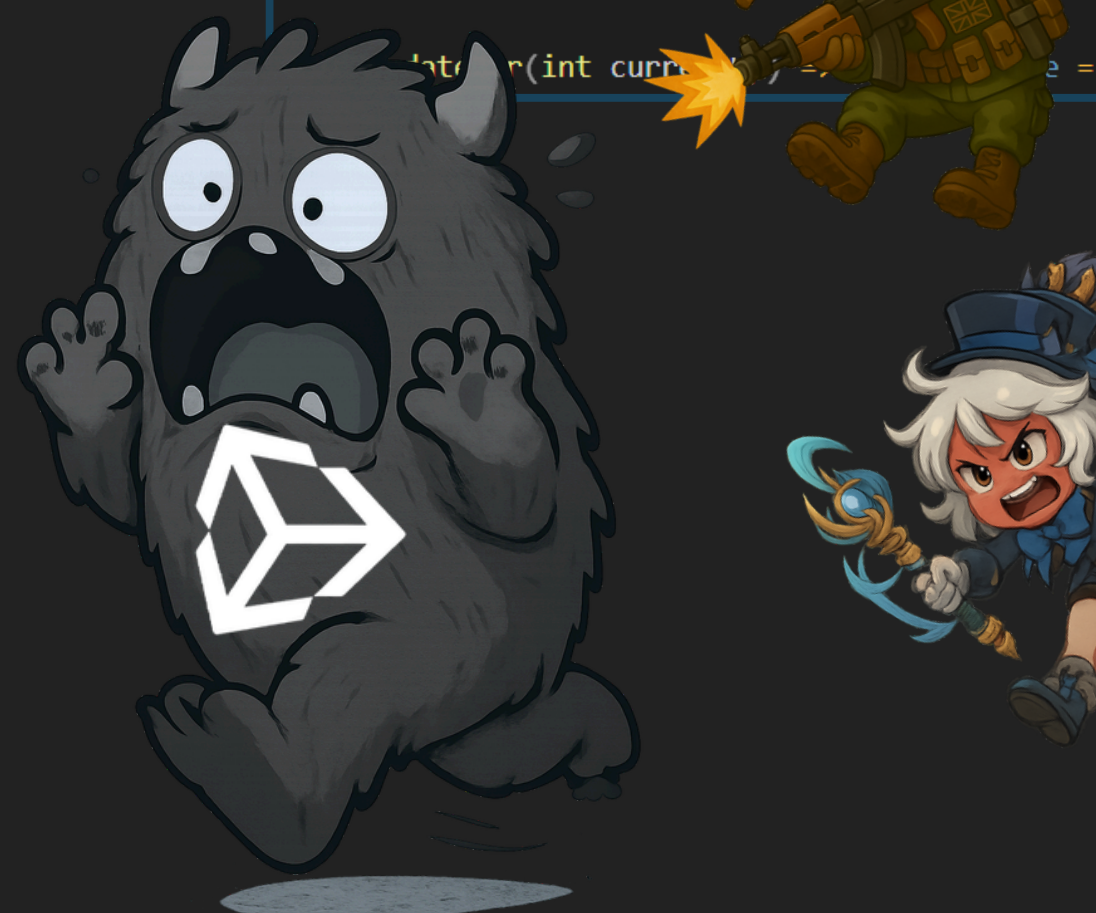
UNITY MINI-HACKS

PART - 1 to 13

GET ALL 64 MINI HACKS



UNITY MINI-HACKS

MINI HACK 1: Input in Update()

```
void FixedUpdate() {
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");

    Vector3 move = new Vector3(h, 0, v) * moveSpeed;
    rb.velocity = move;
}
```

VS

```
void Update() {
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");
    movementInput = new Vector3(h, 0, v);
}

void FixedUpdate() {
    Vector3 move = movementInput * moveSpeed;
    rb.velocity = move;
}
```

MINI HACK 2: Update UI in Update()

```
void Update() {
    scoreText.text = "Score: " + playerScore.ToString();
}
```

VS

```
public void OnScoreChanged(int newScore) {
    scoreText.text = "Score: " + newScore;
}
```

MINI HACK 3: String in Animator

```
animator.SetBool("IsRunning", true);
```

VS

```
static readonly int IsRunningHash =
        Animator.StringToHash("IsRunning");

animator.SetBool(IsRunningHash, true);
```

MINI HACK 4: Use WaitWhile()

```
IEnumerator WaitForCutscene_Bad(){

    // Keep waiting while the cutscene is still playing.
    while (IsCutscenePlaying()) =>
    {
        yield return null;
    }
    EnablePlayerControl();
}
```

VS

```
IEnumerator WaitForCutscene_Good(){
    // This will pause the coroutine while the cutscene is playing.
    yield return new WaitWhile(() => IsCutscenePlaying());
    EnablePlayerControl();
}
```

MINI HACK 5: Set Position & Rotation

```
// BAD CODE
void UpdateTransform(Vector3 newPosition,
                     Quaternion newRotation)
{
    // Causes a transform update
    transform.position = newPosition;
    // Causes another transform update
    transform.rotation = newRotation;
}
```

VS

```
// GOOD CODE
void UpdateTransform(Vector3 newPosition,
                     Quaternion newRotation)
{
    // Sets position and rotation in a single operation
    transform.SetPositionAndRotation(
        newPosition,
        newRotation);
}
```

UNITY MINI-HACKS

PART - 2

MINI HACK 1: Use Namespaces

```
// No namespace, everything in the global scope
public class Enemy : MonoBehaviour
{
    // Implementation details
}
```

VS

```
namespace MyDreamGame.Enemies
{
    public class Enemy : MonoBehaviour
    {
        // Implementation details
    }
}
```

MINI HACK 2: Public Fields vs [SerializeField]

```
public float playerSpeed = 5f;
```

VS

```
[SerializeField] private float playerSpeed = 5f;
```

MINI HACK 3: Invoke() vs Coroutines

```
Invoke("SpawnEnemy", 2f);
```

VS

```
StartCoroutine(DelayedSpawn());

IEnumerator DelayedSpawn()
{
    yield return new WaitForSeconds(2f);
    SpawnEnemy();
}
```

MINI HACK 4: Use readonly

```
public float playerSpeed = 5f;
```

VS

```
public static readonly float GRAVITY = 9.81f;
```

MINI HACK 5: Loadscene vs LoadSceneAsync

```
SceneManager.LoadScene("Level2"); // Synchronous call
```

VS

```
public IEnumerator LoadNextScene(SceneName sceneToLoad)
{
    AsyncOperation loadOp =
    SceneManager.LoadSceneAsync(sceneToLoad.ToString());
    //... more code for showing loading bar
}
```

UNITY MINI-HACKS

PART - 3

MH 1: Use Variable Naming Convention

```
public float maxhealth = 100f;
private float currentHealth;
const string playername = "Hero";
```

VS

```
// Public Fields: PascalCase
public float MaxHealth = 100f;
// Private Fields: _camelCase
private float _currentHealth;
//Constants: ALL_CAPS
private const string PLAYER_DEFAULT_NAME = "Hero";
```

MH 2: Use Auto-Implemented Property

```
private int _score;
public int GetScore() => return _score;
```

VS

```
public int Score { get; private set; }
```

MH 3: Use Enum Flags

```
public bool isWalking;
public bool isIsAir;
public bool isAttacking;
```

VS

```
[System.Flags] // Enables bitwise operations for the enum
public enum PlayerStatus{
    None      = 0,       // 0000 - Represents no specific state
    Walking   = 1 << 0,  // 0001 (Value 1)
    InAir = 1 << 1,      // 0010 (Value 2)
    Attacking     = 1 << 2, // 0100 (Value 4)
}
```

MH 4: Use Structs

```
public float currentHealth;
public float maxHealth;
public float armorValue;
public float moveSpeed;
```

VS

```
public struct EnemyStats{
    public float CurrentHealth;
    public float MaxHealth;
    public float ArmorValue;
    public float MoveSpeed;
    // Add more stats here
    // (e.g., Damage, AttackRange) easily
}
```

MH 5: Use Interfaces

```
public class Sword{
    public void Swing() { Debug.Log("Sword Swing!"); }
}

public class Gun{
  public void Fire() { Debug.Log("Gun Fire!"); }
}
```

VS

```
public interface IWeapon{
    void PerformAttack();
}

public class Sword : IWeapon
public class Gun : IWeapon
```

UNITY MINI-HACKS

PART - 4

MH 1: Use Late update for Follow Camera

```
// Runs before target might
// finish its Update() movement
void Update() {
    camera.transform.position =
            player.transform.position + offset;
}
```

VS

```
void LateUpdate()
{
    camera.transform.position =
            player.transform.position + offset;
}
```

MH 2: Use Unity's Action Events

```
void Update()
{
    slider.value = _player.Hp;
}
```

VS

```
//In Subject Class
public event Action<int> OnHealthChanged;
OnHealthChanged?.Invoke(updatedHp);

//In Listener Class
void OnEnable()  => _player.OnHealthChanged += UpdateBar;
void OnDisable() => _player.OnHealthChanged -= UpdateBar;

void UpdateBar(int currentHp) => slider.value = currentHp;
```

MH 3: Use OnValidate()

```
public Animator anim;            // often left unassigned
void Start()
{
    // Last-second Band-Aids 😬
    if (anim == null) anim = GetComponent<Animator>();
}
```

VS

```
#if UNITY_EDITOR        // stripped from builds
void OnValidate()
{
    // Auto-wire the Animator if the designer forgot
    if (anim == null)
        anim = GetComponent<Animator>();
}
#endif
```

MH 4: Achieve frame rate independence

```
void Update()
{
    // Moves 'speed' units *per frame*.
    // Faster FPS = faster movement.
    transform.Translate(Vector3.forward * speed);
}
```

VS

```
void Update()
{
    // Moves 'speed' units *per second*.
    // Consistent speed regardless of FPS.
    transform.Translate(Vector3.forward * speed * Time.deltaTime);
}
```

MH 5: Cache references in Awake(); do gameplay in Start()

```
void Start()
{
    _gm = FindObjectOfType<GameManager>();
    _gm.RegisterPlayer(this);
}
```

VS

```
void Awake()
{
    _gm = GameManager.Instance;   // singleton set in GameManager.Awake
}

void Start()
{
    _gm.RegisterPlayer(this);     // guaranteed non-null
}
```

UNITY MINI-HACKS

PART - 5

ATTRIBUTES SPECIAL

MH 1: Use [Header()]

```
public float damage;
public float fireRate;
public int   magazineSize;
public AudioClip shootSfx;
public AudioClip reloadSfx;
```

VS

```
[Header("Stats")]
public float damage = 25f;
public float fireRate = 0.15f;
public int   magazineSize = 30;

[Header("Audio")]
public AudioClip shootSfx;
public AudioClip reloadSfx;
```

MH 2: Use [Tooltip]

```
// what is this?
[SerializeField] private float fuseTime = 3f;
```

VS

```
[Tooltip("Seconds before the grenade explodes after being thrown")]
[SerializeField] private float fuseTime = 3f;
```

MH 3: Use [Range()]

```
// can be set 0 or 1000
public float wanderRadius = 50f;
```

VS

```
// editable via neat slider
[Range(5f, 20f)]
public float wanderRadius = 12f;
```

MH 4: Use [RequireComponent()]

```
public class PlayerAudio : MonoBehaviour{
    private AudioSource _src;
    void Awake()    {
        // might be null!
        _src = GetComponent<AudioSource>();
    }
}
```

VS

```
[RequireComponent(typeof(AudioSource))]
public class PlayerAudio : MonoBehaviour{
    private AudioSource _src;
    void Awake()          {
        // guaranteed to succeed
        _src = GetComponent<AudioSource>();
    }
}
```

MH 5: Use [HideInInspector()]

```
// toggled at runtime;
//should not be editable
public bool isOpen;
```

VS

```
[HideInInspector] public bool isOpen;
```

UNITY MINI-HACKS

PART - 6

MH 1: Mathf.Approximately() vs “==”

```
if (health == 0f) {
    Die();
}
```

VS

```
if (Mathf.Approximately(health, 0f)) {
    Die();
}
```

MH 2: Debug Logs with Color

```
Debug.Log("Player reached checkpoint");
```

VS

```
Debug.Log("<color=green>Player reached checkpoint</color>");
```

MH 3: Use contextMenu to Trigger Methods from Inspector

```
private void ResetPosition() {
    transform.position = Vector3.zero;
}
```

VS

```
[ContextMenu("Reset Player Position")]
private void ResetPosition() {
    transform.position = Vector3.zero;
}
```

MH 4: CompareTag() vs “==”

```
void OnCollisionEnter(Collision collision) {
    if (collision.gameObject.tag == "Player") {
        HandleHit();
    }
}
```

VS

```
void OnCollisionEnter(Collision collision) {
    if (collision.gameObject.CompareTag("Player")) {
        HandleHit();
    }
}
```

MH 5: Cache WaitForSeconds

```
IEnumerator FadeOut() {
    DoFadeStep();
    yield return new WaitForSeconds(0.1f);
}
```

VS

```
private WaitForSeconds fadeDelay = new WaitForSeconds(0.1f);
IEnumerator FadeOut() {
    DoFadeStep();
    yield return fadeDelay;
}
```

UNITY MINI-HACKS

PART - 7

MH 1: Use [Header()]

```
string msg = "";
for(int i=0;i<1000;i++) msg += i;
```

VS

```
var sb = new StringBuilder();
for(int i=0;i<1000;i++) sb.Append(i);
```

MH 2: TryGetValue for Dictionaries

```
if(scores.ContainsKey(name))
    score = scores[name];
```

VS

```
if(scores.TryGetValue(name, out score))
    /* use score */;
```

MH 3: Minimize Debug.Log in Builds

```
Debug.Log(transform.position);
```

VS

```
#if UNITY_EDITOR
Debug.Log(transform.position);
#endif
```

MH 4: Object Pool over Instantiate/Destroy

```
Destroy(Instantiate(bullet),2f);
```

VS

```
var bullet = pool.Get();
pool.Release(bullet);
```

MH 5: RaycastNonAlloc vs RaycastAlloc

```
void Update() { Physics.Raycast(ray, out var hit); }
```

VS

```
static readonly RaycastHit[] hits = new RaycastHit[1];
void Update() { Physics.RaycastNonAlloc(ray, hits); }
```

UNITY MINI-HACKS

PART - 8

ATTRIBUTES SPECIAL PART - 2

MH 1: Use [TextArea]

```
public string description;
```

VS

```
[TextArea] public string description;
```

MH 2: Use [DisallowMultipleComponent]

```
public class Health : MonoBehaviour { }
// Two Health scripts accidentally added in Inspector
```

VS

```
[DisallowMultipleComponent]
public class Health : MonoBehaviour { }
```

MH 3: Use [MenuItem]

```
static void ClearPrefs() { PlayerPrefs.DeleteAll(); }
```

VS

```
[MenuItem("Tools/Clear Player Prefs")]
static void ClearPrefs() { PlayerPrefs.DeleteAll();
```

MH 4: Use [Space]

```
public int hp; public int armor;
```

VS

```
public int hp;
[Space]
public int armor;
```

MH 5: Use [System.Flags]

```
public bool walk, jump, shoot;
```

VS

```
[System.Flags]
public enum Abilities { Walk=1, Jump=2, Shoot=4 }
```

UNITY MINI-HACKS

PART - 9

MH 1: Proper Physics Rotation

```
// Frame-based spin – bypasses physics, causes jitter
void Update() {
    transform.Rotate(0, turnSpeed * Time.deltaTime, 0);
}
```

VS

```
// Physics-accurate rotation
void FixedUpdate() {
    rb.MoveRotation(targetRotation);
}
```

MH 2: Smooth Follow Camera

```
void LateUpdate() {
    cam.position = target.position + offset;
}
```

VS

```
Vector3 vel;
void LateUpdate() {
    cam.position = Vector3.SmoothDamp(
    cam.position, target.position + offset, ref vel, 0.15f);
}
```

MH 3: Clamp Rigidbody Velocity

```
// Can overshoot max speed
rb.velocity += transform.forward * accel;
```

VS

```
rb.velocity = Vector3.ClampMagnitude(rb.velocity, maxSpeed);
```

MH 4: Additive Scene Streaming

```
SceneManager.LoadScene(nextScene);
```

VS

```
var op = SceneManager.LoadSceneAsync(s, Additive);
op.allowSceneActivation = false;
op.allowSceneActivation = true;
```

MH 5: Fix 360° Angle Wrap Jitter

```
Mathf.Lerp(350f, 10f, 0.1f);  // goes 340° left!
```

VS

```
Mathf.LerpAngle(350f, 10f, 0.1f);  // shortest path
```

UNITY MINI-HACKS

PART - 10

MH 1: Zero-Garbage Area Scans

```
// Creates new array
Physics.OverlapSphere(pos, rad, mask);
```

VS

```
Collider[] hits = new Collider[32]; // Array once
// Reuses array
Physics.OverlapSphereNonAlloc(pos, rad, hits, mask);
```

MH 2: ScriptableObject Configs

```
float jumpForce = 12f;
```

VS

```
float jumpForce = settings.JumpForce;
```

MH 3: Squared Distance for Cheap Range Checks

```
if(Vector3.Distance(a,b) < r) Chase();
```

VS

```
if((a-b).sqrMagnitude < r*r) Chase();
```

MH 4: Quaternion.Slerp for Smooth Rotation

```
transform.eulerAngles = Vector3.Lerp(a,b,t);
```

VS

```
transform.rotation = Quaternion.Slerp(a,b,t);
```

MH 5: Hidden Inheritance Fix for Unity's Editor

```
public class BaseItem { public string itemName; }
public class Sword : BaseItem { public int damage; }
public class PlayerInventory : MonoBehaviour{
    /*Will lose Sword's 'damage'*/
    public List<BaseItem> items;}
```

VS

```
public class BaseItem { public string itemName; }
public class Sword : BaseItem { public int damage; }
public class PlayerInventory : MonoBehaviour{
    // Saves 'damage' for Swords in the list
    [SerializeReference] public List<BaseItem> items;}
```

UNITY MINI-HACKS

PART - 11

MH 1: Reproducible Randomness (Seeded)

```
// Different each run
float value = UnityEngine.Random.Range(0f, 1f);
```

VS

```
// Same sequence every run
System.Random rng = new System.Random(123);
float value = (float)rng.NextDouble();
```

MH 2: UI/Pause Animation Control

```
// Pauses with the game
fade += Time.deltaTime;
```

VS

```
// Animates during pause
fade += Time.unscaledDeltaTime;
```

MH 3: Easy Sprite Flipping

```
// Can affect children/colliders
transform.localScale = new Vector3(-1, 1, 1);
```

VS

```
// Simple, clean flip
spriteRenderer.flipX = true;
```

MH 4: Live Scripting in Editor

```
// Runs only in Editor build
#if UNITY_EDITOR
void Update(){ Sync(); }
#endif
```

VS

```
[ExecuteAlways] // Runs in Editor and Play
public class MyScript : MonoBehaviour
{
    void Update(){ Sync(); }
}
```

MH 5: Pinpoint Performance Problems

```
// No insight
DoBigTask();
```

VS

```
// See timing in Profiler
Profiler.BeginSample("MyTask");
DoBigTask();
Profiler.EndSample();
```

UNITY MINI-HACKS

PART - 12

MH 1: Register / Unregister Events

```
void Start() {
    GameEvents.OnHit += TakeDamage; // never removed!
}
```

VS

```
void OnEnable()  => GameEvents.OnHit += TakeDamage;
void OnDisable() => GameEvents.OnHit -= TakeDamage;
```

MH 2: Fade Whole UI

```
foreach (var img in GetComponentsInChildren<Image>()) {
    img.color = new Color(
        img.color.r, img.color.g, img.color.b, t
    );
}
```

VS

```
CanvasGroup cg = GetComponent<CanvasGroup>();
cg.alpha = t;
cg.interactable = t > .9f;
```

MH 3: Clamp Player Speed

```
if (move.magnitude > maxSpeed) {
    move = move.normalized * maxSpeed;}
```

VS

```
move = Vector3.ClampMagnitude(move, maxSpeed);
```

MH 4: Cap Mobile FPS

```
// do nothing - relies on default vsync
```

VS

```
void Awake() {
    Application.targetFrameRate = 60; }
```

MH 5: Full-Screen Effects with GPU

```
Texture2D frame = new(Screen.width, Screen.height);
frame.ReadPixels(
    new Rect(0, 0, Screen.width, Screen.height), 0, 0
);
frame.Apply(); // CPU copy each frame!
```

VS

```
void OnRenderImage(RenderTexture src,
                   RenderTexture dst) {
    Graphics.Blit(src, dst, postFxMat);
                        /*GPU-only pass*/}
```

UNITY MINI-HACKS

PART - 13

MH 1: Any() vs. Count() > 0

```cpp
void Update() {
    if (enemies.Where(e => e.health > 50).Count() > 0) {
        // Strong enemies exist
    }

    if (availableWeapons.Count() == 0) {
        // No weapons available
    }
```

VS

```cpp
void Update() {
    if (enemies.Any(e => e.health > 50)) {
        // Strong enemies exist - stops at first match!
    }

    if (!availableWeapons.Any()) {
        // No weapons available - O(1) for lists
    }
}
```

MH 2: First() with Exception vs. FirstOrDefault()

```cpp
void FindTarget() {
    try {
        var target = enemies.Where(e => e.isVisible).First();
        AttackTarget(target);
    }
    catch (InvalidOperationException) {
        // No visible enemies - exception overhead!
    }
}
```

VS

```cpp
void FindTarget() {
    var target = enemies.FirstOrDefault(e => e.isVisible);
    if (target != null) {
        AttackTarget(target);
    }
    // No exception overhead!
}
```

MH 3: Multiple LINQ Chains vs. Single Pass

```cpp
void ProcessEnemies() {
    var activeEnemies = enemies.Where(e => e.isActive);
    var visibleEnemies = activeEnemies.Where(e => e.isVisible);
    var damagedEnemies = visibleEnemies.Where(e => e.health < 50);
    var sortedEnemies = damagedEnemies.OrderBy(e => e.health).ToList();
}
```

VS

```cpp
void ProcessEnemies() {
    var sortedEnemies = enemies
        .Where(e => e.isActive && e.isVisible && e.health < 50)
        .OrderBy(e => e.health)
        .ToList();
}
```

MH 4: Select().ToArray() vs. Pre-sized Array

```cpp
void ExtractHealthValues() {
    float[] healthValues = enemies
        .Where(e => e.isActive)
        .Select(e => e.health)
        .ToArray(); // Multiple resize allocations!
}
```

VS

```cpp
void ExtractHealthValues() {
    int count = 0;
    float[] healthValues = new float[enemies.Count];

    for (int i = 0; i < enemies.Count; i++) {
        if (enemies[i].isActive) {
            healthValues[count++] = enemies[i].health;
        }
    }

    if (count < healthValues.Length) {
        Array.Resize(ref healthValues, count); // Single resize if neede
    }
}
```