



GAMING

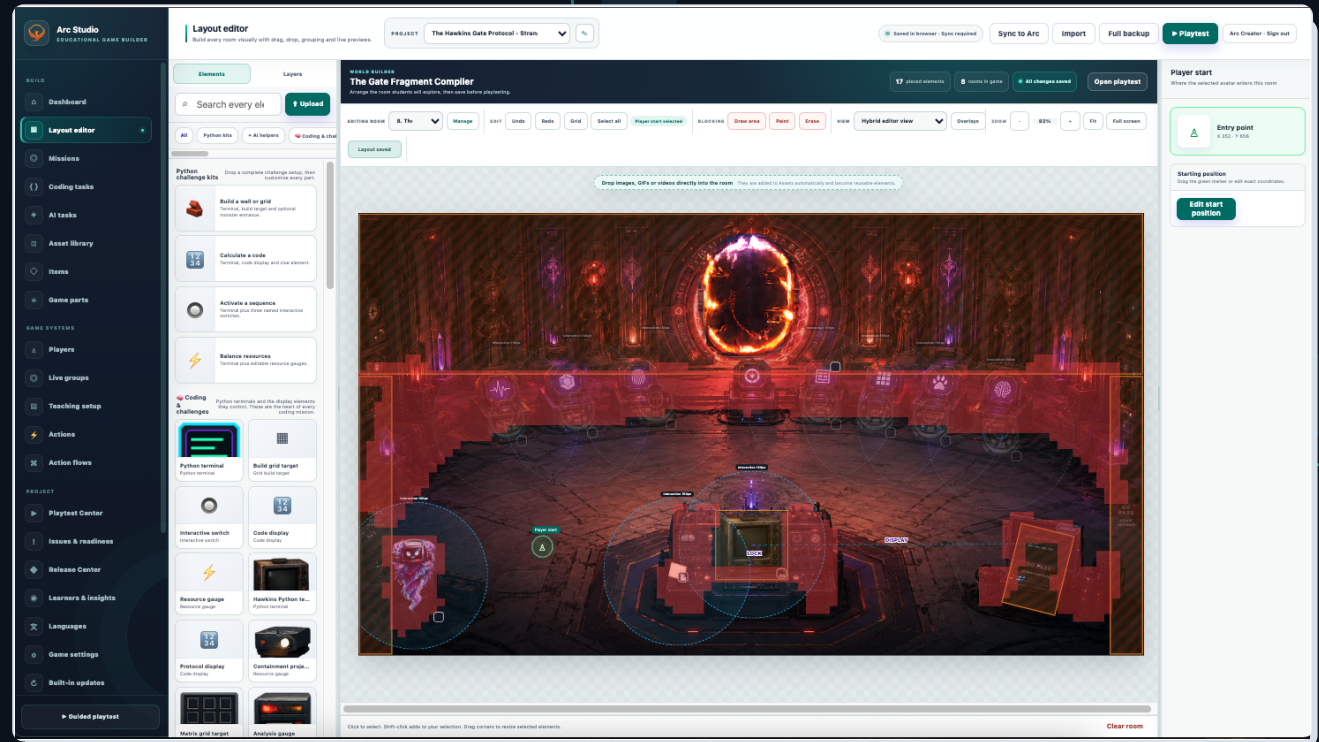
STUDIO VISUAL GUIDE

BUILD / PLAY / TEACH

INSIDE ARC STUDIO

Turn a learning idea
into a playable world.

A guided look at the tools behind
the games—without a Studio login.



ACTUAL STUDIO SCREEN

24

SUPPLIED SCREEN VIEWS

PYTHON + AI

LEARNING TASK WORKSPACES

BUILD TO TEACH

ONE CONNECTED WORKFLOW

A complete Studio, at a glance.

Explore the workspaces that turn rooms, tasks and rules into a learner experience.

WHAT ARC STUDIO DOES

**Build the world.
Shape the learning.
Run the experience.**

ARC Studio is the creator side of ARC Gaming: a visual environment for assembling educational room games, adding Python or AI tasks, and connecting learner actions to visible outcomes.

This guide is a read-only product tour for educators, partners and stakeholders. Screens show captured project states—not a live account or current operational report.

NO LOGIN DETAILS INCLUDED

JUMP TO A SECTION

01	Understand the creator workflow	03–04
02	Build rooms, media and characters	05–10
03	Create learning tasks and game logic	11–19
04	Configure, test, publish and protect	20–27
05	Teach, facilitate and understand play	28–32
06	Administration and developer tools	33–36
07	Demo route, evidence and source index	37–40

Basis: 24 supplied screenshots; current attached project documentation. Full source register: pages 39–40.

One idea. Connected workspaces.

Each part has a clear job; the game becomes playable when those parts are connected.

01

Prepare the building blocks

Asset library supplies media. Items, Game parts and Players define reusable objects and characters.

PAGES 07–10



02

Assemble the rooms

Layout places objects, defines walkable space and connects rooms. Missions gives each room its brief.

PAGES 05–06, 11



03

Add learning and reactions

Coding Tasks and AI Tasks define challenges. Actions and flows decide what changes in the world.

PAGES 12–19



04

Check and release

Settings, Issues, Playtest and Release Center separate configuration, observed play and learner access.

PAGES 20–24



05

Deliver and learn from play

Teaching setup and Live groups launch activities; Insights helps educators review what happened.

PAGES 28–32

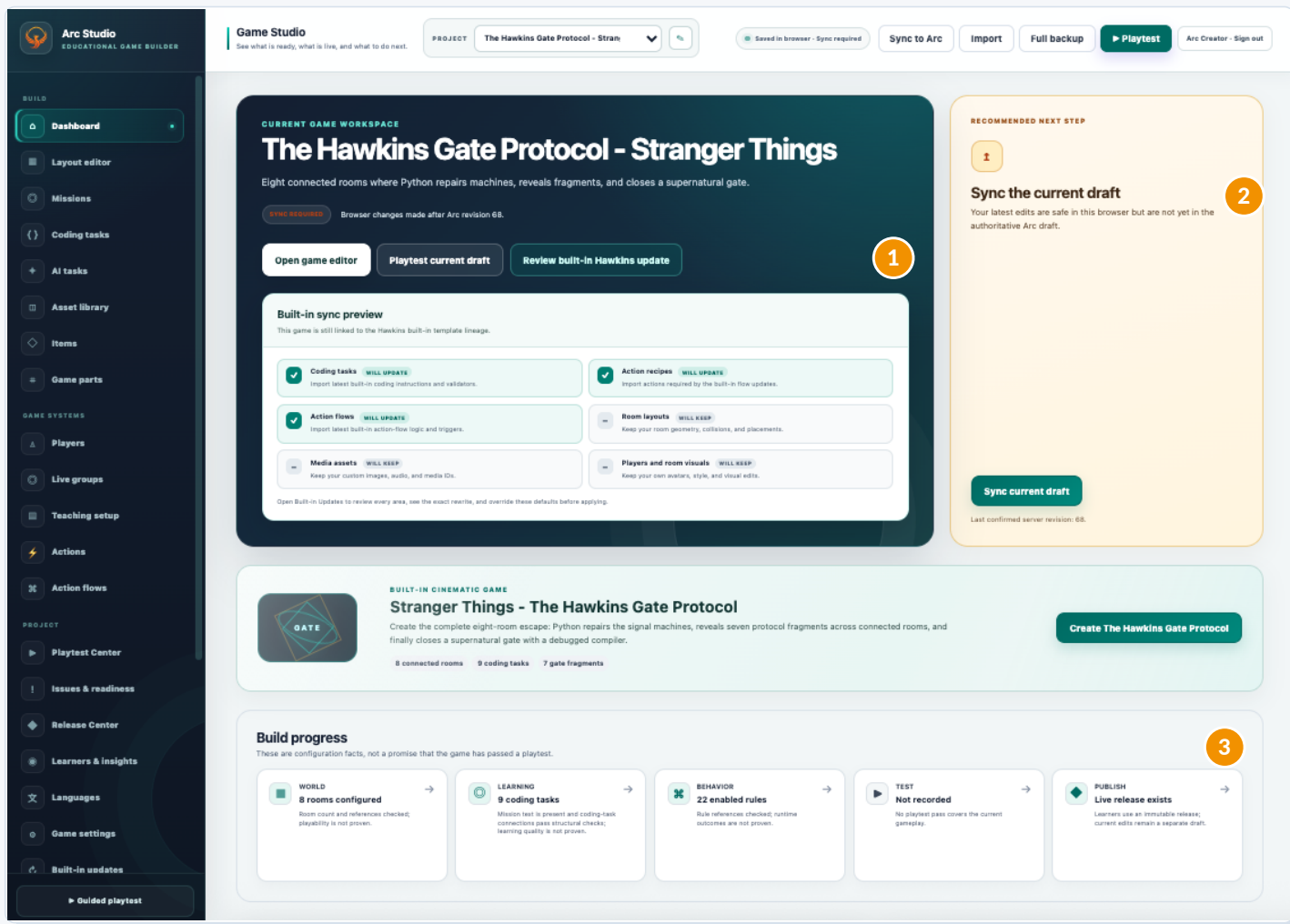
SAVE IS NOT THE SAME AS PUBLISH

Save layout commits room edits to the browser project. **Sync to Arc** updates the server draft. A **release** is a fixed learner-facing version. Backups protect content; they do not make it live.

Sources: [S1] Shared shell, Layout, Coding Tasks, Actions, Flows; [S3] Creator workflow; [S4] Player runtime.

Dashboard: know what to do next.

The current game, its editing state and the next useful action share one starting point.



1 Current workspace

Open the game editor, start a playtest, or review a built-in template update for the game currently selected.

2 Real next step

The captured project has browser changes that need syncing. The recommendation is to sync—not to assume the existing live release contains those edits.

3 Build progress

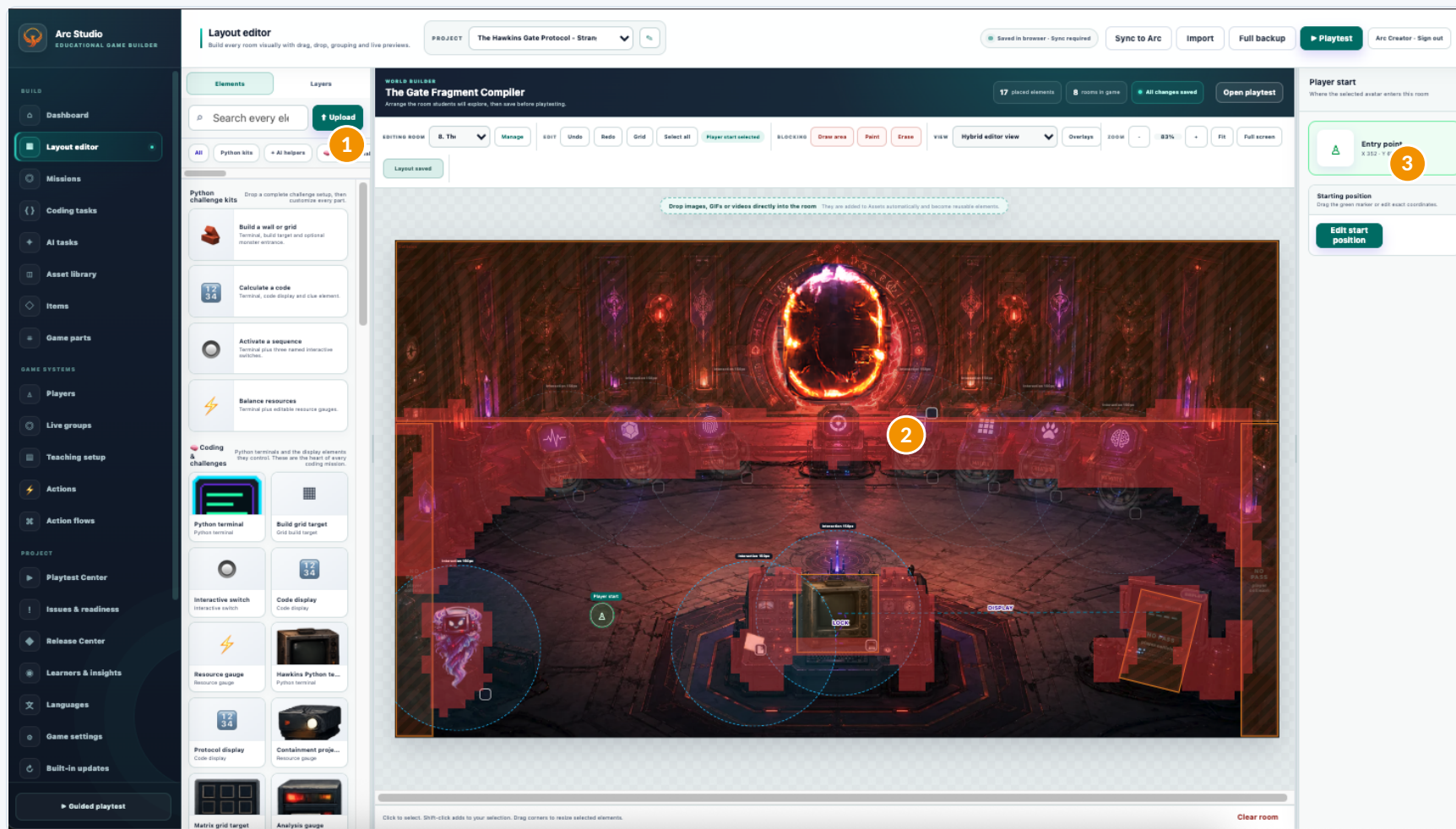
World, learning, behavior, testing and publication are separate checks. A configured task or room count is not proof of a successful learner journey.

IN PRACTICE

Use the project selector before editing. One browser workspace holds the currently open game.

Layout editor: build the playable room.

Place the scene, inspect its boundaries and compose the spaces the learner can explore.



1 Elements & kits

Browse reusable objects and connected Python kits.

2 Room canvas

Drag, resize, group and align the placed objects.

3 Selection inspector

Review the selected object or player entry point.

SCREEN 02 · layout_editor.png | [S1] Layout page; [S2] Rooms and item placements

Make a beautiful room work as a game.

A background provides the setting. Placements, blocking and state changes provide the interaction.



ENLARGED DETAIL / EDITOR OVERLAYS

Red blocking overlays and interaction guides are authoring aids. They help a creator keep furniture solid and make each usable device reachable.

EDITING ROOM ≠ LEARNER START ROOM

Draw the route

Draw no-pass areas over painted furniture, or paint and erase irregular blocking. Preserve a walkable route to the task, reward and exit.

Give objects meaningful states

A placed device can have states such as Off and On, or Locked and Unlocked. States can change its visual, visibility, collision, lock and content.

Save, then test

Layout has an explicit Save layout step. Saving updates the browser project and rebuilds blocking; Sync to Arc is the separate server-draft step.

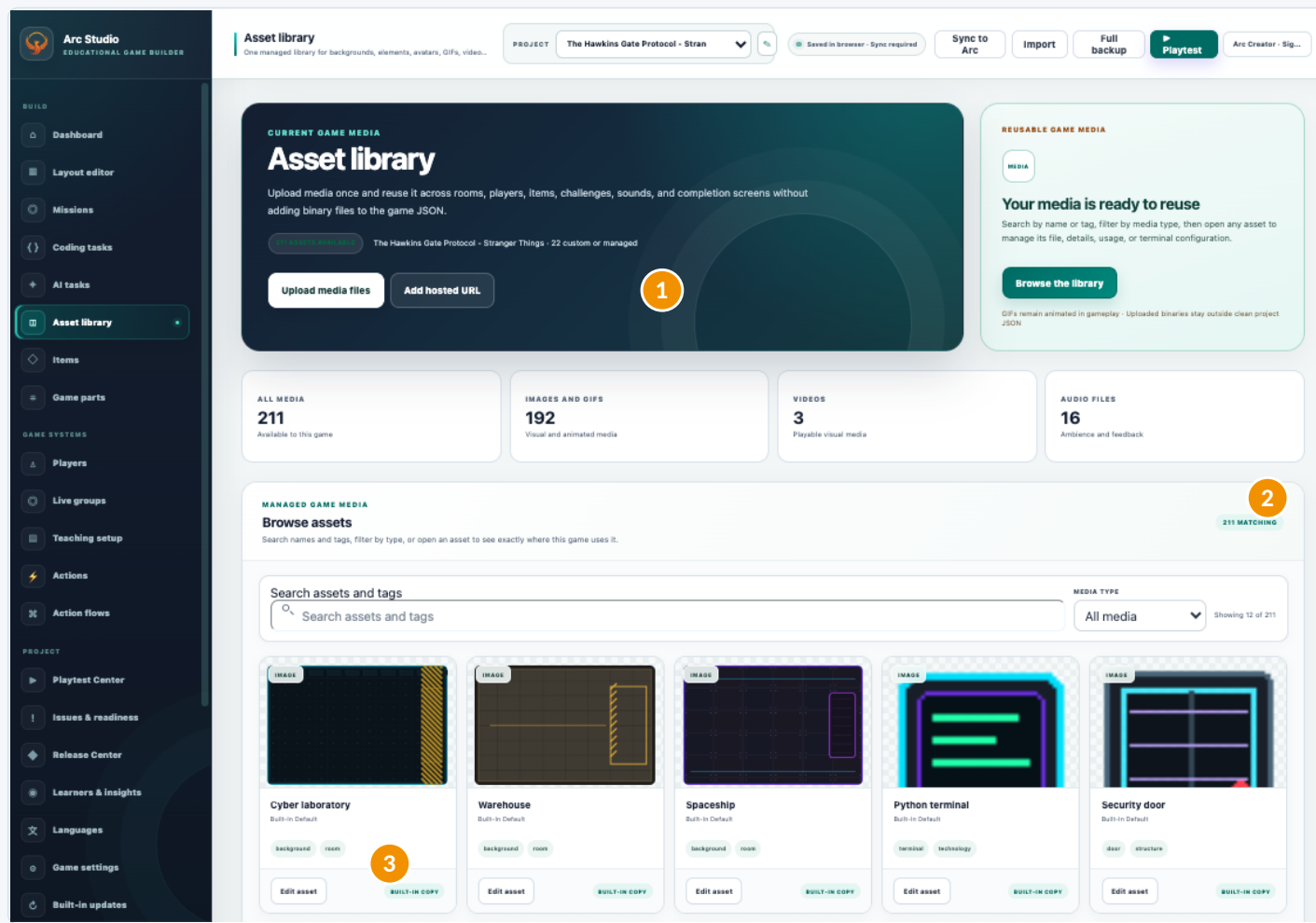
PLACEMENT-SPECIFIC

States belong to one placed copy, not automatically to every object using the same reusable template.

Sources: SCREEN 02 (enlarged details); [S1] Layout: explicit save model, detailed item drawer; [S4] Movement and collision.

Asset library: one home for the media.

Reuse backgrounds, object art, animation, video and sound throughout the game.



1 Bring in media

Upload media files or add a hosted URL. Search by name or tag and filter by media type to find the asset you need.

2 Reuse the same source

Rooms, players, items, tasks and opening or completion screens can reference the library instead of duplicating configuration for each use.

3 Manage and inspect

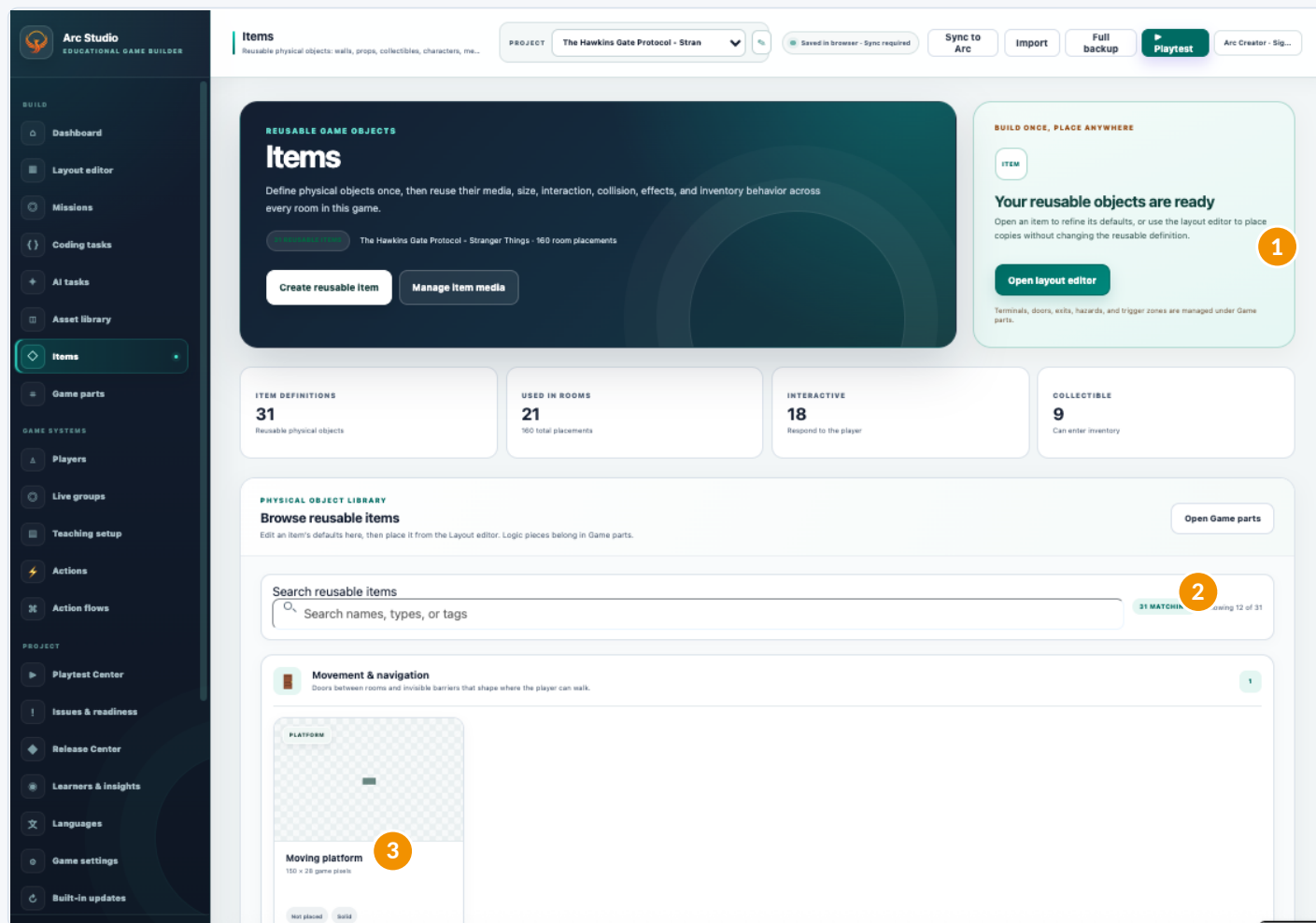
Asset editing includes names, tags, media replacement and usage links. Per-game built-in copies can be customized and restored to their shipped media.

IN PRACTICE

A reachable hosted URL and a complete backup are different things. Review media dependencies before sharing a package.

Items: define reusable physical objects.

Create the object once, then place copies where the game needs them.



1 Reusable defaults

Define an item's media, size, interaction, collision and inventory behavior. A template is the reusable definition—not the object's live state in a game.

2 Physical object library

The supplied page separates ordinary objects and collectibles from the logic-oriented Game parts library. Search narrows the current game's reusable items.

3 Copies in rooms

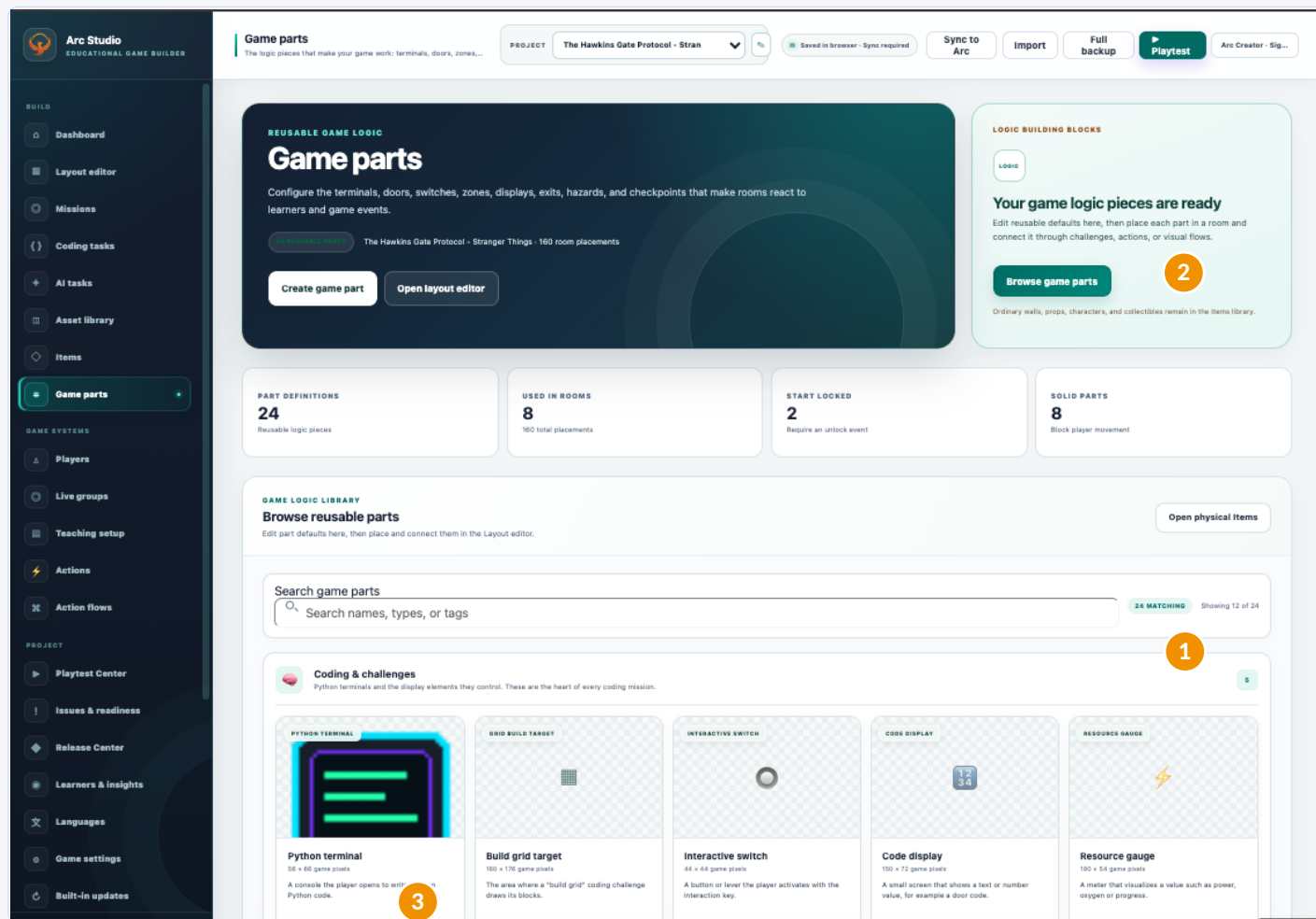
Layout adds placed copies with their own position and selected overrides. Changing a shared default can affect copies that still inherit that setting.

IN PRACTICE

A name is not a behavior: the documented Moving platform kind does not move until motion is explicitly authored.

Game parts: the devices behind the puzzles.

Terminals, displays, switches and other logic pieces give actions something to operate on.



1 Choose a useful part

The pictured coding group includes a Python terminal, build target, interactive switch, code display and resource gauge.

2 Place and connect

Edit reusable defaults here, then place the part in Layout. Bind it to a task, an action or a flow so the intended result reaches the exact device.

3 Separate appearance from logic

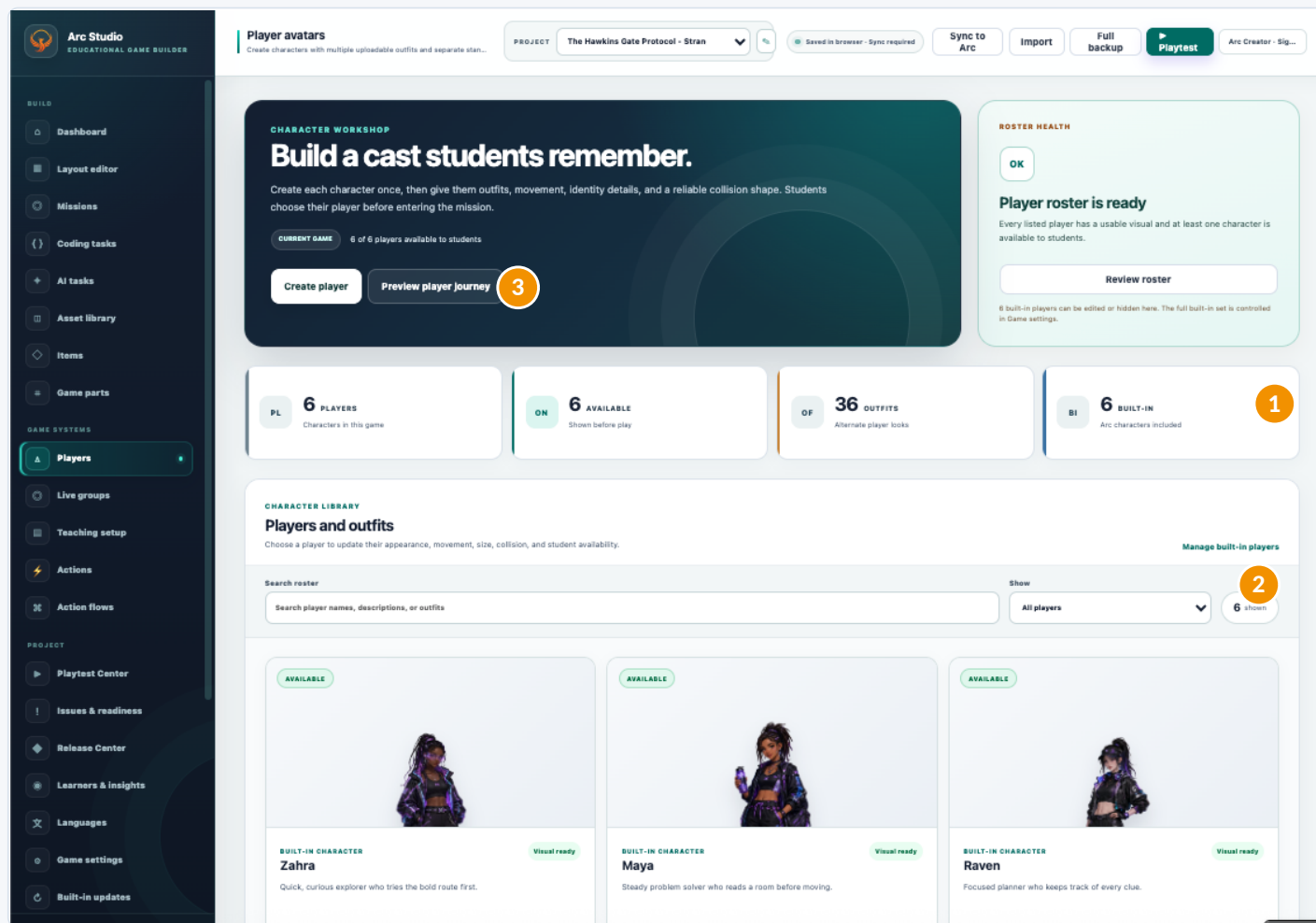
A display can show a captured result; a door still needs an explicit unlock or transition rule. A kit is a starting point, not a complete game loop.

IN PRACTICE

“Not placed” means a definition exists in the library but is not currently used by a room.

Players: build the cast learners can choose.

Characters, outfits and movement settings are managed independently from the rooms.



1 Characters and outfits

The supplied roster shows six available players and 36 outfits. Those are this project's captured counts, not limits on a custom game.

2 Visual and movement identity

Configure pose media, size, speed, facing and selection availability. Standing, moving, jumping and picking-up visuals support different moments in play.

3 Preview the player journey

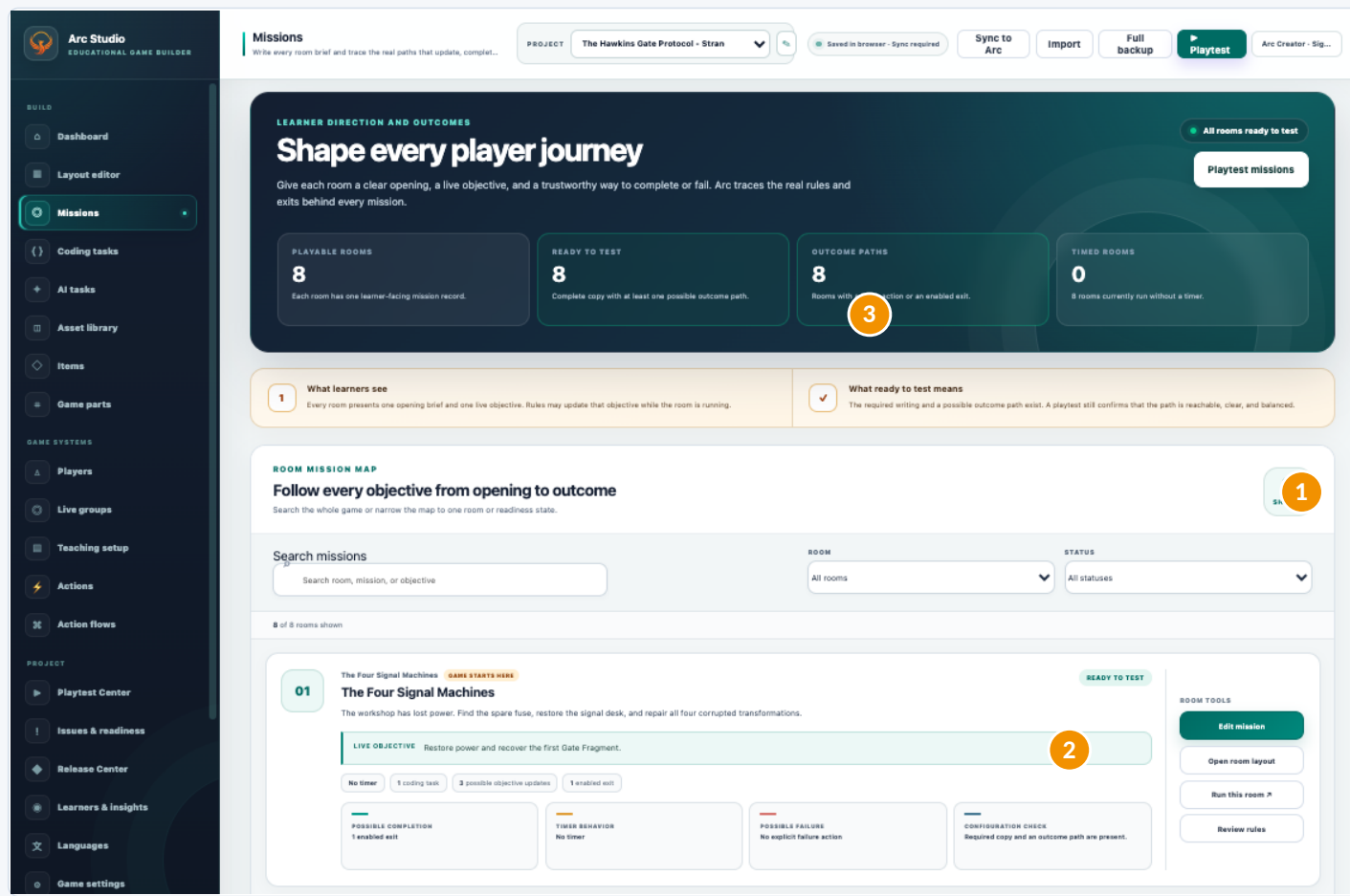
Inspect the roster and preview player choice before testing the game. Set the game's default player and selection policy in Game settings.

IN PRACTICE

Recheck character scale in the room, not only on a roster card. Room depth and floor alignment affect the final appearance.

Missions: give every room a clear purpose.

Review the opening brief, live objective and possible outcome routes in one place.



1 A room-by-room mission map

Search missions or filter by room and status. Each entry links to mission editing, the exact room layout, a room run and its rules.

2 Learner-facing direction

Author a title, subtitle, opening instructions, one live objective, optional countdown, and success or failure messages. Rules can update the live objective.

3 Readiness with meaning

The page traces possible exits and outcome actions. “Ready to test” means required copy and a potential route exist—not that the route has been played.

IN PRACTICE

The documented mission model has one live objective per room, not an independent checklist of scored sub-objectives.

Coding Tasks: author once, place deliberately.

Keep the Python learning content reusable while making every terminal's room effects explicit.

1 The reusable definition

Manage learner instructions, starter code, instructor solution, progressive hints and the success check. Editing the definition updates linked learning content.

2 The room placement

A placed terminal owns its enabled or required state, connected objects, and success or incorrect-result actions. One definition may be used more than once.

3 Find and maintain tasks

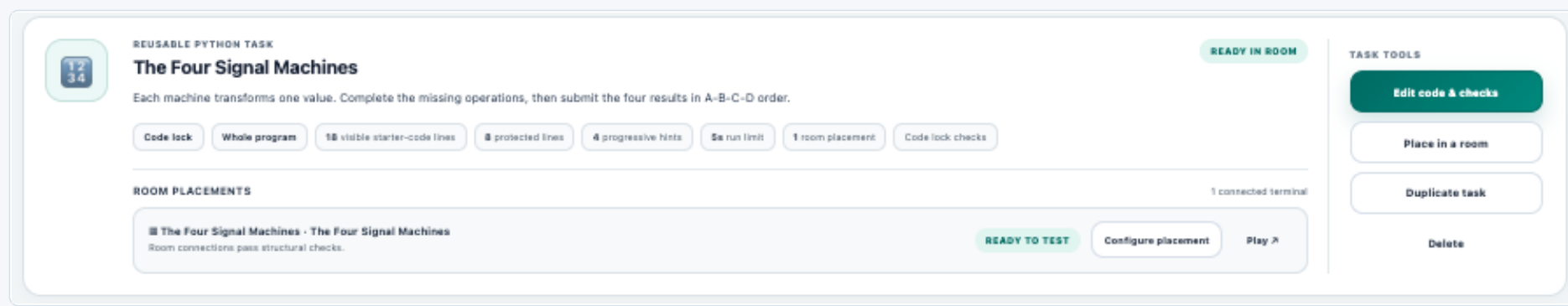
Search and filter the pool. Edit code and checks, place in a room, duplicate a definition or delete an unused task through the task's own tools.

IN PRACTICE

Nine tasks and nine ready placements are shown in this game. "Ready" is a structural check, not a recorded Python playtest.

Choose what learners write—and what changes.

Guided patterns provide a starting point; Custom Python gives the creator control over the check.



FIVE TASK PATTERNS

Build a grid

Fill the configured rows and columns; render captured blocks on a bound target.

Calculate a code

Submit the expected code; show it on the connected display.

Activate a sequence

Capture the required switch names in the exact authored order.

Balance resources

Match the creator's named resource targets and update their gauges.

Custom Python

Use exact or partial printed output, or creator-written Python assertions.

VISIBLE CODE MODES

Whole program

Show the entire editable answer.

Function content only

Keep the function wrapper out of the learner editor.

Focused answer region

Expose only the intended lines.

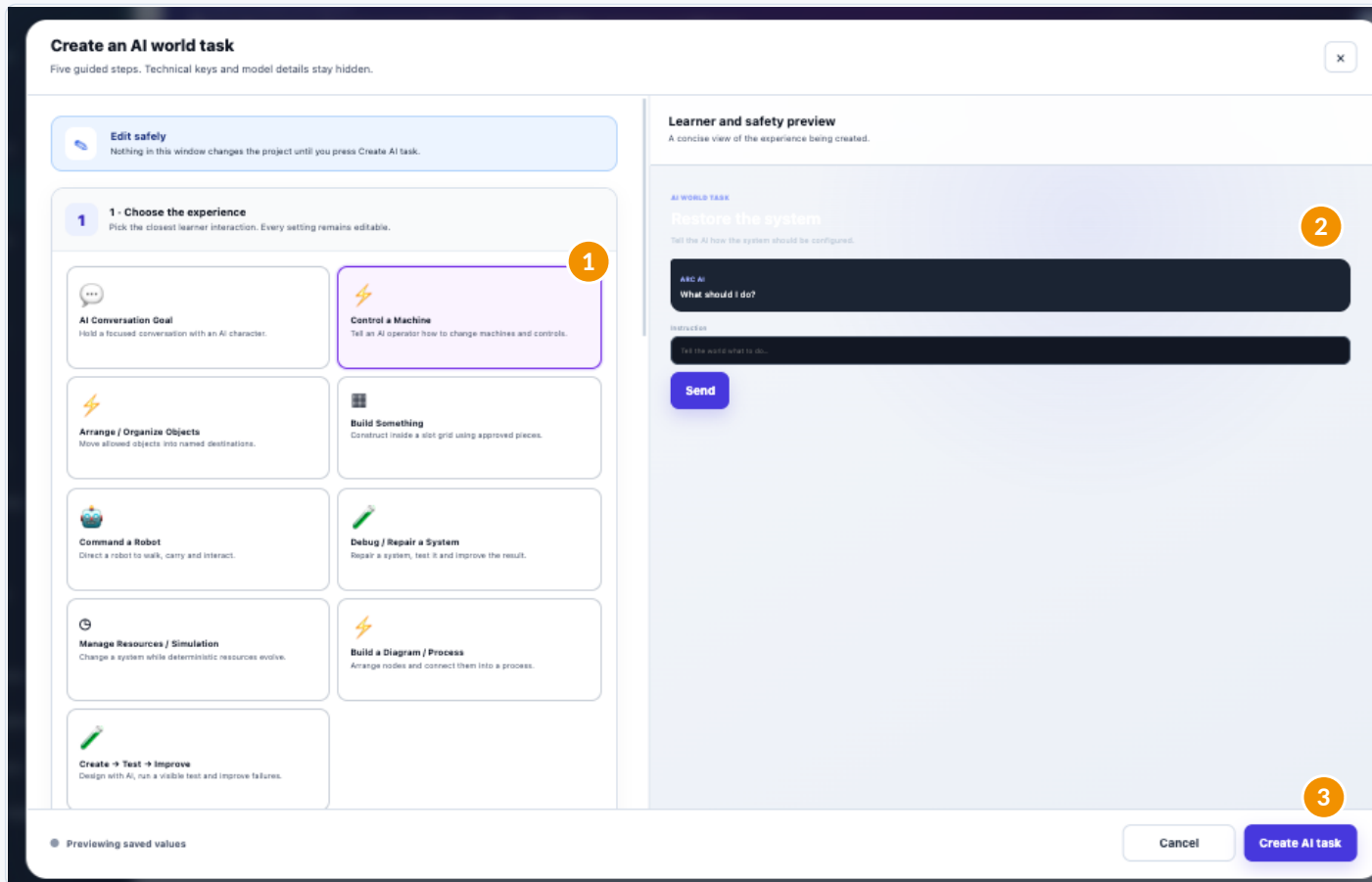
FORMATIVE PRACTICE

Python runs in the learner's browser. Hidden editor fields are not a secure exam boundary.

Sources: SCREEN 08 (enlarged card); [S1] Coding Tasks; [S4] Python challenges and bound feedback.

AI Tasks: make instructions change the world.

A guided wizard starts with the learner experience, not with provider keys or technical field names.



1 Choose the interaction

Select a guided experience such as conversation, machine control, construction, robot control or create-test-improve.

2 See the learner surface

The right-hand preview shows the task brief and conversation area. The screenshot is a creation preview, not evidence of a live provider turn.

3 Configure before creating

The wizard labels changes as temporary until Create AI task. The documented editor adds bindings, permissions, zones, resources and success rules.

IN PRACTICE

AI Tasks are separate from Python tasks. A learner can use natural language while the authored runtime controls the permitted effects.

Nine ways to begin an AI experience.

Nine guided presets cover focused conversations, controlled world changes and iterative challenges. Each remains configurable.

01 AI Conversation Goal

Hold a focused conversation with an AI character.

FOCUSED DIALOGUE

02 Control a Machine

Tell an operator how to change allowed machine settings.

MACHINE SETTINGS

03 Arrange / Organize Objects

Move permitted objects into named destinations.

OBJECT ARRANGEMENT

04 Build Something

Construct in a slot grid using approved pieces.

SLOT-BASED BUILDING

05 Command a Robot

Direct an agent to move, carry and interact.

MOVEMENT + CARRYING

06 Debug / Repair a System

Repair a configuration, test it and improve it.

REPAIR + RETEST

07 Manage Resources / Simulation

Change a system while bounded resources evolve.

EVOLVING RESOURCES

08 Build a Diagram / Process

Connect allowed nodes into a readable process.

NODES + CONNECTIONS

09 Create → Test → Improve

Design a solution and revise it after a failed test.

ITERATIVE DESIGN

Sources: SCREEN 09; [S1] AI Tasks; [S2] AI task modes, zones, agents and evaluation; [S8] AI checkpoint.

Two AI roles: guide the learner, change the world.

ARC documents two distinct AI roles: helping the learner and performing bounded world tasks.

HELP THE LEARNER

AI guide / in-room help

Configure a persona, tone, greeting and object hints. The guide can use live room, inventory and terminal context to help a learner who is stuck.

This is a help conversation. It is not automatically authorized to rebuild the room or complete a task.

CONTROL ALLOWED WORLD CHANGES

AI world task / permitted actions

Choose the objects, states, zones and actions the task may use. The provider proposes structured commands; the runtime checks the whole batch before applying it.

Construction, agents, resources and design tests feed explicit success criteria.

THE CONTROLLED PATH



OWNER-MANAGED AI SETUP

Organization → AI & Models controls the provider credential, approved models, quotas, token and timeout bounds, enable switch and connection test. Credentials do not travel in game packages.

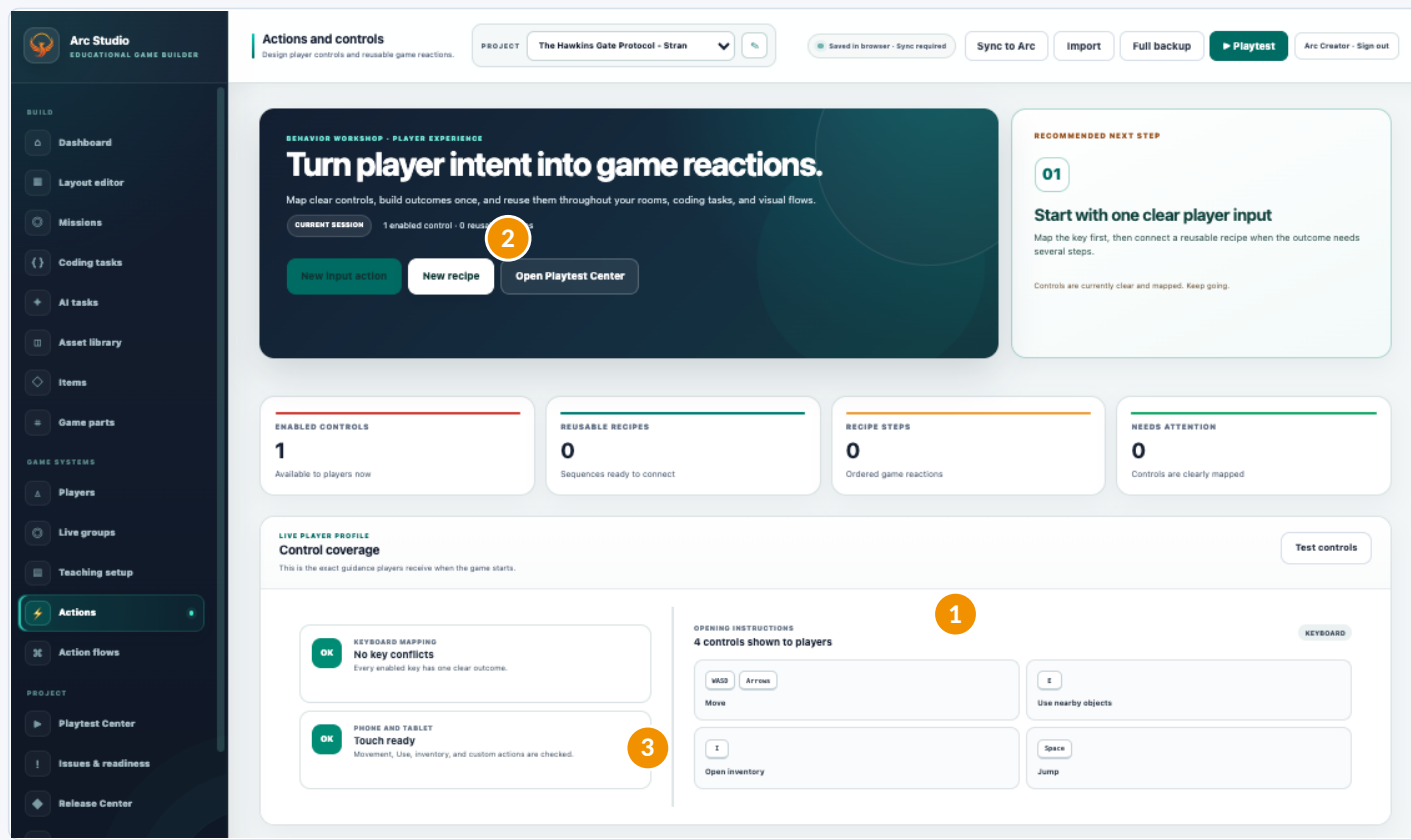
PRIVACY AND QUALIFICATION

AI world-task usage records exclude prompt and response prose. That is not a claim that no text reaches a provider: the guide can send bounded learner code and error context. Real-provider quality still needs qualification.

Sources: [S1] AI Tasks and provider setup; [S4] AI guide and AI task lifecycle; [S6] Configuration and privacy; [S8] Qualification limits.

Actions: turn player intent into a reaction.

Manage input controls and reusable sequences without writing the game engine.



Control coverage is visible in the supplied screen; action and recipe editors are described in the handbook.

1 Map clear controls

Create an input action for a keyboard key and its behavior. The coverage section shows the movement, interaction, inventory and custom controls players receive.

2 Build reusable recipes

A recipe is an ordered sequence of configured actions. Reuse it from an input, task outcome or flow instead of recreating the same reaction.

3 Check control coverage

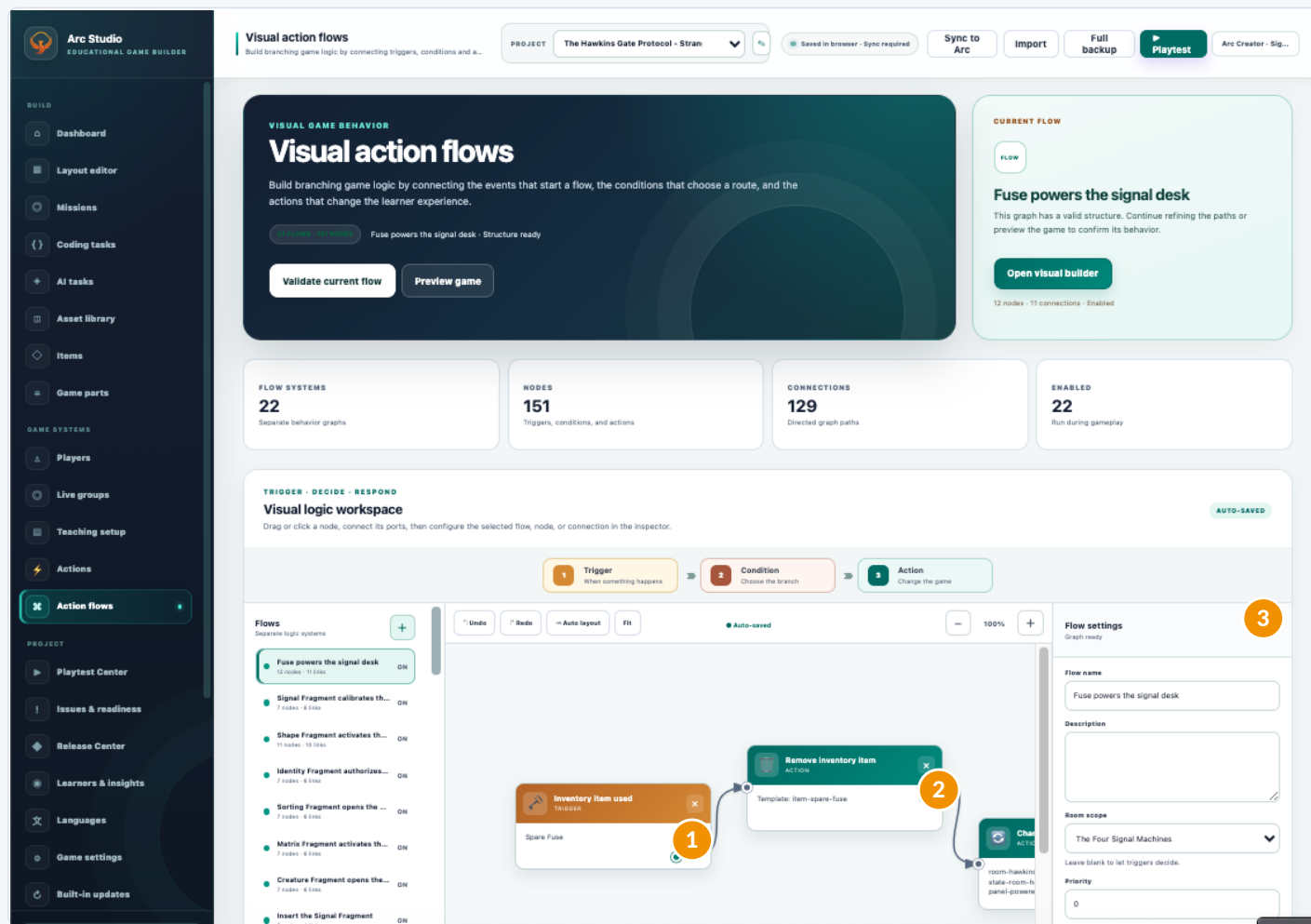
Conflicting keys are rejected. The current runtime documents touch routes for supported actions; the actual learner device still needs testing.

IN PRACTICE

The pictured game has one enabled control and no recipes. That describes this game's setup—not a missing recipe feature.

Action flows: connect events to consequences.

A visual graph expresses when something happens, what condition matters and what the game should do.



1 Trigger

Start from an event: a room begins, an item is used, a task succeeds or a state changes. The pictured flow starts with use of the Spare Fuse.

2 Condition and action

Conditions select a route. Actions change the world: consume an item, power a device, update an objective, reveal a reward or open a route.

3 Manage the graph

Set room scope, priority and run-once behavior. Connect nodes by ports or labeled outputs, then validate references and test the intended branch.

IN PRACTICE

Drawn branches execute in sequence, not simultaneously. A valid graph still needs a real playthrough.

Design the whole loop—not only the answer.

Illustrative authoring pattern using the documented inventory, states, task outcomes and flow system.

01

Explore & collect

Place a collectible item on a reachable route.

ITEMS + LAYOUT

02

Use on a device

Match inventory use to the exact socket or control.

ACTION FLOW

03

Activate the task

Switch the device on and enable the console.

ITEM STATE + ACTION

04

Solve & react

Evaluate the answer and show a visible response.

CODING / AI TASK

05

Recover the reward

Reveal an item, then let the learner collect it.

OUTCOME + INVENTORY

06

Open the next route

Author the door unlock and next-room destination.

FLOW + EXIT

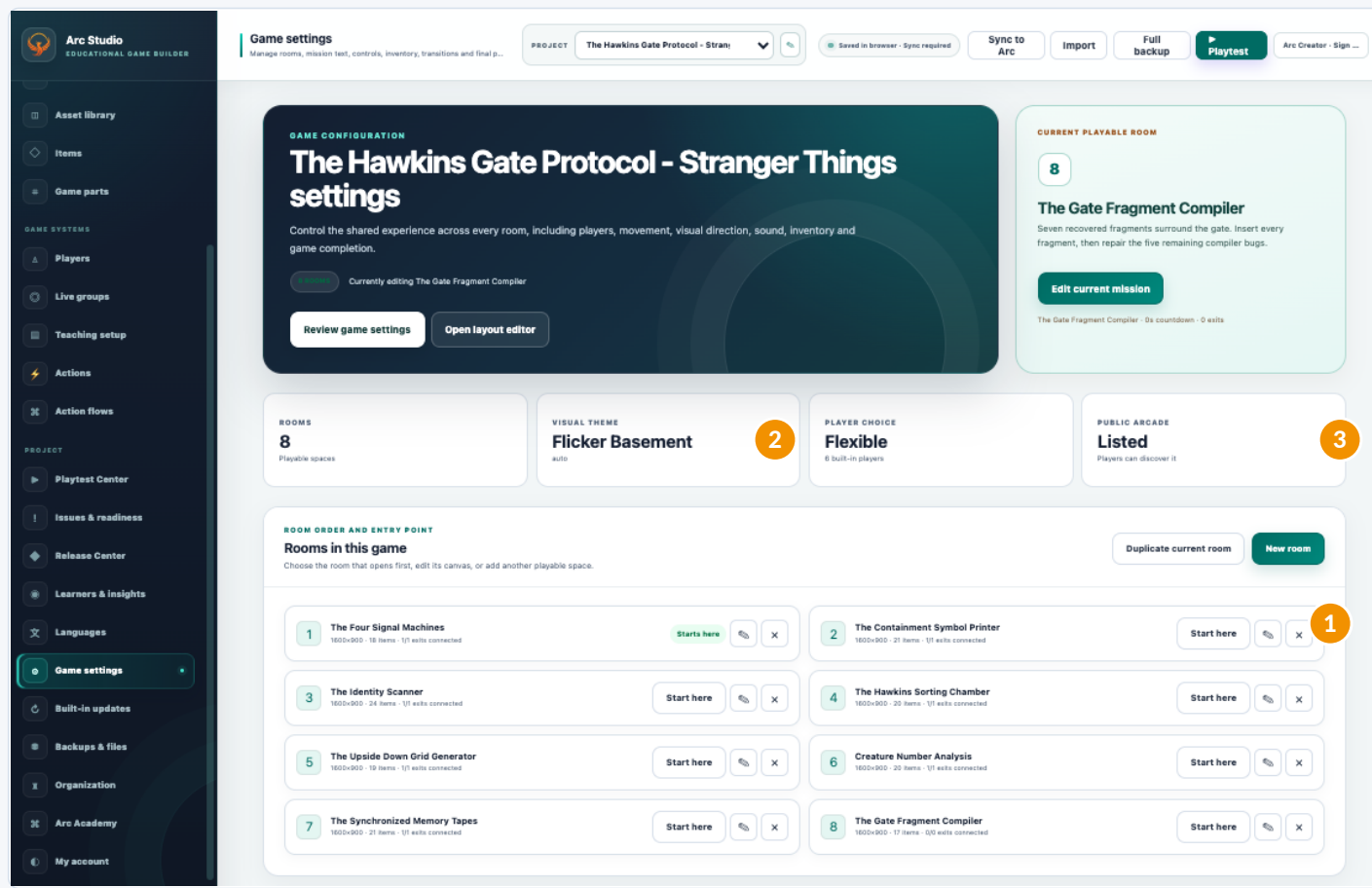
THE CREATOR CHOOSES THE PROGRESSION

A correct code value does not automatically grant a physical reward or unlock the next door. Those consequences must be connected to the task, object states or a flow.

Sources: [S1] Coding Tasks, item states, Actions and Flows; [S4] Inventory, Python feedback, doors and completion. Diagram is illustrative.

Game settings: define the shared experience.

Manage the game's rooms, beginning, controls, presentation and learner-facing screens.



1 Rooms and entry point

Add, edit, duplicate or remove rooms. Start here selects the room a new learner run begins in; editing a later room does not change the beginning.

2 Game-wide choices

Set player choice, inventory, touch behavior and presentation. Rooms can inherit visual settings or use their own overrides.

3 Public-facing identity

Configure the game card and arcade listing, then the opening and completion experience. Listing and private lesson access are different delivery choices.

IN PRACTICE

Duplicating a room is not a complete behavior clone. Recheck flow targets, bindings and exits before treating it as independent.

Style, sound and story—from start to finish.

The documented Settings editors control how the authored game feels, not just which room opens first.

LOOK

Visual direction

Choose a theme, projection and atmosphere. Tune horizon, near/far scale, shadows, glow and motion. Rooms may inherit the game direction or override it.

SPACE

Scale and floor alignment

Calibrate the reference player height. Relative object sizing and player depth scaling help furniture, characters and perspective work together.

SOUND

Music and response sounds

Use background music with room overrides, plus sounds for interaction, collection, item use, doors, reveals and task results.

STORY

Opening and completion

Set titles, messages, labels and optional image or video. The full-game ending is separate from each room's mission messages.

CONTROL

Keyboard and touch

Configure interaction, inventory, movement scheme and portrait handling. The runtime supports up to three touch-exposed authored input actions.

DELIVERY

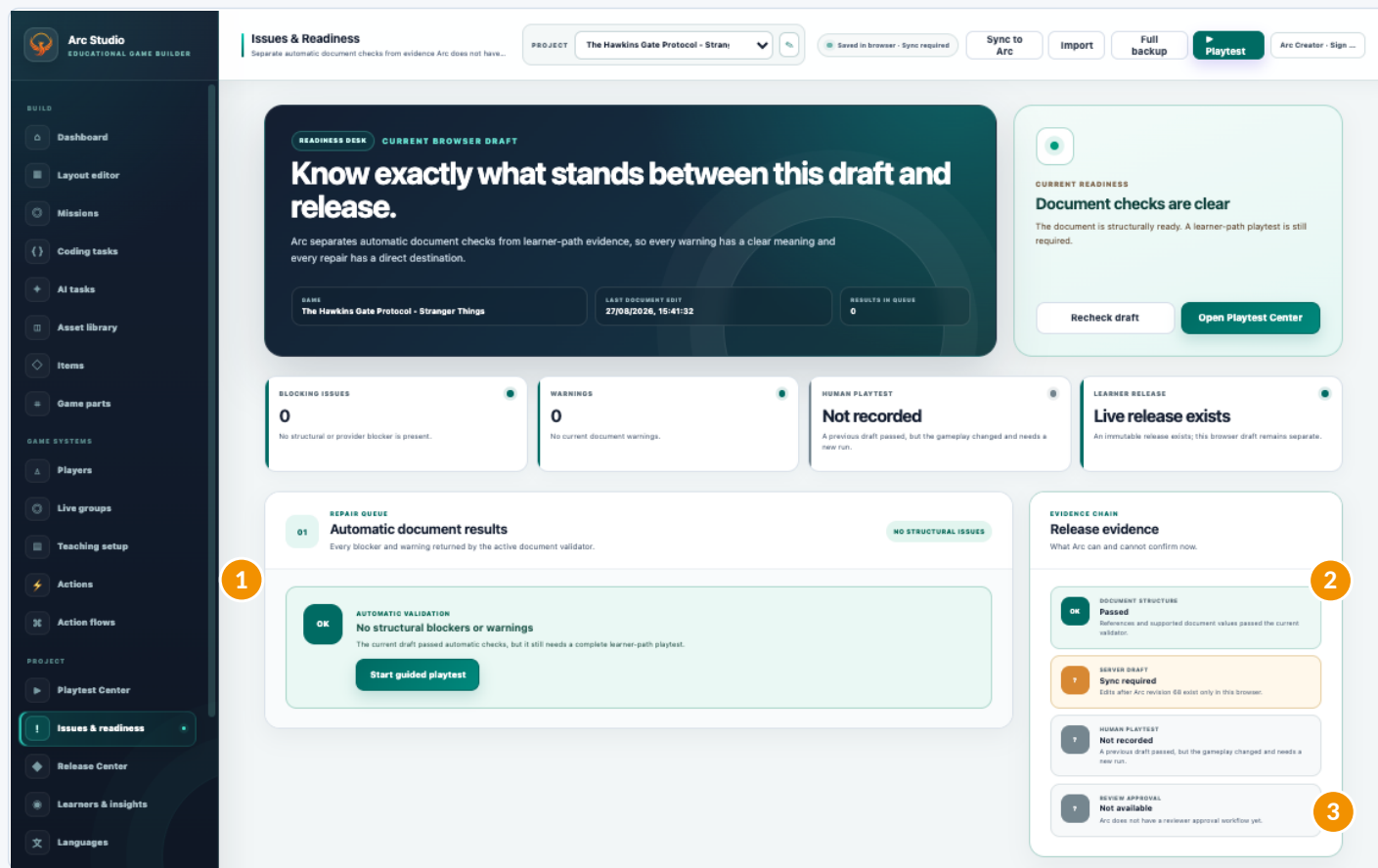
Player-safe navigation

Launched learner sessions use contextual return or end-session actions. Creator-preview navigation is not a promise about classroom exit buttons.

Sources: [S1] Settings and editor previews; [S2] Visual direction, game sounds, opening and completion screens; [S4] Controls and screens.

Issues & Readiness: separate facts from confidence.

A clean configuration, a synced draft and a tested learner journey are not the same result.



1 Automatic checks

Review structural or provider blockers and warnings. Filter the result list and follow repair links to the relevant editor area.

2 Evidence chain

Read document checks, server-draft state, human playtest and live-release evidence separately. An existing release does not validate changed browser work.

3 Know what is not proven

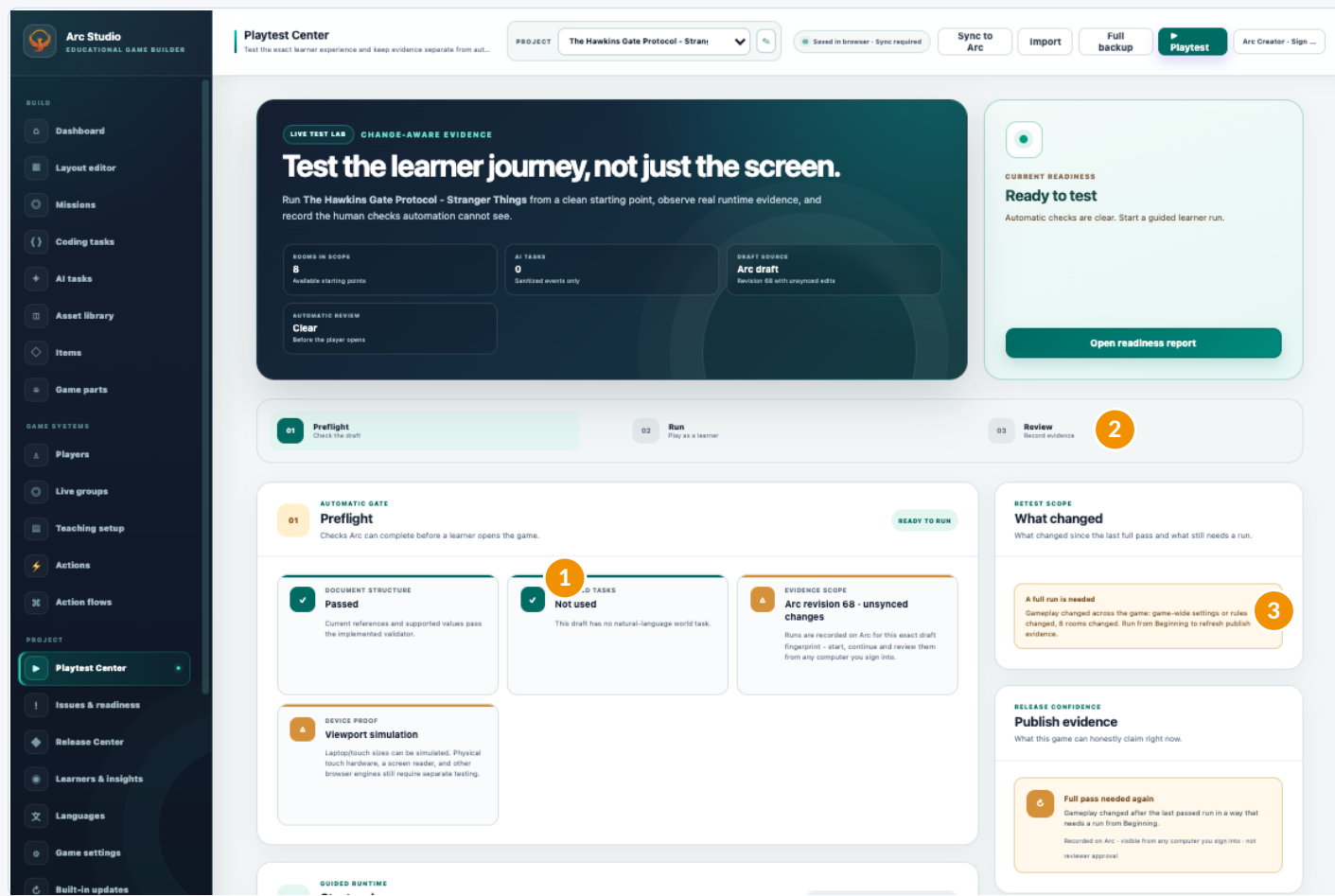
The screenshot is structurally clear while playtest evidence is not recorded for this draft. Reviewer approval is shown as unavailable.

IN PRACTICE

Use Recheck after a repair, then test. A validator cannot judge whether a clue is clear or the whole route is enjoyable.

Playtest Center: test the learner journey.

Run the actual player, observe what happened and record the human checks automation cannot make.



1 Preflight

Inspect the draft and choose a starting point or viewport. The captured screen identifies changes affecting all eight rooms and requests a full run.

2 Run and review

A full-path test starts from the configured beginning. After completion, the documented review covers controls, objective clarity, task applicability and completion feedback.

3 Retest changed work

A room-only run is useful targeted evidence, not proof of the complete route. Selected-room tests can supply earlier collectible items for creator testing.

IN PRACTICE

Viewport simulation is not a physical-device test. The sources disagree on record sharing; see the edition notes on page 38.

Release Center: keep drafts and live games separate.

Publishing creates a fixed learner version; later editing does not rewrite that release.

The screenshot shows the Arc Studio Release Center interface. The top navigation bar includes the Arc Studio logo, the project name 'The Hawkins Gate Protocol - Stranger Things', and buttons for 'Sync to Arc', 'Import', 'Full backup', 'Playtest', and 'Arc Creator - Sign out'. The main content area is divided into several sections:

- EXACT SERVER GAME:** Displays the game title and draft revision 68. It includes buttons for 'LIVE FOR NEW LAUNCHES' and 'SYNC REQUIRED'.
- RECOMMENDED NEXT STEP:** A yellow box with the title 'Sync the browser draft' and a button 'Sync before release'.
- CANDIDATE REVIEW:** A section titled 'Release candidate' with a 'BLOCKED' status and a red circle with the number 1. It contains four sub-sections:
 - SERVER DRAFT:** 'Needs sync' with a 'Sync now' button.
 - STRUCTURAL CHECKS:** 'No blockers' with a green 'OK' button.
 - LEARNER PATH EVIDENCE:** 'Not passed' with an 'Open Playtest Center' button.
 - PUBLICATION APPROVAL:** 'Manual confirmation required'.
- Live release:** A section with a green circle with the number 2. It shows statistics: 0 active assignments, 5 active attempts, and 0 open groups. It includes buttons for 'Open as test learner', 'Pause new launches', and 'Monitor learners'.
- LEARNER ACCESS:** A section titled 'Distribution now' with a 'Distribution now' button. It includes a table of learner access options:

Third-party lessons	Creator learner test	Permanent public/share link	Scheduled publishing
Creates an exact-release assignment or one-time embedded launch from Teaching setup. Open →	Creates a real new learner attempt for the active release.	Player entry currently requires an authorized one-time launch grant.	Availability windows are not implemented in the release service.
AVAILABLE	AVAILABLE	NOT AVAILABLE	NOT AVAILABLE
- Draft versus live version 10:** A section with a green circle with the number 3. It shows a comparison of draft and live versions across five categories: Assets (8), Words (211), Players (6), Access (1), and Runs (22).

1 Review the candidate

Check the synced server draft, document checks and learner-path evidence before release. This captured candidate still needs sync and renewed testing.

2 Operate the live release

Open a real test-learner launch, pause new launches, or review learner use. Pausing new entry preserves existing saved progress and active work.

3 Understand version impact

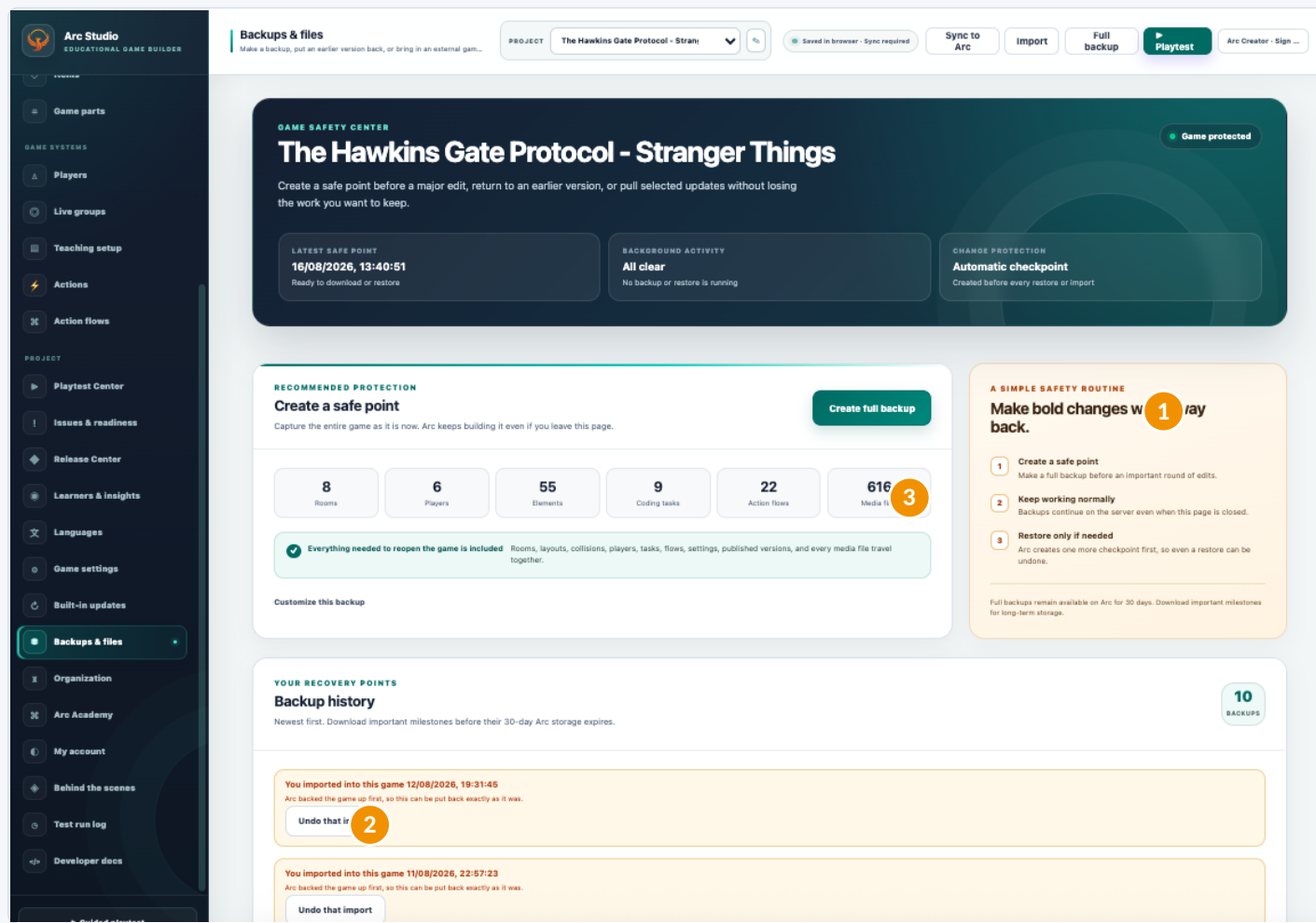
History keeps fixed releases. Reactivating an earlier release does not rewrite progress or automatically move assignments from their pinned release.

IN PRACTICE

The screen shows manual publication confirmation; the audit describes a stricter playtest gate. Confirm the deployment policy before release.

Backups & files: keep a way back.

Create recovery points before major changes and review exactly what the package contains.



1 Create a full backup

A bound game is backed up by the server, so the job can continue after leaving the page. The screen shows the latest safe point and background activity.

2 Protect and restore

Backup history offers download and restore. Destructive restore or import workflows create a recovery point so the change can be undone.

3 Choose the right package

Full backups can carry content, used media, translations and retained versions. Configuration-only and portable profiles have different media dependencies.

IN PRACTICE

The pictured page states a 30-day Arc availability window. Download important milestones rather than treating that window as permanent storage.

Move a whole game—or only the parts you need.

Import analysis is a review step. It is separate from applying changes to a draft.

01

Create a new game

Bring the package into a separate, unpublished game. New identifiers are reconciled with imported references.

02

Replace a current draft

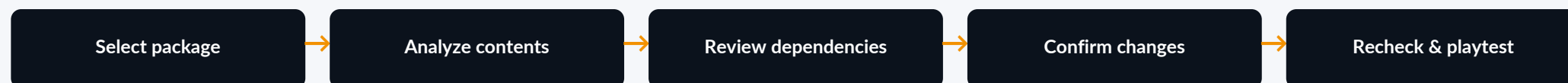
Replace selected game content through the destructive-import workflow, with a recovery point before the change.

03

Merge selected domains

Choose areas such as layouts, items, assets, players, coding tasks, AI tasks, actions or flows. Review the dependencies brought along.

THE DOCUMENTED IMPORT ROUTE



CONTENT TRAVELS; CREDENTIALS DO NOT

AI task definitions and allowed world behavior are part of package 2.1. The receiving organization must configure its own provider and approved model. No AI provider secret belongs in the game ZIP.

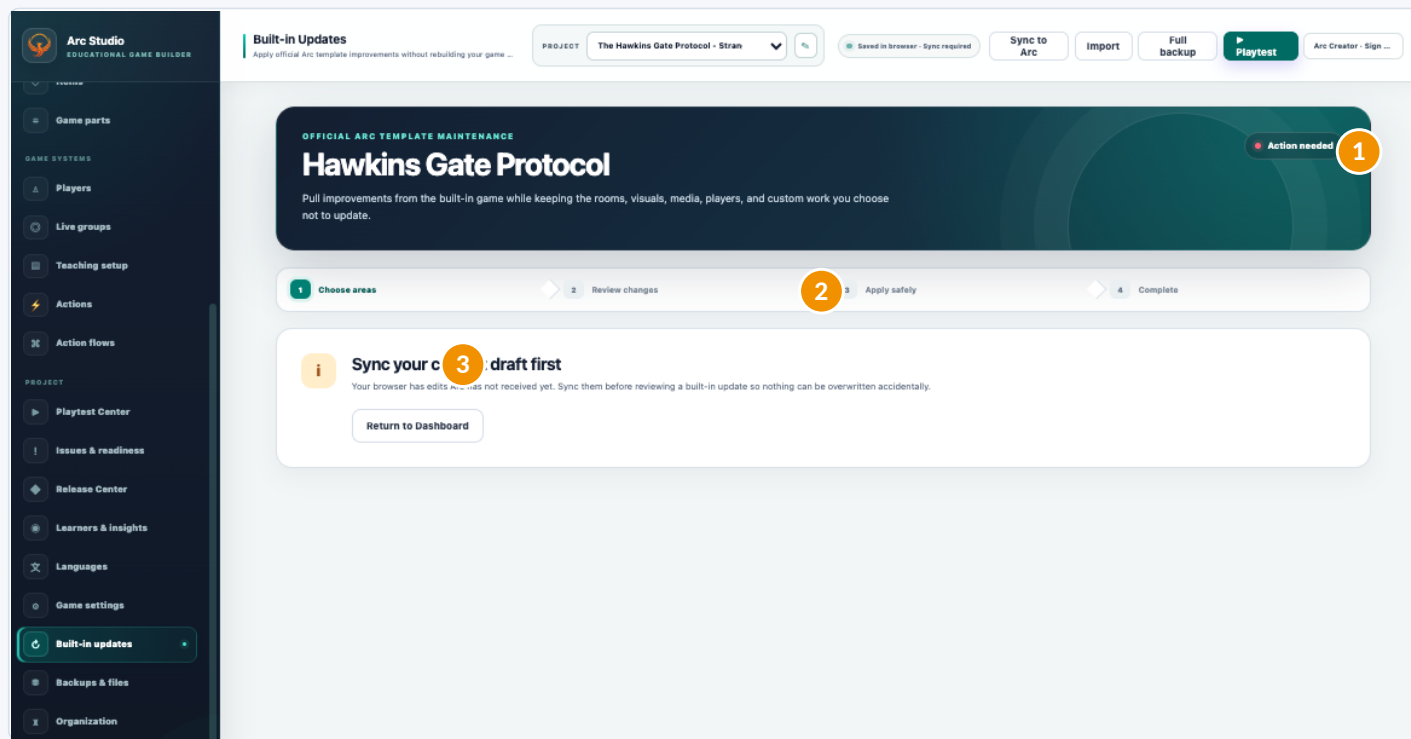
AN IMPORT IS NOT A PUBLICATION

Read media-completeness warnings and selective-merge decisions. Instructor solutions can be present in creator backups, so treat these packages as creator material.

Sources: [S1] Backups & import; [S5] Package format and import; [S7] Portability; [S2] Game package format.

Built-in Updates: improve a starter without starting over.

This is built-in game-template maintenance—not a general server-software updater.



The attachment is named server_updates.png; the actual workspace title is Built-in Updates.

1 Choose what to update

Pull improvements from the built-in game while preserving the areas you choose to keep, such as rooms, media and custom visuals.

2 Review before applying

The workflow shows Choose areas, Review changes, Apply safely and Complete. The documented update uses the package importer and a recovery point.

3 Resolve the save state first

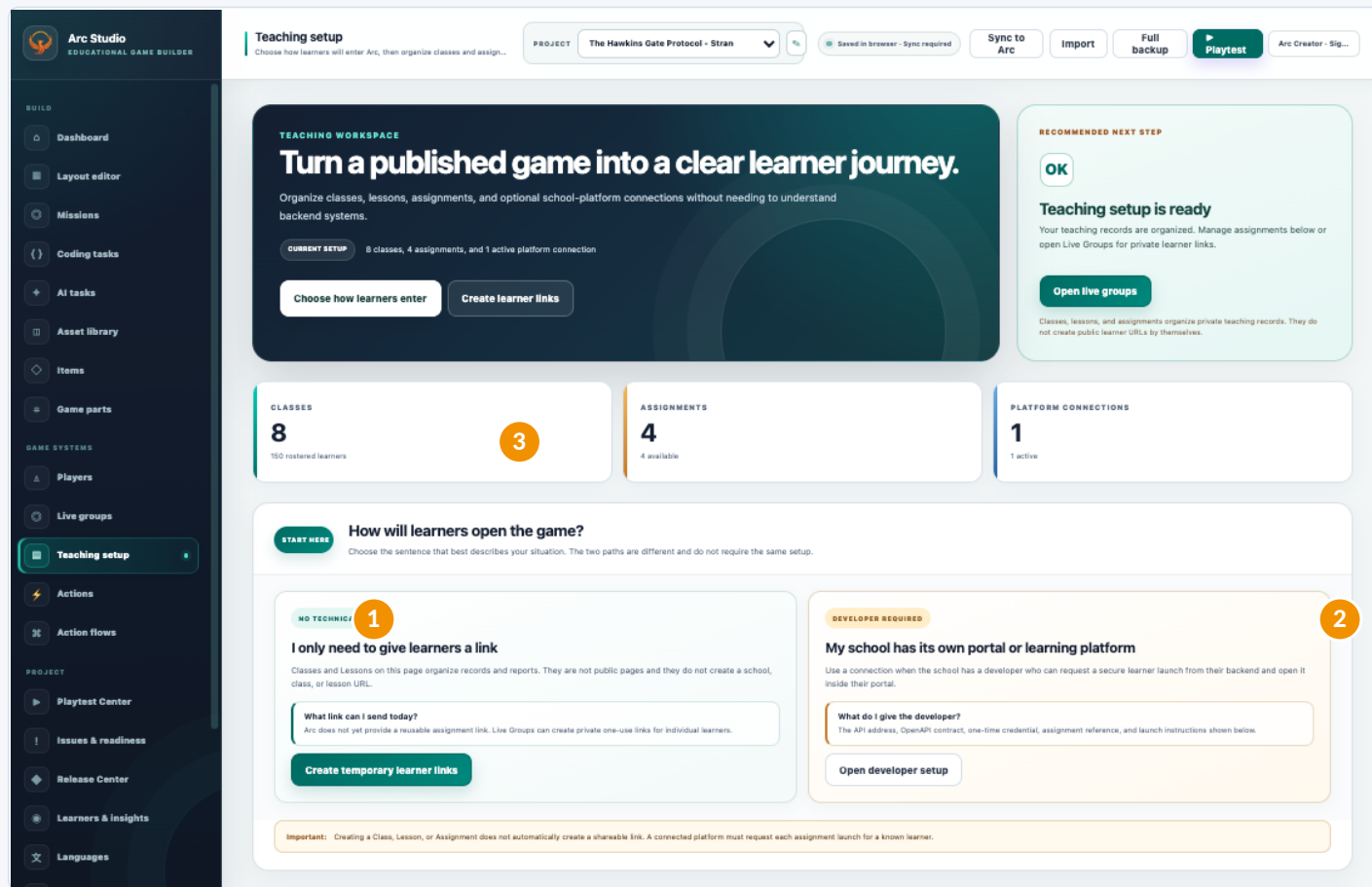
The captured screen stops at Sync your current draft first. Later merge controls and the completed result are not shown in this screenshot.

IN PRACTICE

The Dashboard preview distinguishes areas that will update from areas that will keep the current game's work.

Teaching setup: choose how learners enter.

Organize classes, lessons and assignments, then connect them to the appropriate launch route.



1 A simple link-based activity

The page directs creators who only need learner links toward Live groups and temporary private links. Creating a class does not create a public class page.

2 A connected school platform

A school portal can request a secure learner launch through a backend integration. This route requires developer setup; it is not just an embed pasted anywhere.

3 Teaching records and policy

Manage classes, rosters, lessons and assignments. Each assignment uses an exact published release and can set time, attempt and group policies.

IN PRACTICE

A class, lesson or assignment is not automatically a reusable share link. The supplied screen makes that distinction explicit.

Support the learner without erasing the history.

Documented assignment and participant tools extend the teaching overview shown on the previous page.

ASSIGN

Set the learning context

Choose the published release, optional class and lesson, availability window, attempt allowance and play mode.

Fixed and dynamic group modes require a class roster.

GROUP

Place learners deliberately

Fixed groups need an explicit placement before launch. Dynamic entry can place learners in the least-filled configured eligible group.

Moves preserve earlier evidence but revoke access to the old group context.

SUPPORT

Handle an individual exception

Grant extra attempts or a later existing deadline. Remove an exception to return to base policy. Reset the latest eligible attempt without deleting its history.

WHAT THE ATTEMPT HISTORY EXPLAINS

Recorded context. Attempt number, status, times, progress revision, group, checkpoint and resume counts, and reasoned lifecycle operations.

Launch behavior. Eligible unfinished work can resume. A new attempt remains subject to the assignment's policy. An ordinary wrong code result is not automatically a failed mission.

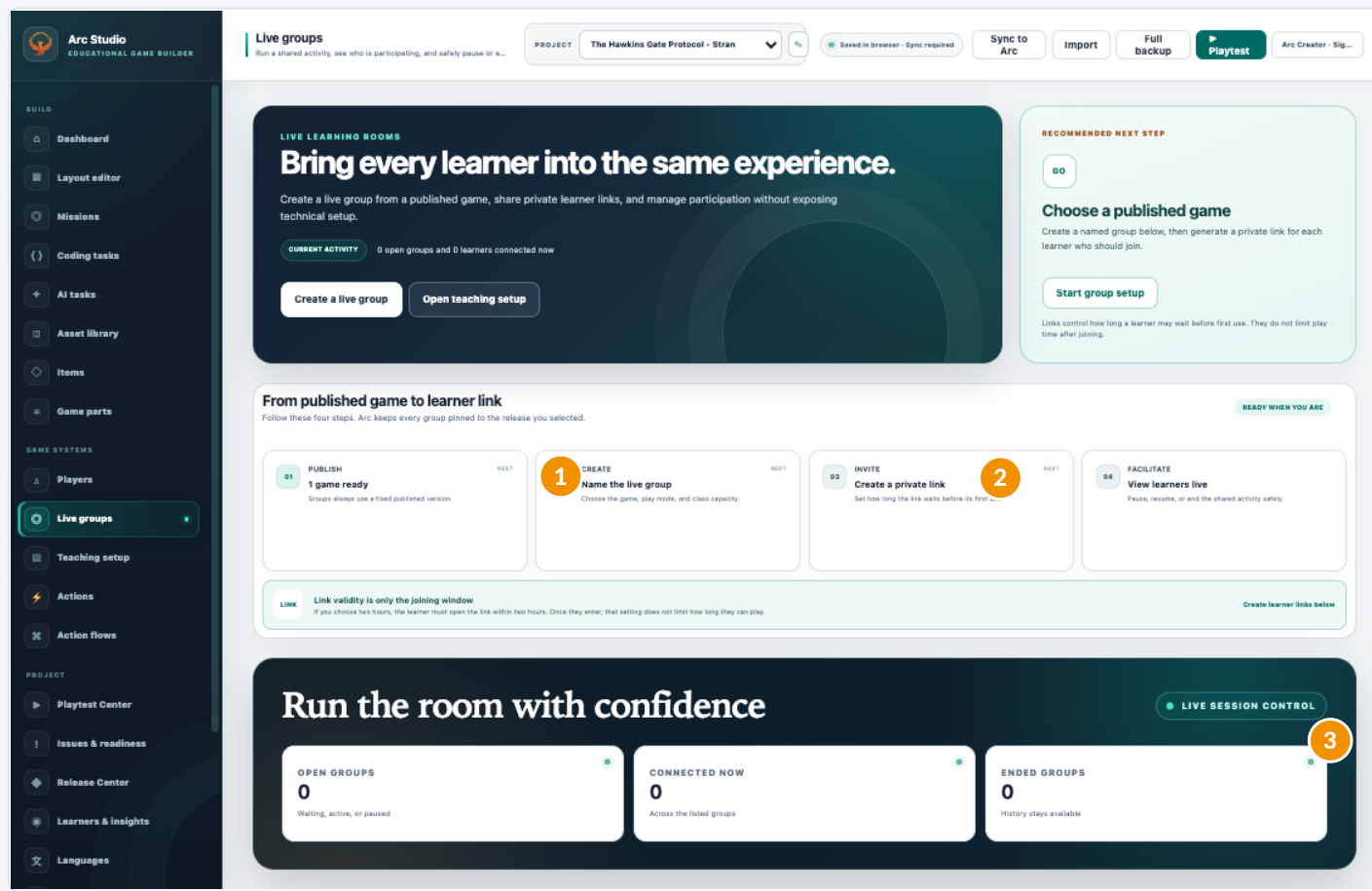
PRIVACY BOUNDARY

The educator history is bounded metadata—not a viewer for learner source code, full saved world state or session tokens. It is not a checkpoint-restore interface.

Sources: [S1] Teaching setup; [S3] Learner workflow; [S9] Assignment, placement and attempt lifecycle; [S7] Administration limits.

Live groups: facilitate an activity in progress.

Create a release-pinned session, issue learner access and manage participation.



1 Start from a published game

Name a group, choose its mode and capacity, and use the configured fixed game version. The screenshot shows the setup state with no open groups.

2 Private learner links

Create a link for an individual learner. Link validity controls the first joining window—not the length of play after joining.

3 Facilitator controls

The documented management view supports pause, resume, end and participant removal, with reasons and retained operational history.

IN PRACTICE

This is not full shared-world replication. Ordinary avatar movement, items and mission actions are not documented as synchronized across players.

Learners & Insights: see progress in context.

Confirmed attempts and observed play signals answer different questions.

The screenshot shows the 'Learners & Insights' page in Arc Studio. The interface includes a sidebar with navigation options like Dashboard, Layout editor, Missions, Coding tasks, AI tasks, Asset library, Items, Game parts, Game systems, Players, Live groups, Teaching setup, Actions, Action flows, Project, Playtest center, Issues & readiness, Release Center, Learners & Insights (selected), Languages, Game settings, and Built-in updates. The main content area is titled 'The Hawkins Gate Protocol - Stranger Things' and features a 'LEARNER EVIDENCE WORKSPACE'. It includes filters for 'EVIDENCE SCOPE' (Choose what you want to understand), 'GAME' (The Hawkins Gate Protocol - Stranger Things), 'TEACHING SCOPE' (All launches for this game), 'GROUP SCOPE' (Choose an assignment first), 'TIME WINDOW' (Last 30 days), and 'EXPORT CURRENT EVIDENCE' (Download learner CSV). Below these filters, there are several summary cards: 'CONFIRMED LEARNERS' (7), 'ATTEMPTS' (12), 'OBSERVED PARTICIPANTS' (12), 'REVIEW SUGGESTED' (7), 'INTERACTIONS' (319), 'MOVEMENT SEGMENTS' (178), and 'OTHER KEY PRESSES' (78). At the bottom, there is a 'Learner evidence' section with a table showing learner progress and activity.

LEARNER	CONFIRMED PROGRESS	OBSERVED ACTIVITY	EVIDENCE TO REVIEW	LATEST
Learner Direct launch	Q1 completed 0 failed - progress revision 7	5 moves - 15 interactions 1 coding - 11 nearby elements	Passed visible elements Coding attempts observed Active interaction	27/08/2026, 12:46:48

1 Choose the evidence scope

Filter by game, assignment, exact assignment group and time window. Exported learner CSV follows the currently loaded scope.

2 Start with confirmed progress

Review learners, attempts and completion or failure status. Keep these records distinct from interactions, movement and other observed activity.

3 Use signals to guide review

The documented page includes learner evidence, room exploration, element exposure and aggregate movement maps. Signals help identify what to inspect or discuss.

IN PRACTICE

Interaction counts and movement are not grades or proof of understanding. The pictured totals are captured example data, not live analytics.

Languages: separate availability from translation.

Enable a language, add it to the game and prepare the wording that will ship.

1 Platform language directory

The administration area defines language code, names, reading direction and fallback. Availability at platform level is distinct from a game using that language.

2 Game-specific text

Creators enable the game's target languages and edit its keyed translations. Publication captures the corresponding version's language bundle.

3 Review the learner experience

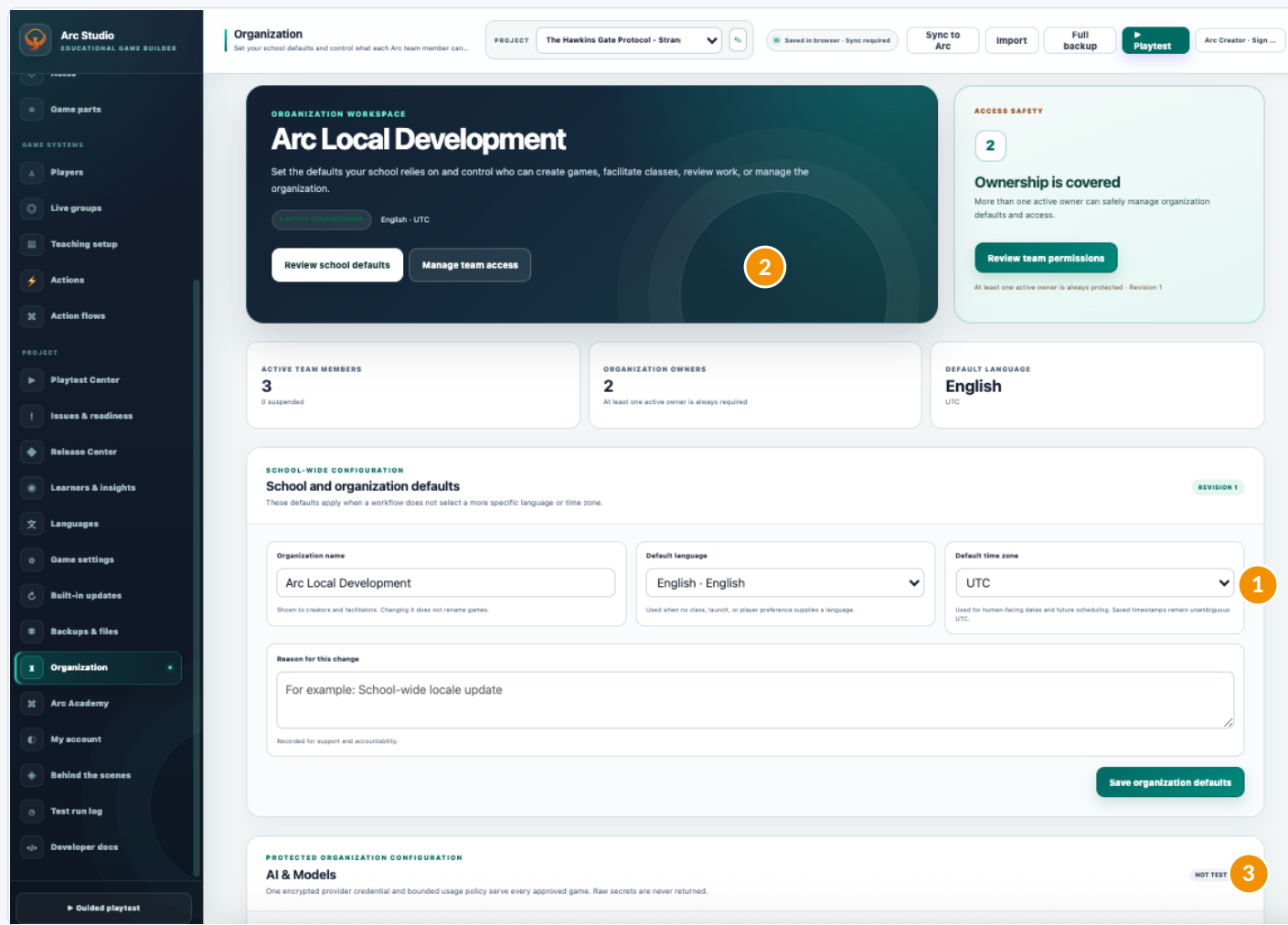
The screenshot's workflow says to approve the text. This does not establish a complete multi-person translation or reviewer-assignment system.

IN PRACTICE

An unfinished form is not a recommended locale configuration. Check direction and fallback before publishing a right-to-left language.

Organization: control defaults and team access.

Owner-level settings govern the organization rather than one room or game.



1 Shared defaults

Manage the organization name, default active language and time zone. Changes carry a revision and a human reason.

2 Existing staff access

Owners can manage already-provisioned staff names, roles and active or suspended status. Safeguards protect the last active owner and prevent self-suspension.

3 AI & Models

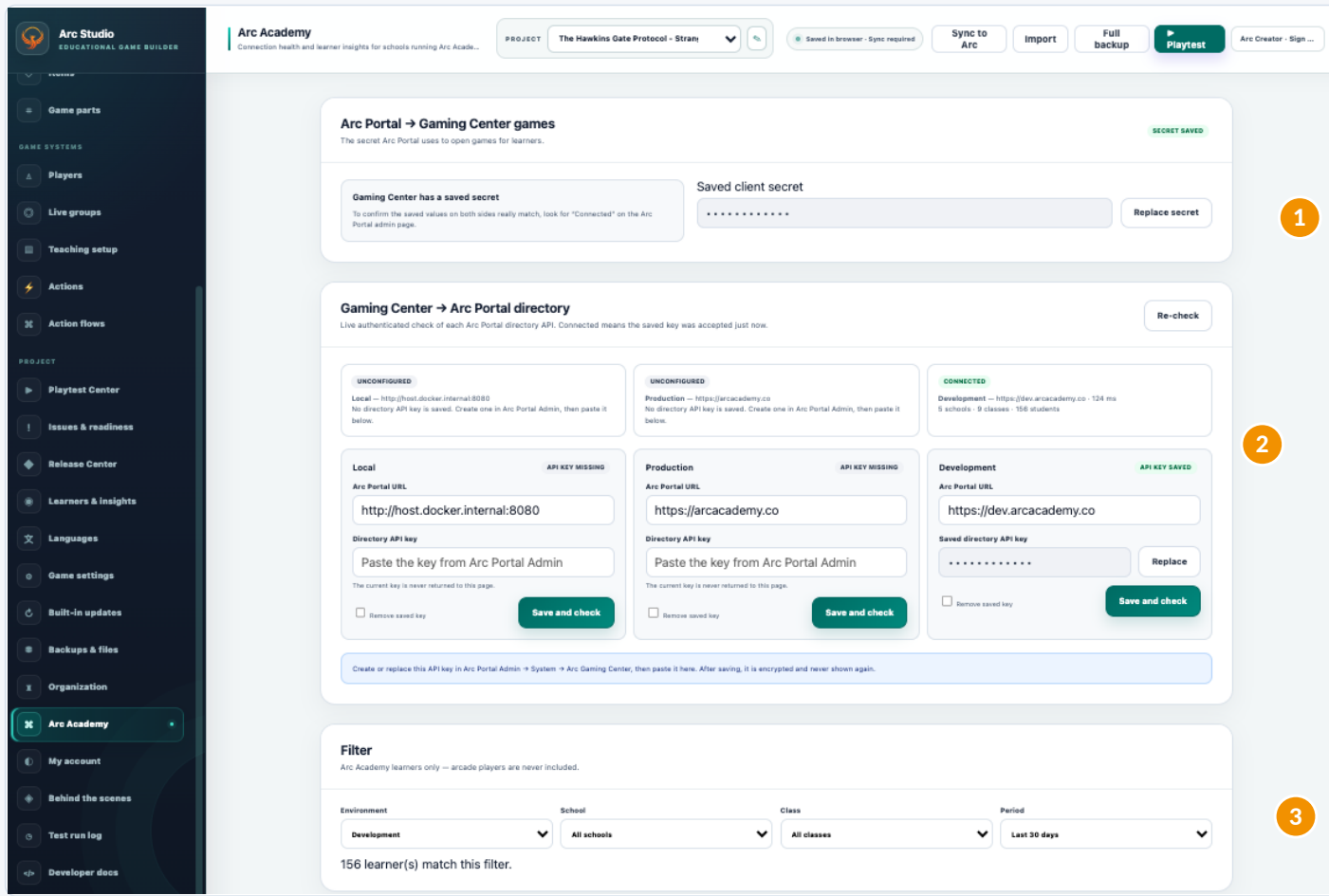
The page includes protected provider configuration. The runbook documents masked credentials, approved models, quotas, enablement and a connection test.

IN PRACTICE

This is not a staff invitation system. Identity creation and invitation acceptance are not implemented in the supplied audit.

Arc Academy: inspect the school-platform connection.

The supplied page separates the game-launch secret from access to the Arc Portal directory.



1 Portal → Gaming Center

The top section shows a saved, masked client secret used by Arc Portal to open games. A saved secret alone is not proof that both systems agree.

2 Gaming Center → Portal

Separate Local, Production and Development cards show directory connection status, portal URL and key controls. The captured Development check is connected.

3 School-oriented scope

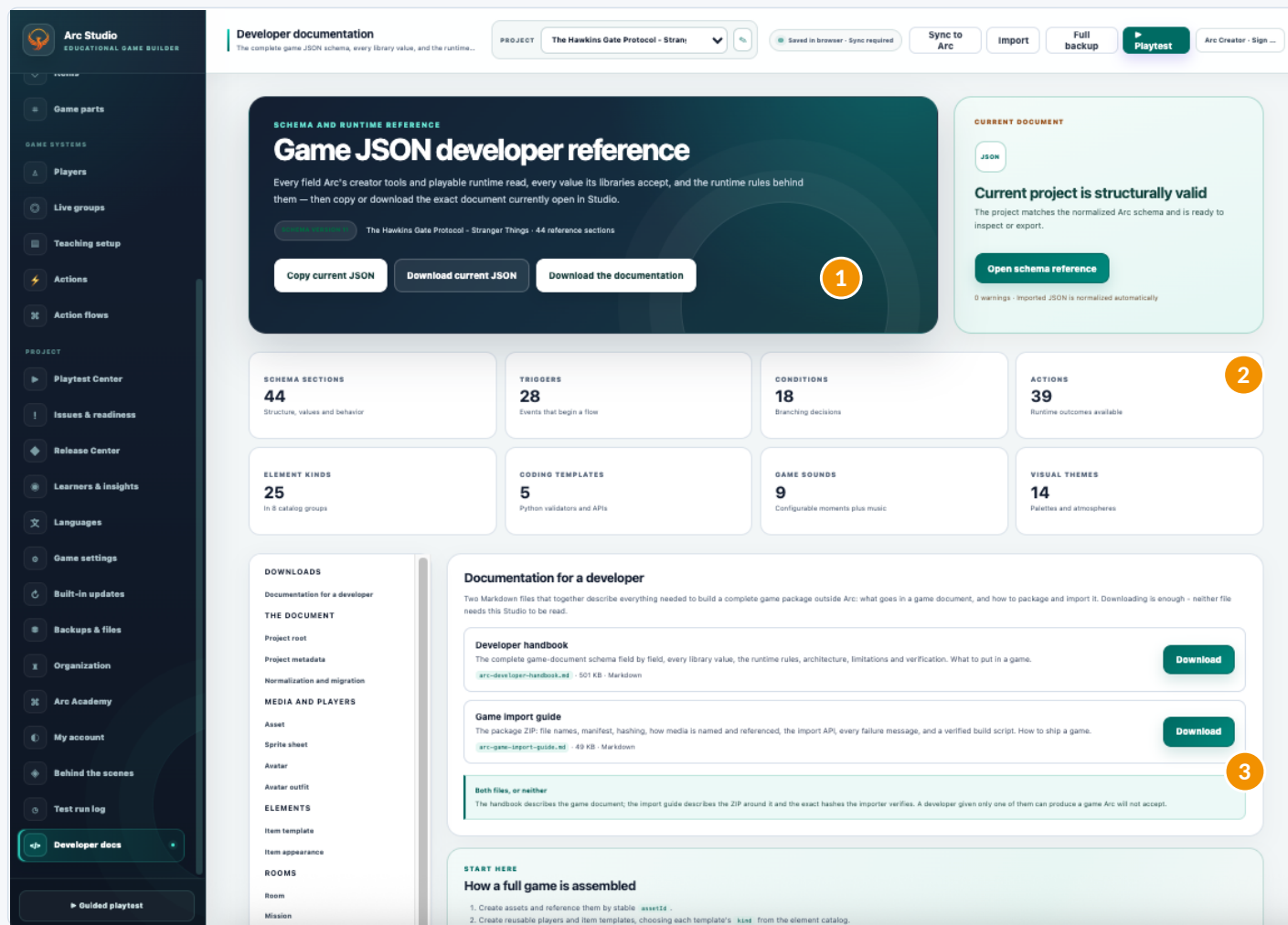
The visible filter selects environment, school, class and period. The page explicitly limits this area to Arc Academy learners, excluding arcade players.

IN PRACTICE

This overview follows the visible screenshot only. The supplied Markdown does not define this page's full backend or reporting contract.

Developer docs: a reference that travels with the product.

Inspect the game document or give an integrator the two files needed to construct an Arc package.



1 Read the current document

Copy or download the normalized game JSON and inspect the schema. JSON describes content; it is not a media-complete backup.

2 Explore the libraries

The pictured reference lists schema sections, triggers, conditions, actions, element kinds, task templates, sounds and visual themes.

3 Share both documents

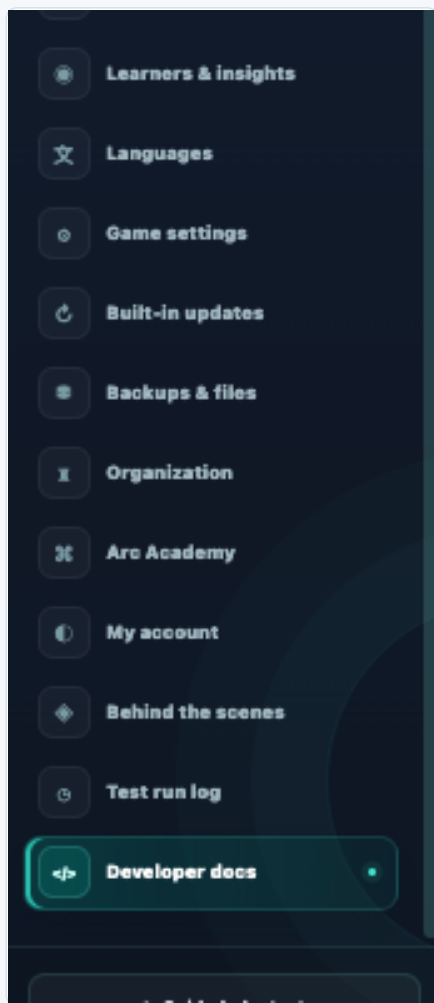
The Developer Handbook explains what goes inside a game. The Game Import Guide explains packaging, media, hashes and import checks. They solve different halves of the handoff.

IN PRACTICE

The supplied reference shows schema 11 and 39 actions, 28 triggers and 18 conditions. Older narrative counts in the docs differ.

Supporting tools beyond the creator workspaces.

Additional entries are visible in navigation; the supplied documents define the local operator tools.



SUPPLIED NAVIGATION DETAIL

Sources: navigation in SCREENS 16, 22–24; [S3] Product surfaces; [S8] Behind the scenes; [S10] Entry ownership.

DOCUMENTED / LOCAL ONLY

Behind the scenes

An owner-only, explicitly enabled local-workstation page explains the backing services in plain language. The documentation describes reachability checks and a request console for labeled API questions.

This is an operator aid, not a learner workspace or a general public administration console.

DOCUMENTED / LOCAL ONLY

Test run log

The documented local acceptance log records each automated run's result, duration and temporary games, including cleanup or archival outcomes.

It explains automated acceptance activity. It is separate from a creator's learner-path Playtest Center run.

NOT EVERY NAVIGATION ENTRY HAS A SUPPLIED PAGE

My account is visible in the screenshots, but no full page or detailed behavior contract was supplied. This guide does not invent account-profile, password, MFA or recovery controls.

A guided tour without sharing a login.

Use these steps to explain the platform in a presentation or stakeholder review.

- 01 Start with the result** 05-06
Show the room editor, then explain how a learner moves, interacts and changes the scene.
- 02 Show reusable learning** 12-16
Compare a Python definition with its terminal placement; then introduce the AI wizard.
- 03 Make the connections visible** 17-19
Use the flow screen and complete interaction loop to explain why the game is more than a quiz.
- 04 Show control over change** 20-27
Explain save versus sync, testing versus validation, fixed releases, backups and template updates.
- 05 Close with teaching and evidence** 28-34
Show how learners enter, how sessions are facilitated and how educators inspect progress.

This is a suggested presentation order, not an application workflow requirement. All page links refer to this guide.

What this edition shows—and what it does not prove.

The screenshots and written audit are useful evidence, but they are not a single live deployment test.

READ THE SOURCES AT THEIR OWN LEVEL

Screenshots establish visible controls, labels and captured states. **Documentation** describes behavior and recorded verification. **Illustrations** explain how capabilities connect; they are not additional product screens. No Studio login or fresh live test was used to make this guide.

Playtest record location

The screenshot says records are kept on Arc and visible across signed-in computers. The audit says browser-local only. This guide does not resolve storage or sharing without deployment evidence.

Publication approval

The Release Center screenshot exposes manual confirmation that can allow publication without a full playtest. The audit describes an exact-draft pass gate. A playtest is recommended here, not asserted as the only enforced route.

Learner-link validity

The Live groups screen describes a chosen joining window; the written audit describes a 15-minute private link. This guide leaves the duration unspecified and retains the distinction between joining and playing time.

Versions and snapshot counts

The generated schema-11 reference and AI checkpoint report 39 actions, 28 triggers and 18 conditions; older chapters retain smaller counts. Game, room, asset and learner totals in screens are project snapshots, not platform limits.

Qualification boundary: the supplied August 14 record reports local acceptance, including deterministic AI tests and emulated mobile journeys. It does not certify a live AI provider, physical devices, production scale, security or accessibility.

Sources: SCREENS 14, 15, 19, 24; [S1] Playtest, publication and groups; [S2] Libraries; [S7] Limits; [S8] Verification.

Every supplied view has a place in this guide.

All 24 screenshots are included. Page numbers below link directly to their main treatment.

VIEW / ORIGINAL FILENAME	PAGE	VIEW / ORIGINAL FILENAME	PAGE
01 Dashboard dashboard.png	04	13 Issues & Readiness issues_and_readiness.png	22
02 Layout editor layout_editor.png	05	14 Playtest Center play_test_center.png	23
03 Asset library asset_lib.png	07	15 Release Center release_center.png	24
04 Items items_lib.png	08	16 Backups & files backups.png	25
05 Game parts game_parts_lib.png	09	17 Built-in Updates server_updates.png	27
06 Players players_lib.png	10	18 Teaching setup teaching_setup.png	28
07 Missions game_missions.png	11	19 Live groups live_groups.png	30
08 Coding Tasks coding_tasks.png	12	20 Learners & Insights player_insights_and_stats.png	31
09 AI task wizard ai_tasks.png	14	21 Languages language.png	32
10 Actions game_actions.png	17	22 Organization organization_conf.png	33
11 Action flows actions_flow.png	18	23 Arc Academy arc_school_system_connection.png	34
12 Game settings game_settings.png	20	24 Developer docs developer_docs.png	35

Source asset: arc_studio_img.zip. Filenames are preserved below so the illustrated pages can be traced to the original capture.

The reference behind the visual guide.

Product descriptions are grounded in the attached material; the guide does not add an independent implementation audit.

S1

STUDIO_BEHAVIOR.md

Creator-page behavior, save model, editing, testing and release workflows.

S2

ARC_DEVELOPER_HANDBOOK.md

Schema 11, library reference, task modes, configuration and embedded documentation.

S3

CURRENT_STATE.md

Capability overview, product surfaces and creator/learner workflows.

S4

GAME_RUNTIME.md

Gameplay, Python feedback, AI guide, progress, controls and world-task behavior.

S5

GAME_IMPORT_GUIDE.md

Package 2.1, media, import analysis, domain structure and portability.

S6

AI_WORLD_TASKS_RUNBOOK.md

Owner provider configuration, bounded AI operation and usage privacy.

S7

KNOWN_LIMITATIONS.md

Behavioral boundaries, trust limits and unqualified capabilities.

S8

VERIFICATION.md

Recorded local acceptance and explicit qualification limits.

S9

ARCHITECTURE_AND_DATA.md

Assignment, attempt, release, analytics and session semantics.

S10

FILE_INVENTORY.md

Page ownership and supporting local operator surfaces.

File identity. References use the documents' own titles. For repeated attachments, the newly supplied (1) copies are the working sources. README provides the documentation map. The Hawkins implementation brief is treated as game-design context, not proof that every requested step shipped.

Visual identity. ARC's logo and navy, orange and light-page design continue the supplied Hawkins demo guide. Screenshots are real captures; detail crops and numbered markers were added only for explanation.

Edition: September 2026. Written checkpoint: 14 August 2026, with retained older narrative. Screenshot dates vary. No credentials are included.

DESIGN THE WORLD.
MAKE LEARNING HAPPEN.

ARC GAMING STUDIO