# LAMPIRAN

`

**MainMenuControlle.cs**

```
using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

using System.Collections;

public class MainMenuController :
MonoBehaviour

{

    public GameObject menuPanel;

    public GameObject infoPanel;

    public GameObject
pengaturanPanel;

    public GameObject
confirmExitPanel;  // Panel
konfirmasi keluar

    public GameObject loadingPanel;
// Panel loading

    public Slider loadingSlider;
// Slider untuk menunjukkan progres

    // Fungsi ini dipanggil saat
tombol "Mulai" ditekan

    public void OnMulaiButton()

    {


StartCoroutine(LoadGameSceneAsync());

    }

    // Fungsi untuk loading scene
secara asinkron

    private IEnumerator
LoadGameSceneAsync()

    {

        // Menampilkan panel loading

        loadingPanel.SetActive(true);



        // Mulai proses loading scene
"game"

        AsyncOperation asyncLoad =
SceneManager.LoadSceneAsync("game");

        // Jangan otomatis
menampilkan scene setelah selesai


asyncLoad.allowSceneActivation =
false;

        // Variabel untuk menunjukkan
progres loading

        float progress = 0f;

        // Proses loading dengan
update progress slider

        while (!asyncLoad.isDone)

        {

            // Update slider secara
perlahan agar progres lebih halus

            if (asyncLoad.progress <
0.9f)

            {

                progress =
Mathf.Lerp(progress,
asyncLoad.progress, Time.deltaTime *
2); // Lebih halus

            }

            else

            {

                // Setelah slider
mendekati 0.9, progresnya bergerak
menuju 1 lebih cepat

                progress =
Mathf.Lerp(progress, 1f,
Time.deltaTime * 2);

            }

            // Update slider dengan
progres yang lebih halus

            loadingSlider.value =
progress;

            // Ketika progres
mencapai 90% (0.9), otomatis pindah
ke scene berikutnya

            if (asyncLoad.progress >=
0.9f &&
!asyncLoad.allowSceneActivation)

            {

 asyncLoad.allowSceneActivation =
true;

            }

            yield return null;
```

`

```
        // Setelah selesai,
sembunyikan panel loading

loadingPanel.SetActive(false);

    }

    // Fungsi ini dipanggil saat
tombol "Pengaturan" ditekan

    public void OnPengaturanButton()

    {

        menuPanel.SetActive(false);

     pengaturanPanel.SetActive(true);

    }

    // Fungsi ini dipanggil saat
tombol "Info" ditekan

    public void OnInfoButton()

    {

        menuPanel.SetActive(false);

        infoPanel.SetActive(true);

    }

    // Fungsi ini dipanggil saat
tombol "Kembali" ditekan dari Info
Panel

    public void
OnKembaliKeMenuButton()

    {

        infoPanel.SetActive(false);

        menuPanel.SetActive(true);

pengaturanPanel.SetActive(false);

    }

    // Fungsi ini dipanggil saat
tombol "Keluar" ditekan

    public void OnKeluarButton()

    {

    confirmExitPanel.SetActive(true);
// Tampilkan panel konfirmasi keluar

        menuPanel.SetActive(false);
// Menyembunyikan menu utama

    }

    // Fungsi ini dipanggil ketika
tombol "Ya" pada konfirmasi keluar
ditekan

    public void OnConfirmExitButton()

    {

        Application.Quit();

#if UNITY_EDITOR

UnityEditor.EditorApplication.isPlayi
ng = false;

#endif

    }

    // Fungsi ini dipanggil ketika
tombol "Tidak" pada konfirmasi keluar
ditekan

    public void OnCancelExitButton()

    {

confirmExitPanel.SetActive(false);
// Menyembunyikan panel konfirmasi
keluar

        menuPanel.SetActive(true);
// Menampilkan kembali menu utama

    }

}
```

**AudioManager.cs**

```
using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class AudioManager :
MonoBehaviour

{

    public AudioSource audioSource;

    public Slider volumeSlider;

    private static AudioManager
instance;

    void Start()

    {

        float savedVolume =
PlayerPrefs.GetFloat("Volume",
0.75f);

        SetVolume(savedVolume);

FindSliderAndBind(savedVolume);

    }
```

`

```
    void OnSceneLoaded(Scene scene,
LoadSceneMode mode)
    {
        float currentVolume =
audioSource.volume;

FindSliderAndBind(currentVolume);
    }
    private void
FindSliderAndBind(float volume)
    {
        GameObject sliderObj =
GameObject.FindGameObjectWithTag("Vol
umeSlider");
        if (sliderObj != null)
        {
            volumeSlider =
sliderObj.GetComponent<Slider>();
            if (volumeSlider != null)
            {
                volumeSlider.value =
volume;

volumeSlider.onValueChanged.RemoveAll
Listeners();

volumeSlider.onValueChanged.AddListen
er(OnVolumeChanged);
            }
        }
    }
    public void OnVolumeChanged(float
volume)
    {
        SetVolume(volume);

PlayerPrefs.SetFloat("Volume",
volume);
    }
    private void SetVolume(float
volume)
    {
        if (audioSource != null)
            audioSource.volume =
volume;
    }
}
```

**UICanvasVideoPlay.cs**

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Video;
using System.Collections;
public class UICanvasVideoPlayer :
MonoBehaviour
{
    [Header("Komponen UI")]
    public GameObject videoPanel;
    public VideoPlayer videoPlayer;
    public Button closeButton;
    private bool isVideoPlaying =
false;
    void Start()
    {
        if (videoPanel != null &&
videoPlayer != null)
        {

videoPanel.SetActive(true);
            isVideoPlaying = true;

StartCoroutine(PlayVideoAndPauseGame(
));

videoPlayer.loopPointReached +=
OnVideoFinished;
            if (closeButton != null)

closeButton.onClick.AddListener(Close
Video);
        }
        else
        {
            Debug.LogError("Komponen
belum lengkap.");
        }
```

`

```
    }

    IEnumerator
PlayVideoAndPauseGame()

    {

        // Pastikan semua sistem
jalan sebelum pause

        yield return null;

        // Tampilkan kursor dan
bebaskan

        //Cursor.visible = true;

        //Cursor.lockState =
CursorLockMode.None;

        videoPlayer.Play();

        yield return null; // Tunggu
frame berikutnya agar video sempat
start

        Time.timeScale = 0f;

        Debug.Log("Time.timeScale =
0, dunia dihentikan.");

    }

    void OnVideoFinished(VideoPlayer
vp)

    {

        CloseVideo();

    }

    public void CloseVideo()

    {

        if (!isVideoPlaying) return;

        isVideoPlaying = false;

        videoPlayer.Stop();

        videoPanel.SetActive(false);

        // Lanjutkan dunia

        Time.timeScale = 1f;

        Debug.Log("Time.timeScale =
1, dunia lanjut.");

        // Kunci kembali kursor jika
kamu memang ingin seperti itu

        //Cursor.visible = false;

        //Cursor.lockState =
CursorLockMode.Locked;


        gameObject.SetActive(false);
    }

}
```

**ThirdPersonController**

```
using UnityEngine;

public class ThirdPersonController :
MonoBehaviour

{

    [Tooltip("Speed at which the
character moves. It is not affected
by gravity or jumping.")]

    public float velocity = 5f;

    [Tooltip("This value is added to
the speed value while the character
is sprinting.")]

    public float sprintAdittion =
3.5f;

    [Tooltip("The higher the value,
the higher the character will
jump.")]

    public float jumpForce = 18f;

    [Tooltip("Stay in the air. The
higher the value, the longer the
character floats before falling.")]

    public float jumpTime = 0.85f;

    [Space]

    [Tooltip("Force that pulls the
player down. Changing this value
causes all movement, jumping and
falling to be changed as well.")]

    public float gravity = 9.8f;

    float jumpElapsedTime = 0;

    // Player states

    bool isJumping = false;

    bool isSprinting = false;

    bool isCrouching = false;

    // Inputs

    float inputHorizontal;

    float inputVertical;

    bool inputJump;

    bool inputCrouch;

    bool inputSprint;
```

`

```
    Animator animator;

    CharacterController cc

    float airTimeBuffer = 0.1f;

    float airTimeElapsed = 0;

    bool isInAir = false;

    bool isAttacking = false;

    [Header("Attack Settings")]

    public float attackRange = 2f;

    public float attackDamage = 25f;

    public LayerMask enemyLayer;

    // Tambahan: attack cooldown

    public float attackCooldown = 2f;
// waktu cooldown dalam detik

    private float attackTimer = 0f;

    public FixedJoystick
fixedJoystick; // atau
FloatingJoystick jika itu yang kamu
pakai

    void Start()

    {

        cc =
GetComponent<CharacterController>();

        animator =
GetComponent<Animator>();

        // Message informing the user
that they forgot to add an animator

        if (animator == null)

            Debug.LogWarning("Hey
buddy, you don't have the Animator
component in your player. Without it,
the animations won't work.");

    }

    bool IsReallyGrounded()

    {

        float rayLength = 0.2f;

        Vector3 rayOrigin =
transform.position + Vector3.up *
0.1f;

        return
Physics.Raycast(rayOrigin,
Vector3.down, rayLength);

    }

    // Update is only being used here
to identify keys and trigger
animations

    void Update()

    {

        // update attack timer tiap
frame

        if (attackTimer > 0)

            attackTimer -=
Time.deltaTime;

        // Input checkers

        //inputHorizontal =
Input.GetAxis("Horizontal");

        //inputVertical =
Input.GetAxis("Vertical");

        //inputJump =
Input.GetAxis("Jump") == 1f;

        //inputSprint =
Input.GetAxis("Fire3") == 1f;

        inputHorizontal =
fixedJoystick.Horizontal;

        inputVertical =
fixedJoystick.Vertical;

        inputJump =
VirtualButtonController.isJumping;

        inputSprint =
VirtualButtonController.isRunning;

        // Unfortunately GetAxis does
not work with GetKeyDown, so inputs
must be taken individually

        inputCrouch =
Input.GetKeyDown(KeyCode.LeftControl)
||
Input.GetKeyDown(KeyCode.JoystickButt
on1)

        // Trigger attack

        AnimatorStateInfo state =
animator.GetCurrentAnimatorStateInfo(
0);

        bool isAttackState =
state.IsTag("Attack");

        if (isAttackState)

        {

            isAttacking = true;

        }
```

`

```
      else
        {
            isAttacking = false;
        }
        //if
(Input.GetMouseButtonDown(0) &&
animator != null &&
IsReallyGrounded() && !isAttacking &&
attackTimer <= 0f)
        if
(VirtualButtonController.isAttacking
&& animator != null &&
IsReallyGrounded() && !isAttacking &&
attackTimer <= 0f)
        {

animator.SetTrigger("attack");
        TryHitEnemy();
        attackTimer = attackCooldown;
// reset timer setelah menyerang
        }
        // Optional: prevent movement
while attacking
        if (isAttacking)
        {
            inputHorizontal = 0;
            inputVertical = 0;
            inputJump = false; //
Tidak bisa lompat saat menyerang
        }
        // Check if you pressed the
crouch input key and change the
player's state
        if (inputCrouch)
            isCrouching =
!isCrouching;
        // Run and Crouch animation
        // If dont have animator
component, this block wont run
        if (cc.isGrounded && animator
!= null)
        {
            // Crouch
            // Note: The crouch
animation does not shrink the
character's collider

animator.SetBool("crouch",
isCrouching);
            // Run
            float minimumSpeed =
0.9f;
            animator.SetBool("run",
cc.velocity.magnitude >
minimumSpeed);
            // Sprint
            isSprinting =
cc.velocity.magnitude > minimumSpeed
&& inputSprint;

animator.SetBool("sprint",
isSprinting);
        }
        // Jump animation
        if (animator != null)
            animator.SetBool("air",
!IsReallyGrounded());
        // Handle can jump or not
        if (inputJump &&
cc.isGrounded)
        {
            isJumping = true;
            // Disable crounching
when jumping
            //isCrouching = false;
        }
        HeadHittingDetect();
        // Reset input satu kali
tekan

VirtualButtonController.isJumping =
false;

VirtualButtonController.isAttacking =
false;
    }
    void TryHitEnemy()
    {
```

`

```
        RaycastHit hit;

        Vector3 origin =
transform.position + Vector3.up * 1f;
// sedikit lebih tinggi dari kaki

        Vector3 direction =
transform.forward;

        if (Physics.Raycast(origin,
direction, out hit, attackRange,
enemyLayer))

        {

            EnemyHealth enemy =
hit.collider.GetComponent<EnemyHealth
>();

            if (enemy != null)

            {


enemy.TakeDamage(attackDamage);

                Debug.Log("Terkena
Damage: " + attackDamage + " Sisa HP:
" + enemy.health);

            }

        }

        // Debug

        Debug.DrawRay(origin,
direction * attackRange, Color.red,
1f);

    }

    // With the inputs and animations
defined, FixedUpdate is responsible
for applying movements and actions to
the player

    private void FixedUpdate()

    {

        // Sprinting velocity boost
or crounching desacelerate

        float velocityAdittion = 0;

        if (isSprinting)

            velocityAdittion =
sprintAdittion;

        if (isCrouching)

            velocityAdittion = -
(velocity * 0.50f); // -50% velocity

        // Direction movement

        float directionX =
inputHorizontal * (velocity +
velocityAdittion) * Time.deltaTime;

        float directionZ =
inputVertical * (velocity +
velocityAdittion) * Time.deltaTime;

        float directionY = 0;



        // Jump handler

        if (isJumping)

        {

            // Apply inertia and
smoothness when climbing the jump

            // It is not necessary
when descending, as gravity itself
will gradually pulls

            directionY =
Mathf.SmoothStep(jumpForce, jumpForce
* 0.30f, jumpElapsedTime / jumpTime)
* Time.deltaTime;

            // Jump timer

            jumpElapsedTime +=
Time.deltaTime;

            if (jumpElapsedTime >=
jumpTime)

            {

                isJumping = false;

                jumpElapsedTime = 0;

            }

        }

        // Add gravity to Y axis

        directionY = directionY -
gravity * Time.deltaTime;

        // --- Character rotation ---

        Vector3 forward =
Camera.main.transform.forward;

        Vector3 right =
Camera.main.transform.right;

        forward.y = 0;

        right.y = 0;

        forward.Normalize();

        right.Normalize();

        // Relate the front with the
Z direction (depth) and right with X
(lateral movement)

        forward = forward *
directionZ;
```

`

```
       right = right * directionX;

        if (directionX != 0 ||
directionZ != 0)

        {

            float angle =
Mathf.Atan2(forward.x + right.x,
forward.z + right.z) * Mathf.Rad2Deg;

            Quaternion rotation =
Quaternion.Euler(0, angle, 0);

            transform.rotation =
Quaternion.Slerp(transform.rotation,
rotation, 0.15f);

        }

        // --- End rotation ---

        Vector3 verticalDirection =
Vector3.up * directionY;

        Vector3 horizontalDirection =
forward + right;

        Vector3 moviment =
verticalDirection +
horizontalDirection;

        cc.Move(moviment);

    }

    //This function makes the
character end his jump if he hits his
head on something

    void HeadHittingDetect()

    {

        float headHitDistance = 1.1f;

        Vector3 ccCenter =
transform.TransformPoint(cc.center);

        float hitCalc = cc.height /
2f * headHitDistance;



        // Uncomment this line to see
the Ray drawed in your characters
head

        // Debug.DrawRay(ccCenter,
Vector3.up * headHeight, Color.red);



        if (Physics.Raycast(ccCenter,
Vector3.up, hitCalc))

        {

            jumpElapsedTime = 0;

            isJumping = false;

        }

    }

}
```

**WanderingAI.cs**

```
using UnityEngine;

using UnityEngine.AI;

[RequireComponent(typeof(Animator))]

[RequireComponent(typeof(NavMeshAgent
))]

public class WanderingAI :
MonoBehaviour

{

    public float moveSpeed = 2f;

    protected float wanderRadius =
50f;

    public float minIdleTime = 3f;

    protected float maxIdleTime =
20f;

    private float idleTimer = 0f;

    private float currentIdleTime;

    private bool isIdle = true;

    private Vector3 targetPosition;

    private Animator animator;

    private NavMeshAgent agent;

    void Start()

    {

        animator =
GetComponent<Animator>();

        agent =
GetComponent<NavMeshAgent>();

        agent.speed = moveSpeed;

        agent.stoppingDistance =
0.1f;

        SetRandomIdleTime();

    }

    void Update()

    {

        if (isIdle)

        {
```

`

```
            idleTimer +=
Time.deltaTime;

animator.SetBool("isWalking", false);
            if (idleTimer >=
currentIdleTime)
            {
                PickNewDestination();
            }
        }
        else
        {
            if (!agent.pathPending &&
agent.remainingDistance <=
agent.stoppingDistance)
            {
                isIdle = true;
                idleTimer = 0f;
                SetRandomIdleTime();
            }
            if
(agent.velocity.sqrMagnitude > 0.1f)
            {

animator.SetBool("isWalking", true);
                // Smoothly rotate to
face direction
                Quaternion targetRot
=
Quaternion.LookRotation(agent.velocit
y.normalized);
                transform.rotation =
Quaternion.Slerp(transform.rotation,
targetRot, Time.deltaTime * 5f);
            }
        }
    }
    void SetRandomIdleTime()
    {
        currentIdleTime =
Random.Range(minIdleTime,
maxIdleTime);
    }
    void PickNewDestination()
    {
        Vector3 randomDirection =
Random.insideUnitSphere *
wanderRadius;
        randomDirection +=
transform.position;
        NavMeshHit navHit;
        if
(NavMesh.SamplePosition(randomDirecti
on, out navHit, wanderRadius,
NavMesh.AllAreas))
        {
            targetPosition =
navHit.position;

agent.SetDestination(targetPosition);
            isIdle = false;
            // Debug.Log("Destination
set: " + targetPosition);
        }
        else
        {
            // Try again shortly

Invoke(nameof(PickNewDestination),
0.2f);
            // Debug.Log("Failed to
find valid destination,
retrying...");
        }
    }
}
```

**WolfEnemyA.cs**

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class WolfEnemyAI :
EnemyHealth
{
    protected override void Start()
    {
        base.Start();
```

`

```
        Howl();
    }
    private void Howl()
    {
        animator.SetTrigger("Howl");
    }
    protected override void
PlayAttackAnimation()
    {
        int attackType =
Random.Range(0, 2);
        if (attackType == 0)

animator.SetTrigger("Attack1");
        else

animator.SetTrigger("Attack2");
    }
}
```

**EnemyHealth.cs**

```
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.SceneManagement;
public abstract class EnemyHealth :
MonoBehaviour
{
    [Header("Components")]
    public Animator animator;
    public NavMeshAgent agent;
    public Transform player;
    [Header("Stats")]
    public float chaseRange = 10f;
    public float attackRange = 2f;
    public float health = 100f;
    public float damageAmount = 20f;
    protected bool isDead = false;
    protected bool isAttacking =
false;
    protected PlayerHealth
playerHealth;
    protected virtual void Start()
    {
        if (agent == null) agent =
GetComponent<NavMeshAgent>();
        if (animator == null)
animator = GetComponent<Animator>();
        if (player == null) player =
GameObject.FindGameObjectWithTag("Pla
yer").transform;
        playerHealth =
player.GetComponent<PlayerHealth>();
    }
    protected virtual void Update()
    {
        if (isDead || isAttacking)
return;
        float distance =
Vector3.Distance(transform.position,
player.position);
        if (distance <= attackRange)
        {
            Attack();
        }
        else if (distance <=
chaseRange)
        {
            ChasePlayer();
        }
        else
        {
            Idle();
        }
    }
    protected virtual void Idle()
    {
        // Check if the agent is
valid and on the NavMesh before
stopping it
        if (agent.isActiveAndEnabled
&& agent.isOnNavMesh)
```

```
        {
            agent.isStopped = true;
        }
        animator.SetFloat("Forward", 0f);  // No movement
        animator.SetFloat("Turn", 0f);     // No rotation
        animator.SetBool("IsWalking", false);  // No walking
        animator.SetBool("IsRunning", false);  // No running
    }
    protected virtual void ChasePlayer()
    {
        if (agent.isActiveAndEnabled && agent.isOnNavMesh)
        {
            agent.isStopped = false;

agent.SetDestination(player.position);
        }
        // Calculate the movement speed for animation
        float speed = agent.velocity.magnitude;
        // Update Forward parameter based on movement speed (Walking or Running)
        if (speed > 1f) // Running
        {

animator.SetFloat("Forward", 1f);

animator.SetBool("IsRunning", true);

animator.SetBool("IsWalking", false); // Ensure this matches the parameter in Animator
        }
        else if (speed > 0.1f) // Walking
        {

animator.SetFloat("Forward", -1f);

animator.SetBool("IsRunning", false);

animator.SetBool("IsWalking", true); // Ensure this matches the parameter in Animator
        }
        else
        {

animator.SetFloat("Forward", 0f); // No movement

animator.SetBool("IsRunning", false);

animator.SetBool("IsWalking", false); // Ensure this matches the parameter in Animator
        }
        // Adjust the turn direction
        float turnDirection = Vector3.Dot(transform.right, player.position - transform.position);
        animator.SetFloat("Turn", Mathf.Sign(turnDirection)); // Set the turn direction
    }
    protected virtual void Attack()
    {
        if (agent.isActiveAndEnabled && agent.isOnNavMesh)
        {
            agent.isStopped = true;
        }
        animator.SetBool("IsRunning", false);
        transform.LookAt(player);
        isAttacking = true;
        PlayAttackAnimation();

        Invoke(nameof(DealDamage), 0.5f);
```

```
        Invoke(nameof(ResetAttack),
1.5f);

    }

    protected abstract void
PlayAttackAnimation();

    protected virtual void
DealDamage()

    {

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

        if (distance <= attackRange
&& playerHealth != null)

        {


playerHealth.TakeDamage(damageAmount)
;

        }

    }

    protected virtual void
ResetAttack()

    {

        isAttacking = false;

    }

    public virtual void
TakeDamage(float amount)

    {

        if (isDead) return;

        health -= amount;

       animator.SetTrigger("Damage");

        if (health <= 0)

        {

            Die();

        }

        Debug.Log(gameObject.name + "
Terkena damage: " + amount);

    }

    public virtual void Die()

    {

        isDead = true;

        if (agent.isActiveAndEnabled
&& agent.isOnNavMesh)

        {

            agent.isStopped = true;

        }

        animator.SetTrigger("Die");

        animator.SetBool("IsLived",
false);

        // Transition to a new scene
when the enemy dies


//SceneManager.LoadScene("game2");

        //Destroy(gameObject, 5f);

    }

}
```

**ChickenFollower.cs**

```
using UnityEngine;

public class ChickenFollower :
MonoBehaviour

{

    [Header("Target Settings")]

    public Transform player; //
Target yang akan diikuti (player)

    [Header("Movement Settings")]

    public float followDistance = 3f;
// Jarak minimum untuk mulai
mengikuti

    public float walkSpeed = 2f;

    public float runSpeed = 5f;

    public float runDistance = 8f; //
Jarak untuk mulai berlari

    public float rotationSpeed = 5f;

    public float stopDistance = 1.5f;
// Jarak untuk berhenti total

    [Header("Animation Settings")]

    public Animator animator;

    [Header("Animation Smoothing")]

    public float animationSmoothTime
= 0.1f; // Waktu untuk smooth
transition



    // Parameter nama animasi -
sesuaikan dengan Animator Controller
Anda
```

```
    private readonly string
ANIM_SPEED = "Speed";

    private readonly string
ANIM_IS_MOVING = "IsMoving";

    private readonly string
ANIM_IS_RUNNING = "IsRunning";

    private readonly string
ANIM_NORMALIZED_SPEED =
"NormalizedSpeed";

    private float currentSpeed;

    private float animationSpeed; //
Kecepatan untuk animasi (smooth)

    private float speedVelocity; //
Untuk SmoothDamp

    private Vector3 targetPosition;

    private bool isMoving;

    private bool isRunning;

    private Vector3 lastPosition;

    private float actualSpeed; //
Kecepatan aktual berdasarkan
pergerakan

    void Start()

    {

        InitializeComponents();

        lastPosition =
transform.position;

    }

    void InitializeComponents()

    {

        // Jika animator tidak di-
assign, coba ambil dari komponen ini

        if (animator == null)

        {

            animator =
GetComponent<Animator>();

            if (animator == null)

            {


Debug.LogWarning("Animator tidak
ditemukan pada " + gameObject.name);

            }

        }

        // Jika player tidak di-
assign, cari GameObject dengan tag
"Player"

        if (player == null)

        {

            GameObject playerObject =
GameObject.FindGameObjectWithTag("Pla
yer");

            if (playerObject != null)

            {

                player =
playerObject.transform;

            }

            else

            {


Debug.LogWarning("Player tidak
ditemukan! Pastikan ada GameObject
dengan tag 'Player'");

            }

        }

    }

    void Update()

    {

        if (player == null) return;

        CalculateActualSpeed();

        FollowPlayer();

        UpdateAnimation();

    }

    void CalculateActualSpeed()

    {

        // Hitung kecepatan aktual
berdasarkan pergerakan

        actualSpeed =
Vector3.Distance(transform.position,
lastPosition) / Time.deltaTime;

        lastPosition =
transform.position;

    }

    void FollowPlayer()

    {
```

```
        float distanceToPlayer =
Vector3.Distance(transform.position,
player.position);

        if (distanceToPlayer >
followDistance)

        {

            // Tentukan target
position dengan offset yang lebih
natural

            Vector3 directionToPlayer
= (player.position -
transform.position).normalized;

            targetPosition =
player.position - directionToPlayer *
(followDistance * 0.7f);

            // Tentukan kecepatan dan
status berdasarkan jarak

            if (distanceToPlayer >
runDistance)

            {

                currentSpeed =
runSpeed;

                isRunning = true;

            }

            else if (distanceToPlayer
> stopDistance)

            {

                currentSpeed =
walkSpeed;

                isRunning = false;

            }

            else

            {

                currentSpeed =
walkSpeed * 0.5f; // Pelan-pelan saat
mendekati

                isRunning = false;

            }

            // Gerakkan ayam

            Vector3 newPosition =
Vector3.MoveTowards(transform.positio
n, targetPosition, currentSpeed *
Time.deltaTime);

            transform.position =
newPosition;

            // Rotasi menghadap arah
gerakan

            RotateTowardsMovement();

            isMoving = true;

        }

        else

        {

            // Berhenti jika sudah
dekat

            currentSpeed = 0f;

            isMoving = false;

            isRunning = false;

        }

    }

    void RotateTowardsMovement()

    {

        Vector3 moveDirection =
(targetPosition -
transform.position).normalized;

        if (moveDirection !=
Vector3.zero)

        {

            // Hanya rotasi pada
sumbu Y (horizontal)

            moveDirection.y = 0;

            Quaternion targetRotation
=
Quaternion.LookRotation(moveDirection
);

            transform.rotation =
Quaternion.Slerp(transform.rotation,
targetRotation, rotationSpeed *
Time.deltaTime);

        }

    }

    void UpdateAnimation()

    {

        if (animator == null) return;

        Vector3 movementDirection =
(targetPosition -
transform.position).normalized;

        Vector3 localDirection =
transform.InverseTransformDirection(m
ovementDirection);
```

```
        float turn =
Mathf.Clamp(localDirection.x, -1f,
1f);

        // Sprinting logic — blend
from 1 to 2 if running

        float baseForward =
Mathf.Clamp(localDirection.z, -1f,
1f);

        float targetForward =
isRunning ? baseForward * 2f :
baseForward;

        // Smooth damping

        float currentForward =
animator.GetFloat("Forward");

        float smoothedForward =
Mathf.Lerp(currentForward,
targetForward, Time.deltaTime * 5f);

        animator.SetFloat("Forward",
isMoving ? smoothedForward : 0f,
animationSmoothTime, Time.deltaTime);

        animator.SetFloat("Turn",
isMoving ? turn : 0f,
animationSmoothTime, Time.deltaTime);

    }

    void
UpdateAnimationParameters(float
normalizedSpeed)

    {

        // Parameter dasar

        animator.SetFloat(ANIM_SPEED,
animationSpeed);


animator.SetFloat(ANIM_NORMALIZED_SPE
ED, normalizedSpeed);


animator.SetBool(ANIM_IS_MOVING,
isMoving && animationSpeed > 0.1f);


animator.SetBool(ANIM_IS_RUNNING,
isRunning && animationSpeed >
walkSpeed * 0.8f);

                // Parameter tambahan
untuk kontrol yang lebih baik

        if
(HasParameter("WalkSpeed"))

        {

            float walkNormalized =
Mathf.Clamp01(animationSpeed /
walkSpeed);


animator.SetFloat("WalkSpeed",
walkNormalized);

        }

                if
(HasParameter("RunSpeed"))

        {

            float runNormalized =
isRunning ?
Mathf.Clamp01(animationSpeed /
runSpeed) : 0f;


animator.SetFloat("RunSpeed",
runNormalized);

        }

    }

        // Helper method untuk
mengecek apakah parameter ada

    bool HasParameter(string
paramName)

    {

        if (animator == null) return
false;

        foreach
(AnimatorControllerParameter param in
animator.parameters)

        {

            if (param.name ==
paramName)

                return true;

        }

        return false;

    }

        // Fungsi public untuk
kontrol eksternal

    public void SetTarget(Transform
newTarget)

    {

        player = newTarget;

    }
```

```
    public void
SetFollowDistance(float newDistance)
    {
        followDistance =
Mathf.Max(0.5f, newDistance);
    }
        public void SetSpeeds(float
newWalkSpeed, float newRunSpeed)
    {
        walkSpeed = Mathf.Max(0.1f,
newWalkSpeed);
        runSpeed =
Mathf.Max(walkSpeed, newRunSpeed);
    }
        public void PauseFollowing()
    {
        enabled = false;
        if (animator != null)
        {

animator.SetFloat(ANIM_SPEED, 0f);

animator.SetBool(ANIM_IS_MOVING,
false);

animator.SetBool(ANIM_IS_RUNNING,
false);
        }
    }
        public void ResumeFollowing()
    {
        enabled = true;
    }
        // Gizmos untuk visualisasi
di Scene view
    void OnDrawGizmosSelected()
    {
        // Jarak follow (kuning)
        Gizmos.color = Color.yellow;

Gizmos.DrawWireSphere(transform.posit
ion, followDistance);
        // Jarak run (merah)
        Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.posit
ion, runDistance);
                // Jarak stop (hijau)
        Gizmos.color = Color.green;

Gizmos.DrawWireSphere(transform.posit
ion, stopDistance);
                // Garis ke player
        if (player != null)
        {
            Gizmos.color = isRunning
? Color.red : (isMoving ?
Color.yellow : Color.green);

Gizmos.DrawLine(transform.position,
player.position);

            // Target position
            if (isMoving)
            {
                Gizmos.color =
Color.cyan;

Gizmos.DrawWireSphere(targetPosition,
0.5f);

Gizmos.DrawLine(transform.position,
targetPosition);
            }
        }
                //// Informasi status
        //if (Application.isPlaying)
        //{
        //
UnityEditor.Handles.Label(transform.p
osition + Vector3.up * 2f,
        //        $"Speed:
{actualSpeed:F1}\nState: {(isRunning
? "Running" : (isMoving ? "Walking" :
"Idle"))}");
        //}
```

```
    }
}
```

**PlayerHealth.cs**

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class PlayerHealth : MonoBehaviour
{
    public float maxHealth = 100f;
    private float currentHealth;

    public Slider healthSlider;
    public GameObject gameOverPanel; // Panel untuk game over
    public Image damageOverlay; // <<<<< TAMBAH INI
    public float overlayFadeSpeed = 2f; // Kecepatan fade out overlay

    private bool isDead = false;
    private Color overlayColor;
    private bool isTakingDamage = false;

    void Start()
    {
        currentHealth = maxHealth;
        UpdateHealthUI();

        if (gameOverPanel != null)
        {
            gameOverPanel.SetActive(false); // Panel disembunyikan di awal
        }

        if (damageOverlay != null)
        {
            overlayColor = damageOverlay.color;
            overlayColor.a = 0f; // Pastikan awalnya transparan
            damageOverlay.color = overlayColor;
        }
    }

    void Update()
    {
        // Smoothly fade out damage overlay
        if (damageOverlay != null && !isDead)
        {
            if (isTakingDamage)
            {
                overlayColor.a = 0.5f; // Ketika kena hit, alpha jadi 0.5 (merah)
                damageOverlay.color = overlayColor;
                isTakingDamage = false; // Reset flag
            }
            else
            {
                // Perlahan fade ke transparan
                overlayColor.a = Mathf.MoveTowards(overlayColor.a, 0f, overlayFadeSpeed * Time.deltaTime);
                damageOverlay.color = overlayColor;
            }
        }
    }

    public void TakeDamage(float amount)
    {
        if (isDead) return;
```

```
        currentHealth -= amount;
        currentHealth =
Mathf.Clamp(currentHealth, 0f,
maxHealth);
        UpdateHealthUI();

        // Efek kena hit
        isTakingDamage = true;

        if (currentHealth <= 0)
        {
            Die();
        }
    }
    public void Heal(float amount)
    {
        if (isDead) return;

        currentHealth += amount;
        currentHealth =
Mathf.Clamp(currentHealth, 0f,
maxHealth);
        UpdateHealthUI();
    }
    void UpdateHealthUI()
    {
        if (healthSlider != null)
        {
            healthSlider.value =
currentHealth / maxHealth;
        }
    }
    void Die()
    {
        isDead = true;

        Debug.Log("Player mati! Game
Over.");
        if (gameOverPanel != null)
        {

gameOverPanel.SetActive(true); //
Menampilkan panel game over
        }

        // Hapus bagian ini untuk
menghindari pindah ke scene Game Over
        //
Invoke("GoToGameOverScene", 2f);
    }

    // Hapus atau biarkan kosong
karena tidak ada transisi ke scene
    // void GoToGameOverScene()
    // {
    //
SceneManager.LoadScene("Gameover");
    // }
}
```

**SceneTransitionManager.cs**

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;
using System.Collections;

public class SceneTransitionManager :
MonoBehaviour
{
    [Header("Loading Screen
Components")]
    public GameObject loadingScreen;
    public Slider progressBar;
    public TextMeshProUGUI
loadingText;

    [Header("Loading Settings")]
```

```
    public float minimumLoadTime =
1f;


    public static
SceneTransitionManager Instance;


    void Awake()

    {

        // Singleton pattern

        if (Instance == null)

        {

            Instance = this;


DontDestroyOnLoad(gameObject);

        }

        else

        {

            Destroy(gameObject);

        }

    }


    void Start()

    {

        // Hide loading screen at
start

        if (loadingScreen != null)


loadingScreen.SetActive(false);

    }


    public void LoadScene(string
sceneName)

    {


StartCoroutine(LoadSceneAsync(sceneNa
me));

    }


    public void LoadScene(int
sceneIndex)

    {


StartCoroutine(LoadSceneAsync(sceneIn
dex));

    }


    IEnumerator LoadSceneAsync(string
sceneName)

    {

        // Show loading screen

        if (loadingScreen != null)


loadingScreen.SetActive(true);


        // Start loading the scene

        AsyncOperation operation =
SceneManager.LoadSceneAsync(sceneName
);


operation.allowSceneActivation =
false;


        float timer = 0f;


        while (!operation.isDone)

        {

            timer +=
Time.unscaledDeltaTime;


            // Calculate progress
(0.9f is max for loading, last 0.1f
is activation)

            float progress =
Mathf.Clamp01(operation.progress /
0.9f);


            // Update progress bar

            if (progressBar != null)

                progressBar.value =
progress;


            // Update loading text

            if (loadingText != null)
```

```
                loadingText.text =
"Loading... " + (progress *
100f).ToString("F0") + "%";


            // Check if loading is
complete and minimum time has passed

            if (operation.progress >=
0.9f && timer >= minimumLoadTime)

            {

                // Complete the
progress bar

                if (progressBar !=
null)

                    progressBar.value
= 1f;

                if (loadingText !=
null)

                    loadingText.text
= "Loading... 100%";


                yield return new
WaitForSecondsRealtime(0.3f);


                // Activate the scene

operation.allowSceneActivation =
true;

            }


            yield return null;

        }


        // Hide loading screen

        if (loadingScreen != null)

loadingScreen.SetActive(false);

    }


    IEnumerator LoadSceneAsync(int
sceneIndex)

    {

        // Show loading screen

        if (loadingScreen != null)

loadingScreen.SetActive(true);


        // Start loading the scene

        AsyncOperation operation =
SceneManager.LoadSceneAsync(sceneInde
x);

operation.allowSceneActivation =
false;

               float timer = 0f;

                while
(!operation.isDone)

        {

            timer +=
Time.unscaledDeltaTime;


            // Calculate progress

            float progress =
Mathf.Clamp01(operation.progress /
0.9f);


            // Update UI

            if (progressBar != null)

                progressBar.value =
progress;


            if (loadingText != null)

                loadingText.text =
"Loading... " + (progress *
100f).ToString("F0") + "%";


            // Check completion

            if (operation.progress >=
0.9f && timer >= minimumLoadTime)

            {

                if (progressBar !=
null)

                    progressBar.value
= 1f;

                if (loadingText !=
null)

                    loadingText.text
= "Loading... 100%";
```

```
                yield return new
WaitForSecondsRealtime(0.3f);

operation.allowSceneActivation =
true;
            }

            yield return null;
        }
              // Hide loading screen
        if (loadingScreen != null)

loadingScreen.SetActive(false);
    }
}
```

**GameManager.cs**

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager :
MonoBehaviour
{
    // Fungsi ini bisa dipanggil jika
ingin berpindah ke scene Game Over
    public void GameOver()
    {

SceneManager.LoadScene("gameover");
    }

}
```

**NPCDialogueTrigger.cs**

```
using UnityEngine;
using TMPro; // Pastikan untuk
mengimpor namespace TextMeshPro
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System.Collections;

public class NPCDialogueTrigger :
MonoBehaviour
{
    public GameObject dialoguePanel;
// Panel untuk dialog
    public TextMeshProUGUI
dialogText; // Referensi ke UI
TextMeshProUGUI untuk menampilkan
dialog
    public Button continueButton; //
Tombol "Continue"
    public Button cancelButton; //
Tombol "Batal"
    public string[] dialogues; //
Array untuk menyimpan dialog
    public float timeBetweenDialogues
= 2f; // Waktu antar dialog
    public string nextScene = "Misi
2"; // Nama scene untuk misi
berikutnya

    private int currentDialogueIndex
= 0; // Menyimpan index dialog saat
ini
    private bool isInTrigger = false;
// Untuk memeriksa apakah player
berada di dalam trigger

    void Start()
    {
        // Menyembunyikan panel,
dialog, dan tombol di awal

dialoguePanel.SetActive(false); //
Panel di-set tidak aktif di awal

continueButton.gameObject.SetActive(f
alse); // Tombol Continue tidak aktif

cancelButton.gameObject.SetActive(fal
se); // Tombol Batal tidak aktif

        // Debugging untuk memastikan
semua referensi ada dan berfungsi
        Debug.Log("Dialogue Panel
active: " +
dialoguePanel.activeSelf);
```

```
        Debug.Log("Continue Button active: " + continueButton.gameObject.activeSelf);

        Debug.Log("Cancel Button active: " + cancelButton.gameObject.activeSelf);


        // Menambahkan event listener ke tombol

continueButton.onClick.AddListener(OnContinueClicked);

cancelButton.onClick.AddListener(OnCancelClicked);

    }


    void OnTriggerEnter(Collider other)

    {

        // Memeriksa apakah yang masuk adalah player

        if (other.CompareTag("Player"))

        {

            isInTrigger = true;

            Debug.Log("Player detected in trigger!");


            // Aktifkan panel dialog dan tampilkan pesan pertama

dialoguePanel.SetActive(true);

            ShowDialog("Halo, kamu masuk ke area saya!");


            // Aktifkan tombol Continue dan Batal

continueButton.gameObject.SetActive(true);

cancelButton.gameObject.SetActive(true);

            // Menampilkan kursor saat interaksi dimulai

            //Cursor.visible = true;

            //Cursor.lockState = CursorLockMode.None; // Membebaskan kursor sehingga bisa bergerak bebas

        }

        else

        {

            Debug.Log("Other object entered trigger: " + other.gameObject.name);

        }

    }


    void OnTriggerExit(Collider other)

    {

        // Menyembunyikan dialog saat player keluar dari trigger

        if (other.CompareTag("Player"))

        {

            isInTrigger = false;

            HideDialog();

            // Kembalikan kursor ke kondisi semula saat keluar dari trigger

            //Cursor.visible = false;

            //Cursor.lockState = CursorLockMode.Locked; // Mengunci kursor agar tidak terlihat

        }

    }

    void ShowDialog(string dialogue)

    {

        // Menampilkan dialog dan UI

dialogText.gameObject.SetActive(true);

        dialogText.text = dialogue;
```

```
        // Debugging untuk memastikan dialog ditampilkan
        Debug.Log("Dialog shown: " + dialogue);
    }

    void HideDialog()
    {
        // Debugging untuk memastikan HideDialog dipanggil
        Debug.Log("Hiding dialog");

        // Menyembunyikan dialog dan tombol
        dialogText.gameObject.SetActive(false);
        continueButton.gameObject.SetActive(false);
        cancelButton.gameObject.SetActive(false);
        dialoguePanel.SetActive(false); // Menyembunyikan panel dialog
    }

    void OnContinueClicked()
    {
        if (currentDialogueIndex < dialogues.Length)
        {
            // Pastikan kita menyembunyikan dialog sebelumnya terlebih dahulu sebelum menampilkan yang baru
            HideDialog();

            // Tampilkan dialog berikutnya
            StartCoroutine(DisplayDialogue());
        }
        else
        {
            // Semua dialog selesai, load scene berikutnya
            SceneManager.LoadScene(nextScene);
        }
    }

    void OnCancelClicked()
    {
        // Menyembunyikan dialog dan tombol jika Batal
        HideDialog();
    }

    IEnumerator DisplayDialogue()
    {
        // Pastikan kita menunggu sedikit waktu setelah menyembunyikan dialog sebelumnya
        yield return new WaitForSeconds(0.1f);

        // Tampilkan dialog dengan delay antar dialog
        ShowDialog(dialogues[currentDialogueIndex]);
        currentDialogueIndex++;
        yield return new WaitForSeconds(timeBetweenDialogues);

        if (currentDialogueIndex < dialogues.Length)
        {
            StartCoroutine(DisplayDialogue());
        }
        else
        {
            // Jika selesai, lanjutkan ke scene misi
            // Pastikan posisi player disimpan sebelum pindah scene
```

```
            GameObject player =
GameObject.FindGameObjectWithTag("Pla
yer");

            if (player != null)

            {

                // Correct way to
access the singleton instance


GameManagerPos.Instance.SavePlayerPos
ition(player.transform.position);

            }


SceneManager.LoadScene(nextScene);

        }

    }

}
```

**PauseMenu.cs**

```
using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.EventSystems;

using UnityEngine.UI;



public class PauseMenu :
MonoBehaviour

{

    public GameObject pauseMenuUI;

    public GameObject optionsMenuUI;
// Panel untuk pengaturan audio

    public Slider audioSlider; //
Slider untuk mengatur volume audio



    public static bool isPaused =
false;



    void Start()

    {

        // Mengatur volume audio
berdasarkan nilai slider saat
permainan dimulai

        audioSlider.value =
AudioListener.volume;  // Set slider
sesuai dengan volume saat ini

    }



    void Update()

    {

        if (EventSystem.current !=
null &&
EventSystem.current.IsPointerOverGame
Object())

            return;



        if
(Input.GetKeyDown(KeyCode.Escape))

        {

            if (isPaused)

                Resume();

            else

                Pause();

        }

    }



    public void Resume()

    {

        pauseMenuUI.SetActive(false);


optionsMenuUI.SetActive(false); //
Menyembunyikan menu opsi saat
melanjutkan permainan

        Time.timeScale = 1f;

        isPaused = false;



        //Cursor.visible = false;

        //Cursor.lockState =
CursorLockMode.Locked;

    }



    public void Pause()

    {

        pauseMenuUI.SetActive(true);

        Time.timeScale = 0f;

        isPaused = true;
```

```
        //Cursor.visible = true;

        //Cursor.lockState =
CursorLockMode.None;

    }


    public void KeluarGame()

    {

        Time.timeScale = 1f; //
Penting! Kembalikan waktu ke normal
sebelum load scene baru

        isPaused = false;


SceneManager.LoadScene("MainMenu");
// Ganti dengan nama scene menu kamu

    }


    // Fungsi ini dipanggil saat
tombol "Restart" ditekan

    public void RestartGame()

    {

        Time.timeScale = 1f; //
Pastikan waktu berjalan normal
sebelum restart

        isPaused = false;


SceneManager.LoadScene(SceneManager.G
etActiveScene().name); // Muat ulang
scene saat ini

    }


    // Fungsi untuk membuka menu
pengaturan audio

    public void OpenOptionsMenu()

    {


optionsMenuUI.SetActive(true);

        pauseMenuUI.SetActive(false);
// Menyembunyikan menu pause saat
membuka opsi

    }


    // Fungsi untuk menutup menu
pengaturan audio

    public void CloseOptionsMenu()

    {


optionsMenuUI.SetActive(false);

        pauseMenuUI.SetActive(true);
// Menampilkan kembali menu pause

    }

}
```

**VirtualButtonControlle.cs**

```
using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;


public class VirtualButtonController
: MonoBehaviour

{

    [Header("Virtual Buttons")]

    public Button attackButton;

    public Button jumpButton;

    public GameObject runButton; //
gunakan EventTrigger (bukan Button)


    [Header("Input States")]

    public static bool isAttacking =
false;

    public static bool isJumping =
false;

    public static bool isRunning =
false;


    void Start()

    {

        // Assign click listeners


attackButton.onClick.AddListener(OnAt
tackPressed);


jumpButton.onClick.AddListener(OnJump
Pressed);


        // Setup EventTrigger for
hold-based input (Run)

        AddRunButtonTriggers();
```

```
    }

    void OnAttackPressed()
    {
        isAttacking = true;
        Debug.Log("Attack triggered!");
        // Reset immediately if needed
        Invoke(nameof(ResetAttack), 0.1f);
    }
    void OnJumpPressed()
    {
        isJumping = true;
        Debug.Log("Jump triggered!");
        Invoke(nameof(ResetJump), 0.1f);
    }

    void AddRunButtonTriggers()
    {
        EventTrigger trigger = runButton.GetComponent<EventTrigger>();
        if (trigger == null)
            trigger = runButton.AddComponent<EventTrigger>();

        // When Run button is pressed down
        EventTrigger.Entry entryDown = new EventTrigger.Entry
        {
            eventID = EventTriggerType.PointerDown
        };
entryDown.callback.AddListener((eventData) => { isRunning = true; Debug.Log("Running..."); });

        // When Run button is released
        EventTrigger.Entry entryUp = new EventTrigger.Entry
        {
            eventID = EventTriggerType.PointerUp
        };
entryUp.callback.AddListener((eventData) => { isRunning = false; Debug.Log("Stopped running"); });

trigger.triggers.Add(entryDown);
trigger.triggers.Add(entryUp);
    }

    void ResetAttack() => isAttacking = false;
    void ResetJump() => isJumping = false;
}
```

**ChickenUserControlle.cs**

```
using UnityEngine;

public class ChickenUserController : MonoBehaviour
{
    public ChickenCharacter chickenCharacter;
    public float upDownInputSpeed = 3f;

    public FixedJoystick movementJoystick; // Tambahkan referensi joystick

    private float attackCooldownTimer = 0f; // Timer for attack cooldown
    private float attackCooldown = 1f;
```

```
    void Start()
    {
        chickenCharacter =
GetComponent<ChickenCharacter>();
    }

    void Update()
    {
        // Update cooldown timer
        if (attackCooldownTimer > 0f)
        {
            attackCooldownTimer -=
Time.deltaTime;
        }

        // Jump / Soar dari tombol
virtual
        if
(VirtualButtonController.isJumping)
        {
            chickenCharacter.Soar();

VirtualButtonController.isJumping =
false; // reset agar tidak lompat
terus
        }

        // Attack dari tombol virtual
+ cooldown
        if
(VirtualButtonController.isAttacking
&& attackCooldownTimer <= 0f)
        {

chickenCharacter.Attack();
            attackCooldownTimer =
attackCooldown;

VirtualButtonController.isAttacking =
false;
        }

        // Run (menahan tombol run)
        if
(VirtualButtonController.isRunning)
        {

chickenCharacter.Gallop(); // atau
chickenCharacter.Run() jika berbeda
        }
        else
        {
            chickenCharacter.Walk();
        }

        // Catatan: fungsi lain
seperti Hit, Groom, Eat di-
nonaktifkan untuk 3 tombol versi ini
    }

    void FixedUpdate()
    {
        // Ambil arah input dari
joystick
        Vector2 inputDir = new
Vector2(movementJoystick.Horizontal,
movementJoystick.Vertical);

        // Jika ada input
        if (inputDir.magnitude >
0.1f)
        {
            // Hitung arah gerak
relatif ke kamera
            Vector3 cameraForward =
Camera.main.transform.forward;
            Vector3 cameraRight =
Camera.main.transform.right;

            cameraForward.y = 0f;
            cameraRight.y = 0f;

cameraForward.Normalize();
            cameraRight.Normalize();
```

```
            Vector3 moveDirection =
(cameraForward * inputDir.y +
cameraRight * inputDir.x).normalized;


            // Arah gerakan karakter
(world space)

chickenCharacter.moveDirection =
moveDirection;


            // Hitung yaw (rotasi
horizontal ke arah gerak)

chickenCharacter.yawVelocity =
Vector3.SignedAngle(transform.forward
, moveDirection, Vector3.up);


            // Set
forwardAcceleration berdasarkan input
vertikal joystick

chickenCharacter.forwardAcceleration
= inputDir.magnitude;
        }
        else
        {

chickenCharacter.forwardAcceleration
= 0f;

chickenCharacter.yawVelocity = 0f;
        }
    }


}
```

**ChickenCharacter.cs**

```
using UnityEngine;
using System.Collections;


public class ChickenCharacter :
MonoBehaviour {
    public Animator chickenAnimator;
    public float chickenSpeed = 1f;
    Rigidbody chickenRigid;
    public bool isFlying = false;
    public float upDown = 0f;
    public float forwardAcceleration
= 0f;
    public float yawVelocity = 0f;
    public float groundCheckDistance
= 5f;
    public bool isGrounded = true;
    public float forwardSpeed = 0f;
    public float maxForwardSpeed =
3f;
    public float meanForwardSpeed =
1.5f;
    public float speedDumpingTime =
.1f;
    public float groundCheckOffset =
0.1f;
    float soaringTime = 0f;
    public bool isLived = true;
    float walkMode = 1f;


    public float damage = 10f;
    public float attackRadius = 2f;
    void Start()
    {
        chickenAnimator =
GetComponent<Animator>();
        chickenAnimator.speed =
chickenSpeed;
        chickenRigid =
GetComponent<Rigidbody>();
    }


    void Update()
    {
        Move();
        soaringTime = soaringTime +
Time.deltaTime;
        GroundedCheck();
    }
```

```
    void GroundedCheck()
    {
        RaycastHit hit;
        if
(Physics.Raycast(transform.position +
Vector3.up * groundCheckOffset,
Vector3.down, out hit,
groundCheckDistance))
        {
            if (!isFlying ||
(isFlying && soaringTime > 1f))
            {
                Landing();
                isGrounded = true;
            }
        }
        else
        {
            isGrounded = false;

chickenAnimator.SetBool("IsGrounded",
false);
        }
    }

    public void SpeedSet(float
animSpeed)
    {
        chickenAnimator.speed =
animSpeed;
    }

    public void Landing()
    {

chickenAnimator.SetBool("IsGrounded",
true);

chickenAnimator.SetBool("IsFlying",
false);

chickenAnimator.applyRootMotion =
true;
        chickenRigid.useGravity =
true;
        isFlying = false;
    }

    public void Soar()
    {
        if (isGrounded && isLived)
        {
            soaringTime = 0f;

chickenAnimator.SetBool("IsGrounded",
false);

chickenAnimator.SetBool("IsFlying",
true);

chickenAnimator.SetTrigger("Soar");
            chickenRigid.useGravity =
false;
            isGrounded = false;
            forwardAcceleration = 0f;
            forwardSpeed = 0f;
            upDown = 0f;

chickenAnimator.applyRootMotion =
false;
            isFlying = true;
        }
    }

    public void Attack()
    {
        // Trigger attack animation

chickenAnimator.SetTrigger("Attack");

        // Perform overlap sphere to
detect enemies
```

```
        Collider[] hitEnemies =
Physics.OverlapSphere(transform.posit
ion, attackRadius);


        // Loop through the detected
colliders
        foreach (Collider enemy in
hitEnemies)
        {
            // Check if the detected
object has the tag "Musuh"
            if
(enemy.CompareTag("Musuh"))
            {
                // Get the
EnemyHealth component and apply
damage
                EnemyHealth
enemyHealth =
enemy.GetComponent<EnemyHealth>();
                if (enemyHealth !=
null)
                {

enemyHealth.TakeDamage(damage);
                    Debug.Log($"Enemy
hit! Damage dealt: {damage}");
                }
            }
        }
    }


    public void Hit()
    {

chickenAnimator.SetTrigger("Hit");
    }



    public void Death()
    {

chickenAnimator.SetBool("IsLived",
false);
        isLived = false;
    }


    public void Rebirth()
    {

chickenAnimator.SetBool("IsLived",
true);
        isLived = true;
    }


    public void Eat()
    {

chickenAnimator.SetTrigger("Eat");
    }


    public void Grooming()
    {

chickenAnimator.SetTrigger("Grooming"
);
    }


    public void Crowing()
    {

chickenAnimator.SetTrigger("Crowing")
;
    }


    public void Walk()
    {
        walkMode=1f;
    }


    public void Gallop()
    {
        walkMode=2f;
    }
```

```
    public Vector3 moveDirection; // Diisi oleh UserController

    public void Move()
    {

chickenAnimator.SetFloat("Turn", yawVelocity);

chickenAnimator.SetFloat("UpDown", upDown);

chickenAnimator.SetFloat("UpVelocity", chickenRigid.linearVelocity.y);

        if (isFlying)
        {

chickenAnimator.SetFloat("Forward", forwardAcceleration);

            if (soaringTime < 1f)
            {
                forwardSpeed = soaringTime * meanForwardSpeed;
                upDown = soaringTime;
            }

            Vector3 moveVec;

            if (forwardAcceleration < 0f)
            {
                moveVec = transform.up * upDown + moveDirection * forwardSpeed;
            }
            else
            {
                moveVec = transform.up * (upDown + (forwardSpeed - meanForwardSpeed)) + moveDirection * forwardSpeed;
            }

chickenRigid.linearVelocity = moveVec;

            transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(moveDirection), Time.deltaTime * 5f);

            forwardSpeed = Mathf.Lerp(forwardSpeed, 0f, Time.deltaTime * speedDumpingTime);

            forwardSpeed = Mathf.Clamp(forwardSpeed + forwardAcceleration * Time.deltaTime, 0f, maxForwardSpeed);

            upDown = Mathf.Lerp(upDown, 0, Time.deltaTime * 3f);

        }

        else
        {

chickenAnimator.SetFloat("Forward", forwardAcceleration * walkMode);

            // Jalan di tanah

            if (moveDirection != Vector3.zero)
            {
                transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(moveDirection), Time.deltaTime * 10f);

chickenRigid.linearVelocity = moveDirection * forwardAcceleration * walkMode;

            }

            else
            {

chickenRigid.linearVelocity = Vector3.zero;

            }

        }

    }
```

```
}
```

**ChickenAnimatorController1.cs**

```
using UnityEngine;


public class
ChickenAnimatorController1 :
MonoBehaviour

{

    [Header("References")]

    private Animator animator;

    private Rigidbody rb;


    [Header("Movement Settings")]

    public float walkSpeed = 2f;

    public float runSpeed = 6f;


    [Header("Combat Settings")]

    public float health = 100f;

    public float maxHealth = 100f;

    public float attackDuration = 1f;


    public float damage = 10f;


    [Header("Debug Info")]

    public float currentSpeed = 0f;

    public bool isAttacking = false;

    public bool isDead = false;


    void Start()

    {

        animator =
GetComponent<Animator>();

        rb =
GetComponent<Rigidbody>();


        // Initialize parameters

        animator.SetFloat("Speed",
0f);

animator.SetBool("IsAttacking",
false);

        animator.SetBool("IsDead",
false);

    }


    void Update()

    {

        if (isDead) return;


        HandleMovement();

        HandleCombat();

        HandleInput();


        // Debug display

        DisplayDebugInfo();

    }


    void HandleMovement()

    {

        // Get input

        float horizontal =
Input.GetAxis("Horizontal");

        float vertical =
Input.GetAxis("Vertical");

        Vector3 movement = new
Vector3(horizontal, 0, vertical);


        // Calculate speed

        currentSpeed =
movement.magnitude;


        if (currentSpeed > 0.1f)

        {

            // Determine movement
speed

            bool isRunning =
Input.GetKey(KeyCode.LeftShift);

            float moveSpeed =
isRunning ? runSpeed : walkSpeed;
```

```
            // Set animator speed
based on input
            if (isRunning)
            {

animator.SetFloat("Speed", 6f); //
Run speed for animator
            }
            else
            {

animator.SetFloat("Speed", 2f); //
Walk speed for animator
            }


            // Move character

transform.Translate(movement *
moveSpeed * Time.deltaTime,
Space.World);


            // Rotate to face
movement direction
            if (movement !=
Vector3.zero)
            {
                transform.rotation =
Quaternion.LookRotation(movement);
            }
        }
        else
        {
            // Idle

animator.SetFloat("Speed", 0f);
        }
    }


    void HandleCombat()
    {
        // Attack input
        if
(Input.GetKeyDown(KeyCode.Space) &&
!isAttacking)
        {
            PerformAttack();
        }
    }


    void HandleInput()
    {
        // Test damage (H key)
        if
(Input.GetKeyDown(KeyCode.H))
        {
            TakeDamage(25f);
        }


        // Test kill (K key)
        if
(Input.GetKeyDown(KeyCode.K))
        {
            TakeDamage(health);
        }
    }


    public void PerformAttack()
    {
        if (isDead || isAttacking)
return;


        isAttacking = true;

animator.SetBool("IsAttacking",
true);


        Debug.Log("Attack
performed!");


        // Reset attack after
duration
```

```
        Invoke("ResetAttack",
attackDuration);
    }

    void ResetAttack()
    {
        isAttacking = false;

animator.SetBool("IsAttacking",
false);
        Debug.Log("Attack
finished!");
    }

    public void TakeDamage(float
damage)
    {
        if (isDead) return;

        health = Mathf.Max(0, health
- damage);

        // Trigger hit animation

animator.SetTrigger("TakeDamage");

        Debug.Log($"Took {damage}
damage! Health: {health}");

        // Check if dead
        if (health <= 0)
        {
            Die();
        }
    }

    public void Die()
    {
        if (isDead) return;

        isDead = true;
        animator.SetBool("IsDead",
true);

        Debug.Log("Chicken died!");

        // Disable movement
        if (rb != null)
        {
            rb.isKinematic = true;
        }

        // Optional: Destroy after
animation
        Invoke("DestroyChicken", 3f);
    }

    void DestroyChicken()
    {
        Debug.Log("Chicken
destroyed!");
        Destroy(gameObject);
    }

    void DisplayDebugInfo()
    {
        // Update current speed for
debugging
        currentSpeed =
animator.GetFloat("Speed");
    }

    // Animation Events (called from
animation clips)
    public void OnAttackHit()
    {
        // Called during attack
animation at hit frame
        Debug.Log("Attack hit
frame!");
```

```
        // Add damage dealing logic
here

    }


    public void OnAttackComplete()

    {

        // Called at end of attack
animation

        ResetAttack();

    }


    public void OnHitComplete()

    {

        // Called at end of hit
animation

        Debug.Log("Hit animation
complete!");

    }


    public void OnDeathComplete()

    {

        // Called at end of death
animation

        Debug.Log("Death animation
complete!");

        // Transition to Exit state
will happen automatically

    }


    // Debug GUI

    void OnGUI()

    {

        if (!Application.isPlaying)
return;


        GUILayout.BeginArea(new
Rect(10, 10, 300, 200));

        GUILayout.Label("=== CHICKEN
ANIMATOR DEBUG ===");

        GUILayout.Label($"Current
State: {GetCurrentStateName()}");

        GUILayout.Label($"Speed
Parameter:
{animator.GetFloat("Speed"):F2}");

        GUILayout.Label($"Health:
{health:F0}/{maxHealth:F0}");


GUILayout.Label($"IsAttacking:
{animator.GetBool("IsAttacking")}");

        GUILayout.Label($"IsDead:
{animator.GetBool("IsDead")}");



        GUILayout.Space(10);

        GUILayout.Label("=== CONTROLS
===");

        GUILayout.Label("WASD: Move
(Hold Shift to Run)");

        GUILayout.Label("Space:
Attack");

        GUILayout.Label("H: Take
Damage");

        GUILayout.Label("K: Kill
(instant death)");



        GUILayout.Space(10);

        if (GUILayout.Button("Take 25
Damage"))

        {

            TakeDamage(25f);

        }

        if
(GUILayout.Button("Attack"))

        {

            PerformAttack();

        }

        if (GUILayout.Button("Kill"))

        {

            TakeDamage(health);

        }



        GUILayout.EndArea();

    }
```

```
    string GetCurrentStateName()
    {
        if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Idle"))
            return "Idle";
        else if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Walk"))
            return "Walk";
        else if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Run"))
            return "Run";
        else if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Attack"))
            return "Attack";
        else if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Hit"))
            return "Hit";
        else if
(animator.GetCurrentAnimatorStateInfo
(0).IsName("Death"))
            return "Death";
        else
            return "Unknown";
    }
}
```

**chickenEnemyAI.cs**

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;  // Make sure
you include this to work with UI
components

public class chickenEnemyAI :
EnemyHealth
{
    public string sceneSelanjutnya;
    public Slider healthSlider;  //
Reference to the health slider
    protected override void Start()
    {
        base.Start();
        Howl();

        // Initialize the slider
value when the script starts
        if (healthSlider != null)
        {
            healthSlider.maxValue =
health;
            healthSlider.value =
health; // Assuming `currentHealth`
is the current health of the enemy
        }
    }

    private void Howl()
    {
        animator.SetTrigger("Howl");
    }

    protected override void
PlayAttackAnimation()
    {
        int attackType =
Random.Range(0, 2);
        if (attackType == 0)

animator.SetTrigger("Attack");
        else

animator.SetTrigger("Hit");
    }

    public GameObject menangPanel;

    public override void Die()
    {
```

```
        base.Die();

StartCoroutine(WaitForDeathAnimationAndShowUI());
    }

    private IEnumerator WaitForDeathAnimationAndShowUI()
    {
        // Tunggu sampai animasi Die selesai
        AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
        while (!stateInfo.IsName("Death") || stateInfo.normalizedTime < 1f)
        {
            yield return null;
            stateInfo = animator.GetCurrentAnimatorStateInfo(0);
        }

        // Tampilkan UI
        if (menangPanel != null)

menangPanel.SetActive(true);

        // Tunggu 5 detik sebelum pindah scene
        yield return new WaitForSeconds(5f);

SceneManager.LoadScene(sceneSelanjutnya);
    }

    // Override this method to update the slider when health changes
    public override void TakeDamage(float amount)
    {
        base.TakeDamage(amount);

        // Update the health slider
        if (healthSlider != null)
        {
            healthSlider.value = health; // Assuming `currentHealth` is updated in `TakeDamage`
        }
    }
}
```

**NPCDialogue.cs**

```
using UnityEngine;
using TMPro; // Pastikan untuk mengimpor namespace TextMeshPro
using UnityEngine.UI;
using System.Collections;

public class NPCDialogue : MonoBehaviour
{
    public GameObject dialoguePanel; // Panel untuk dialog
    public TextMeshProUGUI dialogText; // Referensi ke UI TextMeshProUGUI untuk menampilkan dialog
    public Button continueButton; // Tombol "Lanjutkan"
    public Button cancelButton; // Tombol "Batal"
    public string[] dialogues; // Array untuk menyimpan dialog
    public float timeBetweenDialogues = 2f; // Waktu antar dialog

    private int currentDialogueIndex = 0; // Menyimpan index dialog saat ini
    private bool isInTrigger = false; // Untuk memeriksa apakah player berada di dalam trigger

    void Start()
```

```
    {
        // Menyembunyikan panel, dialog, dan tombol di awal

dialoguePanel.SetActive(false); // Panel di-set tidak aktif di awal

continueButton.gameObject.SetActive(false); // Tombol Continue tidak aktif

cancelButton.gameObject.SetActive(false); // Tombol Batal tidak aktif

        // Menambahkan event listener ke tombol

continueButton.onClick.AddListener(OnContinueClicked);

cancelButton.onClick.AddListener(OnCancelClicked);

        // Menyembunyikan kursor di awal (opsional, jika kamu ingin kursor tidak tampil saat bergerak bebas)
        //Cursor.visible = false;
        //Cursor.lockState = CursorLockMode.Locked;
    }

    void OnTriggerEnter(Collider other)
    {
        // Memeriksa apakah yang masuk adalah player
        if (other.CompareTag("Player"))
        {
            isInTrigger = true;
            Debug.Log("Player detected in trigger!");

            // Aktifkan panel dialog dan tampilkan pesan pertama
dialoguePanel.SetActive(true);
            ShowDialog("Halo, kamu masuk ke area saya!");

            // Aktifkan tombol Continue dan Batal
continueButton.gameObject.SetActive(true);
cancelButton.gameObject.SetActive(true);

            // Menampilkan kursor saat interaksi dimulai
            //Cursor.visible = true;
            //Cursor.lockState = CursorLockMode.None; // Membebaskan kursor sehingga bisa bergerak bebas
        }
    }

    void OnTriggerExit(Collider other)
    {
        // Menyembunyikan dialog saat player keluar dari trigger
        if (other.CompareTag("Player"))
        {
            isInTrigger = false;
            HideDialog();

            // Kembalikan kursor ke kondisi semula saat keluar dari trigger
            //Cursor.visible = false;
            //Cursor.lockState = CursorLockMode.Locked; // Mengunci kursor agar tidak terlihat
        }
    }
```

```
    void ShowDialog(string dialogue)
    {
        // Menampilkan dialog dan UI
dialogText.gameObject.SetActive(true)
;
        dialogText.text = dialogue;
    }

    void HideDialog()
    {
        // Menyembunyikan dialog dan
tombol
dialogText.gameObject.SetActive(false
);
continueButton.gameObject.SetActive(f
alse);
cancelButton.gameObject.SetActive(fal
se);
dialoguePanel.SetActive(false); //
Menyembunyikan panel dialog
    }

    void OnContinueClicked()
    {
        if (currentDialogueIndex <
dialogues.Length)
        {
            // Tampilkan dialog
berikutnya
StartCoroutine(DisplayDialogue());
        }
        else
        {
            // Jika sudah selesai,
sembunyikan dialog dan tombol
            HideDialog();
        }
    }

    void OnCancelClicked()
    {
        // Menyembunyikan dialog dan
tombol jika Batal
        HideDialog();
    }

    IEnumerator DisplayDialogue()
    {
        // Tampilkan dialog dengan
delay antar dialog
ShowDialog(dialogues[currentDialogueI
ndex]);
        currentDialogueIndex++;
        yield return new
WaitForSeconds(timeBetweenDialogues);

        if (currentDialogueIndex <
dialogues.Length)
        {
StartCoroutine(DisplayDialogue());
        }
        else
        {
            // Dialog selesai,
sembunyikan panel dan tombol
            HideDialog();
        }
    }
}
```

**Playermisi.cs**

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;
```

```
public class Playermisi :
MonoBehaviour

{

    public GameObject dialogUI;

    public TextMeshProUGUI
dialogText;

    public Button continueButton;

    public Button cancelButton;



    private enum MissionState {
BelumMulai, DialogNPC, PergiKeRaja,
Selesai }

    private MissionState state =
MissionState.BelumMulai;



    private Coroutine
autoHideCoroutine;



    private void Start()

    {

        dialogUI.SetActive(false);


continueButton.onClick.AddListener(On
ContinueClicked);


cancelButton.onClick.AddListener(OnCa
ncelClicked);

    }



    private void
OnTriggerEnter(Collider other)

    {

        // Cek jika NPC pertama kali
dipicu untuk dialog

        if (other.CompareTag("NPC1"))

        {

            if (state ==
MissionState.BelumMulai)

            {

                state =
MissionState.DialogNPC;


StartCoroutine(ShowDialogSequence(new
string[]

                {

                    "hahahah ternyata
ayam mu jago sampai sampai bisa
mengalahkan semua ayam di sini ayam
mu ini sangat unik.",

                    "info dari
penjaga Katanya Kamu sedang di Cari
oleh raja !.",

                    "Apa mungkin dia
ingin mengalahkan ayam mu dan
mengambil nya?",

                    "Temuilah Raja
Untuk mencaritau Kenapa dia Mencari
Mu",

                     "ikuti lurusan
air mancur dan cari lah rumah paling
besar di sana lah raja tinggal",

                    "Semoga
perjalananmu lancar!"

                }));

            }

        }

        // Setelah dialog selesai,
instruksi pergi ke Raja

        else if (state ==
MissionState.DialogNPC &&
other.CompareTag("Raja"))

        {

            state =
MissionState.PergiKeRaja;

            ShowDialog("Kamu sudah
sampai di Kerajaan. Temui Raja untuk
melanjutkan misi.", false, 5f);

            state =
MissionState.Selesai;

        }

    }



    private void ShowDialog(string
message, bool showButtons, float
autoHideTime = 0f)

    {

        dialogText.text = message;

        dialogUI.SetActive(true);


continueButton.gameObject.SetActive(s
howButtons);
```

```
cancelButton.gameObject.SetActive(sho
wButtons);


        // Auto-hide jika durasi
disetel (> 0)
        if (autoHideCoroutine !=
null)

StopCoroutine(autoHideCoroutine);


        if (autoHideTime > 0f)
            autoHideCoroutine =
StartCoroutine(HideAfterSeconds(autoH
ideTime));
    }


    private IEnumerator
HideAfterSeconds(float seconds)
    {
        yield return new
WaitForSeconds(seconds);
        HideDialog();
    }


    private void HideDialog()
    {
        dialogUI.SetActive(false);

continueButton.gameObject.SetActive(f
alse);

cancelButton.gameObject.SetActive(fal
se);
    }


    private void OnContinueClicked()
    {
        HideDialog();
    }


    private IEnumerator
ShowDialogSequence(string[] messages)
    {
        foreach (string message in
messages)
        {
            ShowDialog(message,
false, 3f); // Durasi 3 detik per
dialog
            yield return new
WaitForSeconds(3f); // Tunggu sampai
dialog selesai
        }


        // Setelah dialog selesai,
beri instruksi pergi ke Raja
        ShowDialog("Pergilah ke Raja
untuk melanjutkan misi.", true, 5f);
    }


    private void OnCancelClicked()
    {
        HideDialog();
    }
}
```

**Playermisi3.cs**

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;
using System.Collections;


public class Playermisi3 :
MonoBehaviour
{
    public GameObject dialogUI;
    public TextMeshProUGUI
dialogText;
    public Button buttonIya;
    public Button buttonTidak;


    private enum MissionState {
BelumMulai, DialogNPC,
```

```
KonfirmasiPergiKeRaja, PergiKeRaja,
Selesai }

    private MissionState state =
MissionState.BelumMulai;



    private Coroutine
autoHideCoroutine;



    private void Start()

    {

        dialogUI.SetActive(false);


buttonIya.onClick.AddListener(OnIyaCl
icked);


buttonTidak.onClick.AddListener(OnTid
akClicked);

    }



    private void
OnTriggerEnter(Collider other)

    {

        if (other.CompareTag("Raja")
&& state == MissionState.BelumMulai)

        {

            state =
MissionState.DialogNPC;


StartCoroutine(ShowDialogSequence(new
string[]

            {

                "Aku mengundang mu ke
sini ingin menantang ayam mu!",

                "kalau kamu bisa
mengalahkan ayam ku , separuh harta
ku akan jadi milik mu!",

                "dan kalau ayam mu
kalah saya akan mengambil ayam mu",

                "apakah anda
setuju?",

                "Semoga berhasil !!"

            }));

        }

        // Di akhir misi, ketika
masuk ke trigger Raja, langsung
munculkan konfirmasi

        else if
(other.CompareTag("Raja") && state ==
MissionState.DialogNPC)

        {

            state =
MissionState.KonfirmasiPergiKeRaja;

            ShowDialog("Apakah kamu
ingin melanjutkan misi dan menemui
Raja sekarang?", true);

        }

    }



    private void ShowDialog(string
message, bool
showConfirmationButtons, float
autoHideTime = 0f)

    {

        dialogText.text = message;

        dialogUI.SetActive(true);



        // Tombol ditampilkan jika
memang dialog konfirmasi


buttonIya.gameObject.SetActive(showCo
nfirmationButtons);


buttonTidak.gameObject.SetActive(show
ConfirmationButtons);



        // Jika autoHide diaktifkan
(untuk dialog biasa), sembunyikan
setelah beberapa detik

        if (autoHideCoroutine !=
null)


StopCoroutine(autoHideCoroutine);



        if (autoHideTime > 0f)

            autoHideCoroutine =
StartCoroutine(HideAfterSeconds(autoH
ideTime));

    }
```

```
    private IEnumerator
HideAfterSeconds(float seconds)
    {
        yield return new
WaitForSeconds(seconds);
        HideDialog();
    }

    private void HideDialog()
    {
        dialogUI.SetActive(false);

buttonIya.gameObject.SetActive(false)
;

buttonTidak.gameObject.SetActive(fals
e);
    }

    private void OnIyaClicked()
    {
        if (state ==
MissionState.KonfirmasiPergiKeRaja)
        {

SceneManager.LoadScene("misi3");
        }
    }

    private void OnTidakClicked()
    {
        HideDialog();
    }

    private IEnumerator
ShowDialogSequence(string[] messages)
    {
        foreach (string message in
messages)
        {
            ShowDialog(message,
false);
            yield return new
WaitForSeconds(3f);
        }

        // 🕹 DI SINI PERBAIKANNYA:
munculkan tombol IYA dan TIDAK
setelah semua dialog selesai
        state =
MissionState.KonfirmasiPergiKeRaja;
        ShowDialog("Apakah kamu ingin
melanjutkan misi3 dan mengalahkan
raja sekarang?", true);
    }
}
```

**BackToMainMenu.cs**

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI; // Pastikan ini
di-include untuk Button

public class BackToMainMenu :
MonoBehaviour
{
    // Fungsi ini akan dipanggil
ketika tombol diklik
    public void OnBackToMainMenu()
    {
        // Memuat scene dengan nama
"MainMenu"

SceneManager.LoadScene("MainMenu");
    }

    // Fungsi Start untuk menambahkan
listener ke tombol
    void Start()
    {
        // Mencari komponen Button
pada GameObject yang sama
        Button btn =
GetComponent<Button>();
```

```
        // Jika komponen Button
ditemukan, tambahkan listener
        if (btn != null)
        {

btn.onClick.AddListener(OnBackToMainM
enu);
        }
    }
}
```