XR Sequencer

A Virtual Reality Music Sequencer in Godot

Project Overview

XR Sequencer is a virtual reality music sequencer built in Godot that allows users to:

- Create musical patterns in 3D space
- Interact with sequencer pads using VR controllers
- Play multiple instrument samples simultaneously
- Experience music creation in mixed reality (passthrough)

Technical Architecture

- **Engine**: Godot 4.x
- **XR Framework**: OpenXR
- **Interaction**: Area3D collision detection
- **Audio**: Multiple AudioStreamPlayer nodes
- **UI**: 3D grid of interactive pads

Scene Structure

```
root (Node3D)
├── start_passthrough (Node)
├── DirectionalLight3D
├── StaticBody3D (Ground)
├── WorldEnvironment
├── player (CharacterBody3D)
│   └── XROrigin (XROrigin3D)
│       ├── XRCamera3D
│       ├── left (XRController3D)
│       └── right (XRController3D)
├── start_stop (Area3D)
├── sequencer (Marker3D)
│   ├── Timer
│   └── timer_ball (MeshInstance3D)
└── sequencer2 (Marker3D)
    ├── Timer
    └── timer_ball (MeshInstance3D)
```

Sequencer Component

The core of the project is the `sequencer.gd` script, which:

1. Loads audio samples from a specified directory
2. Creates a grid of interactive pads
3. Maintains a 2D array representing the sequence
4. Steps through the sequence and plays active samples
5. Provides visual feedback with a moving "timer ball"

Sequencer Implementation

```
# Key parts of sequencer.gd
extends Marker3D

var samples:Array  # Loaded audio samples
var sequence = []  # 2D grid of booleans (active/inactive)
@export var steps = 8  # Number of steps in sequence

func initialise_sequence(rows, cols):
    for i in range(rows):
        var row = []
        for j in range(cols):
            row.append(false)
        sequence.append(row)
    self.rows = rows
    self.cols = cols

func toggle(e, row, col):
    sequence[row][col] = !sequence[row][col]
    play_sample(0, row)  # Play for feedback
```

Sequencer Grid Generation

```
# From sequencer.gd
func make_sequencer():
    for col in range(steps):
        for row in range(samples.size()):
            var pad = pad_scene.instantiate()

            var p = Vector3(s * col * spacer, s * row * spacer, 0)
            pad.position = p
            pad.rotation = rotation
            pad.area_entered.connect(toggle.bind(row, col))
            add_child(pad)
```

This creates a grid of pads in 3D space, with each pad connected to the toggle function.

Sequencer Playback

```
# From sequencer.gd
func _on_timer_timeout() -> void:
    play_step(step)
    step = (step + 1) % steps

func play_step(col):
    var p = Vector3(s * col * spacer, s * -1 * spacer, 0)
    $timer_ball.position = p
    for row in range(rows):
        if sequence[row][col]:
            play_sample(0, row)

func _on_start_stop_area_entered(area: Area3D) -> void:
    if $Timer.is_stopped():
        $Timer.start()
    else:
        $Timer.stop()
```

Pad Component

Each pad in the sequencer grid is an instance of `pad_a.tscn`:

```
Pad (Area3D)
├── MeshInstance3D (BoxMesh)
└── CollisionShape3D (SphereShape3D)
```

The pad changes color when toggled, providing visual feedback to the user.

Pad Implementation

```
# pad_a.gd
extends Area3D

var mat:StandardMaterial3D
@export var out_color:Color  # Color when inactive
@export var in_color:Color   # Color when active
var toggle:bool = false

func _ready() -> void:
    mat = StandardMaterial3D.new()
    $MeshInstance3D.set_surface_override_material(0, mat)
    mat.albedo_color = out_color
    mat.transparency = BaseMaterial3D.TRANSPARENCY_ALPHA

func _on_area_entered(area: Area3D) -> void:
    toggle = !toggle
    set_mat()
```

XR Implementation

The project uses OpenXR for VR functionality:

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

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

        # Change our main viewport to output to the HMD
        get_viewport().use_xr = true
        enable_passthrough()
```

Passthrough Implementation

The project supports mixed reality through passthrough:

```
# From start_passthrough.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
```

Audio System

The sequencer uses a pool of AudioStreamPlayer nodes to play samples:

```
# From sequencer.gd
func _ready():
    load_samples()
    initialise_sequence(samples.size(), steps)
    make_sequencer()

    # Create a pool of audio players
    for i in range(50):
        var asp = AudioStreamPlayer.new()
        add_child(asp)
        players.push_back(asp)

func play_sample(e, i):
    var p:AudioStream = samples[i]
    var asp = players[asp_index]
    asp.stream = p
    asp.play()
```

Sample Loading

The sequencer dynamically loads audio samples from a specified directory:

```
# From sequencer.gd
func load_samples():
    var dir = DirAccess.open(path_str)
    if dir:
        dir.list_dir_begin()
        var file_name = dir.get_next()

        while file_name != "":
            if file_name.ends_with('.wav.import') or file_name.ends_with('.mp3.import'):
                file_name = file_name.left(len(file_name) - len('.import'))
                var stream = load(path_str + "/" + file_name)
                stream.resource_name = file_name
                samples.push_back(stream)
                file_names.push_back(file_name)
            file_name = dir.get_next()
```

VR Interaction

The project uses Area3D nodes and collision detection for VR interaction:

1. VR controllers are represented by Area3D nodes with small collision shapes
2. When a controller enters a pad's area, it triggers the toggle function
3. The sequencer updates its internal state and provides visual/audio feedback
4. A special start_stop Area3D toggles playback when touched

Multiple Sequencers

The main scene includes two separate sequencer instances:

```
# From music_toys.tscn
[node name="sequencer" type="Marker3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 2.5, 0)
script = ExtResource("16_6ue7p")
font = ExtResource("7_150ad")
pad_scene = ExtResource("17_pakd3")
steps = 16

[node name="sequencer2" type="Marker3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, -2.2667618, 2.5, -0.16273975)
script = ExtResource("16_6ue7p")
font = ExtResource("7_150ad")
path_str = "res://piano"  # Different sample set
pad_scene = ExtResource("17_pakd3")
steps = 16
```

Godot Systems Used

1. **XR Framework**:
    - XROrigin3D, XRCamera3D, XRController3D
    - OpenXR interface for device compatibility
2. **Physics & Interaction**:
    - Area3D nodes with collision detection
    - Signals for event handling
3. **Audio**:
    - AudioStreamPlayer nodes
    - Dynamic resource loading
4. **Materials & Visuals**:

Workflow for Creating Music

1. **Setup**: Put on VR headset and start the application
2. **Exploration**: Observe the two sequencer grids in 3D space
3. **Pattern Creation**: Touch pads with VR controllers to toggle them on/off
4. **Playback**: Touch the start_stop cube to begin playback
5. **Iteration**: Continue modifying the pattern while it plays
6. **Multi-Instrument**: Use both sequencers to create layered patterns

Key Design Patterns

1. **Object Pooling**:
    - Pre-creates 50 AudioStreamPlayer nodes for efficient playback
2. **Event-Driven Programming**:
    - Uses signals and callbacks for interaction
3. **Resource Management**:
    - Dynamically loads audio samples at runtime
4. **Component-Based Design**:
    - Separates functionality into reusable components (sequencer, pad)

Performance Considerations

1. **V-Sync Disabled**:
    - `DisplayServer.window_set_vsync_mode(DisplayServer.VSYNC_DISABLED)`
    - Improves performance in VR
2. **Audio Player Pooling**:
    - Reuses AudioStreamPlayer nodes instead of creating new ones
3. **Simple Visuals**:
    - Uses basic shapes and materials for better performance
4. **Efficient Collision Detection**:
    - Small, simple collision shapes for interaction

Potential Enhancements

1. **Visual Enhancements**:
    - More elaborate visual feedback during playback
    - 3D visualizations of audio waveforms
2. **Additional Controls**:
    - Volume and tempo controls
    - Effects processing (reverb, delay, etc.)
3. **Pattern Management**:
    - Save/load functionality for patterns
    - Pattern copying and variation
4. **Expanded Interaction**:

Conclusion

The XR Sequencer demonstrates:

- Effective use of Godot's XR capabilities
- Simple but powerful music creation in VR
- Intuitive 3D interaction design
- Efficient audio handling
- Mixed reality integration

This project serves as an excellent foundation for more complex XR music applications.

Thank You!

XR Sequencer - A Virtual Reality Music Sequencer in Godot

For more information:

- Godot XR Documentation
- OpenXR Specification