**Trigger no Player (um único OnTriggerEnter no Player)**

**Vantagens:**

* Centraliza toda a lógica em um lugar só.
* Mais fácil de **controlar e debugar** o que acontece com o player.
* Evita a necessidade de scripts diferentes em cada item (pode usar um switch por tag ou nome).

**Desvantagens:**

* Código do Player pode virar uma **salada** com muitas condições se você tiver muitos tipos de item.
* Pouca **modularidade** – o Player precisa conhecer os efeitos de todos os itens.
* Pouco flexível para crescer (manutenção difícil com 10+ itens diferentes).

**Trigger em cada Item (um OnTriggerEnter no próprio prefab do item)**

**Vantagens:**

* **Alta modularidade** – cada item sabe o que faz (e só ele).
* Fácil de **reaproveitar e duplicar** (só trocar o efeito).
* **Mais escalável**: adicionar novos tipos de item não exige mudar o script do Player.
* Clean code: Player não precisa saber o que é um “item de força”, “item de vida”, etc.

**Desvantagens:**

* Cada item precisa ter um script (mas pode ser reutilizado com interface/herança).
* Pode ter mais componentes na cena (mas não afeta performance de forma significativa).

**Conclusão (Recomendação prática):**

| **Situação** | **Melhor escolha** |
| --- | --- |
| Poucos itens diferentes e projeto simples | Trigger no **Player** |
| Muitos tipos de itens ou projeto que vai crescer | **Trigger nos Itens** com scripts modulares |

**Cenário**

Você tem 3 tipos de itens no seu jogo:

* **Cura** (aumenta vida)
* **Força** (aumenta dano)
* **Moeda** (aumenta score)

Abordagem 1: **Trigger no Player (centralizado)**

// PlayerController.cs

using UnityEngine;

public class PlayerController : MonoBehaviour

{

public int vida = 100;

public int forca = 10;

public int moedas = 0;

private void OnTriggerEnter(Collider other)

{

switch (other.tag)

{

case "Cura":

vida += 20;

Debug.Log("Pegou item de cura. Vida: " + vida);

Destroy(other.gameObject);

break;

case "Forca":

forca += 5;

Debug.Log("Pegou item de força. Força: " + forca);

Destroy(other.gameObject);

break;

case "Moeda":

moedas += 1;

Debug.Log("Pegou uma moeda. Total: " + moedas);

Destroy(other.gameObject);

break;

default:

Debug.Log("Colidiu com algo desconhecido.");

break;

}

}

}

**Abordagem 2: Trigger nos Itens (scripts modulares)**

**Interface base:**

// IItemColetavel.cs

using UnityEngine;

public interface IItemColetavel

{

void Coletar(PlayerController player);

}

Script genérico nos itens:

// ItemCura.cs

using UnityEngine;

public class ItemCura : MonoBehaviour, IItemColetavel

{

public int valorCura = 20;

public void Coletar(PlayerController player)

{

player.vida += valorCura;

Debug.Log("Cura aplicada. Vida: " + player.vida);

Destroy(gameObject);

}

private void OnTriggerEnter(Collider other)

{

PlayerController p = other.GetComponent<PlayerController>();

if (p != null)

Coletar(p);

}

}

// ItemForca.cs

using UnityEngine;

public class ItemForca : MonoBehaviour, IItemColetavel

{

public int bonusForca = 5;

public void Coletar(PlayerController player)

{

player.forca += bonusForca;

Debug.Log("Força aumentada. Força: " + player.forca);

Destroy(gameObject);

}

private void OnTriggerEnter(Collider other)

{

PlayerController p = other.GetComponent<PlayerController>();

if (p != null)

Coletar(p);

}

}

// ItemMoeda.cs

using UnityEngine;

public class ItemMoeda : MonoBehaviour, IItemColetavel

{

public void Coletar(PlayerController player)

{

player.moedas += 1;

Debug.Log("Pegou moeda. Total: " + player.moedas);

Destroy(gameObject);

}

private void OnTriggerEnter(Collider other)

{

PlayerController p = other.GetComponent<PlayerController>();

if (p != null)

Coletar(p);

}

}

// PlayerController.cs

using UnityEngine;

public class PlayerController : MonoBehaviour

{

public int vida = 100;

public int forca = 10;

public int moedas = 0;

}