Unity 2D: Collision Detection, Triggers & Sword Animation Tutorial

Overview This tutorial extends our Unity 2D Player Controller by adding collision detection, trigger systems, and creating an animated sword with trail effects. We'll walk through each step exactly as demonstrated, building from basic collisions to a complete sword animation system.

Prerequisites: Functional player with movement controller and collider attached

---

Step 1: Testing Basic Collision

We start with our functional player that already has a Box Collider 2D attached.

Create a Wall Object

1. Right-click in Hierarchy → Create Empty (name it "Wall")
2. Add Component → Sprite Renderer
3. Assign a square sprite to the Sprite Renderer
4. Add Component → Physics 2D → Box Collider 2D
5. Position the wall in front of your player

Test: Run the game - your player should stop when hitting the wall. The colliders are working as solid barriers.

Step 2: Converting to Trigger System

Now let's make the wall into a trigger zone instead of a solid barrier.

Make It a Trigger

1. Select the Wall GameObject
2. In the Box Collider 2D component, check "Is Trigger"

Test: Run the game - your player now passes through the wall. The collider is now a detection zone, not a physical barrier.

Step 3: Creating the Trigger Script

Create a new script to detect when the player enters the trigger zone.

Create TriggerDetector Script Create a new C# script called TriggerDetector.cs:

using UnityEngine;

public class TriggerDetector : MonoBehaviour { void OnTriggerEnter2D(Collider2D other) { Debug.Log("Something entered the trigger!"); } }

1. Attach this script to your Wall GameObject
2. Test: Run the game and walk through the wall - check the Console for the debug message

---

Step 4: Understanding OnTriggerEnter2D

Line-by-Line Explanation

void OnTriggerEnter2D(Collider2D other)

Special Unity method that automatically runs when another 2D collider enters this trigger zone. Unity calls this method for you - you don't call it manually.

Parameter other: This is the collider that entered the trigger. It could be the player, an enemy, a projectile - anything with a collider.

When it triggers: The moment another collider crosses into the trigger boundaries.

Step 5: Using other.name for Detection

Let's see what's actually entering our trigger by checking the name.

Update the Script using UnityEngine;

public class TriggerDetector : MonoBehaviour { void OnTriggerEnter2D(Collider2D other) { Debug.Log("Object entered: " + other.name); } }

Test: Run the game and walk through the trigger. The Console will show "Object entered: Player" (or whatever your player GameObject is named).

Why other.name?

- other.name gives you the GameObject's name in the hierarchy
- Useful for debugging but not reliable for gameplay logic
- Names can change, be duplicated, or contain spaces

Step 6: Tags and CompareTag Method

Tags are the proper way to identify game objects in code.

Set Up Player Tag

1. Select your Player GameObject
2. Click the Tag dropdown (top of Inspector)
3. Add Tag…
4. Create a new tag called "Player"
5. Assign the "Player" tag to your Player GameObject

Update Script to Use Tags using UnityEngine;

public class TriggerDetector : MonoBehaviour { void OnTriggerEnter2D(Collider2D other) { Debug.Log("Object entered: " + other.name);

```
    if (other.CompareTag("Player"))
    {
        Debug.Log("Player detected!");
    }
}
```

}

Why CompareTag is Better // Good way (recommended) if (other.CompareTag(“Player”))

// Bad way (works but not recommended) if (other.tag == “Player”)

CompareTag advantages:

- Faster performance
- Prevents typos (will error if tag doesn’t exist)
- More readable code

Step 7: Animation and Animator Windows

Now let’s create an animated sword. First, let’s understand Unity’s animation system.

Opening the Animation Windows

1. Window → Animation → Animation (creates animation clips)
2. Window → Animation → Animator (manages animation state machine)

The Two Windows Explained

Animation Window:

- Creates individual animation clips (like “sword_swing” or “idle”)
- Timeline where you set keyframes
- Records changes to GameObjects over time

Animator Window:

- Shows the state machine (which animation plays when)
- Manages transitions between animations
- Sets up parameters and conditions

Step 8: Creating the Sword GameObject

Set Up the Sword

1. Create → 2D Object → Sprite (name it “Sword”)
2. Assign a sword sprite to the Sprite Renderer

3. Position it near your player
4. Set the Pivot Point to the handle/bottom of the sword:

- Select the sword sprite in Project window
- In Inspector → Sprite Editor
- Set Pivot to “Custom” and adjust to the sword handle
- Apply changes

Why Pivot Matters: The pivot point is where rotation happens. For a sword, we want it to rotate around the handle, not the center.

Step 9: Animate the Sword Swing

Create the Swing Animation

1. Select the Sword GameObject
2. In Animation Window, click “Create”
3. Save as “Sword_Swing” (this creates both the clip and animator controller)
4. Set the timeline length to 1 second (60 frames at 60fps)

Record the Animation Keyframes

1. Click the red Record button
2. At frame 0: Set rotation to (0, 0, 0)
3. Move to frame 30 (0.5 seconds): Set rotation to (0, 0, 90)
4. Move to frame 60 (1 second): Set rotation back to (0, 0, 0)
5. Stop recording

Animation Explanation

- Frame 0: Sword starts at normal position (0 degrees)
- Frame 30: Sword rotated 90 degrees (peak of swing)
- Frame 60: Sword returns to start position
- Total time: 1 second for complete swing motion

Test: Click play in the Animation window to preview your sword swing.

Step 10: Adding Trail Renderer

Create Trail Effect

1. Create → Create Empty (name it “SwordTip”)
2. Make it a child of the Sword GameObject
3. Position it at the tip of the sword
4. Add Component → Effects → Trail Renderer

Trail Renderer Parameters Explained

Time: How long the trail lasts (try 0.3 seconds)

- Longer = trail stays visible longer
- Shorter = trail fades quickly

Width: Trail thickness over its lifetime

- Start Width: How thick the trail is when created
- End Width: How thick at the end (usually 0 for tapering)

Material: What the trail looks like

- Use Default-Line or create a custom material
- Controls color, transparency, glow effects

Color: Trail color over lifetime

- Can fade from one color to another
- Alpha controls transparency

Recommended Settings for Sword Trail Time: 0.3 Start Width: 0.2 End Width: 0.05 Color: White to transparent Material: Default-Line

---

Step 11: Creating Idle Animation

Make the Sword Breathe

1. In Animation Window, create new clip “Sword_Idle”
2. Set length to 2 seconds (120 frames)
3. Record keyframes:

- Frame 0: Scale (1, 1, 1)
- Frame 60: Scale (1.1, 1.1, 1)
- Frame 120: Scale (1, 1, 1)

1. In Animation clip settings, check “Loop Time”

This creates a gentle breathing/pulsing effect for when the sword is idle.

Step 12: Animator Controller Setup

Understanding the Animator

1. Open Animator Window
2. You’ll see two states: Sword_Idle and Sword_Swing
3. Right-click Sword_Idle → Set as Layer Default State (orange color)

Create Animator Parameter

1. In Animator Window, click “Parameters” tab
2. Click + → Bool
3. Name it “IsSwinging”

Set Up Transitions

1. Right-click Sword_Idle → Make Transition → Sword_Swing
2. Click the transition arrow
3. In Inspector, add condition: IsSwinging equals true
4. Uncheck “Has Exit Time” for instant response
5. Right-click Sword_Swing → Make Transition → Sword_Idle
6. Add condition: IsSwinging equals false
7. Check “Has Exit Time” so swing completes before returning to idle

---

Step 13: Sword Controller Script

Create the script that controls when the sword swings.

Create SwordController Script

```
using UnityEngine;

public class SwordController : MonoBehaviour
{
        private Animator animator;
        private bool isSwinging = false;
        private float swingTimer = 0f;

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

        void Update()
        {
            // Check for mouse click
            if (Input.GetMouseButtonDown(0) && !isSwinging)
            {
                // Start swinging
                isSwinging = true;
                animator.SetBool("IsSwinging", true);
                swingTimer = 1f;  // 1 second swing duration
            }

            // Count down the swing timer
            if (isSwinging)
            {
                swingTimer -= Time.deltaTime;

                if (swingTimer <= 0f)
                {
                    // Stop swinging
                    isSwinging = false;
                    animator.SetBool("IsSwinging", false);
                }
            }
        }
}
```

Script Explanation

Line by Line: private Animator animator;

Reference to the Animator component that controls our animation state machine.

private bool isSwinging = false;

Tracks if the sword is currently swinging - prevents clicking multiple times during one swing.

private float swingTimer = 0f;

Counts down how much time is left in the current swing animation.

animator = GetComponent();

Gets the Animator component attached to the same GameObject.

if (Input.GetMouseButtonDown(0) && !isSwinging)

Checks two conditions:

- Player clicked left mouse button (GetMouseButtonDown(0))
- Sword is not already swinging (prevents spam clicking)

isSwinging = true; animator.SetBool(“IsSwinging”, true); swingTimer = 1f;

Starts the swing:

- Sets our tracking variable to true
- Tells the animator to switch to swing animation
- Sets timer to 1 second (our animation length)

swingTimer -= Time.deltaTime;

Counts down the timer by subtracting the time since last frame.

if (swingTimer <= 0f)

Checks if 1 second has passed - time to stop swinging.

Attach the Script

1. Add this script to your Sword GameObject
2. Test: Press Space to see the sword swing with trail effect!

Complete System Test

What you should have:

1. □ Player with movement and collision
2. □ Trigger zone that detects player using tags
3. □ Animated sword with 1-second swing (0° → 90° → 0°)
4. □ Trail renderer at sword tip
5. □ Idle breathing animation

6. ✅ Script-controlled animation transitions

Test everything:

- Move player around (basic movement works)
- Walk through trigger zone (console shows detection)
- Click left mouse button (sword swings with trail effect)
- Wait (sword returns to idle breathing animation)

Key Concepts Learned

Collision vs Triggers:

- Colliders = solid barriers
- Triggers = detection zones

Script Communication:

- OnTriggerEnter2D automatically called by Unity
- other parameter contains the entering object
- Use CompareTag for reliable object identification

Animation Workflow:

- Animation Window creates clips
- Animator Window manages state machine
- Scripts control transitions with parameters

Trail Renderer:

- Follows GameObject movement
- Time controls trail length
- Width controls thickness
- Material controls appearance

This foundation gives you everything needed for interactive game objects, character abilities, and visual effects!