Scheda 1

PARTE 1 — INSPECTOR ED EDITOR UNITY

---

1. Preparare le cartelle del progetto

Obiettivo

Tenere separati sprite, animazioni, controller e script.

Cartelle consigliate

Dentro `Assets`, crea questa struttura:

Assets/
Sprites/
Player/
Environment/
UI/
Obstacles/
Animations/
Player/
Obstacles/
AnimatorControllers/
Scripts/
Prefabs/
Scenes/

Spiegazione

- `Sprites/Player`: immagini del personaggio.
- `Sprites/Environment`: sfondi, terreno, piattaforme.
- `Sprites/UI`: icone, bottoni, elementi interfaccia.
- `Animations/Player`: clip di animazione del player.
- `AnimatorControllers`: controller che gestiscono gli stati animati.
- `Scripts`: codice C#.
- `Prefabs`: oggetti riutilizzabili.

Task 1

Crea le cartelle indicate e sposta ogni file nella posizione corretta.

Controllo finale

Il progetto deve essere ordinato e non deve avere sprite o script sparsi direttamente dentro `Assets`.

---

2. Importare gli sprite

Obiettivo

Inserire immagini 2D nel progetto Unity.

Passaggi

1. Trascina i file immagine dentro la cartella corretta, per esempio `Sprites/Player`.
2. Seleziona lo sprite importato.
3. Guarda l'Inspector.
4. Imposta `Texture Type` su `Sprite (2D and UI)`.
5. Imposta `Sprite Mode`:
    - `Single`, se l'immagine contiene un solo sprite;
    - `Multiple`, se l'immagine contiene più frame in un unico spritesheet.
6. Premi `Apply`.

Parametri importanti nell'Inspector

Texture Type

Deve essere `Sprite (2D and UI)` per usare l'immagine come sprite 2D.

Sprite Mode

- `Single`: una sola immagine.
- `Multiple`: più immagini dentro lo stesso file.

Pixels Per Unit

Indica quanti pixel corrispondono a una unità Unity.

Esempio:

- `100`: valore standard comodo.
- valore più alto: sprite più piccolo nella scena.
- valore più basso: sprite più grande nella scena.

Filter Mode

- `Point`: utile per pixel art.
- `Bilinear`: più morbido, utile per immagini non pixel art.

Compression

Per sprite semplici o pixel art, può essere utile impostare `None` per evitare artefatti visivi.

Task 2

Importa uno sprite del Player e configurarlo come `Sprite (2D and UI)`.

Controllo finale

Lo sprite deve poter essere trascinato nella scena e visualizzato correttamente.

---

3. Usare uno sprite nella scena

Obiettivo

Sostituire forme provvisorie, come cerchi o quadrati, con immagini vere.

Passaggi

1. Seleziona il GameObject `Player`.
2. Nell'Inspector trova il componente `SpriteRenderer`.
3. Nel campo `Sprite`, trascina lo sprite del Player.
4. Regola la scala del GameObject se necessario.

SpriteRenderer

Il componente `SpriteRenderer` serve a mostrare uno sprite nella scena.

Parametri utili:

- `Sprite`: immagine da visualizzare.
- `Color`: colore o tinta applicata allo sprite.
- `Flip X`: ribalta orizzontalmente lo sprite.
- `Flip Y`: ribalta verticalmente lo sprite.
- `Sorting Layer`: livello grafico.
- `Order in Layer`: ordine nello stesso livello.

Task 3

Assegna uno sprite reale al Player usando il componente `SpriteRenderer`.

Controllo finale

Il Player non deve più essere una forma base, ma deve mostrare lo sprite scelto.

---

4. Sorting Layer e ordine degli sprite

Obiettivo

Capire come decidere quale sprite appare davanti o dietro.

Concetto

In un gioco 2D, molti oggetti stanno sullo stesso piano. Per decidere cosa appare davanti, Unity usa:

- `Sorting Layer`;
- `Order in Layer`.

Esempio di Sorting Layer

Crea questi layer:

Background
Environment
Obstacles
Player
UI

Impostazioni consigliate

- Sfondo: `Background`, Order `0`.
- Terreno: `Environment`, Order `0`.
- Tubi / ostacoli: `Obstacles`, Order `0`.
- Player: `Player`, Order `0`.
- UI: gestita dal Canvas.

Task 4

Crea i Sorting Layer e assegna il Player al layer `Player`.

Controllo finale

Il Player deve apparire davanti allo sfondo e agli elementi ambientali.

---

5. Sprite singoli e spritesheet

Obiettivo

Capire la differenza tra uno sprite singolo e uno spritesheet animato.

Sprite singolo

È un file che contiene una sola immagine.

Esempio:

bird_idle.png

Spritesheet

È un file che contiene più frame della stessa animazione.

Esempio:

bird_flying_sheet.png

Per usare uno spritesheet:

1. Seleziona il file.
2. Imposta `Sprite Mode` su `Multiple`.
3. Premi `Sprite Editor`.
4. Usa `Slice`.
5. Scegli `Grid By Cell Size` oppure `Automatic`.
6. Premi `Apply`.

Task 5

Importa uno spritesheet, impostalo su `Multiple` e separa i frame con Sprite Editor.

Controllo finale

Nel Project devi vedere i singoli frame espandendo lo spritesheet.

---

6. Creare una Animation Clip

Obiettivo

Creare un’animazione a partire da più sprite.

Metodo rapido

1. Seleziona i frame dell'animazione nel Project.
2. Trascinali nella scena o sul Player.
3. Unity chiederà di salvare una nuova Animation Clip.
4. Salvala in `Animations/Player`.
5. Dai un nome chiaro, per esempio:
    - `Player_Idle`;
    - `Player_Flap`;
    - `Player_Fall`;
    - `Player_Run`.

Metodo tramite finestra Animation

1. Seleziona il Player.
2. Apri `Window > Animation > Animation`.
3. Clicca `Create`.
4. Salva la clip in `Animations/Player`.
5. Aggiungi i frame sulla timeline.
6. Regola la velocità dell'animazione.

Task 6

Crea una clip chiamata `Player_Flap` usando almeno 2 o 3 frame.

Controllo finale

Premendo Play, l'animazione deve essere riproducibile sul Player.

---

7. Animator Controller

Obiettivo

Gestire più animazioni dello stesso oggetto.

Cos'è l'Animator Controller

L'Animator Controller è una macchina a stati.

Contiene stati come:

- `Idle`;
- `Run`;
- `Jump`;

- `Fall`;
- `Flap`;
- `Dead`.

Ogni stato può riprodurre una Animation Clip.

Creazione

1. Vai nella cartella `AnimatorControllers`.
2. Tasto destro > `Create > Animator Controller`.
3. Chiamalo `PlayerAnimatorController`.
4. Seleziona il Player.
5. Aggiungi componente `Animator`, se non presente.
6. Trascina `PlayerAnimatorController` nel campo `Controller` dell'Animator.

Task 7

Crea un Animator Controller e assegnalo al Player.

Controllo finale

Il Player deve avere un componente `Animator` con il controller assegnato.

---

8. Stati e transizioni nell'Animator

Obiettivo

Far cambiare animazione in base allo stato del Player.

Passaggi

1. Apri la finestra `Animator`.
2. Trascina dentro le clip create.
3. Crea stati, per esempio:
    - `Idle`;
    - `Flap`;
    - `Fall`;
    - `Dead`.
4. Crea transizioni tra gli stati.

Parametri consigliati

Nel tab `Parameters`, crea:

Speed Float
VerticalVelocity Float
IsGrounded Bool
IsAlive Bool
Flap Trigger

Per Flappy Bird possono bastare:

VerticalVelocity Float
IsAlive Bool
Flap Trigger

Esempio per Flappy Bird

- Da `Idle` a `Flap`: parametro Trigger `Flap`.
- Da `Flap` a `Fall`: quando `VerticalVelocity < 0`.
- Da qualsiasi stato a `Dead`: quando `IsAlive == false`.

Task 8

Crea almeno tre stati: `Idle`, `Flap`, `Fall`.

Controllo finale

Gli stati devono essere visibili nella finestra Animator e collegati da transizioni.

---

9. Impostazioni delle transizioni

Obiettivo

Rendere le animazioni reattive.

Parametri importanti

Has Exit Time

Se attivo, Unity aspetta la fine dell'animazione prima di cambiare stato.

Per animazioni reattive, come `Flap`, spesso conviene disattivarlo.

Transition Duration

Indica quanto tempo dura il passaggio da un'animazione all'altra.

Per pixel art o animazioni rapide, puoi usare valori bassi, per esempio `0` o `0.05`.

Conditions

Sono le condizioni che fanno partire una transizione.

Esempio:

VerticalVelocity Less 0

Task 9

Disattiva `Has Exit Time` nelle transizioni più importanti e imposta una durata breve.

Controllo finale

Le animazioni devono cambiare rapidamente quando il Player vola o cade.

---

10. Animazioni per oggetti diversi dal Player

Obiettivo

Capire che anche ostacoli, monete, sfondi e UI possono essere animati.

Esempi

Moneta

Stati possibili:

- `Idle`;
- `Collected`.

Ostacolo

Stati possibili:

- `Move`;
- `Hit`.

UI

Stati possibili:

- `Hidden`;
- `ShowGameOver`.

Task 10

Crea una semplice animazione per una moneta o un elemento UI.

Controllo finale

Almeno un oggetto diverso dal Player deve avere una Animation Clip.

Scheda 2

PARTE 2 — SCRIPT PER COLLEGARE SPRITE E ANIMAZIONI

---

11. Collegare Animator allo script

Obiettivo

Permettere allo script di comunicare con l'Animator.

Concetto

Lo script non cambia direttamente i frame dell'animazione. Lo script aggiorna i parametri dell'Animator.

L'Animator, in base ai parametri, sceglie quale animazione riprodurre.

Codice base

private Animator animator;

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

Spiegazione

private Animator animator;

Crea una variabile privata per memorizzare il componente Animator.

animator = GetComponent<Animator>();

Recupera l'Animator presente sullo stesso GameObject.

Task 11

Aggiungi una variabile `Animator` allo script del Player e recuperala in `Awake()`.

Controllo finale

Lo script deve compilare senza errori e il Player deve avere un componente Animator.

---

12. Aggiornare un parametro Float

Obiettivo

Far capire all'Animator se il Player sta salendo o scendendo.

Codice

```
animator.SetFloat("VerticalVelocity", rb.velocity.y);
```

Spiegazione

- `SetFloat` aggiorna un parametro numerico dell'Animator.
- `VerticalVelocity` deve esistere anche nell'Animator.
- `rb.velocity.y` è la velocità verticale del Rigidbody2D.

Uso

Nel gioco tipo Flappy Bird:

- se `VerticalVelocity > 0`, il Player sta salendo;
- se `VerticalVelocity < 0`, il Player sta cadendo.

Task 12

Crea nell'Animator un parametro Float chiamato `VerticalVelocity` e aggiornalo via script.

Controllo finale

Durante il gioco, il valore deve cambiare quando il Player sale o cade.

---

13. Aggiornare un Trigger

Obiettivo

Far partire un'animazione breve, come il battito d'ali.

Codice

```
animator.SetTrigger("Flap");
```

Spiegazione

Un Trigger è un parametro che si attiva una volta e poi si resetta.

È utile per azioni istantanee:

- salto;
- attacco;
- flap;
- raccolta oggetto;
- danno subito.

Dove inserirlo

PROVA TU :) -> **Task 13**

Controllo finale

Ogni volta che premi Spazio o clicchi, deve partire l'animazione di flap.

---

14. Aggiornare un Bool

Obiettivo

Far capire all'Animator se il Player è vivo o morto.

Codice

animator.SetBool("IsAlive", isAlive);

Spiegazione

Un Bool può essere solo:

- `true`;
- `false`.

È utile per stati persistenti.

Esempi:

- vivo/morto;
- a terra/in aria;
- attivo/disattivo;
- in pausa/non in pausa.

Dove inserirlo

PROVA TU -> Task 14

Controllo finale

Quando il Player collide con un ostacolo, deve partire l’animazione di morte.

Task 15

Completa lo script PlayerFlap e verifica che i parametri siano presenti nell’Animator.

Controllo finale

Il Player deve:

- fare flap quando premi Spazio;
- passare a Fall mentre cade;
- passare a Dead quando collide.

---

16. Script per girare lo sprite in base alla direzione

Obiettivo

Nei platform, far guardare il Player verso destra o sinistra.

Questo è più utile nei platform che in Flappy Bird, ma è importante conoscerlo.

Codice

```
private SpriteRenderer spriteRenderer;

void Awake()
{
spriteRenderer = GetComponent<SpriteRenderer>();
}

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

if (horizontalInput > 0)
{
spriteRenderer.flipX = false;
}
```

```
else if (horizontalInput < 0)
{
spriteRenderer.flipX = true;
}
}
```

Spiegazione

`SpriteRenderer.flipX` ribalta lo sprite sull'asse X.

È utile quando hai una sola animazione di camminata verso destra e vuoi riusarla anche verso sinistra.

Task 16

Integra `SpriteRenderer.flipX` in uno script platform e verifica il cambio direzione.

Controllo finale

Il personaggio deve guardare nella direzione in cui si muove.

---

17. Script PlayerController platform con Animator

Obiettivo

Mostrare come integrare gli stessi concetti in un platform 2D.

Codice

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(Animator))]
[RequireComponent(typeof(SpriteRenderer))]
public class PlayerController2D : MonoBehaviour
{
public float moveSpeed = 6f;
public float jumpForce = 12f;

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

void Awake()
{
```

rb = GetComponent<Rigidbody2D>();
animator = GetComponent<Animator>();
spriteRenderer = GetComponent<SpriteRenderer>();
}

void Update()
{
PROVA TU -> TASK FINALE
}

Parametri Animator necessari

Speed Float
IsGrounded Bool
VerticalVelocity Float

Stati consigliati

Idle
Run
Jump
Fall

Task 17

Crea le animazioni `Idle`, `Run`, `Jump` e `Fall`, poi collegale ai parametri dello script.

Controllo finale

Il personaggio deve cambiare animazione mentre si muove, salta e cade.

BONUS. Animazione raccolta moneta

Obiettivo

Far partire un'animazione prima di distruggere una moneta.

Concetto

Se distruggiamo subito la moneta, non vedremo l'animazione di raccolta.

Serve quindi:

1. disattivare il collider;
2. far partire un trigger di animazione;
3. distruggere l'oggetto dopo un breve tempo.

Script CoinAnimated

```
using UnityEngine;

[RequireComponent(typeof(Collider2D))]
[RequireComponent(typeof(Animator))]
public class CoinAnimated : MonoBehaviour
{
public int value = 1;
public float destroyDelay = 0.3f;

private Animator animator;
private Collider2D coinCollider;
private bool collected;

void Awake()
{
animator = GetComponent<Animator>();
coinCollider = GetComponent<Collider2D>();
}

void OnTriggerEnter2D(Collider2D other)
{
if (collected) return;

if (other.CompareTag("Player"))
{
collected = true;
coinCollider.enabled = false;
GameManager.Instance.AddScore(value);
animator.SetTrigger("Collected");
Destroy(gameObject, destroyDelay);
}
}
}
```

Parametro Animator necessario

Collected Trigger

Stati consigliati

Idle
Collected

Task 18

Crea un’animazione di raccolta per la moneta e collegala con il Trigger `Collected`.

Controllo finale

La moneta deve animarsi prima di sparire.

BONUS. Animazioni UI: Game Over

Obiettivo

Far comparire il testo Game Over con un’animazione.

Metodo semplice

1. Seleziona `GameOverText`.
2. Aggiungi `Animator`.
3. Crea una clip `GameOver_Show`.
4. Anima scala o trasparenza.
5. Crea un parametro Trigger `Show`.

Script GameManager con Animator UI

```
using UnityEngine;
using UnityEngine.UI;

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

public Text scoreText;
public Animator gameOverAnimator;

private int score;
private bool isGameOver;

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

Instance = this;
}

void Start()
```

```
{
score = 0;
UpdateScoreText();
}

public void AddScore(int amount = 1)
{
if (isGameOver) return;

score += amount;
UpdateScoreText();
}

void UpdateScoreText()
{
if (scoreText != null)
{
scoreText.text = score.ToString();
}
}

public void GameOver()
{
if (isGameOver) return;

isGameOver = true;

if (gameOverAnimator != null)
{
gameOverAnimator.SetTrigger("Show");
}
}
}
```

Task 19

Collega l'Animator del GameOverText al GameManager e attiva il trigger `Show`.

Controllo finale

Quando il Player perde, la UI Game Over deve comparire con animazione.