabled off.

For the key request, a condition hides the question after the loan. For a topic that is temporarily unavailable but worth showing, an explicit Enabled flag may be clearer.

In this version, the graph shows Enabled directly on each option. The simple variable-equality condition is supported in the dialogue record, but does not have its own inline node field. The next page shows the exact record entry.

> Do not look for a Condition node or a hidden condition checkbox on a Line. The native node types are Line, Choice, Action and End.

22. Add a simple variable condition

Select the dialogue record, open the Inspector’s Raw JSON (Advanced) tab and locate the desired response inside its Choice node’s options array. Add a condition object to that response.

```
{
  "id": "ask_key",
  "text": "May I borrow the archive key?",
  "next": "give_key",
  "enabled": true,
  "repeatable": false,
  "condition": {
    "variable": "archive_key_given",
    "equals": false
  }
}
```

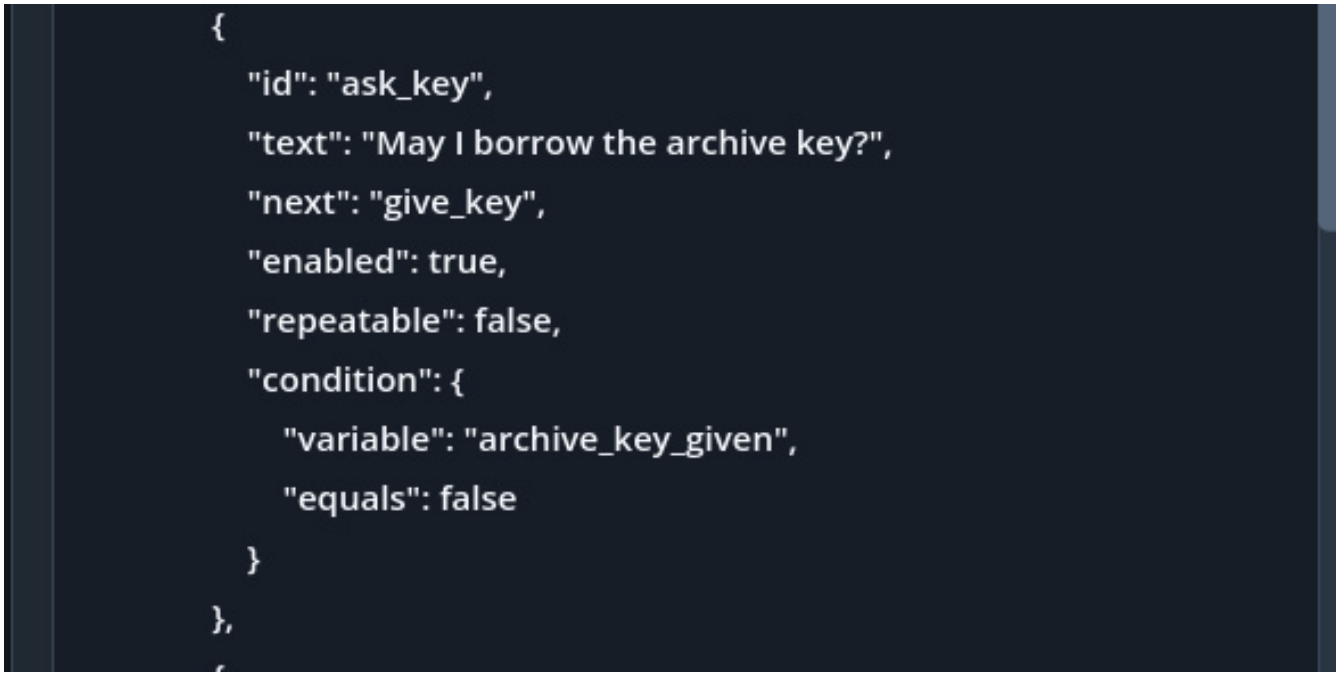

Figure 17. A condition belongs to one option. Keep the surrounding dialogue and nodes intact when editing the record.

Apply the JSON changes, return to the graph and save the project. Do not paste this small object over the entire dialogue record: it is one entry inside topics → options.

Use the Boolean false, without quotation marks. The native check compares the variable value with equals. It does not evaluate a script expression, a greater-than test or an arbitrary natural-language condition.

23. Reveal the archive question after the reward

Give ask_archive the opposite condition: archive_key_given equals true. Before the loan, the menu shows the key request. After the loan, it shows the archive question instead.

Choice ID	Condition	Destination
ask_clock	None	clock_reply
ask_key	archive_key_given equals false	give_key
ask_archive	archive_key_given equals true	archive_reply
goodbye	None; Always Allow Repeat on	end

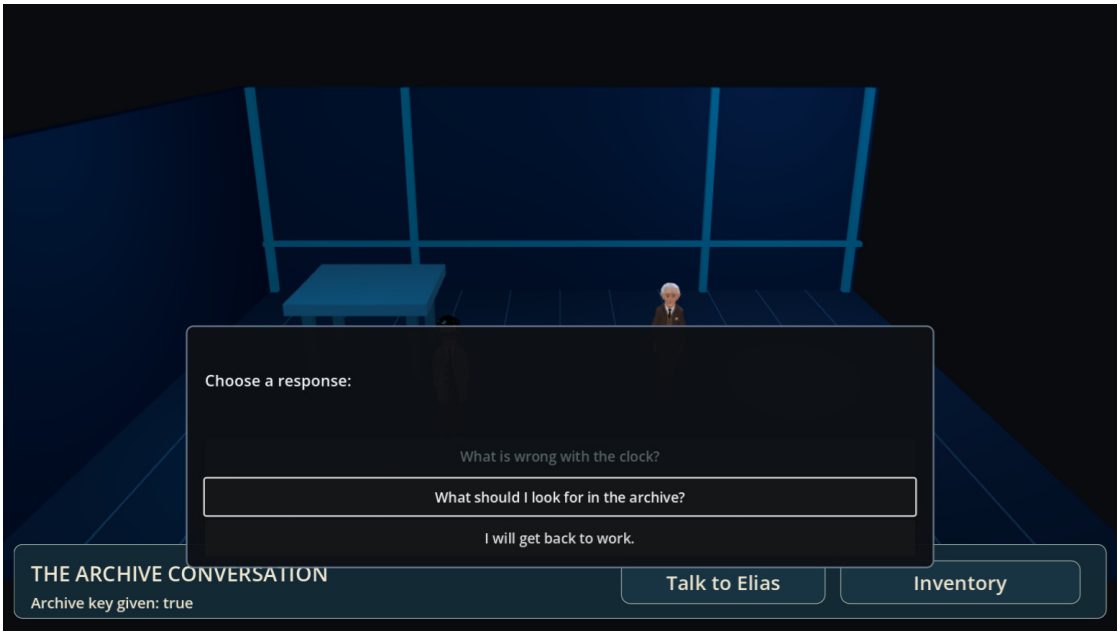

Figure 18. The key request is gone and the archive question is now available.

The Choice node evaluates availability when the menu is displayed. In this branch, the variable changes while the player is away from the menu, and the explicit return to topics displays the updated menu.

If a background system changes a variable while a menu is already open, do not assume the visible menu redraws merely because the variable changed. The dialogue-editing actions trigger a refresh; otherwise deliberately redisplay the menu at a suitable point.

> Always keep an available way out. If every response is hidden or disabled, the runtime ends that branch because there is nothing the player can select.

24. Test conditions with real starting values

1. Start a fresh preview with archive_key_given false.
2. Reach topics. Ask Key should appear; Ask Archive should not.
3. Choose the key request and continue past Elias’s reply.
4. Check real inventory for one Archive Key.
5. At topics, Ask Key should be absent and Ask Archive should appear.
6. End the conversation and start it again. The key request should remain absent.

Also test the record with archive_key_given initially true. This is useful for checking a later-game conversation without replaying every earlier scene.

The simple condition supports one variable-equality check. For a rule such as “carrying the key, trust at least three and the guard has left,” calculate a Boolean variable in a script, then use that Boolean as the condition.

> An undefined variable is not a useful false flag. Define the variable with its intended type and initial value so the condition has a predictable starting state.

25. Change availability from an action or script

Enable / Disable Dialogue Choice targets a specific response by dialogue ID, Choice node ID and Choice ID. It changes the explicit Enabled flag and refreshes an open menu.

```
await SetDialogueChoiceEnabled({
    "dialogueId":"archive_chat",
    "nodeId":"topics",
    "choiceId":"ask_key",
    "enabled":false
})
```

The same fields are available in an Action node and a visual-script action. Use it when the response should remain visible but unavailable, or when a script needs to apply a more involved rule.

Reference	Value
Dialogue	Elias - The Locked Archive / archive_chat
Choice Node ID	topics
Choice ID	ask_key
Enabled	false to disable; true to permit, subject to the other rules.

The complete guarded_talk example in Appendix B checks the saved loan flag, sets the response’s Enabled value and starts the conversation. A condition may still hide that response; enabling it does not remove its condition.

> Read status, explicit Enabled and visibility conditions are separate rules. Resetting one does not silently reset the others.

26. Protect important rewards with a guard

For valuable or repeatable conversations, a guarded function is a stronger reward boundary than relying on menu appearance alone. It can check whether the reward was already issued and whether the player already carries the item.

```
func grant_key() -> void:
    if bool(Runtime.variables.get("archive_key_given", false)):
        return
    if not HasInventory("archive_key"):
        await AddInventory({"itemId":"archive_key", "count":1})
    await SetVariable({"name":"archive_key_given", "value":true})
```

Put this inside a script extending PCAScriptBase, then expose the function so the dialogue can call it:

1. In the Script Designer, open Functions as Nodes and choose Add Function.
2. Leave Expose as Action and Visual Node checked. Set Function Name to grant_key and Display Name to Give Archive Key Once.
3. Do not add parameters for this function. Choose Use Definitions, then save the script/project.
4. In the dialogue Action node, choose Call Script Function and select Give Archive Key Once.

Use Definitions saves the callable definition; it does not run the function. The reward happens when the game reaches its Call Script Function action.

If you switch the reward branch to this function, remove the old unconditional Add Inventory step. Continue to the acknowledgement Line after the function completes.

The first flag checks whether Elias has already given the reward. The inventory check avoids adding another copy when a different route has already supplied the same item.

> This is an optional strengthening of the example. The basic graph already hides the request after a successful loan. Use the guard when other routes, scripts or replay behavior could reach the reward.

27. Mark previously chosen responses

Open Global Settings → Adventure & Cursor. Find Previously Chosen Responses and Read Response Color. These settings control how a selected response looks and behaves when the menu returns.

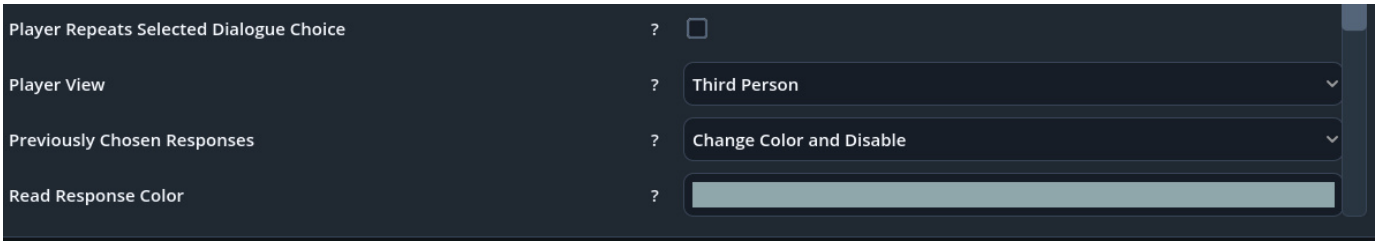

Figure 19. Read-response behavior is a project setting. Each option still has its own repeat and Enabled settings.

Mode	After choosing the response
Keep Available	It keeps its normal appearance and remains selectable.
Change Color	It remains selectable, using Read Response Color.
Disable	It remains visible but cannot be selected again.
Change Color and Disable	It changes color and becomes unavailable.

A response is marked read when it is selected, including a timed automatic selection. A timeout that simply ends the conversation does not mark an option read.

> Read does not mean that its destination was successfully completed. If selecting a response can fail a puzzle check, decide whether that response should remain repeatable.

28. Keep Goodbye and useful help repeatable

Enable Always Allow Repeat on goodbye. The player should not lose the exit merely because that response was used in an earlier conversation.



Figure 20. The clock question is marked read. The exit remains usable because its repeat setting allows it.

Repeatable help is another sensible use. A player may need to hear a clue again after returning from another room. A one-time reward, by contrast, normally needs a condition or guard.

Situation	Does Always Allow Repeat help?
Disabled only by read history	Yes. The response can be selected again.
Enabled is explicitly off	No. It remains unavailable.
Its condition is false	No. It remains hidden.
A reward action has no guard	It permits repeating the branch; it does not make the reward safe.

A read response can still change color while repeatable. Repeatability exempts it from read-based disabling, not necessarily from read styling.

29. Save history without confusing it with position

PCA saves dialogue read history along with variables, inventory and mutable dialogue data. The player can close a conversation, save, load and find the same topics marked as already chosen.

1. Choose the clock question and borrow the key.
2. Leave the conversation, then save the game.
3. Load that save and speak to Elias again.
4. Check that the clock question retains its read behavior.
5. Check that the key remains in inventory and the key request remains hidden.

History is scoped to the dialogue ID, node ID and Choice ID. Keep those IDs stable when revising a game that has existing saves. A newly worded sentence can keep its ID; a genuinely new topic should have its own ID.

Do not assume that loading resumes the exact spoken line, voice playback time or current node execution position. For a reliable checkpoint, test saving with the conversation closed and reopening it from its normal start.

> Test Dialogue launches a fresh preview. Use an actual Play save/load test to check persistent history rather than treating repeated Test Dialogue runs as one continuous game.

30. Reset history at the scope you intend

Reset scope	Fields to supply
One response	Dialogue ID + node ID + Choice ID.
One menu node	Dialogue ID + node ID; leave Choice ID empty.
One conversation	Dialogue ID only.
All dialogue history	Leave Dialogue ID empty. Narrower IDs are ignored.

```
await ResetDialogueHistory({
    "dialogueId":"archive_chat",
    "nodeId":"topics",
    "choiceId":"ask_clock"
})
```

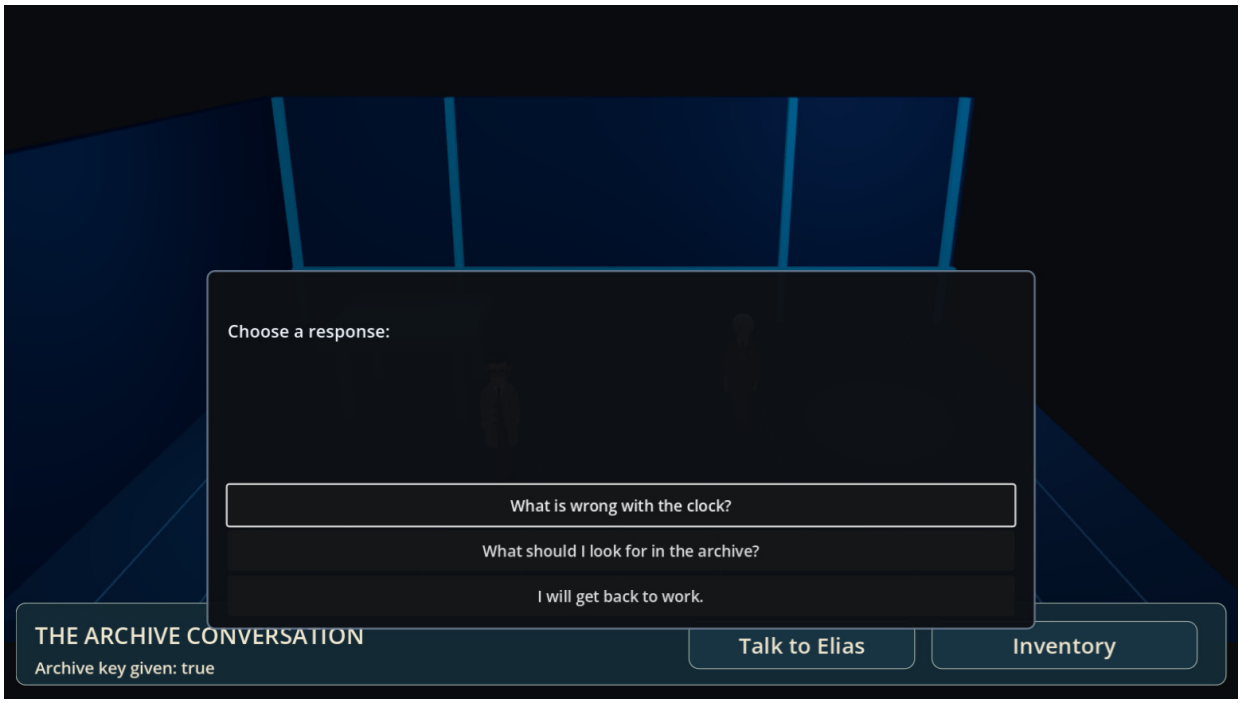

Figure 21. After resetting read history, a response can become available again if no other rule blocks it.

Resetting history does not enable an explicitly disabled response and does not alter its visibility condition. If the key was already given, clearing dialogue history does not set archive_key_given back to false.

Use a narrow reset when the story only makes one topic relevant again. Clearing all history to fix a single response can make unrelated conversations look unread.

31. Put a time limit on a decision

1. Select the Choice node that needs a timer.
2. Set Time limit (seconds; 0 = off). Start with four seconds for a quick test.
3. Choose what happens When time expires.
4. If using Preselected choice, set the Preselected response number.
5. Test with no input, then with a click just before the timer expires.

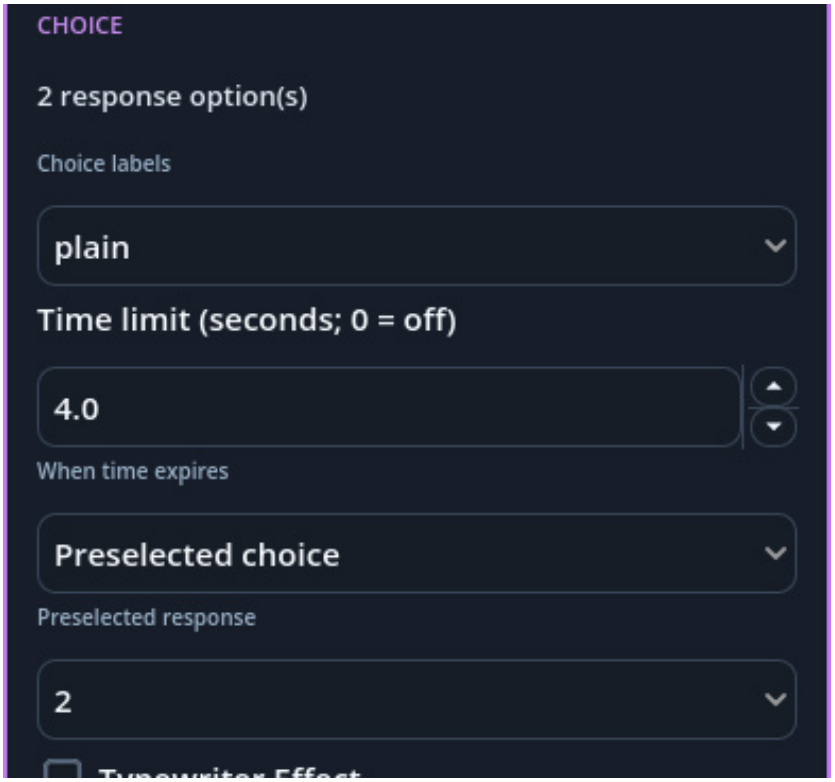

Figure 22. Timing belongs to the Choice node. It is not a timer on every individual response.

Expiry behavior	Result
End conversation	Ends the branch without selecting a response.
Random choice	Selects among available responses.
Preselected choice	Selects the configured authored response when available.

Use a generous limit for real players, particularly when choices are long or translated. Four seconds is a convenient test setting, not a recommendation for reading a paragraph under pressure.

32. Understand preselected response numbers

Preselected response numbers refer to the option’s position in the authored Choice node. They do not renumber the underlying options after a condition hides one.

Authored option	Visible now?	If preselected
1: Stop the train	Disabled or hidden	Cannot be selected by the timer.
2: Give me another minute	Available	Can be selected normally.
Unavailable target chosen as default	No	The runtime falls back to the first selectable response.



Figure 23. The built-in choice presentation shows a countdown while this test decision is waiting.

The editor shows numbers starting at 1. The stored timeoutChoiceIndex starts at 0. Use the displayed numbers in the UI; remember the zero-based value when editing raw data or a script.

Random and preselected policies exclude unavailable choices. If every response is unavailable, the branch ends instead of waiting for an impossible selection.

> An accidental fallback can have story consequences. Test each timer with its intended default available, hidden, explicitly disabled and disabled by read history.

33. Put consequences after the timed choice

The timeout selects an ordinary branch. That branch should contain the same actions and replies that would run if the player clicked the response.

Response	Possible branch
I will stop the train.	Set a decision variable; start a short cutscene; return or end.
Give me another minute.	Record hesitation; show Elias’s warning; end or offer a new choice.
No response / end policy	End the branch without marking a response read.

Do not award a result merely because the timer exists. Put the state change in the branch that represents that result. Otherwise a quick click and a timeout may affect different variables by accident.

If a runtime dialogue-editing action refreshes an open timed menu, its remaining time is retained. A refreshed menu does not automatically grant another complete time allowance.

> An end-on-timeout result needs its own story handling if silence matters. It is not secretly the first option.

34. Decide whether the player repeats a choice

The menu text can be a topic label, a complete player line or both. Global Settings includes Player Repeats Selected Dialogue Choice, which decides whether ordinary selected response text is shown as player speech before entering its branch.

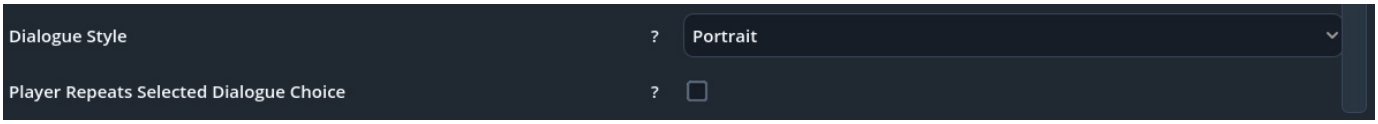

Figure 24. Player Repeats Selected Dialogue Choice is separate from whether an NPC’s answer is repeatable.

Your design	Suggested setup
Short topic labels such as “The clock”	Do not repeat the label as a spoken sentence.
Full player lines such as “May I borrow the key?”	Repeating the response can make the exchange clearer.
A separate player Line already follows the option	Avoid also repeating the option, or the player can say it twice.

A selected option with an assigned voice recording can still play that recording even when the global repeat setting is off. The runtime uses a timed player-speech presentation for that case.

> Typewriter settings on a Choice option apply when that selected response is spoken. They do not make every entry in the waiting menu type itself onto the screen.

35. Start the conversation from the NPC

1. Select Elias’s scene instance.
2. Open Verb Actions and find Talk.
3. Add Start Dialogue and choose Elias - The Locked Archive.
4. Apply the inspector changes if required, then save the project.
5. Choose Play / Restart and interact with Elias using Talk.

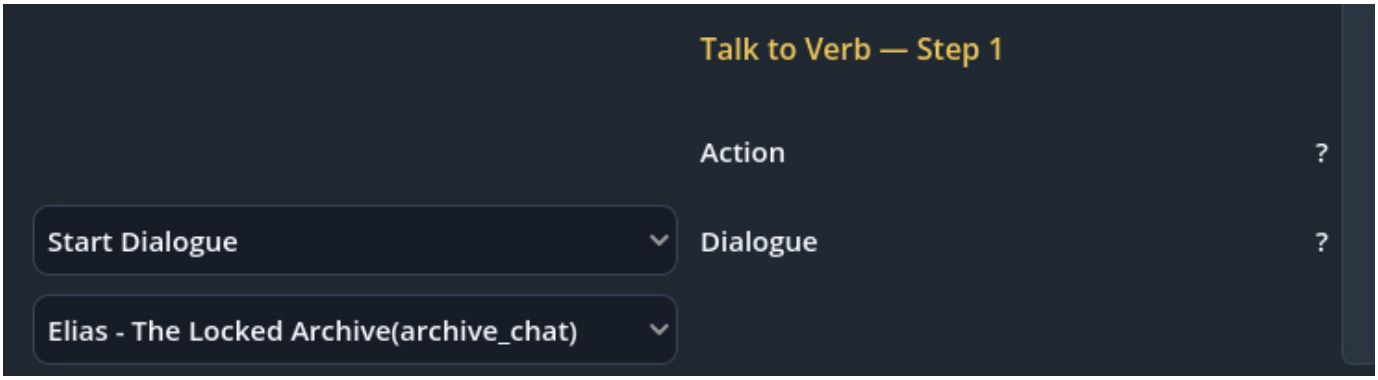

Figure 25. The Talk response points to the dialogue record. The graph’s start node controls where that record begins.

The same Start Dialogue action can be used in a GUI event, action list, visual script or cutscene. A dialogue record is reusable; its existence alone does not attach it to a character.

Check the interaction mode, hotspot availability and any automatic approach settings if the player cannot reach the Talk action. Those are separate from whether the graph itself is valid.

> Test Dialogue checks the selected conversation directly. It cannot prove that Elias’s actual Talk response uses that conversation.

36. Call dialogue from scripts and visual nodes

```
extends PCAScriptBase

func run() -> void:
    await StartDialogue({"dialogueId":"archive_chat"})
```

This starts the same dialogue and waits for it to finish before the script continues. Use the normal Start Dialogue action in a visual graph when no custom script is needed.

Run Script actions can call a project script that chooses which conversation to open. Call Script Function is useful when you expose a function with parameters, such as a reusable conversation launcher.

Entry route	Remember
Talk verb	Attach the Start Dialogue action to the correct scene character.
GUI button	Use a real Click event/action, not only a button label.
Visual graph	Connect its flow to Start Dialogue.
Script	Use the correct dialogue ID and await only when later work depends on completion.
Cutscene	Decide whether the cutscene should wait, and restore its intended presentation afterward.

Do not start the same conversation again from its own OnDialogueStarted event. That creates nested starts instead of a single entry. Keep event hooks focused on observation or deliberate state changes.

37. Call another dialogue and return

An Action node with Start Dialogue can open a smaller conversation from inside the current one. When the child reaches a normal End node, execution returns to the parent Action and continues through its NEXT route.

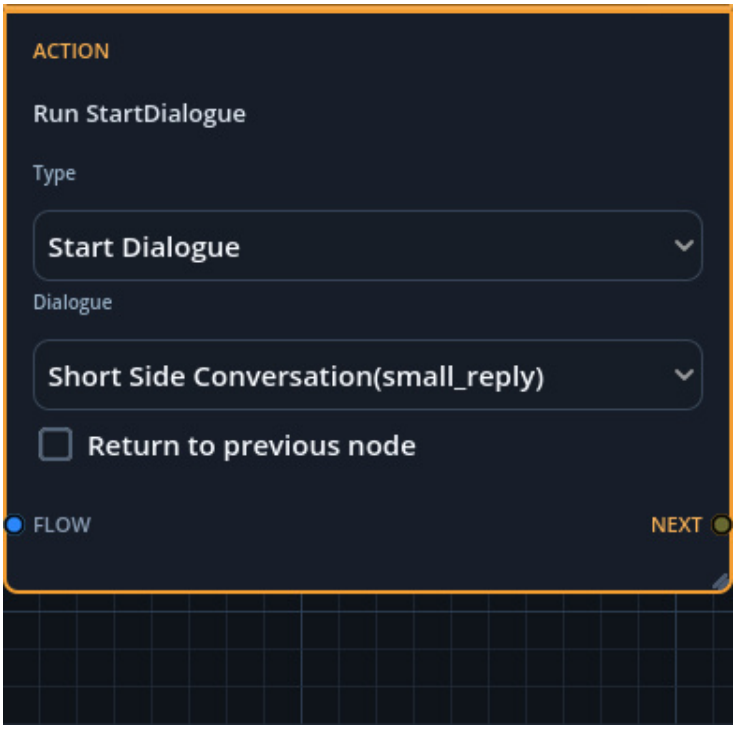

Figure 26. This Action starts a side conversation, then continues into the parent’s next Line.

Flow step	What happens
Parent Action	Start Dialogue: small_reply.
Child Line	Elias gives the extra warning.
Child End	The child finishes normally.
Parent NEXT	Ivo resumes the main conversation.

Turn automatic return off on the parent Action when you want it to continue forward afterward. Keep the child’s exit clear. Nested conversations are useful for reusable side exchanges, not for hiding an endless loop.

> The runtime limits nesting to 16 active dialogue sessions. More practically, keep nesting shallow enough that you can explain which conversation should resume next.

38. End normally or interrupt deliberately

An End node finishes the current conversation normally. An End Dialogue action is an interruption command. In this version it cancels the active dialogue sessions, including a nested parent and child.

Use	Choose
Goodbye from an ordinary conversation	End node.
Finish a reusable side dialogue and resume its caller	End node in the side dialogue.
Interrupt the conversation because something in the game must take over	End Dialogue action, with the interruption behavior tested.

Interruption stops voice playback and closes the dialogue presentation. Do not use End Dialogue as a casual substitute for the End node when you expect a nested parent to keep running.

> OnDialogueEnded event data distinguishes interrupted completion. Use that flag if a reward or quest update should happen only after a conversation finishes normally.

39. Add a recorded voice to a Line

1. Import the recording through the Asset Designer as an audio asset. Give it a clear name and stable ID.
2. Preview the recording. Check that it contains the intended line, with no unwanted silence at either end.
3. Open the dialogue and select the matching Line node.
4. Choose the asset in Voice Audio. Keep the written text consistent with what was recorded.
5. Test the conversation in Play with voice volume audible. Try advancing both during and after the recording.

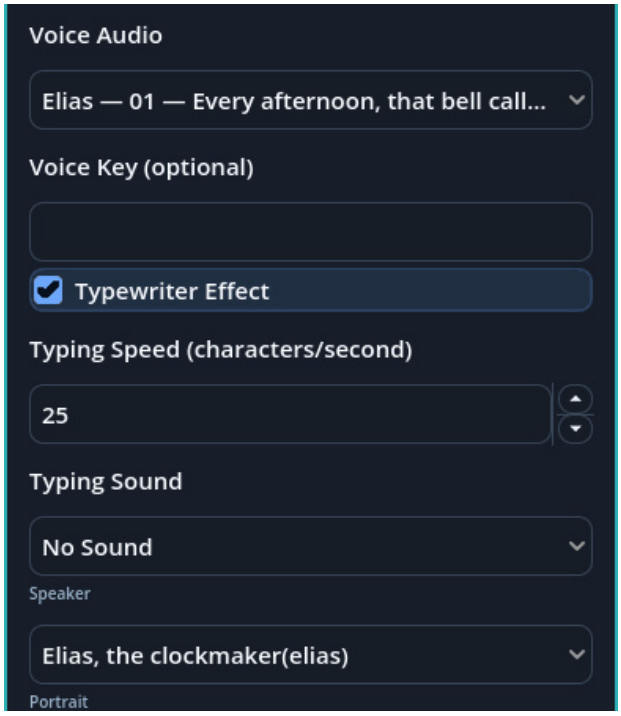

Figure 27. The assigned recording and typewriter settings belong to this spoken Line.

The example uses an existing Elias recording: “Every afternoon, that bell calls the town home.” The next Line uses the player speaker and Ivo’s recording, “But not today.” Each recording is assigned to its own Line.

A normal dialogue Line can be advanced by the player; it is not a command to force the entire sound file to finish. Decide whether that is appropriate before using a recorded line as the only source of an essential clue.

> Text alone does not create a voice recording. Typewriter blips are short interface sounds; they are not spoken dialogue.

40. Use Voice Key for language-specific recordings

Voice Audio is a direct asset reference. Voice Key is an optional lookup key for a translated voice mapping. A stable key such as archive.greeting is easier to maintain than a lookup that depends on the full wording of a sentence.

Route	How the runtime finds a recording
Voice Audio selected	Uses that audio asset, with a language-specific asset replacement when one is mapped.
No direct asset; Voice Key supplied	Looks up the key in the voice translations for the selected voice language.
Neither supplied	Can look up the original source text in the voice mapping. Without a matching recording, text can still appear.

Choose one clear route for each line. When an explicit Voice Audio asset is present, do not expect an unrelated Voice Key entry to override it. Use the asset’s language replacement, or remove the direct asset and use the key-based route.

Subtitle language and voice language can be separate runtime options. Test the combinations you intend to support: for example, Italian subtitles with the original English performance.

> The lookup can fall back through language variants and the default language. A successful preview does not prove that the intended translated recording was found; listen to the actual result.

41. Reveal text with the typewriter effect

Field	Meaning
Typewriter Effect	Reveals the displayed sentence gradually.
Characters per second	Controls reveal speed; the field supports 1-240, with 30 as the normal default.
Typing sound: Default	Uses the built-in short typing sound.
Typing sound: None	Reveals silently; useful with recorded dialogue.
Typing sound: Custom	Uses your selected audio asset as a short typing blip.

Use short, quiet sounds for custom blips. A long spoken recording is not a good typing sound. The blip system does not promise one sound for every letter, and spaces are not useful sound triggers.

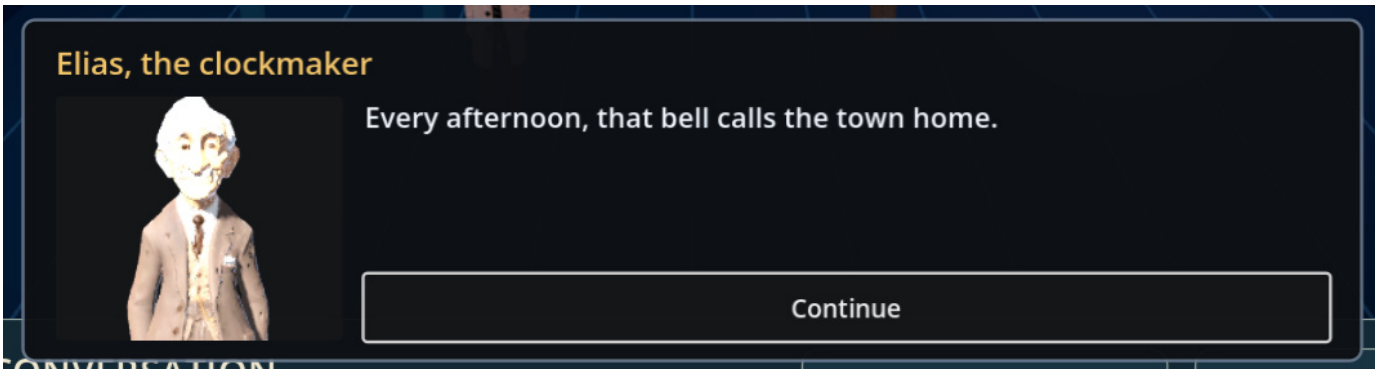

Figure 28. After the first Continue press, the complete sentence is visible and the Line is still waiting.

Continue has two stages while text is revealing. The first press finishes revealing the current sentence. The next press advances the conversation. This lets a fast reader finish a line without accidentally skipping the answer.

> Typewriter settings on a Choice option affect the selected response when it is presented as speech. They do not turn every waiting menu button into a letter-by-letter animation.

42. Assign a default portrait and a Line override

1. Open the speaker in Character Designer → Portraits.
2. Create or select a portrait entry. Give it an ID and a display name.
3. Choose its Source Type and the required assets.
4. Mark it as the default dialogue portrait when it should be used normally.
5. Apply the portrait changes and save the project.
6. Return to the Line. Leave Portrait at Default Portrait, or choose a specific portrait for that moment.

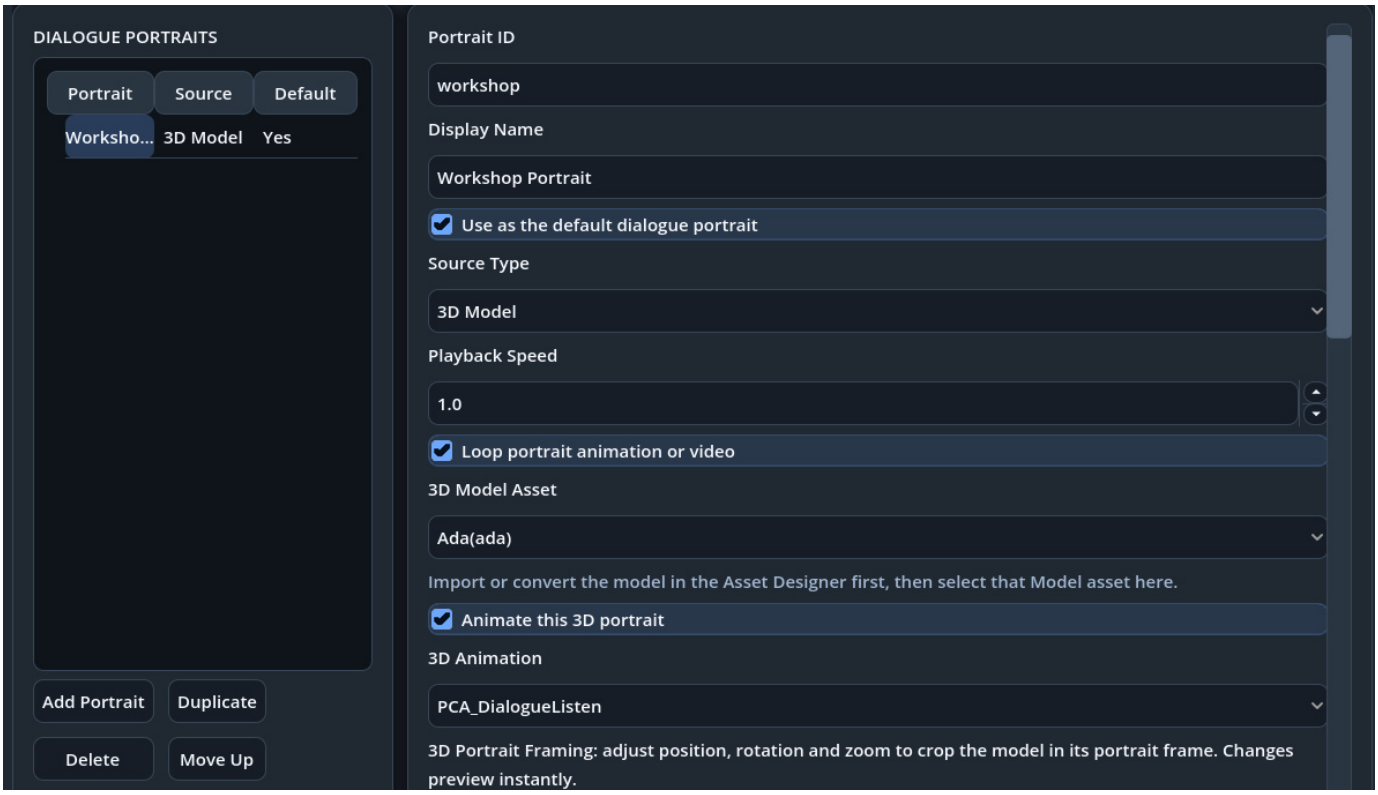

Figure 29. Portrait records belong to the character. A Line chooses among that speaker’s portraits.

A default portrait saves work across many Lines. An override is useful for a different expression, costume or camera framing. Selecting an override on one Line does not redefine the character’s default portrait.

> Set Speaker before choosing Portrait. If a requested portrait ID no longer exists, a fallback portrait may appear, which can hide a broken reference until you inspect it.

43. Choose the right portrait source

Source	Prepare / check
Still Image	An imported image asset. Good for a painted talking head.
Animated Image Frames	The intended sequence of image assets, playback speed and looping.
Video	A supported imported video and sensible loop/playback settings.
3D Model	The model asset, optional animation and portrait framing.
Live2D Cubism	The model and the required Live2D runtime support for your target build.

For a 3D portrait, choose the model and a suitable idle/listening animation. Adjust Move X/Y/Z, Rotate X/Y/Z and Scale/Zoom until the face is readable. Validate and Apply Portrait, then check it in the real dialogue window.

Model portrait movement is relative to the model’s size. Treat it as portrait framing, rather than copying room-world coordinates into those fields. Check both a narrow portrait slot and your final screen aspect ratio.

The source animation and speech mouth movement are separate. A listening animation can supply posture while lip sync drives the mouth. A general full-body animation is not automatically a mouth map.

> A spritesheet is not automatically the same as an Animated Image Frames list. Prepare the actual assets that the chosen portrait source expects. Avoid putting the same voice recording on both the Line and a looping portrait source.

44. Make scene characters and portraits lip-sync

1. Open Character Designer → Lip Sync and enable lip sync.
2. Choose the speech timing mode appropriate to your recording and prepared track.
3. Set Lip-sync Targets to Both if the scene character and portrait should speak together.
4. Configure the Scene Character mouth mapping.
5. Configure the Portrait mapping separately for the portrait you use.
6. Test a voiced Line while both representations are visible.

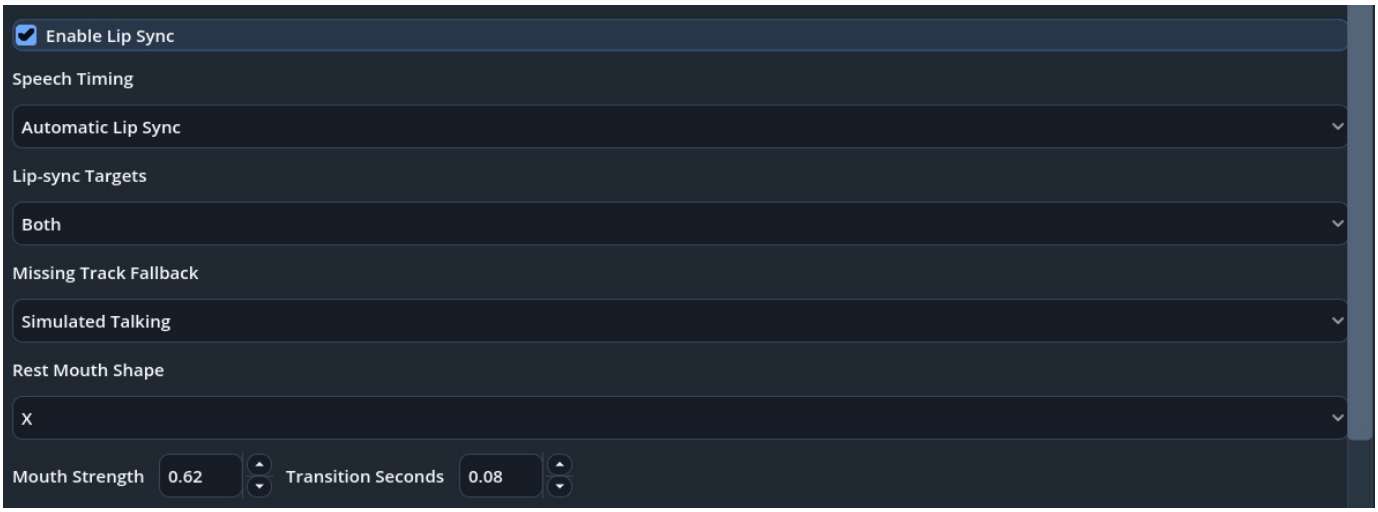

Figure 30. The same voice timing can drive two targets, but each target needs a mouth mapping that matches its own art.

Timing approach	What it provides
Automatic / prepared track	Uses prepared mouth cues when available, with the selected fallback when needed.
Basic Mouth Movement	Uses recording volume for opening and closing; it does not identify phonemes.
Manual / Imported	Uses an authored or imported cue track.
Simulated Talking	Uses a text-driven approximation rather than matching a recorded performance.

A timing track says when the mouth changes. A mapping says which blend shape, bone, frame or other mouth target to use. A correct track with no matching target can play sound while the face stays still.

45. Prepare and correct the voice cue track

1. Select the audio asset in the Asset Designer and open Lip Sync.
2. Supply the matching transcript when useful. Choose the recording language in the advanced settings.
3. Generate Lip Sync, use Basic Mouth Movement, or Import Track according to the material you have.
4. Choose the character and enable Portrait Preview when checking the portrait.
5. Play the recording. Inspect the mouth targets and correct obvious cue boundaries or shapes.
6. Use Save Track, then save the project.

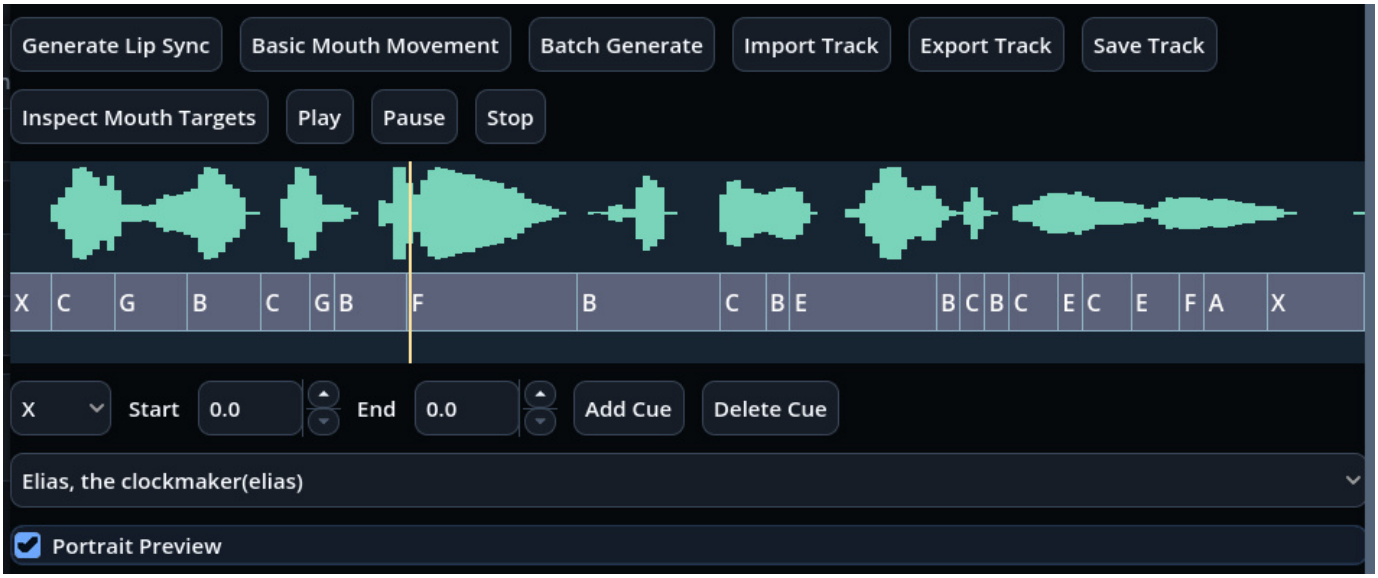

Figure 31. The waveform and cue timeline describe this recording. Preview controls help check the selected character or portrait.

The cue editor includes shape, start and end fields, Add Cue and Delete Cue, plus track import/export. Use small corrections around words that look wrong rather than shifting the whole track because of one awkward syllable.

Prepare a separate track for each translated recording. A French performance has different timing from its English source, even when both express the same idea.

> Rhubarb is an authoring tool used to prepare cues. Players do not need to run the recognizer to play a game with saved tracks. A missing recognizer during authoring is different from a missing mouth mapping at runtime.

46. Replace the spoken-line window

The Dialogue Designer owns the words and flow. The GUI Designer can own the presentation. Create a separate GUI for spoken lines, then assign it through Global Settings → System GUIs.

System GUI role	Used for
Dialogue	Spoken Lines inside a dialogue graph.
Speech Bubble	Standalone Say-style speech outside the graph.
Dialogue Choices	The response menu, separate from the spoken-line window.



Figure 32. The text element’s Group ID connects it to dialogue output. Its ordinary element ID can be anything sensible.

Element / binding	Purpose
Text label: @talkdialogue	Receives the active spoken sentence and supports dialogue continuation.
Name label: $$_dialogue_speaker	Shows the current speaker through a text wildcard.
Image: @playerpic	Receives the player portrait.
Image: @NPCpic	Receives the non-player portrait; preserve the capitalization.

A Group ID is a binding, not an element ID. Start with Cut transitions so visibility and input are easy to test. Add animation after the basic bindings work.

47. Build a response menu with paging

1. Create a GUI for choices. Add a panel and three readable Buttons.
2. Set their Group IDs to @choice1, @choice2 and @choice3, in that order.
3. Add paging controls with Group IDs @choicescrollup and @choicescrolldown.
4. Assign this GUI to the Dialogue Choices system role.
5. Test a conversation with four visible responses, including a read/disabled response.

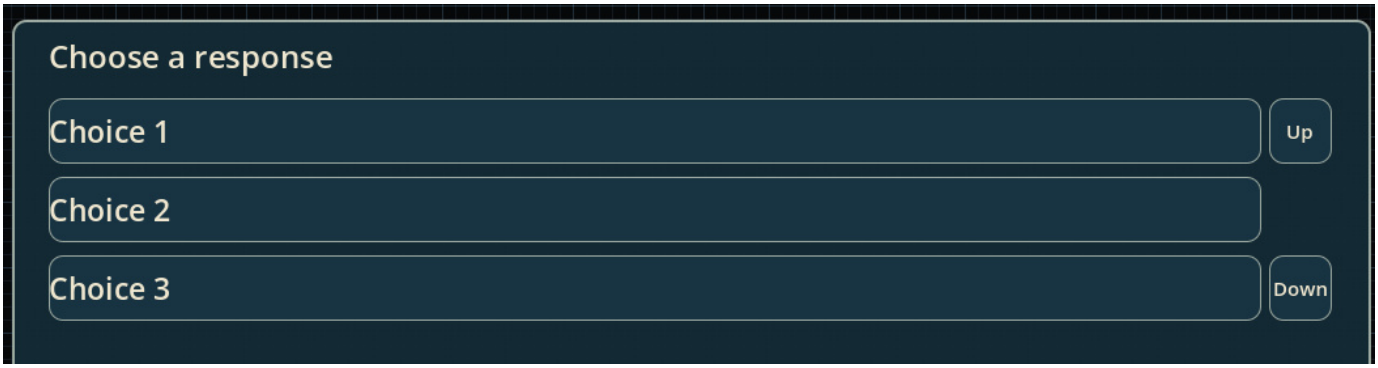

Figure 33. The example has three response slots and two paging buttons. Four visible responses therefore require paging.

Choice groups are numbered slots, not permanent topic IDs. The runtime fills them from the currently visible responses and updates them when you scroll. Keep the group numbers contiguous; the supported bindings run from @choice1 to @choice20.

Do not add a second Start Dialogue or branch action to each choice Button. Its choice group is already the selection binding. A second Click action can start unwanted extra work.

> Built-in numbering/bullets and the built-in timer label are not automatically new elements in your custom GUI. Check your own labels. A custom countdown display needs its own deliberate implementation rather than an assumed timer Group ID.

48. Test custom windows and personal overrides

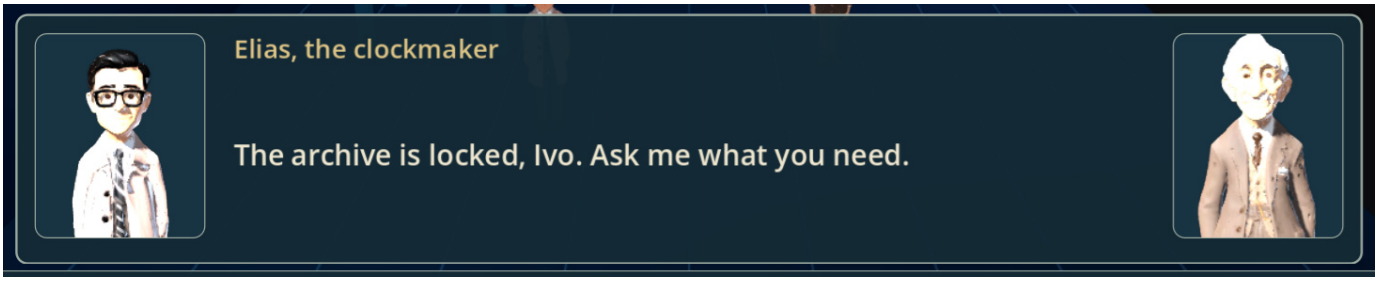

Figure 34. The custom speech window binds the speaker, sentence and both portrait slots.

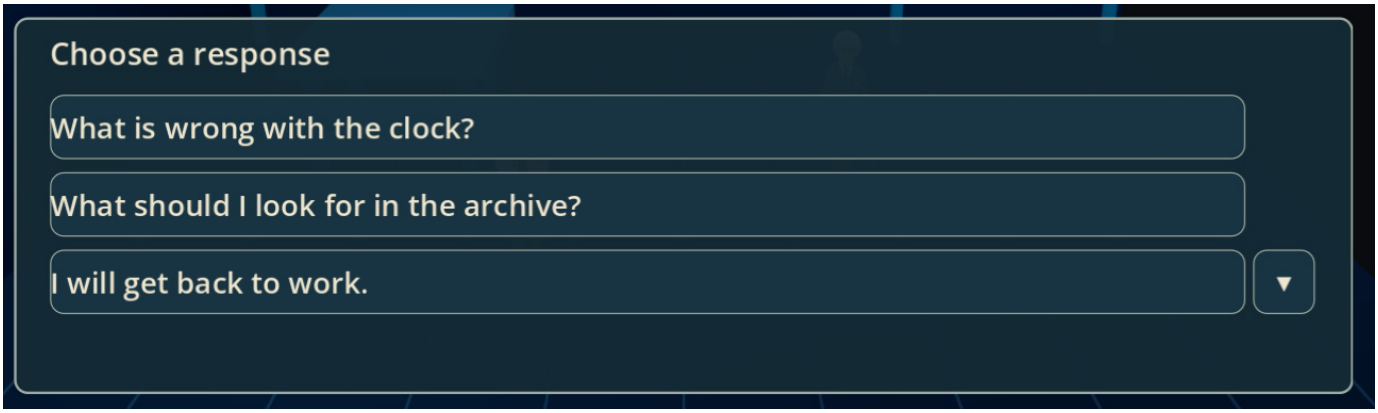

Figure 35. The response window replaces the speech window when a menu is waiting.

Test clicking a response, returning to the menu and paging to a later response. Check that a disabled response cannot be clicked and that the spoken-line GUI closes when the choice menu appears.

A personal Dialogue GUI on the active player can override the global spoken-line presentation. It follows the currently controlled character’s interface choice, even when the person speaking is Elias.

Unexpected result	Check
Global speech GUI seems ignored	The active player’s personal Dialogue GUI override.
NPC portrait does not appear	The exact @NPCpic group and the NPC portrait setup.
Only three of four choices appear	Your slot count and working paging controls.
Global style does not change the authored panel	The custom GUI owns its own appearance.

> Dialogue Choices remains a separately assigned global role in this version. Do not assume the active player’s personal spoken-line GUI also replaces the response menu.

49. Translate dialogue and keep voices aligned

1. Select a spoken Line and choose Translate to open its text in the translation editor.
2. Choose the target language and enter the translation.
3. For Choice text, search the actual response wording in the translation editor; selecting a whole Choice node does not identify one specific option.
4. Export a language CSV when another person will translate it.
5. Import the completed CSV, review the preview, save translations and save the project.
6. Test the actual conversation in the target subtitle and voice languages.

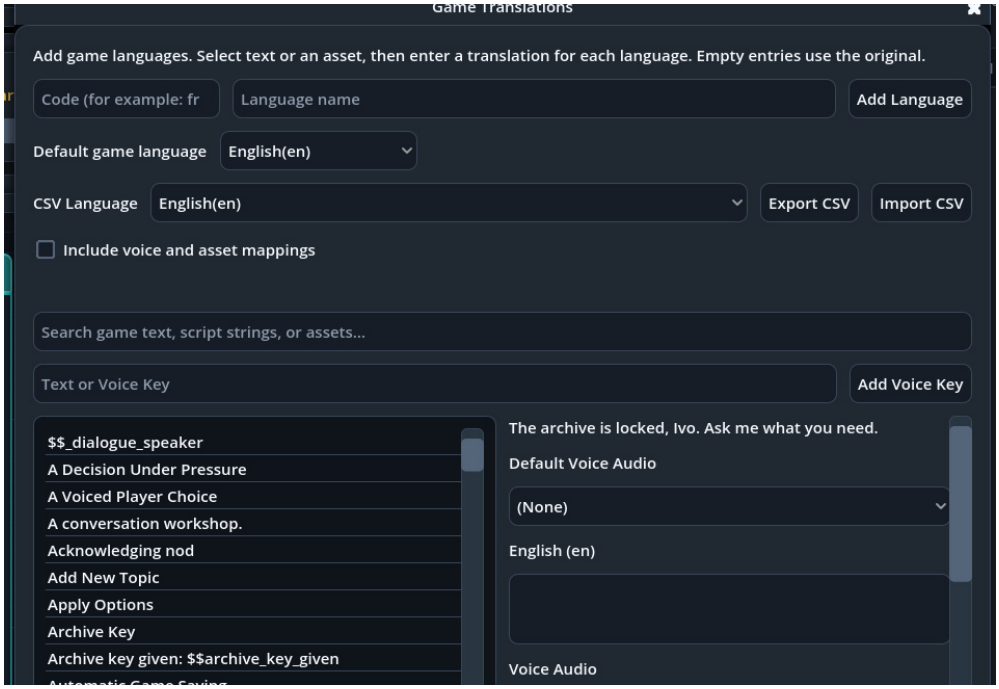

Figure 36. Translate opens the game’s localization tools. Source wording identifies the text entry.

Enable voice and asset mappings in the CSV workflow when those replacements belong in the handoff. Keep audio assets and their cue tracks organized by language; translated subtitles alone do not replace the recorded voice.

If you rewrite source wording after translation, revisit the matching entry. A translation attached to an old source sentence may no longer be found for its new wording.

> Test long translations in both spoken and choice GUIs. A line that fits in English may wrap twice as much in another language. Keep wildcard tokens intact so names and values still expand.

50. Set the style without changing the dialogue

Global dialogue presentation offers portrait, classic, speech-bubble, subtitle and cinematic styles. Choose the one that suits the game, then test a spoken Line and a response menu in the actual Play view.

Begin with the built-in presentation while connecting the conversation. Once the graph is reliable, switch to a custom GUI or style and test again. This makes it much easier to tell a broken branch from a hidden text label.

Check	Reason
Text size and contrast	The player must read it against bright and dark rooms.
Portrait framing	A full-body model may be tiny, cropped or distracting in a talking-head slot.
Choice button height	Long and translated responses need room to wrap.
Typewriter speed and blips	They should suit the sentence and avoid fighting the recorded voice.
Keyboard / controller focus	Every available choice and exit should be reachable.
Screen ratio and UI scale	A comfortable desktop layout may clip on a narrow display.

Use the GUI Designer for panel colors, fonts, border radii, state images, spacing and transitions. The dialogue record should keep the story references and flow; its wording does not need to change merely because the panel has a new border.

> If you assign an authored dialogue GUI, inspect its own style settings. Global built-in style selection does not redesign every authored element for you.

51. Add a new topic while the game is running

Dialogue-editing actions can add a clue topic after a discovery. Add the answer first, then add the response that points to it. Both IDs must be new in the intended scope.

```
await AddDialogueLine({
    "dialogueId":"archive_chat",
    "nodeId":"lantern_reply",
    "speakerId":"elias",
    "text":"The lantern is by the workshop door.",
    "returnToPrevious":true
})
await AddDialogueChoice({
    "dialogueId":"archive_chat", "nodeId":"topics",
    "choiceId":"ask_lantern",
    "text":"Where is the lantern?",
    "next":"lantern_reply"
})
```

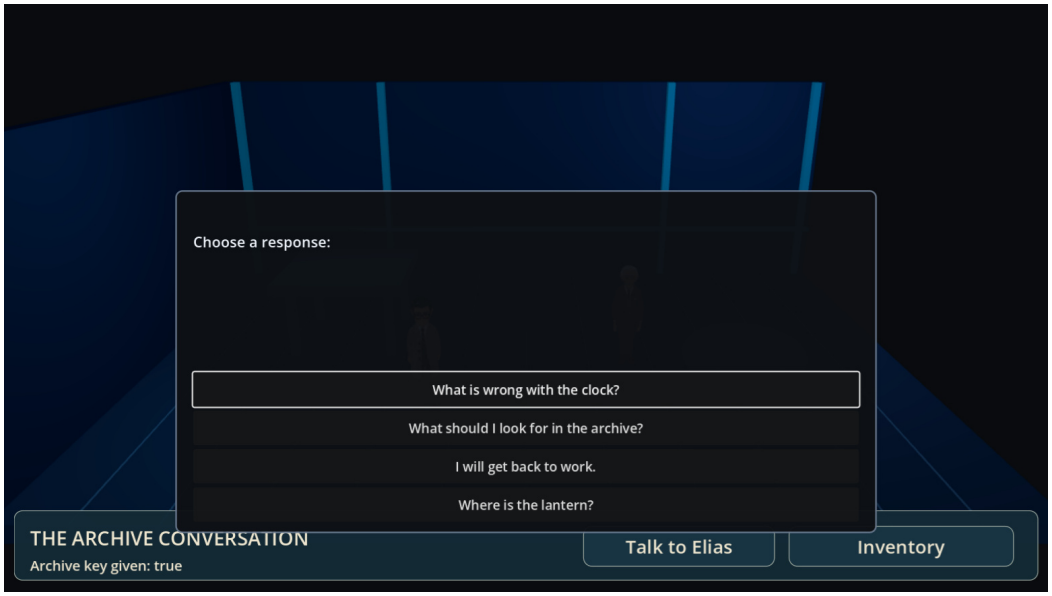

Figure 37. The new lantern question appears in the running conversation.

The same actions are available visually. Add Dialogue Choice Node creates a menu node; it does not automatically invent its options or connect that menu into the conversation.

> These changes affect the running game’s project data. They do not silently rewrite your authored editor file. Save/load can preserve the runtime changes, so test both a fresh game and a loaded save.

52. Remove or update runtime topics deliberately

Action	Targets / effect
Add Dialogue Line	New Line node: dialogue ID, new node ID, speaker, text and flow settings.
Add Dialogue Choice Node	New menu node; add its options and connect it before relying on it.
Add Dialogue Choice	New response in a specified Choice node.
Remove Dialogue Choice	One response by dialogue, node and Choice ID.
Enable / Disable Dialogue Choice	Its explicit Enabled flag; other availability rules still apply.
Remove Dialogue Node	Removes a runtime node and uses the runtime bypass rules for incoming references.
Reset Dialogue History	Clears read status at the requested scope.

Adding an existing ID is not an overwrite command. Check your story flag before adding the same runtime topic twice. A failed or missing reference should be corrected at the source rather than hidden by repeatedly calling Add.

Removing a runtime node is different from deleting a node in the editor. Incoming links can be redirected through its NEXT target, or through the automatic-return target. Removing a starting node that returns automatically can end the dialogue rather than create a new useful start.

For a topic that may return later, disabling or conditioning the response is often clearer than deleting its answer node. Remove nodes only when you have checked every incoming branch.

> Test runtime edits while a menu is open, after closing it and after loading a save. The runtime refreshes open choices after its dialogue-editing actions, but that does not repair a bad destination.

53. Use dialogue events and quest completion

Scripts and visual scripts can listen for OnDialogueStarted and OnDialogueEnded. Register the event in the script’s event settings, then use the event’s dialogueId to decide which conversation it concerns.

Event data	Use
dialogueId	Identifies the conversation that started or ended.
playerId	Identifies the active player at the event.
interrupted, on the ended event	Distinguishes deliberate cancellation from a normal completion.
Caller context	Can carry additional information from the action or script that started the dialogue.

A normal dialogue completion also notifies the quest system. An interrupted session does not count as that normal completion. If a quest only needs a particular clue, recording a variable on that branch is more precise than assuming any ending means the clue was heard.

Start with simple logging or a harmless counter when testing an event. After verifying which dialogue and completion route fired it, add the intended game change.

A global event handler can observe several conversations, including nested ones. Filter by ID. Otherwise one side exchange may trigger a reward intended for the main conversation.

> Do not restart a dialogue from its own started event, or call the same closing sequence recursively from its ended event. An event is a hook, not an instruction to repeat the event that called it.

54. Add camera and character direction

Use Action nodes or a short script to stage the conversation: turn the characters toward each other, choose a camera, play a gesture and then begin the next spoken Line.

Before speaking	After the exchange
Select the intended conversation camera.	Restore the normal gameplay camera.
Face the other character or a look target.	Release a temporary look target when appropriate.
Play a suitable gesture or talking animation.	Return to the intended idle state.
Apply a temporary presentation change.	Remove it on both normal and interrupted exits.

Keep each directed sequence explicit. For example: camera action → character-facing action → Line → reply → restore camera → End. Turn Return to previous node off along that forward sequence.

The dialogue speaker binding handles who is speaking. It does not automatically choose your best close-up, move a character to a mark or frame two people without obstruction. Those are scene and camera decisions.

Use a cutscene when the exchange needs coordinated movement, several camera shots and timed gestures. A simple topic menu usually needs less staging and should remain quick to reopen.

> Test cancellation as well as Goodbye. A restored camera on one normal exit is not enough if an interruption can leave the player looking at the wrong part of the room.

55. Write choices that make sense to the player

A response is a promise about what selecting it will do. “May I borrow the archive key?” should request the key. If it secretly insults Elias or spends money, give the player enough wording to understand that consequence.

Instead of…	Try…
A vague “Tell me more” on every topic	Name the subject: What happened to the brass wheel?
Five answers that all reveal the same clue	Use fewer responses with a useful distinction.
Hiding the only route out after it is read	Keep an available, repeatable Goodbye.
A long speech on a response button	Keep the choice concise; put the explanation in a Line.
Essential information only in audio	Keep readable subtitles or another accessible way to recover the clue.

Use one idea per short Line where practical. Break a long explanation at a natural thought, not simply where the text field becomes inconvenient. Repeated menus are easier to scan when the response wording stays stable.

Give a failed request a useful reply. “No” can become “I cannot lend it while the front door is open,” which explains what the player could change. Use a flag to prevent an important reward from being handed out twice.

> Read the complete exchange aloud. If you cannot comfortably say the line in one breath, consider shortening or splitting it before recording the voice.

56. Run a complete conversation test

Test	Expected result
Fresh game, greeting	Correct speaker; choices appear after Continue.
Ask clock, then return	Answer appears; menu returns; read rule is visible.
Borrow key	Exactly one key and a true loan flag.
Return after loan	Key request hidden; archive question available.
Goodbye, then talk again	Clean exit; conversation can reopen.
Save after closing; load	Inventory, loan flag and read history retained.
Reset history	Read state clears without resetting the loan or explicit Enabled flag.
Timed menu, no input	The chosen timeout policy runs once.
Unavailable timed default	Only a selectable fallback is used.
Nested exchange	Normal End resumes the parent; interruption cancels it.
Voice and typewriter	Right recording; first Continue reveals, next advances.
Custom GUI, four choices	Correct text, disabled state and working paging.
Second language	Translated text, expected recording and readable wrapping.

Run the actual NPC Talk interaction as well as Test Dialogue. Repeat the final pass in a distributable build with its packaged assets, rather than relying only on the editor preview.

The example’s branch, reward, history and save/load behavior was checked in the running editor while preparing this guide. Your own scripts, interfaces and content still need their own end-to-end pass.

57. Fix a conversation that closes or loops

Symptom	First checks
Nothing appears when testing	The selected dialogue’s start still points to its original End node.
Greeting appears, then closes	Greeting automatic return is on with no caller, or NEXT is empty.
An answer returns to the wrong step	Automatic return goes to its immediate caller. Use explicit flow for a multi-node sequence.
Reward runs repeatedly	A loop returns to the reward Action; add a condition/guard and inspect NEXT.
Menu ends unexpectedly	Every response is hidden/disabled, or a selected response has no destination.
Only the last option disappeared	Remove Choice Option removes the last authored response.
The wrong conversation opens in Play	The NPC/GUI action still points to another dialogue ID.
A child ends the whole exchange	An End Dialogue action was used instead of a normal End node.
A Line advances while you are inspecting	The idle-Line watchdog can advance it after 30 seconds.

Reduce the failure to one greeting, one menu, one answer and one End. Confirm that route, then restore actions and conditions one at a time. Inspect the real variable values rather than inferring them from the wording of a reply.

> Before changing a working graph, save a copy. Reconnecting many branches at once can hide the original error beneath several new ones.

58. Fix text, voices, portraits and custom GUIs

Symptom	First checks
Changed text did not stick	Press Enter or leave the Line text field, then Save Project.
Response remains unavailable after reset	Explicit Enabled, its condition, and the configured read mode.
No voice	Correct audio asset/key, voice language, volume and packaged source file.
Wrong translated voice	Direct asset takes its localized asset route; check the route rather than only the Voice Key.
Sound but no mouth movement	Saved track, enabled lip sync, target selection and the actual mouth mapping.
Scene mouth moves; portrait does not	Portrait mapping and Lip-sync Targets; a scene mapping alone is not enough.
Wrong portrait or expression	Line Speaker, Portrait override and that character’s default portrait.
Custom speech is empty	Correct System GUI role, @talkdialogue and the active player’s personal override.
Choice click does nothing	Exact contiguous @choiceN groups, active role and available response.
Later choices are missing	Choice slot count plus @choicescrollup / @choicescrolldown bindings.
Translation missing after rewrite	The new source text and its translation entry.

Check the simplest built-in presentation as a comparison when an authored GUI is failing. If the same graph works there, inspect the GUI’s bindings, visibility, layout and transitions.

59. Make the finished conversation ready to ship

1. Save the authored project and run its validation tools. Resolve missing characters, item IDs, assets and destinations.
2. Test the normal start and every response with a fresh game.
3. Test later story states, repeated visits and previously chosen responses.
4. Test saving and loading with the conversation closed.
5. Check every assigned recording, portrait and lip-sync track.
6. Check custom GUI input, paging, text wrapping and screen sizes.
7. Check translated subtitles and voice language combinations.
8. Build the game and repeat the important routes from the packaged version.

Keep a small conversation checklist with your release notes: dialogue IDs changed, Choice IDs changed, new variables, new recordings and any save compatibility decisions. Stable references matter more than the visual position of nodes.

For a final writing pass, remove temporary debug lines, repeated explanations and choices that do nothing useful. Keep the exit obvious. Let a new reader play the conversation without explaining the puzzle to them first.

> The appendices provide the exact example record, short script recipes and compact field references. They are reading references, not a separate set of practice files.

Appendix A. The complete archive dialogue

The following source blocks form one complete dialogue record. They are useful if you want the exact IDs used in the guide. The graph editor can build the same conversation with its generated node IDs.

1. Create or select the dialogue you intend to replace. Save a project copy first.
2. Confirm that elias is your NPC, archive_key is an inventory item and archive_key_given is a Boolean variable starting false. Substitute your own IDs if needed.
3. Open the selected dialogue’s Inspector → Raw JSON (Advanced). Replace that record only with all of the following blocks in order.
4. Apply the JSON changes and inspect the graph. The start must be greeting.
5. Save the project and run Test Dialogue, then test the actual NPC Talk action.

Copy the code, including the opening and final braces. Do not copy the page headings or the notes below each block. Keep the commas and JSON Boolean values exactly as shown.

> This replaces one dialogue record, not the entire project document. It does not create the character, inventory item, variable, voice files or GUI records for you.

A1. Archive record: lines 1-44

Continue the same JSON object. These blocks are consecutive; they are not separate records.

```
{
  "id": "archive_chat",
  "name": "Elias - The Locked Archive",
  "start": "greeting",
  "nodes": {
    "greeting": {
      "id": "greeting",
      "type": "line",
      "speakerId": "elias",
      "portraitId": "",
      "text": "The archive is locked, Ivo. Ask me what you need.",
      "returnToPrevious": false,
      "next": "topics",
      "x": 40,
      "y": 40
    },
    "topics": {
      "id": "topics",
      "type": "choice",
      "timeoutSeconds": 0,
      "timeoutAction": "end",
      "timeoutChoiceIndex": 0,
      "choiceListStyle": "plain",
      "options": [
        {
          "id": "ask_clock",
          "text": "What is wrong with the clock?",
          "next": "clock_reply",
          "enabled": true,
          "repeatable": false
        },
        {
          "id": "ask_key",
          "text": "May I borrow the archive key?",
          "next": "give_key",
          "enabled": true,
          "repeatable": false,
          "condition": {
            "variable": "archive_key_given",
            "equals": false
          }
        },
        {
          "id": "ask_archive",
```

Continue directly with the next block.

A2. Archive record: lines 45-88

Continue the same JSON object. These blocks are consecutive; they are not separate records.

```
          "text": "What should I look for in the archive?",
          "next": "archive_reply",
          "enabled": true,
          "repeatable": false,
          "condition": {
            "variable": "archive_key_given",
            "equals": true
          }
        },
        {
          "id": "goodbye",
          "text": "I will get back to work.",
          "next": "end",
          "enabled": true,
          "repeatable": true
        }
      ],
      "x": 520,
      "y": 40
    },
    "clock_reply": {
      "id": "clock_reply",
      "type": "line",
      "speakerId": "elias",
      "portraitId": "",
      "text": "The brass wheel is missing. We need it before the bell will ring.",
      "returnToPrevious": true,
      "next": "",
      "x": 1520,
      "y": 40
    },
    "give_key": {
      "id": "give_key",
      "type": "action",
      "action": {
        "type": "AddInventory",
        "itemId": "archive_key",
        "count": 1
      },
      "next": "mark_key",
      "returnToPrevious": false,
      "x": 1000,
      "y": 700
    },
```

Continue directly with the next block.

A3. Archive record: lines 89-131

Continue the same JSON object. These blocks are consecutive; they are not separate records.

```
    "mark_key": {
      "id": "mark_key",
      "type": "action",
      "action": {
        "type": "SetVariable",
        "name": "archive_key_given",
        "value": true
      },
      "next": "key_reply",
      "returnToPrevious": false,
      "x": 1000,
      "y": 1210
    },
    "key_reply": {
      "id": "key_reply",
      "type": "line",
      "speakerId": "elias",
      "portraitId": "",
      "text": "Here you are. Bring it back when you have the wheel.",
      "returnToPrevious": false,
      "next": "topics",
      "x": 1000,
      "y": 1730
    },
    "archive_reply": {
      "id": "archive_reply",
      "type": "line",
      "speakerId": "elias",
      "portraitId": "",
      "text": "Look for a brass wheel in the cabinet beside the window.",
      "returnToPrevious": true,
      "next": "",
      "x": 1520,
      "y": 760
    },
    "end": {
      "id": "end",
      "type": "end",
      "x": 1520,
      "y": 1490
    }
  }
}
```

End of the complete dialogue record. Apply, save and test.

Appendix B. Script recipes

Each recipe belongs in its own project GDScript record. The record must be enabled. Run it through a Run Script action, or expose a named function when a dialogue Action should call that function directly.

These examples are deliberately small. They use the same shared actions as the designer. Do not mark every helper Global or subscribe it to events unless you want it to run automatically.

Recipe	Use
open_archive	Start the conversation and wait for it to finish.
guarded_talk	Set the explicit availability flag before opening.
guarded_give	Issue a one-time key reward through an exposed function.
add_new_topic	Add a new answer and a connected menu response at runtime.
remove_topic	Remove that particular response later.
reset_topics	Clear this conversation’s read history.

Keep script IDs distinct from dialogue IDs. Running the open_archive script and starting the archive_chat dialogue are two different references.

B1-B2. Start and guard a conversation

Use the first recipe for a direct start. The second sets explicit availability from the saved loan flag before starting.

A direct start

```
extends PCAScriptBase

func run() -> void:
    await StartDialogue({"dialogueId":"archive_chat"})
```

Check the loan first

```
extends PCAScriptBase

func run() -> void:
    var already_given: bool = bool(
        Runtime.variables.get("archive_key_given", false))
    await SetDialogueChoiceEnabled({
        "dialogueId":"archive_chat", "nodeId":"topics",
        "choiceId":"ask_key", "enabled":not already_given
    })
    await StartDialogue({"dialogueId":"archive_chat"})
```

The original option condition still applies. Disabling a response and hiding it with a condition are separate rules.

B3. Grant the key once

Expose grant_key as Give Archive Key Once, with no parameters. Call it from the reward branch instead of also keeping the old unconditional Add Inventory Action.

```
extends PCAScriptBase

func grant_key() -> void:
    if bool(Runtime.variables.get("archive_key_given", false)):
        return
    if not HasInventory("archive_key"):
        await AddInventory({"itemId":"archive_key", "count":1})
    await SetVariable({"name":"archive_key_given", "value":true})
```

Use your actual dialogue, node, Choice, item and variable IDs. Save the authored script before testing it.

B4. Add an answer and a question

Run once after the story discovery. Reusing existing IDs is not an overwrite operation.

```
extends PCAScriptBase

func run() -> void:
    await AddDialogueLine({
        "dialogueId":"archive_chat", "nodeId":"lantern_reply",
        "speakerId":"elias",
        "text":"The lantern is by the workshop door.",
        "returnToPrevious":true
    })
    await AddDialogueChoice({
        "dialogueId":"archive_chat", "nodeId":"topics",
        "choiceId":"ask_lantern",
        "text":"Where is the lantern?", "next":"lantern_reply"
    })
```

Use your actual dialogue, node, Choice, item and variable IDs. Save the authored script before testing it.

B5. Remove a topic and reset history

Removing a response does not require deleting its answer. Resetting history affects read status, not story flags.

Remove the lantern response

```
extends PCAScriptBase

func run() -> void:
    await RemoveDialogueChoice({
        "dialogueId":"archive_chat", "nodeId":"topics",
        "choiceId":"ask_lantern"
    })
```

Clear this conversation’s read history

```
extends PCAScriptBase

func run() -> void:
    await ResetDialogueHistory({"dialogueId":"archive_chat"})
```

For a narrower history reset, include nodeId and optionally choiceId. Leaving dialogueId empty resets all dialogue history.

Appendix C. Line and Choice field reference

Field	Check
Line: Speaker	Special player alias or fixed character.
Line: Portrait	Default or an ID owned by that speaker.
Line: Text	Commit with Enter or by leaving the field.
Voice Audio / Voice Key	Direct recording or localized lookup route.
Typewriter	On/off, speed and Default/None/Custom sound.
Return to previous node	Immediate caller; off for forward sequences and a starting greeting.
Choice labels	Plain, bulleted or numbered in the built-in presentation.
Time limit	0 disables the timer; otherwise seconds for the menu.
When time expires	End conversation, random or preselected response.
Preselected response	Authored option number; stored index starts at zero.
Choice ID	Unique within its node and stable for history/edit actions.
Enabled	Explicit availability switch.
Always Allow Repeat	Bypasses read-history disabling only.
Option voice / typewriter	Applies to the chosen player response when presented as speech.
Option condition, in raw record	Simple variable/equals pair; false conditions omit the response.

Appendix D. A quick diagnosis by reference

Reference	Example	Used by
Dialogue	archive_chat	Start Dialogue; read/reset and edit actions.
Node	topics	Choice targeting and graph destinations.
Choice	ask_key	One response’s history and availability.
Character	elias	Speaker and portrait owner.
Inventory item	archive_key	Reward and inventory checks.
Variable	archive_key_given	Story-state condition and reward guard.
Audio asset	voice_elias_01	Recording and prepared mouth cues.
Portrait	workshop	An entry belonging to the selected speaker.
Speech GUI	speech_custom	Dialogue / Speech Bubble presentation.
Choice GUI	choices_custom	Response slots and paging.

When an action appears to do nothing, check the reference type before changing the logic. A character display name in a field that expects an ID is not the same reference. A choice button’s @choice1 group is a display slot, not the ask_key story identity.

> The GUI binding spellings are @talkdialogue, @playerpic, @NPCpic, @choice1 through @choice20, @choicescrollup and @choicescrolldown. Preserve their spelling and capitalization.

Appendix E. Put names and values in a sentence

Dialogue text and response labels can include wildcards. The runtime replaces them when it presents the text, so one authored sentence can refer to the current player or a changing story value.

Token	Replacement
@PlayerName	The active player’s display name.
@PlayerId	The active player’s record ID; mostly useful for debugging.
@SceneName	The current scene’s name.
@Score	The score variable, with zero as its fallback.
$$trust	The current value of the variable named trust.
$$_dialogue_speaker	The current speech name; useful on a custom GUI name label.

For example, enter “Welcome back, @PlayerName.” in a Line. With Ivo active, the displayed greeting includes Professor Ivo. Change the player and the same authored sentence follows the new active player’s name.

Define your own variable before using it. A Line such as “Your trust score is $$trust.” displays the current value; it does not add to that value. A Number may display as 0.0. For special formatting, prepare a separate display string in a script. Use an Action to change trust when the story calls for it.

Unknown variable tokens remain visible, which is useful for catching a spelling error. If $$truts appears in Play, check the variable’s exact ID rather than editing its display name.

> Preserve tokens in translated text. Recorded speech is not assembled from variable values. If a voice must say a changing name or number, plan separate recordings/Lines or wording that avoids speaking the changing value.

E2. Vary a repeated greeting

For small variations, a text field can contain alternatives separated by a special marker. This is useful for short, unvoiced greetings or incidental comments.

Marker	Behavior	Example
@||@	Cycles through alternatives in order on successive calls.	Hello again.@||@Back so soon?
@//@	Chooses an alternative randomly on each call.	A cold day.@//@Looks like rain.

1. Put Hello again.@||@Back so soon? in a short Line.
2. Test that Line, end the conversation and start it again in the same running game.
3. Check that the displayed alternative changes, then cycles back.
4. Try the random marker separately. A random selection is allowed to repeat the same sentence.

Text alternatives change the wording. They do not replace story conditions, prevent a reward from repeating or create separate response histories. Use variables and real branches when the game state matters.

For recorded performances, separate Line nodes with the appropriate recording are easier to manage than assuming the text-alternative marker also selects a matching voice file. Keep the shown words and audible performance aligned.

> Keep the marker intact during translation and use one alternative style in a field. Test sequential wording within one running session; a fresh Test Dialogue preview starts fresh test state.

Version, illustrations and verification

This guide was prepared against PCA 3D Adventure Studio 0.86.3.4. The editor controls, dialogue graph, action catalog, runtime flow, speech/portrait bindings and localization behavior were checked against that version.

Illustrations show the real native editor and its Play view. The workshop is a simple room used to keep the interface examples clear. Professor Ivo, Elias and the example recorded performances come from the bundled Clockmaker content.

The working checks covered branching, hidden and disabled responses, inventory changes, history reset, save/load, runtime topic edits, timed choices and nested returns. Presentation checks covered a prepared voice track mapped to scene and portrait, two-stage typewriter continuation, custom choice clicks/paging and translated text. Test the same links again with your own content and interface.

No engine source changes are required for the tutorial. The code in the appendices is project-level example code. No separate practice files accompany this PDF.

For a later PCA release, compare the field labels and action parameters before copying an advanced recipe unchanged. Keep a saved copy of your project when changing IDs or editing a full record as JSON.