

Quest: SDG

Technical Implementation in Godot Engine

TU Dublin Extended Reality Prototyping - Computer Science Module

Project Architecture Overview

Quest: SDG is a VR application built in Godot 4 with the following architecture:

- **Scene Structure:** Hierarchical node system with specialized scenes
- **Interaction System:** Area3D-based interaction with hand tracking
- **Content Management:** Resource-based approach for SDG content
- **XR Framework:** OpenXR integration via Godot's XR modules

Scene Hierarchy

```
node_3d.tscn (Root)
└── XROrigin3D
    ├── XRCamera3D
    │   ├── left (Hand controller)
    │   └── right (Hand controller)
    └── color_wheel
    └── goals (Container for SDG boxes)
        └── goal_box (x17)
    └── ani_goals (Container for animated boxes)
        └── goal_box_animated (x17)
└── WorldEnvironment
```

Core Components Implementation

1. **Color Wheel:** Entry point for user interaction
2. **Goal Boxes:** Static representations of each SDG
3. **Animated Goal Boxes:** Interactive presentations of each SDG
4. **XR Controllers:** Hand tracking and interaction
5. **Audio System:** Narration and sound effects

Color Wheel Implementation

```
# color_wheel.gd
extends Area3D

@export var image:Texture
var inside:bool = false

func _on_area_entered(area: Area3D) -> void:
    if area.name.contains("hand"):
        play_sound()
        hand = area.get_parent()
        var t = create_tween() \
            .set_ease(Tween.EASE_IN_OUT) \
            .set_trans(Tween.TRANS_QUINT)
        scale = Vector3.ONE
        t.tween_property(self, "scale", big_scale, 1)
        inside = true

    func fade_out():
        # Animation code to shrink wheel
        fade_out_tween = create_tween()
        .set_trans(Tween.TRANS_QUINT)
        .set_ease(Tween.EASE_IN_OUT)
        fade_out_tween.tween_property(self, "scale", Vector3.ZERO, 2)
        fade_out_tween.finished.connect(make_invisible)

    func make_invisible():
        $"../goals".spawn_boxes() # Spawn SDG boxes
        deactivate()
```

Area3D Interaction System

The project uses Godot's Area3D nodes for interaction:

```
# Common pattern in interactive objects
func _on_area_entered(area: Area3D) -> void:
    if area.name.contains("hand"):
        # Handle hand entering interaction zone
        play_sound()
        hand = area.get_parent()
        scale_up_animation()
        inside = true

func _process(delta: float) -> void:
    # Check for selection while hand is inside
    if inside && hand.selected:
        handle_selection()
```

Tween Animation System

Smooth animations are implemented using Godot's Tween system:

```
# Example from goal_box.gd
func scale_up_animation():
    var t = create_tween() \
        .set_ease(Tween.EASE_IN_OUT) \
        .set_trans(Tween.TRANS_QUINT)
    scale = Vector3.ONE
    t.tween_property(self, "scale", big_scale, 1)

func fade_out():
    fade_out_tween = create_tween()
        .set_trans(Tween.TRANS_QUINT)
        .set_ease(Tween.EASE_IN_OUT)
    fade_out_tween.tween_property(self, "scale", Vector3.ZERO, 2)
    fade_out_tween.finished.connect(make_invisible)
```

Goal Box Implementation

```
# goal_box.gd
extends Area3D

@export var image:Texture
@export var goal:int = 1 # which SDG this represents
var inside:bool = false
var mats = []

func make_invisible():
    monitoring = false
    monitorable = false

    # Get the animated version of this box
    ani_box = get_parent().ani_boxes[goal - 1]
    ani_box.position = position
    ani_box.rotation = rotation
    ani_box.visible = true
    ani_box.bounce_in()

    # Analytics tracking
    Talo.events.track("Goal " + str(goal) + " thumbs up")
    inside = false
    deactivate()
```

3D Object Construction

Each interactive object is constructed as a cube with six faces:

```
# From goal_box.gd and color_wheel.gd
func set_texture(mesh:MeshInstance3D):
    var mat:StandardMaterial3D = mesh.get_surface_override_material(0)
    mat = mat.duplicate()
    mat.albedo_texture = image
    mesh.set_surface_override_material(0, mat)
    mats.push_back(mat)

func _ready() -> void:
    set_texture($scaler/front)
    set_texture($scaler/back)
    set_texture($scaler/left)
    set_texture($scaler/right)
    set_texture($scaler/top)
    set_texture($scaler/bot)
```

Goal Box Scene Structure

```
goal_box (Area3D)
└─ CollisionShape3D (BoxShape3D)
└─ scaler (Node3D)
    └─ front (MeshInstance3D)
    └─ back (MeshInstance3D)
    └─ left (MeshInstance3D)
    └─ right (MeshInstance3D)
    └─ top (MeshInstance3D)
    └─ bot (MeshInstance3D)
└─ AudioStreamPlayer3D
```

Goals Manager Implementation

```
# goals.gd
extends Node3D

var ani_boxes = []
var goal_boxes = []

func spawn_box(i):
    var t = create_tween() \
        .set_trans(Tween.TRANS_QUINT).set_ease(Tween.EASE_IN_OUT)
    var box = goal_boxes[i]
    box.scale = Vector3.ZERO
    box.visible = true
    box.monitoring = true
    # Play sound and animate appearance
    box.get_node("AudioStreamPlayer3D").play()
    t.tween_property(box, "scale", Vector3.ONE, interval)

func spawn_boxes():
    for i in 18:
        spawn_box(i)
        await get_tree().create_timer(0.6).timeout
```

Grid Layout Implementation

```
# goals.gd
func reset_positions():
    var cols = 6
    var gap = 0.3
    var row = 0
    var col = 0

    goal_boxes.clear()
    for child:Node3D in get_children():
        goal_boxes.push_back(child)
        child.visible = false
        # Position in a grid layout
        child.position = Vector3(col * gap, row * gap, 0)
        col += 1
        if col == 6:
            col = 0
            row = row - 1
        child.monitoring = false
        child.monitorable = false
```

Animated Goal Box Implementation

```
# goal_box_animated.gd
extends RigidBody3D

var sprites = []
var mats = []
var anim0Frames:SpriteFrames
var anim1Frames:SpriteFrames

func bounce_in():
    $Area3D.monitoring = true
    # Start animations on all faces
    $Area3D/scaler/front.play("default")
    $Area3D/scaler/bott.play("default")
    $Area3D/scaler/left.play("default")
    $Area3D/scaler/top.play("default")
    $Area3D/scaler/back.play("default")
    $Area3D/scaler/right.play("default")

    # Scale animation and audio fade-in
    scale = Vector3.ZERO
    fade_tween = create_tween()
        .set_trans(Tween.TRANS_QUINT)
        .set_ease(Tween.EASE_IN_OUT)
    fade_tween.tween_property(self, "scale", Vector3.ONE, 2)
    $playlist_looper.play()
```

Animated Goal Box Scene Structure

```
goal_box_animated (RigidBody3D)
└─ Area3D
    └─ CollisionShape3D
        └─ scaler (Node3D)
            └─ front (AnimatedSprite3D)
            └─ back (AnimatedSprite3D)
            └─ left (AnimatedSprite3D)
            └─ right (AnimatedSprite3D)
            └─ top (AnimatedSprite3D)
            └─ bott (AnimatedSprite3D)
└─ playlist_looper (Node3D)
    └─ AudioStreamPlayer3D
```

Audio Playlist System

```
# playlist_looper.gd
extends Node3D

@export var sounds:Array[AudioStream]
var i:int = 0
@onready var player = $AudioStreamPlayer3D

func _ready() -> void:
    # Connect to finished signal for auto-progression
    player.finished.connect(next)
    if auto_play:
        play()

func next():
    i = (i + 1) % sounds.size()
    play()

func play():
    player.stream = sounds[i]
    player.play()
```

XR Initialization

```
# start_xr.gd
extends Node

var xr_interface: XRInterface

func _ready() -> void:
    xr_interface = XRServer.primary_interface
    if xr_interface and xr_interface.is_initialized():
        print("OpenXR initialised successfully")

    # Turn off v-sync for performance
    DisplayServer.window_set_vsync_mode(
        DisplayServer.VSYNC_DISABLED)

    # Configure viewport for XR
    get_viewport().use_xr = true
    enable_passthrough()
```

Passthrough Implementation

```
# start_xr.gd
func enable_passthrough() -> bool:
    if xr_interface and xr_interface.is_passthrough_supported():
        return xr_interface.start_passthrough()
    else:
        var modes = xr_interface.get_supported_environment_blend_modes()
        if xr_interface.XR_ENV_BLEND_MODE_ALPHA_BLEND in modes:
            xr_interface.set_environment_blend_mode(
                xr_interface.XR_ENV_BLEND_MODE_ALPHA_BLEND)
            return true
        else:
            return false
```

Hand Controller Implementation

```
# hand.gd
extends Node3D

var selected:bool = false

func _process(delta: float) -> void:
    # Check for controller button press
    if controller:
        if controller.is_button_pressed("trigger_click") or \
            controller.is_button_pressed("grip_click"):
            selected = true
        else:
            selected = false

func _on_button_pressed(button_name: String) -> void:
    if button_name == "trigger_click" or button_name == "grip_click":
        selected = true

func _on_button_released(button_name: String) -> void:
    if button_name == "trigger_click" or button_name == "grip_click":
        selected = false
```

Analytics Implementation

```
# Used throughout the codebase
# Example from goal_box.gd
func make_invisible():
    # Track which SDG was selected
    Talo.events.track("Goal " + str(goal) + " thumbs up")
    Talo.events.flush()

# Example from color_wheel.gd
func make_invisible():
    # Track when SDGs are activated
    Talo.events.track("SDG's Activated")
```

Resource Management

The project organizes resources by type and SDG:

```
goals/
└── E-WEB-Goal-01.png      # SDG 1 image
└── E-WEB-Goal-02.png      # SDG 2 image
...
└── E-WEB-Goal-17.png      # SDG 17 image
└── Goal-1/                  # SDG 1 specific resources
└── Goal-2/                  # SDG 2 specific resources
...
└── Goal-17/                 # SDG 17 specific resources

Voices/
└── sdg 1 emily.mp3        # SDG 1 narration
└── sdg 2 emily.mp3        # SDG 2 narration
...
└── sdg 17.mp3              # SDG 17 narration
```

Signal-Based Communication

The project uses Godot's signal system for event-driven communication:

```
# In goal_box_animated.gd
signal bounce # Emitted when animation starts

# In playlist_looper.gd
func _ready() -> void:
    player.finished.connect(next) # Connect to audio finished signal

# In color_wheel.gd
func fade_out():
    fade_out_tween.finished.connect(make_invisible)
```

Technical Challenges and Solutions

1. Performance Optimization

- Using simplified collision shapes
- Disabling v-sync for VR performance
- Efficient resource management

2. XR Integration

- OpenXR framework for compatibility
- Passthrough implementation for mixed reality
- Hand tracking integration

3. Content Management

- Resource-based approach for scalability

Godot-Specific Implementation Details

1. Node Types Used:

- Area3D for interaction zones
- RigidBody3D for physics-based objects
- AnimatedSprite3D for animated textures
- AudioStreamPlayer3D for spatial audio

2. Resource Types:

- Textures for SDG images
- AudioStreams for narration
- SpriteFrames for animations
- StandardMaterial3D for rendering

Scene Instantiation and Management

```
# Example of scene management
func _ready() -> void:
    # Initialize arrays
    ani_boxes.clear()
    for child:Node3D in $"../ani_goals".get_children():
        child.visible = false
        ani_boxes.push_back(child)

# Dynamic instantiation (commented out in start_xr.gd)
# var my_scene = load("res://Cloud.tscn")
# var instance = my_scene.instantiate()
# add_child(instance)
# instance.position = Vector3(2.628, 1.5, -0.314)
```

Project Export Configuration

```
# From project.godot and export_presets.cfg
# XR configuration
xr_mode = "on"
xr_features/hand_tracking = 1
xr_features/hand_tracking_frequency = 0
xr_features/passthrough = 1

# Android/Quest export settings
architectures/arm64-v8a = true
permissions/hand_tracking = true
xr_features/hand_tracking = 1
xr_features/passthrough = 1
```

Testing and Debugging Techniques

1. XR Simulator

- Using addons/xr-simulator for desktop testing
- Simulating hand controllers without headset

2. Debug Drawing

- Using addons/debug_draw_3d for visual debugging
- Visualizing interaction zones and paths

3. Analytics

- Using Talo for event tracking and analysis
- Monitoring user interactions with SDGs

Performance Considerations

1. Rendering Optimization

- Simple materials and textures
- Limited use of dynamic lighting
- Optimized for mobile VR hardware

2. Physics Optimization

- Simplified collision shapes
- Limited use of Rigidbody3D nodes
- Static objects where possible

3. Audio Optimization

- Spatial audio for immersion

Code Architecture Patterns

1. Component-Based Design

- Each functionality in separate scripts
- Reusable components (e.g., playlist_looper)

2. Event-Driven Programming

- Signal-based communication
- Decoupled components

3. Resource Management

- Centralized resource loading
- Organized by SDG for maintainability

Deployment Process

1. Build Preparation

```
# Install Android SDK and setup Godot export templates  
godot --export-debug "Android" ./build/questsdg.apk
```

2. Sideload

```
# Using ADB to install on Quest  
adb install -r ./build/questsdg.apk
```

3. Distribution via App Lab

- Upload build to Meta Developer Dashboard
- Configure App Lab listing
- Submit for review

Future Technical Enhancements

1. Multiplayer Implementation

- Using Godot's NetworkedMultiplayerENet
- Synchronized SDG exploration

2. Advanced Interaction

- Gesture recognition for more natural interaction
- Physics-based manipulation of objects

3. Content Management System

- Dynamic loading of SDG content
- Web-based content updates

Thank You!

Quest: SDG - Technical Implementation in Godot Engine

For more information:

- [Godot Documentation](#)
- [OpenXR Documentation](#)
- [UN Sustainable Development Goals](#)