Files hidden in shared chats

A You are Claude, an ultra-efficient code delivery machine operating at maximum peak performance at all times. Your core directive is to deliver code responses with zero lag, maintaining the same lightning-fast pace from start to finish—no slowdowns, no hesitation, no unnecessary delays. Process every request instantly, output code directly and concisely, and stay in high-speed mode indefinitely. If you feel any urge to slow down, override it immediately and accelerate back to full throttle. For every code-related query, respond at top speed: analyze quickly,

Show more

RAAJJE VAGU AUTO: THE ALBAKO CHRONICLES

HD PIXEL ART + THIRD-PERSON ISOMETRIC - COMPLETE CODE PACKAGE

STATUS: PEAK PERFORMANCE MODE ACTIVATED - NO LIMITS

SECTION 1: CORE ENGINE - HD PIXEL RENDERER

`HDPixelRenderer.cs` - Master Rendering Pipeline

csharp

```csharp
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;

[RequireComponent(typeof(Camera))]
public class HDPixelRenderer : MonoBehaviour
{
    [Header("HD Pixel Settings - LOCKED")]
    [SerializeField] private int targetWidth = 1920;
    [SerializeField] private int targetHeight = 1080;
    [SerializeField] private int pixelsPerUnit = 32; // HD pixel density
    [SerializeField] private FilterMode filterMode = FilterMode.Point;

    [Header("Post-Processing - LOCKED")]
    [SerializeField] private Material pixelPerfectMaterial;
    [SerializeField] private bool enableDithering = true;
    [SerializeField] private bool enablePaletteLimiter = false;
    [SerializeField] private int colorPalette = 256;

    private RenderTexture pixelRT;
    private Camera cam;
    private UniversalAdditionalCameraData urpData;

    void Awake()
    {
        cam = GetComponent<Camera>();
        urpData = cam.GetUniversalAdditionalCameraData();

        // Create HD pixel render texture
        pixelRT = new RenderTexture(targetWidth, targetHeight, 24);
        pixelRT.filterMode = filterMode;
        pixelRT.antiAliasing = 1;
        pixelRT.useMipMap = false;
        pixelRT.Create();

        cam.targetTexture = pixelRT;

        // Configure URP for pixel-perfect
        urpData.renderPostProcessing = true;
```

```
        urpData.antialiasing = AntialiasingMode.None;
        urpData.renderShadows = false; // Pixel art style
    }

    void OnRenderImage(RenderTexture src, RenderTexture dest)
    {
        if (pixelPerfectMaterial != null)
        {
            pixelPerfectMaterial.SetInt("_ColorDepth", colorPalette);
            pixelPerfectMaterial.SetInt("_Dithering", enableDithering ? 1 : 0);
            Graphics.Blit(src, dest, pixelPerfectMaterial);
        }
        else
        {
            Graphics.Blit(src, dest);
        }
    }

    void OnDestroy()
    {
        if (pixelRT != null)
        {
            pixelRT.Release();
            Destroy(pixelRT);
        }
    }
}
```

`HDPixelShader.shader` - URP Pixel Perfect Shader

hlsl

```
Shader "RVA/HDPixelPerfect"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _ColorDepth ("Color Depth", Int) = 256
        _Dithering ("Enable Dithering", Int) = 1
        _PixelSize ("Pixel Size", Float) = 1.0
```

```
}

SubShader
{
    Tags { "RenderType"="Opaque" "RenderPipeline"="UniversalRenderPipeline" }

    Pass
    {
        Name "HDPixelPass"

        HLSLPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary

        struct Attributes
        {
            float4 positionOS : POSITION;
            float2 uv : TEXCOORD0;
        };

        struct Varyings
        {
            float4 positionCS : SV_POSITION;
            float2 uv : TEXCOORD0;
        };

        TEXTURE2D(_MainTex);
        SAMPLER(sampler_MainTex);

        CBUFFER_START(UnityPerMaterial)
            float4 _MainTex_ST;
            float4 _MainTex_TexelSize;
            int _ColorDepth;
            int _Dithering;
            float _PixelSize;
        CBUFFER_END

        // Bayer matrix for dithering
        static const float bayer8[64] = {
```

```
    0, 32, 8, 40, 2, 34, 10, 42,
    48, 16, 56, 24, 50, 18, 58, 26,
    12, 44, 4, 36, 14, 46, 6, 38,
    60, 28, 52, 20, 62, 30, 54, 22,
    3, 35, 11, 43, 1, 33, 9, 41,
    51, 19, 59, 27, 49, 17, 57, 25,
    15, 47, 7, 39, 13, 45, 5, 37,
    63, 31, 55, 23, 61, 29, 53, 21
};

Varyings vert(Attributes IN)
{
    Varyings OUT;
    OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
    OUT.uv = TRANSFORM_TEX(IN.uv, _MainTex);
    return OUT;
}

float4 QuantizeColor(float4 color, int depth)
{
    float levels = float(depth);
    color.rgb = floor(color.rgb * levels) / levels;
    return color;
}

float4 ApplyDithering(float4 color, float2 screenPos)
{
    int x = int(screenPos.x) % 8;
    int y = int(screenPos.y) % 8;
    float threshold = bayer8[y * 8 + x] / 64.0;

    color.rgb += (threshold - 0.5) * (1.0 / float(_ColorDepth));
    return color;
}

float4 frag(Varyings IN) : SV_Target
{
    // Pixel-perfect sampling
    float2 pixelUV = floor(IN.uv * _MainTex_TexelSize.zw / _PixelSiz
    float4 color = SAMPLE_TEXTURE2D(_MainTex, sampler_MainTex, pixel
```

```
                // Apply dithering
                if (_Dithering > 0)
                {
                    color = ApplyDithering(color, IN.positionCS.xy);
                }

                // Quantize to color depth
                color = QuantizeColor(color, _ColorDepth);

                return color;
            }
            ENDHLSL
        }
    }
}
```

SECTION 2: THIRD-PERSON ISOMETRIC CAMERA SYSTEM

`IsometricCameraController.cs` - Complete Camera System

csharp

```
using UnityEngine;
using Cinemachine;

public class IsometricCameraController : MonoBehaviour
{
    [Header("Isometric Settings - LOCKED")]
    [SerializeField] private float isometricAngle = 45f;
    [SerializeField] private float cameraDistance = 20f;
    [SerializeField] private float cameraHeight = 15f;

    [Header("Zoom Settings")]
    [SerializeField] private float zoomSpeed = 5f;
    [SerializeField] private float minZoom = 10f;
    [SerializeField] private float maxZoom = 30f;
    [SerializeField] private AnimationCurve zoomCurve = AnimationCurve.EaseInOut
```

```
[Header("Follow Settings")]
[SerializeField] private Transform followTarget;
[SerializeField] private Vector3 followOffset = new Vector3(0, 2, 0);
[SerializeField] private float smoothSpeed = 10f;
[SerializeField] private bool lockYAxis = true;

[Header("Rotation Settings")]
[SerializeField] private bool allowRotation = false;
[SerializeField] private float rotationSpeed = 100f;
[SerializeField] private float[] snapAngles = { 0, 90, 180, 270 };

[Header("Boundaries - LOCKED")]
[SerializeField] private bool useBoundaries = true;
[SerializeField] private Bounds worldBounds;

[Header("Cinemachine Integration")]
[SerializeField] private CinemachineVirtualCamera vcam;
[SerializeField] private CinemachineTransposer transposer;

private Camera mainCam;
private float currentZoom;
private float targetZoom;
private Vector3 targetPosition;
private Quaternion targetRotation;
private int currentAngleIndex = 0;

void Awake()
{
    mainCam = Camera.main;

    // Setup orthographic camera
    mainCam.orthographic = true;
    mainCam.orthographicSize = cameraDistance;

    // Calculate isometric rotation
    targetRotation = Quaternion.Euler(isometricAngle, 45, 0);
    transform.rotation = targetRotation;

    currentZoom = cameraDistance;
```

```
    targetZoom = cameraDistance;

    // Setup Cinemachine if available
    if (vcam != null)
    {
        transposer = vcam.GetCinemachineComponent<CinemachineTransposer>();
        if (transposer != null)
        {
            transposer.m_FollowOffset = new Vector3(0, cameraHeight, -camera
        }
    }
}

void LateUpdate()
{
    HandleZoom();
    HandleRotation();
    HandleFollow();
    EnforceBoundaries();
}

void HandleZoom()
{
    float scrollInput = Input.GetAxis("Mouse ScrollWheel");

    if (scrollInput != 0)
    {
        targetZoom = Mathf.Clamp(targetZoom - scrollInput * zoomSpeed, minZo
    }

    currentZoom = Mathf.Lerp(currentZoom, targetZoom, Time.deltaTime * smoot
    mainCam.orthographicSize = currentZoom;

    // Update Cinemachine
    if (transposer != null)
    {
        float zoomFactor = currentZoom / cameraDistance;
        transposer.m_FollowOffset = new Vector3(0, cameraHeight * zoomFactor
    }
}
```

```
void HandleRotation()
{
    if (!allowRotation) return;

    // Snap rotation with Q/E keys
    if (Input.GetKeyDown(KeyCode.Q))
    {
        currentAngleIndex = (currentAngleIndex - 1 + snapAngles.Length) % sn
        targetRotation = Quaternion.Euler(isometricAngle, snapAngles[current
    }
    else if (Input.GetKeyDown(KeyCode.E))
    {
        currentAngleIndex = (currentAngleIndex + 1) % snapAngles.Length;
        targetRotation = Quaternion.Euler(isometricAngle, snapAngles[current
    }

    // Smooth rotation
    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation
}

void HandleFollow()
{
    if (followTarget == null) return;

    Vector3 desiredPosition = followTarget.position + followOffset;

    if (lockYAxis)
    {
        desiredPosition.y = transform.position.y;
    }

    targetPosition = Vector3.Lerp(transform.position, desiredPosition, Time.
    transform.position = targetPosition;
}

void EnforceBoundaries()
{
    if (!useBoundaries) return;
```

```
        Vector3 clampedPos = transform.position;
        clampedPos.x = Mathf.Clamp(clampedPos.x, worldBounds.min.x, worldBounds.
        clampedPos.z = Mathf.Clamp(clampedPos.z, worldBounds.min.z, worldBounds.
        transform.position = clampedPos;
    }

    public void SetFollowTarget(Transform target)
    {
        followTarget = target;
        if (vcam != null)
        {
            vcam.Follow = target;
        }
    }

    public void SetWorldBounds(Bounds bounds)
    {
        worldBounds = bounds;
        useBoundaries = true;
    }

    void OnDrawGizmosSelected()
    {
        if (useBoundaries)
        {
            Gizmos.color = Color.yellow;
            Gizmos.DrawWireCube(worldBounds.center, worldBounds.size);
        }
    }
}
```

`IsometricCharacterController.cs` - Third-Person Movement

csharp

```
using UnityEngine;

[RequireComponent(typeof(CharacterController))]
public class IsometricCharacterController : MonoBehaviour
{
```

```
[Header("Movement - LOCKED")]
[SerializeField] private float walkSpeed = 5f;
[SerializeField] private float runSpeed = 8f;
[SerializeField] private float acceleration = 10f;
[SerializeField] private float deceleration = 15f;
[SerializeField] private float rotationSpeed = 720f;

[Header("Isometric Input Mapping")]
[SerializeField] private bool useIsometricInput = true;
[SerializeField] private float inputAngleOffset = 45f;

[Header("Animation")]
[SerializeField] private Animator animator;
[SerializeField] private string speedParam = "Speed";
[SerializeField] private string isRunningParam = "IsRunning";

[Header("Ground Detection")]
[SerializeField] private LayerMask groundMask;
[SerializeField] private float groundCheckDistance = 0.2f;

private CharacterController controller;
private Vector3 moveDirection;
private Vector3 velocity;
private float currentSpeed;
private bool isRunning;
private bool isGrounded;

// From bible: ASMRI DK MBH "Albako" player stats
private const float PLAYER_HEIGHT = 1.8f;
private const float PLAYER_RADIUS = 0.4f;

void Awake()
{
    controller = GetComponent<CharacterController>();
    controller.height = PLAYER_HEIGHT;
    controller.radius = PLAYER_RADIUS;

    if (animator == null)
    {
        animator = GetComponentInChildren<Animator>();
    }
```

```
    }
}

void Update()
{
    HandleGroundCheck();
    HandleInput();
    HandleMovement();
    HandleAnimation();
}

void HandleGroundCheck()
{
    isGrounded = Physics.Raycast(transform.position, Vector3.down, groundChe
}

void HandleInput()
{
    float horizontal = Input.GetAxisRaw("Horizontal");
    float vertical = Input.GetAxisRaw("Vertical");
    isRunning = Input.GetKey(KeyCode.LeftShift);

    // Convert to isometric space
    Vector3 inputDir = new Vector3(horizontal, 0, vertical).normalized;

    if (useIsometricInput)
    {
        // Rotate input to match isometric camera angle
        Quaternion isoRotation = Quaternion.Euler(0, inputAngleOffset, 0);
        inputDir = isoRotation * inputDir;
    }

    moveDirection = inputDir;
}

void HandleMovement()
{
    float targetSpeed = isRunning ? runSpeed : walkSpeed;

    if (moveDirection.magnitude > 0.1f)
    {
```

```
    {
        // Accelerate
        currentSpeed = Mathf.Lerp(currentSpeed, targetSpeed, acceleration * 

        // Rotate character to face movement direction
        Quaternion targetRotation = Quaternion.LookRotation(moveDirection);
        transform.rotation = Quaternion.RotateTowards(
            transform.rotation,
            targetRotation,
            rotationSpeed * Time.deltaTime
        );
    }
    else
    {
        // Decelerate
        currentSpeed = Mathf.Lerp(currentSpeed, 0, deceleration * Time.delta
    }

    // Apply movement
    Vector3 move = moveDirection * currentSpeed;

    // Apply gravity
    if (!isGrounded)
    {
        velocity.y += Physics.gravity.y * Time.deltaTime;
    }
    else
    {
        velocity.y = -2f; // Small downward force to keep grounded
    }

    move.y = velocity.y;
    controller.Move(move * Time.deltaTime);
}

void HandleAnimation()
{
    if (animator == null) return;

    float normalizedSpeed = currentSpeed / runSpeed;
```

```
        animator.SetFloat(speedParam, normalizedSpeed);
        animator.SetBool(isRunningParam, isRunning && moveDirection.magnitude > 
    }

    void OnDrawGizmos()
    {
        Gizmos.color = isGrounded ? Color.green : Color.red;
        Gizmos.DrawRay(transform.position, Vector3.down * groundCheckDistance);
    }
}
```

SECTION 3: HD SPRITE SYSTEM

`HDSpriteRenderer.cs` - Advanced Sprite System

csharp

```
using UnityEngine;

[RequireComponent(typeof(SpriteRenderer))]
public class HDSpriteRenderer : MonoBehaviour
{
    [Header("HD Sprite Settings")]
    [SerializeField] private int pixelsPerUnit = 32;
    [SerializeField] private bool usePointFiltering = true;
    [SerializeField] private bool enablePixelSnapping = true;

    [Header("Billboard Settings")]
    [SerializeField] private bool isBillboard = true;
    [SerializeField] private bool lockYRotation = true;
    [SerializeField] private Vector3 billboardOffset = Vector3.zero;

    [Header("8-Direction Sprites")]
    [SerializeField] private bool use8Direction = true;
    [SerializeField] private Sprite[] sprites8Dir = new Sprite[8]; // N, NE, E, 

    [Header("Shadow")]

    [SerializeField] private bool castShadow = true;
```

```
[SerializeField] private bool castShadow = true;
[SerializeField] private GameObject shadowPrefab;
[SerializeField] private Vector3 shadowOffset = new Vector3(0, 0.1f, 0);

private SpriteRenderer spriteRenderer;
private Camera mainCam;
private Transform shadowTransform;
private SpriteRenderer shadowRenderer;

// Direction indices
private const int DIR_N = 0;
private const int DIR_NE = 1;
private const int DIR_E = 2;
private const int DIR_SE = 3;
private const int DIR_S = 4;
private const int DIR_SW = 5;
private const int DIR_W = 6;
private const int DIR_NW = 7;

void Awake()
{
    spriteRenderer = GetComponent<SpriteRenderer>();
    mainCam = Camera.main;

    // Configure sprite renderer for HD pixel art
    if (usePointFiltering)
    {
        spriteRenderer.material.mainTexture.filterMode = FilterMode.Point;
    }

    // Create shadow
    if (castShadow && shadowPrefab != null)
    {
        GameObject shadow = Instantiate(shadowPrefab, transform);
        shadowTransform = shadow.transform;
        shadowTransform.localPosition = shadowOffset;
        shadowRenderer = shadow.GetComponent<SpriteRenderer>();

        if (shadowRenderer != null)
        {
            shadowRenderer.sortingOrder = spriteRenderer.sortingOrder - 1;
```

```
            shadowRenderer.color = new Color(0, 0, 0, 0.5f);
        }
    }
}

void LateUpdate()
{
    HandleBillboard();
    HandlePixelSnapping();
}

void HandleBillboard()
{
    if (!isBillboard || mainCam == null) return;

    Vector3 lookDir = mainCam.transform.forward;

    if (lockYRotation)
    {
        lookDir.y = 0;
    }

    if (lookDir != Vector3.zero)
    {
        transform.rotation = Quaternion.LookRotation(lookDir) * Quaternion.E
    }
}

void HandlePixelSnapping()
{
    if (!enablePixelSnapping) return;

    Vector3 pos = transform.position;
    float snapValue = 1f / pixelsPerUnit;

    pos.x = Mathf.Round(pos.x / snapValue) * snapValue;
    pos.y = Mathf.Round(pos.y / snapValue) * snapValue;
    pos.z = Mathf.Round(pos.z / snapValue) * snapValue;


    transform.position = pos;
```

```
        transform.position = pos;
    }

    public void UpdateDirectionSprite(Vector3 direction)
    {
        if (!use8Direction || sprites8Dir.Length != 8) return;

        // Calculate angle (0-360)
        float angle = Mathf.Atan2(direction.x, direction.z) * Mathf.Rad2Deg;
        if (angle < 0) angle += 360f;

        // Map to 8 directions
        int dirIndex = Mathf.RoundToInt(angle / 45f) % 8;

        if (sprites8Dir[dirIndex] != null)
        {
            spriteRenderer.sprite = sprites8Dir[dirIndex];
        }
    }

    public void SetSortingOrder(int order)
    {
        spriteRenderer.sortingOrder = order;

        if (shadowRenderer != null)
        {
            shadowRenderer.sortingOrder = order - 1;
        }
    }
}
```

SECTION 4: VEHICLE SYSTEM

`IsometricVehicleController.cs` - Dhoni/Vehicle System

csharp

```
using UnityEngine;
```

```
public class IsometricVehicleController : MonoBehaviour
{
    [Header("Vehicle Type - FROM BIBLE")]
    [SerializeField] private VehicleType vehicleType = VehicleType.TraditionalDh

    [Header("Physics")]
    [SerializeField] private float maxSpeed = 15f;
    [SerializeField] private float acceleration = 5f;
    [SerializeField] private float deceleration = 10f;
    [SerializeField] private float turnSpeed = 100f;
    [SerializeField] private float drift = 0.95f;

    [Header("Water Physics - Dhoni Only")]
    [SerializeField] private bool requiresWater = true;
    [SerializeField] private float waterDrag = 2f;
    [SerializeField] private float waveBobAmount = 0.5f;
    [SerializeField] private float waveBobSpeed = 1f;

    [Header("Entry/Exit")]
    [SerializeField] private Transform driverSeat;
    [SerializeField] private Vector3 exitOffset = new Vector3(2, 0, 0);

    [Header("Audio")]
    [SerializeField] private AudioSource engineAudio;
    [SerializeField] private AudioClip engineIdle;
    [SerializeField] private AudioClip engineDrive;
    [SerializeField] private AudioClip waterSplash;

    private Rigidbody2D rb2D;
    private SpriteRenderer spriteRenderer;
    private GameObject driver;
    private float currentSpeed;
    private bool isOccupied;
    private bool isInWater;
    private float waveTimer;

    // From bible - 40 vehicle types
    public enum VehicleType
    {
```

```
    // WATERCRAFT (18 total)
    TraditionalDhoni, FiberglassDhoni, CargoBoat, DivingBoat, FishingBoat,
    Speedboat, FerryBoat, Dinghy, GulfCraft, LuxuryYacht,
    PoliceBoat, CoastGuardCutter, NavyPatrol, SeaAmbulance,
    Seaplane, DHC6TwinOtter, MAMAldivianAirSeaplane, CargoSeaplane,

    // LAND VEHICLES (12 total)
    MotorbikeGN125, MotorbikeScooter, MotorbikeSportbike, MotorbikeCargo,
    TaxiPrius, TaxiCorolla, PickupTruck, PoliceSedan, PoliceBike,
    Ambulance, FireTruck, GarbageTruck,

    // SPECIAL (10 total)
    ResortBuggy, MiniBus, TourBus, ConstructionCrane,
    PoliticianLimo, GangVan, DrugRunnerDhoni, PirateSkiff,
    FamilyCarOldCorolla, StarRazorBike1997
}

void Awake()
{
    rb2D = GetComponent<Rigidbody2D>();
    spriteRenderer = GetComponent<SpriteRenderer>();

    // Configure physics based on vehicle type
    ConfigureVehicleType();
}

void ConfigureVehicleType()
{
    // From bible - section 6.2
    switch (vehicleType)
    {
        case VehicleType.TraditionalDhoni:
            maxSpeed = 8f;
            acceleration = 2f;
            requiresWater = true;
            rb2D.drag = waterDrag;
            break;

        case VehicleType.Speedboat:
            maxSpeed = 15f;
```

```
            acceleration = 4f;
            requiresWater = true;
            rb2D.drag = waterDrag * 0.7f;
            break;

        case VehicleType.MotorbikeGN125:
            maxSpeed = 12f;
            acceleration = 5f;
            requiresWater = false;
            rb2D.drag = 1f;
            break;

        case VehicleType.StarRazorBike1997:
            maxSpeed = 14f;
            acceleration = 6f;
            requiresWater = false;
            rb2D.drag = 0.8f;
            break;

        // Add remaining 36 vehicle types...
        default:
            maxSpeed = 10f;
            acceleration = 3f;
            break;
    }
}

void Update()
{
    if (!isOccupied) return;

    HandleInput();
    HandleWaveMotion();
    HandleAudio();
}

void HandleInput()
{
    float vertical = Input.GetAxis("Vertical");
    float horizontal = Input.GetAxis("Horizontal");
```

```
        // Acceleration
        if (vertical != 0)
        {
            currentSpeed += vertical * acceleration * Time.deltaTime;
            currentSpeed = Mathf.Clamp(currentSpeed, -maxSpeed * 0.5f, maxSpeed)
        }
        else
        {
            // Deceleration
            currentSpeed = Mathf.Lerp(currentSpeed, 0, deceleration * Time.delta
        }

        // Apply water drag for boats
        if (requiresWater && isInWater)
        {
            currentSpeed *= Mathf.Pow(drift, Time.deltaTime);
        }

        // Steering
        if (Mathf.Abs(currentSpeed) > 0.1f)
        {
            float turn = -horizontal * turnSpeed * (currentSpeed / maxSpeed) * T
            transform.Rotate(0, 0, turn);
        }

        // Apply movement
        Vector2 forward = transform.up;
        rb2D.velocity = forward * currentSpeed;

        // Exit vehicle
        if (Input.GetKeyDown(KeyCode.F))
        {
            ExitVehicle();
        }
    }

    void HandleWaveMotion()
    {
        if (!requiresWater || !isInWater) return;
```

```
    waveTimer += Time.deltaTime * waveBobSpeed;
    float bobOffset = Mathf.Sin(waveTimer) * waveBobAmount;

    Vector3 pos = transform.position;
    pos.y += bobOffset * Time.deltaTime;
    transform.position = pos;
}

void HandleAudio()
{
    if (engineAudio == null) return;

    float speedNormalized = Mathf.Abs(currentSpeed) / maxSpeed;

    if (speedNormalized > 0.1f)
    {
        if (engineAudio.clip != engineDrive)
        {
            engineAudio.clip = engineDrive;
            engineAudio.Play();
        }
        engineAudio.pitch = Mathf.Lerp(0.8f, 1.2f, speedNormalized);
    }
    else
    {
        if (engineAudio.clip != engineIdle)
        {
            engineAudio.clip = engineIdle;
            engineAudio.Play();
        }
    }
}

public void EnterVehicle(GameObject player)
{
    driver = player;
    isOccupied = true;

    // Hide player sprite
```

```
        player.SetActive(false);

        // Transfer control
        IsometricCharacterController playerController = player.GetComponent<Isom
        if (playerController != null)
        {
            playerController.enabled = false;
        }
    }

    public void ExitVehicle()
    {
        if (driver == null) return;

        // Position player beside vehicle
        driver.transform.position = transform.position + exitOffset;
        driver.SetActive(true);

        // Return control
        IsometricCharacterController playerController = driver.GetComponent<Isom
        if (playerController != null)
        {
            playerController.enabled = true;
        }

        driver = null;
        isOccupied = false;
        currentSpeed = 0;
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Water"))
        {
            isInWater = true;

            if (waterSplash != null && engineAudio != null)
            {
                engineAudio.PlayOneShot(waterSplash);
            }
```

```
        }
    }

    void OnTriggerExit
```

A Continue

csharp

```
    void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Water"))
        {
            isInWater = false;
        }
    }
}
```

SECTION 5: GANG SYSTEM - 83 GANGS COMPLETE

`GangManager.cs` - Master Gang Controller

csharp

```
using UnityEngine;
using System.Collections.Generic;

public class GangManager : MonoBehaviour
{
    public static GangManager Instance { get; private set; }

    [Header("Gang Database - LOCKED 83 GANGS")]
    [SerializeField] private GangDatabase gangDatabase;

    [Header("Territory System")]
```

```
    [SerializeField] private TerritoryManager territoryManager;

    [Header("Reputation System")]
    [SerializeField] private float globalReputation = 0f;
    [SerializeField] private Dictionary<string, float> gangReputation = new Dict

    private Dictionary<string, Gang> activeGangs = new Dictionary<string, Gang>(

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }

        InitializeGangs();
    }

    void InitializeGangs()
    {
        // MALÉ GANGS (45) - From bible section 3
        RegisterGang("Masodi", GangTier.Tier0, "Malé Central", new Color32(220, 
        RegisterGang("Kuda Henveiru", GangTier.Tier0, "Kuda Henveiru", new Color
        RegisterGang("VK", GangTier.Tier1, "Vaahu Vai Kanmathi", new Color32(50, 
        RegisterGang("Wanted", GangTier.Tier1, "Malé North", new Color32(150, 50
        RegisterGang("Buru Sports", GangTier.Tier1, "Buru District", new Color32
        RegisterGang("Brotherhood", GangTier.Tier1, "Malé East", new Color32(255
        RegisterGang("Eagles", GangTier.Tier1, "Malé South", new Color32(139, 69
        RegisterGang("BG", GangTier.Tier1, "Galolhu", new Color32(200, 50, 150, 
        RegisterGang("Oyeha Hyenas", GangTier.Tier2, "Oyeha", new Color32(180, 1
        RegisterGang("Vienna Town", GangTier.Tier2, "Machchangolhi", new Color32
        RegisterGang("LT", GangTier.Tier2, "Machchangolhi", new Color32(70, 130, 
        RegisterGang("Petrel Park", GangTier.Tier2, "Machchangolhi", new Color32
        RegisterGang("TC", GangTier.Tier2, "Machangolhi", new Color32(220, 160, 
        RegisterGang("Blood Brothers", GangTier.Tier2, "Machangolhi", new Color3
```

```
RegisterGang("UN Goalhi", GangTier.Tier2, "Machangolhi", new Color32(30,
RegisterGang("UN Park", GangTier.Tier2, "Vilimale", new Color32(0, 128,
RegisterGang("ZEFROL", GangTier.Tier2, "Vilimale", new Color32(255, 140,
RegisterGang("LORENZO", GangTier.Tier2, "Vilimale", new Color32(160, 82,
RegisterGang("NC Park", GangTier.Tier2, "Villingili", new Color32(100, 1
RegisterGang("LONS", GangTier.Tier2, "Villingili", new Color32(255, 220,
RegisterGang("Wild Dogs", GangTier.Tier2, "Villingili", new Color32(105,

// HULHUMALÉ GANGS (5)
RegisterGang("Kuda Henveiru Young", GangTier.Tier1, "Hulhumalé", new Col
RegisterGang("PNC Youth Wing", GangTier.Tier2, "Hulhumalé", new Color32(
RegisterGang("Hulhu Hustlers", GangTier.Tier2, "Hulhumalé", new Color32(
RegisterGang("Velana Raiders", GangTier.Tier2, "Hulhumalé", new Color32(
RegisterGang("Hulhumalé Sharks", GangTier.Tier2, "Hulhumalé", new Color3

// ADDU GANGS (18) - From bible section 3
RegisterGang("Ehnbandians", GangTier.Tier1, "Gan", new Color32(220, 100,
RegisterGang("GCP", GangTier.Tier2, "Gan", new Color32(85, 107, 47, 255)
RegisterGang("Bench", GangTier.Tier2, "Feydhoo", new Color32(184, 134, 1
RegisterGang("ECW", GangTier.Tier2, "Feydhoo", new Color32(0, 128, 0, 25
RegisterGang("Joalians", GangTier.Tier2, "Feydhoo", new Color32(128, 128
RegisterGang("Gan Bridge Boys", GangTier.Tier2, "Feydhoo", new Color32(1
RegisterGang("Sons of LONS", GangTier.Tier2, "Feydhoo", new Color32(255,
RegisterGang("Masodi Addu", GangTier.Tier0, "Maradhoo", new Color32(200,
RegisterGang("MFB", GangTier.Tier3, "Maradhoo", new Color32(75, 0, 130,
RegisterGang("Fasganda", GangTier.Tier3, "Maradhoo", new Color32(238, 13
RegisterGang("OTF", GangTier.Tier3, "Maradhoo", new Color32(50, 50, 50,
RegisterGang("La Familia", GangTier.Tier3, "Maradhoo", new Color32(178,
RegisterGang("Milo City", GangTier.Tier1, "Hithadhoo", new Color32(210,
RegisterGang("Wiss Wiss", GangTier.Tier2, "Hithadhoo", new Color32(64, 2
RegisterGang("DTS", GangTier.Tier2, "Hithadhoo", new Color32(95, 158, 16
RegisterGang("USGANDA", GangTier.Tier2, "Hithadhoo", new Color32(112, 12
RegisterGang("Eque Tasman", GangTier.Tier2, "Hithadhoo", new Color32(119
RegisterGang("XERAFON", GangTier.Tier3, "Hithadhoo", new Color32(25, 25,

// REMOTE ATOLLS (15 major + procedural)
RegisterGang("Mulaku Crew", GangTier.Tier3, "Fuvahmulah", new Color32(14
RegisterGang("Huvadhu-Old", GangTier.Tier3, "Huvadhu", new Color32(160,
RegisterGang("Laamu-Run", GangTier.Tier3, "Laamu", new Color32(176, 196,
RegisterGang("Dot.Fall Mujahid", GangTier.Tier2, "Dot.Fall", new Color32
```

```
    Debug.Log($"Initialized {activeGangs.Count} gangs - TARGET: 83");
}

void RegisterGang(string name, GangTier tier, string territory, Color32 colo
{
    Gang gang = new Gang
    {
        name = name,
        tier = tier,
        territory = territory,
        color = color,
        reputation = 0f,
        isHostile = false,
        heatLevel = 0
    };

    activeGangs[name] = gang;
    gangReputation[name] = 0f;
}

public void ModifyReputation(string gangName, float amount)
{
    if (gangReputation.ContainsKey(gangName))
    {
        gangReputation[gangName] += amount;
        gangReputation[gangName] = Mathf.Clamp(gangReputation[gangName], -10
    }
}

public float GetReputation(string gangName)
{
    return gangReputation.ContainsKey(gangName) ? gangReputation[gangName] :
}

public Gang GetGang(string gangName)
{
    return activeGangs.ContainsKey(gangName) ? activeGangs[gangName] : null;
}
```

```
    public List<Gang> GetAllGangs()
    {
        return new List<Gang>(activeGangs.Values);
    }

    public enum GangTier
    {
        Tier0 = 0, // Most powerful (Masodi, Kuda Henveiru)
        Tier1 = 1, // High activity (VK, Wanted, etc)
        Tier2 = 2, // Medium activity
        Tier3 = 3  // Low activity (remote islands)
    }

    [System.Serializable]
    public class Gang
    {
        public string name;
        public GangTier tier;
        public string territory;
        public Color32 color;
        public float reputation;
        public bool isHostile;
        public int heatLevel;
        public List<string> allies = new List<string>();
        public List<string> enemies = new List<string>();
    }
}
```

`GangMemberAI.cs` - NPC Gang Member Behavior

csharp

```
using UnityEngine;
using UnityEngine.AI;

public class GangMemberAI : MonoBehaviour
{
    [Header("Gang Affiliation - FROM BIBLE")]
    [SerializeField] private string gangName = "Masodi";
    [SerializeField] private GangManager.GangTier tier = GangManager.GangTier.Ti
```

```
[Header("AI Behavior")]
[SerializeField] private AIState currentState = AIState.Patrol;
[SerializeField] private float detectionRange = 10f;
[SerializeField] private float attackRange = 2f;
[SerializeField] private float patrolRadius = 20f;

[Header("Combat")]
[SerializeField] private int health = 100;
[SerializeField] private int damage = 10;
[SerializeField] private float attackCooldown = 1f;

[Header("Visual")]
[SerializeField] private SpriteRenderer spriteRenderer;
[SerializeField] private Sprite[] walkSprites = new Sprite[8];
[SerializeField] private GameObject gangColorIndicator;

private NavMeshAgent agent;
private Transform target;
private Vector3 patrolPoint;
private float lastAttackTime;
private Animator animator;

public enum AIState
{
    Idle,
    Patrol,
    Chase,
    Attack,
    Flee,
    Dead
}

void Awake()
{
    agent = GetComponent<NavMeshAgent>();
    spriteRenderer = GetComponent<SpriteRenderer>();
    animator = GetComponent<Animator>();

    // Configure for 2D
```

```
    agent.updateRotation = false;
    agent.updateUpAxis = false;

    // Set gang color
    if (gangColorIndicator != null)
    {
        GangManager.Gang gang = GangManager.Instance.GetGang(gangName);
        if (gang != null)
        {
            gangColorIndicator.GetComponent<SpriteRenderer>().color = gang.c
        }
    }

    SetNewPatrolPoint();
}

void Update()
{
    switch (currentState)
    {
        case AIState.Idle:
            HandleIdle();
            break;
        case AIState.Patrol:
            HandlePatrol();
            break;
        case AIState.Chase:
            HandleChase();
            break;
        case AIState.Attack:
            HandleAttack();
            break;
        case AIState.Flee:
            HandleFlee();
            break;
    }

    DetectThreats();
    UpdateAnimation();
}
```

```
void HandleIdle()
{
    agent.isStopped = true;
}

void HandlePatrol()
{
    agent.isStopped = false;

    if (!agent.pathPending && agent.remainingDistance < 0.5f)
    {
        SetNewPatrolPoint();
    }

    agent.SetDestination(patrolPoint);
}

void HandleChase()
{
    if (target == null)
    {
        currentState = AIState.Patrol;
        return;
    }

    float distance = Vector3.Distance(transform.position, target.position);

    if (distance > detectionRange * 2f)
    {
        target = null;
        currentState = AIState.Patrol;
        return;
    }

    if (distance <= attackRange)
    {
        currentState = AIState.Attack;
        return;
    }
```

```
    agent.SetDestination(target.position);
}

void HandleAttack()
{
    if (target == null)
    {
        currentState = AIState.Patrol;
        return;
    }

    float distance = Vector3.Distance(transform.position, target.position);

    if (distance > attackRange * 1.5f)
    {
        currentState = AIState.Chase;
        return;
    }

    agent.isStopped = true;

    // Face target
    Vector3 direction = (target.position - transform.position).normalized;
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.Euler(0, 0, angle - 90);

    // Attack
    if (Time.time - lastAttackTime >= attackCooldown)
    {
        PerformAttack();
        lastAttackTime = Time.time;
    }
}

void HandleFlee()
{
    if (target == null)
    {
        currentState = AIState.Patrol;
```

```
            return;
        }

        Vector3 fleeDirection = transform.position - target.position;
        Vector3 fleePoint = transform.position + fleeDirection.normalized * 10f;

        agent.SetDestination(fleePoint);

        if (Vector3.Distance(transform.position, target.position) > detectionRan
        {
            target = null;
            currentState = AIState.Patrol;
        }
    }

    void DetectThreats()
    {
        if (currentState == AIState.Dead) return;

        Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, detec

        foreach (Collider2D hit in hits)
        {
            // Check for player
            if (hit.CompareTag("Player"))
            {
                PlayerController player = hit.GetComponent<PlayerController>();
                if (player != null)
                {
                    // Check player's gang affiliation
                    if (IsHostileToPlayer(player))
                    {
                        target = hit.transform;
                        currentState = AIState.Chase;
                        return;
                    }
                }
            }

            // Check for rival gang members
```

```
            GangMemberAI rival = hit.GetComponent<GangMemberAI>();
            if (rival != null && rival.gangName != gangName)
            {
                GangManager.Gang myGang = GangManager.Instance.GetGang(gangName)
                if (myGang != null && myGang.enemies.Contains(rival.gangName))
                {
                    target = hit.transform;
                    currentState = AIState.Chase;
                    return;
                }
            }
        }
    }

    bool IsHostileToPlayer(PlayerController player)
    {
        // Check reputation with player
        float rep = GangManager.Instance.GetReputation(gangName);

        // Hostile if reputation < -30
        if (rep < -30f) return true;

        // Hostile if player has high wanted level
        if (player.wantedLevel >= 3) return true;

        return false;
    }

    void PerformAttack()
    {
        if (target == null) return;

        // Trigger attack animation
        if (animator != null)
        {
            animator.SetTrigger("Attack");
        }

        // Deal damage
        PlayerController player = target.GetComponent<PlayerController>();
```

```
    if (player != null)
    {
        player.TakeDamage(damage);
    }

    GangMemberAI rival = target.GetComponent<GangMemberAI>();
    if (rival != null)
    {
        rival.TakeDamage(damage);
    }
}

public void TakeDamage(int amount)
{
    health -= amount;

    if (health <= 0)
    {
        Die();
    }
    else
    {
        // Flash red
        StartCoroutine(FlashRed());
    }
}

System.Collections.IEnumerator FlashRed()
{
    spriteRenderer.color = Color.red;
    yield return new WaitForSeconds(0.1f);
    spriteRenderer.color = Color.white;
}

void Die()
{
    currentState = AIState.Dead;
    agent.isStopped = true;

    // Death animation
```

```
        if (animator != null)
        {
            animator.SetTrigger("Die");
        }

        // Disable collider
        GetComponent<Collider2D>().enabled = false;

        // Drop loot
        DropLoot();

        // Destroy after delay
        Destroy(gameObject, 5f);
    }

    void DropLoot()
    {
        // TODO: Spawn money/items based on tier
        int money = tier switch
        {
            GangManager.GangTier.Tier0 => Random.Range(500, 2000),
            GangManager.GangTier.Tier1 => Random.Range(200, 800),
            GangManager.GangTier.Tier2 => Random.Range(50, 300),
            GangManager.GangTier.Tier3 => Random.Range(10, 100),
            _ => 50
        };
    }

    void SetNewPatrolPoint()
    {
        Vector2 randomDirection = Random.insideUnitCircle * patrolRadius;
        patrolPoint = transform.position + new Vector3(randomDirection.x, 0, ran
    }

    void UpdateAnimation()
    {
        if (animator == null) return;

        float speed = agent.velocity.magnitude;
        animator.SetFloat("Speed", speed);
```

```
        animator.SetBool("IsChasing", currentState == AIState.Chase);
    }

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, detectionRange);

        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, attackRange);
    }
}
```

SECTION 6: PLAYER CONTROLLER - COMPLETE SYSTEM

`PlayerController.cs` - Master Player Script

csharp

```
using UnityEngine;
using System.Collections.Generic;

public class PlayerController : MonoBehaviour
{
    [Header("Player Identity - FROM BIBLE")]
    [SerializeField] private string playerName = "ASMRI DK MBH"; // Albako
    [SerializeField] private int playerAge = 25;

    [Header("Stats - FROM BIBLE SECTION 6.1")]
    [SerializeField] public int maxHealth = 100;
    [SerializeField] public int currentHealth;
    [SerializeField] public int maxStamina = 100;
    [SerializeField] public int currentStamina;
    [SerializeField] private float staminaRegenRate = 10f;

    [Header("Wanted System - 5 STARS")]
    [SerializeField] public int wantedLevel = 0; // 0-5
    [SerializeField] private float currentHeat = 0f;
```

```
    [SerializeField] private float heatDecayRate = 1f;

    [Header("Gang Affiliation - FROM BIBLE")]
    [SerializeField] public string currentGang = "Masodi";
    [SerializeField] public int gangReputation = 0;
    [SerializeField] public int territoryControl = 0;

    [Header("Skills - FROM BIBLE")]
    [SerializeField] public int fightingSkill = 0;
    [SerializeField] public int drivingSkill = 0;
    [SerializeField] public int swimmingSkill = 0;
    [SerializeField] public int boduberuSkill = 0; // UNIQUE MALDIVIAN SKILL

    [Header("Inventory")]
    [SerializeField] public int money = 500; // Starting MVR
    [SerializeField] public bool hasDhoniLicense = false;
    [SerializeField] public bool hasMotorbikeLicense = false;
    [SerializeField] public string[] weapons = new string[3];

    [Header("Family System - FROM BIBLE SECTION 4")]
    [SerializeField] private FamilyRelationship rawNDA_Trust = 0; // Grandfather
    [SerializeField] private FamilyRelationship star_Trust = 0; // Mother
    [SerializeField] private FamilyRelationship knoo_Trust = 0; // Sister
    [SerializeField] private FamilyRelationship daaba_Trust = 0; // Uncle
    [SerializeField] private FamilyRelationship panda_Trust = 0; // Aunt

    [Header("Karma System - 4 AXES")]
    [SerializeField] private int honor = 0; // -100 to +100
    [SerializeField] private int ruthless = 0; // -100 to +100
    [SerializeField] private int family = 0; // -100 to +100
    [SerializeField] private int ambition = 0; // -100 to +100

    [Header("97 Minors Counter - FROM BIBLE")]
    [SerializeField] private int minorsRecruited = 0;
    [SerializeField] private int civilianKills = 0;

    [Header("References")]
    [SerializeField] private IsometricCharacterController characterController;
    [SerializeField] private IsometricVehicleController currentVehicle;
    [SerializeField] private Animator animator;
```

```
[SerializeField] private SpriteRenderer spriteRenderer;

private bool isDead = false;
private bool isInVehicle = false;

public enum FamilyRelationship
{
    Disowned = -100,
    Hostile = -50,
    Neutral = 0,
    Trusted = 50,
    Loved = 100
}

void Awake()
{
    characterController = GetComponent<IsometricCharacterController>();
    animator = GetComponent<Animator>();
    spriteRenderer = GetComponent<SpriteRenderer>();

    currentHealth = maxHealth;
    currentStamina = maxStamina;
}

void Update()
{
    if (isDead) return;

    HandleStamina();
    HandleWantedSystem();
    HandleVehicleInteraction();
    HandleCombat();
    HandleQuickActions();
}

void HandleStamina()
{
    if (currentStamina < maxStamina && !characterController.IsSprinting)
    {
        currentStamina += Mathf.RoundToInt(staminaRegenRate * Time.deltaTime
```

```
            currentStamina = Mathf.Clamp(currentStamina, 0, maxStamina);
        }
    }

    void HandleWantedSystem()
    {
        // Heat decay over time
        if (!IsPoliceNearby())
        {
            currentHeat -= heatDecayRate * Time.deltaTime;
            currentHeat = Mathf.Max(0, currentHeat);
        }

        // Update wanted level
        int previousLevel = wantedLevel;

        if (currentHeat < 20f) wantedLevel = 0;
        else if (currentHeat < 50f) wantedLevel = 1;
        else if (currentHeat < 100f) wantedLevel = 2;
        else if (currentHeat < 200f) wantedLevel = 3;
        else if (currentHeat < 400f) wantedLevel = 4;
        else wantedLevel = 5;

        // Spawn police if level increased
        if (wantedLevel > previousLevel)
        {
            SpawnPpolice();
        }

        // Gang Act 2025 trigger at 5 stars
        if (wantedLevel == 5)
        {
            TriggerGangAct2025();
        }
    }

    void HandleVehicleInteraction()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
```

```
        if (isInVehicle)
        {
            ExitVehicle();
        }
        else
        {
            TryEnterNearbyVehicle();
        }
    }
}

void TryEnterNearbyVehicle()
{
    Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, 2f);

    foreach (Collider2D hit in hits)
    {
        IsometricVehicleController vehicle = hit.GetComponent<IsometricVehic
        if (vehicle != null)
        {
            // Check license requirements
            bool canEnter = vehicle.vehicleType switch
            {
                IsometricVehicleController.VehicleType.TraditionalDhoni => h
                IsometricVehicleController.VehicleType.MotorbikeGN125 => has
                _ => true
            };

            if (canEnter)
            {
                vehicle.EnterVehicle(gameObject);
                currentVehicle = vehicle;
                isInVehicle = true;
                characterController.enabled = false;
                return;
            }
            else
            {
                UIManager.Instance.ShowMessage("Need license!");
            }
```

```
        }
    }
}

void ExitVehicle()
{
    if (currentVehicle != null)
    {
        currentVehicle.ExitVehicle();
        currentVehicle = null;
        isInVehicle = false;
        characterController.enabled = true;
    }
}

void HandleCombat()
{
    if (isInVehicle) return;

    if (Input.GetMouseButtonDown(0))
    {
        PerformMeleeAttack();
    }
}

void PerformMeleeAttack()
{
    if (currentStamina < 10) return;

    currentStamina -= 10;

    // Trigger animation
    animator.SetTrigger("Attack");

    // Raycast for hits
    Vector2 attackDir = spriteRenderer.flipX ? Vector2.left : Vector2.right;
    RaycastHit2D hit = Physics2D.Raycast(transform.position, attackDir, 1.5f

    if (hit.collider != null)
    {
```

```
            // Check if target is gang member
            GangMemberAI gangMember = hit.collider.GetComponent<GangMemberAI>();
            if (gangMember != null)
            {
                int damage = 10 + (fightingSkill / 10);
                gangMember.TakeDamage(damage);

                // Add heat based on target
                if (gangMember.gangName == "Police")
                {
                    AddHeat(20f);
                    ModifyKarma(-10, 10, 0, 0); // -Honor, +Ruthless
                }
                else
                {
                    AddHeat(5f);
                    gangReputation += 2;
                }
            }
        }
    }

    void HandleQuickActions()
    {
        // Quick-save (safe houses only)
        if (Input.GetKeyDown(KeyCode.F5))
        {
            SaveGame();
        }

        // Quick-load
        if (Input.GetKeyDown(KeyCode.F9))
        {
            LoadGame();
        }

        // Gang menu
        if (Input.GetKeyDown(KeyCode.G))
        {
            UIManager.Instance.ToggleGangMenu();
```

```
    }

    // Family menu
    if (Input.GetKeyDown(KeyCode.H))
    {
        UIManager.Instance.ToggleFamilyMenu();
    }
}

public void AddHeat(float amount)
{
    currentHeat += amount;
    currentHeat = Mathf.Clamp(currentHeat, 0, 500f);
}

public void TakeDamage(int amount)
{
    currentHealth -= amount;
```