Guida introduttiva a Unity per giochi Platform 2D

Obiettivo della guida

Questa guida introduce le basi dell'editor di Unity e i concetti fondamentali per creare un semplice videogioco platform 2D. L'obiettivo non è imparare tutta Unity, ma acquisire un metodo di lavoro ordinato e pratico attraverso piccole task progressive.

Alla fine del percorso saprai:

- orientarti nell'editor di Unity;
- creare e configurare GameObject;
- usare Inspector, Hierarchy, Scene e Game View;
- aggiungere e configurare componenti;
- creare prefab riutilizzabili;
- organizzare correttamente le cartelle di progetto;
- creare, assegnare e modificare script C#;
- realizzare movimento, raccolta monete, camera follow e animazioni base.

---

1. L'editor di Unity

1.1 Le finestre principali

Quando apri Unity, l'interfaccia è composta da alcune finestre fondamentali.

Scene View

È lo spazio in cui costruisci il livello. Qui puoi spostare oggetti, modificarne le dimensioni, ruotarli e posizionarli.

Comandi utili:

- tasto destro + movimento mouse: orientarsi nella scena;
- rotellina mouse: zoom;
- tasto `W`: sposta oggetto;
- tasto `E`: ruota oggetto;
- tasto `R`: scala oggetto;
- tasto `F`: centra la vista sull'oggetto selezionato.

Game View

Mostra ciò che vede il giocatore quando il gioco è in esecuzione. È la visuale finale della camera.

Hierarchy

Contiene tutti gli oggetti presenti nella scena corrente. Ogni elemento della Hierarchy è un GameObject.

Esempi:

- Player;
- Ground;
- Coin;
- Enemy;
- Main Camera;
- Canvas.

Inspector

Mostra le proprietà dell'oggetto selezionato. Da qui puoi modificare posizione, componenti, script, collider, rigidbody e parametri pubblici.

Project Window

Mostra i file del progetto: script, sprite, scene, prefab, audio, materiali e animazioni.

Console

Mostra errori, avvisi e messaggi generati dagli script.

Task 1 — Esplora l'editor

1. Crea un nuovo progetto 2D.
2. Crea una nuova scena chiamata `MainScene`.
3. Aggiungi un oggetto 2D Square e rinominalo `Player`.
4. Osserva come cambia la Hierarchy.
5. Seleziona `Player` e modifica posizione e scala dall'Inspector.
6. Premi Play e osserva la Game View.

Obiettivo: capire la differenza tra Scene View, Game View, Hierarchy e Inspector.

---

2. GameObject e Componenti

2.1 Che cos’è un GameObject

Un GameObject è un contenitore vuoto a cui puoi aggiungere componenti.

Da solo, un GameObject non fa quasi nulla. Diventa utile grazie ai componenti.

Esempio: un Player può avere:

- `Transform`;
- `SpriteRenderer`;
- `Rigidbody2D`;
- `BoxCollider2D`;
- `PlayerController`.

2.2 Il Transform

Ogni GameObject ha sempre un componente `Transform`. Gestisce:

- posizione;
- rotazione;
- scala.

Nei giochi 2D userai soprattutto X e Y.

2.3 SpriteRenderer

Serve a mostrare un’immagine 2D. Senza SpriteRenderer, un oggetto può esistere nella scena ma non essere visibile.

2.4 Rigidbody2D

Serve a far interagire un oggetto con la fisica 2D.

Parametri importanti:

- `Gravity Scale`: quantità di gravità applicata;
- `Mass`: massa dell’oggetto;
- `Freeze Rotation Z`: blocca la rotazione, utile per il Player.

2.5 Collider2D

Serve a definire la forma fisica dell'oggetto per collisioni e trigger.

Tipi comuni:

- `BoxCollider2D`;
- `CircleCollider2D`;
- `CapsuleCollider2D`;
- `PolygonCollider2D`.

2.6 Trigger e Collisioni

Un Collider può funzionare in due modi:

- collisione fisica: blocca gli oggetti;
- trigger: rileva il contatto ma non blocca.

Esempi:

- pavimento: collisione normale;
- moneta: trigger;
- porta di fine livello: trigger.

Task 2 — Crea un pavimento e testa la fisica

1. Crea un oggetto Square e rinominalo `Ground`.
2. Scala `Ground` in larghezza.
3. Aggiungi `BoxCollider2D` a `Ground`.
4. Seleziona `Player`.
5. Aggiungi `Rigidbody2D`.
6. Aggiungi `BoxCollider2D`.
7. Imposta `Freeze Rotation Z` sul Player.
8. Premi Play.

Obiettivo: il Player deve cadere e fermarsi sopra il pavimento.

---

3. Organizzazione delle cartelle

Un progetto ordinato è più facile da capire e mantenere.

Crea nella cartella `Assets` questa struttura:

```
None
Assets/
  Scenes/
  Scripts/
  Sprites/
  Prefabs/
  Animations/
  Audio/
  Materials/
  UI/
```

A cosa servono

- `Scenes`: contiene le scene del gioco;
- `Scripts`: contiene gli script C#;
- `Sprites`: contiene immagini e sprite;
- `Prefabs`: contiene oggetti riutilizzabili;
- `Animations`: contiene clip e controller animazioni;
- `Audio`: contiene suoni e musiche;
- `Materials`: contiene materiali;
- `UI`: contiene elementi grafici dell'interfaccia.

Task 3 — Organizza il progetto

1. Crea tutte le cartelle sopra indicate.
2. Sposta `MainScene` dentro `Scenes`.
3. Crea uno script vuoto chiamato `TestScript` dentro `Scripts`.
4. Elimina lo script solo dopo aver verificato che la cartella funzioni correttamente.

Obiettivo: prendere l'abitudine di non lasciare file sparsi in `Assets`.

---

4. Prefab

4.1 Che cos’è un prefab

Un prefab è un GameObject salvato come modello riutilizzabile.

Esempi tipici:

- Coin;
- Enemy;
- Platform;
- Bullet;
- Checkpoint.

Se modifichi il prefab originale, puoi aggiornare tutte le sue copie.

4.2 Creare un prefab

1. Crea un GameObject nella scena.
2. Configuralo con componenti e proprietà.
3. Trascinalo dalla Hierarchy alla cartella `Prefabs`.
4. L’oggetto diventa un prefab.

Task 4 — Crea il prefab Coin

1. Crea un oggetto 2D Circle.
2. Rinominalo `Coin`.
3. Cambia colore in giallo.
4. Aggiungi `CircleCollider2D`.
5. Spunta `Is Trigger`.
6. Trascina `Coin` nella cartella `Prefabs`.
7. Crea 5 copie del prefab nella scena.

Obiettivo: imparare a creare oggetti riutilizzabili.

---

5. Script in Unity

5.1 Creare uno script

1. Vai nella cartella `Scripts`.
2. Tasto destro → Create → C# Script.
3. Dai un nome allo script.
4. Aprilo con doppio click.

Attenzione: il nome del file deve coincidere con il nome della classe.

Esempio:

```CSharp
public class PlayerController : MonoBehaviour
{
}
```

Il file deve chiamarsi:

```None
PlayerController.cs
```

5.2 Assegnare uno script a un GameObject

Puoi farlo in due modi:

1. trascinare lo script sull'oggetto nella Hierarchy;
2. selezionare l'oggetto → Inspector → Add Component → cercare lo script.

5.3 MonoBehaviour

Quasi tutti gli script Unity ereditano da `MonoBehaviour`.

Metodi base:

```CSharp
void Awake()
{
    // chiamato quando l'oggetto viene inizializzato
}
```

```
void Start()
{
    // chiamato prima del primo frame
}

void Update()
{
    // chiamato a ogni frame
}

void FixedUpdate()
{
    // chiamato a intervalli fissi, utile per la fisica
}
```

Task 5 — Primo script

1. Crea uno script chiamato `HelloUnity`.
2. Assegnalo al Player.
3. Scrivi:

```CSharp
using UnityEngine;

public class HelloUnity : MonoBehaviour
{
    void Start()
    {
        Debug.Log("Ciao Unity!");
    }
}
```

4. Premi Play.
5. Controlla la Console.

Obiettivo: capire come Unity esegue uno script assegnato a un GameObject.

---

6. Movimento base del Player

6.1 Concetto

Per un platform 2D, il Player deve:

- muoversi a destra e sinistra;
- saltare;
- essere influenzato dalla gravità;
- non ruotare quando collide.

Componenti richiesti:

- `Rigidbody2D`;
- `BoxCollider2D`;
- script `PlayerController`.

6.2 Script PlayerController

Crea `PlayerController.cs` nella cartella `Scripts`.

CSharp

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class PlayerController : MonoBehaviour
{
    public float moveSpeed = 6f;
    public float jumpForce = 12f;

    private Rigidbody2D rb;
    private bool isGrounded;

    void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        float horizontalInput = Input.GetAxisRaw("Horizontal");
```

```
        rb.velocity = new Vector2(horizontalInput * moveSpeed,
rb.velocity.y);

        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            isGrounded = false;
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Ground"))
        {
            isGrounded = true;
        }
    }
}
```

6.3 Configurazione

1. Assegna il tag `Ground` al pavimento.
2. Assegna lo script al Player.
3. Verifica che il Player abbia `Rigidbody2D` e `BoxCollider2D`.
4. Premi Play.

Task 6 — Movimento del Player

1. Muovi il Player con freccia destra/sinistra o A/D.
2. Fai saltare il Player con Spazio.
3. Modifica `moveSpeed` dall'Inspector.
4. Modifica `jumpForce` dall'Inspector.

Obiettivo: capire il collegamento tra variabili pubbliche e Inspector.

---

7. Raccolta monete

7.1 Concetto

Le monete non devono bloccare il Player. Devono solo rilevare il contatto e scomparire.

Per questo usiamo:

- `Collider2D` con `Is Trigger`;
- `OnTriggerEnter2D`;
- `Destroy`.

7.2 Script Coin

Crea `Coin.cs`.

```CSharp
using UnityEngine;

public class Coin : MonoBehaviour
{
    public int value = 1;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("Moneta raccolta! Valore: " + value);
            Destroy(gameObject);
        }
    }
}
```

7.3 Configurazione

1. Assegna tag `Player` al Player.
2. Assegna `Coin.cs` al prefab Coin.
3. Verifica che il collider della Coin abbia `Is Trigger` attivo.
4. Premi Play.

Task 7 — Raccogli le monete

1. Posiziona 5 monete nel livello.
2. Raccoglile tutte.
3. Controlla la Console.
4. Cambia `value` di alcune monete dall'Inspector.

Obiettivo: capire trigger, tag e distruzione degli oggetti.

---

8. GameManager e punteggio

8.1 Concetto

Il GameManager è un oggetto che controlla informazioni generali del gioco, come punteggio, vite o stato della partita.

8.2 UI del punteggio

1. Crea un Canvas.
2. Crea un Text Legacy o TextMeshPro.
3. Rinominalo `ScoreText`.
4. Scrivi inizialmente `Score: 0`.

8.3 Script GameManager

Crea `GameManager.cs`.

CSharp

```
using UnityEngine;
using UnityEngine.UI;

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

    public Text scoreText;
    private int score;

    void Awake()
    {
        if (Instance != null && Instance != this)
```

```
        {
            Destroy(gameObject);
            return;
        }

        Instance = this;
    }

    void Start()
    {
        UpdateScoreUI();
    }

    public void AddScore(int amount)
    {
        score += amount;
        UpdateScoreUI();
    }

    private void UpdateScoreUI()
    {
        if (scoreText != null)
        {
            scoreText.text = "Score: " + score;
        }
    }
}
```

8.4 Aggiornare Coin.cs

Sostituisci il codice della moneta con:

CSharp

```
using UnityEngine;

public class Coin : MonoBehaviour
{
    public int value = 1;
```

```
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            GameManager.Instance.AddScore(value);
            Destroy(gameObject);
        }
    }
}
```

Task 8 — Punteggio

1. Crea un Empty Object chiamato `GameManager`.
2. Assegna lo script `GameManager`.
3. Trascina `ScoreText` nel campo pubblico dello script.
4. Premi Play.
5. Raccogli monete e verifica che il punteggio aumenti.

Obiettivo: capire il ruolo di un manager centrale e il collegamento tra script e UI.

---

9. Camera follow

9.1 Concetto

Nei platform 2D la camera spesso segue il Player lungo l'asse X e/o Y.

9.2 Script CameraFollow

Crea `CameraFollow.cs`.

```CSharp
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public Transform target;
    public float smoothSpeed = 5f;
```

```
    public Vector3 offset = new Vector3(0f, 1f, -10f);

    void LateUpdate()
    {
        if (target == null) return;

        Vector3 desiredPosition = target.position + offset;
        transform.position = Vector3.Lerp(transform.position,
desiredPosition, smoothSpeed * Time.deltaTime);
    }
}
```

9.3 Configurazione

1. Seleziona Main Camera.
2. Assegna lo script `CameraFollow`.
3. Trascina il Player nel campo `target`.
4. Imposta `offset` a `(0, 1, -10)`.

Task 9 — Camera dinamica

1. Premi Play.
2. Muovi il Player.
3. Osserva la camera che segue.
4. Cambia `smoothSpeed` dall'Inspector.
5. Cambia l'offset.

Obiettivo: capire `Transform`, riferimenti pubblici e `LateUpdate`.

---

10. Animazioni base

10.1 Concetto

Per animare un Player in Unity servono:

- sprite o immagini;
- Animation Clip;
- Animator Controller;
- componente Animator;

- parametri dell’Animator;
- script che aggiorna i parametri.

10.2 Animator

L’Animator Controller è una macchina a stati.

Esempi di stati:

- Idle;
- Run;
- Jump;
- Fall.

Esempi di parametri:

- `Speed`;
- `IsGrounded`;
- `VerticalVelocity`.

10.3 Creare animazioni

1. Crea cartella `Animations`.
2. Seleziona il Player.
3. Apri Window → Animation → Animation.
4. Crea una clip `Player_Idle`.
5. Crea una clip `Player_Run`.
6. Unity creerà anche un Animator Controller.
7. Verifica che il Player abbia il componente Animator.

10.4 Collegare animazioni allo script

Aggiungi al `PlayerController` una variabile:

```CSharp
private Animator animator;
```

In `Awake()`:

```CSharp
animator = GetComponent<Animator>();
```

In `Update()`, dopo aver letto l’input:

CSharp

```
if (animator != null)
{
    animator.SetFloat("Speed", Mathf.Abs(horizontalInput));
    animator.SetBool("IsGrounded", isGrounded);
    animator.SetFloat("VerticalVelocity", rb.velocity.y);
}
```

Versione aggiornata del PlayerController:

CSharp

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class PlayerController : MonoBehaviour
{
    public float moveSpeed = 6f;
    public float jumpForce = 12f;

    private Rigidbody2D rb;
    private Animator animator;
    private bool isGrounded;

    void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        float horizontalInput = Input.GetAxisRaw("Horizontal");

        rb.velocity = new Vector2(horizontalInput * moveSpeed,
rb.velocity.y);

        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            isGrounded = false;
```

```
        }

        if (animator != null)
        {
            animator.SetFloat("Speed",
Mathf.Abs(horizontalInput));
            animator.SetBool("IsGrounded", isGrounded);
            animator.SetFloat("VerticalVelocity", rb.velocity.y);
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Ground"))
        {
            isGrounded = true;
        }
    }
}
```

Task 10 — Animator base

1. Crea due stati: `Idle` e `Run`.
2. Crea un parametro Float chiamato `Speed`.
3. Crea una transizione Idle → Run quando `Speed > 0.1`.
4. Crea una transizione Run → Idle quando `Speed < 0.1`.
5. Premi Play.
6. Muovi il Player.

Obiettivo: capire il collegamento tra input, script e Animator.

---

11. Best practice iniziali

11.1 Nomi chiari

Usa nomi leggibili:

- `PlayerController`;
- `Coin`;
- `CameraFollow`;
- `GameManager`;
- `EnemyPatrol`.

Evita nomi generici come:

- `Script1`;
- `ObjectNew`;
- `provafinale2`.

11.2 Una responsabilità per script

Ogni script dovrebbe fare una cosa principale.

Esempi:

- `PlayerController`: movimento del player;
- `Coin`: comportamento della moneta;
- `GameManager`: punteggio e stato del gioco;
- `CameraFollow`: movimento della camera.

11.3 Usa i prefab

Se un oggetto deve comparire più volte, trasformalo in prefab.

Esempi:

- monete;
- piattaforme;
- nemici;
- oggetti raccoglibili.

11.4 Controlla sempre la Console

Molti errori Unity si capiscono dalla Console. Leggi sempre:

- nome dello script;
- riga dell'errore;
- tipo di errore.

---

12. Progetto finale consigliato

Realizza un mini livello platform con:

- Player che corre e salta;
- pavimento e piattaforme;
- almeno 10 monete;
- punteggio visibile;
- camera che segue il Player;
- animazioni Idle e Run;
- una porta di fine livello opzionale.

Task finale

1. Crea una scena `Level_01`.
2. Organizza tutti i file nelle cartelle corrette.
3. Crea il Player con movimento e salto.
4. Crea almeno 3 piattaforme.
5. Crea un prefab Coin.
6. Posiziona 10 monete.
7. Crea UI punteggio.
8. Aggiungi camera follow.
9. Aggiungi animazioni Idle e Run.
10. Testa il gioco e correggi eventuali errori.

Obiettivo: completare un piccolo prototipo platform 2D funzionante.

---

13. Checklist di controllo

Prima di considerare il progetto completato, verifica:

- Il Player ha `Rigidbody2D`.
- Il Player ha `BoxCollider2D`.
- Il Player ha tag `Player`.
- Il Ground ha tag `Ground`.
- Le monete hanno `Is Trigger` attivo.
- Il prefab Coin contiene lo script `Coin`.

- Il GameManager è presente in scena.
- Il campo `scoreText` è assegnato.
- La Main Camera ha `CameraFollow`.
- Il campo `target` della camera è assegnato al Player.
- L'Animator ha il parametro `Speed`.
- Le transizioni Idle/Run funzionano.

---

14. Errori comuni

Il Player cade all'infinito

Possibili cause:

- il Ground non ha Collider2D;
- il Player non ha Collider2D;
- il Player non ha Rigidbody2D;
- il Ground è troppo lontano.

La moneta non viene raccolta

Possibili cause:

- la moneta non ha `Is Trigger` attivo;
- il Player non ha tag `Player`;
- lo script `Coin` non è assegnato;
- manca un Collider2D su Player o Coin.

Il punteggio non cambia

Possibili cause:

- manca il GameManager in scena;
- `ScoreText` non è collegato nell'Inspector;
- lo script Coin non chiama `GameManager.Instance.AddScore`;
- ci sono errori in Console.

La camera non segue il Player

Possibili cause:

- lo script `CameraFollow` non è sulla Main Camera;

- il campo `target` è vuoto;
- l’offset Z non è corretto;
- la camera è troppo vicina o troppo lontana.

L’animazione non cambia

Possibili cause:

- il parametro `Speed` non esiste o è scritto diversamente;
- le transizioni sono configurate male;
- il Player non ha Animator;
- lo script non aggiorna i parametri dell’Animator.

---

15. Prossimi passi

Dopo aver completato questa guida, puoi aggiungere:

- nemici con movimento pattuglia;
- vite del Player;
- checkpoint;
- menu iniziale;
- suoni;
- particelle quando raccogli una moneta;
- salti più precisi;
- livelli multipli;
- schermata di vittoria e game over.