Manual – Unity 2D Platformer Drone Game

Example Game: Example Game.mp4

This example demonstrates a **Top-Down 2D Arcade Racer** with a multi-lane layout, moving obstacles (cars and helicopters), and a UI system tracking lives and score.

Below is your step-by-step manual.

Step 1: Project Setup & Asset Gathering

To begin, you must install **Unity Hub** and the latest version of **Unity** (e.g., 2022.3.5f1) along with **Visual Studio** for scripting. Create a new **2D Core** project.

The Assets: Avoid spending excessive time on custom art; use these specific "Top-Down" aerial packs:

- **Drones & Aerial Vehicles:** Gathering 2D drone sprites and rotor assets from online stores or provided packages.

  Some resources:
  - https://free-game-assets.itch.io/free-drones-pack-pixel-art
  - https://alianzagames.itch.io/drone-pack-demo
  - https://assetstore.unity.com/2d/characters?srsltid=AfmBOorWXQnHr4qFt17fibWhmT40yNi0_XrYSXElwrGrAd-KzN-E1Aln
- **Environment:** Design your aerial "maze" with landing pads and hazards using imported sceneries and props.

  Some resources:
  - **https://kenney.nl/assets/tiny-dungeon**
  - **https://gamedeveloperstudio.itch.io/top-down-environmental-asset-pack**
  - **https://craftpix.net/freebies/**
  - **https://itch.io/game-assets/free/tag-top-down**

- **Sound:** Import **SFX and music** assets to provide feedback for actions like humming engines or collisions.

  Some resources:
  - https://mixkit.co/free-sound-effects/drone/
  - https://www.zapsplat.com/sound-effect-category/drone/
  - https://elevenlabs.io/sound-effects/drone

Step 2: Player Movement (The "Pilot")

Your drone needs smooth movement that simulates hovering and momentum rather than rigid stops.

1. **Create Player:** Drag a drone sprite into the scene to create a **GameObject**.

2. **Add Physics Components:** Attach a **Rigidbody2D** and a **PolygonCollider2D**. Set the **Gravity Scale to 0** in the Rigidbody2D properties to ensure the drone does not fall "down" the screen.

3. **The Script:** Create a new C# script called DroneMovement.cs that inherits from MonoBehaviour.

4. **Serialization:** Declare a public float speed variable. Using the [SerializeField] attribute or public access allows you to edit these properties directly in the **Unity Inspector** without reopening the code.

+2

The Script: DroneMovement.cs

```
C#
using UnityEngine;

public class DroneMovement : MonoBehaviour
```

```
{
  [Header("Flight Settings")]
  public float moveSpeed = 8f;
  public float acceleration = 5f;
  public float deacceleration = 3f;

  private Rigidbody2D rb;
  private Vector2 moveInput;
  private Vector2 currentVelocity;

  void Start()
  {
    // Start runs once when the script is initialized
    rb = GetComponent<Rigidbody2D>();
    rb.gravityScale = 0;
  }

  void Update()
  {
    // Get Input (Handles WASD and Arrow Keys)
    moveInput.x = Input.GetAxisRaw("Horizontal");
    moveInput.y = Input.GetAxisRaw("Vertical");

    // Normalize movement to ensure speed is consistent on diagonals
    moveInput.Normalize();
  }

  void FixedUpdate()
  {
    // Calculate target velocity
    Vector2 targetVelocity = moveInput * moveSpeed;

    // Smoothly interpolate for a "hovering" feel
    // Time.deltaTime ensures movement is smooth across different computer frame rates
    float lerpSpeed = (moveInput.magnitude > 0) ? acceleration : deacceleration;
    currentVelocity = Vector2.Lerp(currentVelocity, targetVelocity, lerpSpeed * Time.fixedDeltaTime);

    // Apply movement to the Rigidbody physics simulation
    rb.velocity = currentVelocity;
  }
}
```

Step 3: Obstacle Spawning (The "Air Traffic")

In an arcade style, enemy drones and birds move across the screen at random intervals.

- **Prefabs:** Drag your enemy sprite into the **Project Folder** to make it a **Prefab**—a reusable template for GameObjects.

- **The Spawner:** Create an empty GameObject called "Spawner". Use a **Timer** in a script to **Instantiate** (create independent duplicates) of your enemy Prefabs at random heights.

Step 4: Collisions & Health

Detecting interactions is essential for gameplay mechanics like losing health.

1. **Tags:** Click your Enemy Prefab and set its **Tag** to "Enemy" in the Inspector to differentiate it from other objects.

2. **Triggers:** Use the OnTriggerEnter2D method in your script to detect when the drone hits an obstacle without a physical bounce.

```
C#
void OnTriggerEnter2D(Collider2D other) {
  if(other.CompareTag("Enemy")) {
    health--; // Reduce player health variable
    // Add particle system explosions for feedback
  }
}
```

Step 5: UI & Game Management

A **Canvas** is used as a container for UI elements to provide feedback to the user.

- **Create UI:** Add **Text** elements for "Health" and "Score".
- **Game Manager:** Create a central **Game Manager** script to track the game state.

- **Events:** Implement methods for `Win()` and `GameOver()`. Call `GameOver()` when health reaches zero, or `Win()` when the drone collides with a **"WinZone"** tag at the finish line.

- **Restart Button:** Use a UI Button that reloads the scene to allow the player to try again.

Hackathon Cheat Sheet

- **The Moving Background:** Instead of moving the player miles across a map, keep the drone at $X = 0$ and move the background texture offset to simulate flight.

- **Fast Scoring:** Set `score += Time.deltaTime * 5;` in the `Update()` method to increase the score automatically as the player survives.

- **Debugging:** Use the **Console** and `Debug.Log()` to print variables and resolve coding errors if the drone isn't behaving as expected.