DAFTAR PUSTAKA

[1] A. S. F. Hasan and A. Palilu, "PENGEMBANGAN PROTOTIPE DRONE UNTUK KEPERLUAN MENDETEKSI KORBAN BENCANA," Makassar, Sep. 2020.

[2] R. Giga Isnanda *et al.*, "Analisis Komparatif Algoritma C4.5, Naïve Bayes, dan K-Nearest Neighbor untuk…."

[3] I. T. Prayudha and U. Chotijah, "Jurnal Teknik Informatika dan Komputer Pengembangan Game Edukasi 2D Mata Pelajaran IPA Menggunakan Unity Berbasis Mobile", [Online]. Available: https://journal.uhamka.ac.id/index.php/jutikom

[4] M. I. Java and R. E. Putra, "Rancang Bangun Aplikasi Drone Simulator Berbasis Android Menggunakan Game Engine Unity," 2019.

[5] T. Arifianto, D. B. Setyawan, and S. Sunaryo, "Penggunaan RFID (Radio Frequency Identification) CT-1809 Untuk Prototype Pendeteksi Sarana Berbasis Arduino Uno," *Journal of Telecommunication, Electronics, and Control Engineering (JTECE)*, vol. 3, no. 2, pp. 71–80, Nov. 2021, doi: 10.20895/jtece.v3i2.328.

[6] M. Khaerudin, D. B. Srisulistiowati, and J. Warta, "GAME EDUKASI DENGAN MENGGUNAKAN UNITY 3D UNTUK MENUNJANG PROSES PEMBELAJARAN."

[7] S. A. Alamsyah and M. Rivai, "Implementasi Lidar Sebagai Kontrol Ketinggian Quadcopter," 2020.

[8] I. P. Sari, A. H. Hazidar, M. Basri, F. Ramadhani, and A. A. Manurung, "Penerapan Palang Pintu Otomatis Jarak Jauh Berbasis RFID di Perumahan," *Blend Sains Jurnal Teknik*, vol. 2, no. 1, pp. 16–25, May 2023, doi: 10.56211/blendsains.v2i1.246.

[9] A. Lingga Praja and W. S. Huda, "Achmad Lingga Praja, Walidini Syaihul Huda Al-Faqih: Jurnal Ilmu Sosial, Humaniora, Teknik Pengembangan Game Edukasi Pemilahan Sampah Berbasis Unity Menggunakan Metode MDLC." [Online]. Available: https://journal.salahuddinal-ayyubi.com/index.php/AFJISH

[10] R. A. Purnama, A. Tri, and L. Putra, "APLIKASI WEB SERVER BERBASIS BAHASA C SHARP," *Jurnal Teknik Komputer*, vol. 4, no. 1, 2018.

[11] R. Simon Martin, Y. Dewanto, P. Studi Teknik Elektro, and F. Teknologi Industri, "PROTOTIPE KUNCI PINTU OTOMATIS MENGGUNAKAN SENSOR KAMERA BERBASIS RASPBERRY," *Jurnal Teknologi Industri*, vol. 12, no. 1, 2023.

[12] R. S. Pressman, "Software Engineering: A Practitioner's Approach," New York, 2001.

[13] J. Inovasi *et al.*, "Manfaat Canva untuk Melatih Kreativitas Pembuatan Mind

Map Mata Kuliah Alat-Alat Ukur dan Instrumentasi." [Online]. Available: https://jurnal.politap.ac.id/index.php/intern

[14] Noneng Marthiawati, Kevin Kurniawansyah, Hafiz Nugraha, and Fiqa Khairunnisa, "Pelatihan Pembuatan UML (Unified Modelling Language) Menggunakan Aplikasi Draw.io Pada Prodi Sistem Informasi Universitas Muhammadiyah Jambi," *Transformasi Masyarakat : Jurnal Inovasi Sosial dan Pengabdian*, vol. 1, no. 2, pp. 25–33, Mar. 2024, doi: 10.62383/transformasi.v1i2.109.

[15] S. W. Ramdany, S. Aulia Kaidar, B. Aguchino, C. Amelia, A. Putri, and R. Anggie, "Penerapan UML Class Diagram dalam Perancangan Sistem Informasi Perpustakaan Berbasis Web."

[16] A. Zalukhu *et al.*, "PERANGKAT LUNAK APLIKASI PEMBELAJARAN FLOWCHART," *Jurnal Teknologi Informasi dan Industri*, vol. 4, no. 1, 2023.

[17] Rony Setiawan, "Black Box Testing Untuk Menguji Perangkat Lunak," https://www.dicoding.com/blog/black-box-testing/.

[18] T. Menora, C. H. Primasari, Y. P. Wibisono, T. A. P. Sidhi, D. B. Setyohadi, and M. Cininta, "Implementasi Pengujian Alpha dan Beta Testing pada Aplikasi Gamelan Virtual Reality," 2023.

LAMPIRAN

Sebagai penunjang penjelasan pada bab perancangan dan implementasi, berikut disajikan beberapa potongan kode program utama yang digunakan dalam pengembangan aplikasi *Drone Detection Game*. Kode-kode berikut merupakan bagian inti yang memiliki peran penting dalam pengendalian drone, pengelolaan misi, serta penyajian antarmuka pengguna.

Lampiran 1. DroneController.cs

Script ini berfungsi untuk mengatur seluruh mekanisme fisika dan sistem kendali penerbangan drone. Di dalamnya mencakup proses lepas landas, pendaratan, pembatasan ketinggian, serta pengaturan kondisi mesin selama permainan berlangsung.

```
using UnityEngine;
using System;

[RequireComponent(typeof(Rigidbody))]
[RequireComponent(typeof(CapsuleCollider))]
public class DroneController : MonoBehaviour
{
    // ===== Hover Damping (stabilisasi) =====
    [Header("Hover Damping")]
    [Tooltip("Seberapa cepat yaw disaring agar tidak 'nyentak'")]
    public float yawResponse = 8f;            // semakin besar -> respons lebih cepat
    [Tooltip("Tarikan 'angin' saat melayang. 0–2 biasanya cukup")]
    public float hoverLinearDrag = 0.75f;     // drag saat engine ON (melayang/maju)
    [Tooltip("Perlambatan horizontal otomatis saat tidak menekan W/S (m/s^2)")]
    public float hoverBrakeAccel = 10f;       // seberapa cepat drift berhenti
    [Tooltip("Waktu pelan-ing naik/turun (detik)")]
    public float verticalSmoothTime = 0.15f;  // smoothing altitude

    private float _smoothedYaw = 0f;          // internal smoothing yaw
    private float _vertVelRef = 0f;           // ref untuk SmoothDamp
    private float _lastForwardInput = 0f;     // buat tahu kapan harus ngerem drift

    [Header("Jitter Reduction")]
    public float horizontalAccel = 40f;  // percepatan halus ke target speed
```

```
[Header("Flight Speeds (Realistic)")]
[Tooltip("Kecepatan gerak horizontal (13.3–26.9 m/s ≈ 48–97 km/jam)")]
public float speed = 20f;              // nilai tengah, akan di-clamp
public float yawSpeed = 60f;
public float altitudeSpeed = 4f;
public float takeoffHeight = 2f;
public float maxHeight = 300f;
public float landingSpeed = 2f;

[Header("Speed Limits (jangan diubah di runtime)")]
public float minHorizontalSpeed = 13.3f;
public float maxHorizontalSpeed = 26.9f;

[Header("Return-To-Base")]
public float safeReturnAltitude = 100f;

[Header("Terrain Bounds (World XZ)")]
public Vector2 terrainMin = new Vector2(-9999f, -9999f);
public Vector2 terrainMax = new Vector2(9999f, 9999f);

[Header("Grounding")]
[SerializeField] private float groundSnapEpsilon = 0.03f;

[Header("Collision/Slide")]
public LayerMask obstacleMask = ~0;
public float skinWidth = 0.02f;

[Header("Audio")]
public AudioSource engineLoopSource;
public AudioSource sfxSource;
public AudioClip engineStartClip;
public AudioClip engineStopClip;
public AudioClip collisionClip;
[Range(0f, 1f)] public float collisionVolume = 1f;

[Range(0f, 1f)] public float engineIdleVolume = 0.15f;
[Range(0f, 1f)] public float engineFlyVolume = 0.35f;
public float engineIdlePitch = 0.90f;
public float engineFlyPitch = 1.20f;
public float audioSmooth = 4f;

private Rigidbody rb;
private CapsuleCollider col;
private bool engineOn = false;
```

```
    private bool isFlying = false;
    private bool isTakingOff = false;
    private bool isLanding = false;
    private bool isReturning = false;
    private Vector3 basePosition;
    private float baseY;
    private float targetAltitude;
    private float verticalVelocity = 0f;
    private float yawInput = 0f;

    public Action OnReturnToBaseCompleted;
    private bool _notifiedReturnComplete = false;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        col = GetComponent<CapsuleCollider>();

        rb.useGravity = false;
        rb.collisionDetectionMode = CollisionDetectionMode.ContinuousDynamic;
        rb.interpolation = RigidbodyInterpolation.Interpolate;
        rb.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationZ;

        basePosition = transform.position;
        baseY = transform.position.y;
        targetAltitude = baseY;
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;
        rb.Sleep();

        if (engineLoopSource != null)
        {
            engineLoopSource.loop = true;
            engineLoopSource.volume = 0f;
        }
    }

    void Update() => HandleInput();

    void FixedUpdate()
    {
        if (engineOn)
        {
```

```
        if (isTakingOff) HandleTakeOff();
        else if (isLanding) HandleLanding();
        else if (isReturning) HandleReturnToBase();
        else if (isFlying)
        {
            // SMOOTH YAW
            _smoothedYaw = Mathf.Lerp(_smoothedYaw, yawInput,
Time.fixedDeltaTime * yawResponse);
            if (Mathf.Abs(_smoothedYaw) > 0f)
            {
                float deltaYaw = _smoothedYaw * yawSpeed * Time.fixedDeltaTime;
                Quaternion add = Quaternion.AngleAxis(deltaYaw, Vector3.up);
                rb.MoveRotation(rb.rotation * add);
            }

            HandleManualFlight();
        }
        else
        {
            rb.velocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
        }

        // batasi ketinggian
        var pos = rb.position;
        float maxY = baseY + maxHeight;
        if (pos.y > maxY) { pos.y = maxY; verticalVelocity = 0f;
rb.MovePosition(pos); }
        if (pos.y < baseY) { pos.y = baseY; verticalVelocity = 0f;
rb.MovePosition(pos); }

        // >>> Hover damping aktif saat engine ON
        ApplyHoverDamping();
    }
    else
    {
        // engine OFF: pastikan drag nol (biar jatuh natural)
        rb.drag = 0f;

        // jatuh sampai baseY lalu diam
        float y = rb.position.y;
        if (y > baseY + groundSnapEpsilon)
        {
            if (!rb.useGravity) rb.useGravity = true;
```

```
            rb.constraints = RigidbodyConstraints.None;
        }
        else
        {
            if (rb.useGravity) rb.useGravity = false;
            rb.velocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;

            Vector3 pos = rb.position; pos.y = baseY;
            rb.MovePosition(pos);
            rb.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationZ;
            rb.Sleep();
        }
    }

    // clamp XZ ke bounds
    Vector3 p = rb.position;
    float clampedX = Mathf.Clamp(p.x, terrainMin.x, terrainMax.x);
    float clampedZ = Mathf.Clamp(p.z, terrainMin.y, terrainMax.y);
    if (!Mathf.Approximately(clampedX, p.x) || !Mathf.Approximately(clampedZ,
p.z))
        rb.MovePosition(new Vector3(clampedX, p.y, clampedZ));

    // loop audio
    if (engineLoopSource != null)
    {
        if (engineOn)
        {
            if (!engineLoopSource.isPlaying) engineLoopSource.Play();
            bool flightState = isFlying || isTakingOff || isReturning;
            float targetVol = flightState ? engineFlyVolume : engineIdleVolume;
            float targetPitch = flightState ? engineFlyPitch : engineIdlePitch;
            engineLoopSource.volume = Mathf.Lerp(engineLoopSource.volume,
targetVol, Time.deltaTime * audioSmooth);
            engineLoopSource.pitch = Mathf.Lerp(engineLoopSource.pitch,
targetPitch, Time.deltaTime * audioSmooth);
        }
        else
        {
            engineLoopSource.volume = Mathf.Lerp(engineLoopSource.volume, 0f,
Time.deltaTime * audioSmooth * 2f);
            if (engineLoopSource.isPlaying && engineLoopSource.volume < 0.01f)
                engineLoopSource.Stop();
```

```
        }
      }
   }

   void HandleInput()
   {
      if (Input.GetKeyDown(KeyCode.O) && !engineOn)
      {
         engineOn = true;
         rb.useGravity = false;
         rb.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationZ;
         rb.WakeUp();
         verticalVelocity = 0f;
         _notifiedReturnComplete = false;

         if (engineLoopSource && !engineLoopSource.isPlaying)
         {
            engineLoopSource.Play();
            engineLoopSource.volume = 0f;
            engineLoopSource.pitch = engineIdlePitch;
         }
         if (sfxSource && engineStartClip)
            sfxSource.PlayOneShot(engineStartClip);
      }

      if (Input.GetKeyDown(KeyCode.P) && engineOn)
      {
         engineOn = false;
         isFlying = isTakingOff = isLanding = isReturning = false;
         rb.useGravity = true;
         rb.constraints = RigidbodyConstraints.None;
         verticalVelocity = 0f;
         if (sfxSource && engineStopClip)
            sfxSource.PlayOneShot(engineStopClip);
      }

      if (engineOn && Input.GetKeyDown(KeyCode.T) && !isFlying &&
!isTakingOff)
      {
         isTakingOff = true;
         isLanding = false;
         isReturning = false;
         targetAltitude = baseY + takeoffHeight;
```

```
        }

        if (engineOn && Input.GetKeyDown(KeyCode.L) && isFlying)
        {
            isLanding = true;
            isTakingOff = false;
            isReturning = false;
        }

        if (engineOn && Input.GetKeyDown(KeyCode.B) && isFlying)
        {
            isReturning = true;
            isLanding = false;
            isTakingOff = false;
            float safeAlt = baseY + safeReturnAltitude;
            if (targetAltitude < safeAlt) targetAltitude = safeAlt;
            _notifiedReturnComplete = false;
        }

        if (engineOn && isFlying && !isLanding && !isReturning)
        {
            yawInput = 0f;
            if (Input.GetKey(KeyCode.A)) yawInput = -1f;
            if (Input.GetKey(KeyCode.D)) yawInput = 1f;

            if (Input.GetKey(KeyCode.UpArrow))
                targetAltitude = Mathf.Min(targetAltitude + altitudeSpeed *
Time.deltaTime, baseY + maxHeight);
            if (Input.GetKey(KeyCode.DownArrow))
                targetAltitude = Mathf.Max(targetAltitude - altitudeSpeed *
Time.deltaTime, baseY);
        }
        else yawInput = 0f;
    }

    void HandleTakeOff()
    {
        float currentY = rb.position.y;
        if (currentY < targetAltitude - 0.05f)
        {
            verticalVelocity = Mathf.Lerp(verticalVelocity, altitudeSpeed,
Time.fixedDeltaTime * 2f);
            rb.MovePosition(rb.position + Vector3.up * (verticalVelocity *
Time.fixedDeltaTime));
```

```
        }
        else
        {
            rb.velocity = Vector3.zero;
            isTakingOff = false;
            isFlying = true;
            targetAltitude = rb.position.y;
        }
    }

    void HandleLanding()
    {
        float currentY = rb.position.y;
        if (currentY > baseY + 0.05f)
        {
            verticalVelocity = Mathf.Lerp(verticalVelocity, -landingSpeed,
Time.fixedDeltaTime * 2f);
            rb.MovePosition(rb.position + Vector3.up * (verticalVelocity *
Time.fixedDeltaTime));
        }
        else
        {
            rb.velocity = Vector3.zero;
            isLanding = false;
            isFlying = false;
            Vector3 pos = rb.position; pos.y = baseY;
            rb.MovePosition(pos);
        }
    }

    void HandleReturnToBase()
    {
        Vector3 current = rb.position;
        float currentY = current.y;
        float altError = targetAltitude - currentY;

        if (currentY < targetAltitude - 0.05f)
        {
            verticalVelocity = Mathf.Lerp(verticalVelocity, Mathf.Clamp(altError, -
altitudeSpeed, altitudeSpeed), Time.fixedDeltaTime * 2f);
            rb.MovePosition(current + Vector3.up * (verticalVelocity *
Time.fixedDeltaTime));
            return;
        }
```

```
        Vector3 target = new Vector3(basePosition.x, current.y, basePosition.z);
        Vector3 toBase = target - current; toBase.y = 0f;

        if (toBase.sqrMagnitude > 0.0001f)
        {
            Quaternion targetRot = Quaternion.LookRotation(toBase.normalized,
Vector3.up);
            Quaternion stepRot = Quaternion.RotateTowards(rb.rotation, targetRot,
yawSpeed * Time.deltaTime);
            rb.MoveRotation(stepRot);
        }

        float dt = Time.fixedDeltaTime;
        float horizSpeed = Mathf.Clamp(speed, minHorizontalSpeed,
maxHorizontalSpeed);
        Vector3 horizStep = transform.forward * (horizSpeed * dt);
        verticalVelocity = Mathf.Lerp(verticalVelocity, Mathf.Clamp(altError, -
altitudeSpeed, altitudeSpeed), Time.fixedDeltaTime * 2f);
        Vector3 next = current + horizStep + Vector3.up * (verticalVelocity * dt);
        rb.MovePosition(next);

        if (Vector3.Distance(new Vector3(basePosition.x, 0, basePosition.z), new
Vector3(next.x, 0, next.z)) <= 0.5f)
        {
            if (!_notifiedReturnComplete)
            {
                _notifiedReturnComplete = true;
                OnReturnToBaseCompleted?.Invoke();
            }
            isReturning = false;
            isLanding = true;
        }
    }

    void HandleManualFlight()
    {
        float dt = Time.fixedDeltaTime;
        Vector3 currentVel = rb.velocity;

        // --- Input maju / mundur ---
        float forwardInput = 0f;
        if (Input.GetKey(KeyCode.W)) forwardInput = 1f;
        if (Input.GetKey(KeyCode.S)) forwardInput = -1f;
```

```
        // Target speed horizontal (dibatasi 13.3–26.9 m/s)
        float baseSpeed = Mathf.Clamp(speed, minHorizontalSpeed, maxHorizontalSpeed);
        float targetSpeed = baseSpeed * forwardInput; // 0 kalau tidak menekan W/S

        // Pisahkan komponen horizontal terhadap arah maju drone
        Vector3 fwd = transform.forward; fwd.y = 0f; fwd.Normalize();
        Vector3 horizVel = currentVel; horizVel.y = 0f;

        // Proyeksikan ke arah forward lalu akselerasi halus ke target
        float curSpeedAlongFwd = Vector3.Dot(horizVel, fwd);
        float newSpeedAlongFwd = Mathf.MoveTowards(curSpeedAlongFwd, targetSpeed, horizontalAccel * dt);

        // Bangun vektor horizontal baru: komponen sepanjang fwd + sisanya ortho (drift)
        Vector3 ortho = horizVel - fwd * curSpeedAlongFwd;
        Vector3 newHorizVel = fwd * newSpeedAlongFwd + ortho;

        // --- Altitude control: SmoothDamp agar tidak “menggigil” ---
        float altitudeError = targetAltitude - rb.position.y;
        float newVy = Mathf.SmoothDamp(currentVel.y,
                                       Mathf.Clamp(altitudeError / dt, -altitudeSpeed, altitudeSpeed),
                                       ref _vertVelRef, verticalSmoothTime, altitudeSpeed, dt);

        // Terapkan velocity gabungan (batasi horizontal agar tidak melebihi max)
        Vector3 finalVel = newHorizVel + Vector3.up * newVy;
        Vector3 hOnly = finalVel; hOnly.y = 0f;
        float hMag = hOnly.magnitude;
        if (hMag > maxHorizontalSpeed)
        {
            hOnly = hOnly.normalized * maxHorizontalSpeed;
            finalVel = hOnly + Vector3.up * finalVel.y;
        }
        rb.velocity = finalVel;

        // Simpan input terakhir untuk rem otomatis
        _lastForwardInput = forwardInput;
    }

    // Menstabilkan drone saat melayang: tambah drag, rem drift horizontal saat tidak input
```

```
    void ApplyHoverDamping()
    {
        // Saat engine ON pakai drag moderat supaya gerak tidak 'liar'
        rb.drag = hoverLinearDrag;

        // Rem drift horizontal otomatis ketika tidak tekan W/S dan bukan landing/return
        if (isFlying && !isLanding && !isReturning && Mathf.Approximately(_lastForwardInput, 0f))
        {
            Vector3 v = rb.velocity;
            Vector3 horiz = v; horiz.y = 0f;

            float cur = horiz.magnitude;
            if (cur > 0.01f)
            {
                float newMag = Mathf.MoveTowards(cur, 0f, hoverBrakeAccel * Time.fixedDeltaTime);
                Vector3 newHoriz = (cur > 0f ? horiz / cur : Vector3.zero) * newMag;
                rb.velocity = newHoriz + Vector3.up * v.y;
            }
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (sfxSource && collisionClip)
        {
            if (((1 << collision.gameObject.layer) & obstacleMask) != 0)
                sfxSource.PlayOneShot(collisionClip, collisionVolume); // 🔊 tabrakan
        }
    }

    public bool IsFlying() => isFlying;
    public bool EngineOn() => engineOn;

    public void TriggerReturnToBase()
    {
        if (!EngineOn()) return;
        if (!IsFlying()) return;
        var safeAlt = baseY + safeReturnAltitude;
        if (targetAltitude < safeAlt) targetAltitude = safeAlt;
        isReturning = true;
        isLanding = false;
```

```
        isTakingOff = false;
        _notifiedReturnComplete = false;
    }

    // Kecepatan horizontal aktual (m/s) dari Rigidbody
    public float GetHorizontalSpeed()
    {
        if (rb == null) return 0f;
        Vector3 v = rb.velocity; v.y = 0f;
        return Mathf.Min(v.magnitude, maxHorizontalSpeed);
    }

    // Apakah drone dalam keadaan terbalik (flip)?
    // thresholdDot: 1 = tegak lurus ke atas, 0 = miring 90°, <0 = kebalik.
    // Default 0.2 artinya jika sudut > ~78° dari tegak, dianggap terbalik.
    public bool IsUpsideDown(float thresholdDot = 0.2f)
    {
        return Vector3.Dot(transform.up, Vector3.up) < thresholdDot;
    }

}
```

Lampiran 2. DroneGameManager.cs

Script ini bertanggung jawab mengelola keseluruhan alur permainan, termasuk proses deteksi RFID, pencatatan data personel yang terdeteksi, penampilan hasil deteksi pada layar, serta penentuan kondisi *game over* apabila drone mengalami kerusakan atau terbalik.

```
// DroneGameManager.cs — FINAL (10s merah + tick SFX, notify "SUDAH TERDETEKSI", flip->GameOver)
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using TMPro;
using System;
using System.Text;
using System.Collections;
using System.Collections.Generic;

public class DroneGameManager : MonoBehaviour
{
    [Header("Core")]
```

```
public DroneController drone;
public GeoMapper geo;

[System.Serializable]
public class RFIDTarget
{
    public Transform target;
    public TMP_Text detectionText;        // optional floating text per target
    [HideInInspector] public bool hasDetected = false;

    // anti-spam untuk "SUDAH TERDETEKSI"
    [HideInInspector] public float nextAlreadyNotifyTime = 0f;
}

[Header("RFID Detection")]
public float detectionRadius = 10f;
public RFIDTarget[] targets = new RFIDTarget[10];
public float smallPopupDuration = 3f;

[Header("Detection Card Popup (optional)")]
public GameObject cardPanel;
public Image cardPhoto;
public TMP_Text cardNameText;
public TMP_Text cardIdText;
public float cardShowDuration = 3f;

[Header("SFX (optional)")]
public AudioSource sfxSource;
public AudioClip detectedSfx;
[Range(0f, 1f)] public float sfxVolume = 1f;

[Header("Mission UI SFX")]
public AudioClip preBriefSfx;        // saat panel briefing tampil
public AudioClip oneMinuteSfx;       // saat warning 1 menit
public AudioClip countdownTickSfx;   // tiap detik 10..1
public AudioClip countdownFinalSfx;  // detik 0 (opsional)
[Range(0f, 1f)] public float uiSfxVolume = 1f;

[Header("Notify (optional)")]
[Tooltip("Teks global singkat 3 detik (opsional)")]
public TMP_Text notifyText;
public float notifyDuration = 3f;

[Serializable]
```

```
    public class DetectionRecord
    {
        public string personId;
        public string name;
        public Vector3 worldPos;
        public double lat;
        public double lon;
        public float firstDistance;
        public DateTime timeUtc;
    }
    private readonly Dictionary<string, DetectionRecord> _byId = new
Dictionary<string, DetectionRecord>();
    private readonly List<DetectionRecord> _ordered = new
List<DetectionRecord>();

    [Header("Resume Summary UI")]
    public GameObject summaryPanel;
    public TMP_Text summaryHeaderText;
    public TMP_Text summaryBodyText;

    [Header("Resume Hotkey (optional)")]
    public KeyCode stopKey = KeyCode.None;

    [Header("Play/Pause Panel")]
    public GameObject playPausePanel;
    public KeyCode togglePauseKey = KeyCode.None;
    private bool _isPausePanelShown = false;

    [Header("HUD (optional)")]
    public TMP_Text dynamicStatusText;
    public float hudRefreshInterval = 0.25f;
    private float _nextHudAt = 0f;

    private static bool _geoWarnedOnce = false;

    [Header("Mission Flow")]
    public int totalTargetsToDetect = 10;
    public float preBriefSeconds = 10f;
    public float missionSeconds = 600f;

    [Header("Mission UI")]
    public GameObject preBriefPanel;
    public TMP_Text preBriefCountdownText;
    public TMP_Text missionTimerText;
```

```
    public GameObject gameOverPanel;
    public TMP_Text gameOverBodyText;
    public GameObject missionCompletePanel;
    public TMP_Text missionCompleteBodyText;
    public GameObject oneMinuteWarningPanel;
    public TMP_Text oneMinuteWarningText;

    // Crash/Flip → Game Over
    [Header("Crash/Flip Settings")]
    public bool gameOverOnFlip = true;
    [Range(-1f, 1f)] public float flipDotThreshold = 0.2f; // <0.2 ≈ hampir terbalik
    public float flipHoldSeconds = 1.0f;                   // tahan sekian detik
    private float _flipAccum = 0f;

    private enum MissionState { Brief, WaitingForOn, Running, Completed, GameOver }
    private MissionState _state = MissionState.Brief;
    private float _missionRemain;
    private bool _warnedOneMinute = false;
    private bool _lastEngineOn = false;

    // 10 detik terakhir
    private int _lastTickSecond = -1;
    private Color _timerDefaultColor;
    private bool _timerColorCached = false;

    // ===================== Unity ====================
    private void Start()
    {
        if (playPausePanel) playPausePanel.SetActive(false);
        if (summaryPanel) summaryPanel.SetActive(false);
        if (cardPanel) cardPanel.SetActive(false);

        if (!geo)
        {
            geo = FindObjectOfType<GeoMapper>();
            if (!geo && !_geoWarnedOnce)
            {
                Debug.LogWarning("[DroneGameManager] GeoMapper tidak ditemukan. Koordinat fallback X/Z.");
                _geoWarnedOnce = true;
            }
        }
```

```
        if (missionTimerText && !_timerColorCached)
        {
            _timerDefaultColor = missionTimerText.color;
            _timerColorCached = true;
        }

        if (preBriefPanel) preBriefPanel.SetActive(true);
        if (sfxSource && preBriefSfx) sfxSource.PlayOneShot(preBriefSfx,
uiSfxVolume);

        if (gameOverPanel) gameOverPanel.SetActive(false);
        if (missionCompletePanel) missionCompletePanel.SetActive(false);
        if (oneMinuteWarningPanel) oneMinuteWarningPanel.SetActive(false);

        _state = MissionState.Brief;
        _missionRemain = missionSeconds;
        _warnedOneMinute = false;
        _lastTickSecond = -1;
        _flipAccum = 0f;

        StartCoroutine(PreBriefRoutine());
    }

    private void Update()
    {
        if (togglePauseKey != KeyCode.None &&
Input.GetKeyDown(togglePauseKey))
            TogglePlayPausePanel();

        if (stopKey != KeyCode.None && Input.GetKeyDown(stopKey))
            ToggleSummaryPanel();

        RunDetectionTick();
        RunHudTick();
        RunMissionTick();
    }

    // ===================== Brief ====================
    private IEnumerator PreBriefRoutine()
    {
        float t = preBriefSeconds;
        while (t > 0f)
        {
            if (preBriefCountdownText)
```

```
            preBriefCountdownText.text =
                $"Misi: Deteksi {totalTargetsToDetect} personel\nMulai dalam
{Mathf.CeilToInt(t)} detik...";
            yield return null;
            t -= Time.deltaTime;
        }

        if (preBriefPanel) preBriefPanel.SetActive(false);
        _state = MissionState.WaitingForOn; // menunggu Engine ON
    }

    //          ====================          Detection          &          Logging
====================
    private void RunDetectionTick()
    {
        if (targets == null || targets.Length == 0) return;

        Transform origin = (drone != null) ? drone.transform : transform;

        foreach (var t in targets)
        {
            if (t == null || t.target == null) continue;

            // jarak horizontal (abaikan Y)
            Vector3 a = origin.position; a.y = 0f;
            Vector3 b = t.target.position; b.y = 0f;
            float dist = Vector3.Distance(a, b);

            if (dist <= detectionRadius && !t.hasDetected)
            {
                t.hasDetected = true;

                string nama = t.target.name;
                string id = "N/A";
                Sprite foto = null;
                var info = t.target.GetComponent<PersonelInfo>();
                if (info)
                {
                    if (!string.IsNullOrEmpty(info.nama)) nama = info.nama;
                    if (!string.IsNullOrEmpty(info.personelID)) id = info.personelID;
                    if (info.photo) foto = info.photo;
                }

                if (t.detectionText)
```

```
            {
                t.detectionText.text = $"TERDETEKSI: {nama} (ID: {id})";
                t.detectionText.enabled = true;
                StartCoroutine(HideTmpTextAfter(t.detectionText,
smallPopupDuration));
            }

            double lat = 0, lon = 0;
            if (geo) geo.WorldToGeo(t.target.position, out lat, out lon);

            if (sfxSource && detectedSfx) sfxSource.PlayOneShot(detectedSfx,
sfxVolume);

            AddDetectionRecord(id, nama, t.target.position, lat, lon, dist);

            if (cardPanel) ShowCardPopup(nama, id, foto);

            t.nextAlreadyNotifyTime = Time.time + 0.1f;
        }
        else if (dist <= detectionRadius && t.hasDetected)
        {
            // Tampilkan "SUDAH TERDETEKSI" saat mendekat lagi (cooldown 3
detik)
            if (Time.time >= t.nextAlreadyNotifyTime)
            {
                string nama = t.target.name;
                var info = t.target.GetComponent<PersonelInfo>();
                if (info && !string.IsNullOrEmpty(info.nama)) nama = info.nama;

                if (notifyText)
                {
                    ShowNotify($"SUDAH TERDETEKSI: {nama}");
                }
                else if (t.detectionText)
                {
                    t.detectionText.text = $"SUDAH TERDETEKSI: {nama}";
                    t.detectionText.enabled = true;
                    StartCoroutine(HideTmpTextAfter(t.detectionText, 3f));
                }

                t.nextAlreadyNotifyTime = Time.time + notifyDuration;
            }
        }
    }
```

```
    }

    private void AddDetectionRecord(string personId, string name, Vector3
worldPos, double lat, double lon, float distance)
    {
        string key = string.IsNullOrEmpty(personId) || personId == "N/A"
            ? $"{name}@{worldPos.x:F2},{worldPos.z:F2}"
            : personId;

        if (_byId.ContainsKey(key)) return;

        var rec = new DetectionRecord
        {
            personId = personId,
            name = name,
            worldPos = worldPos,
            lat = lat,
            lon = lon,
            firstDistance = distance,
            timeUtc = DateTime.UtcNow
        };
        _byId[key] = rec;
        _ordered.Add(rec);
    }

    // ==================== Card popup ====================
    private Coroutine _cardCo;
    private void ShowCardPopup(string nama, string id, Sprite foto)
    {
        if (!cardPanel) return;

        if (cardNameText) cardNameText.text = nama;
        if (cardIdText) cardIdText.text = string.IsNullOrEmpty(id) ? "ID: N/A" : $"ID:
{id}";

        if (cardPhoto)
        {
            if (foto) { cardPhoto.enabled = true; cardPhoto.sprite = foto; }
            else { cardPhoto.enabled = false; cardPhoto.sprite = null; }
        }

        cardPanel.SetActive(true);
        if (_cardCo != null) StopCoroutine(_cardCo);
        _cardCo = StartCoroutine(HideCardAfter(cardShowDuration));
```

```
    }
    private IEnumerator HideCardAfter(float duration)
    {
        yield return new WaitForSeconds(duration);
        if (cardPanel) cardPanel.SetActive(false);
        _cardCo = null;
    }
    private IEnumerator HideTmpTextAfter(TMP_Text tmp, float duration)
    {
        yield return new WaitForSeconds(duration);
        if (tmp) tmp.enabled = false;
    }

    // ==================== Global Notify ====================
    private Coroutine _notifyCo;
    private void ShowNotify(string msg)
    {
        if (!notifyText) return;
        if (_notifyCo != null) StopCoroutine(_notifyCo);
        _notifyCo = StartCoroutine(NotifyRoutine(msg));
    }
    private IEnumerator NotifyRoutine(string msg)
    {
        notifyText.text = msg;
        notifyText.gameObject.SetActive(true);
        yield return new WaitForSeconds(notifyDuration);
        notifyText.gameObject.SetActive(false);
        _notifyCo = null;
    }

    //              ====================              Summary              (Resume)
====================
    private void BuildAndShowSummary()
    {
        if (summaryHeaderText) summaryHeaderText.text = "RESUME MISI";

        var sb = new StringBuilder();
        if (_ordered.Count > 0)
        {
            for (int i = 0; i < _ordered.Count; i++)
            {
                var r = _ordered[i];
                string coord = (r.lat != 0 || r.lon != 0)
                    ? $"Lon={r.lon:F6}, Lat={r.lat:F6}"
```

```
                : $"X={r.worldPos.x:F2}, Z={r.worldPos.z:F2}";
            sb.AppendLine($"{i + 1}. {r.name} (ID: {r.personId})");
            sb.AppendLine($"    {coord} | {r.firstDistance:F1} m");
        }
    }
    else
    {
        sb.AppendLine("Belum ada personel yang terdeteksi.");
    }

    if (summaryBodyText) summaryBodyText.text = sb.ToString();

    PauseGame(true);
    if (summaryPanel) summaryPanel.SetActive(true);
}

public void ToggleSummaryPanel()
{
    if (!summaryPanel) return;
    bool willShow = !summaryPanel.activeSelf;
    if (willShow) BuildAndShowSummary();
    else { summaryPanel.SetActive(false); PauseGame(false); }
}

// tombol dari panel Pause
public void Button_ShowSummaryFromPause()
{
    if (summaryPanel && !summaryPanel.activeSelf) BuildAndShowSummary();
    if (playPausePanel) playPausePanel.SetActive(false);
    _isPausePanelShown = false;
}
public void Button_RestartFromSummary()
{
    PauseGame(false);
    ClearAllDetections();
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
public void Button_ExitToMenuFromSummary()
{
    PauseGame(false);
    ClearAllDetections();
    SceneManager.LoadScene("MainMenu");
}
public void Button_BackToPauseFromSummary()
```

```
    {
        if (summaryPanel) summaryPanel.SetActive(false);
        if (playPausePanel) playPausePanel.SetActive(true);
        _isPausePanelShown = true;
        PauseGame(true);
    }

    // ==================== Play/Pause ====================
    public void TogglePlayPausePanel()
    {
        _isPausePanelShown = !_isPausePanelShown;
        if (playPausePanel) playPausePanel.SetActive(_isPausePanelShown);
        PauseGame(_isPausePanelShown);
    }
    public void Button_SkipFromPause()
    {
        _isPausePanelShown = false;
        if (playPausePanel) playPausePanel.SetActive(false);
        PauseGame(false);
    }
    public void Button_RestartFromPause()
    {
        PauseGame(false);
        ClearAllDetections();
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    public void Button_ExitFromPause()
    {
        PauseGame(false);
        ClearAllDetections();
        SceneManager.LoadScene("MainMenu");
    }
    public void Button_RestartFromEndPanels() => Button_RestartFromPause();
    public void Button_ExitFromEndPanels() => Button_ExitFromPause();

    // ==================== HUD ====================
    private void RunHudTick()
    {
        if (!dynamicStatusText || !drone) return;
        if (Time.unscaledTime < _nextHudAt) return;
        _nextHudAt = Time.unscaledTime + hudRefreshInterval;

        bool engine = drone.EngineOn();
        bool flying = drone.IsFlying();
```

```
        float altitude = drone.transform.position.y;
        float heading = drone.transform.eulerAngles.y;

        float displaySpeed = drone != null ? drone.GetHorizontalSpeed() : 0f;

        string mode = engine ? (flying ? "Flying" : "Idle") : "Engine OFF";
        dynamicStatusText.text =
            $"Engine   : {(engine ? "<color=green>ON</color>" :
"<color=red>OFF</color>")}\n" +
            $"Mode   : {mode}\n" +
            $"Alt    : {altitude:F1} m\n" +
            $"Head   : {heading:0}°\n" +
            $"Speed  : {displaySpeed:0.0} m/s";
    }

    // ==================== Mission Tick ====================
    private void RunMissionTick()
    {
        bool engineNow = drone ? drone.EngineOn() : false;
        if (_state == MissionState.WaitingForOn && engineNow && !_lastEngineOn)
        {
            _state = MissionState.Running;
            _missionRemain = missionSeconds;
            _warnedOneMinute = false;
            _lastTickSecond = -1;
            _flipAccum = 0f;

            if (missionTimerText && _timerColorCached)
                missionTimerText.color = _timerDefaultColor;
        }
        _lastEngineOn = engineNow;

        if (missionTimerText)
        {
            float show = 0f;
            if (_state == MissionState.Running) show = Mathf.Max(0f,
_missionRemain);
            else if (_state == MissionState.WaitingForOn) show = missionSeconds;
            int m = Mathf.FloorToInt(show / 60f);
            int s = Mathf.FloorToInt(show % 60f);
            missionTimerText.text = $"{m:00}:{s:00}";
        }

        if (_state == MissionState.Running)
```

```
    {
        _missionRemain -= Time.deltaTime;

        // Flip/crash → Game Over (tahan sekian detik)
        if (gameOverOnFlip && drone != null)
        {
            if (drone.IsUpsideDown(flipDotThreshold))
            {
                _flipAccum += Time.deltaTime;
                if (_flipAccum >= flipHoldSeconds)
                {
                    GameOver("Drone terbalik.");
                    return;
                }
            }
            else _flipAccum = 0f;
        }

        // Warning 1 menit
        if (!_warnedOneMinute && _missionRemain <= 60f)
        {
            _warnedOneMinute = true;
            if (oneMinuteWarningText) oneMinuteWarningText.text = "1 menit
tersisa! Drone segera kembali ke markas!";
            if                                              (oneMinuteWarningPanel)
StartCoroutine(FlashPanel(oneMinuteWarningPanel, 4f));
            if (sfxSource && oneMinuteSfx) sfxSource.PlayOneShot(oneMinuteSfx,
uiSfxVolume);
        }

        // 10 detik terakhir: merah + tick SFX
        if (_missionRemain <= 10f && _missionRemain > 0f)
        {
            if (missionTimerText && _timerColorCached) missionTimerText.color =
Color.red;
            int whole = Mathf.CeilToInt(_missionRemain); // 10..1
            if (whole != _lastTickSecond)
            {
                _lastTickSecond = whole;
                if          (sfxSource          &&          countdownTickSfx)
sfxSource.PlayOneShot(countdownTickSfx, uiSfxVolume);
            }
        }
        else
```

```
        {
            if (missionTimerText && _timerColorCached && _missionRemain > 10f)
                missionTimerText.color = _timerDefaultColor;
        }

        // Menang kalau semua terdeteksi
        int detected = CountDetected();
        int targetCount = Mathf.Max(totalTargetsToDetect, targets != null ? targets.Length : 0);
        if (detected >= targetCount)
        {
            MissionComplete();
            return;
        }

        // Waktu habis
        if (_missionRemain <= 0f)
        {
            if (_lastTickSecond != 0)
            {
                _lastTickSecond = 0;
                if (sfxSource && countdownFinalSfx) sfxSource.PlayOneShot(countdownFinalSfx, uiSfxVolume);
            }
            GameOver();
            return;
        }
    }
}

private IEnumerator FlashPanel(GameObject panel, float seconds)
{
    panel.SetActive(true);
    yield return new WaitForSeconds(seconds);
    panel.SetActive(false);
}

private int CountDetected()
{
    int c = 0;
    if (targets != null)
        foreach (var t in targets)
            if (t != null && t.hasDetected) c++;
```

```
        return c;
    }

    private void MissionComplete()
    {
        _state = MissionState.Completed;
        PauseGame(true);
        if (drone) drone.TriggerReturnToBase();

        string body = BuildDetectionSummaryText();
        if (missionCompleteBodyText) missionCompleteBodyText.text = body;
        if (missionCompletePanel) missionCompletePanel.SetActive(true);

        if (missionTimerText && _timerColorCached) missionTimerText.color =
_timerDefaultColor;
    }

    // overload: bisa dipanggil tanpa alasan atau dengan pesan
    private void GameOver(string reason = null)
    {
        _state = MissionState.GameOver;
        PauseGame(true);

        string body = BuildDetectionSummaryText();
        if (!string.IsNullOrEmpty(reason))
            body = reason + "\n\n" + body;

        if (gameOverBodyText) gameOverBodyText.text = body;
        if (gameOverPanel) gameOverPanel.SetActive(true);

        if (missionTimerText && _timerColorCached) missionTimerText.color =
_timerDefaultColor;
    }

    private string BuildDetectionSummaryText()
    {
        var sb = new StringBuilder();
        int detected = _ordered.Count;
        int targetCount = Mathf.Max(totalTargetsToDetect, targets != null ?
targets.Length : 0);

        sb.AppendLine($"Terdeteksi: {detected} / {targetCount}");
        if (_ordered.Count > 0)
        {
```

```
            for (int i = 0; i < _ordered.Count; i++)
            {
                var r = _ordered[i];
                string coord = (r.lat != 0 || r.lon != 0)
                    ? $"Lon={r.lon:F6}, Lat={r.lat:F6}"
                    : $"X={r.worldPos.x:F2}, Z={r.worldPos.z:F2}";
                sb.AppendLine($"{i + 1}. {r.name} (ID: {r.personId})");
                sb.AppendLine($"    {coord} | {r.firstDistance:F1} m");
            }
        }
        else sb.AppendLine("Belum ada personel yang terdeteksi.");
        return sb.ToString();
    }

    // ====================== Util ======================
    private void PauseGame(bool pause)
    {
        Time.timeScale = pause ? 0f : 1f;
        AudioListener.pause = pause;
    }

    private void ClearAllDetections()
    {
        _byId.Clear();
        _ordered.Clear();
        if (targets != null)
            foreach (var t in targets) if (t != null) t.hasDetected = false;
    }
}
```

Lampiran 3. DroneUI.cs

Script ini digunakan untuk menampilkan antarmuka utama permainan atau *Heads-Up Display (HUD)*. Informasi yang ditampilkan meliputi status mesin drone, ketinggian, arah hadap, kecepatan, serta daftar personel yang telah berhasil terdeteksi.

```
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;
using System.Text;
using System.Collections.Generic;

public class DroneUI : MonoBehaviour
```

```
{
    public DroneController drone;

    [Header("Dynamic Status (HUD opsional)")]
    public TMP_Text dynamicStatusText;
    public float refreshInterval = 0.25f;
    private float _nextRefreshAt = 0f;
    private Vector3 _prevPos;
    private float _smoothedSpeed;

    [Header("Detection Summary (Text Only)")]
    public DroneGameManager gameManager;      // <- ganti RFIDDetector ke
DroneGameManager
    public GameObject detectionPanel;
    public TMP_Text detectionText;
    public string detectionHeader = "PERSONEL TERDETEKSI";
    public int maxEntries = 10;

    [Header("Geo Mapping")]
    public GeoMapper geo;

    private void Awake()
    {
        if                (!gameManager)                gameManager                =
FindObjectOfType<DroneGameManager>();
    }

    void Update()
    {
        // ===== Dynamic HUD (opsional) =====
        if (dynamicStatusText != null && drone != null && Time.unscaledTime >=
_nextRefreshAt)
        {
            _nextRefreshAt = Time.unscaledTime + refreshInterval;

            bool engine = drone.EngineOn();
            bool flying = drone.IsFlying();

            float altitude = drone.transform.position.y;
            float heading = drone.transform.eulerAngles.y;

            // Ambil speed dari Rigidbody (sudah ter-clamp 13.3–26.9 m/s)
            float targetSpeed = drone.GetHorizontalSpeed();
            _smoothedSpeed = Mathf.Lerp(_smoothedSpeed, targetSpeed, 0.25f);
```

```
        string mode = engine ? (flying ? "Flying" : "Idle") : "Engine OFF";
        dynamicStatusText.text =
            $"Engine   : {(engine ? "<color=green>ON</color>" : "<color=red>OFF</color>")}\n" +
            $"Mode   : {mode}\n" +
            $"Alt    : {altitude:F1} m\n" +
            $"Head   : {heading:0}°\n" +
            $"Speed  : {_smoothedSpeed:0.0} m/s";
    }

    // ===== Detection Summary (ambil dari GameManager.targets) =====
    if (gameManager == null || gameManager.targets == null || gameManager.targets.Length == 0
        || detectionPanel == null || detectionText == null)
    {
        if (detectionPanel != null && detectionPanel.activeSelf) detectionPanel.SetActive(false);
        return;
    }

    var recs = new List<(string name, string id, string coord, float dist)>();
    foreach (var t in gameManager.targets)
    {
        if (t == null || t.target == null) continue;

        // radius & pusat dari GameManager
        Vector3 a = gameManager.transform.position; a.y = 0f;
        Vector3 b = t.target.position; b.y = 0f;
        float dist = Vector3.Distance(a, b);

        if (dist <= gameManager.detectionRadius)
        {
            string nama = t.target.name;
            string id = "N/A";
            var info = t.target.GetComponent<PersonelInfo>();
            if (info != null)
            {
                if (!string.IsNullOrEmpty(info.nama)) nama = info.nama;
                if (!string.IsNullOrEmpty(info.personelID)) id = info.personelID;
            }

            string coordLine;
            if (geo != null)
```

```
            {
                geo.WorldToGeo(t.target.position, out double lat, out double lon);
                coordLine = $"Lon={lon:F6}, Lat={lat:F6}";
            }
            else
            {
                Vector3 p = t.target.position;
                coordLine = $"X={p.x:F2}, Z={p.z:F2}";
            }

            recs.Add((nama, id, coordLine, dist));
            if (recs.Count >= maxEntries) break;
        }
    }

    if (recs.Count > 0)
    {
        if (!detectionPanel.activeSelf) detectionPanel.SetActive(true);

        StringBuilder sb = new StringBuilder();
        if (!string.IsNullOrEmpty(detectionHeader))
            sb.AppendLine(detectionHeader);

        for (int i = 0; i < recs.Count; i++)
        {
            var r = recs[i];
            sb.AppendLine($"{i + 1}. {r.name} (ID: {r.id})");
            sb.AppendLine($"    {r.coord} | {r.dist:F1} m");
        }

        detectionText.text = sb.ToString();
    }
    else
    {
        if (detectionPanel.activeSelf) detectionPanel.SetActive(false);
        detectionText.text = "";
    }
}

public void back_to_menu()
{
    SceneManager.LoadScene("MainMenu");
}
```

```
    public List<string> GetAllDetectedRecords()
    {
        var recs = new List<string>();
        if (gameManager == null || gameManager.targets == null) return recs;

        foreach (var t in gameManager.targets)
        {
            if (t == null || t.target == null) continue;

            Vector3 a = gameManager.transform.position; a.y = 0f;
            Vector3 b = t.target.position; b.y = 0f;
            float dist = Vector3.Distance(a, b);

            if (dist <= gameManager.detectionRadius)
            {
                string nama = t.target.name;
                string id = "N/A";
                var info = t.target.GetComponent<PersonelInfo>();
                if (info != null)
                {
                    if (!string.IsNullOrEmpty(info.nama)) nama = info.nama;
                    if (!string.IsNullOrEmpty(info.personelID)) id = info.personelID;
                }

                string coordLine;
                if (geo != null)
                {
                    geo.WorldToGeo(t.target.position, out double lat, out double lon);
                    coordLine = $"Lon={lon:F6}, Lat={lat:F6}";
                }
                else
                {
                    Vector3 p = t.target.position;
                    coordLine = $"X={p.x:F2}, Z={p.z:F2}";
                }

                recs.Add($"{nama} (ID: {id}) | {coordLine} | {dist:F1} m");
            }
        }
        return recs;
    }
}
```