Enemy Controller - Fardowsa

```
using UnityEngine;
using UnityEngine.AI;

// Creates the enemy controller
// This makes the enemy move such as running, chasing and staying idle

public class EnemyController : MonoBehaviour
{
    public Transform[] waypoints;
    public float idleTime = 2f;
    public float walkSpeed = 2f; // Walking speed.
    public float chaseSpeed = 4f; // Chasing speed.
    public float sightDistance = 10f;
    public float fieldOfView = 45f; // Field of view angle for detection.
    public AudioClip idleSound;
    public AudioClip walkingSound;
    public AudioClip chasingSound;

    private int currentWaypointIndex = 0;
    private NavMeshAgent agent;
    private Animator animator;
    private float idleTimer = 0f;
    private Transform player;
    private AudioSource audioSource;

    private enum EnemyState { Idle, Walk, Chase }
    private EnemyState currentState = EnemyState.Idle;

    private void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        animator = GetComponent<Animator>();
        player = GameObject.FindGameObjectWithTag("Player").transform;
        audioSource = GetComponent<AudioSource>();

        if (waypoints.Length == 0)
        {
            Debug.LogError("Waypoints not assigned to EnemyController.");
            return;
        }
```

```
    SetDestinationToWaypoint();
    StartCoroutine(PlayerDetectionCoroutine());
}

private void Update()
{
    switch (currentState)
    {
        case EnemyState.Idle:
            HandleIdleState();
            break;

        case EnemyState.Walk:
            HandleWalkState();
            break;

        case EnemyState.Chase:
            HandleChaseState();
            break;
    }
}

private void HandleIdleState()
{
    idleTimer += Time.deltaTime;
    animator.SetBool("IsWalking", false);
    animator.SetBool("IsChasing", false);
    PlaySound(idleSound);

    if (idleTimer >= idleTime)
    {
        NextWaypoint();
    }
}

private void HandleWalkState()
{
    idleTimer = 0f;
    animator.SetBool("IsWalking", true);
    animator.SetBool("IsChasing", false);
    PlaySound(walkingSound);
```

```
        if (agent.remainingDistance <= agent.stoppingDistance)
        {
            currentState = EnemyState.Idle;
        }
    }

    private void HandleChaseState()
    {
        agent.speed = chaseSpeed;
        agent.SetDestination(player.position);
        animator.SetBool("IsWalking", false);
        animator.SetBool("IsChasing", true);
        PlaySound(chasingSound);

        if (Vector3.Distance(transform.position, player.position) > sightDistance)
        {
            currentState = EnemyState.Walk;
            agent.speed = walkSpeed;
        }
    }

    private IEnumerator PlayerDetectionCoroutine()
    {
        while (true)
        {
            CheckForPlayerDetection();
            yield return new WaitForSeconds(0.2f); // Check for detection 5 times per second.
        }
    }

    private void CheckForPlayerDetection()
    {
        Vector3 playerDirection = player.position - transform.position;
        float angleToPlayer = Vector3.Angle(transform.forward, playerDirection);

        if (angleToPlayer < fieldOfView)
        {
            RaycastHit hit;
            if (Physics.Raycast(transform.position, playerDirection.normalized, out hit,
sightDistance))
            {
                if (hit.collider.CompareTag("Player"))
```

```
            {
                currentState = EnemyState.Chase;
                Debug.Log("Player detected!");
            }
        }
    }
}

private void NextWaypoint()
{
    currentWaypointIndex = (currentWaypointIndex + 1) % waypoints.Length;
    SetDestinationToWaypoint();
}

private void SetDestinationToWaypoint()
{
    if (waypoints.Length > 0)
    {
        agent.SetDestination(waypoints[currentWaypointIndex].position);
        currentState = EnemyState.Walk;
        agent.speed = walkSpeed;
    }
}

private void PlaySound(AudioClip soundClip)
{
    if (!audioSource.isPlaying || audioSource.clip != soundClip)
    {
        audioSource.clip = soundClip;
        audioSource.Play();
    }
}

private void OnDrawGizmos()
{
    if (player != null)
    {
        Gizmos.color = currentState == EnemyState.Chase ? Color.red : Color.green;
        Gizmos.DrawLine(transform.position, player.position);
    }
}
}
```

Enemy Health - Fardowsa

```
using UnityEngine;

public class EnemyHealth : MonoBehaviour
{
    public int maxHealth = 100; // Maximum health of the enemy.
    private int currentHealth; // Current health of the enemy. public
    GameObject deathEffect; // Optional: Particle effect for death. public
    AudioClip deathSound; // Optional: Sound for death. private
    AudioSource audioSource;
    private Animator animator;
    private bool isDead = false; // To prevent multiple death triggers.

    private void Start()
    {
        currentHealth = maxHealth; // Initialize health to the maximum value.
        animator = GetComponent<Animator>();
        audioSource = GetComponent<AudioSource>();
    }

    // Function to apply damage to the enemy.
    public void TakeDamage(int damageAmount)
    {
        if (isDead) return; // Prevent further damage if already dead.

        currentHealth -= damageAmount; // Reduce the health.

        // Optionally trigger a "hit" animation if it exists.
        animator?.SetTrigger("Hit");

        if (currentHealth <= 0)
        {
            Die(); // Handle death when health is depleted.
        }
    }

    // Function to handle the enemy's death.
    private void Die()
```

```
{
    if (isDead) return; // Prevent multiple death triggers.
    isDead = true;

    // Play death animation if it exists.
    if (animator != null)
    {
        animator.SetBool("Death", true);
    }

    // Play death sound if assigned.
    if (deathSound != null && audioSource != null)
    {
        audioSource.PlayOneShot(deathSound);
    }

    // Spawn death effect if assigned.
    if (deathEffect != null)
    {
        Instantiate(deathEffect, transform.position, Quaternion.identity);
    }

    // Disable enemy's functionality (e.g., movement, attacks).
    DisableEnemy();

    // Destroy the enemy GameObject after a delay (e.g., after death animation).
    Destroy(gameObject, 2f);
}

// Function to disable the enemy's functionality upon death.
private void DisableEnemy()
{
    // Disable any movement scripts or nav mesh agents.
    var navAgent = GetComponent<UnityEngine.AI.NavMeshAgent>();
    if (navAgent != null)
    {
        navAgent.enabled = false;
    }

    // Disable collider to prevent interactions.
    var collider = GetComponent<Collider>();
    if (collider != null)
    {
```

```
            collider.enabled = false;
        }

        // Stop other components like attacking or AI behaviors (if applicable).
    }
}
```

Menu - samira

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenuLogic : MonoBehaviour
{
    private Canvas mainMenuCanvas;
    private Canvas optionsMenuCanvas;
    private Canvas extrasMenuCanvas;
    private Canvas loadingCanvas;

    public AudioSource buttonSound;

    void Start()
    {
        mainMenuCanvas = GameObject.Find("MainMenuCanvas")?.GetComponent<Canvas>();
        optionsMenuCanvas = GameObject.Find("OptionsCanvas")?.GetComponent<Canvas>();
        extrasMenuCanvas = GameObject.Find("ExtrasCanvas")?.GetComponent<Canvas>();
        loadingCanvas = GameObject.Find("LoadingCanvas")?.GetComponent<Canvas>();

        if (mainMenuCanvas == null || optionsMenuCanvas == null || extrasMenuCanvas == null ||
loadingCanvas == null)
        {
            Debug.LogError("Missing canvas references. Ensure all menus are correctly named.");
            return;
        }

        mainMenuCanvas.enabled = true;
        optionsMenuCanvas.enabled = false;
        extrasMenuCanvas.enabled = false;
        loadingCanvas.enabled = false;
    }

    public void OnStartGame()
    {
```

```
        if (buttonSound) buttonSound.Play();

        loadingCanvas.enabled = true;
        mainMenuCanvas.enabled = false;

        SceneManager.LoadScene("SampleScene");
    }

    public void ShowOptionsMenu()
    {
        if (buttonSound) buttonSound.Play();

        mainMenuCanvas.enabled = false;
        optionsMenuCanvas.enabled = true;
    }

    public void ShowExtrasMenu()
    {
        if (buttonSound) buttonSound.Play();

        mainMenuCanvas.enabled = false;
    extrasMenuCanvas.enabled = true; }

    public void QuitGame()
    {
        if (buttonSound) buttonSound.Play();

        Debug.Log("Game Exited");
        Application.Quit();
    }

    public void GoBackToMainMenu()
    {
        if (buttonSound) buttonSound.Play();

        mainMenuCanvas.enabled = true;
        optionsMenuCanvas.enabled = false;
    extrasMenuCanvas.enabled = false; }
}
```

Enemy Damage - Samira

```
using UnityEngine;

public class EnemyDamage : MonoBehaviour
{
    public GameObject player;
    private float damageAmount;
    public float fixedDamage = 25f;
    public float minRandomDamage;
    public float maxRandomDamage;
    public bool useRandomDamage;
    public bool useFixedDamage;

    public AudioClip[] damageSounds;
    private AudioSource audioSource;

    void Start()
    {
        damageAmount = Random.Range(minRandomDamage, maxRandomDamage);
        audioSource = player.GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (useRandomDamage)
            {
                player.GetComponent<PlayerHealth>().health -= damageAmount;
                PlayRandomDamageSound();
            }

            if (useFixedDamage)
            {
                player.GetComponent<PlayerHealth>().health -= fixedDamage;
                PlayRandomDamageSound();
            }
```

```
        }
    }

    private void PlayRandomDamageSound()
    {
        if (damageSounds.Length > 0)
        {
            audioSource.clip = damageSounds[Random.Range(0, damageSounds.Length)];
            audioSource.Play();
        }
    }
}
```

Player Health - Samira

```
using UnityEngine;
using UnityStandardAssets.Characters.FirstPerson;

public class PlayerHealth : MonoBehaviour
{
    public GameObject hud;
    public GameObject inventory;
    public GameObject deathScreen;
    public GameObject player;

    public float health = 100f;

    void Start()
    {
        // Ensure the death screen is hidden at the start
        deathScreen.SetActive(false);
    }

    void Update()
    {
        // Check if the player's health has dropped to 0 or below
        if (health <= 0)
        {
            HandlePlayerDeath();
        }
```

```
        // Clamp health to a maximum of 100
        if (health > 100)
        {
            health = 100;
        }
    }

    private void HandlePlayerDeath()
    {
        // Disable player movement
        player.GetComponent<FirstPersonController>().enabled = false;
        // Show cursor and unlock it
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;

        // Hide HUD and inventory, show death screen
        hud.SetActive(false);
        inventory.SetActive(false);
        deathScreen.SetActive(true);
    }
}
```

Kill Player - Samira

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class KillPlayer : MonoBehaviour
{
    [Header("Scene Transition Settings")]
    public string nextSceneName; // The name of the next scene to load.
    public float delay = 0.5f; // Delay in seconds before loading the next scene.

    [Header("UI Settings")]
    public GameObject fadeout; // Fadeout UI element.

    private bool isPlayerInsideTrigger = false; // Track if the player is inside the trigger area.
```

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            isPlayerInsideTrigger = true; // Mark that the player is in the trigger zone.
            if (fadeout != null)
            {
                fadeout.SetActive(true); // Activate the fadeout UI if assigned.
            }
            Invoke(nameof(LoadNextScene), delay); // Schedule the scene load after the delay.
        }
    }

    private void LoadNextScene()
    {
        if (isPlayerInsideTrigger)
        {
            SceneManager.LoadScene(nextSceneName); // Load the specified scene.
        }
    }
}
```

Pistol - Sofiya

```
using UnityEngine;

public class Handgun : MonoBehaviour
{
    [Header("Ammo Settings")]
    public int magCapacity = 10; // Max bullets in a single magazine
    public int reserveCapacity = 30; // Max bullets in reserve
    public float fireCooldown = 0.5f; // Time between shots
    public float reloadTime = 0.5f; // Time taken to reload
    private float actionCooldown = 0.5f; // Cooldown between switching actions
    public float maxRange = 100f; // Effective shooting range

    [Header("Effects")]
    public ParticleSystem hitEffect; // Effect displayed on bullet impact
    public ParticleSystem muzzleEffect;
    public GameObject muzzleLight;

    [Header("Cartridge Settings")]
    public Transform ejectionPoint; // Where the cartridge is ejected
    public GameObject cartridgePrefab;
```

```
public float ejectionForce = 5f;

[Header("Gun Stats")]
public Animator gunAnimator;
public AudioSource fireSound;
public int damage = 10; // Damage dealt per shot
public bool isReadyToShoot = true;
private bool isReloading = false;

private int bulletsInMag;
private int bulletsInReserve;
private float fireCooldownTimer;

void Start()
{
    bulletsInMag = magCapacity;
    bulletsInReserve = reserveCapacity;
    isReadyToShoot = true;
    muzzleLight.SetActive(false);
}

void Update()
{
    // Ammo management
    bulletsInMag = Mathf.Clamp(bulletsInMag, 0, magCapacity);
    bulletsInReserve = Mathf.Clamp(bulletsInReserve, 0, reserveCapacity);

    // Shooting input
    if (Input.GetButtonDown("Fire1") && isReadyToShoot && !isReloading)
    {
        actionCooldown = fireCooldown;
        Fire();
    }

    // Reload input
    if (Input.GetKeyDown(KeyCode.R))
    {
        actionCooldown = reloadTime;
        ReloadWeapon();
    }

    // Update cooldown timer
    if (fireCooldownTimer > 0)
```

```
        {
            fireCooldownTimer -= Time.deltaTime;
        }
    }

    void Fire()
    {
        if (bulletsInMag > 0 && fireCooldownTimer <= 0)
        {
            isReadyToShoot = false;
            fireSound.Play();
            muzzleEffect.Play();
            muzzleLight.SetActive(true);
            gunAnimator.SetBool("shoot", true);

            // Handle shooting logic
            RaycastHit hit;
            if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward,
out hit, maxRange))
            {
                if (hit.collider.CompareTag("Enemy"))
                {
                    var enemyHealth = hit.collider.GetComponent<EnemyHealth>();
                    enemyHealth?.TakeDamage(damage);
                }
                Instantiate(hitEffect, hit.point, Quaternion.LookRotation(hit.normal));
            }

            // Eject cartridge
            GameObject cartridge = Instantiate(cartridgePrefab, ejectionPoint.position,
ejectionPoint.rotation);
            cartridge.GetComponent<Rigidbody>().AddForce(ejectionPoint.right * ejectionForce,
ForceMode.Impulse);

            StartCoroutine(ResetEffects());
            StartCoroutine(AllowAction());

            bulletsInMag--;
            fireCooldownTimer = fireCooldown;
        }
        else
        {
            Debug.Log("Out of ammo!");
```

```
        }
    }

    void ReloadWeapon()
    {
        if (isReloading || bulletsInReserve <= 0)
            return;

        int bulletsNeeded = magCapacity - bulletsInMag;
        int bulletsReloaded = Mathf.Min(bulletsNeeded, bulletsInReserve);

        bulletsInMag += bulletsReloaded;
        bulletsInReserve -= bulletsReloaded;

        gunAnimator.SetBool("reload", true);
        StartCoroutine(ResetEffects());
        StartCoroutine(ReloadCooldown());
    }
    IEnumerator ReloadCooldown()
    {
        isReloading = true;
        isReadyToShoot = false;

        yield return new WaitForSeconds(reloadTime);

        isReloading = false;
        isReadyToShoot = true;
    }

    IEnumerator ResetEffects()
    {
        yield return new WaitForSeconds(0.1f);
        gunAnimator.SetBool("shoot", false);
        gunAnimator.SetBool("reload", false);
        muzzleLight.SetActive(false);
    }

    IEnumerator AllowAction()
    {
        yield return new WaitForSeconds(fireCooldown);
        isReadyToShoot = true;
    }
}
```

Cursor Controller - Sofiya

```
using UnityEngine;

public class CursorManager : MonoBehaviour
{
    private void Awake()
    {
        // Ensure the cursor is visible and not restricted
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }
}
```

Death Main - Sofiya

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuController : MonoBehaviour
{
    // Method to load the game scene
    public void LoadGameScene()
    {
        SceneManager.LoadScene("Game");
    }

    // Method to exit the application
    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Use chest - Fadumo

```
using UnityEngine;

public class UseChest : MonoBehaviour
{
    [Header("Chest Interaction Settings")]
    public GameObject handUI; // UI prompt to display when in reach.
    public GameObject objToActivate; // Object to activate when the chest is opened.
```

```
private bool isPlayerInReach = false; // Tracks if the player is within interaction range.
private Animator chestAnimator; // Reference to the chest's Animator component.
private BoxCollider chestCollider; // Reference to the chest's BoxCollider component.

private void Start()
{
    // Initialize components and set initial states.
    handUI.SetActive(false); // Ensure the hand UI is hidden initially.
    objToActivate.SetActive(false); // Ensure the object to activate is disabled initially.

    chestAnimator = GetComponent<Animator>();
    chestCollider = GetComponent<BoxCollider>();
}

private void OnTriggerEnter(Collider other)
{
    // Detect if the player enters the chest's trigger zone.
    if (other.CompareTag("Reach"))
    {
        isPlayerInReach = true;
        handUI.SetActive(true); // Display the hand UI prompt.
    }
}

private void OnTriggerExit(Collider other)
{
    // Detect if the player exits the chest's trigger zone.
    if (other.CompareTag("Reach"))
    {
        isPlayerInReach = false;
        handUI.SetActive(false); // Hide the hand UI prompt.
    }
}

private void Update()
{
    // Check if the player is in reach and presses the "Interact" button.
    if (isPlayerInReach && Input.GetButtonDown("Interact")) {
        OpenChest();
    }
}
```

```
    private void OpenChest()
    {
        // Hide the hand UI and activate the associated object.
        handUI.SetActive(false);
        objToActivate.SetActive(true);

        // Trigger the chest's open animation and disable its collider.
        if (chestAnimator != null)
        {
            chestAnimator.SetBool("open", true);
        }

        if (chestCollider != null)
        {
            chestCollider.enabled = false;
        }
    }
}
```

Key Pickup - Fadumo

```
using UnityEngine;

public class KeyPickupHandler : MonoBehaviour
{
    public GameObject interactionPrompt; // UI to indicate the player can pick up the key public
    GameObject keyInventorySlot; // Object representing the key in the player's inventory

    private GameObject keyObject; // Reference to the key object
    private bool isWithinRange = false; // Tracks if the player is close enough to interact

    void Start()
    {
        // Disable UI and inventory slot display initially
        interactionPrompt.SetActive(false);
        keyInventorySlot.SetActive(false);

        // Assign the key object
        keyObject = this.gameObject;
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Reach"))
```

```
        {
            isWithinRange = true;
            interactionPrompt.SetActive(true); // Show prompt when player is nearby
        }
    }

    void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Reach"))
        {
            isWithinRange = false;
            interactionPrompt.SetActive(false); // Hide prompt when player moves away
        }
    }

    void Update()
    {
        // Handle key pickup interaction
        if (isWithinRange && Input.GetButtonDown("Interact"))
        {
            interactionPrompt.SetActive(false); // Hide the interaction prompt
            keyInventorySlot.SetActive(true); // Display the key in the inventory
            keyObject.GetComponent<MeshRenderer>().enabled = false; // Hide the key object
        }
    }
}
```

Main Menu - Sofiya

```
using UnityEngine;

public class LanternPickup : MonoBehaviour
{
    private GameObject currentItem;
    public GameObject handInteractionUI;
    public GameObject lanternObject;

    private bool isPlayerNearby;

    void Start()
```

```
{
    currentItem = gameObject;

    handInteractionUI.SetActive(false);
    lanternObject.SetActive(false);
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Reach"))
    {
        isPlayerNearby = true;
        handInteractionUI.SetActive(true);
    }
}

private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Reach"))
    {
        isPlayerNearby = false;
        handInteractionUI.SetActive(false);
    }
}

void Update()
{
    if (isPlayerNearby && Input.GetButtonDown("Interact"))
    {
        HandleLanternPickup();
    }
}

private void HandleLanternPickup()
{
    handInteractionUI.SetActive(false);
    lanternObject.SetActive(true);
    StartCoroutine(RemoveItemAfterDelay());
}
```

```
    private IEnumerator RemoveItemAfterDelay()
    {
        yield return new WaitForSeconds(0.01f);
        Destroy(currentItem);
    }
}
```

End game - Fadumo

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    [Header("Scene Names")]
    public string gameSceneName = "Game"; // Name of the scene to load when starting the
game.

    /// <summary>
    /// Loads the game scene to start the game.
    /// </summary>
    public void B_LoadScene()
    {
        if (!string.IsNullOrEmpty(gameSceneName))
        {
            SceneManager.LoadScene(gameSceneName);
        }
        else
        {
            Debug.LogWarning("Game scene name is not set. Please set it in the inspector.");
        }
    }

    /// <summary>
    /// Quits the application.
    /// </summary>
    public void B_QuitGame()
    {
        Debug.Log("Quit button pressed. Application will exit.");
        Application.Quit();

#if UNITY_EDITOR
```

```
        // Ensures the quit function works during testing in the editor.
        UnityEditor.EditorApplication.isPlaying = false;
#endif
    }
}
```

Lantern Pickup - Wintana

```
using UnityEngine;
using System.Collections;

public class LanternPickup : MonoBehaviour
{
    private GameObject heldItem;
    public GameObject interactionUI;
    public GameObject lanternPrefab;

    private bool playerIsNearby;

    void Start()
    {
        heldItem = gameObject;
        interactionUI.SetActive(false);
        lanternPrefab.SetActive(false);
    }

    private void OnTriggerEnter(Collider collider)
    {
        if (collider.CompareTag("Reach"))
        {
            playerIsNearby = true;
            interactionUI.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider collider)
    {
        if (collider.CompareTag("Reach"))
        {
            playerIsNearby = false;
```

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

    void Update()
    {
        if (playerIsNearby && Input.GetButtonDown("Interact"))
        {
            PickupLantern();
        }
    }

    private void PickupLantern()
    {
        interactionUI.SetActive(false);
        lanternPrefab.SetActive(true);
        StartCoroutine(DestroyItemAfterDelay());
    }

    private IEnumerator
    DestroyItemAfterDelay() {
        yield return new WaitForSeconds(0.01f);
        Destroy(heldItem);
    }
}
```

Sway - Wintana

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

public class Sway : MonoBehaviour
{
    public float swayAmount; // How much the object sways in response to mouse
movement
    public float swayMax; // The maximum amount the object can sway public
```

```
    float swaySmoothSpeed; // Speed at which swaying returns to normal

    private Vector3 originalPosition;

    void Start()
    {
        // Save the object's initial local position
        originalPosition = transform.localPosition;
    }

    void Update()
    {
        // Capture mouse movement and calculate the sway amount
        float swayInputX = -Input.GetAxis("Mouse X") * swayAmount;
        float swayInputY = -Input.GetAxis("Mouse Y") * swayAmount;

        // Clamp the calculated sway values to stay within the defined max range
        swayInputX = Mathf.Clamp(swayInputX, -swayMax, swayMax);
        swayInputY = Mathf.Clamp(swayInputY, -swayMax, swayMax);

        // Set the target position by combining the sway input with the original position Vector3
        targetSwayPosition = new Vector3(swayInputX, swayInputY, 0) + originalPosition;

        // Smoothly interpolate the object's position towards the desired target position
        transform.localPosition = Vector3.Lerp(transform.localPosition, targetSwayPosition,
Time.deltaTime * swaySmoothSpeed);
    }
}
```

Door - Wintana

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class DoorInteraction : MonoBehaviour
{
    public GameObject interactionPrompt; // UI element to indicate interaction possibility
public GameObject warningMessage; // UI element for feedback when the door can't be
opened
    public GameObject playerKey; // Simulates the inventory key
    public GameObject transitionEffect; // Visual effect for scene transition
    public string targetScene; // Name of the scene to load
```

```
private bool isPlayerNearby = false; // Tracks whether the player is near the door

void Start()
{
    // Initially deactivate all UI elements and effects
    interactionPrompt.SetActive(false);
    warningMessage.SetActive(false);
    playerKey.SetActive(false);
    transitionEffect.SetActive(false);
}

void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Reach"))
    {
        isPlayerNearby = true;
        interactionPrompt.SetActive(true); // Show interaction prompt when in range
    }
}

void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Reach"))
    {
        isPlayerNearby = false;
        interactionPrompt.SetActive(false); // Hide interaction prompt when out of range
        warningMessage.SetActive(false); // Clear warning message
    }
}

void Update()
{
    // Handle interaction logic
    if (isPlayerNearby && Input.GetButtonDown("Interact"))
    {
        if (!playerKey.activeInHierarchy)
        {
            // If the key is not collected, show a warning
            warningMessage.SetActive(true);
        }
        else
        {
```

```
            // If the key is present, trigger transition and load the next scene
            interactionPrompt.SetActive(false);
            warningMessage.SetActive(false);
            transitionEffect.SetActive(true);
            StartCoroutine(LoadTargetScene());
        }
    }
}

IEnumerator LoadTargetScene()
{
    // Add a slight delay before transitioning to the next scene
    yield return new WaitForSeconds(0.6f);
    SceneManager.LoadScene(targetScene);
}
}
```