PCA 3D ADVENTURE STUDIO

Eight Adventure Interface Workshops

Build the HUD, inventory and save screens. Make their animation and input work together.



Figure 1. Four of the working PCA layouts built in this guide.

Zak McKracken · Day of the Tentacle · Sam & Max Hit the Road
Beneath a Steel Sky · Monkey Island 2 · The Curse of Monkey Island
Resonance · Gemini Rue

Reference build 0.86.3.4 | Windows, macOS and Linux

Before you begin

This book is a set of build tutorials for PCA 3D Adventure Studio. Each workshop explains the visible interface, the inventory, the save screen and the animation choices. Work through the shared foundations once, then follow the chapter for the design you want.

The examples use the original desktop interface families as the reference. The Beneath chapter covers the 1994 game, as requested. Monkey Island 2 means the original word-bank layout rather than its later Special Edition alternative.

Every editor and Play screenshot was captured from PCA. The workshop uses sample art and readable placeholder captions, not copied portraits, backgrounds or menu graphics from the reference games. The save thumbnails are real engine captures of the test scene.

Read first	Then choose
Chapters 1-2	Layout, events, inventory ownership and safe save-menu construction.
Chapter 3	Transitions, image frames, FPS, state feedback and one-shot animation.
Chapters 4-11	Eight separate interface workshops, in the order listed on the cover.
Chapter 12	Complete scripts for the details that need a small controller or adapter.
Chapters 13-14	Tests, troubleshooting and the historical design references.

Use your own project and assets. No practice project or companion download is required. The engine itself is not changed by following the guide.

Contents

Click a chapter title, or use the PDF bookmarks for the continued pages.

1. Choose one interface to build 4

2. Build a dependable inventory 11

3. Animate the right thing 18

4. Zak McKracken: text commands 26

5. Day of the Tentacle: verbs and faces 34

6. Sam & Max: let the cursor carry the action 42

7. Beneath a Steel Sky: a hidden top strip 50

8. Monkey Island 2: the eight-icon verb HUD 58

9. Curse: the action coin, chest and logbook 66

10. Resonance: items and memories 77

11. Gemini Rue: actions around a target 87

12. Small scripts for the missing details 96

13. Test the interface as a player 111

14. Reference notes and design sources 116

1. Choose one interface to build

Each workshop builds a playable PCA interface around a familiar design. Start with one chapter. You do not need to build all eight interfaces into one game, and you should not leave eight HUDs visible together.

Reference	Main idea	Inventory
Zak McKracken	Text verbs and a command line	Four text entries below the verbs
Day of the Tentacle	Nine verbs and character portraits	Six icons beside the verbs
Sam & Max Hit the Road	Actions carried by the cursor	A floating inventory box
Beneath a Steel Sky	Two-button interaction	An inventory strip at the top
Monkey Island 2	Nine verbs and a sentence line	Eight icons in the lower-right area
The Curse of Monkey Island	A three-choice action coin	An illustrated chest
Resonance	A compact console and character switching	Physical items, STM and LTM
Gemini Rue	Four target actions and a communicator	Items inside the target-action panel

The screenshots use the same workshop illustration and sample items so that the interface differences are easy to compare. The drawing is a demonstration backdrop. In your game, the room designer supplies the scene behind the interface.

> The recipes reproduce useful layouts and interaction patterns. They identify changes from the original games, especially mouse gestures, memory dragging and page-turn effects. No copied game artwork is needed.

Use the same coordinate system

1. Open your game, then open Project Settings. Set the design resolution to 1280 × 720 for these examples.
2. Open the GUI Designer. Add a GUI and give it both a clear Name and a short, unique ID.
3. Use an authored layout. The tables use a top-left origin: X increases to the right; Y increases downward.
4. Place the main HUD on a 1280 × 720 design canvas. Leave the GUI Window background and title bar off for an overlay HUD.
5. Use the listed X, Y, Width and Height values as a starting layout. Test the game at its intended display size before deciding the final font and hit-target sizes.

Term	What it means here
HUD	The interface left visible while the player explores.
Popup	A GUI shown when the player needs it, such as an inventory.
Modal	An interface that temporarily blocks ordinary room input. Use it for save menus.
Design coordinates	Positions in your authored 1280 × 720 layout, before the game scales to the display.
Name and ID	Players see captions; scripts and actions use IDs. A selector can show both.

These are readable, enlarged adaptations. They do not set the whole game to an original 320 × 200 resolution. A low-resolution game also needs suitable artwork, font handling and display scaling.

Find the controls you will use

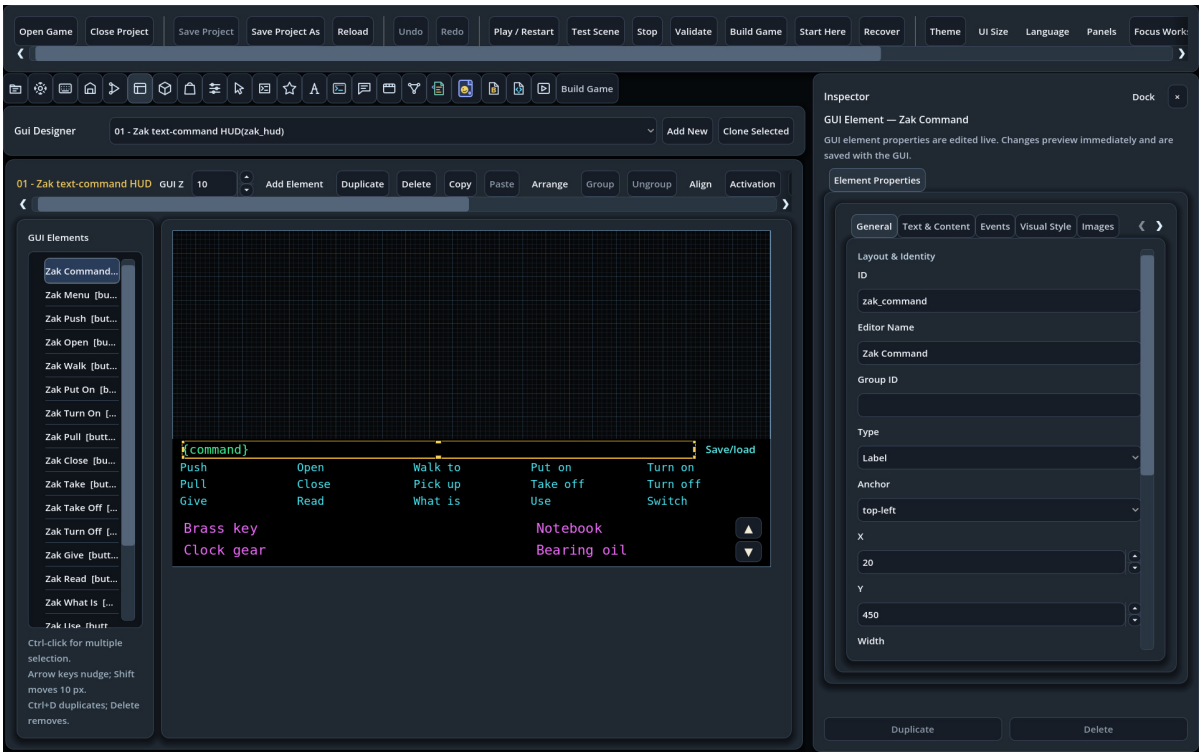

Figure 2. The GUI Designer, element list, canvas and inspector work together.

Location	Use it for
General	Position, size, drawing order, visibility and enabled state.
Text & Content	Text, font, color, alignment and text opacity.
Type Settings	Inventory rows, slot sizes, save behavior and image animation frames.
Visual Style	Colors, corners, borders and surface opacity.
Images	Normal, hover and clicked artwork; separate widget-part images.
Events	Select a verb, open a GUI, run a script or run shared actions.
Activation / Transitions	When the GUI appears and how it enters or leaves.

Build the picture in layers

A Panel can provide a colored area. A Button adds a caption and a clickable target. An Image supplies artwork. An inventory or save widget builds repeated slots for you.

Layer	Suggested Z Index	Reason
Background illustration	0	Keep the illustration behind the interface.
Large HUD or popup panel	1	This supplies the main surface.
Page panels or decorative frames	2	These sit above the outer cover or panel.
Buttons and widgets	4	These must remain visible and receive input.
Independent labels	5	Place captions clearly above their background.

Give overlapping elements different drawing orders. Do not rely on two elements with the same Z Index being drawn in the order you happen to prefer.

A decorative Image should have no click action. It can then sit behind an interactive area without becoming another button. A transparent color does not itself make a control ignore mouse input.

> An element that looks like a child of a Panel is not automatically part of that Panel. A visual stack, a tab relationship and an ordinary drawing order are different things.

Make one button work before copying it

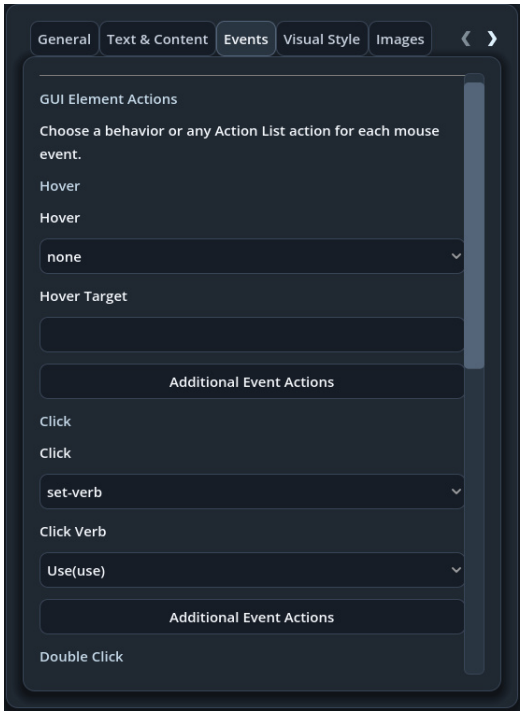

Figure 3. A verb button selects a command. The object supplies that command’s result.

1. Add a Button. Give it a unique element ID, such as mi_use.
2. Set its caption to Use in Text & Content.
3. Open Events. For Click, choose Set Verb and select your Use verb.
4. In the scene, select a real object, enable its hotspot and add a result to that object’s Use verb interaction.
5. Run Play. Click the Use button, then the object. Confirm that the result belongs to that object.

Use the Set Verb event behavior for ordinary verb buttons. It handles the special relationship between a verb button and a selected inventory item. A generic action list containing Change Current Verb Command is useful elsewhere, but it is not an identical GUI button setup.

> Copy the finished button, then change its ID, caption and selected verb. Changing the caption alone does not change what the button does.

Choose the input order deliberately

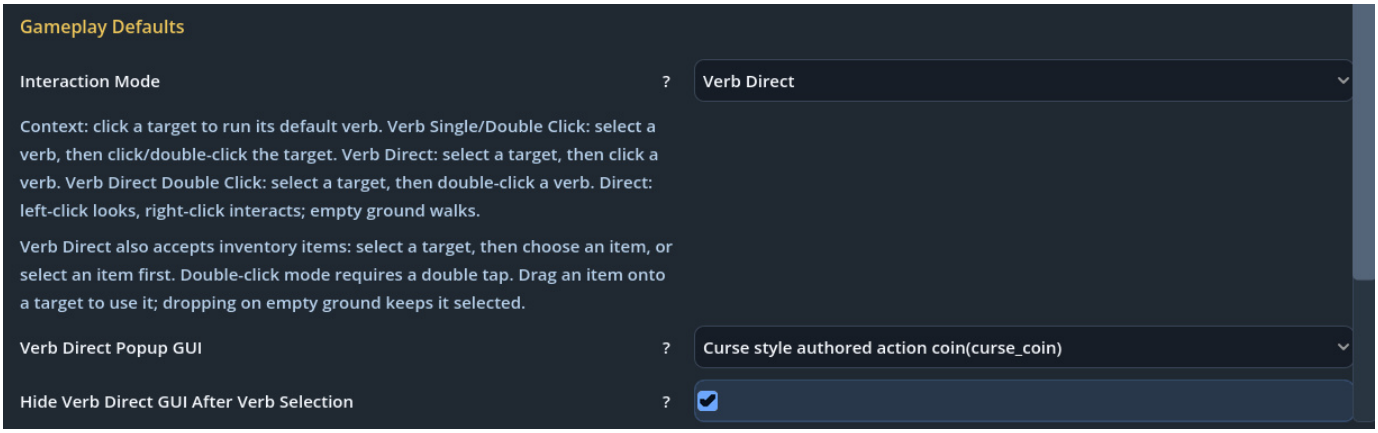

Figure 4. Project Settings chooses the input order and the optional target-first popup GUI.

PCA mode	Player’s sequence	Use in this guide
Verb Single Click	Choose a verb; click a target.	DOTT and Monkey Island 2 adaptations.
Verb Double Click	Choose a verb; double-click a target.	An optional extra confirmation for Zak.
Cycling Cursors	Cycle the action, then click its target.	Sam & Max’s cursor-first pattern.
Direct	Left-click looks; right-click interacts.	Beneath a Steel Sky, with the item adapter.
Verb Direct	Choose a target; choose a verb or item.	Authored Curse and Gemini-style popups.
Hotspot Then Verb Coin	Choose a hotspot, then a built-in radial action.	A convenient stock alternative to drawing a coin.

A GUI’s colors do not change the input mode. Configure both the look and the input order. The exact original gestures are discussed in the relevant chapter.

Map your own screens to the game

1. Open Global Settings and find the system GUI assignments.
2. Assign your finished HUD to the main HUD role.
3. Assign your popup inventory to the inventory-screen role when you use a popup inventory.
4. Assign your authored save screen and load screen to their corresponding roles. A combined save/load GUI can serve both.
5. Keep a visible Menu or Save/load button while building. Add keyboard shortcuts only after the clickable route works.
6. Use Play to test the mapped screens. A GUI Designer preview checks appearance; it does not execute the game’s event logic.

A visible-on-start HUD stays available during exploration. An inventory popup normally starts hidden and is opened by its button, a game command or an activation zone. A save menu normally starts hidden and is modal.

Shared action	Fields to fill
Show GUI	GUI: the screen you want to reveal.
Hide GUI	GUI: the screen you want to close.
Save Game Widget Command	GUI, save-widget element, command and optional slot.
Transfer Inventory	Source character, receiving character, item and count.

Only assign the interface you are currently using. The workshop screenshots show alternative designs, not a recommendation to stack them all into the same running game.

2. Build a dependable inventory

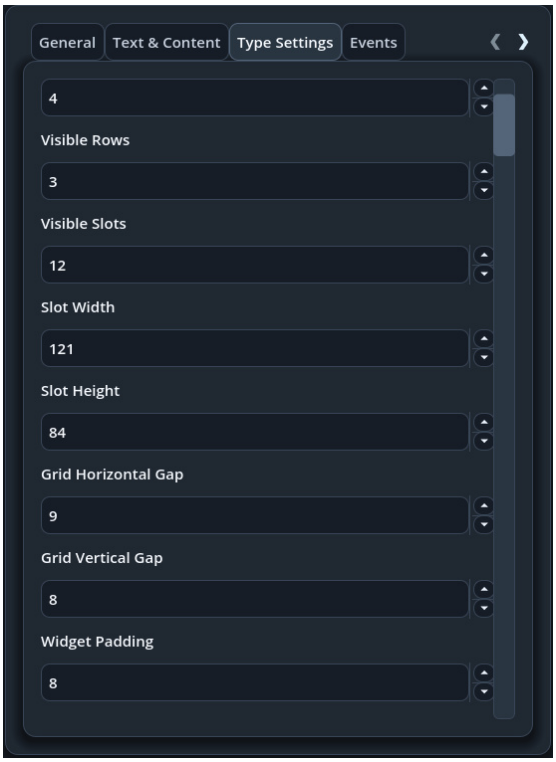

Figure 5. An Inventory Grid repeats slots from your inventory definitions.

1. Create inventory definitions first. Give every item an ID, a readable name and an icon asset when the design uses pictures.
2. Add the test items to the starting inventory, or grant them with Add Inventory / Give Inventory actions.
3. Add an Inventory Grid element to the appropriate GUI.
4. Set Columns, Rows and Items per Page to agree. Two columns and two rows display at most four slots.
5. Choose whether slots show icons, text or both. Choose the owner mode and the interactions next.

> Do not build a real inventory out of permanent images of keys and notebooks. The widget must read the inventory definitions and current ownership so that picking up, using and removing items changes what appears.

Make the grid fit its frame

Work out the minimum space before filling in the dimensions. If a grid clips its last row, making the outer frame prettier will not fix the missing space.

Direction	Minimum before headers or arrows
Width	Columns × Slot Width + horizontal gaps between columns + left/right padding.
Height	Rows × Slot Height + vertical gaps between rows + top/bottom padding.
Arrows below	Allow an extra navigation row and its separation.
Arrows on the right	Allow an extra navigation column. A long vertical arrow group also needs height.
Header enabled	Allow space for the title/header above the grid.

For the DOTT example: three 174-pixel slots with two 6-pixel horizontal gaps need 534 pixels before padding. The right-side navigation and padding fit inside the 619-pixel widget width.

Disable arrows you do not use. A pair of up/down arrows is easier to fit into a shallow HUD than four arrows arranged vertically. Page arrows move between groups; row arrows move by rows.

> Item count is not the same as page size. A four-slot display can still show ten carried items through navigation. Test with one more item than fits on the first page.

Selection, verbs and combinations differ

Option	Effect
Use Selected Verb	The current verb can examine or act on an inventory item.
Use Selected Verb off	Ordinary clicks follow item selection behavior rather than running each current verb.
Drag Items Together	Allows the widget’s drag-combination route. Define a valid item-pair result as well.
Player Drag Sorting	Changes the display order of carried items. This is separate from combining them.
Per-character inventory	Each controllable character can carry a different collection.
Specific-character owner view	Displays another character’s items; treat that view as read-only for normal selection.

With Use Selected Verb enabled, Walk is useful as the ordinary selection state. To let Use collect a first object for a two-object command, enable the Use verb’s preposition and opt the inventory item into that preposition. Give also needs its own separator and second-target response.

For a target-first Gemini-style popup, begin with the simpler selection route: Use Selected Verb off, and no pending preposition on the selected item. The engine can then apply the chosen inventory item directly to the target it stored.

> An item’s normal Use action, an item-on-world response and an item-on-item combination are three different cases. Put the result on the appropriate interaction rule.

Use a key on a real object

1. Create an inventory definition called Brass key, ID brass_key.
2. Create or select your scene’s locked desk. Give it a clear object ID and enable its hotspot.
3. Add an inventory interaction rule for using brass_key on that desk.
4. Make the rule set a desk_unlocked Boolean variable to true. Add a short message or an opening animation if you have one.
5. If the key should disappear after use, add Remove Inventory after the successful result. Leave it in the inventory if the story expects it to be reusable.
6. Test an unrelated item on the desk. It should give reasonable feedback without unlocking anything.

An item-on-item recipe works similarly: define the source item, the target inventory item and the resulting actions. If the recipe makes a new object, remove the consumed ingredients and add the resulting inventory definition.

When using raw rules, a scene target is written as entity: followed by its object ID; an inventory target is inventory: followed by its item ID. Prefer the inspector’s target selectors when they are available.

> Do not use the player’s visual proximity to guess whether an item worked. Test the saved variable, resulting object state and inventory contents.

Build a safe save menu once

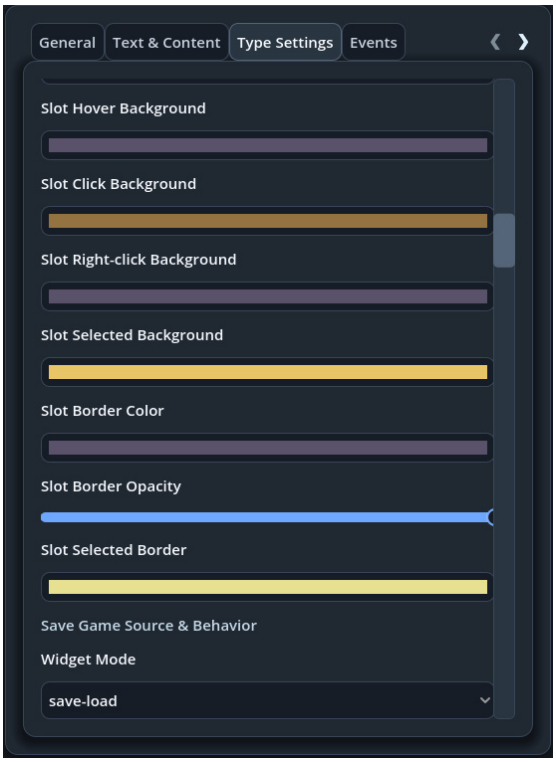

Figure 6. A Save Game element handles the slot data; ordinary buttons issue commands.

1. In Global Settings, choose the number of manual save slots. The examples use twelve so there is a second page to test.
2. Create a hidden, modal GUI and add a Save Game element.
3. Set Widget Mode to save-load. Start with manual slot 1 and end at 12.
4. Set Click to select and Double-click to none while learning. Do not let an accidental double-click load a game.
5. Enable Prompt for Name and Confirm Overwrite. Keep autosave outside the ordinary manual range unless you deliberately support it.
6. Add separate Save selected, Load selected and Back/Play buttons.

> The Save Game element reads real slot files. A List Box containing typed titles is not a working substitute for the save system.

Wire Save, Load and Back correctly

Button	Click action	Required fields
Save selected	Save Game Widget Command	GUI: your save GUI; Element: its save widget; Command: save-selected; Slot: -1.
Load selected	Save Game Widget Command	Same GUI and widget; Command: load-selected; Slot: -1.
Rename	Save Game Widget Command	rename-selected; enable renaming in the widget too.
Back / Play	Hide GUI	Your save GUI. The game returns to exploration.

Slot -1 uses the widget’s selected slot, but this build can fall back to the first available slot if nothing is selected. Select a row first. If you require explicit selection, use the guard script in Chapter 12. A slot number supplied explicitly is a different route; use it for a deliberately fixed slot or the two-widget logbook dispatcher.

Keep the name prompt and overwrite prompt visible until the player answers. Do not hide the menu, run another modal prompt or change the scene at the same moment. The prompt’s Cancel must leave the old slot intact.

> The native naming and confirmation dialogs are functional PCA controls. Their appearance is not an exact copy of each historical game’s in-place text field. Replacing those dialogs with an authored name-entry screen is a separate customization.

Choose the correct save presentation

Presentation	Set these properties
Text list	One column; screenshots off; short rows; simple name template.
Screenshot card	Screenshots on; choose placement, dimensions and fit; allow room for the caption.
Illustrated logbook	Draw the pages behind the widgets. Keep page art, slot art and screenshot frames separate.
Empty manual slot	Show empty slots and give them an understandable label.
Autosave	Treat its slot and permissions explicitly; a manual save button should not overwrite it casually.

Useful save-text tokens include {slotNumber}, {name}, {scene}, {date}, {time} and {empty}. These belong to the Save Game element’s text templates. They are not universal tokens for every ordinary Label.

The editor includes presentation presets. Apply a preset first, then change the dimensions and style. Applying a preset again can replace your layout values. A preset called Curse-era illustrated uses wide rows; it does not automatically produce the original game’s six-box two-page logbook.

> A save thumbnail is a picture of the scene captured by the save system. It is not a replacement for the stored scene, character, inventory and variable state.

3. Animate the right thing

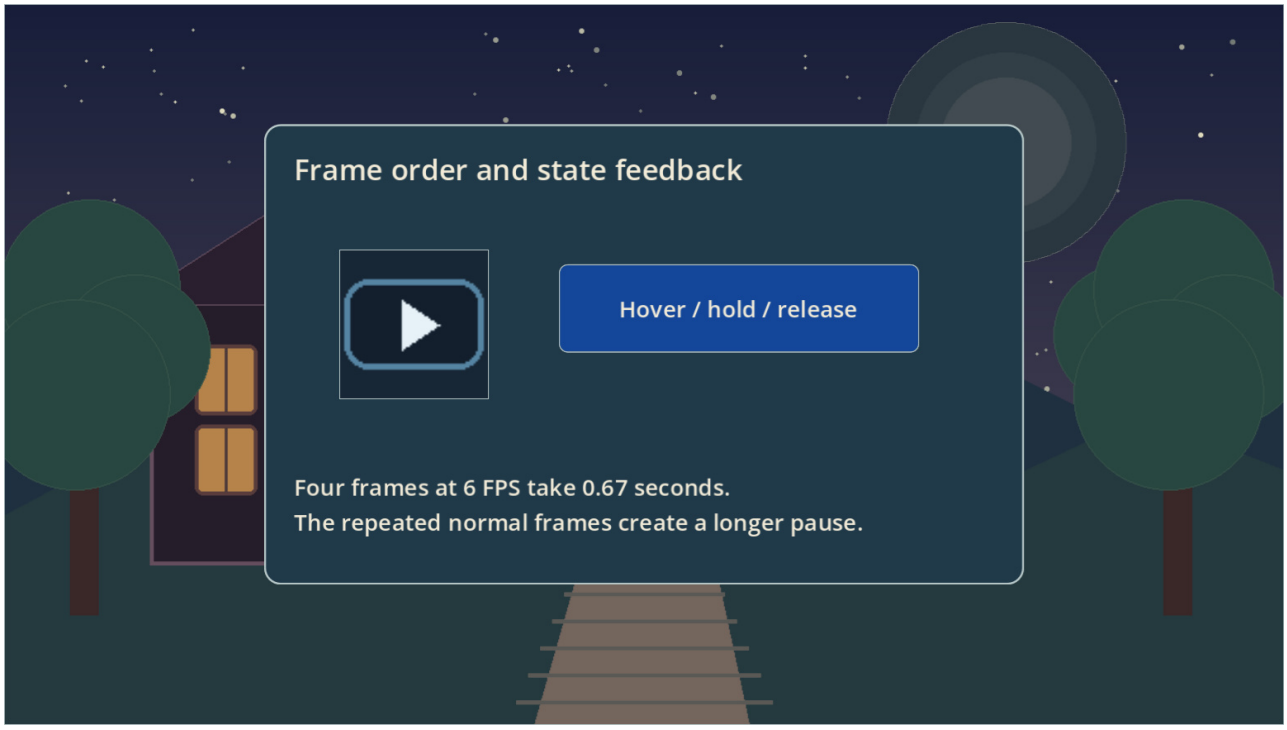

Figure 7. A real Image sequence and a stateful Button, rendered in Play.

You want…	Use…
A menu to enter or leave	The GUI’s open and close transitions.
A button to react while held	Normal, hover and clicked images or colors.
A small drawing to move repeatedly	An Image element with ordered frame asset IDs and FPS.
An inventory icon to animate	The item’s icon frames plus Animate Icons on the widget.
A book/chest drawing to play once	A non-looping Image sequence and a short completion script.
The exact original input gesture	A tested input adapter, separate from the picture.

A scale transition can make a chest GUI appear. It does not animate the wooden lid. A colored hover state is feedback, not a multi-frame movie.

Give a GUI an entrance and an exit

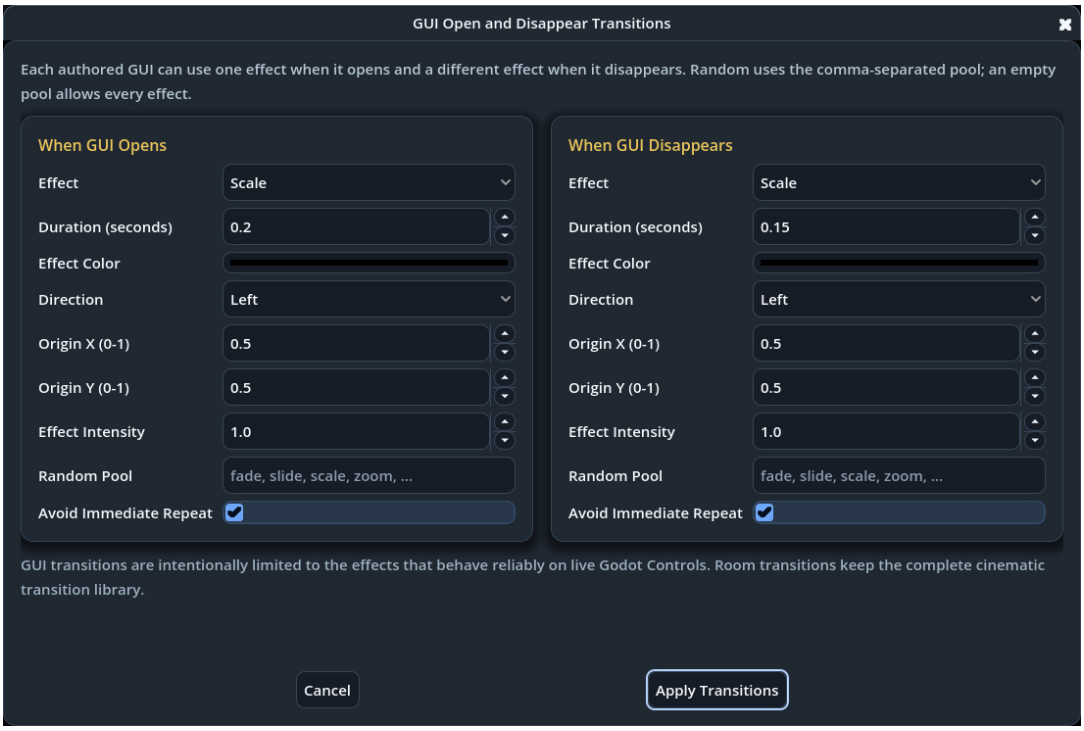

Figure 8. Transitions belong to the selected GUI, not to a single inventory slot.

1. Select the complete GUI, then click Transitions in the toolbar.
2. Set When GUI Opens to the chosen effect and duration.
3. Set When GUI Disappears separately. A shorter closing duration usually feels responsive.
4. Use Direction for a slide. Use Origin for a scale/zoom effect.
5. Apply the transition and test with Show GUI / Hide GUI during Play.

Start with Cut for early text interfaces, 0.12-0.20 seconds for a compact popup, and roughly 0.20-0.28 seconds for a larger chest or book. These are the workshop’s timing choices, not measured timings from the original games.

> Awaiting Show GUI does not by itself mean its animation has finished. If a script must wait before the next step, wait for a known duration or the actual completion condition.

Read an animation as a small schedule

Time	What is drawn	What accepts input
Before opening	The HUD or closed-state icon.	The opener.
Opening frames	The entering panel or opening drawing.	Keep live contents unavailable if they would move under the pointer.
Fully open	The stable interface with readable content.	Buttons, slots and navigation.
Closing frames	The reverse motion or disappearing panel.	Block another open/close request until the intended state is clear.
Closed	The gameplay HUD again.	The ordinary room and the opener.

Decide what happens if the player clicks twice during opening. A simple busy flag can reject another request until a one-shot drawing finishes. For stock transitions, test rapid Show/Hide calls as well as slow, polite clicks.

Do not save a temporary “halfway through the page turn” as the intended menu state. Keep the persistent value as the destination page, selected character or learned clue. Rebuild the final presentation from that value when restoring a game.

> A pause before the first frame is different from holding the last frame. Write the schedule first; then set FPS, frame order and any explicit waits to match it.

Prepare image frames without a wobble

1. Draw each frame on the same transparent canvas. Keep its width and height identical throughout.
2. Keep the stationary part in exactly the same place. For a chest, its base should not drift while the lid rises.
3. Export numbered PNGs: chest_open_01, chest_open_02 and so on. Import each PNG as an image asset in PCA.
4. Give the imported assets short IDs with the same order. Check the IDs in the importer; the filename and asset ID need not be identical.
5. Put only the moving artwork in the animation. Keep the inventory grid, text and buttons as live elements above or below it.
6. Test every frame over both a dark and a bright background so that unwanted solid edges are visible.

Problem	Likely repair
The drawing jumps	Align the base and use identical canvas dimensions.
The image changes size	Keep identical dimensions and the same Content Fit.
A border moves with the lid	Separate the stationary frame from the moving artwork.
The contents tilt or blur	Do not paint live inventory items into the lid animation.

A sheet containing several drawings is still one image until you import/slice it as appropriate. The frame-ID field used here expects separately addressable image assets, not a guess at positions inside an unsliced sheet.

Set frame order and playback speed

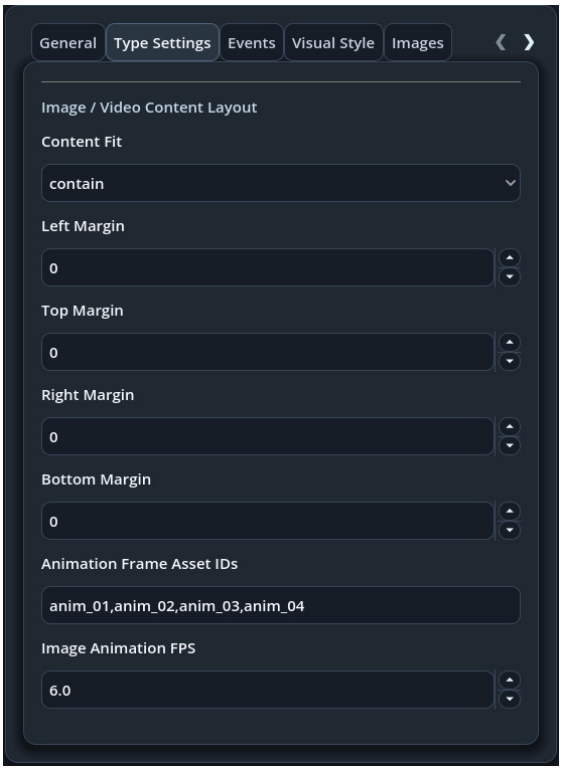

Figure 9. An Image’s Type Settings lists frame asset IDs and the playback rate.

1. Add an Image element. Assign a normal Media Asset so that it has a useful initial picture.
2. In Type Settings, enter Animation Frame Asset IDs in playback order, separated by commas.
3. Set Image Animation FPS. The example uses anim_01, anim_02, anim_03, anim_04 at 6 FPS.
4. Choose Content Fit and keep the element’s bounds constant.
5. Run Play and watch the sequence. The ordinary Image sequence loops unless you use the one-shot script later in this guide.

A sequence takes frame count ÷ FPS seconds. Four frames at 6 FPS take about 0.67 seconds; six frames at 12 FPS take 0.50 seconds. In this build, repeated asset IDs are deduplicated. To hold a drawing for extra intervals, export distinct frame assets with distinct IDs, even when two frames contain identical pixels.

> FPS belongs to this sequence. Increasing it does not speed up the whole game or the character’s walk animation.

Normal, hover, held and released

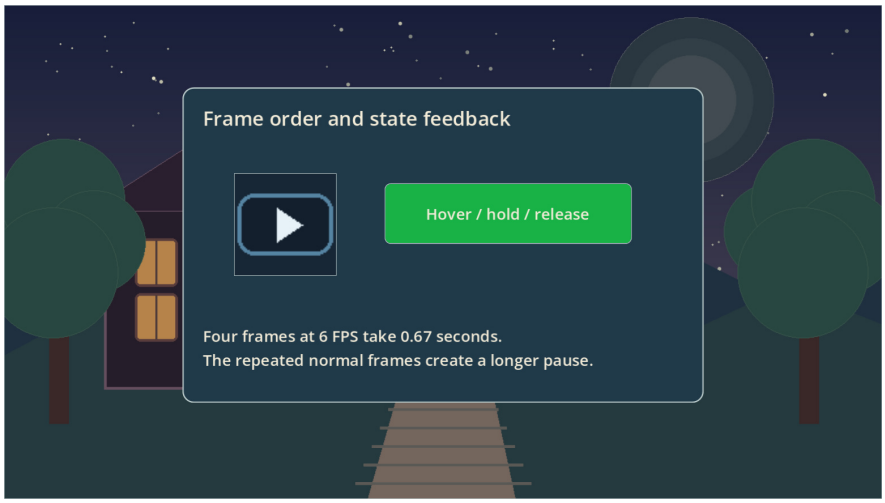

Figure 10. Hover uses the hover image before a press.

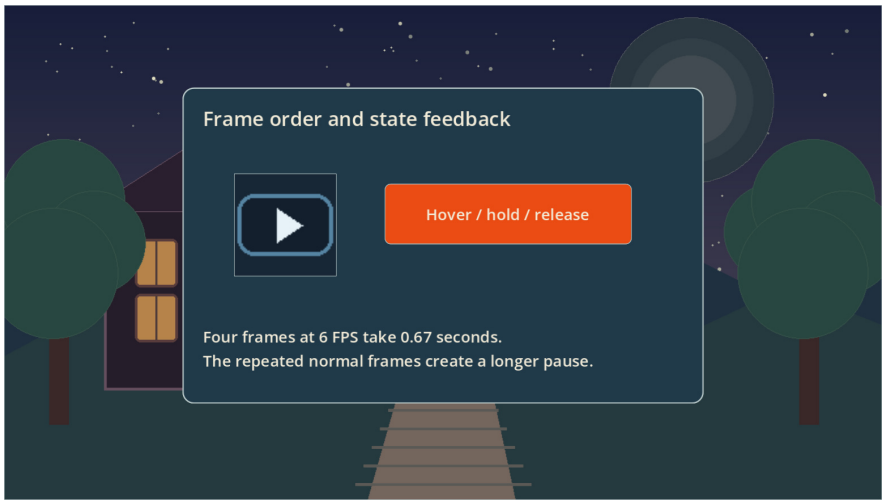

Figure 11. Holding the button uses the clicked image.

Assign the three pictures in Images: Background Image, Hover Image and Clicked Image. Keep their canvas size and alignment identical. A small change in brightness or pose is usually enough.

> A clicked image describes a held press. It should not become the permanent selected appearance of a tab, radio option or inventory slot.

Check release before polishing the art

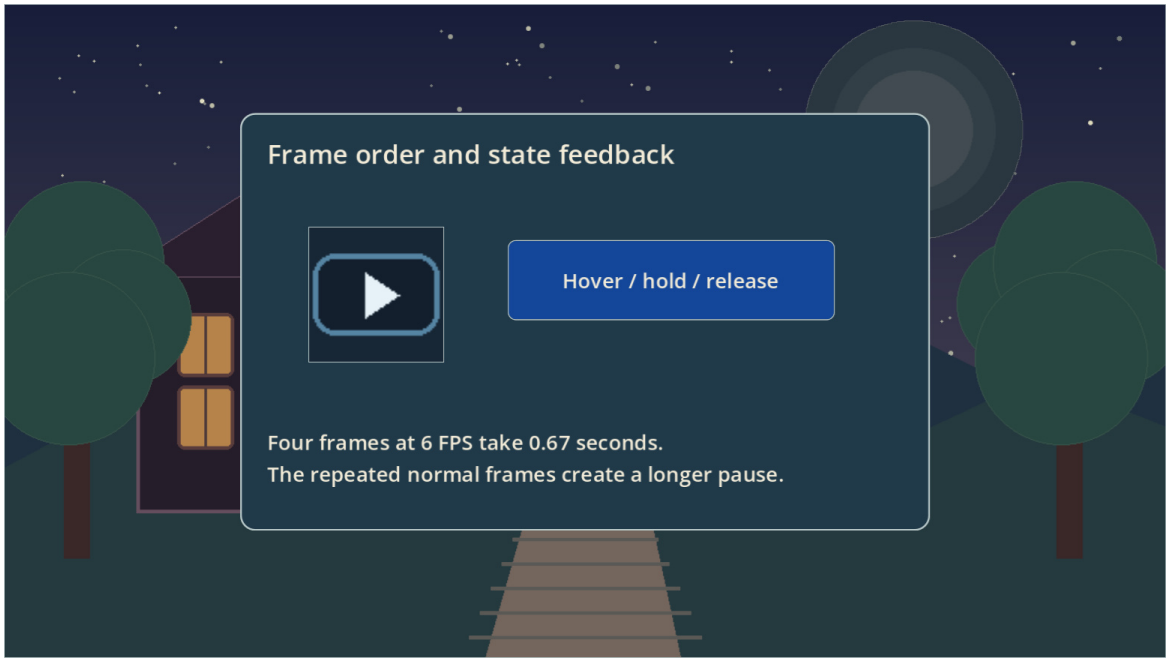

Figure 12. After release, the ordinary picture is restored.

1. Move onto the button and check its hover picture.
2. Press and hold. Check that only the held picture appears.
3. Release while still over the control. Confirm that the clicked picture does not stick.
4. Move out and back in. Check hover again.
5. Press, drag outside and release. Check that the control does not keep the held state.
6. Repeat with keyboard focus or touch if your game supports them.

PCA may suppress a fresh hover until the pointer leaves and returns after a press. Treat release-to-normal as the important result. A persistent inventory selection has a separate selected background or border.

> If clicking immediately hides the GUI, use a harmless test button to inspect its held state. A vanished button cannot show you its release picture.

Animate icons and widget parts separately

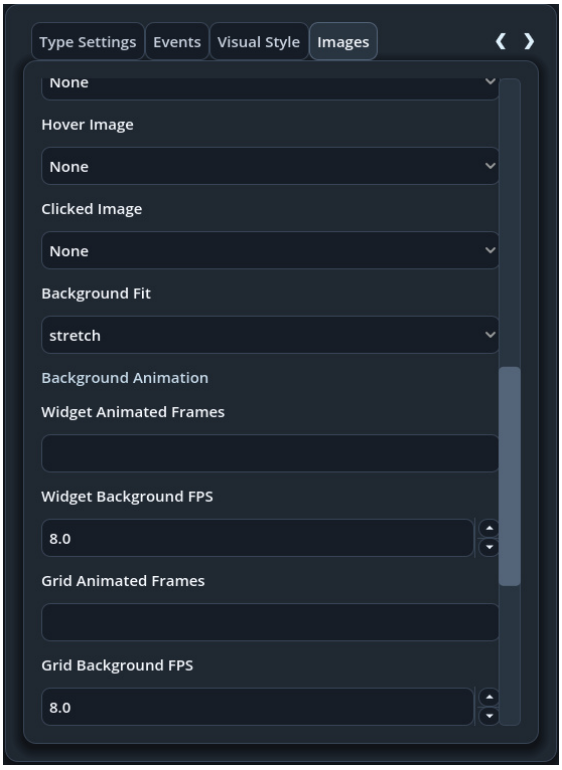

Figure 13. Widget and grid image sequences have separate frame fields and rates.

Inventory items can carry their own iconFrames and iconFps settings. Turn on Animate Icons in the inventory widget when you want to display those frames. A deliberately moving radio indicator can be useful; an entire bag of endlessly blinking objects is hard to inspect.

The Images tab also exposes supported frame, grid and navigation-image animation fields. The outer widget’s sequence is different from a grid background sequence, a slot’s image state or the item icon inside that slot.

Ordinary Button hover/click image pickers select state artwork. They are not general multi-frame hover timelines. Use an Image sequence, the supported item-icon frame fields or an explicit script when a particular state needs several frames.

> Keep a static option. A player should be able to read the inventory, choose a save and complete a conversation without following decorative motion.

4. Zak McKracken: text commands

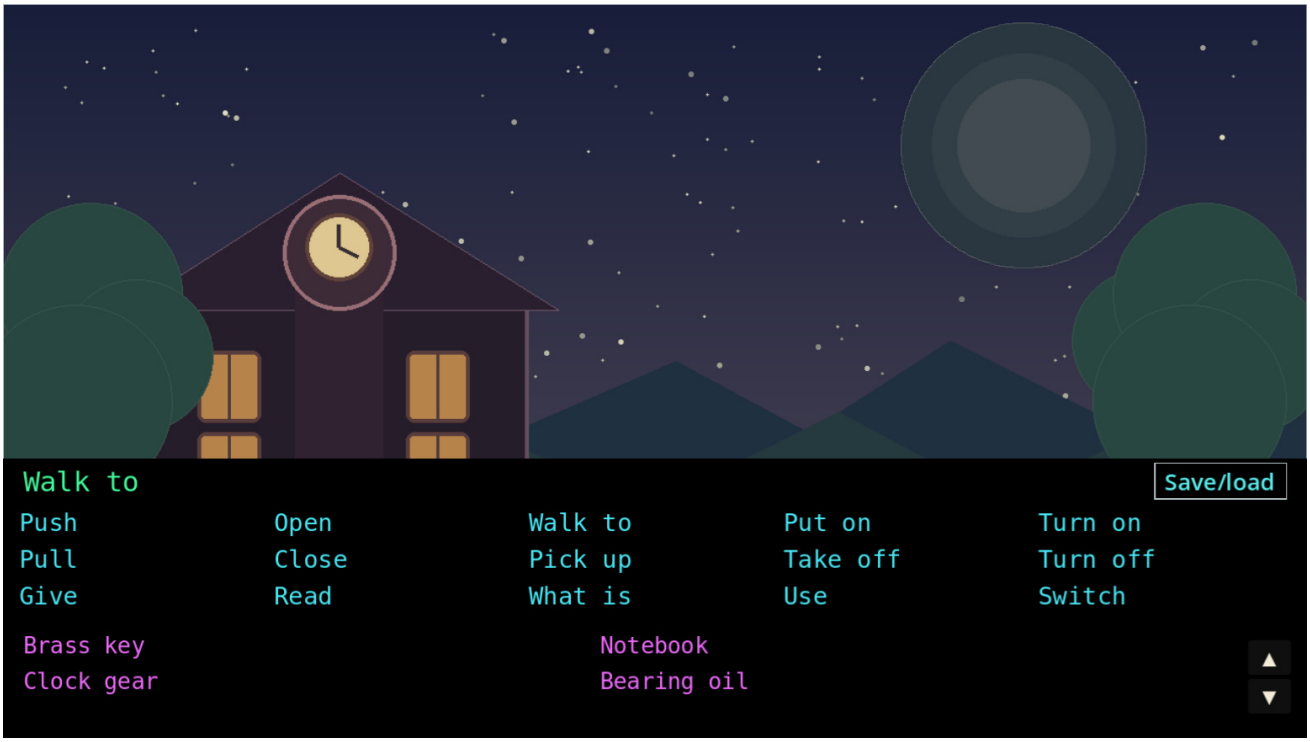

Figure 14. A text-command interface with a four-entry inventory beneath the verb bank.

The early Zak interface makes the words themselves do the work: a sentence line, fifteen command positions and a text inventory below them. The original also has a separate command-confirmation step. Its manual describes selecting words and confirming the resulting sentence. [1]

We will build the layout first. For the first working version, use Verb Single Click. Once the buttons and object results work, try Verb Double Click if you prefer a deliberate final confirmation.

> Verb Double Click is a practical PCA option, not a complete implementation of every original Zak sentence-editing shortcut. In particular, clicking the sentence line itself is not automatically a universal Execute command.

Lay out the black command area

1. Create the GUI zak_hud. Start it visible and assign it as your main HUD.
2. Add zak_band, a black Panel at X 0, Y 446, Width 1280, Height 274. Set Z Index to 1.
3. Add zak_command, a Label at 20, 450, size 1100 × 38. Use green #42F899, a 27-pixel monospaced font and no visible background.
4. Begin with {command} as the label text. The sentence-line helper later adds the selected item and preposition prefix.
5. Add the fifteen text Buttons from the next page. Set their Z Index above the Panel.
6. Add a small Save/load button at 1130, 450, size 130 × 36, opening zak_save.

Style	Value
Button background	Black; no rounded frame; use a restrained dark hover background.
Verb text	#4AE9F5; 24 pixels; left aligned; monospaced.
Inventory text	#EF6CFC; 22 pixels; monospaced.
Border Radius	0 on the band, buttons, grid and slots.

The example reserves the lower portion for controls. Make sure important room hotspots are not hidden behind that area. Design or frame the room with the visible play area in mind.

Build the fifteen verb positions

Row	Column 1	Column 2	Column 3	Column 4	Column 5
1	Push	Open	Walk to	Put on	Turn on
2	Pull	Close	Pick up	Take off	Turn off
3	Give	Read	What is	Use	Switch

For column c and row r, counting from zero, place the button at X = 16 + 250c and Y = 492 + 36r. Each button is 240 × 34. Use unique element IDs such as zak_push and zak_read.

1. Create any missing verbs in the Verb Designer. A button caption is not a verb definition.
2. Set each ordinary Button’s Click behavior to Set Verb and select its matching verb.
3. Give scene objects meaningful results for the verbs you expose. Read might show a document; Turn on might change a lamp’s state.
4. Treat What is as a discovery command. Name the hotspots clearly. If you want it to reveal labels or highlights only while selected, author that behavior explicitly.
5. Make Switch open a small character-choice GUI rather than merely selecting a verb called Switch.

> Only expose Put on, Take off or similar commands if your game gives them something useful to do. A faithful-looking button bank can still be frustrating if most commands never have a response.

Make inventory entries out of words

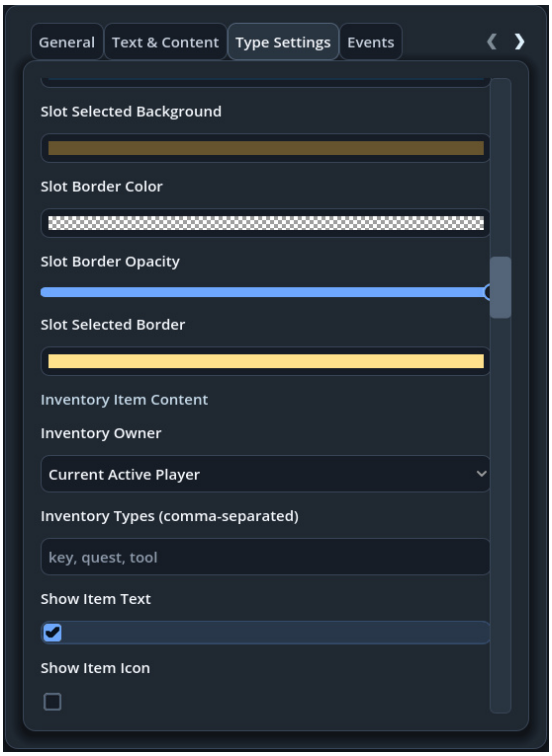

Figure 15. Text slots do not need inventory icons to be visible.

Property	Value
Element	zak_items; Inventory Grid at 8, 604; size 1264 × 112.
Capacity	2 columns, 2 rows, 4 items per page.
Slot size	560 × 34; vertical gap 1.
Slot content	Show Text on; Show Icon off; Show Quantity off.
Navigation	Arrows on the right; enable Up and Down; disable Left and Right.
Owner	Current player; per-character inventory in Project Settings.
Behavior	Use Selected Verb on; Player Drag Sorting off.

Keep the inventory definition’s Name readable. That name becomes the visible entry. The item ID remains the stable reference used by scripts and interaction rules.

Build a two-object sentence

1. In the Verb Designer, enable Allow Preposition for Use and enter with.
2. On Brass key, opt the Use interaction into the preposition. This marks the key as a possible first object.
3. Add its second-target response for the locked desk, or use its matching item-on-desk inventory interaction.
4. Click Use, then Brass key. The game should now wait for a second target.
5. Point at and click the desk. Confirm that the correct result happens once.
6. Try cancelling the unfinished command, changing the active character and leaving the scene. An old first object must not remain active in the wrong context.

Install the sentence_line global script from Chapter 12 if you want the authored label to display the pending prefix and selected-item name. A plain {command} label is a useful first test, but it is not a complete sentence-builder display in every state.

For Give, use to as the separator. Define the receiving character and the transfer result. Do not interpret every object pair as valid just because its words form a grammatical sentence.

> Keep the original IDs when you rename an item for the player. Otherwise you will be repairing every pair rule instead of writing the next puzzle.

Create the expedition save list



Figure 16. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	zak_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	zak_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #000000; text #3BEACB; hover #131B36; selected #70F3ED.

Use a quiet text list and immediate Cut transitions. The workshop has twelve manual slots to test paging; the historical manual describes a ten-save arrangement on some supported systems. This is an authored capacity choice, not an imposed original-game limit.

Wire the text menu and character switch

1. Set the Save button to Save Game Widget Command: GUI zak_save, Element zak_saves, command save-selected, slot -1.
2. Use load-selected for Load and rename-selected for Rename. Leave renaming enabled if you include that button.
3. Set Play to Hide GUI, targeting zak_save.
4. For the Switch command, create a separate hidden GUI containing one Button per playable character.
5. Set those character Buttons to Switch Character and select the character definition. Confirm that each character is marked playable and switch-enabled.
6. Close the character-choice GUI after a successful switch. Check that zak_items now reads the new player’s inventory.

Use the original’s restrained visual rhythm. The command words should react immediately. A long fade for every word selection would make the interface feel slow even if it looked attractive.

> Save under a descriptive name, change the active character and use an item, then load. Check ownership and game state, not just whether the old row title appears.

Finish and test the text interface

Element	Animation / feedback plan
Verb Button	Normal cyan text, a readable hover change and a held-press state. No constant movement.
Command line	Update the text when the verb, item or target changes. Do not type it out letter by letter.
Inventory	Scroll or replace the visible entries immediately. Keep text stable while the pointer is over a row.
Save menu	Cut open and Cut close. Selection is a stable color change.
Optional cursor	Use a static pointer or a deliberately authored busy indicator. Do not block a command while decorative frames loop.

1. Test every verb on one useful target and on an unsuitable target.
2. Carry at least five items so that the four-entry inventory must navigate.
3. Check the complete Use key with desk path and a cancelled first-object selection.
4. Switch characters and verify independent inventories.
5. Save, rename, cancel an overwrite and restore the saved puzzle state.

> The appeal of this interface is its legibility. Spend the final pass on names, spacing and consistent command feedback.

5. Day of the Tentacle: verbs and faces

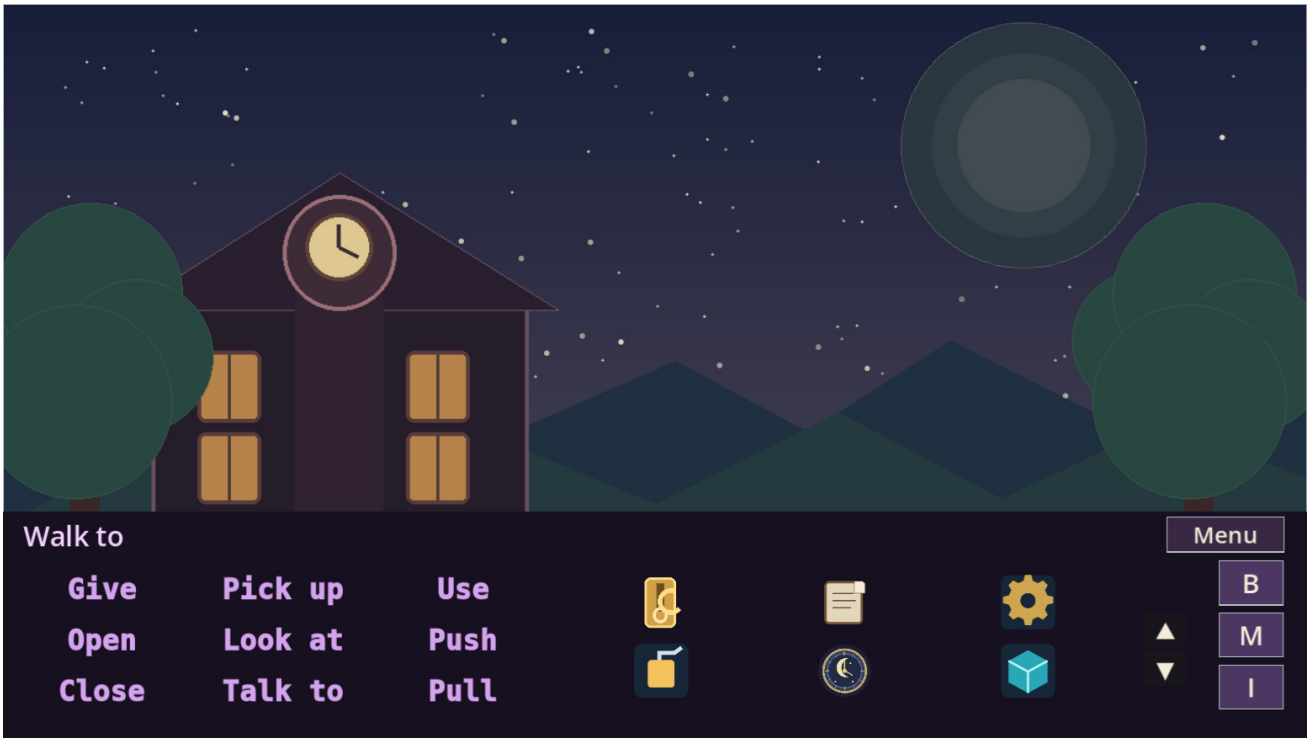

Figure 17. Nine verbs, six inventory icons and a narrow strip of character controls.

DOTT keeps the verb bank visible, uses an icon inventory and lets the player switch between the available characters by clicking their portraits. Its manual describes six visible inventory objects and item transfer through another character’s portrait. [2]

This workshop uses B, M and I as placeholder portrait captions for Builder, Mara and Ivo. Replace them with your own portraits after the controls work. Build the six-slot inventory first; do not copy the eight-slot Monkey Island 2 layout by accident.

> The screenshots adapt the layout to 1280 × 720 and use sample artwork. They do not reproduce Bernard, Hoagie, Laverne or the original portrait drawings.

Draw the lower control strip

Element	Position and size	Appearance
dot_band	0, 498; 1280 × 222	Background #171020; no visible border.
dot_command	20, 503; 1080 × 36	27-pixel text; #F3D2FF.
Verb bank	Start at 12, 552	3 columns; horizontal step 177; vertical step 50.
Each verb Button	171 × 46	28-pixel bold monospaced text; #D4A4EE.
Inventory	551, 547; 619 × 173	Six icon slots, configured next.
Character controls	X 1193; Y 546, 597, 648	64 × 44 each.
Menu	1142, 503; 116 × 35	Opens dot_save.

Arrange the verbs as Give / Pick up / Use, then Open / Look at / Push, then Close / Talk to / Pull. Use a dark, quiet surface so that the words and item icons remain the important shapes.

> The bank is a set of nine Buttons, not one picture containing nine words. Every command needs its own clickable rectangle.

Connect the verbs and sentence line

1. Set Project Settings → Interaction Mode to Verb Single Click.
2. Create the nine verb records and give them stable IDs. Keep Walk available as the normal scene movement command even though it has no large word button here.
3. Set each Button’s Click behavior to Set Verb and select the appropriate record.
4. Set the command Label to {command} for the first pass, then install sentence_line if you want the full selected-item prefix.
5. Give one door an Open result and one character a Talk to result. Confirm that the same GUI selects two different types of action.
6. Enable Use prepositions on the items that genuinely need a second object.

A direct click on an inventory object in the original is a convenient Use shortcut. In PCA, choose the intended inventory behavior explicitly. For the verb-sensitive version here, Walk selects an item; Use plus the item’s preposition setting starts a pair command. If you instead turn Use Selected Verb off, item clicks always follow ordinary selection, and examining them needs another route.

> Do not quietly lose Look at on inventory items while simplifying selection. Test both examining an item and using it on another object.

Make the six-slot icon inventory

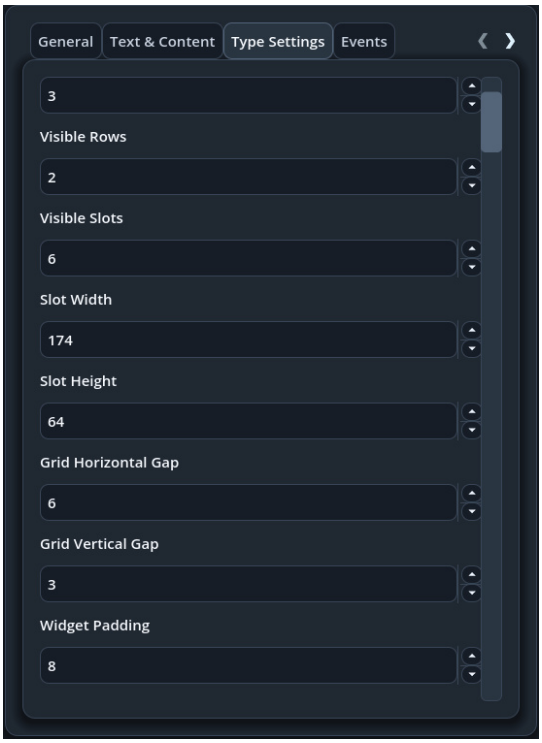

Figure 18. Three columns by two rows make the six-item display.

Property	Value
Element	dot_items at 551, 547; size 619 × 173.
Grid	3 columns; 2 rows; 6 items per page.
Slots	174 × 64; horizontal gap 6; vertical gap 3; padding 8.
Content	Icons on; text off; icon size 56; quantities off.
Arrows	Place on the right; Up/Down on; Left/Right off.
Feedback	Hover #4D305F; selected #775137.
Ownership	Current player; Project inventory mode per-character.

Give every icon enough empty space around its edge. If a sprite fills the whole slot, hovering and selected borders can become hard to see.

Switch characters or pass an item

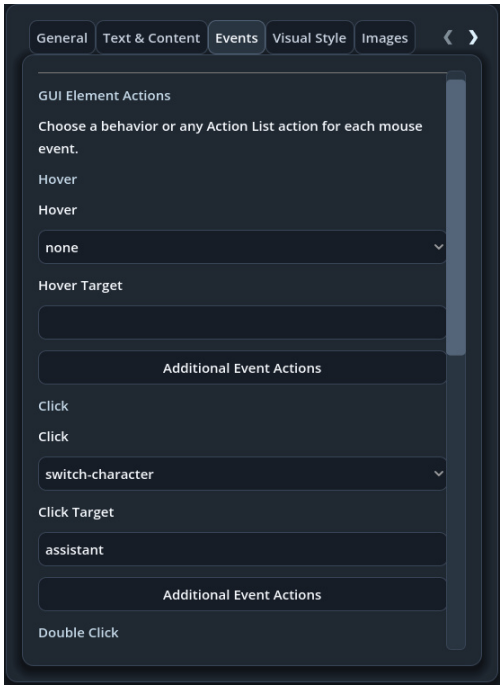

Figure 19. A portrait can switch control. A transfer-aware portrait needs a small script instead.

For simple switching, use the Button’s Switch Character behavior. For the DOTT-style “selected item onto portrait” shortcut, use transfer_to_mara from Chapter 12 on Mara’s portrait.

Player state	Portrait result
No item selected	Switch to the chosen character.
An item selected	Transfer one unit from the current character to the recipient. Keep control of the current character.
Transfer forbidden	Keep the item and provide feedback appropriate to the puzzle.
Character unavailable	Disable or hide the portrait until the story unlocks it.

Create a separate script or recipient parameter for each portrait. Do not paste the same Mara recipient into every Button. If a puzzle requires a transfer machine or a story condition, check that condition before Transfer Inventory.

> The original’s time-travel rules belong to its story. Your own transfer shortcut should follow your game’s rules, rather than moving every item anywhere without a check.

Give the laboratory a matching save menu

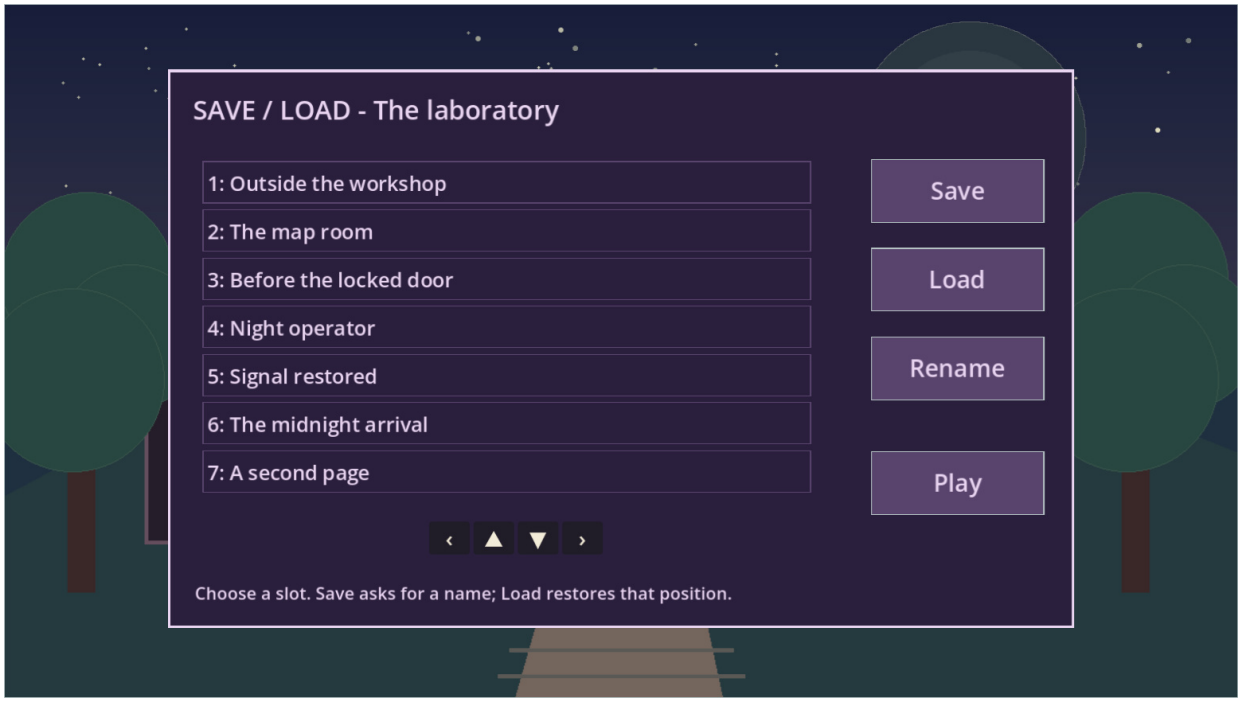

Figure 20. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	dot_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	dot_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #2A203D; text #EAD5F2; hover #58446C; selected #F2CC7F.

Keep screenshots off for this text-menu recipe. The seven visible rows and twelve-slot capacity are workshop choices. Use Save, Load and Play labels so that the purpose of each control stays obvious.

Animate new portraits without moving targets

Part	Suggested schedule
New character unlocked	Flash or pulse the portrait three times, then leave it in its stable available state.
Portrait hover	A brighter frame or a small expression change.
Portrait press	A held pose that returns on release.
Character switch	Update the active indicator and inventory once the switch succeeds.
Inventory arrows	A quick state response; keep the arrow’s hit rectangle fixed.
Save menu	Cut open and close; stable row selection.

The optional flash_portrait script in Chapter 12 pulses the ordinary Button’s color without rebuilding the inventory. Use it when the portrait first becomes available, not every time the player opens a menu.

If you draw several portrait frames, keep the eyes, face border and crop aligned. Put expressions inside a fixed frame; do not resize the character’s clickable portrait as it smiles or blinks.

> A flash means “newly available,” not “currently selected.” The steady active-character indicator must remain understandable after the flash stops.

Play a complete transfer test

1. Start with Builder carrying a key and Mara carrying a notebook.
2. Select the key. Click Mara’s portrait. Builder should remain active and lose one key; Mara should gain it.
3. Clear the selected item and click Mara’s portrait again. Control should switch and the displayed inventory should change.
4. Use the key on a suitable target as Mara. Confirm that an unsuitable character or target cannot use a forbidden item.
5. Carry seven items to test the six-slot inventory’s navigation.
6. Save, transfer another item, then load. Verify both characters’ inventories and the active portrait.

For dialogue, use the game’s dialogue-choice GUI rather than putting temporary conversation phrases into the inventory. Hide or replace the verb bank during a conversation if that suits your layout.

> A successful screenshot is not enough here. This interface’s main feature is that the portraits, ownership and active player agree.

6. Sam & Max: let the cursor carry the action



Figure 21. A mostly clear play area with an inventory opener and a small status line.

Hit the Road moves the command choice out of a permanent word bank. Its cursor actions include Walk to, Look at, Pick up, Talk to and Use; the original also reacts visually over usable hotspots. [3]

Use PCA’s Cycling Cursors for the basic interaction order. Build a floating inventory separately. The artwork in this chapter is a neutral evidence-box design rather than a copy of the original inventory illustration.

> A different cursor picture is not a different action unless its verb record and object responses are configured too.

Set up the five cursor actions

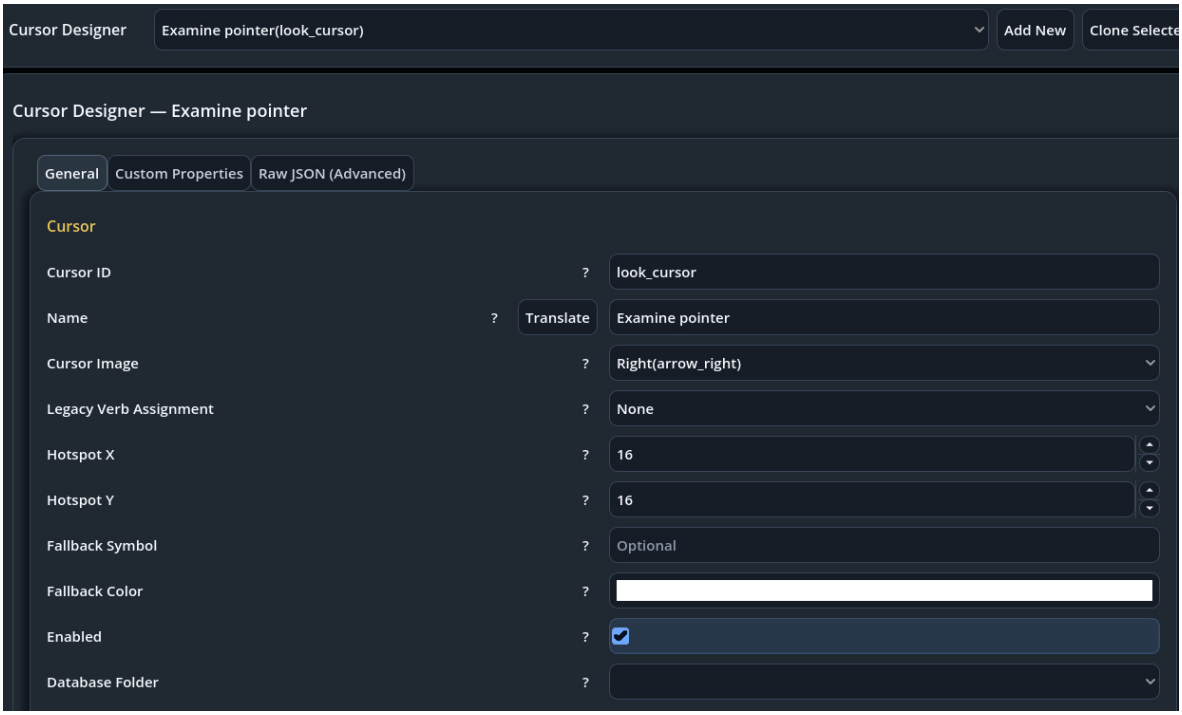

Figure 22. A cursor definition has an image and a hotspot; a verb selects the appropriate cursor.

1. Keep the enabled cycling set to Walk, Look, Pick up, Talk and Use. Disable unrelated verbs in this project so cycling does not visit fifteen unwanted actions.
2. In Project Settings, choose Cycling Cursors. In the runtime-controls settings, keep the control system and cursor cycling enabled.
3. Create a cursor record for each action. Assign an imported pointer image and set its hotspot.
4. Assign each cursor record to its matching verb, then enable custom cursors in Global Settings.
5. Run Play. Right-click with no inventory item selected to cycle, then left-click a target to execute the displayed action.
6. Test clearing a selected inventory item too; selected-item cancellation and cursor cycling should not compete.

PCA’s ordinary hardware cursor record is a static image. Do not assume a GUI Image’s FPS field also animates the operating-system pointer. Multi-frame hotspot cursor movies need a dedicated pointer implementation.

Build the minimal HUD and evidence box

Element	Position / size	Purpose
sam_bag	1150, 595; 104 × 96	Open sam_inventory.
sam_status	24, 662; 1050 × 38	Use {verb} and {target} while learning.
sam_box	665, 100; 585 × 470	Inventory Panel, #B7D6C7; border #263D44; radius 8.
Inventory heading	690, 115; 450 × 43	Dark text, “Carried evidence.”
Close Button	1170, 116; 56 × 44	Hide sam_inventory.
sam_items	687, 174; 540 × 332	The live inventory grid.

Keep the inventory hidden until requested. Its light surface should contrast with the room, and its Close button should remain in a predictable place. A picture of a bag can replace the temporary BAG caption later.

If you add a buddy-assistance command, make it a deliberate verb or a dedicated action with target-specific responses. A companion portrait alone does not make the companion solve puzzles.

> Do not cycle through the inventory box’s GUI controls as though they were room verbs. The pointer over a menu needs clear ordinary control feedback.

Make the floating inventory useful

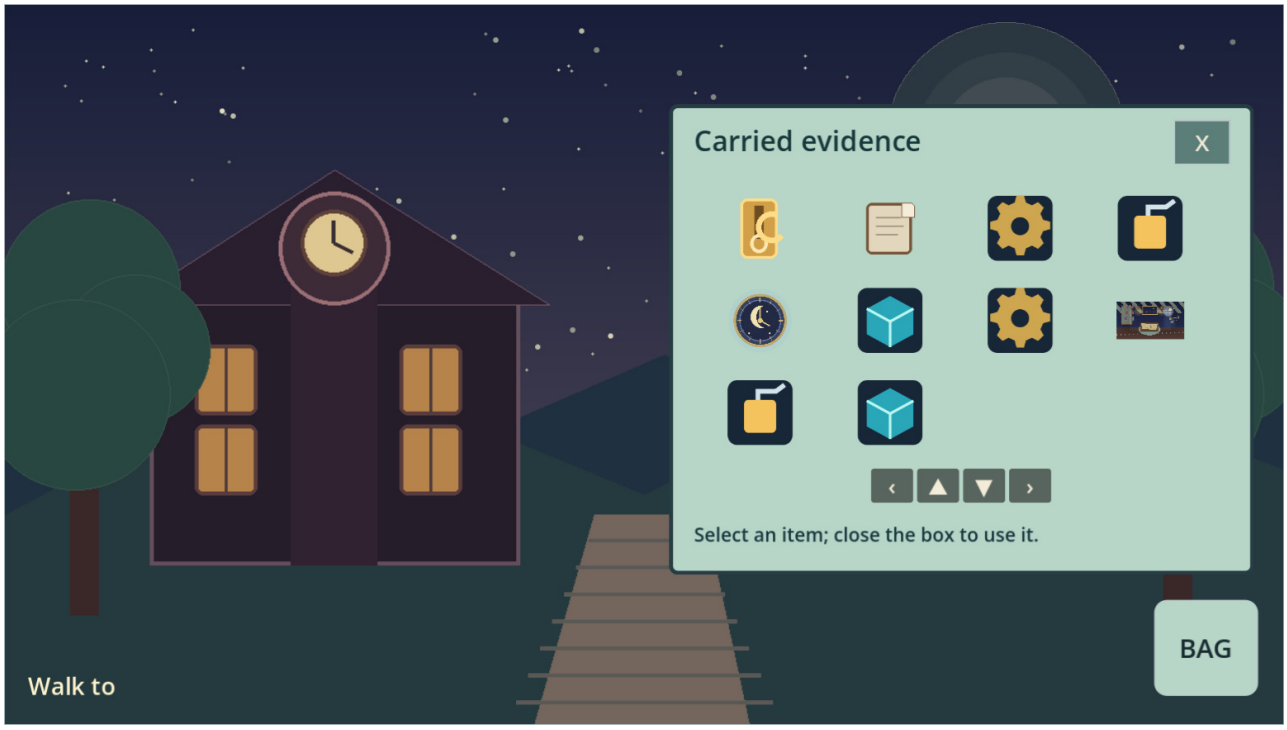

Figure 23. A floating box of items leaves the room visible around it.

Property	Value
Grid	4 columns; 3 rows; 12 items per page.
Slots	121 × 84; gaps 9 horizontal / 8 vertical; icon size 68.
Content	Icons on; text and quantity off.
Feedback	Hover #E4EFCD; selected #C2A578.
Behavior	Use Selected Verb on for inspecting with the Look cursor; combinations on when your puzzle defines them.
Ownership	Current player; one protagonist is enough for this recipe.

Use Walk as the item-selection state if you keep verb-sensitive inventory behavior. Alternatively, turn Use Selected Verb off for simple selection and add an explicit Examine selected command. Choose one consistent rule and tell the player how it works.

Give hotspots an active pose

Prepare two drawings of each cursor idea: an ordinary pose and a clear active-hotspot pose. The original’s eye opening is a good example of feedback that says “there is something here.” Your own symbol can do the same job without copying that drawing.

1. Import your ordinary and active cursor images as separate assets.
2. Create cursor definitions with the correct hotspots. Keep both cursor drawings aligned to the same tip or center.
3. On the relevant object, find Cursor While Hovering and assign the desired active cursor when appropriate.
4. Test the pointer entering and leaving the object, then pressing and releasing.
5. If a hover cursor should depend on the current verb, add that conditional logic explicitly. A single fixed hover-cursor assignment is not a full per-verb animated cursor system.

A two-pose change is enough for the basic tutorial. To animate several active poses, build a software pointer or another dedicated cursor controller; keep it separate from the gameplay verb logic and restore the system pointer when leaving the game.

> Avoid advertising a grab with the active cursor over an object that cannot be taken. Good cursor feedback follows actual interactions.

Create a matching case-file save screen

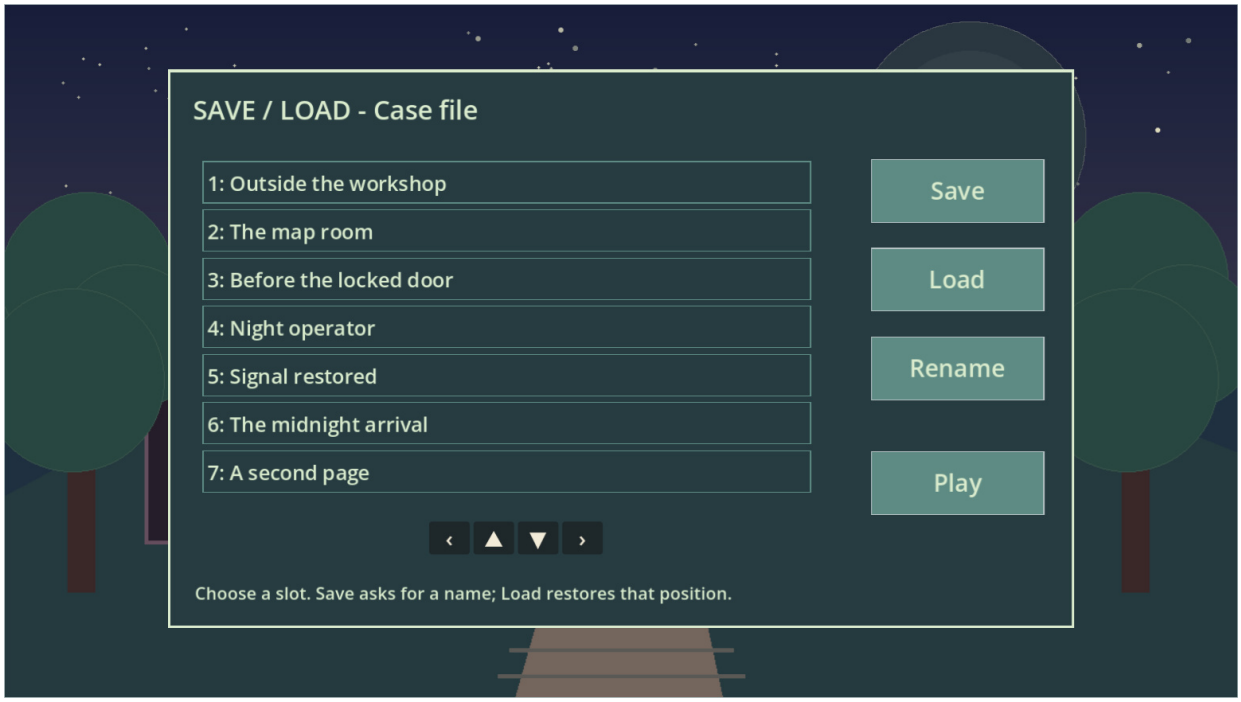

Figure 24. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	sam_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	sam_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #263C41; text #DBEDCE; hover #5E8B84; selected #E7BE84.

This is a portable PCA case-file adaptation. Historical releases differ in their save-menu shell; for example, the Macintosh manual describes Macintosh file dialogs. Keep your game’s save screen consistent across its supported platforms.

Animate a quick, light inventory popup

Motion	Recipe
Inventory opening	Scale for 0.18 seconds; Origin X 0.75, Y 0.47 on the full 1280 × 720 overlay.
Inventory closing	Scale for 0.12 seconds, using the same origin.
Inventory icon hover	A single brighter image or pose; keep the item centered.
Conversation topics	A brief state change on an icon; leave the reading area still.
Save screen	Cut; reserve longer motion for a special illustrated menu if desired.

The box’s scale is a presentation choice in this workshop, not a claim about the original’s exact animation. To draw a lid-opening box instead, keep the frame fixed and play a non-looping Image sequence before enabling the live slots.

When the player selects an item, decide whether the box remains open for combining or closes for world use. Provide a clear Close route. Automatic close-on-leave is optional; implement and test it rather than assuming Show GUI supplies it.

> The more expressive the cursor, the less extra pulsing and flashing the rest of the interface needs.

Test the cursor-first loop

1. Cycle through exactly the enabled five actions and back to Walk.
2. Look at one object, take another, talk to a character and use a mechanism.
3. Open the box, examine an inventory item, select it and close the box.
4. Use that item on a world target and test a valid inventory combination.
5. Cancel a selected item. Confirm that the next right-click has the intended cursor-cycling behavior.
6. Save, change a puzzle state, then load. Confirm that the right cursor and inventory state remain understandable.

For a touch or controller version, add accessible on-screen action choices. A mouse-only right-click cycle cannot be the only way to choose a verb on a touchscreen.

> Judge the interface while solving a short puzzle. If the player spends more time cycling past unnecessary actions than thinking about the puzzle, reduce the action set.

7. Beneath a Steel Sky: a hidden top strip

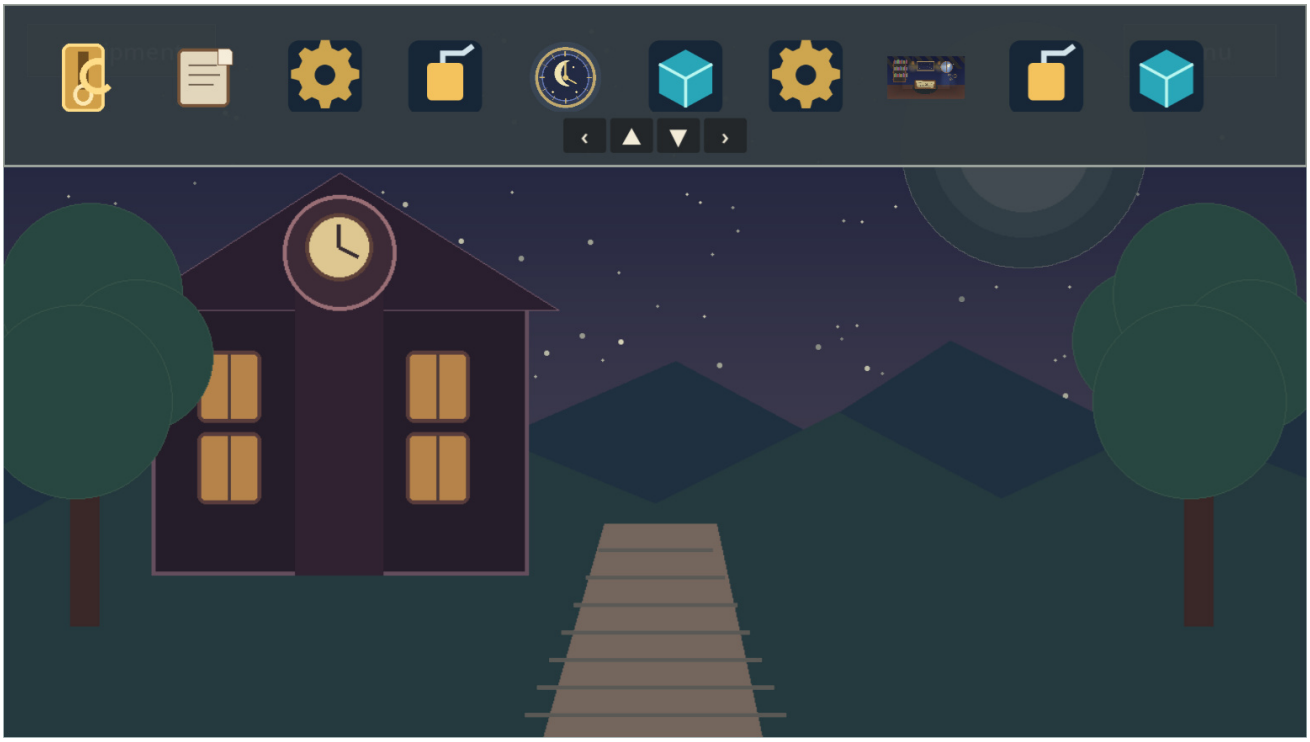

Figure 25. The equipment strip appears at the top while leaving most of the scene clear.

The 1994 interface uses a top inventory and a two-button distinction: examining an item and selecting/using it are different operations. Its original control panel provides saving and restoring. [4]

Use a small HUD, a top-edge inventory GUI and a square industrial save screen. We will add a short inventory-button adapter so that left examines and right selects the carried item, rather than accepting the widget’s ordinary button behavior.

> This chapter is about Beneath a Steel Sky, not the later 3D game Beyond a Steel Sky.

Create the quiet HUD and top inventory

Element	Position and size	Settings
bass_status	24, 655; 1100 × 44	Label, {target}; 24-pixel #E7DFBD.
Equipment opener	24, 20; 185 × 52	Show bass_inventory. Keep it while testing.
Menu	1100, 20; 150 × 52	Show bass_save.
GUI Window	0, 0; 1280 × 160	Set the actual window bounds to the strip; background/title off.
bass_inv_panel	0, 0; 1280 × 160	Background #343D42F7; border #9CA29C.
bass_items	16, 12; 1248 × 142	10 columns; 1 row; 10 items per page.
Slots	112 × 104	Icon size 76; hover #65695A; selected #8C7144.

The ten-slot enlarged strip is a workshop layout choice. It is intended to preserve the top-edge inventory idea without making the icons tiny at a modern display size.

Turn text off inside the slots if you want a picture-led strip. Keep item names and examination text available so that similar tools can be distinguished.

> Choose the inventory’s alpha deliberately. A barely visible item frame can be hard to use over bright machinery or white room backgrounds.

Open it from the top edge

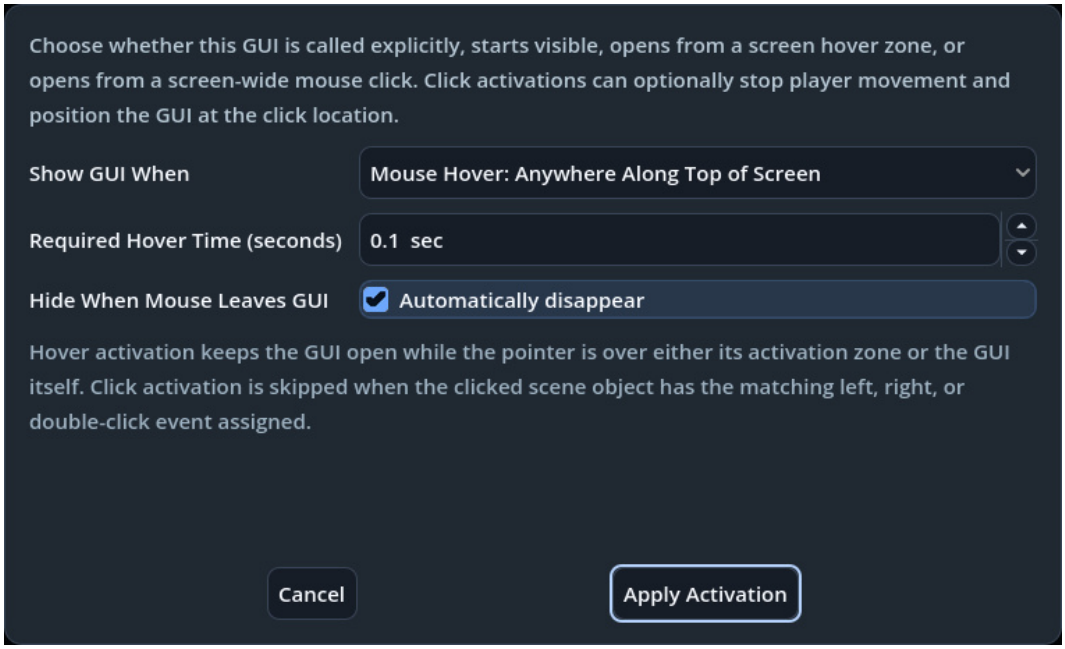

Figure 26. The activation zone can reveal the GUI and keep it open while the pointer is over the strip.

1. Assign bass_inventory as the Inventory Screen in Global Settings, so this is the active inventory rather than an unused alternative.
2. Select bass_inventory and click Activation.
3. Set Show GUI When to the top-edge hover mode.
4. Start with Required Hover Time of 0.10 seconds.
5. Enable Hide When Mouse Leaves GUI.
6. Run Play. Move to the top edge, then down over the items. The strip should stay available while the pointer is over it.
7. Move back into the room. Check that the strip closes without repeatedly opening and closing under the pointer.

Keep the Equipment button for touch, controller use and straightforward testing. It can open the same GUI explicitly. A screen-edge hover gesture should not be the sole route on devices without a mouse. The actual GUI window must be only as tall as the strip; an invisible 720-pixel window would still count as “inside” after the pointer leaves the drawn panel.

> Keep the top strip nonmodal. If it locks room input while merely appearing, you will change the feel of the original interaction pattern.

Set up left-look and right-use in the room

1. Set Project Settings → Interaction Mode to Direct. In Global Settings, turn off Right Click Clears Selected Inventory for this two-button tool-use recipe.
2. For an ordinary scene object, write its description. Left-click should give that description.
3. Enable the hotspot and define the useful action in the object’s Use verb interaction.
4. Add right_use from Chapter 12 to the object’s explicit On Right Click pointer event. The script sets Use and executes the clicked target.
5. For taking an object, put Add Inventory and the appropriate scene-object hiding/removal after the successful Use result.
6. Create a real item-on-object response for a selected tool, then test that response as well.

Direct mode and the original game’s complete two-button behavior are not identical in every built-in object/widget path. The explicit right-click script avoids leaving an ordinary object’s right interaction unspecified.

Attach the script to the clicked object’s pointer event, where the event context supplies entityId. Running it from an unrelated menu Button without a target context will correctly do nothing.

> Do not put the same script inside the object’s Use response: that would execute Use again from inside itself.

Make inventory left and right clicks agree

Install bass_buttons from Chapter 12 as a Global Script. Leave Use Selected Verb enabled on bass_items. The adapter listens to that widget’s item event before its built-in interaction runs.

Input	Adapter state	Result
Left-click an item	Clear an old selection; choose Look.	The built-in item interaction examines it.
Right-click an item	Clear an old selection; choose Walk.	The built-in item interaction selects it.
Hover	No selection change.	The pointer can inspect the strip without using anything.
Use a selected item on a target	The object’s explicit right-use route executes the target.	The matching item rule supplies the result.

The selected-item world route in this tutorial is right-click. The original can apply a carried item with either button in relevant contexts. Extending left-click selected-item use is a separate input rule; do not silently replace left-look everywhere.

> The script is scoped to bass_items. If you change that element ID, update the script too. It should not rewrite inventory clicks in every other interface in this book.

Build the industrial control-panel saves

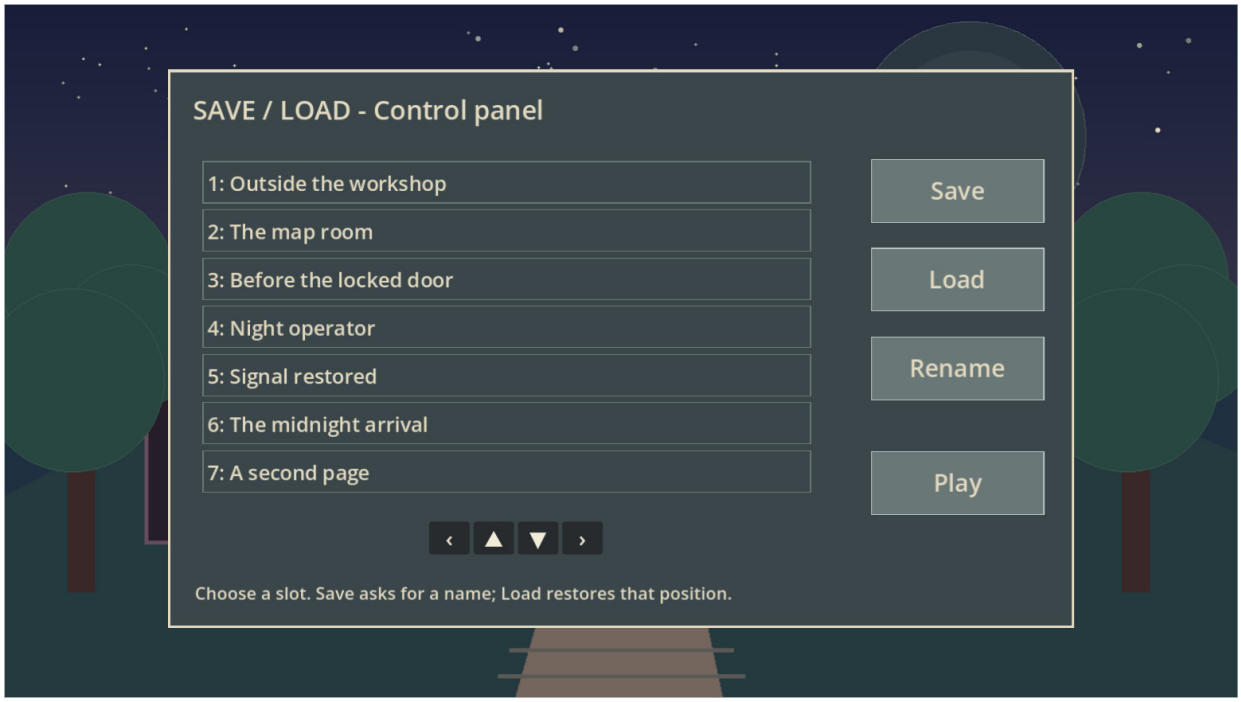

Figure 27. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	bass_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	bass_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #3B464B; text #E3DBC1; hover #6A7777; selected #E4D1A1.

Use a plain list and a restrained selected row. This workshop retains native PCA naming prompts; the original manual describes typing into the selected control-panel slot. Keep that distinction clear while testing.

Give the strip a short reveal animation

Phase	Chosen settings / behavior
Hover delay	0.10 seconds before revealing the top strip.
Open	Slide downward, 0.18 seconds.
While over the strip	Keep the GUI available and its slots stable.
Close	Slide upward, 0.14 seconds, after the leave condition.
Item click	A short held picture or color; selected state remains separate.
Control panel	Cut open and close.

For a stricter period feel, use Cut instead of Slide. The short slide is an optional workshop embellishment, not a required historical effect.

Do not add a moving “scanner” over the item icons if it makes their colors and shapes hard to read. If you animate a border, keep the animation on a separate decorative Image.

> A hover-open interface needs a comfortable travel path from the activation zone into its contents. Test it slowly and quickly, including the corners of the display.

Run the complete two-button test

1. Move to the top edge, then move through the entire strip without it collapsing.
2. Left-click an item. Confirm that it is examined and that no old item remains selected.
3. Right-click a different item. Confirm that this item becomes the selected tool.
4. Right-click a suitable world target. Confirm that the correct tool response occurs once.
5. Left-click an ordinary room object with no tool selected. Confirm that the description appears.
6. Open the control panel, save, change a puzzle state and restore the saved position.

If an item combines with another instead of being examined, check whether an earlier selection was cleared before choosing Look. That ordering is the reason for the adapter.

> Give a controller/touch user an explicit Equipment control and action choices. The visible result can keep the same industrial style even when the input method changes.

8. Monkey Island 2: the eight-icon verb HUD

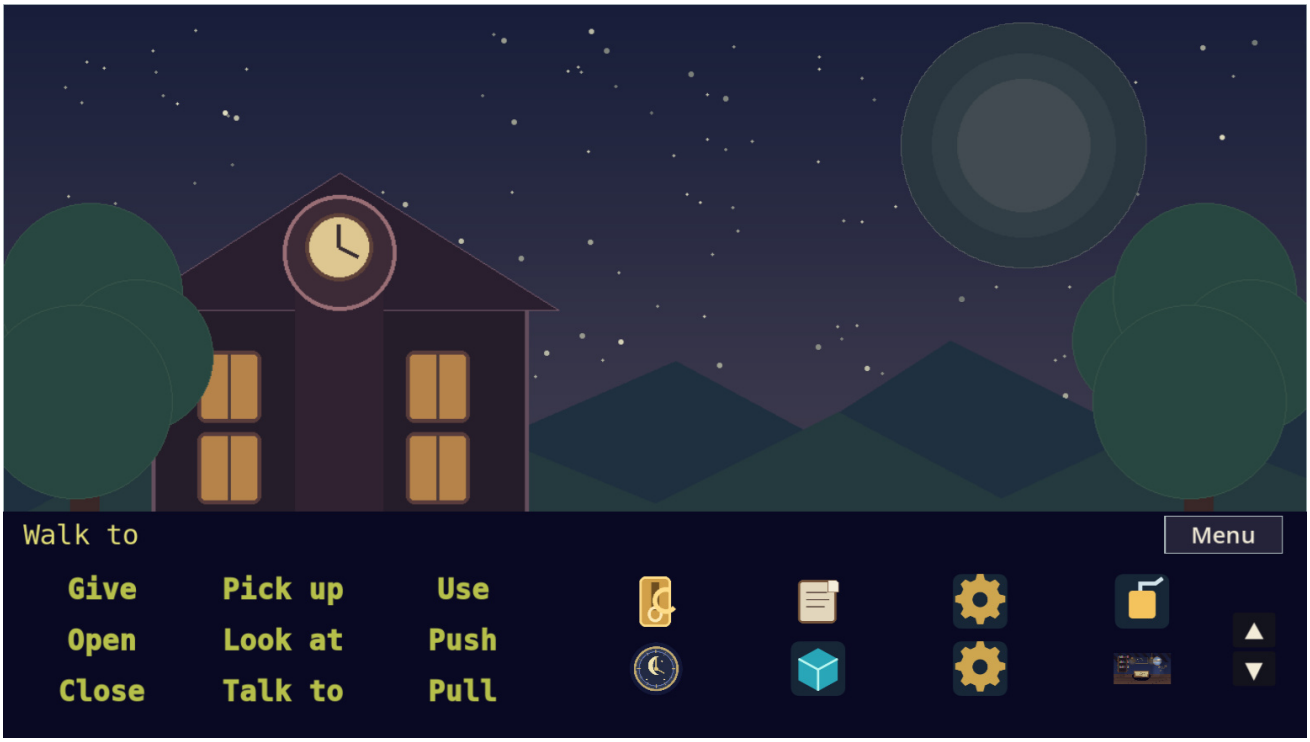

Figure 28. A dark lower band, nine word commands and an eight-icon inventory.

The original LeChuck’s Revenge uses a sentence line below the scene, verbs on the left and inventory icons on the right. Its manual describes eight visible inventory icons before scrolling is needed. This chapter follows that original layout family, rather than the Special Edition’s alternate interface. [5]

The construction resembles DOTT, but the palette, slot count and lack of a three-character portrait strip make a different interface. Build this version independently instead of leaving the DOTT portrait controls in place.

> The colors are a workshop adaptation: dark navy, muted yellow-green command words and warm selection feedback. Use your own font and room artwork for the finished game.

Position the sentence line and nine commands

Element	Position / size	Settings
mi_band	0, 498; 1280 × 222	Background #0A0924; Z Index 1.
mi_command	20, 503; 1090 × 36	27-pixel monospaced text; #E3DF77.
Verb bank	Start 12, 552	3 columns; X step 177; Y step 50.
Verb Buttons	171 × 46	28-pixel bold monospaced text; #B6C04A; square shape.
Inventory	557, 546; 700 × 174	Eight icon slots.
Menu	1140, 502; 116 × 37	Opens mi_save.

Use the same nine commands as the DOTT bank: Give / Pick up / Use, Open / Look at / Push, Close / Talk to / Pull. Each is an ordinary Button with its own Set Verb event.

1. Set the project to Verb Single Click.
2. Assign mi_hud to the main HUD role.
3. Keep Walk as the normal movement verb for empty ground.
4. Put useful responses on the actual scene objects, not on the command Buttons.

Fit eight icons without crushing them

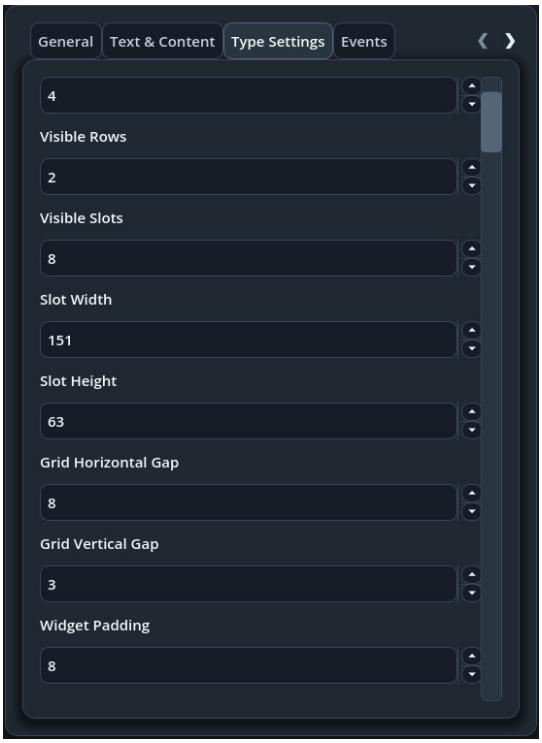

Figure 29. Four columns and two rows distinguish this inventory from the six-slot DOTT layout.

Property	Value
Element	mi_items, 557, 546; size 700 × 174.
Capacity	4 columns, 2 rows, 8 items per page.
Slot dimensions	151 × 63; gaps 8 horizontal / 3 vertical; padding 8.
Content	Show Icon on; Show Text off; icon size 56.
Navigation	Right-side Up/Down arrows; Left/Right disabled.
Feedback	Hover #3B3559; selected #786044.
Behavior	Use Selected Verb on; combinations enabled only where you define them.

Test nine carried items. It is easy to overlook a broken next-row command when the starting inventory fits entirely on the first page.

Write a useful sentence, then execute it

1. Create a useful object such as a locked chest. Give it Look at and Open responses.
2. Create Brass key as a carried item and add its matching item-on-chest response.
3. For a pair-command workflow, enable Allow Preposition on Use, enter with, and opt the key into that preposition.
4. Select Use, then the key, then the chest. Confirm that the key response runs and the chest’s state changes.
5. Install sentence_line from Chapter 12 to display the selected-item or pending-pair prefix in mi_command.
6. Try changing verbs midway through a pending command and cancelling an item. The caption and actual command should agree.

The original also highlights a suitable default verb over a target and supports a shortcut to use that default. A selected command color is not that feature. If you want an automatic default in your PCA game, configure or script that separate choice and check it against the target’s available interactions.

> Do not highlight Open over a locked mechanism while the click actually performs Look. Feedback and execution need to refer to the same rule.

Keep dialogue choices readable

A conversation temporarily needs a reading area. Build a dialogue GUI using the engine’s choice controls/groups and style it with the same navy background and light yellow text. Keep the ordinary verb bank out of the way while the choices are active.

1. Create a dialogue with at least three choices and a clear way to end the conversation.
2. Use the authored dialogue GUI role or system groups to display the current choices.
3. Give the choice area enough height for wrapped text. Do not size every choice for a short English phrase only.
4. Restore the exploration HUD when the dialogue closes.
5. Test a line that starts another conversation or changes a variable, then check the restored HUD state.

The sentence line should remain immediate and functional. Reserve typewriter text for spoken lines or messages if you want that effect, rather than making the player wait for a command caption to finish typing.

> Choice selection, selected inventory and selected verb are separate states. Closing a conversation should not leave a hidden choice control accepting the next click.

Build the captain’s text save list

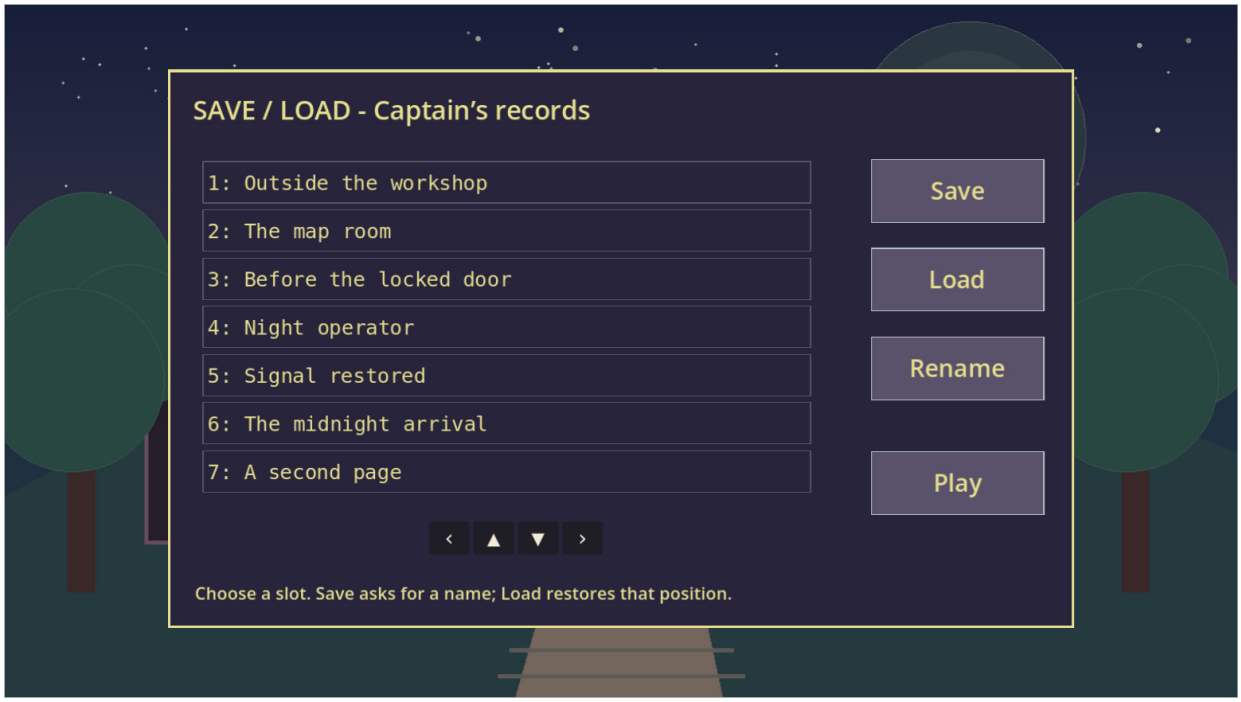

Figure 30. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	mi_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	mi_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #27243C; text #E8E091; hover #5A526A; selected #E8C566.

Use the text-list recipe for the original interface family. The workshop’s seven visible rows and twelve slots are deliberate layout choices. Keep the original-style command area separate from the modal save GUI.

Use restrained animation in the verb HUD

Part	Motion plan
Verb words	Immediate hover and held feedback. No sliding words.
Item icons	Normally static; animate an individual plot device only when it conveys information.
Scrolling	Replace the visible row promptly; do not move the hit targets away from the pointer.
Dialogue area	Cut or a very short fade. Keep text stable once it is available.
Save menu	Cut open and close.
Optional special object	Use a short non-looping Image sequence without moving its containing slot.

If a small item’s animation needs more space, show an enlarged examination GUI. It is usually better than making every inventory slot larger or trying to read moving details at icon size.

> An old interface does not need a CRT effect to look convincing. Font choice, spacing and a consistent palette do more of the work. Apply any optional shader lightly and check every command word.

Test the island-style command loop

1. Use all nine commands on appropriate objects and inspect their fallback responses.
2. Move on empty ground after using a command. Confirm that the movement rule feels consistent.
3. Select a key, use it on a target and cancel it without leaving a stuck item cursor.
4. Carry nine items and navigate both rows of the inventory.
5. Start and end a dialogue, then make another room interaction.
6. Save, use a consumable item and open a mechanism, then load. Check that the saved inventory and mechanism state return.

Keep the completed HUD as a visual baseline. Before adding decorative effects, take a plain screenshot and make sure every command can be read without moving the pointer.

> This chapter creates the original word-bank approach. The later three-choice coin is built next; the two designs should not quietly share incompatible input settings.

9. Curse: the action coin, chest and logbook

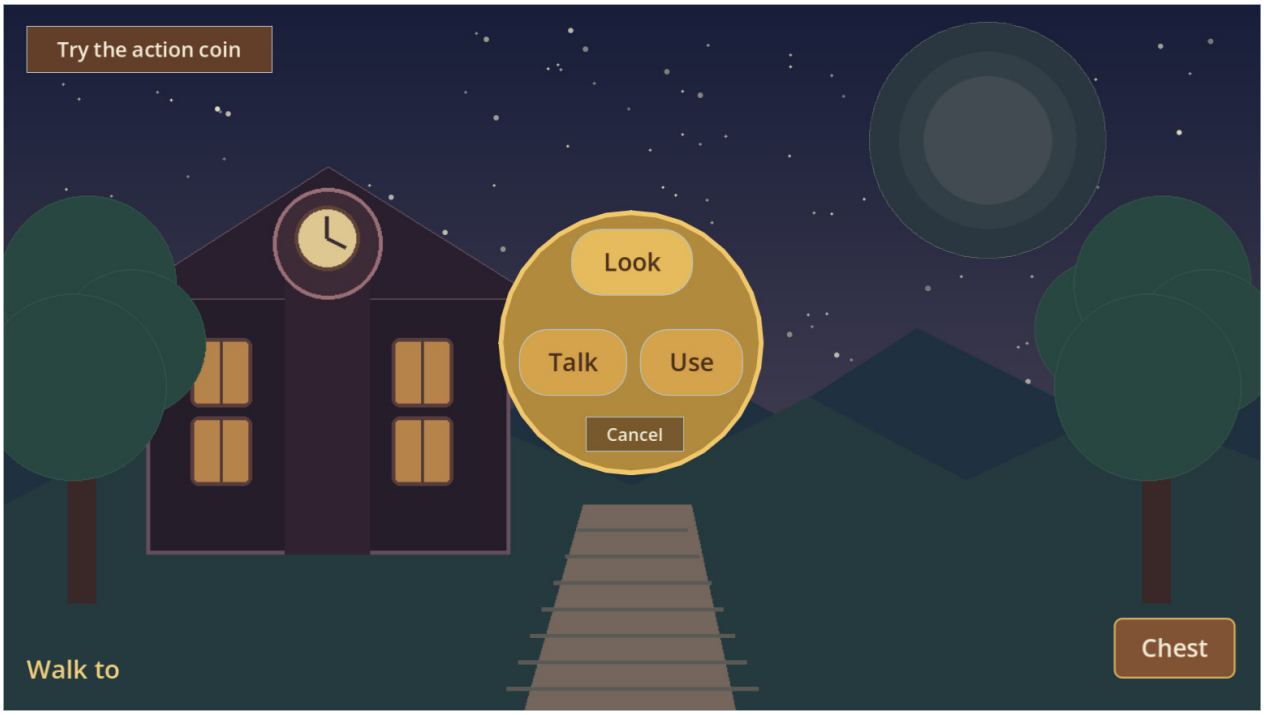

Figure 31. An authored three-choice coin with original workshop styling.

Curse’s original action coin uses three animated symbols: a hand for physical actions, a skull for examining and a parrot for mouth-related actions. Its hold-and-release gesture and illustrated inventory chest distinguish it from Monkey Island 2. [6]

The first PCA version uses an authored GUI and Verb Direct: click a target, then choose Look, Use or Talk. After that works, the optional curse_release global script adds held-button release selection. The workshop uses readable word captions where your finished game could use original icons.

> PCA’s stock Hotspot Then Verb Coin is a useful alternative, but its stock radial layout is not the same authored gold artwork shown here.

Build an authored three-choice coin

Element	Position and size	Style / event
curse_coin_disc	504, 210; 270 × 270	Panel; #B28A3D; gold border; radius 135.
curse_coin_look	578, 229; 124 × 68	Rounded Button; Click → Set Verb → Look.
curse_coin_talk	525, 331; 110 × 68	Click → Set Verb → Talk.
curse_coin_use	648, 331; 105 × 68	Click → Set Verb → Use.
Cancel	593, 420; 100 × 36	Run cancel_target_menu (Chapter 12).

1. Create the hidden GUI curse_coin. Keep it nonmodal for this target-first setup.
2. Make the circular backdrop a Panel below the Buttons in Z Index.
3. Use three well-separated Buttons. Keep the artwork inside stable, generous clickable rectangles.
4. Add normal, hover and clicked images when your own icon art is ready.
5. Keep a visible Cancel control while testing.

A circular backdrop does not make the three Button hit areas circular. Leave enough separation that the visible choices and their clickable rectangles remain easy to distinguish.

Let the engine remember the target

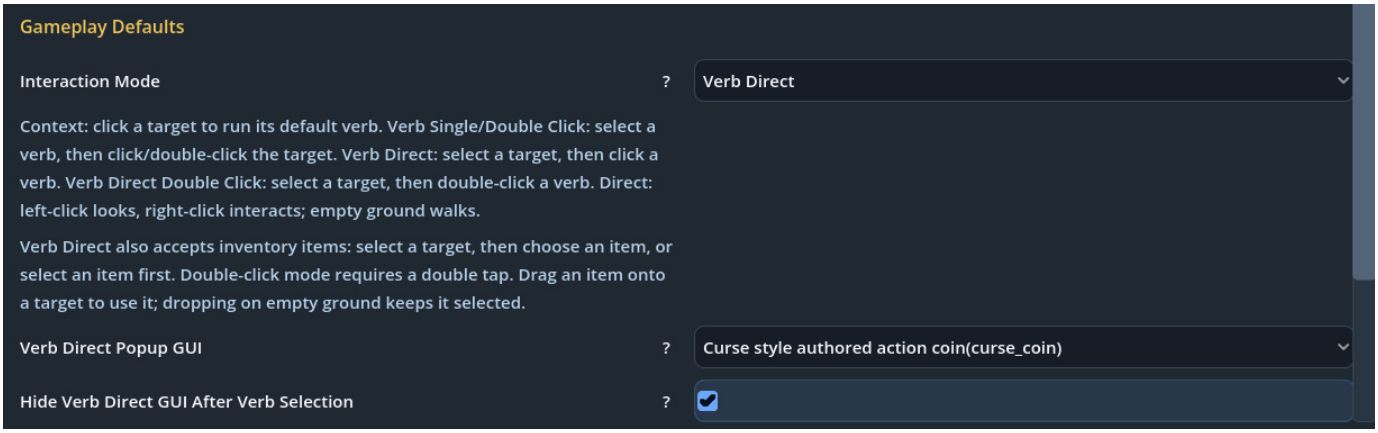

Figure 32. Verb Direct can open an authored popup and execute a selected verb on the stored target.

1. Set Project Settings → Interaction Mode to Verb Direct.
2. Set Verb Direct Popup GUI to curse_coin.
3. Enable Hide Verb Direct GUI After Verb Selection.
4. On the room object, enable its hotspot and give it Look, Use and Talk responses as appropriate.
5. Run Play. Click the object, then the coin’s Look Button. Confirm that the engine examines the original object.
6. Repeat with Use. Do not retarget the object by looking at the pointer position over the coin.

Use is the hand-like combined physical action in this recipe. An ordinary takeable object can have Add Inventory and scene-object hiding in its Use result; a door can open from Use; a mechanism can operate from Use.

> The first version uses the authored popup position and a second click. It does not yet implement the original hold threshold, near-pointer placement or release-to-select gesture.

Add the hold-and-release behavior

1. Keep the working Verb Direct setup. Do not change it to a second competing coin system.
2. Create the curse_release GDScript record from Chapter 12 and enable Global Script.
3. Keep the exact element IDs curse_coin_look, curse_coin_use and curse_coin_talk, or update the script’s choice table.
4. Press on a room target and keep the mouse button held. Move onto a coin choice, then release.
5. Confirm that one verb executes and the popup closes.
6. Repeat but release outside the choices. It should cancel without acting on a different object.

The adapter deliberately adds release selection to the engine’s existing stored-target mechanism. It does not add the original delayed hold threshold or automatically move the authored coin under the pointer. Those are separate refinements, not hidden results of installing the script.

Keep the two-click route for touch, controller use or a player preference. A held mouse gesture should not be the only accessible route through a puzzle.

> Test rapid press/release, cancelling and changing scenes. The script should never carry an old target into the next interaction.

Draw the chest around a live inventory

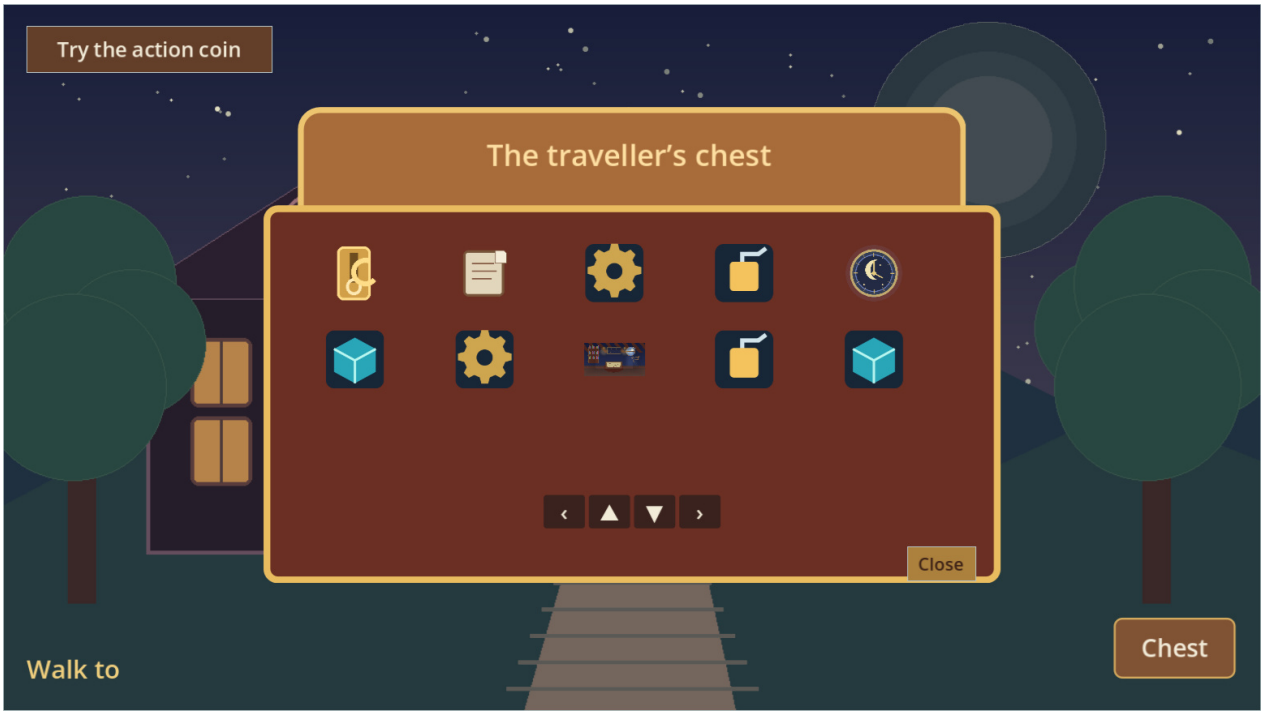

Figure 33. The frame and lid are decoration; the icons are a live Inventory Grid.

Part	Position / size	Starting settings
Lid panel	300, 105; 680 × 110	Warm wood; gold border; rounded top corners.
Body panel	265, 205; 750 × 385	Dark red wood; 7-pixel border; radius 14.
curse_items	287, 226; 706 × 316	5 columns, 3 rows, 15 items per page.
Slots	125 × 80	Gaps 7 / 8; icon size 62; icon-only.
Close	920, 552; 70 × 36	Hide curse_inventory.

Assign curse_inventory to the inventory-screen role. Begin with Use Selected Verb off for simple item selection in the target-first workflow. Leave item preposition opt-ins off for that direct-target selection route.

Select an item, combine or use it

1. Open the chest and choose a carried item. Confirm that its item cursor or selected state appears.
2. Close the chest with its Close control and click a matching world target.
3. Alternatively, click a target first in Verb Direct, open the inventory and choose the item to apply it to the stored target.
4. For combining, define the item-pair result and enable the intended combination route.
5. Test cancellation and an invalid target. The wrong pair must not silently produce the successful item.
6. Only add close-on-leave or long-press inventory coin behavior after these ordinary routes work.

The original chest can disappear when the selected item leaves it and supports a longer press for the action interface. A fixed Close Button is the reliable starting route here. Those exact leave/hold refinements need their own input controller; the chest artwork does not create them.

> Keep the contents as real slots. Painting the items into one chest image prevents ordinary pickup, removal, selection and inventory rules from updating them.

Make the two-page illustrated logbook



Figure 34. Two independent one-column widgets preserve the left-page 1-3 / right-page 4-6 order.

The original save logbook shows six boxes on a spread, with 1-3 down the left page and 4-6 down the right. A single row-major 2-column grid would fill in a different order. Use two one-column widgets for this version. [6]

Part	Settings
Cover	92, 38; 1096 × 636; #5D3026 with a warm gold border.
Left page	116, 62; 515 × 578; #E7D3A0.
Right page	649, 62; 515 × 578; same paper color.
Left widget	curse_save_saves_0 at 134, 124; 480 × 431; slots 1-3.
Right widget	curse_save_saves_1 at 664, 124; same size; slots 4-6.

Configure the six save cards

Property	Each page widget
Grid	1 column; 3 rows; 3 slots per page; no internal arrow buttons.
Slots	448 × 126; vertical gap 8; padding 7.
Screenshot	On; left placement; 174 × 98; cover fit.
Text	{slotNumber}: {name}; empty template {slotNumber}: {empty}.
Colors	Normal #DAC28B; hover #EFDBA5; selected #C58D5A; selected outline #B3372B.
Selection	Click selects; selected-save variable selected_save; double-click none.
Safety	Prompt for Name and Confirm Overwrite on.

Both widgets write the same selected_save variable so that one pair of Save/Load Buttons can serve either page. Use the book_save dispatcher in Chapter 12 to route the explicit slot number to the right widget. Create book_load by changing its command to load-slot.

The original boxes emphasize a snapshot with its title. This enlarged workshop version places the screenshot beside the caption for readability. To move the caption underneath, use top screenshot placement and increase the slot height accordingly.

> A preset name is not a guarantee of this layout. Set the ranges, column order and screenshot dimensions yourself.

Add the next spread without rebuilding slots

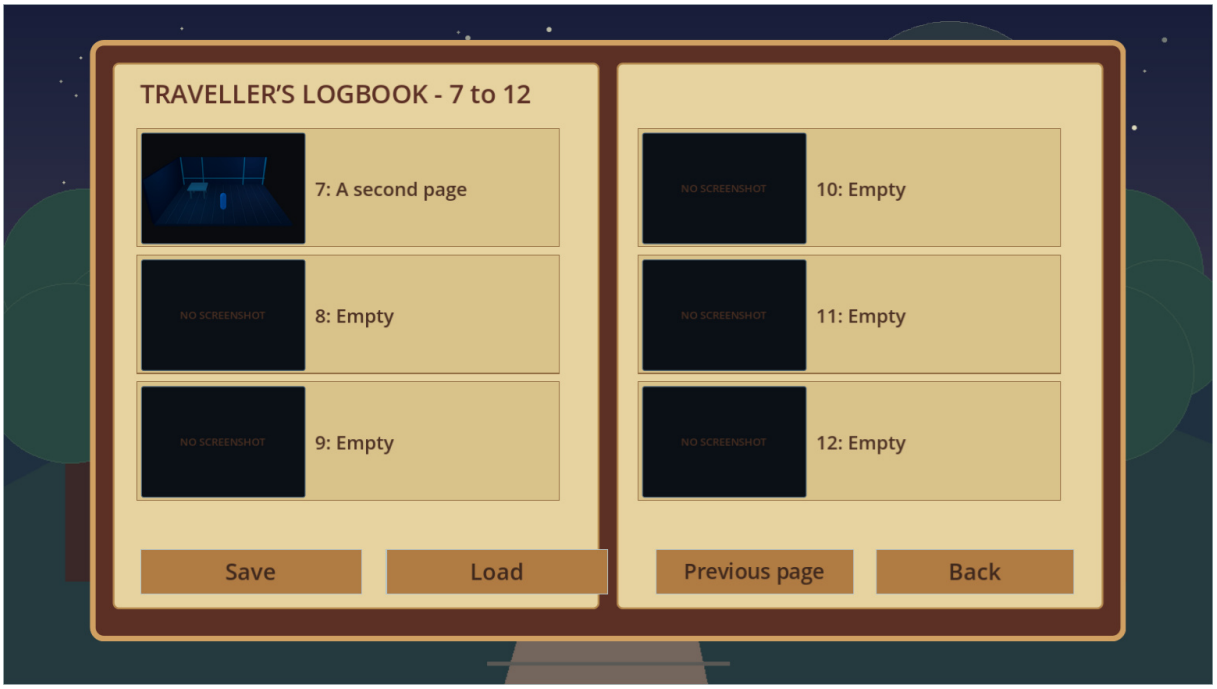

Figure 35. The second authored spread covers slots 7-12.

1. Clone the complete spread into curse_save2. Give every cloned element a new unique ID.
2. Set the left widget to slots 7-9 and the right widget to 10-12.
3. Set Next page on curse_save to hide curse_save and show curse_save2.
4. Set Previous page on curse_save2 to hide it and show curse_save.
5. Before each page change, clear selected_save to -1 so a Save/Load press cannot act on an unseen row selected on the previous spread.
6. Keep the same shared Save/Load dispatcher. Test a slot in the left column and one in the right column of each spread.

This fixed twelve-slot implementation uses authored pages. It avoids changing a live widget's source range repeatedly. Add more authored spreads or a carefully tested general paging controller if your game needs many more slots.

Animate the chest and logbook honestly

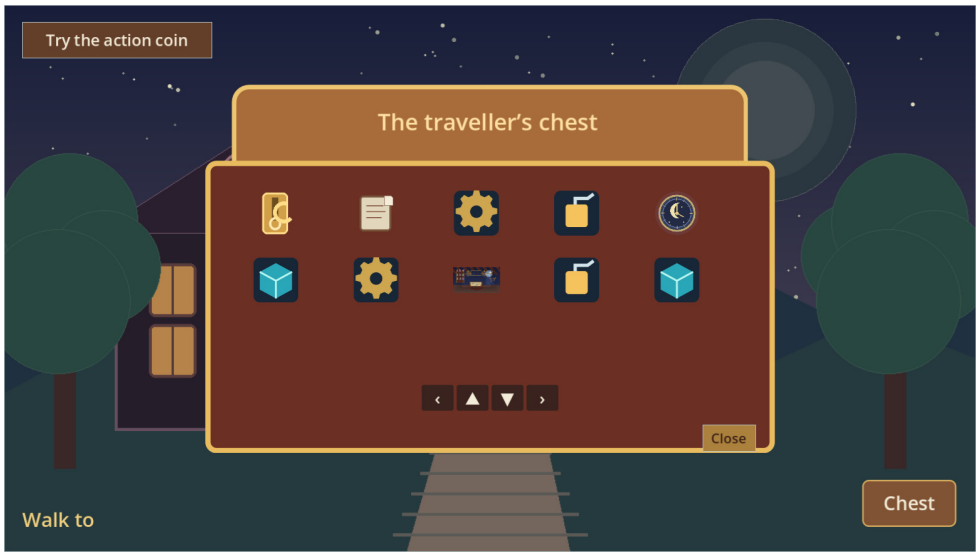

Figure 36. A sampled frame of the stock scale entrance. This moves the GUI; it is not a wooden-lid animation.

Part	First working version	Drawn-animation upgrade
Coin	Scale open 0.12 s; Cut close.	Change icon pose on hover; add a dedicated sequence for multi-frame acting.
Chest	Scale open 0.22 s; close 0.16 s.	Draw 5-7 aligned lid frames; play once; reveal live slots when open.
Logbook	Scale open 0.28 s; close 0.20 s.	Draw an opening cover sequence separately from the live save cards.
Page change	A short cut or fade between stable spreads.	Hide live cards; play a page-turn Image once; reveal the destination spread.

For a page-turn drawing, keep the paper edge, spine and outer book bounds aligned. The animation is a visual bridge. The saved slot data stays in the Save Game widgets.

> There is no automatic page-curl command in these recipes. A scale effect, comic-panel transition and drawn page turn are different effects.

Test the entire coin/chest/book sequence

1. Click a target, then choose each appropriate coin action. Confirm that the target stays the same while choosing.
2. If using curse_release, hold, move and release over a choice; then repeat a cancelled release outside it.
3. Select a chest item, use it on a target, cancel it and test a valid item combination.
4. Save in slots 1, 4, 8 and 11 so that both columns on both spreads are covered.
5. Cancel a name prompt and an overwrite prompt; verify the old save survives.
6. Change spreads after selecting a slot. The shared selection must be cleared before Save or Load can act again.
7. Restore a saved game and confirm the actual puzzle, inventory and character state.

Keep a reduced-motion route that uses a Cut or short Fade. The player should not have to watch the chest lid or page turn every time an item or save is needed.

> Finish the input and save behavior first. Then replace the temporary wood and paper Panels with your own illustrated assets while keeping the same element IDs and actions.

10. Resonance: items and memories

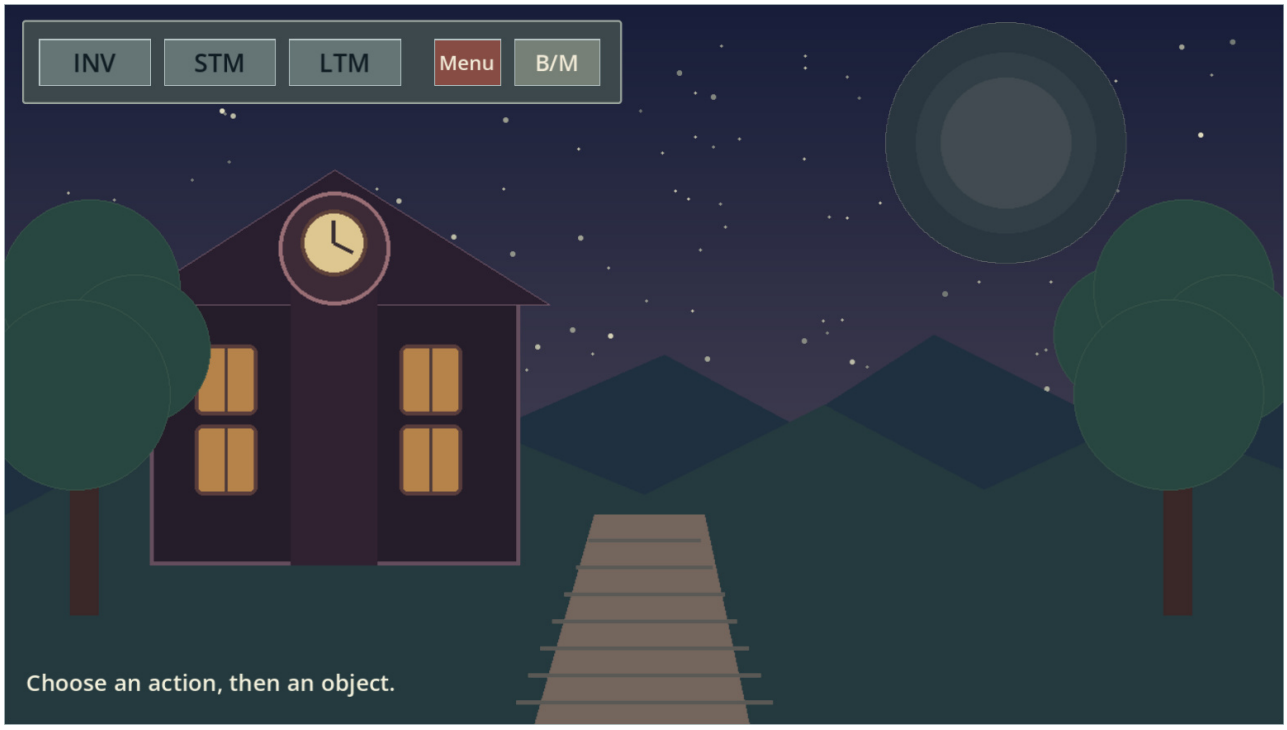

Figure 37. A compact console with distinct INV, STM and LTM controls.

Resonance’s distinguishing ideas are its four controllable characters and its short- and long-term memory system. Those memories can become conversation topics. [7]

Build physical inventory with an Inventory Grid, and memories with List Boxes backed by saved arrays. Do not represent every remembered object as a physical item that characters can give away.

> This is a working click-based PCA memory adaptation. It does not claim to add the original game’s arbitrary world-object-to-memory drag gesture automatically.

Build the compact console

Element	Position and size	Use
res_console	18, 16; 600 × 84	Metal-gray Panel; #3C484B; square corners.
INV	35, 35; 112 × 47	Show res_inventory.
STM	160, 35; 112 × 47	Show res_stm.
LTM	285, 35; 112 × 47	Show res_ltm.
Menu	430, 35; 67 × 47	Show res_save.
Character control	510, 35; 86 × 47	Open your four-character selector, or test one switch.
res_note	22, 658; 1230 × 39	Label reading $$ui_note.

Give the console buttons separate hit areas. A metal frame can surround them, but it should not obscure which part opens which panel.

The screenshot’s B/M control is a test shortcut. For a finished four-character game, use a character-choice popup with one portrait per available character. Keep unavailable characters out of the active choice set.

> For a screen-edge reveal, use a top-left hover activation GUI. For a precise fly-out anchored to a small console tab, use explicit button/hover events and test the leave behavior; a corner activation zone is not the same shape as one small tab.

Keep physical items in the inventory



Figure 38. Physical items remain a normal per-character inventory.

Property	Value
res_inventory	Hidden, nonmodal GUI; panel at 24, 110; 358 × 550.
res_items	40, 176; 326 × 460.
Grid	1 column; 5 rows; 5 items per page.
Slots	282 × 73; icons 58 pixels; name text on, 20 pixels.
Feedback	Normal #3D5658; hover #58757A; selected #9C704E.
Owner	Current player; per-character project inventory.

Use the ordinary item-on-world and item-on-item rules for physical items. A wrench can move between characters; a remembered conversation is handled by the memory arrays instead.

Create persistent memory stores

1. Create array variables named stm_tester and ltm_tester for the first character. Give them empty arrays as their starting values.
2. Create a short- and long-term array for each other playable character, using that character’s actual ID after the underscore.
3. Create the display arrays memory_stm_active and memory_ltm_active.
4. Create the string variable memory_topic for the currently chosen topic.
5. Add memory_tools from Chapter 12. Its remember function stores an ID and caption once, under the active character’s memory key.
6. Call its refresh function after a character switch and after restoring a saved game, so the visible lists match the active owner.

Store	Meaning
stm_ + character ID	Short-term observations deliberately remembered by that character.
ltm_ + character ID	Longer-term story events learned by that character.
memory_stm_active / memory_ltm_active	Display copies for the currently controlled character.
memory_topic	The selected topic ID, not a piece of physical inventory.

The sample short-term list keeps the most recent ten entries by dropping the oldest. That is a simple workshop policy. Use an explicit replacement choice if your puzzle needs the player to decide what to forget.

Display short-term and long-term memories

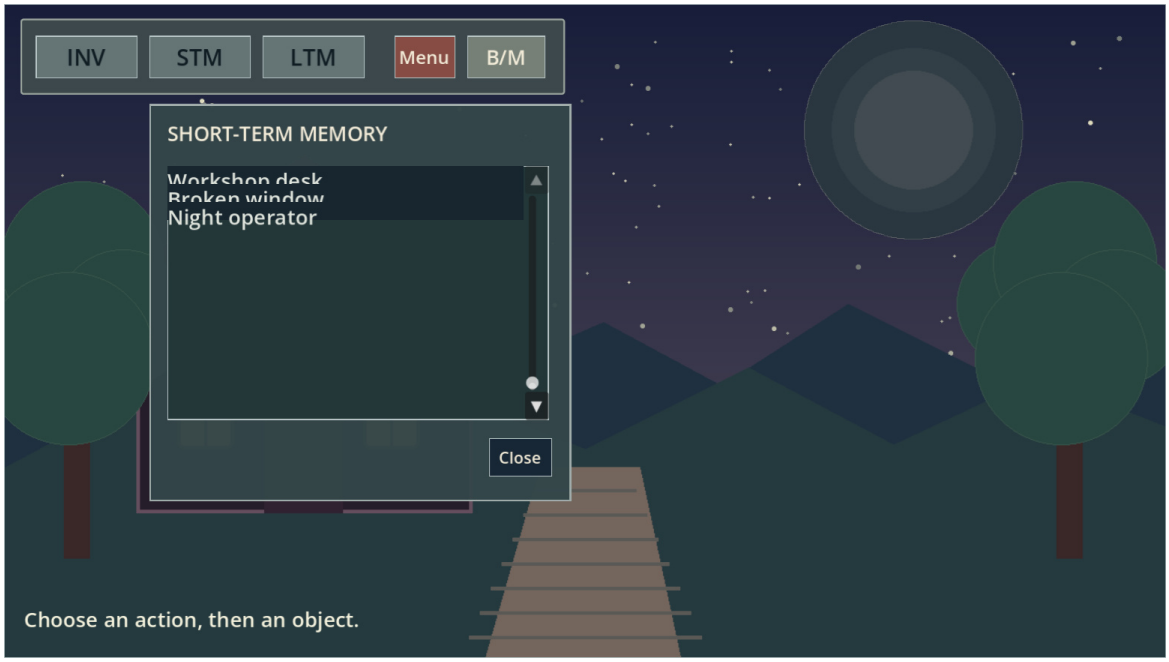

Figure 39. The STM panel reads a saved array rather than the physical inventory.

Control	Source configuration
res_stm_list	List Source: array; variable memory_stm_active.
res_ltm_list	List Source: array; variable memory_ltm_active.
Item record	id is the stable topic; text is its readable caption.
Selection Variable	memory_topic.
Close Button	Hide the corresponding memory GUI.

Do not add remembered entries only with a temporary List Box Add Item call and assume they are your permanent game state. Keep the authoritative memory in saved variables, then let the array source rebuild the presentation.

> Character ownership is part of the clue. Decide which characters witnessed an event, then add that long-term entry only to the intended owners.

Remember an object and use it in conversation

1. Give the broken window a Remember or relevant examination action.
2. From that action, call memory_tools.remember with topic window, caption Broken window and long_term false. Use a normal wrapper script if you are not calling a parameterized function node.
3. Open STM and choose Broken window. Its ID should be written to memory_topic.
4. Add a dialogue choice such as Ask about selected topic, or a dedicated Ask Button near the conversation choices.
5. Check memory_topic with an If Variable action. If it is window, run the appropriate dialogue line or dialogue graph.
6. Provide a sensible response for an empty or irrelevant topic. Do not make the conversation silently stop.

To record a long-term event, call the same helper with long_term true when the story event occurs. Use a stable topic ID even if the displayed caption is translated.

> A click-to-remember command is fully usable. The original drag-from-world gesture requires a drag source, remembered-object metadata and a drop target; it is not supplied by the List Box’s appearance.

Save the investigation and its memory data

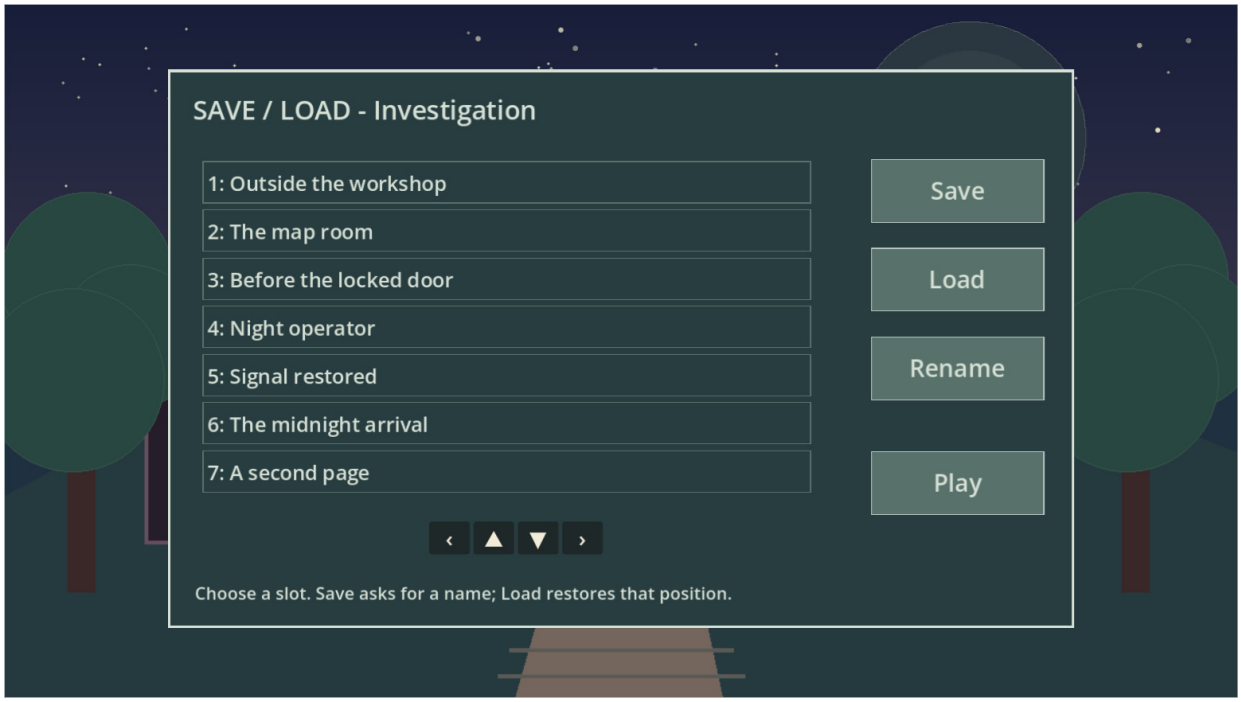

Figure 40. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	res_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	res_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #283D40; text #D7E1D6; hover #58716B; selected #D69C71.

Use a compact text menu for the workshop. The important difference is what is saved: current character, each inventory and the authoritative memory arrays. Do not save only the titles currently drawn in the console.

Refresh the owner after switching or loading

1. Use a character-choice Button to run a wrapper that switches the player and then calls memory_tools.refresh.
2. The memory_switch example switches to assistant. Make equivalent wrappers for the other recipients, or expose a recipient parameter through your function definitions.
3. When a save is loaded, refresh the active memory arrays from the restored active character. Use a global scene-loaded/player-changed hook if you want that refresh to be automatic.
4. Clear memory_topic after a character switch so that a previous character’s selected topic is not reused accidentally.
5. Check both panels after switching: STM and LTM must describe the same active owner.

The two active display arrays are convenient views, not the full record of all characters’ knowledge. Keep stm_character and ltm_character as the stored source of truth.

> The sample refresh function copies the saved arrays. A later change to one character’s memory should not mutate another character’s array through a shared reference.

Animate the console without hiding the clues

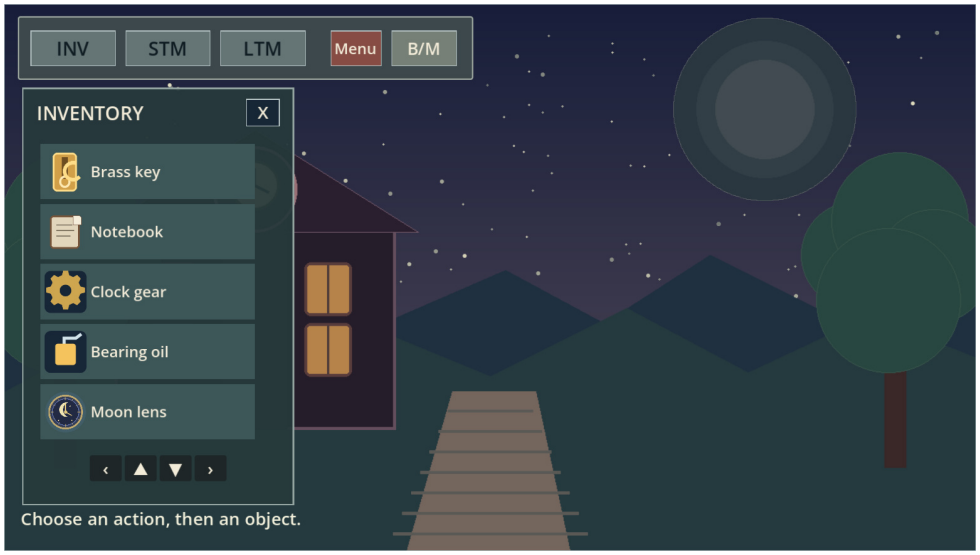

Figure 41. A short slide reveals the inventory column below the console.

Part	Motion / feedback plan
Inventory column	Slide down 0.18 seconds; slide up 0.12 seconds.
STM and LTM panels	Use the same timing and visual alignment.
Selected topic	A steady highlight. Keep the text readable.
New memory	One short icon/label indication, then a stable stored entry.
Active character	A steady indicator; any pulse is an optional notification.
Save menu	Cut; keep confirmation and naming prompts still.

Keep the entry text at a readable size while the panel moves. The point is to reveal a useful tool, not to make the player chase it. Use a static alternative for reduced motion.

> Opening one memory panel should not leave another full panel covering its controls. Either arrange them deliberately or hide the previous panel when the new one opens.

Test knowledge as well as possessions

1. Remember the broken window twice. It should appear once.
2. Record one long-term story event. It should be in LTM, not the physical inventory.
3. Switch to another character. Their memory lists should be independent.
4. Select a topic and ask about it. Check the correct dialogue branch and an irrelevant-topic fallback.
5. Add more than ten short-term entries and confirm your chosen replacement policy.
6. Save, remember a new clue and transfer an item, then load. Check every affected character’s saved knowledge and possessions.
7. Restore while a different character was previously active. Refresh the visible arrays and clear an obsolete selected topic.

If a clue disappears after loading, look first at the authoritative variables and the refresh hook. Repainting the List Box is not a repair for missing stored knowledge.

> Once the small version works, expand the clue catalog. Add a new topic because a puzzle uses it, not merely because every object can have another row.

11. Gemini Rue: actions around a target

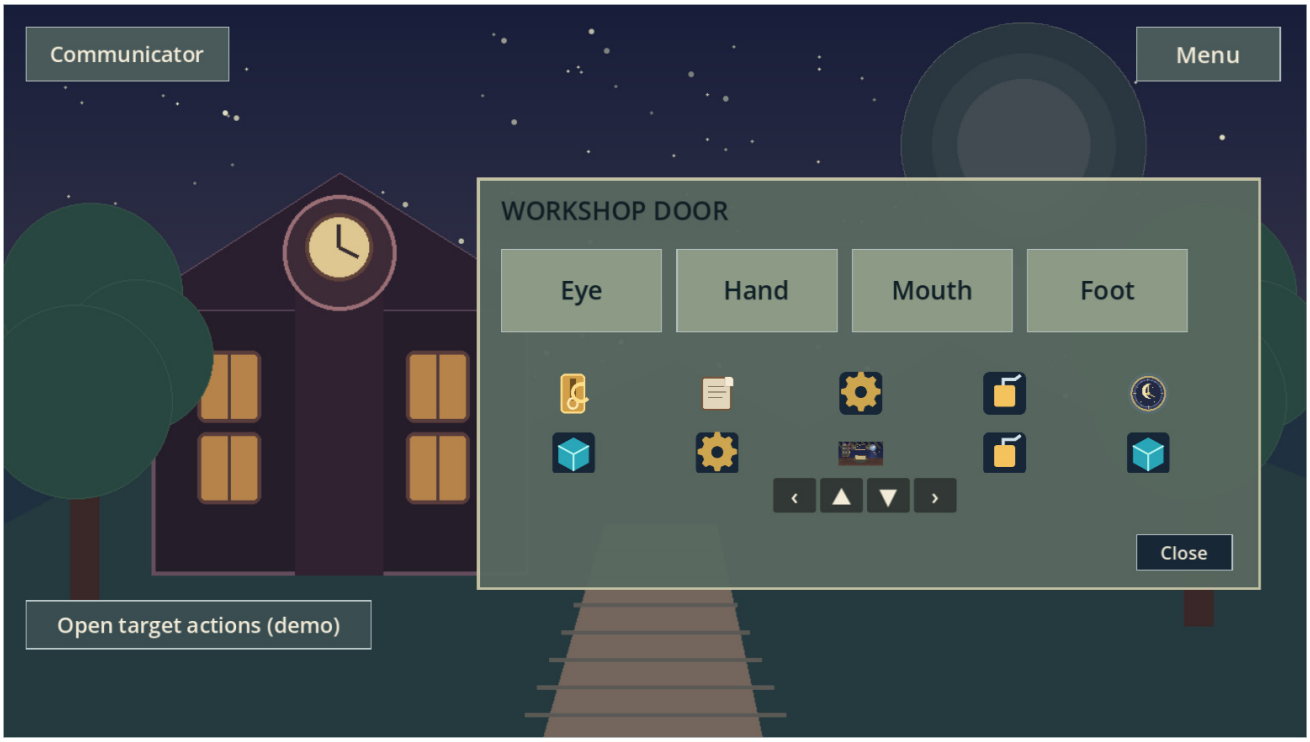

Figure 42. Four action Buttons above the carried items in one target-action panel.

Gemini Rue’s target menu groups eye, hand, mouth and foot actions with the inventory. The communicator provides another investigation surface. [8]

PCA’s Verb Direct mode is a useful starting point because it stores the chosen target and accepts either an action or an inventory item next. The first workshop version uses that native target-first route and a fixed authored popup position.

> The original favors right-click for the target panel. Native Verb Direct can open it from either mouse button over a target. Restricting the opener to right-click is a separate input refinement; changing the panel artwork does not enforce it.

Build four fixed action targets

Element	Position and size	Purpose
gem_context_panel	465, 170; 770 × 405	Background #596960F2; pale square border.
Target heading	489, 183; 600 × 40	Name the target or show a clear context heading.
Eye	489, 240; 158 × 82	Set Verb → Look.
Hand	661, 240; 158 × 82	Set Verb → Use.
Mouth	833, 240; 158 × 82	Set Verb → Talk.
Foot	1005, 240; 158 × 82	Set Verb → your custom Foot verb.
Inventory	485, 347; 722 × 160	Two rows of carried items.

1. Create gem_context as a hidden, nonmodal GUI.
2. Set the four Button captions first, then replace them with your own icons if desired.
3. Keep all four hit rectangles equally generous. A small foot drawing does not need a tiny clickable target.
4. Add a Close or Cancel Button. Run cancel_target_menu from Chapter 12 so cancellation clears the stored target as well as the visible panel.

Configure target-first actions and inventory

1. Set Project Settings → Interaction Mode to Verb Direct.
2. Assign gem_context as the Verb Direct Popup GUI.
3. Enable Hide Verb Direct GUI After Verb Selection.
4. Make each action Button use the Set Verb event behavior with its actual verb record.
5. Set Use Selected Verb off on the inventory for the simple direct-item route. Avoid starting a pending item preposition in this popup.
6. Click a scene target, then choose an action. Repeat by choosing a matching inventory item instead of an action.

The engine’s stored target is the important part. Once the panel is open, the pointer is over a GUI Button, not over the original object. Do not run a fresh world raycast at the Button’s screen position to decide which object to use.

> For this setup, the popup and inventory must remain usable by the inventory input rules. A new unrelated modal GUI can block item selection; keep the target-action popup nonmodal and use the designated popup role.

Give each action a real meaning

Action	Example response	What you must author
Eye / Look	Read a label or inspect a scanner.	Description or Look interaction.
Hand / Use	Open a mechanism or collect an item.	Use result, inventory grant and scene-state change.
Mouth / Talk	Speak to a person or call through a door.	Talk response or dialogue.
Foot	Kick a weak panel or climb a suitable ledge.	A custom verb result, movement/animation and any required checks.

Adding a Foot verb does not create a general climbing or combat system. For a small climbing interaction, author the movement destination, a suitable animation and a state check. Keep that puzzle behavior separate from the menu Button that requests it.

Give unavailable actions useful feedback or a clear unavailable presentation. Do not leave the player guessing whether the foot icon failed to register or whether the panel cannot be kicked.

> A target-first action must still check its current world state. The target may have been removed, disabled or changed while a menu was open.

Place ten items beneath the actions

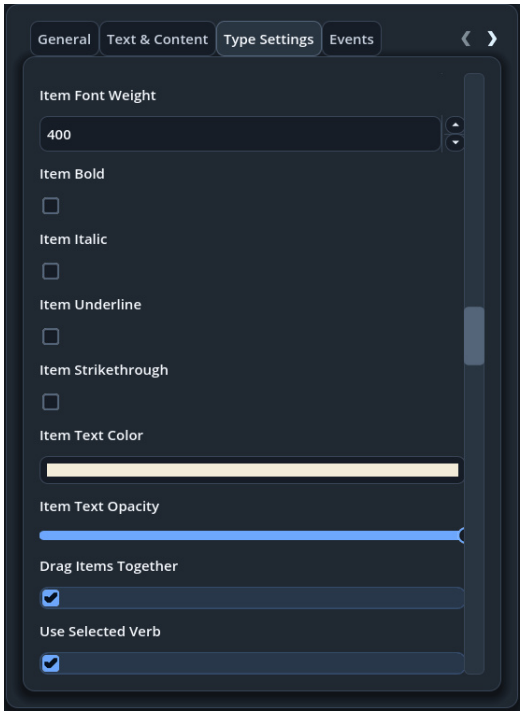

Figure 43. The integrated inventory is still a real Inventory Grid.

Property	Value
Element	gem_items at 485, 347; size 722 × 160.
Grid	5 columns; 2 rows; 10 items per page.
Slots	133 × 54; gaps 8 horizontal / 5 vertical; icon size 44.
Content	Icons on; text and quantity off.
Behavior	Use Selected Verb off for direct target application; drag sorting off.
Feedback	Hover #96A38E; selected #B29462.
Close	Run cancel_target_menu to hide the popup and clear its stored target.

If your own game carries more than ten objects, keep navigation visible and test the next page. If you want a deliberate ten-item limit, enforce that in pickup logic rather than merely making the eleventh object invisible.

Build a useful communicator

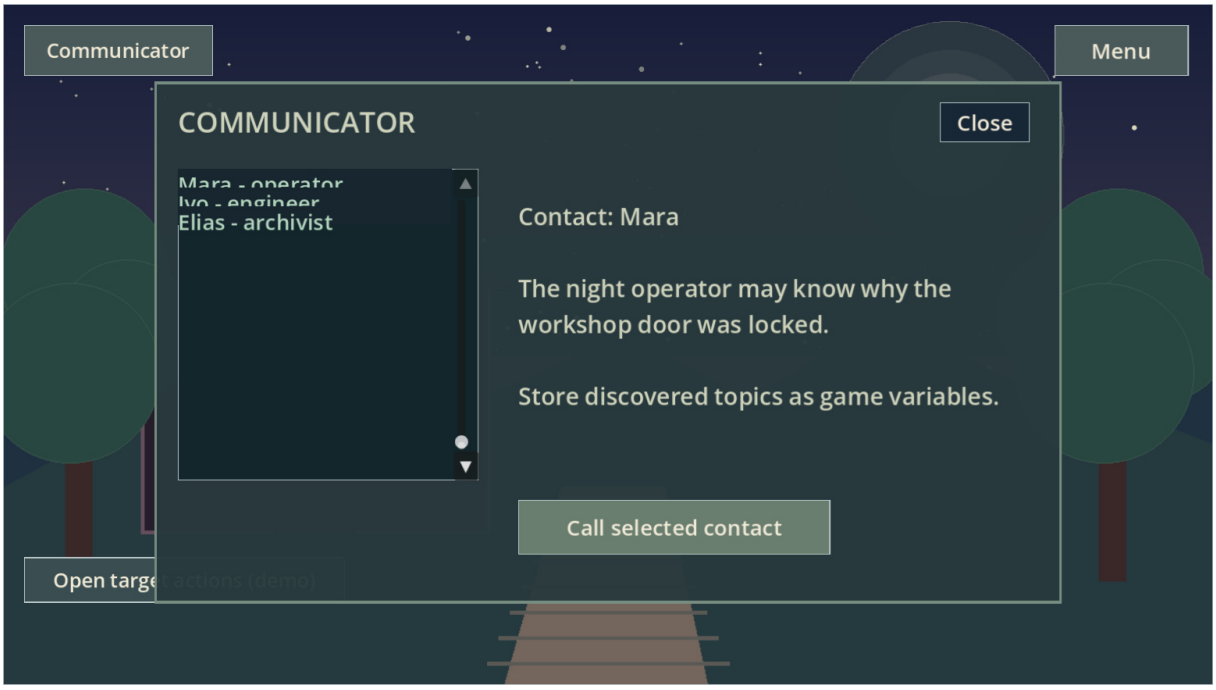

Figure 44. A contact list and readable detail area make a separate investigation tool.

Part	Starting layout
Panel	160, 82; 960 × 552; background #28393DF5.
Contacts List Box	185, 174; 318 × 330; background #12262B; text #B4D7BF.
Detail Label	545, 190; 520 × 260; text #D0D4BD.
Call Button	545, 524; 330 × 58.
Close Button	991, 104; 95 × 42.

Store the selected contact ID in a Selection Variable. Make Call branch on that ID and start the correct dialogue or script. Store newly discovered contacts and facts in saved variables, then rebuild the list from that state.

> The screenshot’s sample Call Button writes a test message. Replace that feedback action with your real contact branch; do not have every contact silently call the same person.

Create a matching saved-position menu

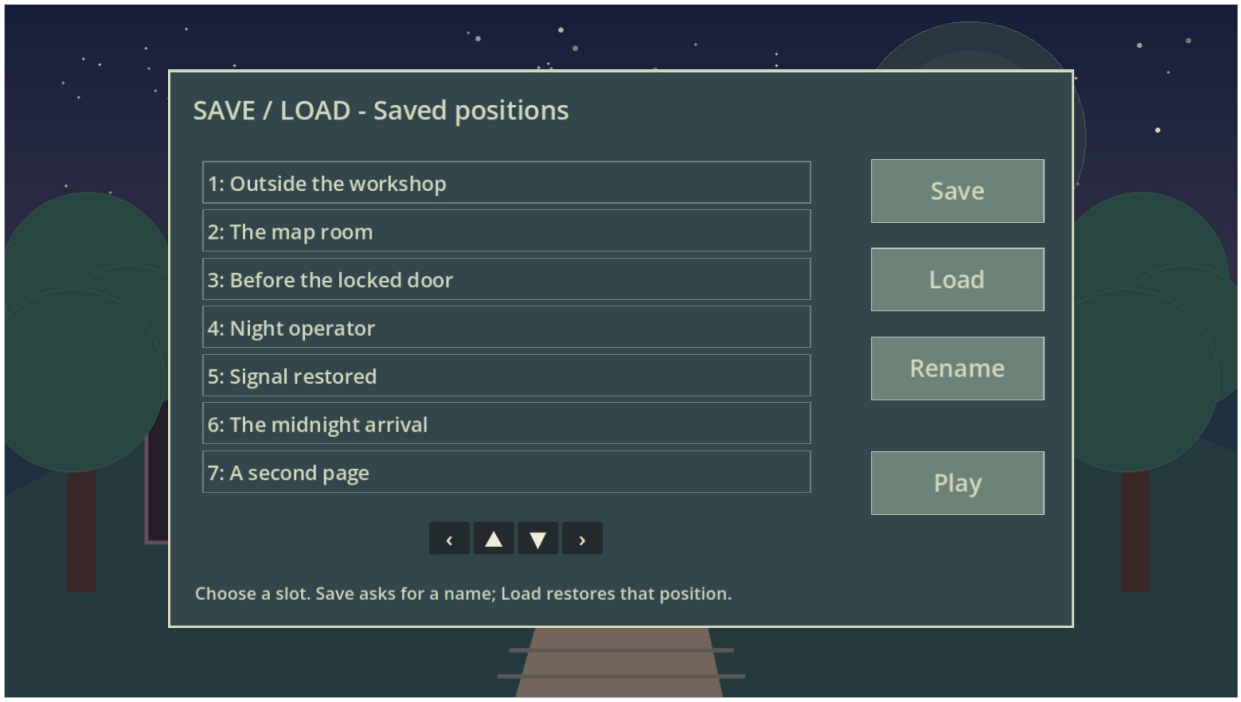

Figure 45. A real Save Game widget, styled to match this interface.

Part	Starting settings
Popup GUI	gem_save; hidden; modal; Cut open and close transitions.
Outer panel	X 170, Y 68, Width 940, Height 580; square corners.
Save widget	gem_saves at 194, 151; size 675 × 432.
List	1 column, 7 rows, 7 items per page; slots 631 × 44; 6-pixel vertical gap; padding 12.
Content	Manual slots 1-12; screenshots off; click selects; double-click none.
Text	Saved: {slotNumber}: {name}. Empty: {slotNumber}: {empty}.
Palette	Panel #32464B; text #D4D9BE; hover #6E8378; selected #C1AF7E.

This text-list adaptation uses PCA’s manual save slots and prompts. Add a separate protected autosave route if your game contains dangerous sequences. The interface styling does not create checkpoints or a combat retry policy by itself.

Animate the target panel and communicator

Part	Motion / feedback plan
Target-action panel	Cut open and close so choices are available immediately.
Action icon hover	A clear brighter pose or outline.
Action press	A held pose; restore the original image after release.
Inventory item	Stable selection; use a quiet highlight.
Communicator	Fade open 0.16 seconds; close 0.12 seconds.
Contact selection	Update text promptly; avoid scrolling or typing the whole directory again.

Keep the atmosphere in the art and color rather than obscuring essential controls. A faint cool finish can suit a device frame; noise, distortion and constant flicker can make small contact names unnecessarily hard to read.

The panel’s title should identify the chosen target. If you make it dynamic, use the stored target or a saved target caption, not the new hover label after the pointer enters the menu.

Test both cancelling the popup and taking a different action. A cancelled popup should not leave an invisible old target waiting for the next inventory click.

Test the four-action investigation loop

1. Open the target panel on a real object. Use Eye, Hand, Mouth and Foot where appropriate.
2. Choose an item after choosing the target. Confirm that it is applied to the stored object and the popup closes.
3. Cancel the popup, move to another target and reopen it. No action should use the previous target.
4. Open the communicator, choose two different contacts and verify the two different results.
5. Learn a contact or fact, save, change it and load. Confirm the restored communicator state.
6. Test a long target name, translated captions, a smaller display and touch/controller navigation if supported.

Keep a simple, successful door interaction as a regression test: inspect it, use it, choose a matching item and cancel. Run that test whenever you change the popup’s input or animation.

> Combat, companion-assisted actions and unrestricted name-dragging are game systems beyond this interface recipe. Add them as explicit mechanics after the menu and inventory behave reliably.

12. Small scripts for the missing details

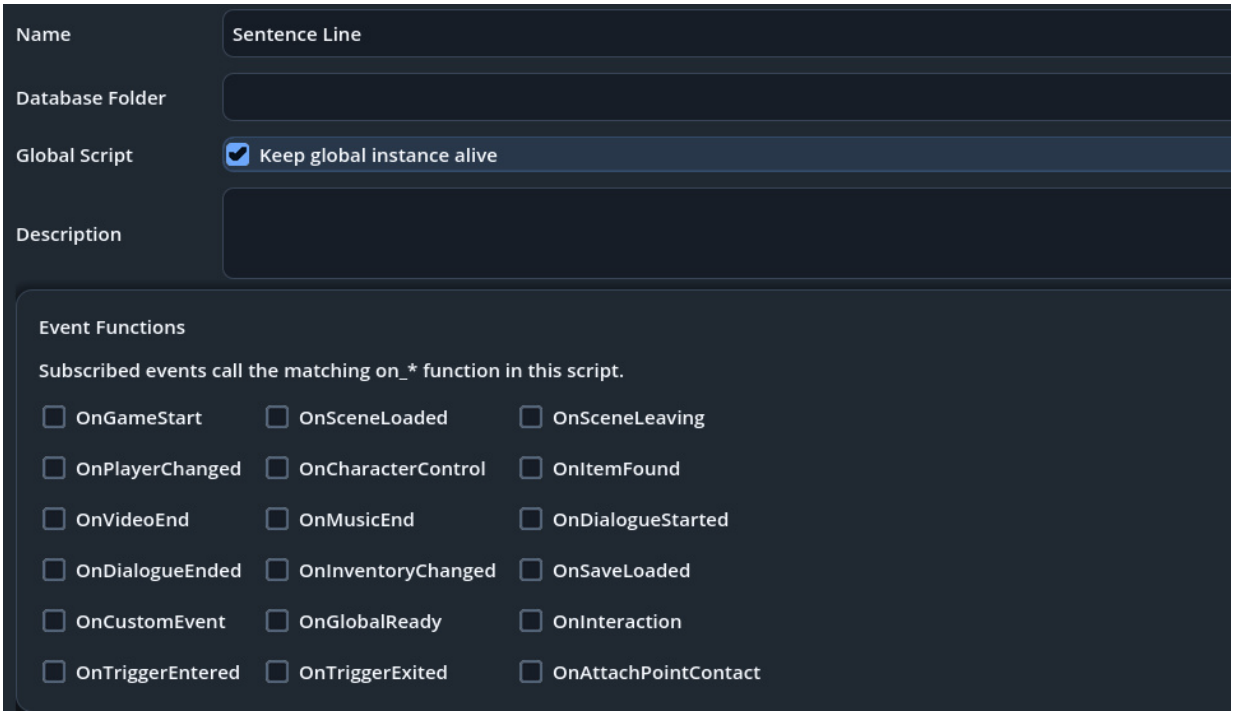

Figure 46. Global Script keeps a small controller available during play.

1. Open the GDScript editor and add a Script record with the ID given for the example.
2. Replace the record’s source with the complete example. Keep the first extends line.
3. Enable the script. Turn on Global Script only for the examples marked Global.
4. Keep the IDs used in the example, or change the script and the GUI together.
5. Run Validate, then test in Play. A compiled script can still refer to an element that you did not create.
6. For ordinary event scripts, choose Run Script/Run Visual Script in the event action and select the record. That shared action can run a GDScript record as well as a visual graph.

> These scripts target the reference build, 0.86.3.4. They use its runtime GUI classes where ordinary shared actions do not cover the detail. Keep a project copy before adapting them to a later engine build.

Keep the sentence caption complete

Record ID: sentence_line. Global Script: on. This writes the pending pair/item prefix into the four named command Labels. Keep those Labels’ authored text simple; the script owns their live caption.

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

func on_global_ready(_context: Dictionary = {}) -> void:
    process_priority = 100
    set_process(true)

func _process(_delta: float) -> void:
    if not is_instance_valid(Runtime):
        return
    var manager: Node = Runtime.get("authored_gui_manager")
    if not is_instance_valid(manager):
        return
    var prefix: String = str(Runtime.call("get_current_verb_display_name"))
    var pair: Variant = Runtime.get("preposition_interaction")
    if pair != null and pair.call("active"):
        prefix = str(pair.call("prefix"))
    elif not str(Runtime.get("selected_inventory")).is_empty():
        var item_id: String = str(Runtime.get("selected_inventory"))
        var item: Dictionary = Runtime.get("project").get_inventory(item_id)
        prefix = "Use " + str(item.get("name", item_id)) + " with"
    var caption: String = (prefix + " " + str(Runtime.get("current_hover_label"))).strip_edges()
    var nodes: Dictionary = manager.get("element_nodes")
    for label_id: String in ["zak_command", "dot_command", "mi_command", "curse_object_line"]:
        var host: Control = nodes.get(label_id) as Control
        var label: Label = host.get_node_or_null("Text") if is_instance_valid(host) else null
        if is_instance_valid(label) and label.is_visible_in_tree():
            label.text = caption
```

The Label element’s live control contains a Text child. This script updates that child; it does not rebuild the control or replace every word with a new GUI element.

Adapt the top inventory’s mouse buttons

Record ID: bass_buttons. Global Script: on. The widget remains verb-sensitive. The adapter selects Look for an item left-click and Walk for an item right-click before the widget performs its built-in item action.

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

func on_global_ready(_context: Dictionary = {}) -> void:
    set_process(true)

func _process(_delta: float) -> void:
    if not is_instance_valid(Runtime):
        return
    var manager: Node = Runtime.get("authored_gui_manager")
    if not is_instance_valid(manager):
        return
    var nodes: Dictionary = manager.get("element_nodes")
    var widget: Node = nodes.get("bass_items")
    if not is_instance_valid(widget):
        return
    if not widget.is_connected("inventory_event", _item_event):
        widget.connect("inventory_event", _item_event)

func _item_event(_element: String, event: String, _item: String) -> void:
    if event not in ["click", "rightClick"]:
        return
    Runtime.call("clear_selected_inventory_item")
    Runtime.call("set_verb", "look" if event == "click" else "walk")
```

The controller reconnects when it finds the widget again, without adding duplicate signal connections. It changes only bass_items. It does not change every inventory in the project.

Run Use from an object’s right-click event

Record ID: right_use. Global Script: off. Attach this to the ordinary scene object’s On Right Click pointer event in the Direct-mode recipe.

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

func run(event_context: Dictionary = {}) -> Variant:
    var target: String = str(event_context.get("entityId", ""))
    if target.is_empty():
        return false
    Runtime.call("set_verb", "use")
    await Runtime.call("execute_selected_verb", target)
    return true
```

1. Enable the object’s hotspot and define its Use verb result.
2. Add a Run Script action to its On Right Click event, selecting right_use.
3. Test without an item, then with a selected matching item.
4. Keep the script out of the Use result itself, or it would call the same interaction recursively.

The script expects entityId from a world-object event. It is intentionally not a general “use whatever is under the current pointer” function.

Cancel the target as well as the picture

Record ID: cancel_target_menu. Global Script: off. Use this on the Cancel/Close Button of curse_coin and gem_context instead of hiding only the visible GUI.

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

func run(_context: Dictionary = {}) -> Variant:
    Runtime.set("verb_direct_target", "")
    Runtime.call("clear_selected_inventory_item")
    var popup: String = str(GetProjectProperty(
        "meta.verbDirectGuiId", ""))
    if not popup.is_empty():
        await HideGUI({"guiId":popup})
    return true
```

The visible popup and the stored target are separate state. This script clears the pending target and the selected inventory item, then hides the currently configured Verb Direct Popup GUI.

Use this for cancelling a target-action menu. Do not put it on a chest Close Button when you intentionally want to retain a selected item for world use.

Store and refresh character memories - part 1

Record ID: memory_tools. Global Script: on. Create one record containing Part 1 followed by Part 2. The scene-loaded and player-changed hooks refresh the displayed arrays after the relevant game event.

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

func on_scene_loaded(_context: Dictionary = {}) -> void:
    await refresh()

func on_player_changed(_context: Dictionary = {}) -> void:
    await refresh()
    await SetVariable({"name":"memory_topic", "value":""})

func refresh() -> void:
    var values: Dictionary = Runtime.get("variables")
    var actor: String = ActivePlayer()
    for kind: String in ["stm", "ltm"]:
        var source: Array = values.get(kind + "_" + actor, [])
        await SetVariable({"name":"memory_" + kind + "_active",
            "value":source.duplicate(true)})
```

Declare the character-specific arrays and the two active display arrays in the Variable Designer. Set the memory List Boxes to array sources, reading memory_stm_active and memory_ltm_active.

This is the first part of one script, not a separate record. Do not add a second extends line before Part 2.

Store and refresh character memories - part 2

```
func remember(topic: String, caption: String, long_term: bool = false) -> void:
    var kind: String = "ltm" if long_term else "stm"
    var key: String = kind + "_" + ActivePlayer()
    var values: Dictionary = Runtime.get("variables")
    var entries: Array = values.get(key, []).duplicate(true)
    for entry: Dictionary in entries:
        if str(entry.get("id", "")) == topic:
            return
    if not long_term and entries.size() >= 10:
        entries.pop_front()
    entries.append({"id":topic, "text":caption})
    await SetVariable({"name":key, "value":entries})
    await refresh()

func forget_short_term() -> void:
    await SetVariable({"name":"stm_" + ActivePlayer(), "value":[]})
    await refresh()

func run(_context: Dictionary = {}) -> Variant:
    await remember("window", "Broken window")
    return true
```

The default run method remembers the broken window as a convenient first test. Calling remember with long_term true writes to the active character’s long-term store instead.

Duplicate topic IDs are ignored. The short-term store’s ten-entry policy drops the oldest entry; change that policy if your game needs a visible “which memory should be replaced?” choice.

Call a memory function from a normal event

For a Remember Window Button or an object’s Remember result, selecting memory_tools as the ordinary script runs its default example. For another clue, make a small wrapper record like this:

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

func run(_context: Dictionary = {}) -> Variant:
    await CallScript("memory_tools", "remember",
        ["signal", "The missing signal", true])
    return true
```

If you already use Functions as Nodes, expose the same remember function with topic, caption and long_term parameters through your function definitions. That changes how the action is authored; it does not change the saved memory format.

The following ordinary wrapper changes the player and refreshes the displayed lists. Record ID: memory_switch; Global Script: off.

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

func run(_context: Dictionary = {}) -> Variant:
    await Runtime.call("switch_player_character", "assistant")
    await CallScript("memory_tools", "refresh")
    return true
```

With the global player-changed hook enabled, the explicit refresh is harmless but redundant. Keeping it in a first test makes the order of operations easy to follow.

Make a portrait switch or receive an item

Record ID: transfer_to_mara. Global Script: off. Put this on Mara’s portrait instead of the plain Switch Character behavior.

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

func run(_context: Dictionary = {}) -> Variant:
    var item: String = str(Runtime.get("selected_inventory"))
    if item.is_empty():
        await Runtime.call("switch_player_character", "assistant")
        return true
    await TransferInventory({"fromCharacterId":ActivePlayer(),
        "toCharacterId":"assistant", "itemId":item, "count":1})
    Runtime.call("clear_selected_inventory_item")
    return true
```

Copy the record for another recipient and change assistant to that character’s ID. Keep ActivePlayer() as the source; hard-coding the source would transfer from the wrong character after a switch.

Add story or ownership checks before Transfer Inventory when needed. A transfer-aware portrait should explain why an unavailable recipient or a forbidden item cannot be used, rather than behaving like a dead Button.

Use the item’s actual selected ID. Its caption, icon and translated name are presentation; they are not reliable transfer keys.

Pulse an unlocked portrait three times

Record ID: flash_portrait. Global Script: off. Call it when Mara becomes available. The Button keeps its size and position throughout the pulse.

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

func run(_context: Dictionary = {}) -> Variant:
    var manager: Node = Runtime.get("authored_gui_manager")
    var nodes: Dictionary = manager.get("element_nodes")
    var portrait: Control = nodes.get("dot_actor_assistant") as Control
    if not is_instance_valid(portrait):
        return false
    var original: Color = portrait.modulate
    var tween: Tween = create_tween().set_loops(3)
    tween.tween_property(portrait, "modulate", Color(1.0, 0.68, 0.5), 0.12)
    tween.tween_property(portrait, "modulate", original, 0.12)
    await tween.finished
    if is_instance_valid(portrait):
        portrait.modulate = original
    return true
```

Each cycle is 0.12 seconds to the warm tint and 0.12 seconds back. Three cycles take about 0.72 seconds. The original color is restored at the end.

For reduced motion, use a one-time steady highlight or a short message instead. Do not use repeated flashing to show permanent selection.

Play an Image sequence once

Record ID: one_shot. Global Script: off. Create animation_lab with the Image anim_arrow and four distinct frame IDs at 6 FPS, as in Chapter 3. Show that GUI before running this example.

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

func run(_context: Dictionary = {}) -> Variant:
    var values: Dictionary = Runtime.get("variables")
    if bool(values.get("ui_busy", false)):
        return false
    var manager: Node = Runtime.get("authored_gui_manager")
    var nodes: Dictionary = manager.get("element_nodes")
    var host: Control = nodes.get("anim_arrow") as Control
    if not is_instance_valid(host):
        return false
    var image: PCAGUISpritePlayer = host.get_node_or_null("GUIContentImage")
    if not is_instance_valid(image):
        return false
    await SetVariable({"name":"ui_busy", "value":true})
    image.loop_animation = false
    image.restart()
    await get_tree().create_timer(4.0 / 6.0).timeout
    if is_instance_valid(image):
        image.playing = false
    await SetVariable({"name":"ui_busy", "value":false})
    return true
```

The script restarts the native Image player, disables looping and holds the last frame. Adapt the element ID and the duration together when using a chest-lid or page-turn drawing.

The example uses a four-frame, 6-FPS schedule. If you change the frame count or rate, change the wait as well. Keep live inventory/save widgets separate from the moving drawing.

Select a coin action on mouse release

Record ID: curse_release. Global Script: on. Keep Verb Direct configured for curse_coin and retain the three listed choice IDs.

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

var tracking: bool = false
const CHOICES: Dictionary = {"curse_coin_look":"look",
    "curse_coin_use":"use", "curse_coin_talk":"talk"}

func on_global_ready(_context: Dictionary = {}) -> void:
    set_process(true)

func _process(_delta: float) -> void:
    if not is_instance_valid(Runtime):
        return
    var manager: Node = Runtime.get("authored_gui_manager")
    if not is_instance_valid(manager):
        return
    var visible_now: bool = bool(manager.call("is_gui_visible", "curse_coin"))
    var holding: bool = Input.is_mouse_button_pressed(MOUSE_BUTTON_LEFT)
    if holding and visible_now and not str(Runtime.get("verb_direct_target")).is_empty():
        tracking = true
    if not tracking or holding:
        return
    tracking = false
    if not visible_now:
        return
    var nodes: Dictionary = manager.get("element_nodes")
    var pointer: Vector2 = get_viewport().get_mouse_position()
    for element_id: String in CHOICES:
        var control: Control = nodes.get(element_id) as Control
        if not is_instance_valid(control):
            continue
        var local: Vector2 = control.get_global_transform_with_canvas().affine_inverse() * pointer
        if Rect2(Vector2.ZERO, control.size).has_point(local):
            Runtime.call("choose_verb_from_gui", str(CHOICES[element_id]), false)
            return
    manager.call("hide_gui", "curse_coin")
    Runtime.set("verb_direct_target", "")
```

This adds a release gesture to a working target-first popup. It does not add a delayed hold threshold, an animated software cursor or automatic near-pointer positioning.

Send a logbook command to the correct page

Record ID: book_save. Global Script: off. Put it on each spread’s Save Button. Both page widgets must write their selected slot number into selected_save.

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

func run(_context: Dictionary = {}) -> Variant:
    var values: Dictionary = Runtime.get("variables")
    var slot: int = int(values.get("selected_save", -1))
    if slot < 1 or slot > 12:
        return false
    var page: int = 0 if slot <= 6 else 1
    var column: int = 0 if (slot - 1) % 6 < 3 else 1
    var gui_id: String = "curse_save" if page == 0 else "curse_save2"
    await SaveGameWidgetCommand({"guiId":gui_id,
        "elementId":gui_id+"_saves_"+str(column),
        "command":"save-slot", "slot":slot})
    return true
```

Create book_load from the same source and change the command from save-slot to load-slot. Put that record on the Load Buttons. The explicit slot routes the command to the correct left/right widget on the correct spread.

On each page-change Button, first set selected_save to -1, then hide the old spread and show the destination. This guard makes a new selection necessary before the shared dispatcher can save or load.

> For more than two spreads, extend the dispatcher and page setup together. Do not claim a slot range that the corresponding widget does not own.

Keep one logbook selection visible

Record ID: book_selection. Global Script: on. The two page widgets have independent internal selections. This controller clears the other highlights when the shared selected_save value changes.

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

var previous: int = -999

func _process(_delta: float) -> void:
    if not is_instance_valid(Runtime):
        return
    var manager: Node = Runtime.get("authored_gui_manager")
    if not is_instance_valid(manager):
        return
    var values: Dictionary = Runtime.get("variables")
    var selected: int = int(values.get("selected_save", -1))
    if selected == previous:
        return
    previous = selected
    var nodes: Dictionary = manager.get("element_nodes")
    for page_id: String in ["curse_save", "curse_save2"]:
        for column: int in range(2):
            var id: String = page_id + "_saves_" + str(column)
            var widget: PCASaveGameWidget = nodes.get(id) as PCASaveGameWidget
            if not is_instance_valid(widget):
                continue
            if widget.selected_slot != selected:
                widget.selected_slot = -1
                widget.element["saveSelectedSlot"] = -1
                widget.refresh()
```

This refreshes the existing widget’s slots. It does not repeatedly change its source range or rebuild the complete authored inventory/save element.

Require a selected slot before saving

Record ID: strict_save. Global Script: off. This example is wired for mi_save / mi_saves. Replace those two IDs for another single-widget menu.

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

func run(_context: Dictionary = {}) -> Variant:
    var gui_id: String = "mi_save"
    var widget_id: String = "mi_saves"
    var slot: int = int(GetGUIElementProperty(
        gui_id, widget_id, "selected_slot", -1))
    if slot < 1:
        return false
    await SaveGameWidgetCommand({"guiId":gui_id,
        "elementId":widget_id, "command":"save-slot", "slot":slot})
    return true
```

The ordinary selected-slot command can fall back to the first available slot if no explicit row was selected. This guard returns without saving instead. Create a load variant by changing save-slot to load-slot.

1. Put the wrapper on the Save Button instead of a direct save-selected action.
2. Run the menu without choosing a row. Nothing should be saved.
3. Select a row, run Save, name the position and confirm the result.
4. Keep Prompt for Name and Confirm Overwrite enabled on the widget.

For an even clearer interface, disable Save/Load until a valid row is selected and provide a short instruction. A silent guard is a safety net, not the whole user experience.

13. Test the interface as a player

Test	What should be true
First interaction	A new player can discover the action order without guessing.
Every command	Its caption, cursor and actual result agree.
Hover / hold / release	The clicked image returns to normal and does not become a permanent selection.
Inventory overflow	An extra item remains reachable through navigation.
Character switch	The displayed inventory, active portrait and memory owner agree.
Target popup	The chosen action applies to the stored target, including after pointer movement.
Cancellation	No old item, target or topic causes the next unrelated click to act incorrectly.
Save / load	The restored scene, variables, ownership and character state match the saved position.

Run these tests on a short puzzle, not only on a blank menu. A key, a locked mechanism and one speaking character are enough to expose most input-order mistakes.

> GUI Preview is for appearance and control feedback. Use Play to test scripts, action lists, item use, character switching and save files.

Troubleshoot input and inventory

Symptom	Check first
A verb Button does nothing	Its Click behavior, selected verb ID and the target object’s hotspot/verb result.
The popup acts on the wrong thing	Stored target versus a new raycast from the GUI Button’s location.
Use on an item gives a fallback	Use Selected Verb and the item’s preposition/Use response configuration.
Gemini item selection waits for another target	An unwanted preposition opt-in or the wrong inventory behavior.
Beneath left-click combines items	An old selection was not cleared before the Look interaction.
A transferred item stays on both characters	Check the source/recipient IDs, count and whether the display refresh follows ownership.
A new character will not switch	Playable/switch-enabled settings, scene placement and current input locks.
The last inventory row is missing	Grid minimum dimensions, padding, header and navigation space.

When a control disappears behind a background Panel, check Z Index before changing its color. A correct font cannot be read through an opaque element drawn over it.

Troubleshoot animation and artwork

Symptom	Check first
The animation only has two frames	Distinct frame asset IDs. This build deduplicates repeated IDs.
A sequence jumps in size or position	Frame canvas dimensions, registration and Content Fit.
The opening picture loops forever	Use a non-looping player for one-shot art; the ordinary Image sequence loops.
The chest contents move with the lid	Separate live inventory slots from the animated drawing.
A page flip loads the wrong slot	Keep visual animation separate from page range and selected_save.
A pressed image remains selected	Check release handling separately from persistent selected/checked state.
An “animated cursor” is static	The ordinary hardware cursor takes a static image; GUI frames do not animate it.
A shader hides the words	Remove the effect, confirm the palette, then reapply it only to a suitable decorative element.

On the reference build, avoid repeatedly changing live inventory/save widget properties as a way to animate them. Use authored alternatives, the supported frame fields and dedicated decorative Images instead.

> Prefer one short motion with a clear purpose. A menu full of independent loops makes every part look equally urgent.

Troubleshoot saves and remembered state

Symptom	Check first
Save chooses a row unexpectedly	Selected-slot fallback. Add the strict selection guard if required.
A text-list preset changed the layout	Apply the preset before customization; it can overwrite dimensions and behavior.
The book numbers fill across rows	Use the two one-column widgets for column-major left/right order.
Both book pages show selection	Install book_selection, and clear the shared selection on a spread change.
A prompt appears behind another prompt	Do not open a second modal dialog while naming or confirming a save.
Memories disappear after restoring	Check the saved character-specific arrays and the refresh hook.
A contact list survives only until restart	Store discovered contacts/facts in saved variables, not only temporary visible List Box items.
A thumbnail looks unlike the GUI backdrop	The save system captures the scene view; a decorative GUI illustration is not the world renderer.

Give save tests their own test project or disposable slots. Keep an older known-good position while changing the menu or its data model.

Make the same interface usable elsewhere

1. Test long and translated names in every narrow slot, caption and portrait hint.
2. Give touch users an explicit opener for hover-edge inventories.
3. Provide a controller or on-screen route to actions that were originally selected by right-click cycling.
4. Keep a click-based alternative to hold-and-release menus.
5. Use a stable selected border as well as a color change. Do not rely only on a faint hue difference.
6. Offer a reduced-motion presentation that cuts or briefly fades instead of replaying decorative art.
7. Check the real exported game, including focus, tab order and input at a smaller display size.

Keep the original interface’s useful organizing idea while adapting its input to the device. A top inventory can still look like a top inventory when it opens from a visible Equipment Button.

> The finished version should feel like one interface. Consistent names, input order and cancellation rules matter more than adding another decorative effect.

14. Reference notes and design sources

Historical descriptions were checked against the original manuals where available, publisher material and contemporary interface descriptions. The instructional layouts, colors, timings and PCA screenshots are workshop adaptations. Game titles identify the design references; the screenshots are not captures of those commercial games.

[1] Zak McKracken and the Alien Mindbenders - Lucasfilm Games manual

Zak section: screen layout, word inventory, command confirmation and special verbs.

[2] Day of the Tentacle - LucasArts manual

Playing the Game / Using the Kids: six inventory entries, portraits and transfer shortcut; Save/Load instructions.

[3] Sam & Max Hit the Road - LucasArts manual

Action cursors, inventory and hotspot feedback. The archived manual includes Macintosh-specific menu behavior; the workshop save shell is an authored cross-platform adaptation.

[4] Beneath a Steel Sky - Revolution / Virgin manual

Playing the Game and Control Panel: inventory, left/right mouse behavior, save and restore.

Original releases, ports and remasters can use different shortcuts and save-menu shells. This guide is a PCA construction manual, not a key-reference sheet for running every historical release.

Reference notes continued

[5] Monkey Island 2: LeChuck’s Revenge - original manual

Playing the Game: sentence line, verbs and eight-icon inventory; save/load instructions.

[6] The Curse of Monkey Island - LucasArts manual

Printed pages 16-19: action coin and inventory; pages 22-25: logbook and six save boxes.

[7] Resonance - Wadjet Eye Games

Publisher overview: four controllable characters and the long-/short-term memory system.

Resonance - contemporary interface description

Console, physical inventory and memory behavior. The workshop deliberately uses a click-to-remember adaptation.

[8] Gemini Rue - Wadjet Eye Games

Publisher overview and investigation context.

Gemini Rue - contemporary control description

Four target actions, carried items and communicator. The workshop uses PCA’s native target-first behavior.

Keep a finished baseline

When your chosen interface works, save a clean project copy. Keep the IDs and actions stable while replacing temporary artwork with the final drawings.

Keep	Why
One known-good puzzle	It tests choosing an action, using an item and getting feedback.
One inventory-overflow test	It prevents the next added item from becoming unreachable.
One save on each page/column	It checks range, selection and command routing.
One restored multi-character state	It checks ownership and knowledge together.
A static animation option	It gives you a reliable baseline for input and readability.

After that, polish one layer at a time: readable text first, then spacing and colors, then image states, then motion. Run the same short puzzle again after each meaningful change.

> Reference build: PCA 3D Adventure Studio 0.86.3.4. The guide includes real editor and Play captures. No practice project, copied commercial artwork or companion asset download is required.