DOORS Project Manual

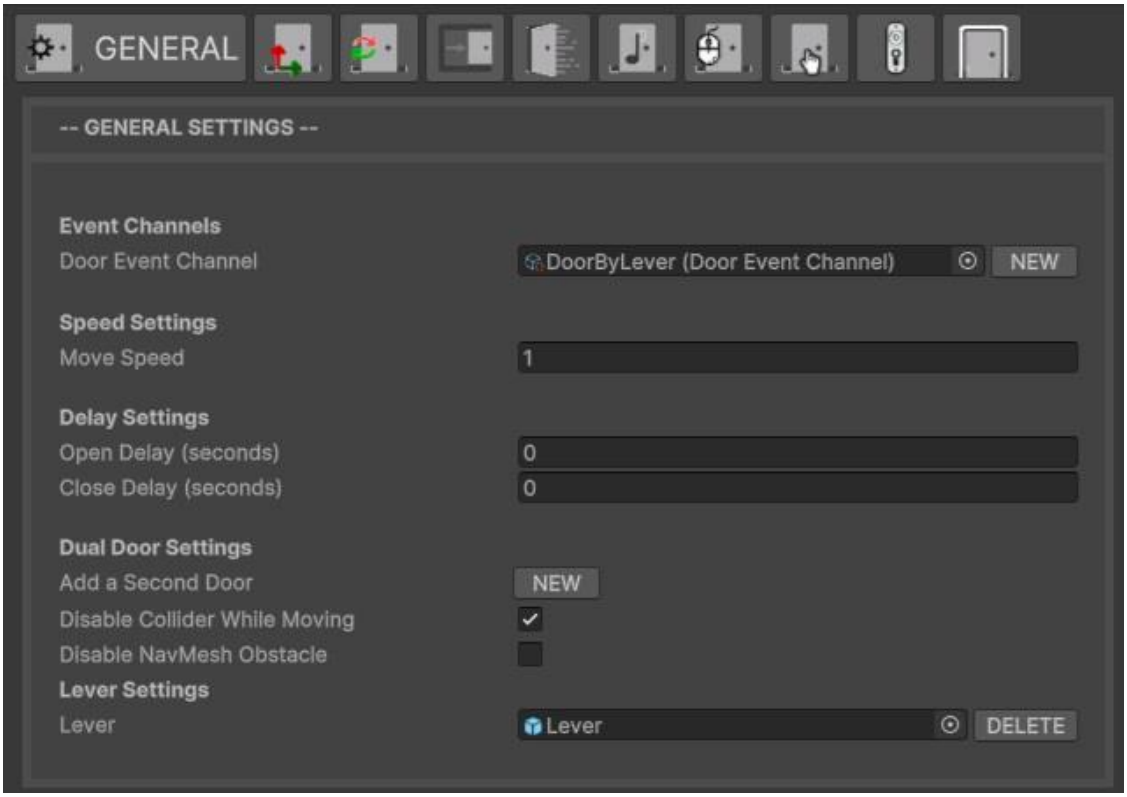
This manual will guide you through each section of the DOORS Project, explaining the settings, UI components, and workflows necessary to configure doors efficiently in your project.

The **DOORS Project** provides a highly customizable and interactive way to implement door mechanics in Unity. This system supports various functionalities, including pivot adjustments, rotational and sliding doors, animations, sounds, interactivity, and more. The system is integrated with Unity's editor tools, allowing precise configuration using intuitive UI elements and scene handles.

This guide comprehensively explains the configuration and usage of the DOORS Project in Unity, Each section provides descriptions of UI components, key functionalities, and inspector workflows to help you utilize the system efficiently.

CONTENTS

I. GENERAL SETTINGS



General Settings Section

Overview:

The General Settings section in the Door System Editor allows you to configure key properties related to door functionality, including speed, delays, event channels, dual door behavior, and lever control. This section ensures your door operates as intended with adjustable parameters and proper event handling.

General Settings UI Components:

- **Speed Settings**: Adjust the speed at which the door opens and closes.
- **Delay Settings**: Set delay times for opening and closing actions.
- **Event Channels**: Assign or create a new *DoorEventChannel* to handle door interactions.
- **Dual Door Settings**: Configure secondary doors and manage their assignments.
- **Door Enablers**: Options to disable colliders and NavMesh obstacles while the door moves.
- **Lever Settings**: Enable and assign a lever for door control.

Key Functionalities:

1. **Speed Control**:

   Set the door's movement speed using the Move Speed field.

2. **Delay Configuration**:

   Open Delay: Time before the door opens after interaction.

   Close Delay: Time before the door closes after interaction.

3. **Event Channels Management**:

   Assign an existing DoorEventChannel or create a new one using the NEW button.

   DoorEventChannel is essential for raising events when the door opens or closes.

4. **Dual Door Management**:

   Enable or disable dual doors.

   Add a secondary door by assigning a prefab or a scene object.

   Remove the secondary door using the DELETE button.

5. **Door Enablers**:

   Disable Collider While Moving: Disables the door's collider during movement.

   Disable NavMesh Obstacle: Disables the NavMesh obstacle during door movement.

6. **Lever Integration**:

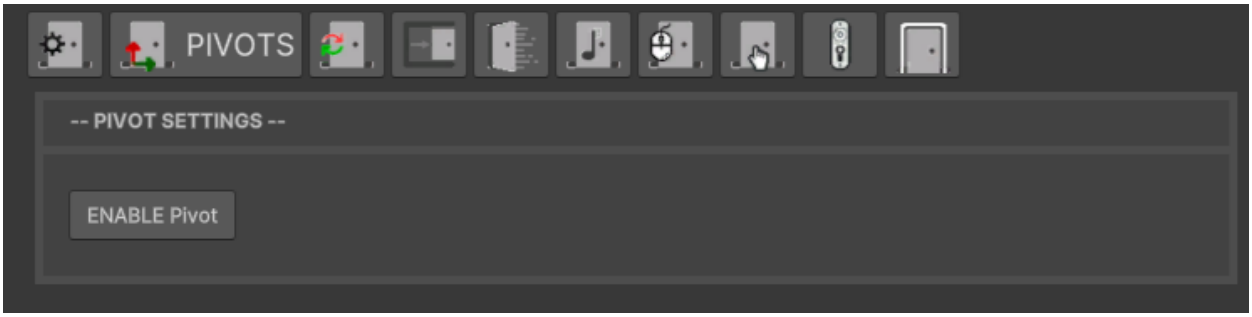
   Add a lever for door control if IsLeverControlled is enabled.

   Assign a lever prefab or delete an existing lever using the UI buttons.

Inspector Workflow:

- Open the Door component in the Unity Inspector.
- Navigate to the General tab in the Door Editor.
- Adjust settings such as speed, delays, and event channels as needed.
- Add secondary doors or levers if required.
- Use the provided buttons to create new components directly from the inspector.

II. PIVOT SETTINGS



Hinge Settings Section screenshot

Overview:

The Hinge Settings section lets you configure pivot and hinge behaviors, ensuring precise control over the door's rotation.

Hinge Settings UI Components:

- **Pivot Assignment**: Enable, assign, or clear the door's pivot object.
- **Pivot Name**: Set or modify the pivot name.
- **Pivot Transform**: Assign or adjust the pivot transform.
- **Pivot Adjustment Mode**: Toggle manual adjustment with scene handles.
- **Grid Snapping**: Enable snapping and set grid size.
- **Bounds Limiting**: Restrict pivot movement within the door's collider bounds.
- **Preset Buttons**: Auto-position the pivot at various points (e.g., center, corners).

Key Functionalities:

1. **Pivot Management**: Assign the door's own GameObject as the pivot.
2. **Error Handling**: Automatically create a parent for the door if none exists.

Error handling inspector screenshot

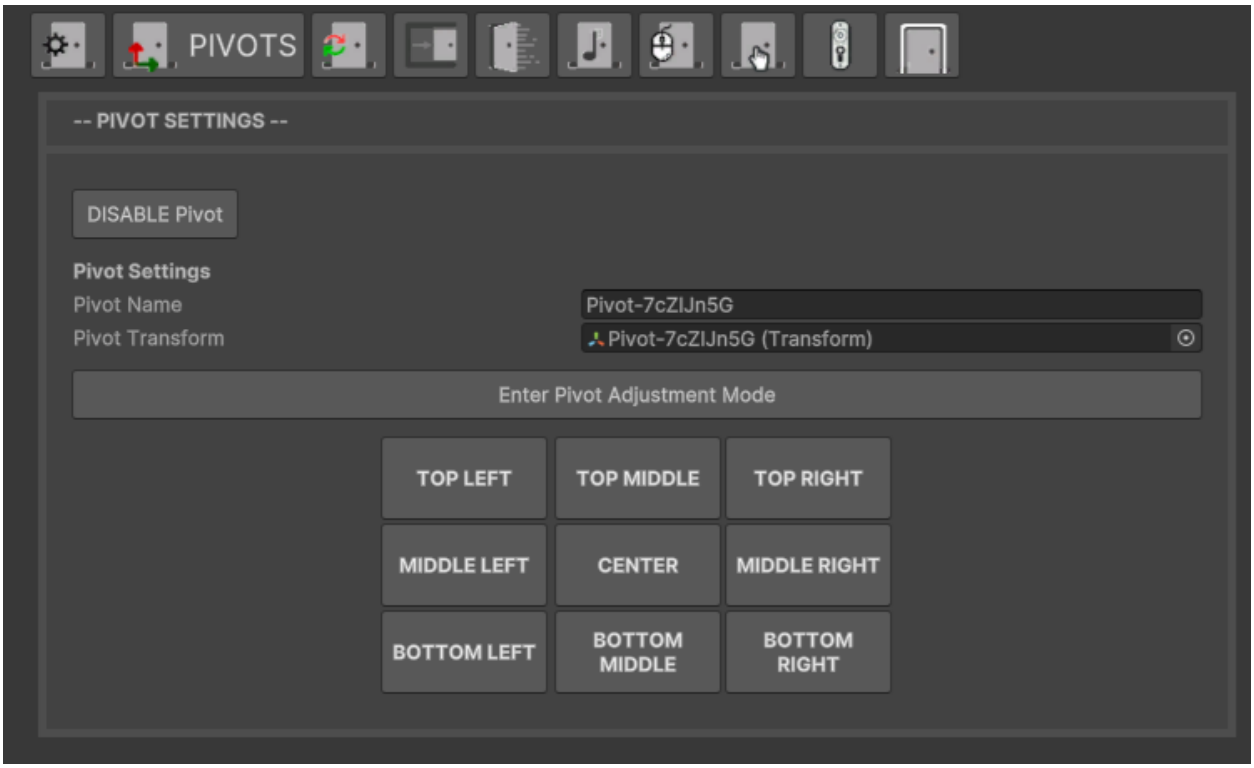
3. **Grid Snapping**: Adjust pivot position precisely using a configurable grid.
4. **Bounds Restriction**: Prevent pivot from moving outside the collider.
5. **Preset Positions**: Automatically place the pivot using preset options.

Inspector Workflow:

- Go to the Hinge tab.
- Assign the pivot or enable Pivot Adjustment Mode.
- Use the grid and bounds settings for precise alignment.
- Apply preset positions for quick setup.

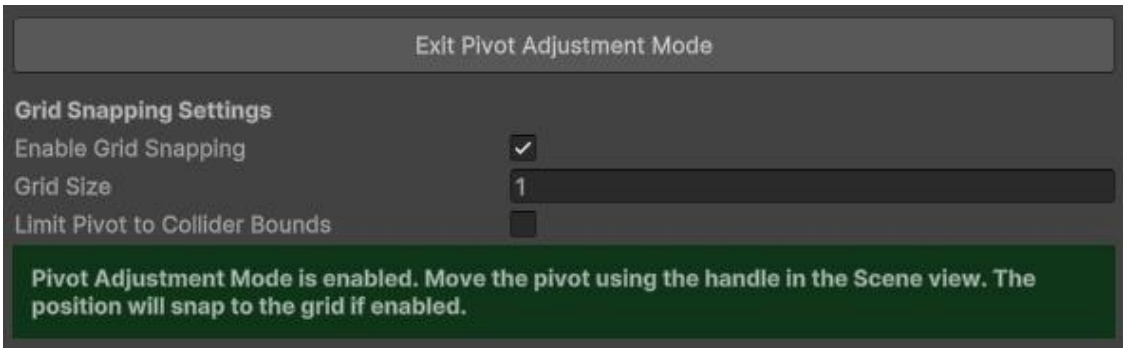
II. 1. MAIN PIVOT SETTINGS:

This part focuses on assigning the pivot object, naming it, and using preset buttons for automatic positioning. The preset buttons allow you to place the pivot at specific points like center, corners, and edges quickly.



Hinge settings image

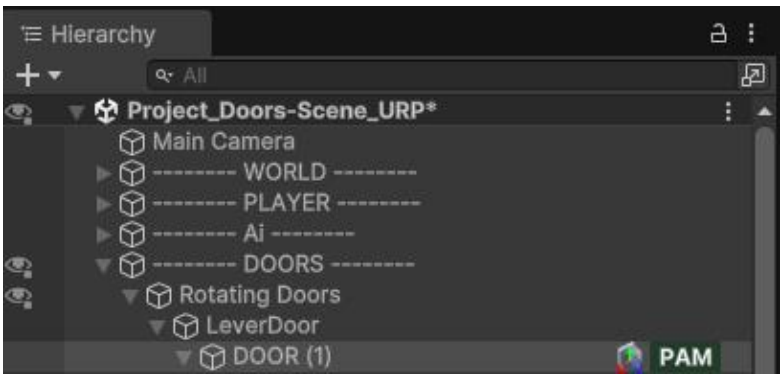
II. 2. PIVOT ADJUSTMENT MODE:



Pivot adjustment mode image

This mode enables manual pivot adjustment using scene handles. It supports grid snapping, bounds limiting, and dynamic adjustments. Toggle this mode to freely move and refine the pivot position with optional grid constraints and boundary checks.

II. 3.PIVOT ADJUSTMENT MODE IN HIERARCHY:

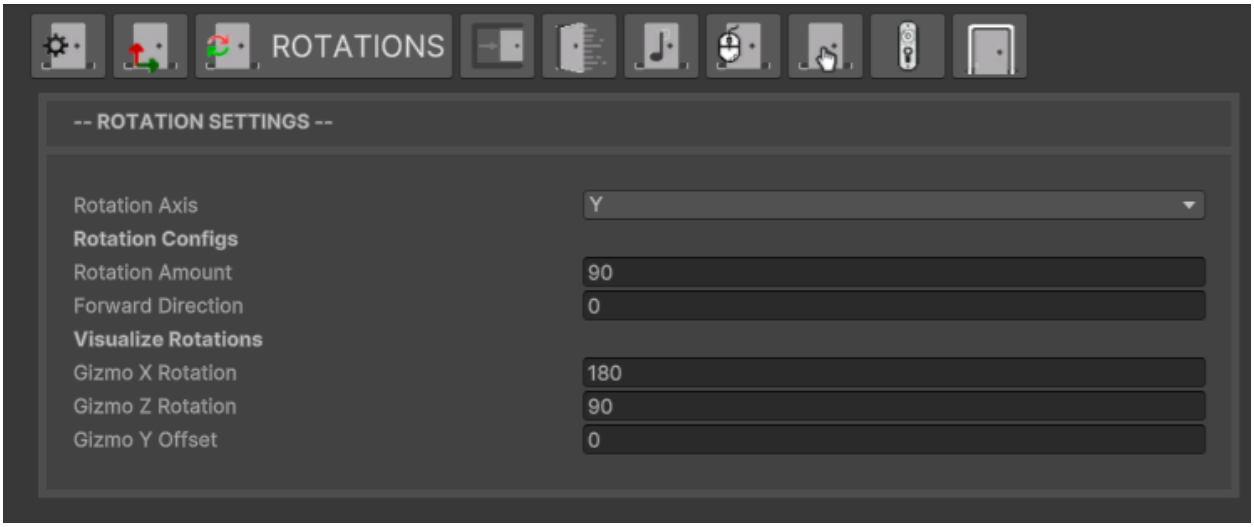


Hierarchy integration

It highlights the active pivot adjustment mode directly in the Unity Hierarchy with an icon and a [1]**PAM** label. This ensures you can see which doors have pivot adjustment enabled without opening the inspector.

[1] *Pivot Adjustment Mode*

III. ROTATION SETTINGS



Rotations Settings Screenshot

Overview:

The **Rotation** tab defines how a hinged door swings around its pivot. Here you choose the hinge axis, the opening angle, the direction logic (so the door opens away from the player), and fine-tune a gizmo that previews the arc in the Scene view.

Rotation Settings UI controls

Control	Purpose
Rotation Axis *(X / Y / Z)*	Axis around which the door rotates.
Rotation Amount (°)	How far the door turns from *Closed* → *Open* (e.g., 90 °).
Forward Direction	Value -1 … 1 that decides whether the door uses the positive or negative side of the chosen axis so it opens away from the player.
Gizmo X Rotation	Spins the Scene-view gizmo around X for a clearer preview.
Gizmo Z Rotation	Spins the gizmo around Z.
Gizmo Y Offset	Raises / lowers the gizmo arc when it's hidden by floor or walls.

> **Note:** When **Sliding Mode** is enabled in the *Sliding* tab these controls are automatically disabled, preventing accidental edits to a sliding door.

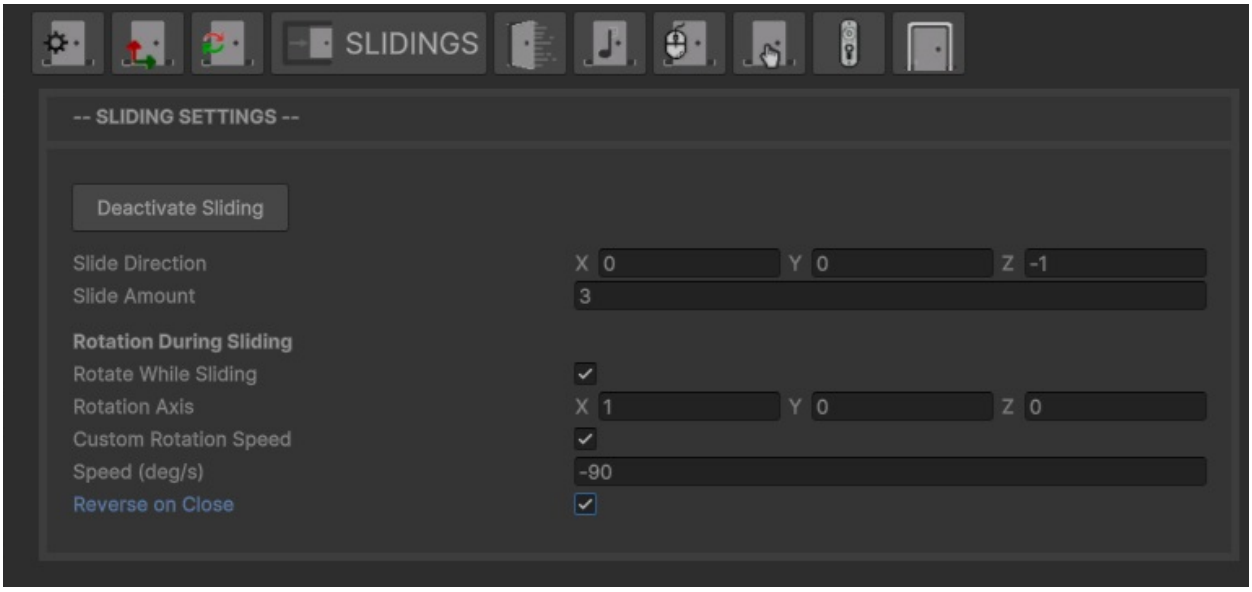
Key Functionalities:

- **Axis control:** Pick the correct hinge axis for the model.
- **Angle control:** Set the exact swing angle with **Rotation Amount**.
- **Automatic direction: Forward Direction** logic flips the sign of the rotation so the door opens away from whoever interacts.
- **Gizmo customisation:** Use X, Z rotations and Y offset to make the cyan preview arc match the real swing.

Inspector Workflow:

- Select the door and open the **Rotation** tab.
- Choose **Rotation Axis**, set **Rotation Amount**, and adjust **Forward Direction** if the door swings the wrong way.
- Fine-tune **Gizmo X/Z** and **Y Offset** until the arc aligns with the door's path.
- Enter Play mode, approach the door from both sides, and confirm it always swings away. If not, nudge **Forward Direction**.

IV. SLIDING SETTINGS



Sliding settings screenshot

Overview:

The **Sliding** tab lets you turn a door into a panel that glides along a straight line. You pick the slide vector and distance, choose whether it should spin while moving, control the spin speed, and decide if that spin should reverse on close.

Sliding Settings UI Components:

Control	Purpose
Activate Sliding	Master toggle — when **off** the door behaves as a rotating door and the rest of these controls are disabled.
Slide Direction *(Vector3)*	Direction the door moves in world-space (e.g., (1 0 0) for right).
Slide Amount *(m)*	Distance travelled from *Closed* → *Open*.
Rotate While Sliding	Adds a continuous spin during the slide.
Rotation Axis *(Vector3)*	Axis of that spin (shown only when rotation is enabled).
Custom Rotation Speed	Choose between automatic speed (360° over the full slide) or a manual value.
Speed (deg/s)	Manual spin speed shown when *Custom Rotation Speed* is on.
Invert Rotation on Close	Reverses the spin direction when the door is closing.

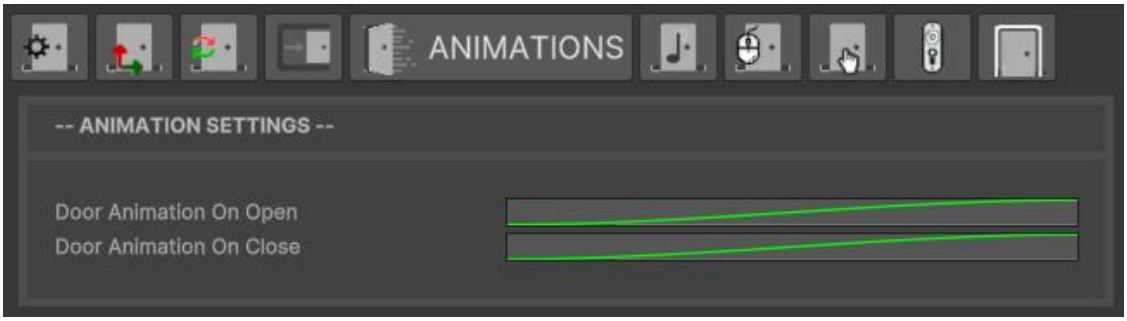
Key Functionalities:

- **Linear motion:** Define any slide vector and distance for horizontal, vertical, or diagonal panels.
- **Optional spin:** Enable **Rotate While Sliding** for more dynamic doors.
- **Auto-vs-manual speed:** Leave *Custom Rotation Speed* off to let the script match a full 360° turn to the slide duration—or specify an exact degrees-per-second value when precision matters. Door
- **Directional inversion:** Use *Invert Rotation on Close* to mirror the spin as the door returns.

Inspector Workflow:

1. Open the **Sliding** tab and press **Activate Sliding**.
2. Set **Slide Direction** and **Slide Amount** so the cyan gizmo arrow matches the desired path.
3. (Optional) Enable **Rotate While Sliding**, choose a **Rotation Axis**, and decide between automatic or **Custom Rotation Speed**.
4. Toggle **Invert Rotation on Close** if you want the closing animation to spin the opposite way.
5. Enter Play mode and test from both sides; tweak values until movement and spin feel right.

V. ANIMATION SETTINGS



Animation settings screenshot

Overview:

The Animation Settings section allows you to customize how the door opens and closes using animation curves.

Animation Settings UI Components:

- **Door Animation On Open**: Define the animation curve for the door opening.
- **Door Animation On Close**: Set the animation curve for the door closing.

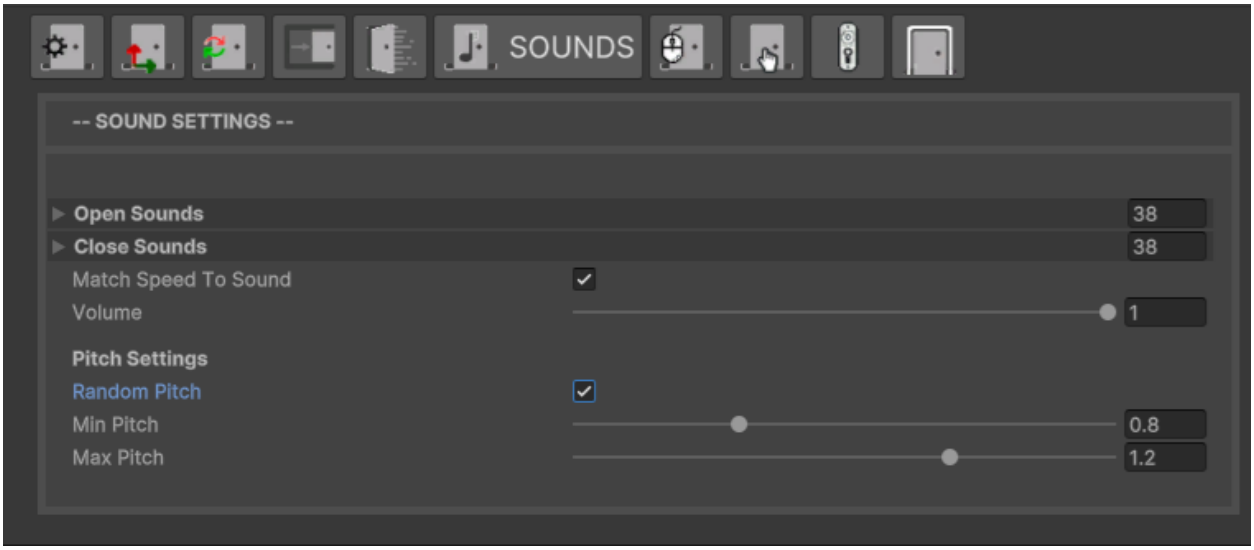
Key Functionalities:

1. **Smooth Animations**: Use custom curves to create smooth and realistic door movements.
2. **Curve-Based Control**: Precisely control how the door opens and closes over time.

Inspector Workflow:

- Open the Animations tab in the Door Editor.
- Assign or create animation curves for door opening and closing.
- Preview animations directly in the Scene view by adjusting curves in real-time, playing the door animation with Unity's playback controls, and observing changes instantly to refine movements.

VI. SOUND SETTINGS



Sound settings screenshot

Overview:

The **Sounds** tab adds immersive audio feedback to every door interaction. You assign one-or-many clips for opening and closing, let the system pick a random one each time, match the animation speed to the clip's length, and fine-tune loudness and pitch (fixed or random) for extra variation.

Sound Settings UI Controls:

Control	Purpose
Open Sounds *(AudioClip list)*	Clips randomly chosen when the door opens. If empty, no sound plays.
Close Sounds *(AudioClip list)*	Clips randomly chosen when the door closes.
Match Speed to Sound	When on, the door's open/close animation duration is set to the length of the chosen clip, so motion and audio end together.
Volume *(0 – 1)*	Master volume for every playback; applied directly to the AudioSource.
Random Pitch	Single fixed pitch when *Random Pitch* is **off**.
Pitch	Raises / lowers the gizmo arc when it's hidden by floor or walls.
Min / Max Pitch	Range used to pick a new random pitch per playback when *Random Pitch* is **on**.

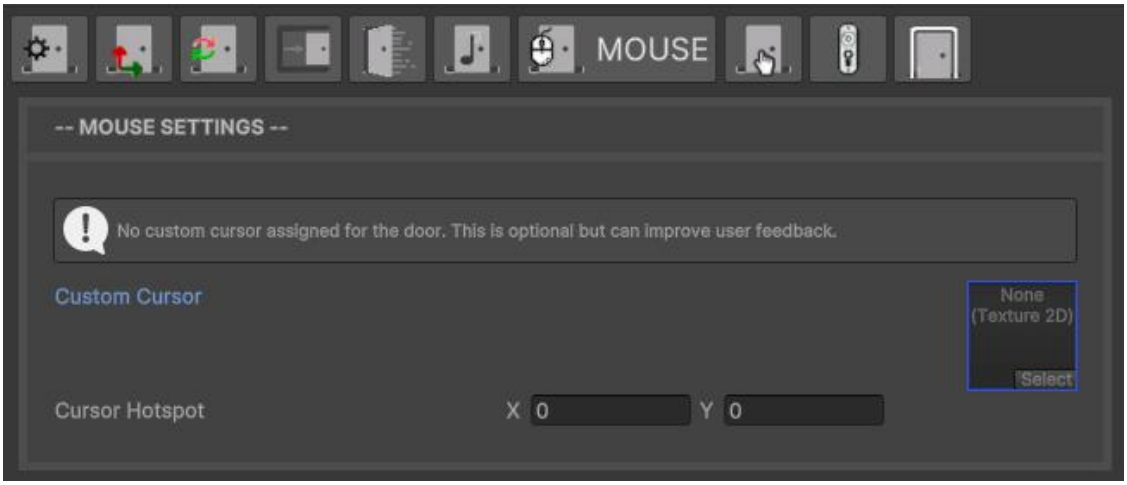
Key Functionalities:

- **Clip management & variety:** Add as many clips as you like to **Open Sounds** and **Close Sounds**; one is chosen with Random.Range on every interaction for natural variation. Door
- **Speed matching:** If **Match Speed to Sound** is enabled, playback length is returned by PlayRandomSound, and the animation's duration is set to that value, so movement ends exactly with the audio tail. Door
- **Volume control:** The slider feeds straight into audioSource.volume, giving scene-wide consistency while still allowing mixer-group overrides. Door
- **Pitch variation:** Enable **Random Pitch** to pick a fresh value between **Min** and **Max** each time, or keep it off and use a fixed **Pitch** for stylised effects (e.g., metallic vs. wooden doors).

Inspector Workflow:

1. Open the **Sounds** tab.
2. Drag clips into **Open Sounds** and/or **Close Sounds**.
3. Check **Match Speed to Sound** if you want the door to finish moving exactly when the clip ends.
4. Adjust **Volume** to blend with your environment.
5. (Optional) Turn on **Random Pitch** and set **Min / Max** (or leave it off and set a single **Pitch**).
6. Enter Play mode and trigger the door several times to verify variation, volume balance, and timing.

VII. MOUSE SETTINGS



Mouse settings screenshot

Overview:

The Mouse Settings section configures mouse-related interactions with the door, including custom cursors and interaction hotspots.

Mouse Settings UI Components:

- **Custom Cursor**: Assign a texture to be displayed when the mouse hovers over the door.
- **Cursor Hotspot**: Set the point within the cursor texture that acts as the interaction point.

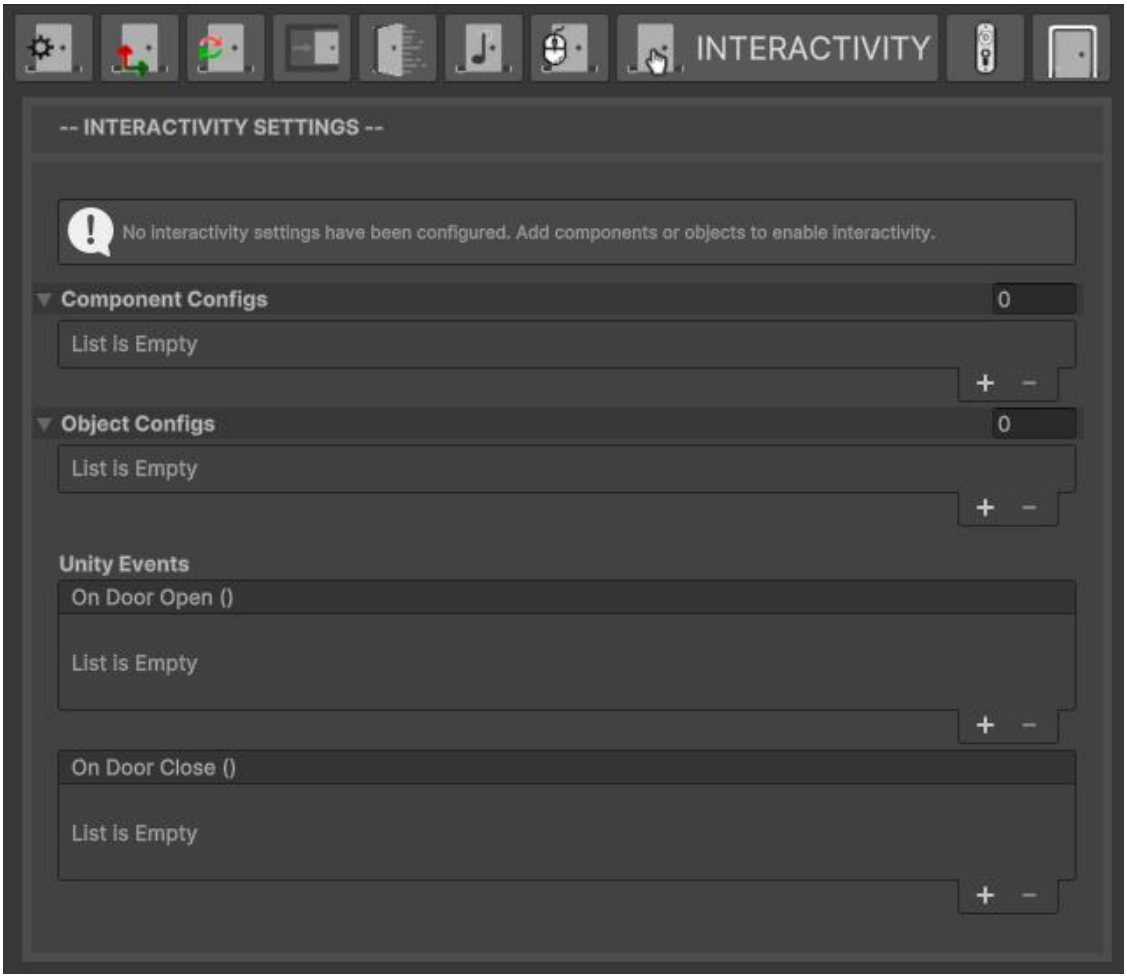
Key Functionalities:

1. **Custom Cursors**: Change the cursor icon when hovering over the door for better visual feedback.
2. **Hotspot Control**: Precisely adjust the interaction point within the cursor texture.

Inspector Workflow:

- Open the Mouse tab in the Door Editor.
- Assign a custom cursor texture and define its hotspot.
- Preview the cursor interaction in the Scene view by hovering the mouse cursor over the assigned door object.

VIII. INTERACTIVITY SETTINGS



Interactivity settings screenshot

Overview:

The Interactivity Settings section manages how the door interacts with other components and objects, including enabling or disabling elements when the door opens or closes.

Interactivity Settings UI Components:

- **Component Configs**: List of components to enable/disable on door open/close.
- **Object Configs**: List of GameObjects to enable/disable on door open/close.
- **Events on Open/Close**: Unity Events triggered when the door opens or closes.

Key Functionalities:

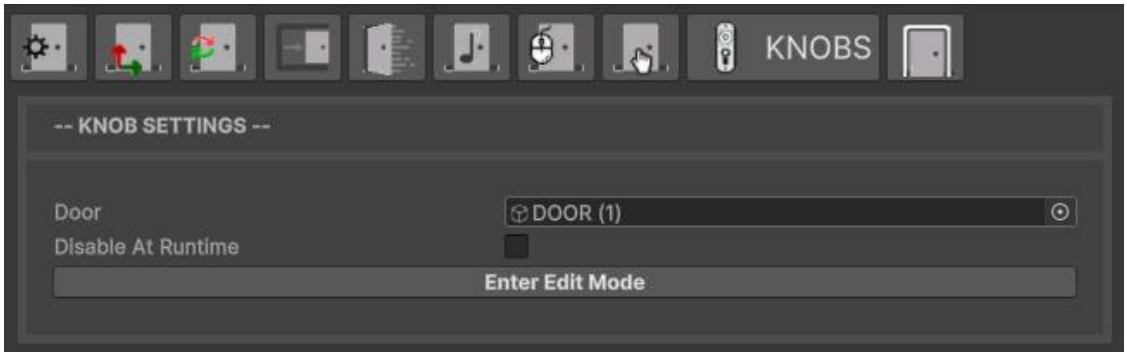
1. **Component Control**: Enable or disable specific components when the door changes state, with options to set delays for enabling or disabling each component.
2. **Object Control**: Activate or deactivate GameObjects during door interactions, with configurable delays for each object.
3. **Event Management**: Use Unity Events to trigger custom behaviors when the door opens or closes.

Inspector Workflow:

- Open the Interactivity tab in the Door Editor.
- Add components and objects to the interactivity lists.
- Assign Unity Events to be triggered during door operations.

This section ensures that door interactions are highly customizable, allowing dynamic changes to components and objects within your scene during door operations.

IX. KNOB SETTINGS

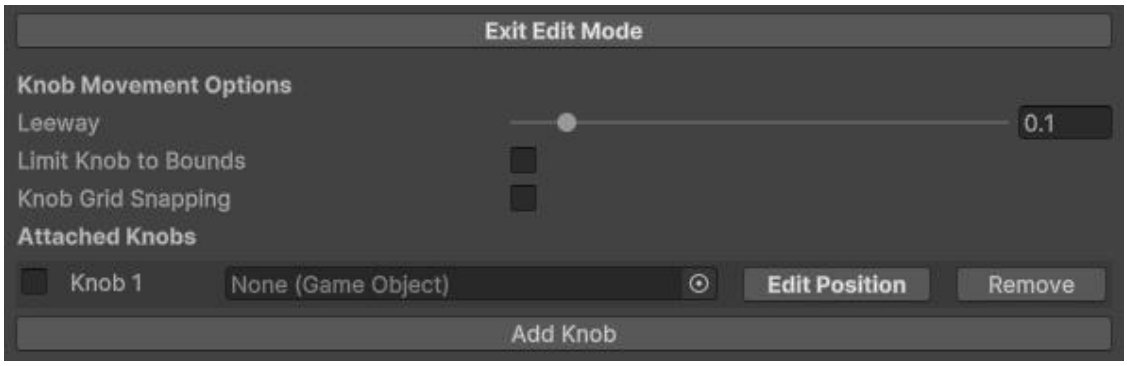


Knob settings main knob settings screenshot

IX. 1. MAIN KNOB SETTINGS:

This section allows you to assign, manage, and configure knobs attached to the door, including adding, removing, and locking knobs.

IX. 2. KNOB EDIT MODE:



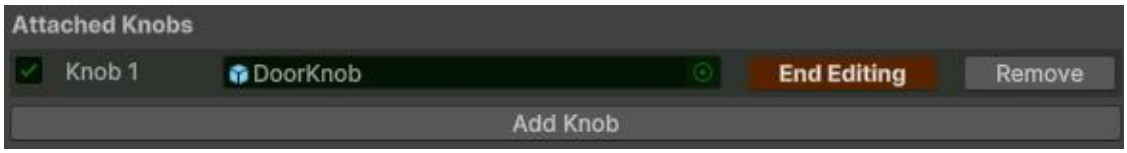
Knob edit mode screenshot

This mode enables manual knob adjustment using scene handles. The options available in this mode include:

- **Grid Snapping**: Enable snapping knobs to a defined grid for precise alignment.
- **Bounds Limiting**: Restrict knob movement within door collider bounds.
- **Knob Locking**: Lock knobs to prevent accidental movement.
- **Position Handles**: Use Unity handles to manually move, rotate, and position knobs.
- **Leeway Adjustment**: Adjust additional movement allowance beyond bounds.

- **Knob Grid Size**: Set the size of the grid cells for snapping. These options provide full control when editing knobs, ensuring accuracy and ease of use. This mode enables manual knob adjustment using scene handles, with options for grid snapping, bounds limiting, and precise position/rotation editing.

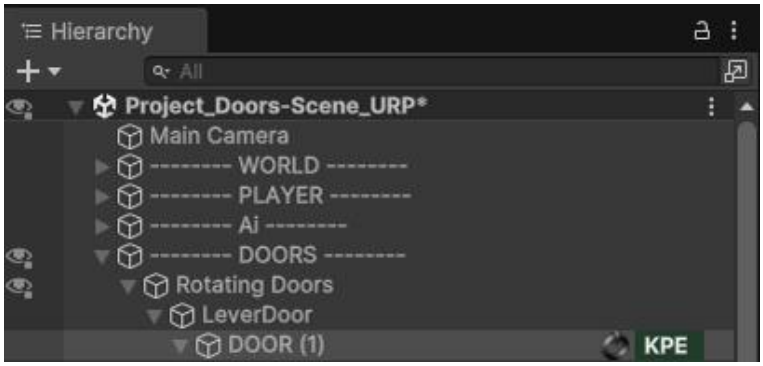
IX. 3. KNOB POSITION EDIT:



Knob position edit screenshot

In this mode, you can manually position knobs, lock/unlock their movement, and utilize grid snapping for precise adjustments.

IX. 4. KNOB EDIT MODE IN HIERARCHY:

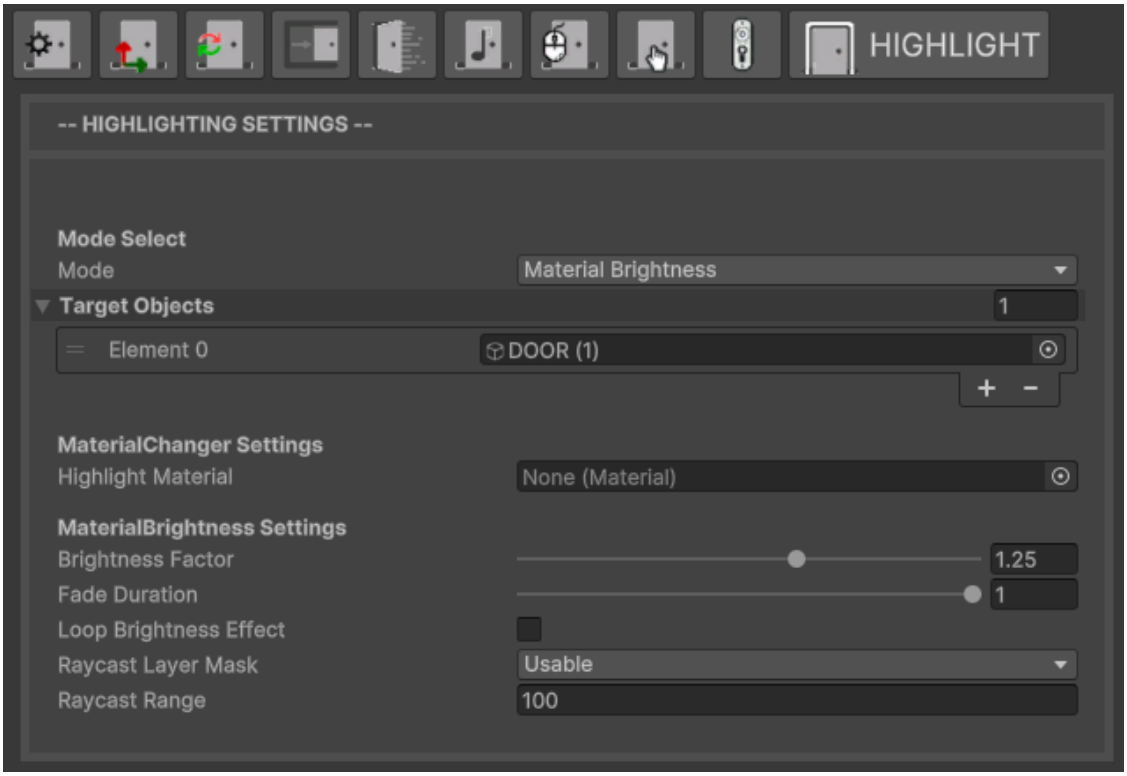


Knob edit mode hierarchy screenshot

Displays active knob edit mode in the Unity Hierarchy with a [2]**KPE** label, clearly showing which doors have knobs in edit position.

[2] Knob Position Edit

X. HIGHLIGHTING SETTINGS



Highlighting settings screenshot

Overview:

This section manages the visual highlighting of doors when interacted with or hovered over.

Highlighting Settings UI Components:

- **Highlight Material**: Assign a material used to highlight the door.
- **Brightness Factor**: Adjust the brightness level of the highlight.
- **Fade Duration**: Set how quickly the highlight fades in and out.
- **Loop Brightness Effect**: Enable looping brightness effects.
- **Raycast Layer Mask**: Set which layers the highlighting raycast can interact with.
- **Raycast Range**: Define the maximum distance for raycast detection.

Key Functionalities:

1. **Highlight Material**: Apply a custom material for visual feedback.
2. **Brightness Control**: Adjust highlight brightness dynamically.
3. **Fade and Looping**: Add fade transitions and looping effects.
4. **Layer Masking**: Control which objects trigger the highlight.
5. **Raycast Distance**: Set how far the mouse can interact with the door.

Inspector Workflow:

- Open the Highlighting tab in the Door Editor.
- Assign highlight materials and adjust brightness, fade, and raycast settings.

This section ensures that doors are easily identifiable during interaction, with customizable visual highlights.

XI. DOOR SETUP WIZARD

The wizard is an **Editor-only** tool that streamlines first-time configuration of players and doors.

How to open: Tools ▸ DOORS Project ▸ Setup Wizard

Workflow

1. **Welcome** – Press **Proceed**.
2. **Player / Door choice** – Pick **PLAYER SETUP** or **DOOR SETUP**.
3. **Character Setup** – Drag a player model / prefab into the drop-zone, then click **SETUP PLAYER**.
4. **Door Setup** – Drag a door model / prefab and click **SETUP DOOR**.
   The wizard automatically:
   - Spawns the prefab, unpacks it if needed.
   - Adds colliders, Door, DoorEventChannel, ProximityStopper, NavMeshObstacle, AudioSource, and LayeredNavMeshObstacle. DoorReplacer
   - Sets layer **Usable** and default tags.
5. **Finish** – A reminder window prompts you to create a uniquely-named Door Event Channel.

Tips & Troubleshooting

- Re-run the wizard any time; existing objects are detected and won't be duplicated.
- If a prefab drops in as a **scene object**, the wizard offers to delete the original to keep scenes tidy.
- All operations are Undo-aware; press **Ctrl/Cmd + Z** to revert.

XII. PLAYER ACTIONS

The **Player Actions** component glues user-input, UI prompts, and door-logic together. It shows a floating **"Open / Close"** label, halts the player while they decide, and raises an InteractionEventChannel request (used by the door to fire its own DoorEventChannel) and allows to rebind the keys

Inspector field	Default	Purpose
interactionEventChannel	*PlayerInteractionEventChannel* single-ton	Publishes RaiseInteractRequest() when the user commits to a door action
usableTextTag	"usableText"	Tag used to locate a disabled TMP label that floats near the door
Use Layers	*LayerMask* → layer **Usable**	Hit-test filter for mouse rays & proximity checks
interactActionReference	*Player/Use* action from **DoorInputs**	Key / button that triggers OnUse(); auto-filled by OnValidate() if empty
OpenText / CloseText	"Open" / "Close"	Localisable strings swapped into the TMP label
x/y/z Offset	0.1 / 1.4 / 0.13 m	Fine-tune label placement against hit-point & normal
ActivationDistance	1.5 m	Max reach for both mouse and key interactions
enableLogging	False	Flood-gate for verbose conditional logs

Update-loop behaviour – executed every frame when the label exists:

1. **HandleMouseInteraction()**
   Continuous ray from the cursor onto **Use Layers**. If the hit object is a Door and the player is ≤ ActivationDistance:
   - shows the label (OpenText/CloseText),
   - freezes movement via ClickAgentController.StopAgentMovement(),
   - hides the label + resumes on exit. PlayerActions
2. **HandleKeyInteraction()**
   Key/button defined by **interactActionReference** triggers a sphere-overlap around the player; the first door in range is toggled (open ↔ close) through its DoorEventChannel. PlayerActions
3. **Optional HandleProximityInteraction()**
   Straight-ahead raycast, handy for game-pad workflows.

Event path: Player input ➜ PlayerActions ➜ InteractionEventChannel ➜ Door (via its DoorEventChannel) – movement is paused/restored automatically.

INPUT ACTION BINDINGS – INSTANT IN-EDITOR KEY REMAPPER

The “Input Action Bindings” fold-out that appears under **Player Actions** is **not** a runtime UI; it’s a **custom Inspector tool** that lets you change the **Interact** key (or any game-pad button) right inside the Unity Editor and see the result immediately—no Play-mode required.

What happens
The custom Inspector (PlayerActionsEditor.OnInspectorGUI) lists every binding of the **Interact** action except mouse/pointer devices to keep the list clean.
Rebind button → calls StartRebinding(action, index). The action is **disabled**, Unity’s *interactive rebinding* window opens, and the next key/button you press becomes the new binding.
When you finish, SaveBindingOverride(action) writes the override path to **PlayerPrefs**, so the change persists across editor sessions and into builds.
Reset Bindings to Defaults wipes the saved overrides (ResetBindings(action)), clears them from the action, and refreshes the Inspector list.
At runtime PlayerActions.LoadBindingOverrides() reads those PlayerPrefs keys on Awake() and applies the same overrides, so players get the exact bindings you set in the editor.

How to use

1. Select the GameObject that owns **Player Actions**.
2. Expand **Input Action Bindings**.
3. Click **Rebind** next to the binding you want to change → press the new key or controller button; the label updates instantly and a log action is reported to the console confirming the action.
4. Test the new key in Play-mode—PlayerActions will already be using it.
5. If you ever need to roll back, hit **Reset Bindings to Defaults**.

Because rebinding happens entirely in the Inspector, designers can iterate on control schemes without entering Play-mode, and the chosen layout is included automatically when you build the game.

XIII. DOOR LEVER

A scene-placed switch that lets players open or close a linked door by clicking the lever itself.

- **What it does:** Detects the player within a set radius, shows a “Use” prompt, waits for a click on the lever mesh, then toggles the door through its DoorEventChannel.
- **Advantages:** Offers a clear, physical interaction and works even if the door is otherwise locked, because it sends a *force-toggle* event.
- **Polished feel:** Plays a smooth rotation animation on the lever handle and—when you add multiple audio clips—automatically matches the animation speed to the length of the randomly-chosen sound, so motion and audio stay perfectly in sync.

Inspector Breakdown

Field	Purpose
Controlled Door	Door to command (auto-finds parent if empty).
Door Event Channel	Usually the door’s own channel.
Activation Distance	Player proximity required.
Object To Rotate	Transform that visually swings (defaults to self).
Lever Rotation Angle / Speed	Range and rate of the swing.
Match Rotation Speed To Sound	Auto-match animation length to the chosen audio clip.
Lever Sounds	Play random clips on toggle.

Runtime Flow

1. When the player enters range the script shows a TMP prompt ("Open \"Click\"" / "Close \"Click\"").
2. A raycast from the camera detects left-clicks on the lever mesh.
3. ToggleLever() animates the rotation coroutine, then raises DoorEventChannel.Open/Close (with force = true to bypass lever-locking).

XIV. CLICK AGENT CONTROLLER

The **Click Agent Controller** serves as a single on/off valve for the player's locomotion. Whether your character moves via a NavMeshAgent or a CharacterController, this script gives every other system—Proximity Stopper, Player Actions, cut-scenes, dialogue windows, and more—a single, unified API to freeze or release the avatar. By funnelling all "stop" and "resume" calls through one place, you eliminate the tangle of scripts that would otherwise have to juggle two different movement components.

Beyond the convenience of one-line control, Click Agent Controller keeps the whole game in sync: whenever movement is paused or resumed it raises MovementControlEventChannel events. HUD icons can switch states, footstep audio can mute or unmute, and animation layers can blend smoothly—all in real time and without extra polling. In short, the controller decouples your codebase and broadcasts clear, centralised signals so locomotion, UI, and audio stay perfectly coordinated.

Public API (runtime)

Method	What it does
StopAgentMovement()	• Sets NavMeshAgent.isStopped = true *and* agent.ResetPath() (kill residual velocity). • Flags CharacterController movement as blocked. • Broadcasts movementControlChannel.RaiseStopMovement() so listeners can fade animations, mute footsteps, etc. ClickAgentController
ResumeAgentMovement()	Clears both locks, restarts the agent if it's on a navmesh, then emits RaiseResumeMovement(). ClickAgentController
IsMovementAllowed()	Returns **false** while a CharacterController is frozen—handy inside your own CC-driven movement script. ClickAgentController
IsMovementAllowed()	Returns **false** while a CharacterController is frozen—handy inside your own CC-driven movement script. ClickAgentController
(Optional) HandleProximity(bool shouldStop)	Lightweight helper some scripts call instead of remembering which method to invoke.

Key advantages

1. **One-liner control** – Collapses two movement APIs into one.
2. **Event driven** – External systems can subscribe to **MovementControlEventChannel** instead of polling the player's state.
3. **Fail-safe** – Automatically chooses the relevant component at runtime; works even if you switch the player prefab from NavMesh to CharacterController later.
4. **Undo-friendly** – Resets paths and flags only, so undoing or re-enabling movement snaps the agent back to the expected behaviour without hiccups.

XV. PROXIMITY STOPPER

Locks the player in place when they get close enough to a small interactable (lever, keypad, door handle, etc.), so they don't walk straight past it or drift while clicking. It also broadcasts **enter/exit proximity events** so UI widgets, sound cues, or AI logic can respond.

How it works (runtime loop)

1. **Distance check each frame** – measures the gap between the object and the player.
   If distance ≤ stopRange (default ≈ 1.44 m) and the player has just arrived:
   - raises proximityEventChannel.RaiseEnterProximity(transform.position) and calls StopPlayerMovement()ProximityStopper
   - stores the player's current position to keep them anchored.
     When the player walks out of range the inverse happens: RaiseExitProximity → ResumePlayerMovement()ProximityStopper.
2. **Mouse-click guard** – while the cursor is held down:
   - casts a ray from the camera; if that ray is **on this object**, movement stays locked, the player agent is **Warped** back to the stored position, and a flag (playerHasClickedOnObject) prevents escape until the button is released.
   - on release, a small releaseDelay (0.1 s by default) gives the player time to let go of the mouse before re-enabling movementProximityStopper.
3. **Central stop/resume** – the script never touches locomotion directly; instead it funnels everything through ClickAgentController.StopAgentMovement() / ResumeAgentMovement() so it works equally well with NavMeshAgent- or CharacterController-driven avatarsProximityStopper.

Key inspector fields

Field	Default	Effect
proximityEventChannel	Auto-linked in OnValidate()	Emits OnEnterProximity / OnExitProximity for HUD, SFX, etc.
stopRange	1.44 m	Radius of the invisible "freeze bubble".
releaseDelay	0.1 s	Grace period after mouse-up before movement resumes.

A green wire-sphere gizmo is drawn in Scene view so you can size the bubble precisely.

Advantages at a glance

- **Pixel-perfect interactions** – players can't overshoot tiny objects or slide away mid-click.
- **Event-driven** – other systems can simply subscribe to the ProximityEventChannel instead of polling distances.
- **Controller-agnostic** – relies on Click Agent Controller, so switching the player from path-finding to physics walk-mode requires no changes here.
- **False-movement prevention** – warps the agent back if they try to queue a new path while the mouse is still on the object, guaranteeing rock-solid position locking.

XVI. DOOR TRIGGER

Turns any collider into a proximity sensor that opens or closes its assigned Door as tagged objects enter or leave. It removes the need for manual "OnTriggerEnter" code in every scene and lets you tweak behaviour entirely from the Inspector.

Feature	What it gives you
Self-setup	On Reset, Awake, OnEnable, or OnValidate the component tries to **find the nearest door**, fetches that door's DoorEventChannel, and forces GetComponent<Collider>().isTrigger = true, so designers can just drag the prefab into a scene with zero extra clicks.
Tag filtering	allowedTags[] lets you restrict the trigger to players only ("Player") or include NPCs, vehicles, etc.
Auto-open / Auto-close	autoOpen and autoClose toggles. If autoClose is enabled the script optionally waits closeDelay seconds before shutting the door—handy for keeping it open while a party passes through.
Overlap counting	Internal allowedCount ensures the door doesn't slam shut while someone is still inside the trigger volume.
Lever compatibility	If the door is flagged IsLeverControlled, the trigger bypasses the event channel and calls Door.Open/Close directly so the lever's animation stays in sync.
Designer hooks	Two UnityEvent slots—**On Trigger Enter** and **On Trigger Exit**—fire whenever a valid object crosses the boundary, making it trivial to play sounds, flash lights, or run timeline cues without code.

Inspector breakdown

Field	Default	Meaning
Controlled Door	(auto-assigned)	Door this trigger commands.
Door Event Channel	(auto-assigned)	Communication bus used to request open/close.
Allowed Tags	{"Untagged"}	Only objects with these tags count.
Auto Open / Auto Close	true / true	Enable hands-free operation.
Close Delay	2 s	Wait time before auto-closing (only if allowedCount == 0).
On Trigger Enter / Exit	(empty)	Designer-defined UnityEvents.

Why use Door Trigger?

- **Drop-in ease** – One prefab dropped around a doorway gives you fully-functional automatic doors.
- **Flexible access control** – Anything from players to forklifts can trip the sensor by simply adding their tag to allowedTags.
- **No accidental shutdowns** – Overlap counting prevents premature closure in crowded areas.

- **Leverages the event system** – Uses the same DoorEventChannel as UI buttons and levers, keeping all door state changes funnelled through a single, debuggable point.
- **Designer-friendly** – Built-in UnityEvents mean level artists can chain lights, sounds, or camera shakes without touching code.

Attach **Door Trigger** to any collider volume, tweak a few toggles, and your doors will respond intelligently to approaching characters—no scripting required.