Supporting Document: Sign & Spell VR

Name: Karl Negrillo
Student Number: C22386123
Supervisor: Colette Kirwan
Project Title: Sign & Spell VR: ISL Alphabet and Word Practice in Virtual Reality

Code

```
@tool
class_name HandPose
extends Resource


## Hand Pose Resource
##
## This resource defines a hand pose. It consists of the pose name, and a
## collection of rules defining the pose.


## Hand pose name
@export var pose_name : String

## Detection threshold
@export_range(0.0, 1.0) var threshold : float = 0.5

## Hold time before pose is detected
@export_range(0.01, 2.0) var hold_time : float = 0.2

## Release time before pose is lost
@export_range(0.01, 2.0) var release_time : float = 0.2

# Flexion group
@export_group("Flexion", "flexion_")

## Flexion Thumb Function
@export var flexion_thumb : FitnessFunction

## Flexion Index Function
@export var flexion_index : FitnessFunction

## Flexion Middle Function
@export var flexion_middle : FitnessFunction

## Flexion Ring Function
@export var flexion_ring : FitnessFunction

## Flexion Pinky Function
@export var flexion_pinky : FitnessFunction

# Curl group
@export_group("Curl", "curl_")

## Curl Thumb Function
@export var curl_thumb : FitnessFunction

## Curl Index Function
@export var curl_index : FitnessFunction

## Curl Middle Function
@export var curl_middle : FitnessFunction

## Curl Ring Function
@export var curl_ring : FitnessFunction

## Curl Pinky Function
@export var curl_pinky : FitnessFunction

# Abduction group
@export_group("Abduction", "abduction_")

## Abduction Thumb-Index Function
@export var abduction_thumb_index : FitnessFunction

## Abduction Index-Middle Function
@export var abduction_index_middle : FitnessFunction

## Abduction Middle-Ring Function
@export var abduction_middle_ring : FitnessFunction

## Abduction Ring-Pinky Function
@export var abduction_ring_pinky : FitnessFunction

# Tip-distance group
@export_group("Tip-Distance", "distance_")

## Tip-Distance Thumb-Index Function
@export var distance_thumb_index : FitnessFunction

## Tip-Distance Thumb-Middle Function
@export var distance_thumb_middle : FitnessFunction

## Tip-Distance Thumb-Ring Function
@export var distance_thumb_ring : FitnessFunction

## Tip-Distance Thumb-Pinky Function
@export var distance_thumb_pinky : FitnessFunction


## Returns a fitness value in the range 0..1 as to how well the [param hand]
## matches the hand pose rules.
func get_fitness(hand : HandPoseData) -> float:
    # Process the fitness of all rules
    var fitness := 1.0

    # Apply flexion rules
    if flexion_thumb:  fitness *= flexion_thumb.calculate(hand.flx_thumb)
    if flexion_index:  fitness *= flexion_index.calculate(hand.flx_index)
    if flexion_middle: fitness *= flexion_middle.calculate(hand.flx_middle)
    if flexion_ring:   fitness *= flexion_ring.calculate(hand.flx_ring)
    if flexion_pinky:  fitness *= flexion_pinky.calculate(hand.flx_pinky)

    # Apply curl rules
    if curl_thumb:  fitness *= curl_thumb.calculate(hand.crl_thumb)
    if curl_index:  fitness *= curl_index.calculate(hand.crl_index)
    if curl_middle: fitness *= curl_middle.calculate(hand.crl_middle)
    if curl_ring:   fitness *= curl_ring.calculate(hand.crl_ring)
    if curl_pinky:  fitness *= curl_pinky.calculate(hand.crl_pinky)

    # Apply abduction rules
    if abduction_thumb_index:  fitness *= abduction_thumb_index.calculate(hand.abd_thumb)
    if abduction_index_middle: fitness *= abduction_index_middle.calculate(hand.abd_index)
    if abduction_middle_ring:  fitness *= abduction_middle_ring.calculate(hand.abd_middle)
    if abduction_ring_pinky:   fitness *= abduction_ring_pinky.calculate(hand.abd_ring)

    # Apply tip-distance rules
    if distance_thumb_index:  fitness *= distance_thumb_index.calculate(hand.dst_index)
    if distance_thumb_middle: fitness *= distance_thumb_middle.calculate(hand.dst_middle)
    if distance_thumb_ring:   fitness *= distance_thumb_ring.calculate(hand.dst_ring)
    if distance_thumb_pinky:  fitness *= distance_thumb_pinky.calculate(hand.dst_pinky)

    # If the fitness is below the threshold then fail
    if fitness < threshold:
        return 0.0

    # Return the total fitness
    return fitness
```

Figure 1: hand_pose.gd GDScript Code

```gdscript
@tool
class_name HandPoseDetector
extends Node


## Hand Pose Detector Script
##
## This script checks for hand poses and reports them as events.


## Signal reported when a hand pose starts
signal pose_started(p_name : String)

## Signal reported when a hand pose ends
signal pose_ended(p_name : String)


@export_group("Hand", "hand_")

## Name of the hand pose tracker
@export var hand_tracker_name : String = "/user/hand_tracker/left"

## Current hand pose set
@export var hand_pose_set : HandPoseSet


## Current hand tracker
var hand_tracker : XRHandTracker


# Current hand pose data
var _current_data : HandPoseData = HandPoseData.new()

# Current hand pose
var _current_pose : HandPose

# Current pose hold
var _current_hold : float = 0.0

# New hand pose
var _new_pose : HandPose

# New pose hold
var _new_hold : float = 0.0


# Customize the properties
func _validate_property(property: Dictionary) -> void:
    if property.name == "hand_tracker_name":
        property.hint = PROPERTY_HINT_ENUM_SUGGESTION
        property.hint_string = "/user/hand_tracker/left,/user/hand_tracker/right"


# Called when the node enters the scene tree for the first time.
func _ready() -> void:
    # Listen for tracker changes
    XRServer.tracker_added.connect(_on_tracker_changed)
    XRServer.tracker_updated.connect(_on_tracker_changed)
    XRServer.tracker_removed.connect(_on_tracker_changed)


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta: float) -> void:
    # Skip when running in the engine
    if Engine.is_editor_hint():
        return

    # Skip if no tracker or hand pose set
    if not hand_tracker or not hand_pose_set:
        return

    # If the palm is not tracked then skip pose detection. Any current pose will
    # remain active until we see the hand again.
    var flags := hand_tracker.get_hand_joint_flags(XRHandTracker.HAND_JOINT_PALM)
    if (flags & XRHandTracker.HAND_JOINT_FLAG_POSITION_TRACKED) == 0:
        return;
    if (flags & XRHandTracker.HAND_JOINT_FLAG_ORIENTATION_TRACKED) == 0:
        return;

    # Save the active pose before updates (to report changes)
    var active_pos := _current_pose

    # Find the pose
    _current_data.update(hand_tracker)
    var pose := hand_pose_set.find_pose(_current_data)

    # Manage current pose
    if _current_pose:
        # Test if we detect the current pose
        if pose == _current_pose:
            # Restore hold on current pose
            _current_hold = 1.0
        else:
            # Decay hold on current pose
            _current_hold -= delta / _current_pose.release_time
            if _current_hold <= 0.0:
                # Current pose lost
                _current_hold = 0.0
                _current_pose = null

    # Handle ramp of new pose
    if pose != _new_pose:
        # New pose detected
        _new_pose = pose
        _new_hold = 0.0
    elif _new_pose:
        # Ramp hold on new pose
        _new_hold += delta / _new_pose.hold_time
        if _new_hold >= 1.0:
            # New pose "ready"
            _new_hold = 1.0
            if not _current_pose:
                # No current pose, transition to new pose
                _current_pose = _new_pose
                _current_hold = 1.0

    # Detect change in active pose
    if _current_pose != active_pos:
        # Report loss of old pose
        if active_pos:
            pose_ended.emit(active_pos.pose_name)

        # Report start of new pose
        active_pos = _current_pose
        if active_pos:
            pose_started.emit(active_pos.pose_name)


# Get configuration warnings
func _get_configuration_warnings() -> PackedStringArray:
    var warnings := PackedStringArray()

    # Verify hand tracker name is set
    if hand_tracker_name == "":
        warnings.append("Hand tracker name not set")

    # Return the warnings
    return warnings


# If the tracker changed then try to get the updated handle
func _on_tracker_changed(p_name : StringName, _type) -> void:
    if p_name == hand_tracker_name:
        hand_tracker = XRServer.get_tracker(hand_tracker_name)
```

Figure 2: hand_pose_detector.gd GDScript Code

```gdscript
extends Node

# Get References to Hand Pose Detectors
@onready var left_detector = $XROrigin3D/LeftTrackedHand/HandPoseDetector
@onready var right_detector = $XROrigin3D/RightTrackedHand/HandPoseDetector

# Reference to Label3D (SignText)
@onready var sign_text = $XROrigin3D/ISLWall/SignText

var left_letter := ""
var right_letter := ""

func _ready() -> void:
    # Connect signals from both detectors
    left_detector.pose_started.connect(_on_left_pose_detected)
    right_detector.pose_started.connect(_on_right_pose_detected)

# Signal Handlers
func _on_left_pose_detected(pose_name: String) -> void:
    # Debug Message:
    print("Left Pose: ", pose_name)
    left_letter = pose_to_letter(pose_name)
    update_display()

func _on_right_pose_detected(pose_name: String) -> void:
    # Debug Message:
    print("Right Pose: ", pose_name)
    right_letter = pose_to_letter(pose_name)
    update_display()

# Pose -> Letter Mapping
func pose_to_letter(pose_name: String) -> String:
    # Debug message
    print("pose_to_letter() called with: ", pose_name)
    match pose_name:
        "Fist":
            return "A"
        "Peace Sign":
            return "V"
        "Point":
            return "D"
        _:
            return ""

# Update Text on Label
func update_display() -> void:
    sign_text.text = "Left hand = %s\nRight hand = %s" % [
        left_letter,
        right_letter
    ]
```

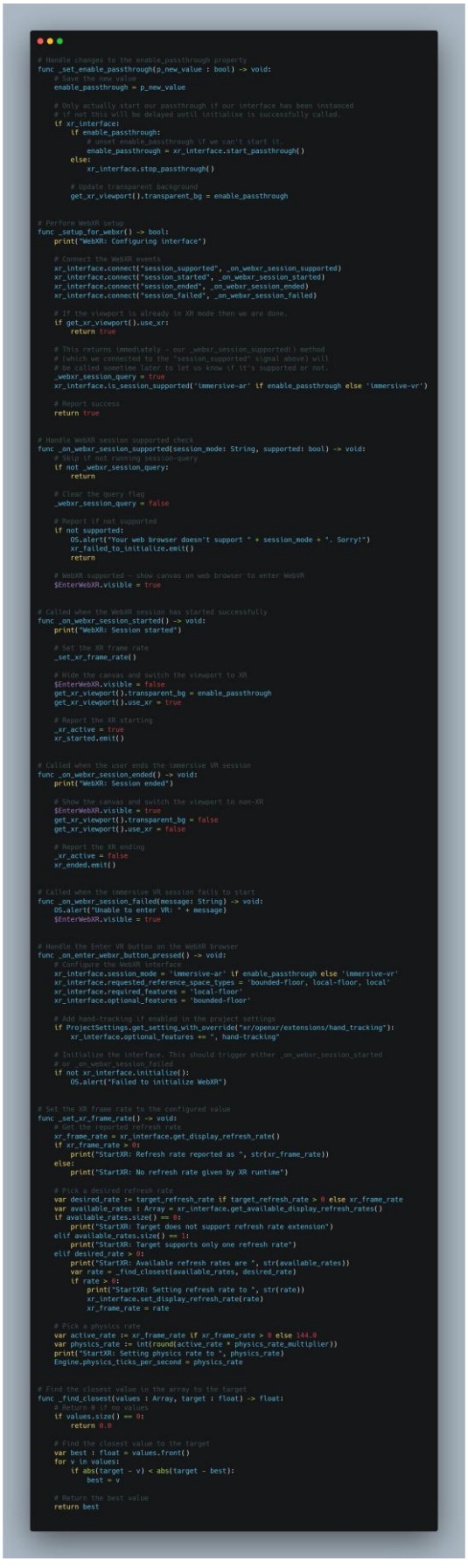
Figure 3: sign_feedback.gd GDScript Code

```gdscript
@tool
class_name XRToolsStartXR
extends Node


## XRTools Start XR Class
##
## This class supports both the OpenXR and WebXR interfaces, and handles
## the initialization of the interface as well as reporting when the user
## starts and ends the VR session.
##
## For OpenXR this class also supports passthrough on compatible devices.


## This signal is emitted when XR becomes active. For OpenXR this corresponds
## with the 'openxr_focused_state' signal which occurs when the application
## starts receiving XR input, and for WebXR this corresponds with the
## 'session_started' signal.
signal xr_started

## This signal is emitted when XR ends. For OpenXR this corresponds with the
## 'openxr_visible_state' state which occurs when the application has lost
## XR input focus, and for WebXR this corresponds with the 'session_ended'
## signal.
signal xr_ended

## This signal is emitted if XR fails to initialize.
signal xr_failed_to_initialize


## XR active flag
static var _xr_active : bool = false


## Optional viewport to control
@export var viewport : Viewport

## Adjusts the pixel density on the rendering target
@export var render_target_size_multiplier : float = 1.0

## If true, the XR passthrough is enabled (OpenXR only)
@export var enable_passthrough : bool = false: set = _set_enable_passthrough

## Physics rate multiplier compared to HMD frame rate
@export var physics_rate_multiplier : int = 1

## If non-zero, specifies the target refresh rate
@export var target_refresh_rate : float = 0


## Current XR interface
var xr_interface : XRInterface

## XR frame rate
var xr_frame_rate : float = 0

# Is a WebXR is_session_supported query running
var _webxr_session_query : bool = false


# Handle auto-initialization when ready
func _ready() -> void:
    if !Engine.is_editor_hint():
        _initialize()


## Initialize the XR interface
func _initialize() -> bool:
    # Check for OpenXR interface
    xr_interface = XRServer.find_interface('OpenXR')
    if xr_interface:
        return _setup_for_openxr()

    # Check for WebXR interface
    xr_interface = XRServer.find_interface('WebXR')
    if xr_interface:
        return _setup_for_webxr()

    # No XR interface
    xr_interface = null
    print("No XR interface detected")
    xr_failed_to_initialize.emit()
    return false


## End the XR experience
func end_xr() -> void:
    # For WebXR drop the interactive experience and go back to the web page
    if xr_interface is WebXRInterface:
        # Uninitialize the WebXR interface
        xr_interface.uninitialize()
        return

    # Terminate the application
    get_tree().quit()
```

Figure 4: start_xr.gd GDScript Code

```gdscript
## Test if XR is active
static func is_xr_active() -> bool:
    return _xr_active


## Get the XR viewport
func get_xr_viewport() -> Viewport:
    # Use the specified viewport if set
    if viewport:
        return viewport

    # Use the default viewport
    return get_viewport()


# Check for configuration issues
func _get_configuration_warnings() -> PackedStringArray:
    var warnings := PackedStringArray()

    if physics_rate_multiplier < 1:
        warnings.append("Physics rate multiplier should be at least 1x the HMD rate")

    return warnings


# Perform OpenXR setup
func _setup_for_openxr() -> bool:
    print("OpenXR: Configuring interface")

    # Set the render target size multiplier
    xr_interface.render_target_size_multiplier = render_target_size_multiplier

    # Initialize the OpenXR interface
    if not xr_interface.is_initialized():
        print("OpenXR: Initializing interface")
        if not xr_interface.initialize():
            push_error("OpenXR: Failed to initialize")
            xr_failed_to_initialize.emit()
            return false

    # Connect the OpenXR events
    xr_interface.connect("session_begun", _on_openxr_session_begun)
    xr_interface.connect("session_visible", _on_openxr_visible_state)
    xr_interface.connect("session_focussed", _on_openxr_focused_state)

    # Check for passthrough
    if enable_passthrough and xr_interface.is_passthrough_supported():
        enable_passthrough = xr_interface.start_passthrough()

    # Disable vsync
    DisplayServer.window_set_vsync_mode(DisplayServer.VSYNC_DISABLED)

    # Switch the viewport to XR
    get_xr_viewport().transparent_bg = enable_passthrough
    get_xr_viewport().use_xr = true

    # Report success
    return true


# Handle OpenXR session ready
func _on_openxr_session_begun() -> void:
    print("OpenXR: Session begun")

    # Set the XR frame rate
    _set_xr_frame_rate()


# Handle OpenXR visible state
func _on_openxr_visible_state() -> void:
    # Report the XR ending
    if _xr_active:
        print("OpenXR: XR ended (visible_state)")
        _xr_active = false
        xr_ended.emit()


# Handle OpenXR focused state
func _on_openxr_focused_state() -> void:
    # Report the XR starting
    if not _xr_active:
        print("OpenXR: XR started (focused_state)")
        _xr_active = true
        xr_started.emit()
```

Figure 4.1: start_xr.gd GDScript Code



```
# Handle changes to the enable_passthrough property
func _set_enable_passthrough(p_new_value : bool) -> void:
    # Save the new value
    enable_passthrough = p_new_value

    # Only actually start our passthrough if our interface has been instanced
    # if not this will be delayed until initialise is successfully called.
    if xr_interface:
        if enable_passthrough:
            # we can't enable passthrough if we can't start it...
            enable_passthrough = xr_interface.start_passthrough()
        else:
            xr_interface.stop_passthrough()

        # Update transparent background
        get_xr_viewport().transparent_bg = enable_passthrough


# Perform WebXR setup
func _setup_for_webxr() -> bool:
    print("WebXR: Configuring interface")

    # Connect the WebXR events
    xr_interface.connect("session_supported", _on_webxr_session_supported)
    xr_interface.connect("session_started", _on_webxr_session_started)
    xr_interface.connect("session_ended", _on_webxr_session_ended)
    xr_interface.connect("session_failed", _on_webxr_session_failed)

    # If the viewport is already in XR mode then we are done.
    if get_xr_viewport().use_xr:
        return true

    # This returns immediately - our _webxr_session_supported() method
    # (which we connected to the "session_supported" signal above) will
    # be called sometime later to let us know if it's supported or not.
    _webxr_session_query = true
    xr_interface.is_session_supported('immersive-ar' if enable_passthrough else 'immersive-vr')

    # Report success
    return true


# Handle WebXR session supported check
func _on_webxr_session_supported(session_mode: String, supported: bool) -> void:
    # Skip if not running session query
    if not _webxr_session_query:
        return

    # Clear the query flag
    _webxr_session_query = false

    # Report if not supported
    if not supported:
        OS.alert("Your web browser doesn't support " + session_mode + ". Sorry!")
        xr_failed_to_initialize.emit()
        return

    # WebXR supported - show canvas on web browser to enter WebVR
    $EnterWebXR.visible = true


# Called when the WebXR session has started successfully
func _on_webxr_session_started() -> void:
    print("WebXR: Session started")

    # Set the XR frame rate
    _set_xr_frame_rate()

    # Hide the canvas and switch the viewport to XR
    $EnterWebXR.visible = false
    get_xr_viewport().transparent_bg = enable_passthrough
    get_xr_viewport().use_xr = true

    # Report the XR starting
    _xr_active = true
    xr_started.emit()


# Called when the user ends the immersive VR session
func _on_webxr_session_ended() -> void:
    print("WebXR: Session ended")

    # Show the canvas and switch the viewport to non-XR
    $EnterWebXR.visible = true
    get_xr_viewport().transparent_bg = false
    get_xr_viewport().use_xr = false

    # Report the XR ending
    _xr_active = false
    xr_ended.emit()


# Called when the immersive VR session fails to start
func _on_webxr_session_failed(message: String) -> void:
    OS.alert("Unable to enter VR: " + message)
    $EnterWebXR.visible = true


# Handle the Enter VR button on the WebXR browser
func _on_enter_webxr_button_pressed() -> void:
    # Configure the WebXR interface
    xr_interface.session_mode = 'immersive-ar' if enable_passthrough else 'immersive-vr'
    xr_interface.requested_reference_space_types = 'bounded-floor, local-floor, local'
    xr_interface.required_features = 'local-floor'
    xr_interface.optional_features = 'bounded-floor'

    # Add hand-tracking if enabled in the project settings
    if ProjectSettings.get_setting_with_override("xr/openxr/extensions/hand_tracking"):
        xr_interface.optional_features += ", hand-tracking"

    # Initialize the interface. This should trigger either _on_webxr_session_started
    # or _on_webxr_session_failed
    if not xr_interface.initialize():
        OS.alert("Failed to initialize WebXR")


# Set the XR frame rate to the configured value
func _set_xr_frame_rate() -> void:
    # Get the reported refresh rate
    xr_frame_rate = xr_interface.get_display_refresh_rate()
    if xr_frame_rate > 0:
        print("StartXR: Refresh rate reported as ", str(xr_frame_rate))
    else:
        print("StartXR: No refresh rate given by XR runtime")

    # Pick a desired refresh rate
    var desired_rate := target_refresh_rate if target_refresh_rate > 0 else xr_frame_rate
    var available_rates : Array = xr_interface.get_available_display_refresh_rates()
    if available_rates.size() == 0:
        print("StartXR: Target does not support refresh rate extension")
    elif available_rates.size() == 1:
        print("StartXR: Target supports only one refresh rate")
    elif desired_rate > 0:
        print("StartXR: Available refresh rates are ", str(available_rates))
        var rate = _find_closest(available_rates, desired_rate)
        if rate > 0:
            print("StartXR: Setting refresh rate to ", str(rate))
            xr_interface.set_display_refresh_rate(rate)
            xr_frame_rate = rate

    # Pick a physics rate
    var active_rate := xr_frame_rate if xr_frame_rate > 0 else 144.0
    var physics_rate := int(round(active_rate * physics_rate_multiplier))
    print("StartXR: Setting physics rate to ", physics_rate)
    Engine.physics_ticks_per_second = physics_rate


# Find the closest value in the array to the target
func _find_closest(values : Array, target : float) -> float:
    # Return 0 if no values
    if values.size() == 0:
        return 0.0

    # Find the closest value to the target
    var best : float = values.front()
    for v in values:
        if abs(target - v) < abs(target - best):
            best = v

    # Return the best value
    return best
```

Figure 4.2: start_xr.gd GDScript Code

Screenshots

Tests

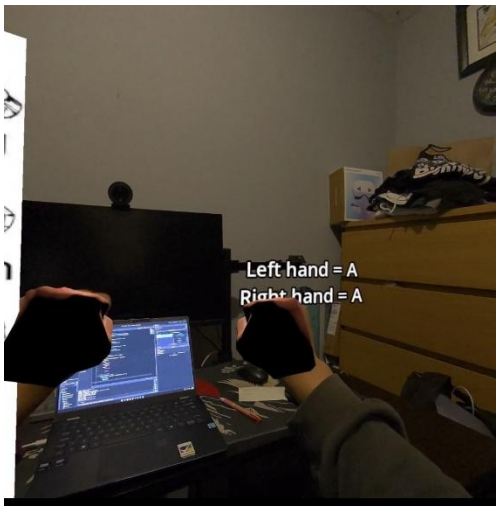


Figure 5: Test For Signing "A"

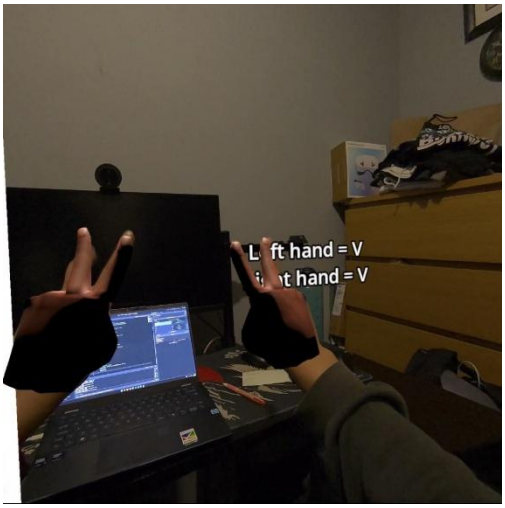


Figure 5.1: Test For Signing "V"

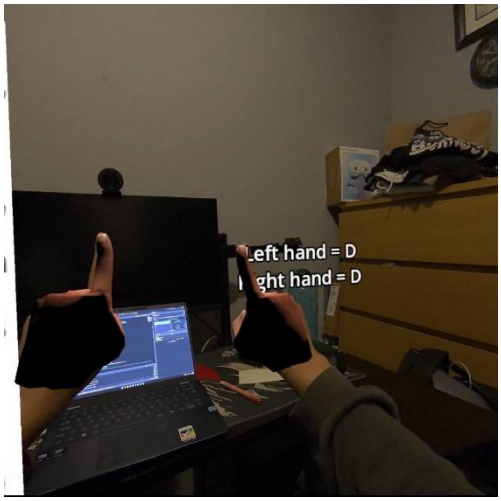


Figure 5.2: Test For Signing "D"

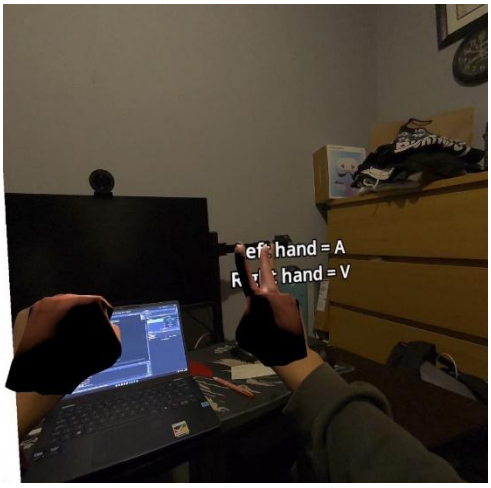


Figure 5.3: Test For Signing "A" and "V" on hands

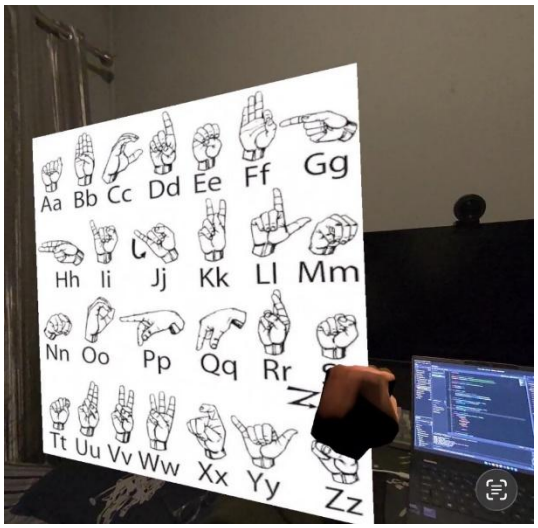


Figure 5.3: Irish Sign Language Wall on Environment

Figure 6: Supervisor Testing Prototype

Figure 7: User Testing Prototype

Hierarchy

Figure 8: Current Hierarchy for Prototype

Config

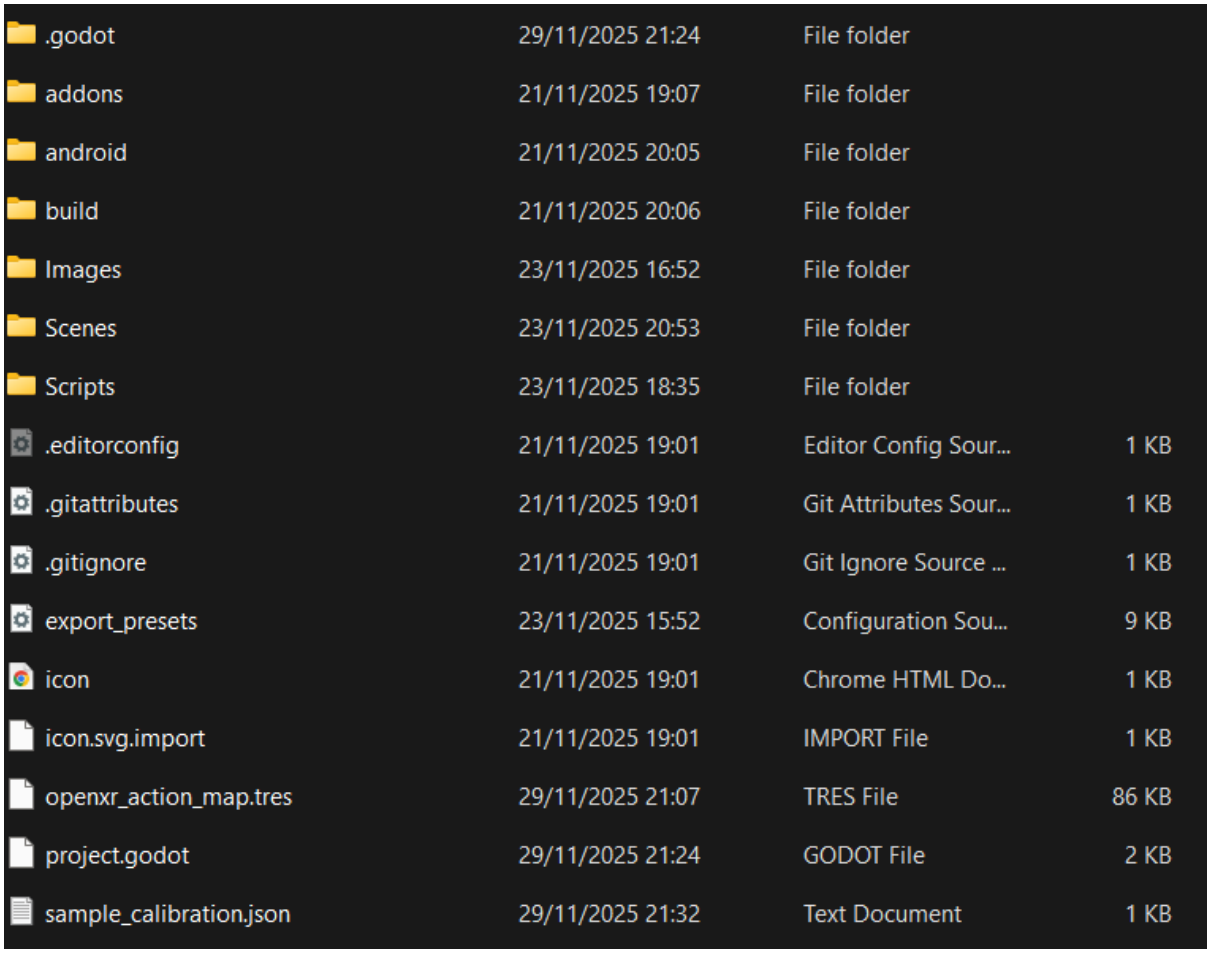


Figure 9: Config for Sign & Spell VR