

Unity FPS: Low-Poly Zombie Survival — Implementation & Build Blueprint

1. Project Initialization & Render Pipeline

- **Engine Version:** Unity 2022.3 LTS or 2023.2+ (Universal Render Pipeline - URP).
- **Package Dependencies:**
 - Universal RP (`com.unity.render-pipelines.universal`)
 - AI Navigation (`com.unity.ai.navigation`)
 - TextMeshPro (`com.unity.textmeshpro`)
 - Input System (`com.unity.inputsystem`)
- **Layers & Physics Matrix:**
 - Configure layers: Default, TransparentFX, Ignore Raycast, Water, UI, Player, Enemy, Ground, Interactable, Projectile.
 - Open **Project Settings > Physics:** Uncheck collision between Enemy and Enemy (to avoid snagging), and ensure Projectile interacts with Enemy and Ground while ignoring Player.

1.2 Synty Studios Asset Ingestion

- **Model Import Settings:** Select all character FBX files from your Synty pack (e.g., *POLYGON - Apocalypse, Zombies, or City*).
- In the **Rig** tab, change **Animation Type** to *Humanoid*. Click **Apply**.
- Under the **Materials** tab, ensure the shader is converted to `Universal Render Pipeline/Lit` or `Universal Render Pipeline/Simple Lit` using the bundled Synty texture palette atlas.
- **Socket Hierarchy:**
 - Expand the Player character model hierarchy: find `Root > Pelvis > Spine > Upper_Chest > R_Clavicle > R_UpperArm > R_Forearm > R_Hand`.
 - Create an empty child GameObject named `Weapon_Socket_R` to act as the parent anchor for equipped guns.

-----2. Core System Scripts

2.1 Player Controller (`PlayerController.cs`)

Handles first-person locomotion, camera pitch/yaw clamping, sprinting, and head bob.
using UnityEngine;
using UnityEngine.InputSystem;

```
[RequireComponent(typeof(CharacterController))]  
public class PlayerController : MonoBehaviour  
{  
    [Header("Movement Settings")]  
    [SerializeField] private float walkSpeed = 5.0f;  
    [SerializeField] private float sprintSpeed = 8.5f;  
    [SerializeField] private float gravity = -19.62f;  
    [SerializeField] private float jumpHeight = 1.2f;  
  
    [Header("Look Settings")]
```

```

[SerializeField] private Transform cameraHolder;
[SerializeField] private float mouseSensitivity = 2.0f;
[SerializeField] private float upDownLookLimit = 80.0f;

private CharacterController characterController;
private Vector3 velocity;
private Vector2 moveInput;
private Vector2 lookInput;
private float verticalRotation = 0f;
private bool isSprinting;
private bool jumpRequested;

private void Awake()
{
    characterController = GetComponent<CharacterController>();
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}

private void Update()
{
    HandleRotation();
    HandleMovement();
}

public void OnMove(InputValue value) => moveInput = value.Get<Vector2>();
public void OnLook(InputValue value) => lookInput = value.Get<Vector2>();
public void OnSprint(InputValue value) => isSprinting = value.isPressed;
public void OnJump(InputValue value) { if (value.isPressed &&
characterController.isGrounded) jumpRequested = true; }

private void HandleRotation()
{
    float mouseX = lookInput.x * mouseSensitivity * 0.1f;
    float mouseY = lookInput.y * mouseSensitivity * 0.1f;

    transform.Rotate(Vector3.up * mouseX);

    verticalRotation -= mouseY;
    verticalRotation = Mathf.Clamp(verticalRotation, -upDownLookLimit,
upDownLookLimit);
    cameraHolder.localRotation = Quaternion.Euler(verticalRotation, 0f, 0f);
}

private void HandleMovement()
{
    if (characterController.isGrounded && velocity.y < 0)
    {

```

```

        velocity.y = -2f;
    }

    float currentSpeed = isSprinting ? sprintSpeed : walkSpeed;
    Vector3 moveDirection = transform.right * moveInput.x + transform.forward *
moveInput.y;
    characterController.Move(moveDirection * currentSpeed * Time.deltaTime);

    if (jumpRequested)
    {
        velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        jumpRequested = false;
    }

    velocity.y += gravity * Time.deltaTime;
    characterController.Move(velocity * Time.deltaTime);
}
}

```

2.2 Player Health Controller ([PlayerHealthController.cs](#))

Manages player hit points, damage overlays, and game-over routing.
using UnityEngine;
using System;

```

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

    [SerializeField] private int maxHealth = 100;
    private int currentHealth;

    public event Action<int, int> OnHealthChanged;
    public event Action OnPlayerDied;

    private void Awake()
    {
        if (Instance != null && Instance != this) Destroy(gameObject);
        else Instance = this;
    }

    private void Start()
    {
        currentHealth = maxHealth;
        OnHealthChanged?.Invoke(currentHealth, maxHealth);
    }

    public void TakeDamage(int damageAmount)

```

```

{
    if (currentHealth <= 0) return;

    currentHealth = Mathf.Max(0, currentHealth - damageAmount);
    OnHealthChanged?.Invoke(currentHealth, maxHealth);

    if (currentHealth <= 0)
    {
        Die();
    }
}

public void Heal(int healAmount)
{
    currentHealth = Mathf.Min(maxHealth, currentHealth + healAmount);
    OnHealthChanged?.Invoke(currentHealth, maxHealth);
}

private void Die()
{
    OnPlayerDied?.Invoke();
    UIController.Instance.ShowGameOverScreen();
}
}

```

2.3 Weapons & Ballistics ([Weapon.cs](#) & [WeaponsController.cs](#))

Raycast shooting system with bullet spread, reload timers, range constraints, and particle triggers.

```

using System.Collections;
using UnityEngine;

```

```

public class Weapon : MonoBehaviour
{
    [Header("Weapon Stats")]
    public string weaponName = "Assault Rifle";
    public int damage = 25;
    public float fireRate = 0.12f;
    public float range = 100f;
    public int maxAmmo = 30;
    public int currentAmmo;
    public float reloadTime = 1.5f;

    [Header("Effects")]
    [SerializeField] private ParticleSystem muzzleFlash;
    [SerializeField] private GameObject impactEffectPrefab;
    [SerializeField] private Transform muzzlePoint;
}

```

```

private float nextFireTime;
private bool isReloading;

private void Awake() => currentAmmo = maxAmmo;

public bool CanShoot() => Time.time >= nextFireTime && !isReloading && currentAmmo >
0;

public void Shoot(Camera playerCamera, LayerMask hitMask)
{
    nextFireTime = Time.time + fireRate;
    currentAmmo--;

    if (muzzleFlash) muzzleFlash.Play();

    Ray ray = playerCamera.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
    if (Physics.Raycast(ray, out RaycastHit hit, range, hitMask))
    {
        if (hit.collider.TryGetComponent<EnemyController>(out var enemy))
        {
            enemy.TakeDamage(damage);
        }

        if (impactEffectPrefab)
        {
            GameObject impact = Instantiate(impactEffectPrefab, hit.point,
Quaternion.LookRotation(hit.normal));
            Destroy(impact, 2.0f);
        }
    }
}

public IEnumerator ReloadRoutine()
{
    if (isReloading || currentAmmo == maxAmmo) yield break;

    isReloading = true;
    yield return new WaitForSeconds(reloadTime);

    currentAmmo = maxAmmo;
    isReloading = false;
}
}
using UnityEngine;
using UnityEngine.InputSystem;

public class WeaponsController : MonoBehaviour
{

```

```

[SerializeField] private Weapon[] weaponSlots;
[SerializeField] private Camera playerCamera;
[SerializeField] private LayerMask hitMask;

private int activeWeaponIndex = 0;
private bool isFiring;

private void Start() => EquipWeapon(0);

private void Update()
{
    if (isFiring && weaponSlots[activeWeaponIndex].CanShoot())
    {
        weaponSlots[activeWeaponIndex].Shoot(playerCamera, hitMask);

        UIController.Instance.UpdateAmmoDisplay(weaponSlots[activeWeaponIndex].currentAmmo,
        weaponSlots[activeWeaponIndex].maxAmmo);
    }
}

public void OnFire(InputValue value) => isFiring = value.isPressed;

public void OnReload(InputValue value)
{
    if (value.isPressed)
    {
        StartCoroutine(weaponSlots[activeWeaponIndex].ReloadRoutine());
    }
}

public void OnNextWeapon(InputValue value)
{
    if (value.isPressed)
    {
        int next = (activeWeaponIndex + 1) % weaponSlots.Length;
        EquipWeapon(next);
    }
}

private void EquipWeapon(int index)
{
    for (int i = 0; i < weaponSlots.Length; i++)
    {
        weaponSlots[i].gameObject.SetActive(i == index);
    }
    activeWeaponIndex = index;
}

```

```

UIController.Instance.UpdateAmmoDisplay(weaponSlots[activeWeaponIndex].currentAmmo,
weaponSlots[activeWeaponIndex].maxAmmo);
    }
}

```

2.4 Zombie AI (`EnemyController.cs`)

Uses Unity's NavMesh Agent for pathfinding, player detection, attack ranges, and death synchronization.

```
using UnityEngine;
```

```
using UnityEngine.AI;
```

```

[RequireComponent(typeof(NavMeshAgent))]
public class EnemyController : MonoBehaviour
{
    public enum State { Idle, Chase, Attack, Dead }
    private State currentState = State.Idle;

    [Header("Stats")]
    [SerializeField] private int maxHealth = 60;
    [SerializeField] private int attackDamage = 15;
    [SerializeField] private float attackRange = 1.8f;
    [SerializeField] private float attackCooldown = 1.2f;
    [SerializeField] private float detectionRadius = 18f;

    [Header("References")]
    [SerializeField] private Animator animator;

    private NavMeshAgent agent;
    private Transform playerTransform;
    private int currentHealth;
    private float nextAttackTime;

    private void Awake()
    {
        agent = GetComponent<NavMeshAgent>();
        currentHealth = maxHealth;
    }

    private void Start()
    {
        playerTransform = GameObject.FindGameObjectWithTag("Player")?.transform;
        AreaClearanceManager.Instance?.RegisterEnemy(this);
    }

    private void Update()
    {

```

```

    if (currentState == State.Dead || playerTransform == null) return;

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

    switch (currentState)
    {
        case State.Idle:
            if (distanceToPlayer <= detectionRadius)
                currentState = State.Chase;
            break;

        case State.Chase:
            agent.SetDestination(playerTransform.position);
            animator.SetFloat("Speed", agent.velocity.magnitude);

            if (distanceToPlayer <= attackRange)
            {
                currentState = State.Attack;
                agent.ResetPath();
            }
            break;

        case State.Attack:
            transform.LookAt(new Vector3(playerTransform.position.x, transform.position.y,
playerTransform.position.z));

            if (Time.time >= nextAttackTime)
            {
                animator.SetTrigger("Attack");
                PlayerHealthController.Instance.TakeDamage(attackDamage);
                nextAttackTime = Time.time + attackCooldown;
            }

            if (distanceToPlayer > attackRange)
                currentState = State.Chase;
            break;
    }
}

public void TakeDamage(int damage)
{
    if (currentState == State.Dead) return;

    currentHealth -= damage;
    animator.SetTrigger("Hit");

    if (currentHealth <= 0)

```

```

    {
        Die();
    }
}

private void Die()
{
    currentState = State.Dead;
    agent.enabled = false;
    animator.SetBool("IsDead", true);
    GetComponent<Collider>().enabled = false;
    AreaClearanceManager.Instance?.UnregisterEnemy(this);
    Destroy(gameObject, 5f);
}
}

```

2.5 Area Objective Manager ([AreaClearanceManager.cs](#))

Tracks the elimination of all zombies in the area and unlocks the level exit.

using System.Collections.Generic;

using UnityEngine;

```

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

    [SerializeField] private LevelExit levelExit;
    private readonly HashSet<EnemyController> activeEnemies = new
    HashSet<EnemyController>();
    private bool areaCleared = false;

    private void Awake()
    {
        if (Instance != null && Instance != this) Destroy(gameObject);
        else Instance = this;
    }

    private void Start()
    {
        if (levelExit != null)
        {
            levelExit.gameObject.SetActive(false);
        }
    }

    public void RegisterEnemy(EnemyController enemy)
    {
        if (!activeEnemies.Contains(enemy))

```

```

    {
        activeEnemies.Add(enemy);
        UpdateUI();
    }
}

public void UnregisterEnemy(EnemyController enemy)
{
    if (activeEnemies.Contains(enemy))
    {
        activeEnemies.Remove(enemy);
        UpdateUI();

        if (activeEnemies.Count == 0 && !areaCleared)
        {
            UnlockArea();
        }
    }
}

private void UnlockArea()
{
    areaCleared = true;
    if (levelExit != null)
    {
        levelExit.gameObject.SetActive(true);
    }
    UIController.Instance.ShowNotification("Area Cleared! Proceed to Exit.");
}

private void UpdateUI()
{
    UIController.Instance.UpdateZombiesRemaining(activeEnemies.Count);
}
}

```

2.6 Level Exit & Transition ([LevelExit.cs](#))

Transitions the player to the next scene when entered.

```
using UnityEngine;
```

```
using UnityEngine.SceneManagement;
```

```
public class LevelExit : MonoBehaviour
```

```
{
    [SerializeField] private string nextSceneName = "Level 2";
```

```
    private void OnTriggerEnter(Collider other)
```

```
{
```

```

        if (other.CompareTag("Player"))
        {
            SceneManager.LoadScene(nextSceneName);
        }
    }
}

```

2.7 UI Controller (UIController.cs)

Displays remaining zombies, current ammo, health, notifications, and menus.

```

using UnityEngine;
using TMPro;
using UnityEngine.UI;

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

    [Header("HUD Elements")]
    [SerializeField] private Slider healthBar;
    [SerializeField] private TextMeshProUGUI ammoText;
    [SerializeField] private TextMeshProUGUI zombieCountText;
    [SerializeField] private TextMeshProUGUI notificationText;
    [SerializeField] private GameObject gameOverPanel;

    private void Awake()
    {
        if (Instance != null && Instance != this) Destroy(gameObject);
        else Instance = this;
    }

    private void Start()
    {
        if (PlayerHealthController.Instance != null)
        {
            PlayerHealthController.Instance.OnHealthChanged += UpdateHealthBar;
        }
        if (gameOverPanel != null) gameOverPanel.SetActive(false);
    }

    public void UpdateHealthBar(int current, int max)
    {
        if (healthBar) healthBar.value = (float)current / max;
    }

    public void UpdateAmmoDisplay(int current, int max)
    {
        if (ammoText) ammoText.text = $"{current} / {max}";
    }
}

```

```

}

public void UpdateZombiesRemaining(int count)
{
    if (zombieCountText) zombieCountText.text = $"Zombies Left: {count}";
}

public void ShowNotification(string message)
{
    if (notificationText)
    {
        notificationText.text = message;
        CancelInvoke(nameof(ClearNotification));
        Invoke(nameof(ClearNotification), 4f);
    }
}

private void ClearNotification()
{
    if (notificationText) notificationText.text = "";
}

public void ShowGameOverScreen()
{
    if (gameOverPanel)
    {
        gameOverPanel.SetActive(true);
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
}

```

-----3. Unity Scene Construction Step-by-Step3.1 NavMesh & Environment Setup

- **Layout Level Geometry:** Build the arena using Synty environment modular prefabs (walls, floors, barricades). Mark all static environment colliders as **Static** in the top-right Inspector.
- **Bake NavMesh:** Go to **Window > AI > Navigation** (or add a **NavMeshSurface** component to an empty root GameObject named **Environment**). Set Agent Type to *Humanoid*. Click **Bake**. Ensure all walkways are highlighted blue.

3.2 Player Setup

- Create an empty GameObject named **Player** at (0, 1, 0).
- Add a **CharacterController** component: Height: 1.8, Radius: 0.35, Center: (0, 0.9, 0).

- Add `PlayerController.cs`, `PlayerHealthController.cs`, and `WeaponsController.cs`.
- Create a child GameObject named `CameraHolder` at position `(0, 1.6, 0)`. Attach the `Main Camera` as a child of `CameraHolder` at position `(0, 0, 0)`.
- Attach a Synty gun model to the camera or character hand socket:
 - Add the `Weapon.cs` component to the gun.
 - Assign a child `MuzzlePoint` transform at the tip of the barrel.
 - Wire `Weapon` into the `WeaponsController` slots.
- Tag the root GameObject as `Player`. Set its Layer to `Player`.

3.3 Zombie Prefab Construction

- Drag a Synty Zombie FBX model into the scene hierarchy.
- Add a `CapsuleCollider`: Center: `(0, 0.9, 0)`, Radius: `0.3`, Height: `1.8`.
- Add a `NavMeshAgent`: Speed: `3.5`, Angular Speed: `180`, Stopping Distance: `1.5`, Height: `1.8`, Radius: `0.3`.
- Add an `Animator` controller: Set parameters: `Speed` (Float), `Attack` (Trigger), `Hit` (Trigger), `IsDead` (Bool). Hook up Synty Zombie animations.
- Add `EnemyController.cs`. Assign the Animator reference.
- Tag the GameObject as `Enemy` and set its layer to `Enemy`.
- Drag this setup into your `Assets/Prefabs/Enemies` folder to generate `Enemy.prefab`.

3.4 Objective & Exit Wiring

- Place an empty GameObject named `GameManager` in the scene. Attach `AreaClearanceManager.cs`.
- Create a level exit zone: Add a 3D Cube with an emissive material or Synty portal asset. Check **Is Trigger** on its `BoxCollider`.
- Add `LevelExit.cs` and configure *Next Scene Name* to `Level 2`. Assign the exit object to the *LevelExit* field on the `AreaClearanceManager`.
- Scatter `Enemy.prefab` instances around the area. Each will auto-register on spawn and decrement the clearance tracker when defeated.

-----4. UI Canvas & Post-Processing (URP)4.1 HUD Hierarchy

- Create a Canvas set to **Screen Space - Overlay**. Add a Canvas Scaler (Reference Resolution: `1920x1080`, Match Width/Height: `0.5`).
- Build the following child hierarchy:
 - `HealthBar` (UI Slider with green fill and dark red background)
 - `AmmoText` (`TextMeshProUGUI` positioned at the bottom right)
 - `ZombieCountText` (`TextMeshProUGUI` positioned top-center: "`Zombies Left: X`")

- `NotificationText` (`TextMeshProUGUI` centered, large dynamic alert font)
- `GameOverPanel` (Image overlay covering the full screen, containing a "Restart" button that calls `SceneManager.LoadScene(SceneManager.GetActiveScene().name)`)
- Attach `UIController.cs` to the Canvas root and drag all references into the Inspector fields.

4.2 URP Global Volume (Post-Processing)

- Add an empty GameObject named `Global Volume`.
- Add a Volume component, set Mode to **Global**, and assign your `DefaultVolumeProfile`.
- Add the following overrides for a gritty low-poly look:
 - **Vignette:** Intensity `0.35`, Smoothness `0.4`.
 - **Color Adjustments:** Post Exposure `0.2`, Contrast `20`, Saturation `-15` (for a muted horror feel).
 - **Bloom:** Threshold `1.0`, Intensity `1.2`.
 - **Film Grain:** Type *Thin 1*, Intensity `0.3`.

-----5. Verification Checklist

- **Locomotion:** Player moves, jumps, sprints, and looks around without vertical jitter or camera flipping.
- **NavMesh Pathing:** Zombies track player line-of-sight and navigate around modular obstacles.
- **Raycast Registration:** Firing hits zombies, spawns impact sparks, applies damage, and updates the ammo counter.
- **Attack Triggers:** Closing within 1.8 units triggers the zombie attack animation and reduces player health.
- **Area Clearance Logic:** Zombie deaths decrement the UI counter; eliminating the final zombie displays the alert banner and activates the exit trigger.
- **Level Transition:** Walking into the active exit loads the next designated scene cleanly.