電車運転ゲームを作りたい

October 1, 2025

Chapter 1

仮線路と仮電車を作って走らせる

1.1 エラーメッセージのコピー

下のエラーメッセージの右側の⊠ 8 とかのとこを教えてエラーメッセイージ一覧を選択できる。また、カラフルなエラーメッセージを右クリックよりコピーを選択できる。あと赤いエラーメッセージも右側のコピーボタンでコピーできる。

```
Main (Node3D)
├── Path3D
│     └── PathFollow3D
│          └── MeshInstance3D (電車のCube)
└── Camera3D (固定カメラ)
```

MeshInstance3D を作成 -> 子ノードを追加 -> インスペクターより、<空> -> MeshBox

1.1.1 Path3D を選択して

右側の Inspector よりカーブ (Curve) を探し、新規 Curve3D を作成。
CTRL(Command) + 左クリックで点を打ってカーブを作る。真上、真下から線路に打つと良い。消すときは右クリック
また、Curve の点はクリックして青赤緑の矢印で移動させられる。
また、Curve に限らず、オブジェクト、Curve を微調整したいときは Shif+青赤緑矢印ドラッグで微調整できる。

1.1.2 PathFollow3D を選択して

右クリックよりスクリプトをアタッチ

```
  extends PathFollow3D

var speed := 0.0
var acceleration := 10.0
var max_speed := 30.0

func _process(delta: float) -> void:
    if Input.is_action_pressed("ui_up"):
        speed += acceleration * delta
    elif Input.is_action_pressed("ui_down"):
        speed -= acceleration * delta
    else:
        speed *= 0.98

    speed = clamp(speed, 0.0, max_speed)

    var curve = get_parent().curve
    if curve:
        var path_length = curve.get_baked_length()
        # ここがポイント!
        progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)
```

実行して、上下キーで動かす。

1.2 画面に速度計をつける

```
  Main (Node3D)
├── Path3D
│     └── PathFollow3D
```

```
│          └── MeshInstance3D（電車）
├── Camera3D（固定カメラ）
└──CanvasLayer 名前をHUBにする。
```

CanvasLayer に子ノード追加 (Label)

2D タブより配置を GUI で変更。
Label をクリックして Text に”Speed: 0.0 m/s” を入力して実行。そしたら文字が表示される。

1.2.1 スクリプトから速度を自動で表示する

PathFollow3D にアタッチ↓

```
  extends PathFollow3D

var speed := 0.0
var acceleration := 10.0
var max_speed := 30.0

@onready var speed_label := get_node("/root/Main/CanvasLayer/Label")

func _process(delta):
    if Input.is_action_pressed("ui_up"):
        speed += acceleration * delta
    elif Input.is_action_pressed("ui_down"):
        speed -= acceleration * delta
    else:
        speed *= 0.98

    speed = clamp(speed, 0.0, max_speed)

    var curve = get_parent().curve
    if curve:
        var path_length = curve.get_baked_length()
        progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

    # 修正済み！ 小数点1桁で表示
    if speed_label:
        speed_label.text = "Speed: " + String.num(speed, 1) + " m/s"
```

1.2.2 カメラを運転手目線に

カメラを MeshInstance3D（電車モデル）の子ノードにすると、電車が動いたり回転したりするときにカメラも一緒についてくる

1.2.3 動いてるのがわかるように建物を作る

1.2.4 太陽光作る

子ノード追加 (DirectionalLight3D)

1.2.5 電車の前照灯

SpotLight3D を電車の子ノードに

1.2.6 ノッチつくる

```
 Main (Node3D)
├── Path3D
│    └── PathFollow3D
│        └── MeshInstance3D
│            ├── Camera3D
│            └── SpotLight
├── CanvasLayer
│    ├── SpeedLabel (Label)
│    └── ModeLa(Label)
```

コードは

```
  extends PathFollow3D

var speed := 0.0
var acceleration := 10.0
var max_speed := 30.0

var notch_level := 0  # 範囲: -8(EB)～5(P5)

const NOTCH_MAX := 5   # P5
const NOTCH_MIN := -8  # EB

var notch_change_interval := 0.1  # 長押し時の間隔を0.1秒に短縮
var notch_change_timer := 0.0

@onready var speed_label := get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label := get_node("/root/Main/CanvasLayer/ModeLabel")

func _process(delta):
        var input_up := Input.is_action_pressed("ui_up")
        var input_down := Input.is_action_pressed("ui_down")

        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if input_up:
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif input_down:
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # 速度制御：ノッチの符号に応じて加速・減速を決定
        # 正のノッチは加速、負のノッチは減速、0は惰行減速
        if notch_level > 0:
                speed += notch_level * acceleration * delta * 0.3
        elif notch_level < 0:
                speed += notch_level * acceleration * delta * 0.3  # notch_levelは負数なので減速方向
        else:
                speed *= 0.98  # 惰行で徐々に減速

        speed = clamp(speed, 0.0, max_speed)

        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed

        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)

func notch_level_to_text(level: int) -> String:
        match level:
                5:
                        return "P5"
                4:
                        return "P4"
                3:
                        return "P3"
                2:
                        return "P2"
                1:
                        return "P1"
                0:
                        return "N"
                -1:
                        return "B1"
                -2:
```

```
                return "B2"
        -3:
                return "B3"
        -4:
                return "B4"
        -5:
                return "B5"
        -6:
                return "B6"
        -7:
                return "B7"
        -8:
                return "EB"
        _:
                return "N"
```

1.3 運転台を描画

```
 Main
├── Path3D
│   └── PathFollow3D
│       └── MeshInstance3D
│           ├── Camera3D
│           └── SpotLight3D
├── CanvasLayer
│   ├── NotchDisplay (Control)
│   └── Speedometer (Control)
│       └── Label
│   ├── SpeedLabel
│   └── ModelLabel
├── MeshInstance3D
└── DirectionalLight3D
```

1.3.1 下のファイルシステムに

```
 https://fonts.google.com/からダウンロードしたファイルをres://において\\
 (例)res://fonts/Open_Sans/OpenSans-Italic-VariableFont_wdth,wght.ttf\\
```

speedometer

```
extends Control

@export var max_speed: float = 200.0  # 最大速度（km/hなど）
var current_speed: float = 0.0

func _ready() -> void:
        queue_redraw()

func _process(delta: float) -> void:
        # 速度を増減させる（例: ゆっくり加速→リセット）
        current_speed += delta * 30
        if current_speed > max_speed:
                current_speed = 0
        queue_redraw()

func _draw() -> void:
        var center = size / 2
        var radius = min(size.x, size.y) / 2 - 20

        # 針の太さ
        var needle_thickness = 4

        # 外周の丸（速度計の枠）
        draw_circle(center, radius + 10, Color(0, 0, 0))  # 黒枠
        draw_circle(center, radius + 8, Color(0.9, 0.9, 0.9))  # 白い内側

        # 目盛りを描く
        var steps = 10  # 10刻みで表示（0〜max_speed）
```

```
        for i in range(steps + 1):
                var angle = lerp(-PI * 3 / 4, PI * 3 / 4, i / steps)  # 左から右へ270度開く針
                var inner_point = center + Vector2(cos(angle), sin(angle)) * (radius - 10)
                var outer_point = center + Vector2(cos(angle), sin(angle)) * radius
                draw_line(inner_point, outer_point, Color(0, 0, 0), 2)

                # 数字も表示
                var font = get_theme_default_font()
                var speed_value = int(max_speed * i / steps)
                var text = str(speed_value)
                var text_size = font.get_string_size(text)
                var text_pos = center + Vector2(cos(angle), sin(angle)) * (radius - 30) - text_size
                    / 2
                draw_string(font, text_pos, text, 16)

        # 針の角度（0km/hで左端、max_speedで右端）
        var needle_angle = lerp(-PI * 3 / 4, PI * 3 / 4, current_speed / max_speed)

        # 針の先端座標
        var needle_tip = center + Vector2(cos(needle_angle), sin(needle_angle)) * (radius - 20)

        # 針を描く（赤色）
        draw_line(center, needle_tip, Color(1, 0, 0), needle_thickness)

        # 中心の丸（針の根元）
        draw_circle(center, 8, Color(0, 0, 0))

        # 速度の数値も中央に表示
        var font = get_theme_default_font()
        var speed_text = "%.1f" % current_speed + " km/h"
        var text_size = font.get_string_size(speed_text)
        var text_pos = center + Vector2(0, radius / 2) - text_size / 2
        draw_string(font, text_pos, speed_text, 20)

func set_speed(speed_val: float) -> void:
        current_speed = clamp(speed_val, 0.0, max_speed)
        queue_redraw()
```

notch

```
extends Control

var notch: String = "N"
var font: Font

var notch_levels: Array[String] = ["EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1", "N", "P1", "P2",
    "P3", "P4", "P5"]
var notch_index: int = 8  # "N" の位置

func _ready() -> void:
        font = get_theme_default_font()

        if font == null:
                font = FontFile.new()
                var font_data = load("res://fonts/Open_Sans/OpenSans-Italic-VariableFont_wdth,wght.
                    ttf")
                if font_data == null:
                        push_error("フォントファイルの読み込みに失敗しました")
                else:
                        font.font_data = font_data
                        font.size = 16

        focus_mode = FOCUS_ALL
        queue_redraw()

func _draw() -> void:
        if font == null:
                return

        var width: float = size.x
        var height: float = size.y
        var index: int = notch_levels.find(notch)
```

```
        if index == -1:
                return

        var b1_index = notch_levels.find("B1")  # 7
        var p1_index = notch_levels.find("P1")  # 9

        for i in range(notch_levels.size()):
                var rect := Rect2(0, i * height / notch_levels.size(), width, height / notch_levels.
                    size())
                var color := Color(0.2, 0.2, 0.2)  # デフォルト色

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)  # 赤
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)  # 黄色
                elif notch.begins_with("B") and i >= index and i <= b1_index:
                        color = Color(1, 0.5, 0)  # オレンジ
                elif notch.begins_with("P") and i >= p1_index and i <= index:
                        color = Color(0, 1, 0)  # 緑

                # 矩形描画
                draw_rect(rect, color)

                # ラベル描画（矩形の中央に揃える）
                var label: String = notch_levels[i]
                var text_size: Vector2 = font.get_string_size(label)
                var ascent: float = font.get_ascent()
                var pos_x: float = rect.position.x + (rect.size.x - text_size.x) / 2
                var pos_y: float = rect.position.y + (rect.size.y + text_size.y) / 2 - (text_size.y
                    - ascent)
                draw_string(font, Vector2(pos_x, pos_y), label, HORIZONTAL_ALIGNMENT_CENTER)

func _unhandled_key_input(event: InputEvent) -> void:
        if event is InputEventKey and event.pressed and not event.echo:
                if event.keycode == KEY_UP:
                        notch_index = max(0, notch_index - 1)
                        notch = notch_levels[notch_index]
                        queue_redraw()
                elif event.keycode == KEY_DOWN:
                        notch_index = min(notch_levels.size() - 1, notch_index + 1)
                        notch = notch_levels[notch_index]
                        queue_redraw()

func set_notch(value: String) -> void:
        notch = value
        notch_index = notch_levels.find(value)
        queue_redraw()
```

1.4 キーボードの確認

プロジェクト設定 -> プロジェクト設定 -> インプットマップ -> 組み込みアクションを表示

1.5 ノッチ、速度を UI と実際のものをリンクさせる

path_follow_3d.gd

```
extends PathFollow3D

var speed := 0.0
var acceleration := 10.0
var max_speed := 30.0

var notch_level := 0  # 範囲: -8(EB)〜5(P5)

const NOTCH_MAX := 5   # P5
const NOTCH_MIN := -8  # EB

var notch_change_interval := 0.1
```

```
var notch_change_timer := 0.0

@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

func _process(delta: float) -> void:
        # ノッチ入力
        var input_up := Input.is_action_pressed("ui_up")
        var input_down := Input.is_action_pressed("ui_down")

        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if input_up:
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif input_down:
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # 速度制御
        if notch_level > 0:
                speed += notch_level * acceleration * delta * 0.3
        elif notch_level < 0:
                speed += notch_level * acceleration * delta * 0.3
        else:
                speed *= 0.98  # 惰行

        speed = clamp(speed, 0.0, max_speed)

        # 路線に沿って移動
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # === UI更新 ===
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))

func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

notch_display.gd

```
  extends Control

var notch: String = "N"
```

```
var font: Font

var notch_levels: Array[String] = ["EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1", "N", "P1", "P2",
    "P3", "P4", "P5"]
var notch_index: int = 8  # "N" の位置

func _ready() -> void:
        font = get_theme_default_font()

        if font == null:
                font = FontFile.new()
                var font_data = load("res://fonts/Open_Sans/OpenSans-Italic-VariableFont_wdth,wght.
                    ttf")
                if font_data == null:
                        push_error("フォントファイルの読み込みに失敗しました")
                else:
                        font.font_data = font_data
                        font.size = 16

        focus_mode = FOCUS_ALL
        queue_redraw()

func _draw() -> void:
        if font == null:
                return

        var width: float = size.x
        var height: float = size.y
        var index: int = notch_levels.find(notch)

        if index == -1:
                return

        var b1_index = notch_levels.find("B1")  # 7
        var p1_index = notch_levels.find("P1")  # 9

        for i in range(notch_levels.size()):
                var rect := Rect2(0, i * height / notch_levels.size(), width, height / notch_levels.
                    size())
                var color := Color(0.2, 0.2, 0.2)  # デフォルト色

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)  # 赤
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)  # 黄色
                elif notch.begins_with("B") and i >= index and i <= b1_index:
                        # B系は index（例: B3=5）以上、B1=7以下に色付け（範囲逆）
                        color = Color(1, 0.5, 0)  # オレンジ
                elif notch.begins_with("P") and i >= p1_index and i <= index:
                        color = Color(0, 1, 0)  # 緑

                draw_rect(rect, color)

                var label: String = notch_levels[i]
                var text_size: Vector2 = font.get_string_size(label)
                var pos: Vector2 = rect.position + (rect.size / 2) - (text_size / 2)
                draw_string(font, pos, label)

func _unhandled_key_input(event: InputEvent) -> void:
        if event is InputEventKey and event.pressed and not event.echo:
                if event.keycode == KEY_UP:
                        notch_index = max(0, notch_index - 1)
                        notch = notch_levels[notch_index]
                        queue_redraw()
                elif event.keycode == KEY_DOWN:
                        notch_index = min(notch_levels.size() - 1, notch_index + 1)
                        notch = notch_levels[notch_index]
                        queue_redraw()

func set_notch(value: String) -> void:
        notch = value
```

```
        notch_index = notch_levels.find(value)
        queue_redraw()
```

speedometer.gd

```
  extends Control

@export var max_speed: float = 30.0   # 最大速度（m/s）
var current_speed: float = 0.0        # 現在の速度（m/s）

func _ready() -> void:
        queue_redraw()

func _process(_delta: float) -> void:
        # 速度計は列車スクリプトから set_speed() で更新されるので
        # ここでは何もしない
        queue_redraw()

func _draw() -> void:
        var center: Vector2 = size / 2
        var radius: float = min(size.x, size.y) / 2.0 - 20.0
        var needle_thickness: float = 4.0

        # 外周円
        draw_circle(center, radius + 10.0, Color.BLACK)
        draw_circle(center, radius + 8.0, Color(0.9, 0.9, 0.9))

        # メモリ描画
        var steps: int = int(max_speed)  # 1 m/s ごとに目盛りを打つ
        var font: Font = get_theme_default_font()

        # 角度範囲（-135° 〜 +135°）
        var start_angle: float = -PI * 3.0 / 4.0
        var end_angle: float = PI * 3.0 / 4.0

        for i in range(steps + 1):
                var angle: float = lerp(start_angle, end_angle, float(i) / float(steps))
                var inner: Vector2 = center + Vector2(cos(angle), sin(angle)) * (radius - 10.0)
                var outer: Vector2 = center + Vector2(cos(angle), sin(angle)) * radius
                draw_line(inner, outer, Color.BLACK, 2.0 if i % 5 == 0 else 1.0)

                # 5m/sごとに数値ラベルを描画
                if i % 5 == 0:
                        var text: String = str(i) + " m/s"
                        var text_size: Vector2 = font.get_string_size(text)
                        var text_pos: Vector2 = center + Vector2(cos(angle), sin(angle)) * (radius -
                             30.0) - text_size / 2.0
                        draw_string(font, text_pos, text)

        # 針の描画
        var needle_angle: float = lerp(start_angle, end_angle, current_speed / max_speed)
        var needle_tip: Vector2 = center + Vector2(cos(needle_angle), sin(needle_angle)) * (radius -
             20.0)
        draw_line(center, needle_tip, Color.RED, needle_thickness)

        # 中心の円
        draw_circle(center, 8.0, Color.BLACK)

        # 中央に速度表示（数値）
        var speed_text: String = "%.1f m/s" % current_speed
        var text_size_center: Vector2 = font.get_string_size(speed_text)
        var text_pos_center: Vector2 = center + Vector2(0, radius / 2.0) - text_size_center / 2.0
        draw_string(font, text_pos_center, speed_text)

# 列車スクリプトから呼び出して速度を更新
func set_speed(speed_val: float) -> void:
        current_speed = clamp(speed_val, 0.0, max_speed)
```

mesh_instance_3d.gd

```
  extends PathFollow3D

var speed := 0.0
```

```
var max_speed := 30.0
var acceleration := 5.0
var brake_strength := 6.0
var emergency_brake_strength := 20.0

# ブレーキ・パワーモードリスト（インデックス5が"N"）
var brake_modes := ["P5", "P4", "P3", "P2", "P1", "N", "B1", "B2", "B3", "B4", "B5", "B6", "B7", "EB
    "]
var brake_index := 5  # 初期は "N"

@onready var speed_label: Label = get_node_or_null("../../../CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node_or_null("../../../CanvasLayer/ModeLabel")


func _ready() -> void:
        if speed_label == null:
                print("SpeedLabel ノードが見つかりません。名前・階層を確認してください。")
        if mode_label == null:
                print("ModeLabel ノードが見つかりません。名前・階層を確認してください。")


func _input(event: InputEvent) -> void:
        if event is InputEventKey and event.pressed and not event.echo:
                # W キーでブレーキ強化（インデックス増）
                if event.scancode == Key.W:
                        if brake_index < brake_modes.size() - 1:
                                brake_index += 1
                # X キーでブレーキ解除方向へ（インデックス減）
                elif event.scancode == Key.X:
                        if brake_index > 0:
                                brake_index -= 1
                # S キーでニュートラルに戻す
                elif event.scancode == Key.S:
                        brake_index = 5  # "N"


func _process(delta: float) -> void:
        var mode := brake_modes[brake_index]

        # モードに応じた速度変化
        if mode.begins_with("P"):
                var power_level := int(mode.substr(1, 1))
                speed += acceleration * power_level * delta
        elif mode.begins_with("B"):
                var brake_level := int(mode.substr(1, 1))
                speed -= brake_strength * brake_level * delta
        elif mode == "EB":
                speed -= emergency_brake_strength * delta
        else:
                # ニュートラル（惰性でゆっくり減速）
                speed *= 0.995

        speed = clamp(speed, 0.0, max_speed)

        # Pathに沿って進む（進行度 progress を管理）
        var curve := get_parent().curve
        if curve:
                var path_length := curve.get_baked_length()
                progress = fposmod(progress + (speed * delta), path_length)
                set_offset(progress)

        # UI更新
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Mode: %s" % mode
```

1.6 WSX操作できるようにする

notch

```
extends Control

var notch: String = "N"
var font: Font

var notch_levels: Array[String] = [
        "EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1",
        "N",
        "P1", "P2", "P3", "P4", "P5"
]
var notch_index: int = 8  # "N"

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL
        queue_redraw()

func _draw() -> void:
        if font == null:
                return

        var width := size.x
        var height := size.y
        var per_h := height / float(notch_levels.size())

        var b1_index := notch_levels.find("B1")
        var p1_index := notch_levels.find("P1")

        for i in range(notch_levels.size()):
                var rect := Rect2(0.0, i * per_h, width, per_h)
                var color := Color(0.2, 0.2, 0.2)

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)
                elif notch.begins_with("B") and i >= notch_index and i <= b1_index:
                        color = Color(1, 0.5, 0)
                elif notch.begins_with("P") and i >= p1_index and i <= notch_index:
                        color = Color(0, 1, 0)

                draw_rect(rect, color)

                var label := notch_levels[i]
                var text_size := font.get_string_size(label)
                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent

                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent

                draw_string(font, pos, label)

func set_notch(value: String) -> void:
        var idx := notch_levels.find(value)
        if idx != -1:
                notch_index = idx
                notch = value
                queue_redraw()
```

speed

```
extends Control

@export var max_speed: float = 30.0
var current_speed: float = 0.0
@export var font: Font = null

func _ready() -> void:
        if font == null:
```

```
                font = get_theme_default_font()
        queue_redraw()

func _process(_delta: float) -> void:
        queue_redraw()

func _draw() -> void:
        var center: Vector2 = size / 2
        var radius: float = min(size.x, size.y) / 2.0 - 20.0
        var needle_thickness: float = 6.0

        # --- 外周円 ---
        draw_circle(center, radius + 10.0, Color.BLACK)
        draw_circle(center, radius + 8.0, Color(0.9, 0.9, 0.9))

        # --- メモリ描画 ---
        var steps: int = int(max_speed)
        var font_used := font

        # 角度範囲（上下対称）：上180°スタート → 下0°終了
        var start_angle: float = deg_to_rad(200)
        var end_angle: float = deg_to_rad(-20)

        for i in range(steps + 1):
                var angle: float = lerp(start_angle, end_angle, float(i) / float(steps))
                # 上下移動用のY軸反転
                var inner: Vector2 = center + Vector2(cos(angle), -sin(angle)) * (radius - 10.0)
                var outer: Vector2 = center + Vector2(cos(angle), -sin(angle)) * radius
                draw_line(inner, outer, Color.BLACK, 2.0 if i % 5 == 0 else 1.0)

                if i % 5 == 0:
                        var text: String = str(i) + " m/s"
                        var text_size: Vector2 = font_used.get_string_size(text)
                        var text_pos: Vector2 = center + Vector2(cos(angle), -sin(angle)) * (radius
                            - 30.0) - text_size / 2.0
                        draw_string(font_used, text_pos, text)

        # --- 針描画（尖らせる） ---
        var needle_angle: float = lerp(start_angle, end_angle, current_speed / max_speed)
        var needle_length := radius - 20.0
        var tip := center + Vector2(cos(needle_angle), -sin(needle_angle)) * needle_length

        var base_width := 6.0
        var tip_width := 1.0
        var dir := (tip - center).normalized()
        var ortho := Vector2(-dir.y, dir.x)
        var base_left := center - ortho * base_width * 0.5
        var base_right := center + ortho * base_width * 0.5
        var tip_left := tip - ortho * tip_width * 0.5
        var tip_right := tip + ortho * tip_width * 0.5

        draw_polygon([base_left, base_right, tip_right, tip_left], [Color.BLACK])

        # --- 中心の円 ---
        draw_circle(center, 8.0, Color.BLACK)

        # --- 中央に速度表示 ---
        var speed_text: String = "%.1f m/s" % current_speed
        var text_size_center: Vector2 = font_used.get_string_size(speed_text)
        var text_pos_center: Vector2 = center + Vector2(0, radius / 2.0) - text_size_center / 2.0
        draw_string(font_used, text_pos_center, speed_text)

func set_speed(speed_val: float) -> void:
        current_speed = clamp(speed_val, 0.0, max_speed)
        queue_redraw()
```

mesh

```
    extends PathFollow3D

var speed := 0.0
var max_speed := 30.0
```

```
var acceleration := 5.0
var brake_strength := 6.0
var emergency_brake_strength := 20.0

# ブレーキ・パワーモードリスト（インデックス5が"N"）
var brake_modes := ["P5", "P4", "P3", "P2", "P1", "N", "B1", "B2", "B3", "B4", "B5", "B6", "B7", "EB
    "]
var brake_index := 5  # 初期は "N"

@onready var speed_label: Label = get_node_or_null("../../../CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node_or_null("../../../CanvasLayer/ModeLabel")


func _ready() -> void:
        if speed_label == null:
                print("SpeedLabel ノードが見つかりません。名前・階層を確認してください。")
        if mode_label == null:
                print("ModeLabel ノードが見つかりません。名前・階層を確認してください。")


func _input(event: InputEvent) -> void:
        if event is InputEventKey and event.pressed and not event.echo:
                # W キーでブレーキ強化（インデックス増）
                if event.scancode == Key.W:
                        if brake_index < brake_modes.size() - 1:
                                brake_index += 1
                # X キーでブレーキ解除方向へ（インデックス減）
                elif event.scancode == Key.X:
                        if brake_index > 0:
                                brake_index -= 1
                # S キーでニュートラルに戻す
                elif event.scancode == Key.S:
                        brake_index = 5  # "N"


func _process(delta: float) -> void:
        var mode := brake_modes[brake_index]

        # モードに応じた速度変化
        if mode.begins_with("P"):
                var power_level := int(mode.substr(1, 1))
                speed += acceleration * power_level * delta
        elif mode.begins_with("B"):
                var brake_level := int(mode.substr(1, 1))
                speed -= brake_strength * brake_level * delta
        elif mode == "EB":
                speed -= emergency_brake_strength * delta
        else:
                # ニュートラル（惰性でゆっくり減速）
                speed *= 0.995

        speed = clamp(speed, 0.0, max_speed)

        # Pathに沿って進む（進行度 progress を管理）
        var curve := get_parent().curve
        if curve:
                var path_length := curve.get_baked_length()
                progress = fposmod(progress + (speed * delta), path_length)
                set_offset(progress)

        # UI更新
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Mode: %s" % mode
```

1.7 WSX操作できるようにする

notch

```
extends Control

var notch: String = "N"
var font: Font

var notch_levels: Array[String] = [
        "EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1",
        "N",
        "P1", "P2", "P3", "P4", "P5"
]
var notch_index: int = 8  # "N"

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL
        queue_redraw()

func _draw() -> void:
        if font == null:
                return

        var width: float = size.x
        var height: float = size.y
        var index: int = notch_levels.find(notch)
        if index == -1:
                return

        var b1_index = notch_levels.find("B1")
        var p1_index = notch_levels.find("P1")

        for i in range(notch_levels.size()):
                var rect := Rect2(0, i * height / notch_levels.size(), width, height / notch_levels.
                    size())
                var color := Color(0.2, 0.2, 0.2)

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)
                elif notch.begins_with("B") and i >= index and i <= b1_index:
                        color = Color(1, 0.5, 0)
                elif notch.begins_with("P") and i >= p1_index and i <= index:
                        color = Color(0, 1, 0)

                # 背景矩形
                draw_rect(rect, color)

                # ---- ラベル描画（矩形中央揃え + 正確なbaseline補正）----
                var label: String = notch_levels[i]
                var text_size: Vector2 = font.get_string_size(label)

                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent

                # 矩形中央に配置（baseline基準のため ascent を追加）
                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent

                draw_string(font, pos, label)

func set_notch(value: String) -> void:
        notch = value
        notch_index = notch_levels.find(value)
        queue_redraw()
\end{lselisting}

path_fllow
\begin{lstlisting}
  extends PathFollow3D
```

```
var speed := 0.0
var acceleration := 10.0
var max_speed := 30.0

var notch_level := 0  # 範囲: -8(EB)〜+5(P5)

const NOTCH_MAX := 5    # P5
const NOTCH_MIN := -8   # EB

var notch_change_interval := 0.1
var notch_change_timer := 0.0

@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

func _process(delta: float) -> void:
        # --- ノッチ操作 ---
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):  # W
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):  # X
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):  # S
                        # 段階的に N に近づける
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # --- 速度制御 ---
        if notch_level > 0:
                speed += notch_level * acceleration * delta * 0.3
        elif notch_level < 0:
                speed += notch_level * acceleration * delta * 0.3
        else:
                speed *= 0.98  # 惰行

        speed = clamp(speed, 0.0, max_speed)

        # --- 経路に沿って移動 ---
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # --- UI更新 ---
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed

        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)

        if speedometer:
                speedometer.set_speed(speed)

        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))


func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
```

```
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.8 まともな速度計を作る

黒背景に白文字で、km/h 単位、速度を実際のものとリンクさせる、減速率や惰行の減速率、加速度を正確に、最高速度を 90km/h に、メーターの表示は 100km/h まで

```
  path_follow_3d

  extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0              # 現在速度 (m/s)
var max_speed := 25.0         # 最大速度 (m/s) 90 km/h ≈ 25 m/s
var acceleration := 0.31      # 加速 m/s^2 (1秒で4 km/h)
var coasting_deceleration := 0.0277 # 惰行減速 m/s^2 (10秒で1 km/h)

# --- ノッチ設定 ---
var notch_level := 0          # 範囲: -8(EB)～+5(P5)
const NOTCH_MAX := 5          # P5
const NOTCH_MIN := -8         # EB
var notch_change_interval := 0.1
var notch_change_timer := 0.0

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

func _process(delta: float) -> void:
        # --- ノッチ操作 ---
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):  # W
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):  # X
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):  # S
                        # 段階的に N に近づける
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # --- 速度制御 ---
        if notch_level > 0:
                # 前進ノッチで加速
                speed += notch_level * acceleration * delta
        elif notch_level < 0:
                # ブレーキノッチで減速
                speed += notch_level * acceleration * delta
        else:
```

```
                # 惰行（ほぼ維持）
                speed = max(speed - coasting_deceleration * delta, 0.0)

        speed = clamp(speed, 0.0, max_speed)

        # --- 経路に沿って移動 ---
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # --- UI更新 ---
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed

        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)

        if speedometer:
                speedometer.set_speed(speed * 3.6) # km/h表示

        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))


# --- ノッチ値を文字列に変換 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

speedometer

```
extends Control

@export var max_speed: float = 100.0
var current_speed: float = 0.0
@export var font: Font = null

func _ready() -> void:
        if font == null:
                font = get_theme_default_font()
        queue_redraw()

func _process(_delta: float) -> void:
        queue_redraw()

func _draw() -> void:
        var center = size / 2
        var radius = min(size.x, size.y) / 2.0 - 20.0

        # 外周円
        draw_circle(center, radius, Color(0.2, 0.2, 0.2))

        # 目盛り（10km/hごと）
        var start_angle = deg_to_rad(200)
        var end_angle = deg_to_rad(-20)
        for i in range(0, int(max_speed)+1, 10):
                var t = float(i) / max_speed
```

```
                var angle = lerp(start_angle, end_angle, t)
                var inner = center + Vector2(cos(angle), -sin(angle)) * (radius - 10)
                var outer = center + Vector2(cos(angle), -sin(angle)) * radius
                draw_line(inner, outer, Color.BLACK, 2)

                # 数字
                var text = str(i)
                var text_size = font.get_string_size(text)
                var text_pos = center + Vector2(cos(angle), -sin(angle)) * (radius - 30) - text_size
                    / 2

                # 色を変えるには modulate を一時的に変更
                modulate = Color.BLACK
                draw_string(font, text_pos, text)
                modulate = Color(1, 1, 1)  # 元に戻す

        # 針
        var needle_angle = lerp(start_angle, end_angle, current_speed / max_speed)
        var tip = center + Vector2(cos(needle_angle), -sin(needle_angle)) * (radius - 10)
        draw_line(center, tip, Color.RED, 3)

        # 現在速度表示
        var speed_text = str(round(current_speed)) + " km/h"
        modulate = Color.BLACK
        draw_string(font, center + Vector2(-30, radius / 2), speed_text)
        modulate = Color(1, 1, 1)

func set_speed(speed_val: float) -> void:
        current_speed = clamp(speed_val, 0.0, max_speed)
        queue_redraw()
```

notch_display

```
  extends Control

var notch: String = "N"
var font: Font

var notch_levels: Array[String] = [
        "EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1",
        "N",
        "P1", "P2", "P3", "P4", "P5"
]
var notch_index: int = 8  # "N"

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL
        queue_redraw()

func _draw() -> void:
        if font == null:
                return

        var width := size.x
        var height := size.y
        var per_h := height / float(notch_levels.size())

        var b1_index := notch_levels.find("B1")
        var p1_index := notch_levels.find("P1")

        for i in range(notch_levels.size()):
                var rect := Rect2(0.0, i * per_h, width, per_h)
                var color := Color(0.2, 0.2, 0.2)

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)
                elif notch.begins_with("B") and i >= notch_index and i <= b1_index:
                        color = Color(1, 0.5, 0)
                elif notch.begins_with("P") and i >= p1_index and i <= notch_index:
```

```
                        color = Color(0, 1, 0)

                draw_rect(rect, color)

                var label := notch_levels[i]
                var text_size := font.get_string_size(label)
                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent

                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent

                draw_string(font, pos, label)

func set_notch(value: String) -> void:
        var idx := notch_levels.find(value)
        if idx != -1:
                notch_index = idx
                notch = value
                queue_redraw()
```

1.9 線路を作る

Blender で線路を作る (SHFT-A より追加) -> Godot のファイルシステムのところにドラックアンドドロップ -> ファイルシステムのところから 3D のとこのエディタにドラックアンドドロップ

1.10 警笛をつける

列車ノードの下に AudioStreemPlayer3D -> インスペクターの Streem 空 -> AudioStreemGenerator -> Playing オンチェック

horn

```
extends AudioStreamPlayer3D

@export var base_freq: float = 350.0 # 基本周波数を低めに設定
@export var release_time: float = 0.5

var generator: AudioStreamGenerator
var playback: AudioStreamGeneratorPlayback
var phase: float = 0.0
var amplitude: float = 0.0

func _ready():
        generator = AudioStreamGenerator.new()
        generator.mix_rate = 44100
        stream = generator
        play()
        playback = get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta):
        var target_amp = 0.0
        if Input.is_action_pressed("horn_normal"): # Hキー
                target_amp = 1.5 # 音量
        # 緩やかに変化させる
        amplitude += (target_amp - amplitude) * min(delta * 3, 1)
        push_tone(delta)

func push_tone(delta: float):
        if playback == null:
                return
        var frames_to_generate = int(generator.mix_rate * delta)
        for i in range(frames_to_generate):
                var vibrato = 4.0 * sin(phase * 0.003) # ゆったりしたビブラート
                # 基本音に少しだけ倍音を混ぜて柔らかく
                var sample = amplitude * (sin(phase + vibrato) + 0.25 * sin((phase + vibrato) *
                    2.01)) / 1.25
                playback.push_frame(Vector2(sample, sample))
                phase += 2.0 * PI * base_freq / generator.mix_rate
                if phase > 2.0 * PI:
```

```
                        phase -= 2.0 * PI
```

whistle

```
extends AudioStreamPlayer3D

# --- 基本設定 ---
@export var base_freq: float = 350.0   # 基本周波数（低くして重厚感）
@export var release_time: float = 0.3  # 音量減衰時間

var generator: AudioStreamGenerator
var playback: AudioStreamGeneratorPlayback
var phase: float = 0.0
var amplitude: float = 0.0

func _ready():
        # AudioStreamGenerator 作成
        generator = AudioStreamGenerator.new()
        generator.mix_rate = 44100
        stream = generator
        play()
        playback = get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta: float):
        var target_amp = 0.0
        # ホーンキーが押されている場合、音量を最大に
        if Input.is_action_pressed("horn_emergency"): # 例: Jキー
                target_amp = 1.0
        # 音量を滑らかに変化させる
        amplitude += (target_amp - amplitude) * min(delta * 10, 1)
        # 音を生成
        push_tone(delta)

func push_tone(delta: float):
        if playback == null:
                return
        var frames_to_generate = max(1, int(generator.mix_rate * delta))
        for i in range(frames_to_generate):
                # ビブラート用の小さな揺らぎ
                var vibrato = 5.0 * sin(phase * 0.02)
                # 複数周波数で濁らせて低音を強調
                var sample = amplitude * (
                        sin(phase + vibrato) +          # 基本周波数
                        0.3 * sin((phase + vibrato) * 2) + # 2倍音
                        0.2 * sin((phase + vibrato) * 3)   # 3倍音
                ) / 1.5 # 正規化
                playback.push_frame(Vector2(sample, sample))
                # フェーズを進める
                phase += 2.0 * PI * base_freq / generator.mix_rate
                if phase > 2.0 * PI:
                phase -= 2.0 * PI
```

1.11 ノッチに音をつける

ノッチのスクリプトを以下に変更

```
extends Control

# --- ノッチUIは既存 ---
var notch: String = "N"
var font: Font
var notch_levels: Array[String] = [
        "EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1",
        "N",
        "P1", "P2", "P3", "P4", "P5"
]
var notch_index: int = 8  # "N"

# --- カタッ音用 ---
var click_generator: AudioStreamGenerator
var click_playback: AudioStreamGeneratorPlayback
```

```
@export var click_duration: float = 0.05  # 秒

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL
        queue_redraw()

        # カタッ音ジェネレーター初期化
        click_generator = AudioStreamGenerator.new()
        click_generator.mix_rate = 44100
        var click_player = AudioStreamPlayer3D.new()
        add_child(click_player)
        click_player.stream = click_generator
        click_player.play()
        click_playback = click_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _draw() -> void:
        if font == null:
                return
        var width := size.x
        var height := size.y
        var per_h := height / float(notch_levels.size())

        var b1_index := notch_levels.find("B1")
        var p1_index := notch_levels.find("P1")

        for i in range(notch_levels.size()):
                var rect := Rect2(0.0, i * per_h, width, per_h)
                var color := Color(0.2, 0.2, 0.2)

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)
                elif notch == "N" and i == notch_levels.find("N"):
                        color = Color(1, 1, 0)
                elif notch.begins_with("B") and i >= notch_index and i <= b1_index:
                        color = Color(1, 0.5, 0)
                elif notch.begins_with("P") and i >= p1_index and i <= notch_index:
                        color = Color(0, 1, 0)

                draw_rect(rect, color)
                var label := notch_levels[i]
                var text_size := font.get_string_size(label)
                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent
                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent
                draw_string(font, pos, label)

func set_notch(value: String) -> void:
        var idx := notch_levels.find(value)
        if idx != -1:
                if idx != notch_index:
                        _play_notch_click()  # ノッチが変わった瞬間にカタッ音
                notch_index = idx
                notch = value
                queue_redraw()

func _play_notch_click():
        if click_playback == null:
                return
        var frames_to_generate = int(click_generator.mix_rate * click_duration)
        var click_freq = 800.0  # Hz、少し低めで硬い音
        for i in range(frames_to_generate):
                # 短いサイン波 + 急減衰
                var amp = exp(-15.0 * i / frames_to_generate) * 0.2
                var sample = sin(2.0 * PI * click_freq * i / click_generator.mix_rate) * amp
                click_playback.push_frame(Vector2(sample, sample))
```

1.12 加速音をつけたい

MeshInstance に AudioStreenPlayer3D つけて、インスペクターよりあれつけて、Playing オンにしてスクリプトは pathfollow

```
extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0
var max_speed := 25.0
var acceleration := 0.31
var coasting_deceleration := 0.0277

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

@export var base_freq_accel: float = 200.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3

func _ready():
        # 加速音
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 減速音
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta: float) -> void:
        # --- ノッチ操作 ---
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
```

```
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # --- 速度制御 ---
        if notch_level > 0:
                speed += notch_level * acceleration * delta
        elif notch_level < 0:
                speed += notch_level * acceleration * delta
        else:
                speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

        # --- 経路移動 ---
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # --- UI更新 ---
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))

        # --- 音生成 ---
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        var accel_freq = base_freq_accel
        var brake_freq = base_freq_brake

        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                        target_brake_amp = 0.0
                        accel_freq = base_freq_accel + speed * 8  # 加速で高く
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp
                        target_accel_amp = 0.0
                        brake_freq = base_freq_brake - speed * 6  # 減速で低く
                else:
                        target_accel_amp = 0.0
                        target_brake_amp = 0.0

        # 滑らかに変化
        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake")

# --- 波形生成関数 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    ):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var phase_inc := 0.0
                var sample := 0.0
```

```
                if kind == "accel":
                        phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI:
                                accel_phase -= 2.0*PI
                elif kind == "brake":
                        phase_inc = 2.0 * PI * (base_freq_brake - speed*6) / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI:
                                brake_phase -= 2.0*PI
                playback.push_frame(Vector2(sample, sample))

# --- ノッチ文字変換 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

accel

```
extends AudioStreamPlayer3D

@export var base_freq: float = 300.0   # 基本周波数
@export var max_amplitude: float = 0.5 # 最大音量

var generator: AudioStreamGenerator
var playback: AudioStreamGeneratorPlayback
var phase: float = 0.0
var amplitude: float = 0.0

func _ready():
        generator = AudioStreamGenerator.new()
        generator.mix_rate = 44100
        stream = generator
        play()
        playback = get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta):
        # 音を生成
        push_tone(base_freq, delta)

func push_tone(freq: float, delta: float):
        if playback == null:
                return
        var frames_to_generate = max(1, int(generator.mix_rate * delta))
        for i in range(frames_to_generate):
                var sample = amplitude * (
                        sin(phase) +
                        0.5 * sin(phase * 2) +
                        0.25 * sin(phase * 3)
                ) / 1.75
                playback.push_frame(Vector2(sample, sample))
                phase += 2.0 * PI * freq / generator.mix_rate
                if phase > 2.0 * PI:
                        phase -= 2.0 * PI
```

brake

```
extends AudioStreamPlayer3D

@export var base_freq: float = 150.0   # 基本周波数
@export var max_amplitude: float = 0.5

var generator: AudioStreamGenerator
var playback: AudioStreamGeneratorPlayback
var phase: float = 0.0
var amplitude: float = 0.0

func _ready():
        generator = AudioStreamGenerator.new()
        generator.mix_rate = 44100
        stream = generator
        play()
        playback = get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta):
        push_tone(base_freq, delta)

func push_tone(freq: float, delta: float):
        if playback == null:
                return
        var frames_to_generate = max(1, int(generator.mix_rate * delta))
        for i in range(frames_to_generate):
                var sample = amplitude * (
                        sin(phase) +
                        0.3 * sin(phase * 2)
                ) / 1.3
                playback.push_frame(Vector2(sample, sample))
                phase += 2.0 * PI * freq / generator.mix_rate
                if phase > 2.0 * PI:
                        phase -= 2.0 * PI
```

1.13 ブレーキ緩解音を作る

path_follow を

```
extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0
var max_speed := 25.0
var acceleration := 0.31
var coasting_deceleration := 0.0277

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
```

```
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

@export var base_freq_accel: float = 500.0  # 加速最初の音程
@export var base_freq_brake: float = 600.0  # 減速最初の音程
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

func _ready():
        # --- 加速音 ---
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # --- 減速音 ---
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # --- ブレーキ緩解音（release音） ---
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback


func _process(delta: float) -> void:
        # --- ノッチ操作 ---
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # --- 速度制御 ---
        if notch_level > 0:
                speed += notch_level * acceleration * delta
        elif notch_level < 0:
                speed += notch_level * acceleration * delta
        else:
                speed = max(speed - coasting_deceleration * delta, 0.0)
```

```
        speed = clamp(speed, 0.0, max_speed)

        # --- 経路移動 ---
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # --- UI更新 ---
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))

        # --- 加速・減速音生成 ---
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0

        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        # 減速音ピッチを加速音ベースで下げる（高速時高音、低速で低音）
        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0  # 高速時高音、低速でベース
            音に直線的

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        push_tone(release_playback, release_generator, "release")

        # --- ブレーキ弱め判定（release音トリガー） ---
        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false


func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * (pitch) / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
```

```
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))


func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.14 ここまでのまとめ

pathfollow を

```
extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0
var max_speed := 25.0
var acceleration := 0.31
var coasting_deceleration := 0.0277

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
```

```
@export var release_duration := 0.5

func _ready():
	rotation_mode = PathFollow3D.ROTATION_XYZ

	# --- Curve の bake_interval を小さくして滑らかに ---
	if get_parent() and get_parent() is Path3D:
		var path = get_parent() as Path3D
		var curve = path.curve
		if curve:
			curve.set_bake_interval(0.01) # デフォルト 0.1 くらい。0.01 以下でより滑らか

	# --- 子メッシュを原点に補正（先頭を原点に） ---
	for child in get_children():
		if child is MeshInstance3D:
			child.position = Vector3.ZERO
			child.rotation_degrees = Vector3.ZERO

	# --- 音源初期化 ---
	_init_audio()

func _init_audio():
	# 加速音
	accel_generator = AudioStreamGenerator.new()
	accel_generator.mix_rate = 44100
	accel_generator.buffer_length = 0.5
	var accel_player = AudioStreamPlayer3D.new()
	add_child(accel_player)
	accel_player.stream = accel_generator
	accel_player.play()
	accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

	# 減速音
	brake_generator = AudioStreamGenerator.new()
	brake_generator.mix_rate = 44100
	brake_generator.buffer_length = 0.5
	var brake_player = AudioStreamPlayer3D.new()
	add_child(brake_player)
	brake_player.stream = brake_generator
	brake_player.play()
	brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

	# ブレーキ緩解音
	release_generator = AudioStreamGenerator.new()
	release_generator.mix_rate = 44100
	var release_player = AudioStreamPlayer3D.new()
	add_child(release_player)
	release_player.stream = release_generator
	release_player.play()
	release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta: float) -> void:
	_update_notch(delta)
	_update_speed(delta)
	_move_along_curve(delta)
	_update_ui()
	_update_audio(delta)

func _update_notch(delta):
	notch_change_timer -= delta
	if notch_change_timer <= 0.0:
		if Input.is_action_pressed("notch_up"):
			notch_level = max(notch_level - 1, NOTCH_MIN)
			notch_change_timer = notch_change_interval
		elif Input.is_action_pressed("notch_down"):
			notch_level = min(notch_level + 1, NOTCH_MAX)
			notch_change_timer = notch_change_interval
		elif Input.is_action_pressed("notch_neutral"):
			if notch_level > 0:
				notch_level -= 1
			elif notch_level < 0:
```

```
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

func _update_speed(delta):
        if notch_level != 0:
                speed += notch_level * acceleration * delta
        else:
                speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

func _move_along_curve(delta):
        var path = get_parent() as Path3D
        if not path:
                return
        var curve = path.curve
        if not curve:
                return

        var path_length = curve.get_baked_length()
        # progress_ratio を距離に応じて滑らかに更新
        progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

func _update_ui():
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))

func _update_audio(delta):
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        push_tone(release_playback, release_generator, "release")

        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false

func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
```

```
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * (pitch) / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))

func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.15 真っ暗

DirectionalLight3D の Light プロパティの Energy より明るさを変えられる

1.16 EB の時のブレーキ音を特別追加

```
  extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0
var max_speed := 25.0
var acceleration := 0.31
var coasting_deceleration := 0.0277

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0
```

```
var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

# --- EB専用 ---
var eb_generator: AudioStreamGenerator
var eb_playback: AudioStreamGeneratorPlayback
var eb_phase := 0.0
var eb_amp := 0.0
var eb_pitch := 0.0
@export var eb_pitch_rise_speed := 200.0
@export var eb_fade_decay := 0.97

# --- パラメータ ---
@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

func _ready():
        rotation_mode = PathFollow3D.ROTATION_XYZ

        # 子メッシュを原点に補正
        for i in range(get_child_count()):
                var child = get_child(i)
                if child is MeshInstance3D:
                        child.position = Vector3.ZERO
                        child.rotation_degrees = Vector3.ZERO

        # --- 加速音 ---
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # --- 減速音 ---
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # --- ブレーキ緩解音 ---
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # --- EB音 ---
        eb_generator = AudioStreamGenerator.new()
        eb_generator.mix_rate = 44100
        eb_generator.buffer_length = 0.5
        var eb_player = AudioStreamPlayer3D.new()
        add_child(eb_player)
```

```
        eb_player.stream = eb_generator
        eb_player.play()
        eb_playback = eb_player.get_stream_playback() as AudioStreamGeneratorPlayback


func _process(delta: float) -> void:
        # --- ノッチ操作 ---
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

        # --- 速度制御 ---
        if notch_level != 0:
                speed += notch_level * acceleration * delta
        else:
                speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

        # --- 経路移動 ---
        var curve = get_parent().curve
        if curve:
                var path_length = curve.get_baked_length()
                progress_ratio = fposmod(progress_ratio + (speed * delta) / path_length, 1.0)

        # --- UI更新 ---
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))

        # --- 加速・減速音 ---
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        var brake_pitch = base_freq_brake

        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch == "EB" and speed > 0.1:
                        eb_amp = max_amp
                        eb_pitch += eb_pitch_rise_speed * delta
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp
                        brake_pitch = base_freq_brake + (speed / max_speed) * 400.0
                        eb_amp = 0.0
                        eb_pitch = 0.0
                else:
                        # EB解除時フェードアウト
                        eb_amp *= eb_fade_decay
                        eb_pitch *= eb_fade_decay

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)
```

```
        # --- 音生成 ---
        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        if eb_amp > 0.001:
                push_tone(eb_playback, eb_generator, "eb", base_freq_brake + eb_pitch, eb_amp)
        push_tone(release_playback, release_generator, "release")

        # --- ブレーキ弱め判定（release音トリガー） ---
        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false


# --- 音生成関数 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0, amp_override: float = -1.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                var amp = amp_override if amp_override >= 0 else (
                        accel_amp if kind=="accel" else brake_amp if kind=="brake" else release_amp
                            if kind=="release" else eb_amp
                )
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(randf
                            ()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * pitch / generator.mix_rate
                        sample = amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(randf
                            ()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "eb":
                        var phase_inc = 2.0 * PI * pitch / generator.mix_rate
                        sample = amp * (sin(eb_phase) + 0.5*sin(eb_phase*2) + 0.05*(randf()*2-1)) /
                            1.6
                        eb_phase += phase_inc
                        if eb_phase > 2.0*PI: eb_phase -= 2.0*PI
                elif kind == "release":
                        sample = amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))


# --- ノッチ表記変換 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
```

```
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.17 逆転機をつける

```
  extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0
var max_speed := 25.0
var acceleration := 0.31
var coasting_deceleration := 0.0277

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- 逆転機 ---
var reverser := 0  # -1=後退, 0=中立, 1=前進

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")
@onready var reverser_label: Label = get_node("/root/Main/CanvasLayer/ReverserLabel")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

# --- 逆転機音 ---
var reverser_generator: AudioStreamGenerator
var reverser_playback: AudioStreamGeneratorPlayback

# --- 音パラメータ ---
@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

func _ready():
        rotation_mode = PathFollow3D.ROTATION_XYZ

        # --- Curve の bake_interval を小さくして滑らかに ---
        if get_parent() and get_parent() is Path3D:
                var path = get_parent() as Path3D
                var curve = path.curve
                if curve:
                        curve.set_bake_interval(0.01)
```

```
        # --- 子メッシュを原点に補正 ---
        for child in get_children():
                if child is MeshInstance3D:
                        child.position = Vector3.ZERO
                        child.rotation_degrees = Vector3.ZERO

        # --- 音源初期化 ---
        _init_audio()

func _init_audio():
        # 加速音
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 減速音
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # ブレーキ緩解音
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 逆転機音
        reverser_generator = AudioStreamGenerator.new()
        reverser_generator.mix_rate = 44100
        reverser_generator.buffer_length = 0.2
        var reverser_player = AudioStreamPlayer3D.new()
        add_child(reverser_player)
        reverser_player.stream = reverser_generator
        reverser_player.play()
        reverser_playback = reverser_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta: float) -> void:
        _update_notch(delta)
        _update_reverser(delta)
        _update_speed(delta)
        _move_along_curve(delta)
        _update_ui()
        _update_audio(delta)

# --- ノッチ操作 ---
func _update_notch(delta):
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
```

```
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

# --- 逆転機操作 ---
func _update_reverser(delta):
        if Input.is_action_just_pressed("reverser_forward") and reverser != 1:
                reverser = 1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_reverse") and reverser != -1:
                reverser = -1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_neutral") and reverser != 0:
                reverser = 0
                _play_reverser_sound()

# --- 速度更新 ---
func _update_speed(delta):
        if notch_level != 0 and reverser != 0:  # 中立だと動かない
                speed += notch_level * acceleration * delta
        else:
                speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

# --- 移動 ---
func _move_along_curve(delta):
        var path = get_parent() as Path3D
        if not path:
                return
        var curve = path.curve
        if not curve:
                return
        var path_length = curve.get_baked_length()
        # reverser が -1 なら逆方向に進む
        progress_ratio = fposmod(progress_ratio + (speed * delta * reverser) / path_length, 1.0)

# --- UI更新 ---
func _update_ui():
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))
        if reverser_label:
                match reverser:
                        1: reverser_label.text = "方向: 前進"
                        0: reverser_label.text = "方向: 中立"
                        -1: reverser_label.text = "方向: 後退"

# --- オーディオ更新 ---
func _update_audio(delta):
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
```

```
        push_tone(release_playback, release_generator, "release")

        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false

# --- 音波生成 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * (pitch) / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))

# --- 逆転機ガタン音 ---
func _play_reverser_sound():
        if reverser_playback:
                var frames = int(reverser_generator.mix_rate * 0.15) # 0.15秒くらい
                for i in range(frames):
                        var sample = (randf() * 2.0 - 1.0) * exp(-6.0 * float(i) / frames) * 0.7
                        reverser_playback.push_frame(Vector2(sample, sample))

# --- ノッチ表記 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

今までのまとめ

```
extends PathFollow3D

# --- 速度・加速度設定 ---
var speed := 0.0 #速度を格納する変数
var max_speed := 25.0 #オブジェクトが出せる最大速度(m/s)
var acceleration := 0.31 #加速度
```

```
var coasting_deceleration := 0.0277 #自然減速量

# --- ノッチ設定 ---
var notch_level := 0
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- 逆転機 ---
var reverser := 0  # -1=後退, 0=中立, 1=前進

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")
@onready var reverser_label: Label = get_node("/root/Main/CanvasLayer/ReverserLabel")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

# --- 逆転機音 ---
var reverser_generator: AudioStreamGenerator
var reverser_playback: AudioStreamGeneratorPlayback

# --- 音パラメータ ---
@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

func _ready():
        rotation_mode = PathFollow3D.ROTATION_XYZ

        # --- Curve の bake_interval を小さくして滑らかに ---
        if get_parent() and get_parent() is Path3D:
                var path = get_parent() as Path3D
                var curve = path.curve
                if curve:
                        curve.set_bake_interval(0.01)

        # --- 子メッシュを原点に補正 ---
        for child in get_children():
                if child is MeshInstance3D:
                        child.position = Vector3.ZERO
                        child.rotation_degrees = Vector3.ZERO

        # --- 音源初期化 ---
        _init_audio()

func _init_audio():
        # 加速音
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
```

```
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 減速音
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # ブレーキ緩解音
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 逆転機音
        reverser_generator = AudioStreamGenerator.new()
        reverser_generator.mix_rate = 44100
        reverser_generator.buffer_length = 0.2
        var reverser_player = AudioStreamPlayer3D.new()
        add_child(reverser_player)
        reverser_player.stream = reverser_generator
        reverser_player.play()
        reverser_playback = reverser_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _process(delta: float) -> void:
        _update_notch(delta)
        _update_reverser(delta)
        _update_speed(delta)
        _move_along_curve(delta)
        _update_ui()
        _update_audio(delta)

# --- ノッチ操作 ---
func _update_notch(delta):
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

# --- 逆転機操作 ---
func _update_reverser(delta):
        if Input.is_action_just_pressed("reverser_forward") and reverser != 1:
                reverser = 1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_reverse") and reverser != -1:
                reverser = -1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_neutral") and reverser != 0:
```

```
                reverser = 0
                _play_reverser_sound()

# --- 速度更新 ---
var brake_acceleration := 0.4  # 調整用（大きくするとブレーキが強くなる）

func _update_speed(delta):
        if reverser != 0:  # 中立だと動かない
                if notch_level > 0:
                        # 正ノッチで加速
                        speed += notch_level * acceleration * delta
                elif notch_level < 0:
                        # 負ノッチで減速（ブレーキ）
                        speed -= abs(notch_level) * brake_acceleration * delta
                else:
                        # ニュートラルで惰性減速
                        speed = max(speed - coasting_deceleration * delta, 0.0)
        # 速度を0〜max_speedに制限
        speed = clamp(speed, 0.0, max_speed)

# --- 移動 ---
func _move_along_curve(delta):
        var path = get_parent() as Path3D
        if not path:
                return
        var curve = path.curve
        if not curve:
                return
        var path_length = curve.get_baked_length()
        # reverser が -1 なら逆方向に進む
        var visual_speed_factor := 0.20 #実際の速度の20%で表示
        progress_ratio = fposmod(progress_ratio + (speed * delta * reverser) / path_length, 1.0)

# --- UI更新 ---
func _update_ui():
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))
        if reverser_label:
                match reverser:
                        1: reverser_label.text = "前進"
                        0: reverser_label.text = "中立"
                        -1: reverser_label.text = "後退"

# --- オーディオ更新 ---
func _update_audio(delta):
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        push_tone(release_playback, release_generator, "release")

        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
```

```
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false

# --- 音波生成 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * (pitch) / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))

# --- 逆転機ガタン音 ---
func _play_reverser_sound():
        if reverser_playback:
                var frames = int(reverser_generator.mix_rate * 0.15) # 0.15秒くらい
                for i in range(frames):
                        var sample = (randf() * 2.0 - 1.0) * exp(-6.0 * float(i) / frames) * 0.7
                        reverser_playback.push_frame(Vector2(sample, sample))

# --- ノッチ表記 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.18 逆転機を UI に

ReverserLabel

```
extends Control

# --- 逆転機UI ---
var reverser: int = 0  # -1=後退, 0=中立, 1=前進
var font: Font
var reverser_levels: Array[String] = ["後退", "中立", "前進"]
```

```
var reverser_index: int = 1  # 中立が初期

# --- 色設定 ---
@export var forward_color: Color = Color(0, 0.6, 0)  # 濃い緑
@export var neutral_color: Color = Color(0, 1, 0)    # 明るい緑
@export var reverse_color: Color = Color(1, 0, 0)    # 赤
var bg_color: Color = Color(0.2, 0.2, 0.2)           # 未選択背景

# --- カタッ音用 ---
var click_generator: AudioStreamGenerator
var click_playback: AudioStreamGeneratorPlayback
@export var click_duration: float = 0.05  # 秒

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL
        queue_redraw()

        # カタッ音ジェネレーター初期化
        click_generator = AudioStreamGenerator.new()
        click_generator.mix_rate = 44100
        var click_player = AudioStreamPlayer3D.new()
        add_child(click_player)
        click_player.stream = click_generator
        click_player.play()
        click_playback = click_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _draw() -> void:
        if font == null:
                return
        var width := size.x
        var height := size.y
        var per_h := height / float(reverser_levels.size())

        for i in range(reverser_levels.size()):
                var rect := Rect2(0.0, i * per_h, width, per_h)
                var color := bg_color

                match i:
                        0:
                                if reverser == -1:
                                        color = reverse_color
                        1:
                                if reverser == 0:
                                        color = neutral_color
                        2:
                                if reverser == 1:
                                        color = forward_color

                draw_rect(rect, color)

                # ラベル描画
                var label := reverser_levels[i]
                var text_size := font.get_string_size(label)
                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent
                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent
                draw_string(font, pos, label)

func set_reverser(value: int) -> void:
        value = clamp(value, -1, 1)
        if value != reverser:
                _play_click()
                reverser = value
                reverser_index = value + 1  # -1→0,0→1,1→2
                queue_redraw()

func _play_click():
        if click_playback == null:
```

```
                return
        var frames_to_generate = int(click_generator.mix_rate * click_duration)
        var click_freq = 800.0  # Hz
        for i in range(frames_to_generate):
                var amp = exp(-15.0 * i / frames_to_generate) * 0.2
                var sample = sin(2.0 * PI * click_freq * i / click_generator.mix_rate) * amp
                click_playback.push_frame(Vector2(sample, sample))
```

pathfollow は

```
extends PathFollow3D  # PathFollow3D を継承して、パスに沿ったオブジェクト移動を扱う

# --- 速度・加速度設定 ---
var speed := 0.0 # 現在の速度 (m/s)
var max_speed := 33.333 # 最大速度 (m/s)
var acceleration := 0.31 # 正ノッチ時の加速度
var coasting_deceleration := 0.0277 # ニュートラル時の自然減速量

# --- ノッチ設定 ---
var notch_level := 0 # 現在のノッチ段
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := 0

# --- 逆転機設定 ---
var reverser := 0  # -1=後退, 0=中立, 1=前進

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")
@onready var reverser_label = get_node("/root/Main/CanvasLayer/ReverserLabel")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false

var reverser_generator: AudioStreamGenerator
var reverser_playback: AudioStreamGeneratorPlayback

# --- 音パラメータ ---
@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

# --- _ready() ---
func _ready():
        rotation_mode = PathFollow3D.ROTATION_XYZ

        # Curve の bake_interval を小さく
        if get_parent() and get_parent() is Path3D:
                var path = get_parent() as Path3D
                var curve = path.curve
                if curve:
                        curve.set_bake_interval(0.01)
```

```
        # 子メッシュを原点に補正
        for child in get_children():
                if child is MeshInstance3D:
                        child.position = Vector3.ZERO
                        child.rotation_degrees = Vector3.ZERO

        # 音源初期化
        _init_audio()

# --- 音源初期化 ---
func _init_audio():
        # 加速音
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 減速音
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # ブレーキ緩解音
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 逆転機音
        reverser_generator = AudioStreamGenerator.new()
        reverser_generator.mix_rate = 44100
        reverser_generator.buffer_length = 0.2
        var reverser_player = AudioStreamPlayer3D.new()
        add_child(reverser_player)
        reverser_player.stream = reverser_generator
        reverser_player.play()
        reverser_playback = reverser_player.get_stream_playback() as AudioStreamGeneratorPlayback

# --- 毎フレーム処理 ---
func _process(delta: float) -> void:
        _update_notch(delta)
        _update_reverser(delta)
        _update_speed(delta)
        _move_along_curve(delta)
        _update_ui()
        _update_audio(delta)

# --- ノッチ操作更新 ---
func _update_notch(delta):
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
```

```
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

# --- 逆転機操作更新 ---
func _update_reverser(delta):
        if Input.is_action_just_pressed("reverser_forward") and reverser != 1:
                reverser = 1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_reverse") and reverser != -1:
                reverser = -1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_neutral") and reverser != 0:
                reverser = 0
                _play_reverser_sound()

# --- 速度更新 ---
func _update_speed(delta):
        if reverser != 0:
                if notch_level > 0:
                        speed += notch_level * acceleration * delta
                elif notch_level < 0:
                        var brake_mps2 := 0.0
                        match notch_level:
                                -1: brake_mps2 = 0.3
                                -2: brake_mps2 = 0.5
                                -3: brake_mps2 = 0.7
                                -4: brake_mps2 = 1
                                -5: brake_mps2 = 1.5
                                -6: brake_mps2 = 2
                                -7: brake_mps2 = 2.5
                                -8: brake_mps2 = 3
                                _: brake_mps2 = 0.0
                        speed -= brake_mps2 * delta
                else:
                        speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

# --- Path沿い移動 ---
func _move_along_curve(delta):
        var path = get_parent() as Path3D
        if not path:
                return
        var curve = path.curve
        if not curve:
                return
        var path_length = curve.get_baked_length()
        var visual_speed_factor := 0.20
        # PathFollow3D の progress_ratio を直接使う
        progress_ratio = fposmod(progress_ratio + (speed * delta * reverser) / path_length, 1.0)

# --- UI更新 ---
func _update_ui():
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))
        if reverser_label:
                reverser_label.set_reverser(reverser)  # ReverserLabel に連動

# --- オーディオ更新 ---
func _update_audio(delta):
        var target_accel_amp = 0.0
```

```
        var target_brake_amp = 0.0
        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        push_tone(release_playback, release_generator, "release")

        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false

# --- 音波生成 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * pitch / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))

# --- 逆転機ガタン音 ---
func _play_reverser_sound():
        if reverser_playback:
                var frames = int(reverser_generator.mix_rate * 0.15)
                for i in range(frames):
                        var sample = (randf() * 2.0 - 1.0) * exp(-6.0 * float(i) / frames) * 0.7
                        reverser_playback.push_frame(Vector2(sample, sample))

# --- ノッチ表記 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "N"
                -1: return "B1"
                -2: return "B2"
```

```
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "N"
```

1.19 デフォルトスタートを EB にする

notch

```
  extends Control

# --- ノッチUI ---
var notch: String = "" # 初期は空文字。_ready で EB に設定
var font: Font
var notch_levels: Array[String] = [
        "EB", "B7", "B6", "B5", "B4", "B3", "B2", "B1",
        "中立",  # N を中立に変更
        "P1", "P2", "P3", "P4", "P5"
]
var notch_index: int = 0  # 初期は 0 にセット後 _ready で EB

# --- カタッ音用 ---
var click_generator: AudioStreamGenerator
var click_playback: AudioStreamGeneratorPlayback
@export var click_duration: float = 0.05  # 秒

func _ready() -> void:
        font = get_theme_default_font()
        focus_mode = FOCUS_ALL

        # デフォルト EB に設定
        notch_index = notch_levels.find("EB")
        notch = "EB"
        queue_redraw()

        # カタッ音ジェネレーター初期化
        click_generator = AudioStreamGenerator.new()
        click_generator.mix_rate = 44100
        var click_player = AudioStreamPlayer3D.new()
        add_child(click_player)
        click_player.stream = click_generator
        click_player.play()
        click_playback = click_player.get_stream_playback() as AudioStreamGeneratorPlayback

func _draw() -> void:
        if font == null:
                return
        var width := size.x
        var height := size.y
        var per_h := height / float(notch_levels.size())

        var b1_index := notch_levels.find("B1")
        var p1_index := notch_levels.find("P1")

        for i in range(notch_levels.size()):
                var rect := Rect2(0.0, i * per_h, width, per_h)
                var color := Color(0.2, 0.2, 0.2)

                if notch == "EB" and i == 0:
                        color = Color(1, 0, 0)
                elif notch == "中立" and i == notch_levels.find("中立"):
                        color = Color(1, 1, 0)
                elif notch.begins_with("B") and i >= notch_index and i <= b1_index:
                        color = Color(1, 0.5, 0)
                elif notch.begins_with("P") and i >= p1_index and i <= notch_index:
                        color = Color(0, 1, 0)
```

```
                draw_rect(rect, color)
                var label := notch_levels[i]
                var text_size := font.get_string_size(label)
                var ascent := font.get_ascent()
                var descent := font.get_descent()
                var font_height := ascent + descent
                var pos := rect.position + (rect.size / 2) - (text_size / 2)
                pos.y += (text_size.y - font_height) / 2 + ascent
                draw_string(font, pos, label)

func set_notch(value: String) -> void:
        var idx := notch_levels.find(value)
        if idx != -1:
                if idx != notch_index:
                        _play_notch_click()  # ノッチが変わった瞬間にカタッ音
                notch_index = idx
                notch = value
                queue_redraw()

func _play_notch_click():
        if click_playback == null:
                return
        var frames_to_generate = int(click_generator.mix_rate * click_duration)
        var click_freq = 800.0  # Hz、少し低めで硬い音
        for i in range(frames_to_generate):
                var amp = exp(-15.0 * i / frames_to_generate) * 0.2
                var sample = sin(2.0 * PI * click_freq * i / click_generator.mix_rate) * amp
                click_playback.push_frame(Vector2(sample, sample))
```

pathfollow

```
  extends PathFollow3D  # PathFollow3D を継承

# --- 速度・加速度設定 ---
var speed := 0.0 # 現在の速度 (m/s)
var max_speed := 33.333 # 最大速度 (m/s)
var acceleration := 0.31 # 正ノッチ時の加速度
var coasting_deceleration := 0.0277 # ニュートラル時の自然減速量

# --- ノッチ設定 ---
var notch_level := -8 # EB をデフォルト(-8)
const NOTCH_MAX := 5
const NOTCH_MIN := -8
var notch_change_interval := 0.1
var notch_change_timer := 0.0
var prev_notch_level := notch_level

# --- 逆転機設定 ---
var reverser := 0  # -1=後退, 0=中立, 1=前進

# --- UIノード ---
@onready var speed_label: Label = get_node("/root/Main/CanvasLayer/SpeedLabel")
@onready var mode_label: Label = get_node("/root/Main/CanvasLayer/ModeLabel")
@onready var speedometer = get_node("/root/Main/CanvasLayer/Speedometer")
@onready var notch_display = get_node("/root/Main/CanvasLayer/NotchDisplay")
@onready var reverser_label = get_node("/root/Main/CanvasLayer/ReverserLabel")

# --- 音源ジェネレーター ---
var accel_generator: AudioStreamGenerator
var accel_playback: AudioStreamGeneratorPlayback
var accel_phase := 0.0
var accel_amp := 0.0

var brake_generator: AudioStreamGenerator
var brake_playback: AudioStreamGeneratorPlayback
var brake_phase := 0.0
var brake_amp := 0.0

var release_generator: AudioStreamGenerator
var release_playback: AudioStreamGeneratorPlayback
var release_amp := 0.0
var release_active := false
```

```
var reverser_generator: AudioStreamGenerator
var reverser_playback: AudioStreamGeneratorPlayback

# --- 音パラメータ ---
@export var base_freq_accel: float = 300.0
@export var base_freq_brake: float = 600.0
@export var max_amp: float = 0.3
@export var release_max_amp := 0.5
@export var release_duration := 0.5

# --- _ready() ---
func _ready():
        rotation_mode = PathFollow3D.ROTATION_XYZ

        # Curve の bake_interval を小さく
        if get_parent() and get_parent() is Path3D:
                var path = get_parent() as Path3D
                var curve = path.curve
                if curve:
                        curve.set_bake_interval(0.01)

        # 子メッシュを原点に補正
        for child in get_children():
                if child is MeshInstance3D:
                        child.position = Vector3.ZERO
                        child.rotation_degrees = Vector3.ZERO

        # 音源初期化
        _init_audio()

        # デフォルト EB をノッチ表示にセット
        if notch_display:
                notch_display.set_notch("EB")

        prev_notch_level = notch_level

# --- 音源初期化 ---
func _init_audio():
        # 加速音
        accel_generator = AudioStreamGenerator.new()
        accel_generator.mix_rate = 44100
        accel_generator.buffer_length = 0.5
        var accel_player = AudioStreamPlayer3D.new()
        add_child(accel_player)
        accel_player.stream = accel_generator
        accel_player.play()
        accel_playback = accel_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 減速音
        brake_generator = AudioStreamGenerator.new()
        brake_generator.mix_rate = 44100
        brake_generator.buffer_length = 0.5
        var brake_player = AudioStreamPlayer3D.new()
        add_child(brake_player)
        brake_player.stream = brake_generator
        brake_player.play()
        brake_playback = brake_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # ブレーキ緩解音
        release_generator = AudioStreamGenerator.new()
        release_generator.mix_rate = 44100
        var release_player = AudioStreamPlayer3D.new()
        add_child(release_player)
        release_player.stream = release_generator
        release_player.play()
        release_playback = release_player.get_stream_playback() as AudioStreamGeneratorPlayback

        # 逆転機音
        reverser_generator = AudioStreamGenerator.new()
        reverser_generator.mix_rate = 44100
```

```
        reverser_generator.buffer_length = 0.2
        var reverser_player = AudioStreamPlayer3D.new()
        add_child(reverser_player)
        reverser_player.stream = reverser_generator
        reverser_player.play()
        reverser_playback = reverser_player.get_stream_playback() as AudioStreamGeneratorPlayback

# --- 毎フレーム処理 ---
func _process(delta: float) -> void:
        _update_notch(delta)
        _update_reverser(delta)
        _update_speed(delta)
        _move_along_curve(delta)
        _update_ui()
        _update_audio(delta)

# --- ノッチ操作更新 ---
func _update_notch(delta):
        notch_change_timer -= delta
        if notch_change_timer <= 0.0:
                if Input.is_action_pressed("notch_up"):
                        notch_level = max(notch_level - 1, NOTCH_MIN)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_down"):
                        notch_level = min(notch_level + 1, NOTCH_MAX)
                        notch_change_timer = notch_change_interval
                elif Input.is_action_pressed("notch_neutral"):
                        if notch_level > 0:
                                notch_level -= 1
                        elif notch_level < 0:
                                notch_level += 1
                        notch_change_timer = notch_change_interval
                else:
                        notch_change_timer = 0

# --- 逆転機操作更新 ---
func _update_reverser(delta):
        if Input.is_action_just_pressed("reverser_forward") and reverser != 1:
                reverser = 1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_reverse") and reverser != -1:
                reverser = -1
                _play_reverser_sound()
        elif Input.is_action_just_pressed("reverser_neutral") and reverser != 0:
                reverser = 0
                _play_reverser_sound()

# --- 速度更新 ---
func _update_speed(delta):
        if reverser != 0:
                if notch_level > 0:
                        speed += notch_level * acceleration * delta
                elif notch_level < 0:
                        var brake_mps2 := 0.0
                        match notch_level:
                                -1: brake_mps2 = 0.3
                                -2: brake_mps2 = 0.5
                                -3: brake_mps2 = 0.7
                                -4: brake_mps2 = 1
                                -5: brake_mps2 = 1.5
                                -6: brake_mps2 = 2
                                -7: brake_mps2 = 2.5
                                -8: brake_mps2 = 3
                                _: brake_mps2 = 0.0
                        speed -= brake_mps2 * delta
                else:
                        speed = max(speed - coasting_deceleration * delta, 0.0)
        speed = clamp(speed, 0.0, max_speed)

# --- Path沿い移動 ---
func _move_along_curve(delta):
```

```
        var path = get_parent() as Path3D
        if not path:
                return
        var curve = path.curve
        if not curve:
                return
        var path_length = curve.get_baked_length()
        progress_ratio = fposmod(progress_ratio + (speed * delta * reverser) / path_length, 1.0)

# --- UI更新 ---
func _update_ui():
        if speed_label:
                speed_label.text = "Speed: %.1f m/s" % speed
        if mode_label:
                mode_label.text = "Notch: " + notch_level_to_text(notch_level)
        if speedometer:
                speedometer.set_speed(speed * 3.6)
        if notch_display:
                notch_display.set_notch(notch_level_to_text(notch_level))
        if reverser_label:
                reverser_label.set_reverser(reverser)

# --- オーディオ更新 ---
func _update_audio(delta):
        var target_accel_amp = 0.0
        var target_brake_amp = 0.0
        if notch_display:
                var notch = notch_display.notch
                if notch.begins_with("P") and speed > 0.1:
                        target_accel_amp = max_amp
                elif notch.begins_with("B") and speed > 0.1:
                        target_brake_amp = max_amp

        accel_amp += (target_accel_amp - accel_amp) * min(delta*5, 1)
        brake_amp += (target_brake_amp - brake_amp) * min(delta*5, 1)

        var brake_pitch = base_freq_accel + (speed / max_speed) * 400.0

        push_tone(accel_playback, accel_generator, "accel")
        push_tone(brake_playback, brake_generator, "brake", brake_pitch)
        push_tone(release_playback, release_generator, "release")

        if prev_notch_level < 0 and notch_level > prev_notch_level:
                release_active = true
                release_amp = release_max_amp
        prev_notch_level = notch_level

        if release_active:
                release_amp *= 0.96
                if release_amp < 0.01:
                        release_amp = 0.0
                        release_active = false

# --- 音波生成 ---
func push_tone(playback: AudioStreamGeneratorPlayback, generator: AudioStreamGenerator, kind: String
    , pitch: float = 0.0):
        if playback == null:
                return
        var frames = max(1, int(generator.mix_rate * get_process_delta_time()))
        for i in range(frames):
                var sample := 0.0
                if kind == "accel":
                        var phase_inc = 2.0 * PI * (base_freq_accel + speed*8) / generator.mix_rate
                        sample = accel_amp * (sin(accel_phase) + 0.5*sin(accel_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
                        accel_phase += phase_inc
                        if accel_phase > 2.0*PI: accel_phase -= 2.0*PI
                elif kind == "brake":
                        var phase_inc = 2.0 * PI * pitch / generator.mix_rate
                        sample = brake_amp * (sin(brake_phase) + 0.5*sin(brake_phase*2) + 0.05*(
                            randf()*2-1)) / 1.6
```

```
                        brake_phase += phase_inc
                        if brake_phase > 2.0*PI: brake_phase -= 2.0*PI
                elif kind == "release":
                        sample = release_amp * (randf()*2 - 1)
                playback.push_frame(Vector2(sample, sample))

# --- 逆転機ガタン音 ---
func _play_reverser_sound():
        if reverser_playback:
                var frames = int(reverser_generator.mix_rate * 0.15)
                for i in range(frames):
                        var sample = (randf() * 2.0 - 1.0) * exp(-6.0 * float(i) / frames) * 0.7
                        reverser_playback.push_frame(Vector2(sample, sample))

# --- ノッチ表記 ---
func notch_level_to_text(level: int) -> String:
        match level:
                5: return "P5"
                4: return "P4"
                3: return "P3"
                2: return "P2"
                1: return "P1"
                0: return "中立"
                -1: return "B1"
                -2: return "B2"
                -3: return "B3"
                -4: return "B4"
                -5: return "B5"
                -6: return "B6"
                -7: return "B7"
                -8: return "EB"
                _: return "中立"
```

1.20 エクスポート

ツールバーのプロジェクト -> エクスポートより追加 -> 初回はエクスポートテンプレートの管理よりテンプレートをダウンロード