**JSon**

databaseMan.cs

This script is the closest one-for-one translation of the Json simple file we received from Praeses. The classes within it was built to accommodate the architecture of the original Json structure as close as we could. With the exception that some class names couldn’t be only numbers. And we introduced new classes like current inspection values, and non-displayable fields for UX purposes.

Because certain newtonSoft.json.dll can only work for the UNITY\_EDITOR and another only works for WINDOWS\_UWP, we have to switch between the two dll files everytime we want to play back the scene in the unity editor, using version 3.5, or for the deploy version we use 4.x. Respectively when we do this we have to run certain lines and comment them out so it won’t throw an error during building.

addAnnotation(GameObject)

This function adds a new node item inside the databaseMan class, alongside the comments and medias within them

removeNode(GameObject)

This function deletes a node in the class

formToClassValueSync(string keyword, string value)

This function is performed at the end of editing fields with a keyboard and adds a new current inspection item inside the class

nodeToClassValueSync(int nodeIndex, string value, int valueType)

This function is performed when a new node is created within the session

commentToClassValueSync(int nodeIndex, tempComment comment)

This function is run when a new comment is created this may be simple text, photo, or video

syncViolation(violationController violation)

This function is performed when reading a violation item from Json, or when a new violation is created.

updateVio(violationController violation)

This function is performed when an existing violation is being edited.

JU\_databaseMan.cs

This script is an attempt at making the readable data to be more streamline from databaseMan. Most of the functionalities inside this script only reads from databaseMan.cs, in this respect they are run automatically every time the databaseMan class gets an update. The difference is no field generation or node population within the unity scene ever reads directly from databaseMan. JU\_databaseMan.cs serves as a translator so definitions and values can be retrieved in a less cumbersome way.

categoryParser(string numbers, int category)

This function is for string splitting the violations category.subcategory.specific numbers

violationParser(databaseMan.ViolationsClass incomingItem)

This is for translating the violations class from databaseMan.cs to JU\_databaseMan.cs

List<string> categoryStringer(ViolationsItem violation)

This is an automation converter between the number codes for the violations categories to its text dictionary pair that is defined in violationsLib.cs

addNodeFromJSon.cs

This script is responsible in populating the 3D nodes in the scene for the onsite as well as the offsite PC companion app. It features the reordering of comments by date within the nodes for onsite.

The offsite version is less robust because you don’t have the ability to click the nodes directly to play medias on them, they come as a media thumbnail list that displays the location of the node as you click to play them.

violationsLib.cs

This script is a library of the violations category trees, it is only partially done for the POC, it will need to be completed per its complete library it is reading from. Right now if a certain path does not exist it will revert to a hard coded string value.

**Keyboard**

keyboardScript.cs

This is the manager for any operation pertaining keyboard actions. It features:

keyboardToggle()

- the activating and deactivating of the keyboard

nestedOn()

- the type whether it’s numpad or QWERTY

capsToggle()

-caps lock by double tapping

startRecording()

- activating voice dictation, works in conjunction with Dictationizer.cs under the InputManager prefab

typeSuggestion()

- displaying previous value on a quick access fill button

typeObject()

- typing a specific character based on the object name via split string (key\_#) where # is the desired character to type, this allows repurposing of this script across every character in the keyboard prefab

editChangeSync()

- the done button sends the function to accordingly update comments or field items and their corresponding json objects in inputFieldManager.cs and databaseMan.cs

**Minimap Prefab**

onModelDragHybrid.cs

This script is attached to a minimap prefab child group called “tumbledScript”. Its purpose is to detect objects with the tag “miniMapMesh”. When the gaze hits on object with this tag, pinch and holding would enable the mesh collider plane on this object, activating the hand cursor and the quadrants and buttons for tumbling the minimap. This script also controls how the menu appears to the user, it will scale smaller accordingly if the user is closer to the minimap. It has a clamp value for its maximum size so if the user is really far from the map it won’t appear too large. This script controls the hand manipulation of the hand cursor as well. There is a separate cameraOrient.cs script that is attached to a null object oriented to the main camera, so we can grab the local position of the object for hand manipulation instead of world.

radialOperationsHybrid.cs

This is attached to every icon/button of the minimap tumbler. They have typing int that will tell what kind of operation is being performed(scale or rotate) with a settable factor to go positive or negative. When the hand cursor sphere collider enters these colliders then it will trigger the operations. Scale and rotation are done on separate groups under the minimap hierarchy so the pivot will stay consistently on the boiler as the center.

minimize.cs

This is attached to a live size avatar “person” inside the minimap prefab in the scene that follows the main camera’s transform. By default, the mesh renderer is turned off because we don’t want to see it ever. This script on finalize will duplicate a smaller version of it where it will mimic its movement in real space. Both will need to exist so they can talk to each other.

followCam.cs

This is attached to the avatar as well, with some variables like the head so the shoulders of the avatar can follow main camera rotateY, while rotateX and RotateZ will be tied to the head object.

followViz.cs

This is a script to sync the mesh visibility of a List<GameObject> to a specific one, for ease of turning on and off their visuals at once.

minimapTransferObject.cs

This script is for easily duplicating objects from real space into minimap space.

**Minimap Spatial Mapping Items**

boilerTransformRecorder.cs

This saves out the boiler transform and typing(pop up boiler or regular) in a json file in the persistent data path for offsite “boilerTransform.json”

minimapSpawn.cs

This script duplicates the contents of the spatial map into another container that then gets manipulated to a smaller scale, alongside minimapTransferObject.cs it will bring the nodes and avatar also into mini nodes in the minimap container.

spatialMappingObserver.cs

This script saves out the room mesh into a file in the persistent data path called “JU\_spatialMesh.room”

**Offsite**