DETAIL GitHub Guide

V1

Compiled by Cat Bluemke for the MacKenzie Art Gallery's Digital Exhibitions Toolkit and Art Installation Launcher (DETAIL).

Table of Contents

Table of Contents

Acknowledgements

Introduction

This user guide is intended for use alongside the MacKenzie Art Gallery's DETAIL GitHub repository, found here. It will guide you through the resources available on the GitHub and indicate how they were used within our three pilot project digital exhibitions.

The guide is a simplified overview, and familiarity with Unity is suggested.

I Setup

1. Overview

This section covers the initial setup process for working with the DETAIL toolkit. We'll guide you through:

- Installing the Unity game engine and setting up Unity Hub
- Accessing the DETAIL repository
- Creating your first project

The setup process requires a computer with adequate specifications for Unity development. While the minimum requirements can be found on Unity's website, it's recommended you use:

- Windows 10/11 or macOS 10.15+

- 8GB RAM (16GB recommended)
- Graphics card with DX10, DX11, or DX12 capabilities
- Stable internet connection for downloads and cloud integration

2. Unity and Unity Hub

When using the DETAIL resources, it is recommended you use the following versions of Unity to ensure compatibility with the GitHub resources:

For browser-based exhibitions: Unity 2021.3.X LTS or Unity 2022.3.X LTS
For VR/AR exhibitions: Unity 2022.3.X LTS
For mobile-focused exhibitions: Unity 2022.3.X LTS or Unity 6.X

Unity and Unity Hub download process

Visit unity.com/download to download the Unity Hub application.

Create a Unity account if you don't already have one. The personal license is applicable to most projects created at the scale we worked with for the Gallery's digital exhibition pilot projects. Follow the installation instructions for your operating system.

The Unity Hub application manages your Unity version installations and projects in the same place. Once Unity Hub has been installed:

- *Launch Unity Hub* after installation
- *Sign in* with your Unity account
- Navigate to the *Installs* tab
- Click *Add* to install a new Unity version
- Select the recommended Unity version (see above recommendations based on your chosen end platform)
- Ensure the following modules are selected during installation:
    - WebGL Build Support (essential for browser exhibitions)
    - Android Build Support (for VR/mobile deployment)
    - iOS Build Support (if targeting iOS devices)
    - Documentation (recommended for reference)

3. Accessing the repository

The DETAIL toolkit is hosted on GitHub. There are two ways to access it:

Using Git (recommended)

If you're familiar with Git:

1. Install [Git](https://git-scm.com/downloads) if you don't have it already
2. Open your terminal/command prompt
3. Navigate to your desired project location
4. Clone the repository with:
   ```
   git clone https://github.com/MacKenzieArtGallery/DETAIL.git
   ```
5. Wait for the repository to download

Direct download

If you're not familiar with Git, you can still use the resources. Instead of using Git, do the following:

1. Visit the DETAIL GitHub repository page: https://github.com/MacKenzieArtGallery/DETAIL
2. Click the green *Code* button
3. Select *Download ZIP*
4. Extract the ZIP file and drop it into the Assets folder in your Unity project (you'll be setting that up in the next step!)

4. Setting up your project

Now that you have Unity Hub installed and the DETAIL repository downloaded, you're ready to set up your project:

1. Open *Unity Hub*
2. Click the *Projects* tab
3. Click *Add* and select the folder containing the DETAIL repository
4. Select the appropriate Unity version from the dropdown menu
5. Click *Add Project*

The project will now appear in your Projects list. When you open it for the first time, Unity will take some time to import all assets and generate library files.

Once your project opens, set up your project through the following:

1. Navigate to *File* > *Build Settings*
2. Select your target platform (WebGL for browser exhibitions)
3. Click *Switch Platform* if needed
4. Open the *Player Settings* and configure the project name

Importing Assets

Using the methods above, you can either clone the DETAIL GitHub repository into your Unity project's asset folder or download it. If downloaded, unzip and drag everything into your Unity project's Assets folder.

We'll now look at these resources in the Assets folder:

II Resources

1. Animation

Character Animations

- **Character-Auto-Go.anim**: Animation that sets the speed parameter of a character controller to 1, allowing automated movement. Implemented by attaching to a character GameObject and triggered through the Motor animator controller.
- **Character-Auto-Stop.anim**: Animation that sets the speed parameter of a character controller to 0, stopping automated movement. Implemented as a state in the Motor animator controller and triggered via script or animation triggers.
- **Motor.controller**: Animator controller that manages transitions between Character-Auto-Go and Character-Auto-Stop animations based on "go" and "stop" triggers. Implemented by attaching to a GameObject with movement scripts and triggered through code using Animator.SetTrigger().

UI Editor Animations

- **PageContainer-Editor.controller**: Animator controller that manages the states of the page container editor panel using "Show" and "Hide" triggers. Implemented on UI panels containing page editing tools and toggled via UI buttons or code.
- **PageContainerEditor-Off.anim**: Animation that moves a page editor UI element off-screen by adjusting its Y position. Implemented as a state in the PageContainer-Editor controller and triggered when editing mode is deactivated.

- **PageContainerEditor-On.anim**: Animation that shows a page editor UI element by setting its position and size to display in the center of the screen. Implemented as a state in the PageContainer-Editor controller and triggered when entering editing mode.
- **PageEditor-Off.anim**: Animation that hides a page editor by moving its X position to off-screen. Implemented as a state in the PageEditor controller and activated through UI interactions.
- **PageEditor-On.anim**: Animation that displays the page editor by setting its position, scale, and size parameters for optimal viewing. Implemented as a state in the PageEditor controller and triggered when editing functions are requested.
- **PageEditor.controller**: Animator controller that manages transitions between the visible and hidden states of the page editor UI using "Show" and "Hide" triggers. Implemented on UI canvases containing editing tools and toggled via button clicks or code calls.
- **UIEditor-Off.anim**: Animation that slides a UI editing panel off-screen by setting its X position. Implemented as a state in UIPageEditor controller and triggered when exiting editing mode.
- **UIEditor-On.anim**: Animation that displays a UI editor by setting its position parameters to center it on screen. Implemented as a state in UIPageEditor controller and triggered when entering UI editing mode.
- **UIPageEditor.controller**: Animator controller that manages transitions between editor visibility states using "Show" and "Hide" triggers. Implemented on UI editor panels and controlled via UI button events or programmatic calls.

Additional Animations

- **CameraPan.anim**: Animation that smoothly moves a camera through predefined positions to showcase the exhibition environment. Implemented by attaching to the main camera and played during introductory sequences or guided tours.
- **Title.controller**: Animator controller that manages the fade-in and fade-out transitions for title elements using appropriate triggers. Implemented on text or image GameObjects representing exhibition titles or section headers.
- **TitleFade.anim**: Animation that gradually changes the alpha value of a title element to create a smooth fade effect. Implemented as a state in the Title controller and played during scene transitions or when introducing new content sections.

2. DropdownPrefabs

- **DropdownPrefabList.cs**: Maintains a static list of GameObject prefabs that can be selected from dropdown menus in the Unity editor. This serves as the data storage

component for the dropdown system, making prefab references accessible across the project without having to manually assign them each time.

- **DropdownPrefabListAttribute.cs**: A custom PropertyAttribute that marks fields in the Inspector as being selectable from a dropdown list of prefabs rather than using the standard object picker. When applied to a string field using **DropdownPrefabList**, it signals to the editor that this field should display available prefabs from the **DropdownPrefabList** rather than accepting free text input.
- **DropDownPrefabListEditor.cs**: PropertyDrawer that creates the actual dropdown interface in the Inspector, replacing the default string field with a popup menu of available prefabs.

3. Firebase Utilities

The Firebase Utilities assets were used when implementing the server components necessary for installing Xuan Ye's *Belly of the Whale* (2018-) within the THERE IS NO CENTRE (2023) digital exhibition.

- **CheckDatabase.cs**: Establishes a connection to Firebase and verifies if specific data exists in the real-time database, allowing exhibitions to check for previously saved user data or exhibition states.
- **Database.prefab**: A pre-configured Unity prefab that contains all the necessary Firebase components and references, eliminating the need to manually set up Firebase for each new scene or project.
- **DownloadFromDatabase.cs**: A script that retrieves data from the Firebase real-time database and converts it into usable formats within Unity, enabling exhibitions to load saved user progress, artwork configurations, or visitor interactions.
- **README.md**: Text file with a link to Edric Williem's **unity-firebase-realtime-database** documentation (see Plugins below).
- **UploadToDatabase.cs**: A script that takes data from Unity and pushes it to the Firebase real-time database, allowing exhibitions to save visitor interactions, preferences, or artwork states for later retrieval.

4. Materials

- **360Material.mat**: This material is specifically designed for displaying 360-degree content like panoramic images or videos on the interior of spheres, creating immersive environments that surround the viewer. It uses the

UnlitFrontCull shader (see below) which renders only the inside faces of geometry.

- **ImageMaterial.mat**: For the display of 2D images on flat surfaces in the 3D environment, using Unity's standard unlit texture shader to ensure consistent visibility regardless of environmental lighting conditions.
- **UnlitFrontCull.shader**: Custom shader that renders only the interior faces of 3D objects while ignoring exterior faces, a technique essential for creating viewing experiences where users are inside looking out, such as 360-degree video environments.
- **UnlitVertexColor.mat**: This material renders 3D models using only their vertex colors without being affected by scene lighting, making it useful for stylized or low-poly models that need consistent coloration. In the Unity editor, it's applied to objects that should maintain their visual appearance regardless of environmental lighting changes.
- **VideoSkybox.mat**: Wraps video content around the entire viewing environment as a skybox, creating a fully immersive video background that surrounds the viewer from all directions. It's implemented in Unity by assigning it to the scene's skybox setting along with a video render texture, and was used when implementing lighting changes or environmental shifts triggered by proximity.

5. Plugins

This folder contains a fork of Edric Williem's **unity-firebase-realtime-database** for Firebase Realitime Database functionality using REST API calls rather than Firebase's official SDK. This was used when installing Xuan Ye's *Belly of the Whale* (2018-) within the THERE IS NO CENTRE (2023) digital exhibition. Contributors to this repo include Edric Williem, Léo Antoine, and Gal Pasternak.

This plugin allows Unity applications to connect with Firebase databases using simple HTTP requests, which is particularly valuable for WebGL builds (like the browser-based exhibitions described in Exhibit 4.2 and 4.4) where Firebase's official plugins might have compatibility issues. The implementation focuses on core database operations: writing data, reading data, querying with filters, deleting data, and streaming real-time updates.

Documentation for using this plugin is available through the repo and also through this link.

6. Prefabs

Prefabs top folder

- **AddForce.prefab:** Allows adding force to a rigidbody in a specific direction. It uses multiple script components to trigger force application based on events, with configurable force magnitude and direction through scriptable object references.
- **Audio-URL.prefab**: Will load and play audio from a specified URL, utilizing the AudioclipLoader script to download and play audio files dynamically. It includes an AudioSource component configured for playback with options like looping and volume control.
- **Character.prefab:** Unity character prefab with advanced movement and camera control systems, including components for first-person navigation, camera rotation, raycast targeting, and movement toggling. It features modular scripts for character movement, camera positioning, and interaction with game environments.
- **DetectDevice.prefab:** Detects the current device type (mobile or desktop) using a DeviceDetector script, with configurable events to trigger device type detection.
- **Image-Random.prefab:** For randomly selecting and displaying sprites, featuring physics interactions, camera tracking, and the ability to dynamically load and render sprites from a predefined list.
- **Image-URL.prefab:** Prefab designed to load and render an image from a specified URL onto a quad, with an aspect ratio adjustment script to ensure proper image scaling and display.
- **MakeObjectFollowThisObject.prefab:** Prefab with multiple interconnected scripts that enable object tracking, game object search, and position setting through a series of event-driven interactions.
- **MatchPlayerPosition.prefab:** Tracks a player's position in a Unity scene and transfers it to other objects.
- **Spawn.prefab:** Creates a randomized spawning system for game objects. It includes a spawn controller that instantiates objects from a predefined list within a specified range, a timer component that triggers spawning events at randomized intervals, and an event trigger that activates the timer when enabled.
- **Text.prefab:** 3D text display prefab that shows customizable text in the Unity scene. It consists of a parent object with a SetText script that manages the text content, and a child object containing a TextMesh component.

Rooms

- Room.prefab: Template for creating rooms in the digital exhibition. It includes a dictionary-based system for handling named events and string references, allowing flexible configuration of room-specific interactions and content management.
- Room Web.prefab: Room prefab designed for web-based content, featuring world canvases and web view capabilities. This prefab enables the embedding of web

content within the digital exhibition, supporting both 3D and 2D rendering modes with interactive UI elements.

- Room Video.prefab: Room prefab optimized for video content display, inheriting core functionality from the base Room prefab. This template provides a standardized approach to integrating video experiences within the digital exhibition environment.
- Room Object.prefab: A room prefab with object spawning capabilities, featuring a configurable system for dynamically generating and managing game objects within the exhibition space. It supports randomized object placement with customizable quantity and spawn range parameters.
- Room Images.prefab: A room prefab specialized in image display, with multiple world canvases and a sprite list component. This template allows for flexible image presentation across different orientations and supports dynamic image loading and display strategies.

UI

- **Button-ArtistPageLoad.prefab:** For navigating to artist-specific pages within the exhibition interface. Used in the DETAIL exhibitions to provide dedicated navigation to individual artist showcases. This component integrates with the **StringEvent** system to trigger page changes through the **StringEvent-UIPageToActivate** event channel, allowing visitors to smoothly transition between different artist sections while maintaining exhibition coherence.
- **Button-PageLoad.prefab:** A general-purpose navigation button prefab that triggers transitions between different UI pages in the exhibition. This component integrates with the page management system to facilitate visitor navigation through the exhibition's various sections.
- **FontMultiplier.prefab:** A UI utility prefab that dynamically adjusts text size across the exhibition interface based on device characteristics and user preferences. This component works with the **FloatEvent-Font** system to maintain consistent text legibility across different display sizes. It was particularly useful in the mobile-first approach of *Wake Windows*, ensuring text remained readable across various devices.
- **Image.prefab**: Image display prefab optimized for exhibition use, supporting various texture formats and display modes. This component includes adaptive scaling capabilities to maintain proper aspect ratios when displaying artwork. In *THERE IS NO CENTRE*, this prefab was used to display 2D artwork and was configured to maintain proper art representation while functioning effectively within the 3D environment.
- **MainUI-old.prefab:** Legacy user interface prefab from earlier versions of DETAIL that provided core navigation and information display.

- **Page-Content.prefab:** Content display prefab designed for presenting text, images, and interactive elements in a cohesive format. This component serves as a container for exhibition information and artwork descriptions.
- **Page-Menu.prefab:** Menu system prefab that provides structured navigation options for the exhibition. This component typically serves as the main interface through which visitors access different sections of the exhibition. Used extensively in all three DETAIL pilot projects, this prefab offers configurable menu items that can be customized.
- **UI.prefab:** Integrates navigation, information display, and interaction controls into a cohesive system. This component serves as the main interface layer between visitors and the exhibition content. The UI prefab implements responsive design principles to function effectively across different display sizes and platforms, as demonstrated in the multi-platform approach of "Echoes From the Future."

UI Element

- **UIElement.prefab:** Provides a flexible container for user interface components. It includes an input field, a submit button, and event-driven interactions that allow dynamic content manipulation and user input handling.
- **UIElement Text.prefab:** Designed to display text with configurable properties like font, size, and alignment. It's part of a larger UI system that can dynamically set and update text content through Unity's event system.
- **UIElement Text Title.prefab:** UI element optimized for displaying titles, with specific styling or formatting characteristics distinct from body text. It appears to be part of a modular UI design system that allows different text elements for various purposes.
- **UIElement Text Subtitle.prefab**: Similar to the title prefab, this is a text element specifically configured for subtitles, offering a consistent way to render secondary or supporting text in the user interface.
- **UIElement Text Body.prefab:** A text UI element tailored for rendering body or paragraph text, with layout and formatting settings optimized for longer text content within the user interface.
- **UIElement Spawner.prefab:** A dynamic UI element generator that can programmatically create and populate UI components based on predefined configurations.
- **UIElement DefaultPage.prefab:** A default page template for the UI system, configured to be the initial or fallback page layout.
- **UIElement Button.prefab:** A reusable button UI component with built-in text, styling, and event handling capabilities.
- **UIElement Button Scene Load.prefab:** A specialized button configured to load specific scenes.

- **UIElement Button Page Open.prefab:** Another specialized button designed to open specific pages within the application's UI, supporting dynamic page navigation and content switching.

7. Scenes

Pre-made scenes ready for you to drop your digital exhibition assets!

- **UI.unity:** A UI scene setup containing a main camera with black background, directional light, and two UI prefab instances. The main UI canvas is positioned to cover the entire screen, while a secondary "UI Editor" canvas is available but disabled. The scene is configured with proper event system handling for UI interactions. It has a structure suitable for menu interfaces or UI-focused experiences, with anchoring and sizing for responsive design.
- **Video.unity:** A scene dedicated to video playback demonstrations with three different video player prefab implementations: "2DVideo", "3DVideo-Flat", and "3DVideo-360" plus a "Video-360Skybox" for immersive 360° viewing. It includes a basic plane for grounding and a character prefab for viewer perspective. The scene is set up with an event system for UI interaction with video controls, and the video players utilize render textures for displaying content in various formats (2D, 3D flat surface, and 360° immersive).

8. ScriptableObjects

Events

- BoolEvent-UIElementEditToggler.asset: Broadcasts boolean values to toggle the edit mode for UI elements in the digital exhibition editor.
- Event-ActivateBackButton.asset:Triggers the display or activation of a "back" navigation button in the exhibition interface when users navigate deeper into content.
- Event-ActivateMenuButton.asset: Activates the main menu button in the exhibition interface, making it visible or functional when appropriate.
- Event-BackButton.asset: Fired when a user clicks the back button, triggering the navigation system to return to the previous page or state.
- Event-DownloadAllAssets.asset: Initiates the process of downloading all necessary assets (like images, videos, or 3D models) for the exhibition, typically triggered at startup or when entering new exhibition areas.
- Event-MoveToShelf.asset: Triggers the animation or transition of exhibition content to a "shelf" position or state, used in organizing and displaying digital artifacts. When invoked, this event signals objects to reposition themselves into an organized

display format, in our case used for displaying Thoreau Bakker's *VR Cat* (2018-2022) in *THERE IS NO CENTRE* (2023).

- Event-OnShelf.asset: Fired when digital content has completed its transition to the "shelf" position, signaling that the organization process is complete.
- Event-PopCat.asset: This event triggers the spawning function for Thoreau Bakker's *VR Cat* (2018-2022) artwork in the *THERE IS NO CENTRE* exhibition (2023), generating new instances of 3D cat models. The event functions as part of the custom interaction design developed specifically for Bakker's piece, allowing visitors to populate their personal instance of the exhibition with multiple unique cat models that would persist until they exited the exhibition.
- Event-UIActivateEditor.asset: Activates the exhibition editor interface, allowing exhibition creators to modify content and layouts.
- Event-UIToggle.asset: Toggles the visibility of UI elements throughout the exhibition, providing a clean way to show or hide interface components.
- FloatEvent-Font.asset: Broadcasts floating-point values to control font sizing throughout the exhibition interface, enabling responsive typography.
- GameObjectEvent-UIActivatedPage.asset: Broadcasts references to newly activated UI page GameObjects when users navigate to different sections of the exhibition. When a page is successfully loaded and activated, this event carries a reference to the page object, allowing other systems to respond to the new active page.
- GameObjectEvent-UIPageContentParent.asset: Transmits references to the parent container that holds content for UI pages, facilitating the proper placement of dynamically generated UI elements. When new UI content needs to be generated, this event helps identify where in the hierarchy that content should be placed, maintaining proper visual layout and organization.
- GameObjectEvent-UISpawnedPage.asset: Broadcasts references to newly spawned UI page GameObjects as they're instantiated in the exhibition. When a new page is created but before it's fully activated, this event provides access to the new page object, allowing for initial setup and configuration before the page becomes visible.
- StringEvent-LoadScene-Additive.asset: Triggers the loading of a Unity scene additively (without unloading the current scene) using a string parameter to specify the scene name. It's used to load artwork environments or supplementary content while maintaining the main exhibition interface, creating seamless transitions between different exhibition spaces.
- StringEvent-LoadScene.asset: Triggers the loading of a Unity scene using a string parameter to specify which scene to load, replacing the current scene. It's used for major transitions between exhibition areas or completely different exhibition spaces, such as moving from a lobby to a specific artist's showcase.
- StringEvent-Message.asset: Broadcasts text messages throughout the exhibition system, often used for user feedback, notifications, or debug information. It's

connected to UI text elements, notification systems, or logging functionality to display information to users or developers about exhibition state changes.

- StringEvent-PageLoadButton.asset: Transmits the name of a page that should be loaded when a page navigation button is pressed in the exhibition interface. It connects UI buttons to the navigation system, allowing buttons to specify which page should be displayed next when they're activated.
- StringEvent-UIButtonPageOpen.asset: Broadcasts the name of a UI page that should be opened when triggered, initiating the page transition process. Similar to PageLoadButton but potentially used in different contexts, this event communicates to the navigation system which page to display next.
- StringEvent-UINameOfElement.asset: This event broadcasts the name of a UI element that needs to be referenced or manipulated, facilitating component lookup by name instead of direct references. It's used when components need to communicate about specific UI elements without maintaining direct references, supporting more flexible and dynamic UI structures.
- StringEvent-UIPage.asset: This event transmits the name of a UI page to various systems in the exhibition, used for navigation and state tracking.
- StringEvent-UIPageFromList.asset: Broadcasts the name of a UI page selected from a list or menu of available pages, helping manage navigation through collections of content.
- StringEvent-UIPageToActivate.asset: Transmits the name of a UI page that should be activated next in the navigation sequence.
- StringEvent-UIPageToSpawn.asset: Broadcasts the name of a UI page that needs to be dynamically instantiated in the exhibition. Unlike pages that are pre-built in the scene, this event triggers the creation of entirely new page instances, supporting more dynamic and content-driven exhibition experiences.
- StringListEvent - Names of UIElements.asset: Broadcasts lists of UI element names available in the exhibition, supporting dynamic menus and selection interfaces.
- StringListEvent - Names of UIPages.asset: Transmits lists of available UI page names, supporting navigation menus and content selection interfaces.
- TransformEvent-MoveTarget.asset: Broadcasts references to Transform components that need to be repositioned or animated within the exhibition. It's used to signal when objects should move to new positions, rotate, or scale, supporting the dynamic and interactive nature of digital exhibitions.

UI

- _UIElements.asset: Serves as a centralized registry of UI element prefabs (like buttons, text fields, and page containers) that can be instantiated through the UI system. The asset maintains a list of named UI elements along with references to

their prefab objects, enabling the spawning of UI components by name rather than directly referencing prefabs.

- _UIPages.asset: Functions as a dictionary-style container that manages different "pages" or UI screen configurations for the digital exhibition interface, with each page representing a distinct section of the user interface. It works alongside the UI navigation system to track which pages are available, handle transitions between them, and maintain their content associations, serving as the backbone of the exhibition's menu and navigation structure.
- String-StartPage.asset: Stores the name of the initial UI page that should be loaded when the exhibition first launches, acting as a configurable entry point to the user interface.

GameObjectFunctions

- ActivateGameObject.asset: Allowed us to define whether a specific game object should be activated or deactivated, which can be useful for managing visibility, interaction, or performance in digital exhibitions by dynamically showing or hiding elements based on user interactions or specific exhibition states.
- GameObjectEvent-ActivatePage.asset: Manages pages activation events in the UI system of the DETAIL digital exhibitions. It works in conjunction with other UI scripts to trigger the display or hiding of specific pages or content sections, allowing for dynamic and interactive navigation through the exhibition's interface by responding to user interactions or system events.
- GameObjectFunction-TriggerAnimOff.asset: Provides a mechanism to trigger the "Off" state of an animator, which can be used to control animations in the digital exhibition. It allows developers to programmatically switch animations to an "off" or deactivated state, potentially helping manage complex animation sequences or user-triggered visual transitions within the exhibition experience.

- ToggleRigidbody.asset: Enables or disables the Rigidbody component of a GameObject, which controls physics interactions in Unity.

9. Scripts

- AdjustQuadSizeToVideoAspectRatio.cs: Resizes a quad (rectangular plane) to match the aspect ratio of a video being played, ensuring that video content is displayed without distortion. By automatically scaling the quad's dimensions based on the video's width and height, it solves the common visual problem of stretched or squeezed video content.

- AnimatorEvent.cs: Allows you to trigger specific actions at precise moments during an animation sequence. This script enables more complex and interactive animations by providing a mechanism to invoke custom Unity events directly from animation states.
- ApplicationQuit.cs: Quits the application/exhibition.
- AssetLoader.cs: Dynamically loads and instantiate game objects during runtime.
- AudioclipLoader.cs: For loading and playing audio files from URLs, supporting multiple audio formats and handling potential loading errors.
- BasicMovement.cs: Character movement script that allows for basic first-person or third-person navigation using standard input axes (horizontal and vertical movement).
- CheckDate.cs: Compares the current date against a predefined target date, enabling time-based interactions or content visibility. In the context of digital exhibitions, this can be used to reveal or hide certain artworks, create time-specific experiences, or manage exhibition lifecycles.
- DestroyGameObject.cs: Provides a method to programmatically destroy game objects, which is useful for managing dynamic object lifecycles in interactive exhibitions.
- DestroyOnLoadController.cs: Manages game objects that should persist across scene loads or be destroyed in a controlled manner. This helps maintain consistent application state and manage resources effectively during scene transitions in digital exhibitions.
- DeviceDetector.cs: A script that detects the current device platform (mobile or desktop) and triggers corresponding events, enabling platform-specific optimizations or user experiences. This was crucial in the DETAIL exhibitions for adapting the exhibition interface and performance to different device types.
- DictionaryOfDictionaries.cs: Manages nested dictionary structures, allowing for data management and event-driven interactions with string-based configurations. This script provides a flexible system for managing exhibition metadata, UI configurations, and dynamic content loading.
- DownloadAllAddressables.cs: For managing the download of all addressable assets in a Unity project, providing progress tracking and user confirmation mechanisms. This script was essential in the DETAIL exhibitions for managing large artwork downloads.

- FontSizeMultiplier.cs: Adjusts text font sizes based on a multiplier, enabling responsive typography across different device sizes or user preferences. This script supports accessibility and adaptable user interfaces in digital exhibitions.
- GetTextureFromRawImage.cs: Extracts textures from RawImage components, facilitating texture manipulation and processing in UI contexts.

- GuidePlayer.cs: For creating guided audio-visual experiences, allowing for complex waypoint-based navigation with synchronized audio playback. Used in creating video documentation of the exhibitions.
- ImageLoader.cs: For dynamically loading images from URLs, with error handling and texture output events.
- IsCameraLookingAtCollider.cs: Determines whether a camera is currently looking at a specific collider, enabling interaction detection.
- ListOfNamedUIElements.cs: For managing lists of named UI elements, providing a flexible system for dynamically populating and managing UI configurations.
- ListOfTextures.cs: Manages a list of textures and provides methods for randomly selecting or outputting textures, used for Thoreau Bakker's *VR Cat* (2018-2022).

10. Textures

- VideoRenderTexture.renderTexture: Specialized Unity render texture that captures and processes video content dynamically within the rendering pipeline. This render texture serves as an off-screen image buffer that can capture a video player's output and apply it flexibly across different rendering contexts, such as texturing 3D objects or creating complex UI elements. In the DETAIL digital exhibitions, this texture enabled dynamic video integration, allowing curators and developers to creatively display video art across various digital exhibition interfaces.

11. Utilities

- AddForceAtPoint.cs: Applies force to a Rigidbody at a specific point, enabling object interactions and dynamic movement.
- AddForceInDirection.cs: Applies directional force to a Rigidbody, allowing for controlled and predictable object movement in a specified direction.
- AudioAnalysis.cs: Audio processing script that performs real-time analysis of audio signals, extracting features like frequency, amplitude, or spectral data. Enables interactive audio-visual experiences where visual elements respond dynamically to sound.
- AudioDistanceInterval.cs: Manages audio playback based on distance between objects, creating spatially aware sound experiences. Used to create immersive audio environments where sound intensity or playback changes based on viewer proximity.
- AverageFloatList.cs: A utility script that calculates the average value from a list of floating-point numbers, useful for smoothing data or creating normalized representations. Can be used to help manage dynamic visual or interactive elements that require consistent, averaged input.

- CountFiles.cs: A file system utility that counts the number of files in a specified directory, potentially useful for dynamic content loading or inventory management. Can help manage asset libraries or track the number of available artworks.
- CreateRenderTexture.cs: A script for programmatically generating render textures in Unity, enabling dynamic texture creation for complex rendering scenarios. Can support creating dynamic video or image display surfaces.
- CustomEvents.cs: An event management script that provides a framework for creating and managing custom Unity events with specific parameters.
- DetectNavMeshDestinationDistance.cs: A navigation utility that tracks the distance between an object and its navigation mesh destination, useful for movement and interaction tracking.
- DirectionFromTransform.cs: Calculates directional vectors between transforms, enabling precise spatial relationship calculations. Supports interactive elements that respond to viewer positioning or create directional-based interactions.
- DirectionToTransform.cs: Calculates the directional vector pointing from one transform to another, enabling precise spatial relationship tracking. Supports interactive elements that orient themselves based on relative positioning.
- FindGameObject.cs: Locates specific game objects within a scene by name or other criteria.
- FindGameObjectByTag.cs: Identifies game objects in a scene using Unity tags.
- FloatConditional.cs: Performs conditional operations on float values, enabling complex numerical logic and decision-making within interactive systems. Supports nuanced interaction design in digital exhibitions, like when creating threshold-based visual or audio responses.
- FloatOperation.cs: Executes mathematical operations on float values.
- FloatToString.cs: Converts float values to string representations, facilitating text display and formatting of numerical data. Used for creating dynamic labels, information displays, or user feedback mechanisms.
- FollowTransform.cs: Enables one object to smoothly track the movement and rotation of another transform, creating dynamic following behaviors. Used to create guided experiences or interactive elements that respond to viewer or artwork movements.
- ForwardVectorEvent.cs: Generates events based on the forward direction of a transform, enabling directional-based interactions and triggers. Used to create navigation-sensitive or orientation-dependent interactive experiences.
- GameObjectPool.cs: Manages a pool of reusable game objects, optimizing performance by reducing instantiation and destruction overhead. Can significantly improve rendering and interaction efficiency.
- GenericEvent.cs: Provides a flexible, type-agnostic event system for creating custom interactions between exhibition components. This utility supports modular, extensible interaction design across different artworks and exhibition interfaces.

- IncrementFloat.cs: Systematically increases or decreases a float value by a specified amount, enabling controlled numerical progression. Can be used for creating dynamic animations, scoring systems, or time-based interactive elements that evolve incrementally.
- InstantiateGameObject.cs: Creates new game objects during runtime, allowing dynamic population of exhibition spaces with interactive or visual elements.
- IterateChildrenIntoList.cs: Collects all child objects of a parent transform into a list, providing a mechanism for managing hierarchical object relationships. Can help organize complex scene structures or create systematic interactions between parent and child elements.
- IterateInt.cs: Manages iterative processes using integer values, enabling controlled stepping through numerical sequences.
- KeyInputEvents.cs: Captures and responds to keyboard input, creating interaction triggers based on specific key presses. This utility supports keyboard-navigable exhibition interfaces or add alternative interaction methods for artworks, including text submission like in *Wake Windows*.
- LifeCycleTriggerEvents.cs: Manages events related to object lifecycle stages like initialization, activation, or destruction.
- MaintainDistanceFromTerrain.cs: Ensures objects remain at a consistent height or distance from terrain surfaces, preventing unintended spatial interactions or clipping of visitor avatars into the terrain.
- MouseClickEvents.cs: Detects and responds to mouse click interactions, enabling point-and-click style engagement with digital exhibition elements.
- MouseHoverRaycast.cs: Implements hover detection by casting rays from the mouse cursor to identify objects underneath, creating interactive hover states.
- MoveBetweenTransforms.cs: Smoothly transitions objects between different predefined transform positions, creating guided movement sequences.
- NamedToggleEvents.cs: Manages a collection of named boolean toggles with associated events, allowing complex state management for interactive elements. Supports intricate user interface interactions where multiple named states can be tracked and triggered independently.
- RandomAudioClip.cs: Selects and plays a random audio clip from a predefined list, introducing variety and unpredictability into sound experiences.
- SaveLoadScriptableObjects.cs: Provides mechanisms for saving and loading ScriptableObject data, enabling persistent state management across exhibition sessions.
- ShowObjectName.cs: Displays the name of a game object, facilitating debugging and providing information about scene elements.
- SpawnObjects.cs: Dynamically generates game objects at specified locations, allowing for on-demand content creation within the exhibition space. Supports adaptive, responsive exhibition designs where content can be generated based on user interactions or predefined triggers.

- SpinLogic.cs: Implements rotational movement for game objects, creating spinning or rotating interactive elements. Used to add dynamic motion to artworks, create attention-grabbing visual effects, or simulate mechanical interactions.
- StateVariables.cs: Manages a collection of variables representing different states, enabling complex state-based interactions and logic. Supports intricate exhibition experiences with multiple interconnected interactive states.
- StringDictionary.cs: Creates and manages dictionary-based string storage and retrieval, providing a flexible system for managing text-based data.
- SuperLerp.cs: Implements advanced linear interpolation techniques, allowing smooth transitions between values with customizable acceleration and deceleration.
- SwitchEvent.cs: Manages event switching mechanisms, enabling complex conditional event triggering and state changes. Digital exhibitions might use this to create branching interactions or dynamically modify exhibition experiences based on user actions.
- TextureToTexture2D.cs: Converts render textures or other texture types into standard Texture2D format, enabling flexible image processing and manipulation.
- TimedEvent.cs: Triggers events after a specified time interval, creating timed interactions or scheduled changes within the exhibition space. Used when triggering time-sensitive dialogue options in *Wake Windows.*
- ToggleEvent.cs: Manages boolean toggle states with associated events, allowing for binary on/off interactions with complex underlying behaviors.
- TransformMoveToward.cs: Smoothly moves objects towards a target transform, creating controlled and precise spatial transitions.
- TransformRotateTowards.cs: Implements rotation towards a specific target, enabling objects to orient themselves dynamically based on spatial relationships. Used to create interactive elements that respond to viewer positioning or artwork interactions.
- TriggeringEvents.cs: Manages complex event triggering mechanisms with multiple conditions and response types, supporting intricate interactive systems.
- UserController.cs: System for managing user interactions, input handling, and movement within the exhibition space. Supports navigation, interaction tracking, and creating consistent user experience across different artworks.
- Vector3List.cs: Manages collections of 3D vector positions, enabling complex spatial data manipulation and tracking.
- VectorMath.cs: Performs mathematical operations on vector values, supporting spatial calculations and transformations.
- VectorSamplePosition.cs: Generates sample positions within a defined vector space, supporting procedural content generation or spatial distribution. Can be used to create randomized object placements or generate dynamic exhibition layouts.

- VectorToString.cs: Converts vector values to string representations, facilitating text display and formatting of spatial data. Supports information displays, debugging, or creating human-readable spatial information in the exhibition.

III Pilot project workflows

1. THERE IS NO CENTRE

The THERE IS NO CENTRE exhibition was a single-user browser-based WebGL experience with:

- A hub-and-spoke layout with elevator navigation
- Multiple independent artwork environments
- Cloud-based asset loading for efficient browser performance
- Custom interactions for each artist's work
- First-person WASD navigation

The following assets in the GitHub folder were used to develop the pilot project:

Core Navigation & Player Movement

For implementing first-person WASD navigation:

- **Character.prefab**: Provides a complete first-person character controller with movement and camera systems
- **BasicMovement.cs**: Handles foundational character movement mechanics
- **Scripts/UserController.cs**: Manages user input and movement in exhibition spaces

Hub-and-Spoke Layout & Elevator System

For the central hub navigation system with elevator:

- **Room.prefab**: Template for creating individual room environments
- **MoveBetweenTransforms.cs**: Enables smooth transitions between positions (useful for elevator movement)
- **FollowTransform.cs**: Can be used to make the elevator follow specific paths
- **AnimatorEvent.cs**: For triggering events during elevator transitions

Cloud-Based Asset Loading

For efficient browser performance with cloud storage:

- **DownloadAllAddressables.cs**: Manages downloading of assets from cloud storage
- **AssetLoader.cs**: Handles loading and instantiation of assets with progress tracking
- **LoadingProgress events**: For updating users on download status
- **Firebase Utilities**: Used in Xuan Ye's *Belly of the Whale* installation

Independent Artwork Environments

For creating separate artwork spaces:

- **Room Object.prefab**: For artwork spaces with interactive 3D objects
- **Room Video.prefab**: For video-based artwork installations
- **Room Images.prefab**: For image-based installations
- **Room Web.prefab**: For web-based artwork integration

Custom Artist Interactions

For implementing artist-specific interactions:

- **Event-PopCat.asset**: Example of a custom interaction for Thoreau Bakker's *VR Cat*
- **IsCameraLookingAtCollider.cs**: For gaze-based interactions
- **AddForceInDirection.cs**: For physics-based interactions
- **RandomMaterial.cs**: For randomized visual elements (used in VR Cat)
- **ListOfTextures.cs**: For managing collections of visual assets
- **SpawnObjects.cs**: For generating multiple instances of artwork elements

WebGL & Browser Optimization

For ensuring good performance in browser environments:

- **UnlitVertexColor.mat**: Lightweight material for consistent visuals without lighting calculations
- **DeviceDetector.cs**: For adapting the experience based on device capabilities
- **DestroyOnLoadController.cs**: For managing scene transitions and memory optimization
- **VideoRenderTexture.renderTexture**: For efficient video display

UI & Navigation Helpers

For implementing user interface elements:

- **UIElement prefabs**: For creating consistent UI components
- **StringEvent-Message.asset**: For displaying information to users
- **Event-ActivateBackButton.asset**: For navigation controls

- **Event-UIToggle.asset**: For toggling interface visibility

2. Echoes From the Future

The *Echoes From the Future* exhibition was a downloadable exhibition that supported both virtual reality headsets and computer access, featuring:

- A single, large open-world with all artworks installed
- Cloud-based asset loading for efficient browser performance
- Custom interactions for each artist's work
- First-person WASD navigation with a minimap for wayfinding
- Multi-user functionality with voice chat

It is recommended you use the 3D Navigation Multi-User template guide to set up an exhibition like *Echoes* with free, open-source multi-user functionality. Please note that the *Echoes* multi-user functionality relied on the proprietary Normcore multi-user plugin due to time constraints, and the functionality for full multi-user is not held within the DETAIL GitHub but through the template linked above.

Other available material in the DETAIL GitHub includes:

Cross-Platform Support (VR and Desktop)

For supporting both VR headsets and desktop:

- **VRHeadsetDetection.cs**: Detects if a VR headset is connected
- **VRDetection.cs**: Provides more comprehensive VR platform detection
- **PlatformCheck.cs**: Identifies the runtime platform for appropriate adaptations
- **DeviceDetector.cs**: For detecting device type and capabilities

Open-World Environment

For creating a large, unified exhibition space:

- **Room.prefab**: Base template adaptable for larger environment sections
- **MaintainDistanceFromTerrain.cs**: Ensures proper player height above terrain
- **FindGameObjectByTag.cs**: Useful for locating distant objects in a large world
- **SetSkybox.cs**: For creating atmospheric environmental effects

Navigation and Wayfinding

For implementing first-person movement with minimap:

- **Character.prefab**: Provides first-person character controller
- **BasicMovement.cs**: Handles character movement mechanics
- **RectFollowTransform.cs**: Can be used to create a player indicator on a minimap
- **UIRotateToward.cs**: For orienting directional indicators on the minimap

Multi-User Functionality

For implementing multi-user experiences:

- **Firebase Utilities**: For synchronizing user positions and states
- **UploadToDatabase.cs**: For sharing player positions and interactions
- **DownloadFromDatabase.cs**: For retrieving other players' data
- **CheckDatabase.cs**: For verifying connection to shared environment

Voice Chat Integration

While specific voice chat implementation isn't directly available in the assets, these would help integrate with external voice solutions:

- **AudioDistanceInterval.cs**: For distance-based voice attenuation
- **AudioclipLoader.cs**: For handling audio streams
- **AudioAnalysis.cs**: For audio processing and visualization

Artwork Installations in Open World

For placing and managing multiple artworks in a single environment:

- **SetGameObjectMaterial.cs**: For consistent material management across artwork installations
- **ImageLoader.cs**: For loading 2D artwork assets
- **VideoController.cs**: For managing video-based artworks
- **360Material.mat**: For immersive 360-degree environments
- **VideoSkybox.mat**: For creating environmental video backdrops

Interactive Artwork Elements

For implementing artist-specific interactions:

- **AddForceAtPoint.cs** and **AddForceInDirection.cs**: For physics-based interactions like with Reiner Maria Matysik's *Beings* installation
- **IsCameraLookingAtCollider.cs**: For gaze-based interactions as used in Laura Colmenares Guerra's *Ríos Trilogy*, *Chapter N. 3 R€¥€R$€*

- **RandomMaterial.cs** and **ListOfTextures.cs**: For randomized visual elements used within Tamiko Theil and /p's *What You Sow*
- **MouseHoverRaycast.cs**: For interactive elements that respond to user attention, used in Laurea Colemnares Guerra's installation
- **ToggleChildrenOnTrigger.cs**: For proximity-based artwork activation such as activating videos in Bianca Shonee Arroyo-Kriemes's L*ast Species On Earth*

Cloud-Based Asset Management

For efficient asset loading and management:

- **DownloadAllAddressables.cs**: For managing large asset downloads
- **AssetLoader.cs**: For loading and instantiation with progress tracking
- **DestroyOnLoadController.cs**: For efficient memory management

Environmental Effects and Lighting

For creating atmospheric conditions mentioned in the exhibition:

- **SetSkybox.cs**: For changing the environmental backdrop
- **Materials/VideoSkybox.mat**: For creating immersive skyboxes
- **Materials/UnlitFrontCull.shader**: For interior-facing display surfaces

Navigation Helpers for Large Environment

For helping users find their way in the large space:

- **GuidePlayer.cs**: For creating guided tours or pathways between artworks
- **MoveBetweenTransforms.cs**: For smooth transitions between points of interest
- **Event-MoveToShelf.asset**: Can be adapted for movement between exhibition areas

3. Wake Windows: The Witching Hour

Wake Windows: The Witching Hour was a browser-based WebGL exhibition, featuring:

- Text-based navigation for single users with branching narrative and time of day variables.
- Prioritization of mobile browser access.
- Cloud-based asset loading for efficient browser performance.
- Custom interactions for each artist's work

Text-Based Navigation System

Please note that the Pixel Crushers's Dialogue System plugin was used to handle branching narrative development by the exhibition's narrative designer. Other tools we used for implementing narrative-driven text navigation are however available, and include:

- **StringEvent-Message.asset**: For displaying dialogue text and narrative content
- **StringEvent-UIPage.asset**: For managing page/scene transitions in the narrative flow
- **UIElement Text.prefab**: For displaying text content with configurable properties
- **UIElement Text Title.prefab**: For dialogue titles or character names
- **UIElement Text Body.prefab**: For main dialogue content

Branching Narrative Structure

For creating interactive narrative paths:

- **DictionaryOfDictionaries.cs**: For managing complex dialogue trees and narrative branches
- **StringDictionary.cs**: For storing and retrieving dialogue options and responses
- **NamedToggleEvents.cs**: For tracking user choices and narrative state
- **StringPool.cs**: For managing collections of available text content

Time-of-Day Variables

For implementing time-sensitive content:

- **TimeOutput.cs**: Outputs the current system time and can calculate percentage of day passed
- **CheckDate.cs**: For comparing current time against predefined targets
- **FloatConditional.cs**: For creating conditional logic based on time values
- **Event-UIToggle.asset**: For showing/hiding time-specific content

Cloud-Based Asset Loading

For efficient browser performance with cloud storage:

- **DownloadAllAddressables.cs**: Manages downloading of assets from cloud storage
- **AssetLoader.cs**: Handles loading and instantiation of assets with progress tracking
- **StringEvent-LoadScene-Additive.asset**: For loading additional content without disrupting the narrative

Custom Artist Interactions

For implementing artist-specific interactive elements:

- **ImageLoader.cs**: For loading artwork images from URLs
- **VideoController.cs**: For managing video-based artworks with playback controls
- **VideoScrubbingControl.cs**: For interactive video timeline scrubbing
- **ScrollRectVideoController.cs**: For managing videos in scrollable UI elements

Mobile and Browser Optimization

For ensuring good performance across devices:

- **DeviceDetector.cs**: For detecting mobile vs. desktop devices
- **ScreenOrientationDetector.cs**: For adapting to portrait/landscape orientations
- **SafeArea.cs**: For ensuring content appears in the safe area on mobile devices
- **DestroyOnLoadController.cs**: For efficient memory management

UI Components for Narrative Interface

For building the narrative interface:

- **UIElement Button.prefab**: For dialogue choices and navigation options
- **UIPageEditor controller**: For managing different narrative pages
- **StringEvent-UIButtonPress.asset**: For capturing user selections
- **SpawnUIElements.cs**: For dynamically generating dialogue options

Cookie/State Management

For remembering returning visitors:

- **CheckDatabase.cs**: Can be adapted to check for cookies/local storage
- **StringEvent-UIPageToActivate.asset**: For loading appropriate starting pages for new vs. returning visitors
- **StringEvent-UIPageToSpawn.asset**: For dynamically generating content based on visitor status

IV. Troubleshooting

Common issues and solutions:

1. **Video Playback Issues**:
    - Ensure video codecs are web-compatible (H.264, WebM)
    - Use VideoPreparer.cs to preload content
    - For iOS, recommend Safari for best compatibility

2. **Memory Limitations**:
    - Implement cloud-based asset loading with AssetLoader.cs
    - Load and unload assets based on proximity
    - Split large environments into separate scenes
3. **Cross-Platform Compatibility**:
    - Use platform detection scripts to adapt the experience
    - Test on target platforms regularly throughout development
    - Provide fallback options for unsupported features

V. Links to references

DETAIL GitHub Repository
Unity WebGL Documentation
MacKenzie Art Gallery DETAIL webpage
DETAIL Pilot Project documentation