

Code Documentation: Event-Driven System in Unity

This documentation explains how events are defined, invoked, and subscribed to in the event-driven system.

1. EventManager (Central Event Hub)

Purpose:

The EventManager acts as a singleton that manages events and inventory storage. It defines UnityEvents for item collection and door opening.

```
using UnityEngine;
using UnityEngine.Events;
using System.Collections.Generic;

public class EventManager : MonoBehaviour
{
    // Singleton instance of EventManager
    public static EventManager Instance;

    // UnityEvents for item collection and door opening
    public UnityEvent onItemCollected;
    public UnityEvent onDoorOpened;

    // List to store collected items
    public List<string> inventory = new List<string>();

    private void Awake()
    {
        // Ensure only one instance of EventManager exists
        if (Instance == null)
        {
            Instance = this;
        }
        Debug.Log("EventManager Initialized!"); // Check if EventManager is active
    }

    // Method to collect an item and trigger an event
    public void CollectItem(string itemName)
    {
        inventory.Add(itemName);
        Debug.Log("Item Collected: " + itemName); // Check if item is collected
        onItemCollected?.Invoke(); // Invoke the event if there are subscribers
    }
}
```

```
}  
}
```

Key Features:

- Defines two UnityEvents (onItemCollected and onDoorOpened).
- Implements the **Singleton pattern** to ensure a single event manager instance.
- Stores collected items in the inventory list.
- Calls onItemCollected.Invoke() when an item is collected.

2. ItemPickup (Triggering an Event on Collision)

Purpose:

Handles item pickup and invokes the onItemCollected event when the player interacts with the object.

using UnityEngine;

```
public class ItemPickup : MonoBehaviour  
{  
    // Name of the item being picked up  
    public string itemName;  
  
    private void OnTriggerEnter(Collider other)  
    {  
        // Check if the colliding object is the Player  
        if (other.CompareTag("Player"))  
        {  
            // Add the item to the inventory  
            EventManager.Instance.inventory.Add(itemName);  
  
            // Invoke the event to notify other systems  
            EventManager.Instance.onItemCollected?.Invoke();  
  
            Debug.Log("Item picked up: " + itemName);  
  
            // Destroy the item after collection  
            Destroy(gameObject);  
        }  
    }  
}
```

```
}
```

Key Features:

- Checks if the player interacts with the object.
- Adds the item to the inventory list.
- Invokes onItemCollected to notify other components.
- Destroys the object after collection.

3. DoorController (Listening for an Event and Responding)

Purpose:

Listens to the onDoorOpened event and rotates the door when triggered.

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class DoorController : MonoBehaviour
```

```
{
```

```
    // Target rotation for door opening
```

```
    public Vector3 openRotation = new Vector3(0, 90, 0);
```

```
    // Speed of the door opening animation
```

```
    public float openSpeed = 2f;
```

```
    // Prevents multiple door openings
```

```
    private bool isOpen = false;
```

```
    private void Start()
```

```
    {
```

```
        // Subscribe to the door open event
```

```
        if (EventManager.Instance != null)
```

```
        {
```

```
            EventManager.Instance.onDoorOpened.AddListener(OpenDoor);
```

```
            Debug.Log("Event Listener Added to DoorController");
```

```
        }
```

```
    else
```

```
    {
```

```

        Debug.LogError("EventManager Instance is NULL!");
    }
}

// Called when the door open event is triggered
public void OpenDoor()
{
    if (!isOpen)
    {
        Debug.Log("Door Rotating...");
        StartCoroutine(RotateDoor()); // Smoothly rotate the door
        isOpen = true;
    }
    else
    {
        Debug.Log("Door is already open!");
    }
}

// Coroutine to smoothly rotate the door
private IEnumerator RotateDoor()
{
    Quaternion startRotation = transform.rotation;
    Quaternion endRotation = Quaternion.Euler(openRotation);
    float time = 0;

    while (time < 1)
    {
        transform.rotation = Quaternion.Lerp(startRotation, endRotation, time);
        time += Time.deltaTime * openSpeed;
        yield return null;
    }

    transform.rotation = endRotation;
    Debug.Log("Door Rotation Completed");
}
}

```

Key Features:

- Subscribes to onDoorOpened in Start().
- Calls OpenDoor() when the event is triggered.

- Uses a coroutine to smoothly rotate the door.
- Prevents re-opening using the isOpen flag.

4. UIManager (Updating UI on Event Triggers)

Purpose:

Listens to onItemCollected and updates the UI to reflect item collection and score changes.

```
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    // UI elements for score and inventory display
    public Text scoreText;
    public Text inventoryText;

    private int score = 0;

    private void Start()
    {
        Debug.Log("UIManager Started!"); // Check if script is running

        // Subscribe to item collected event
        EventManager.Instance.onItemCollected.AddListener(UpdateScore);
        EventManager.Instance.onItemCollected.AddListener(UpdateUI);
    }

    // Updates the score when an item is collected
    public void UpdateScore()
    {
        score++;
        Debug.Log("Score Updated: " + score); // Check if score updates
        scoreText.text = "Score: " + score;
    }

    // Updates the inventory UI when an item is collected
    private void UpdateUI()
    {
        Debug.Log("Updating Inventory UI...");
        inventoryText.text = "Inventory: " + string.Join(", ", EventManager.Instance.inventory);
        Debug.Log("Updated Inventory: " + inventoryText.text);
    }
}
```

```
}  
}
```

Key Features:

- Subscribes to `onItemCollected` to update the score and inventory UI.
- Updates the UI dynamically whenever an item is collected.

5. TriggerEvent (Triggering the Door Event)

Purpose:

Detects when the player enters a trigger zone and invokes the `onDoorOpened` event.

using UnityEngine;

```
public class TriggerEvent : MonoBehaviour  
{  
    private void OnTriggerEnter(Collider other)  
    {  
        // Ensure the collider has the "Player" tag before invoking the event  
        if (other.CompareTag("Player"))  
        {  
            if (EventManager.Instance != null)  
            {  
                EventManager.Instance.onDoorOpened?.Invoke();  
                Debug.Log("Player entered the trigger zone. Event triggered!");  
            }  
            else  
            {  
                Debug.LogWarning("EventManager instance is null! Ensure it is assigned.");  
            }  
        }  
    }  
}
```

Key Features:

- Detects player entry into a trigger zone.
- Invokes `onDoorOpened` to open the door.

Conclusion

This event-driven system enables **modular, scalable, and efficient** interactions in the game. By using events instead of direct dependencies, it allows new features to be added with minimal changes to existing code.

This approach ensures **separation of concerns**:

- EventManager handles event definitions.
- ItemPickup, TriggerEvent, etc., invoke events when necessary.
- UIManager and DoorController listen and respond to events dynamically.