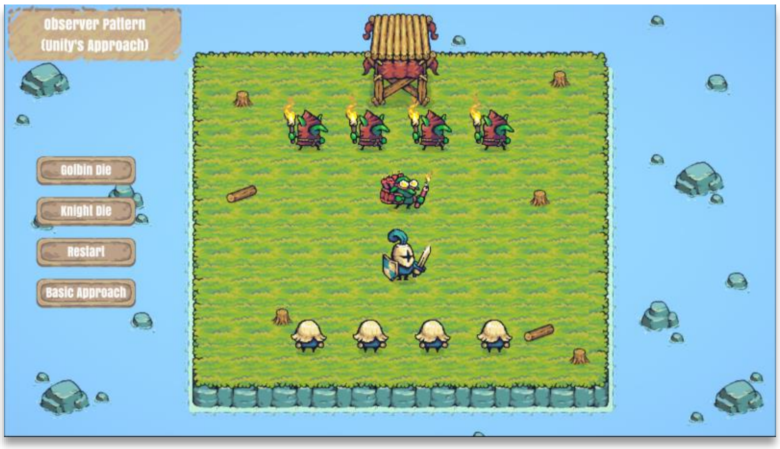
Observer Pattern:

The Observer Pattern is basically a way to let objects know when something has happened, like sending notifications. In simpler terms, it's like telling certain objects about an event so they can react accordingly. For example, if something happens (like a tax increase on property**)** and certain objects or people need to know about it and act on it (like realtors reacting to the news of tax increasing), that's where the Observer Pattern comes in.

There's a basic C# way using OnNotify() interface to implement the Observer Pattern, and Unity has its own approach using events. Both achieve the same result, but I'll focus on explaining the Unity way. You can still check out the basic approach in the project files and look through the scripts for more details.

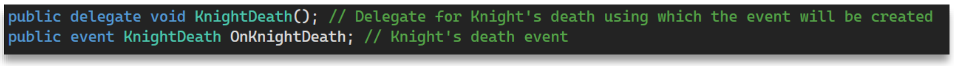
Let's take a simple example of a Knight and his troops versus a King Goblin and his troops. What we want to achieve is this: when the King Goblin dies, we notify the Knight so that he and his troops are happy, while the King Goblin's troops are sad. Similarly, if the Knight dies, we notify the King Goblin so he and his troops are happy, and the Knight's troops are sad.



There are a few ways to achieve this. For example, we could give the Knight's class a reference to the King Goblin and keep checking in the Update method if the Knight dies (or vice versa). Another way could be, when the Knight dies, he references the King Goblin class and calls a method from that class to trigger the necessary behavior. These approaches work, but they can lead to tight coupling and waste resources.

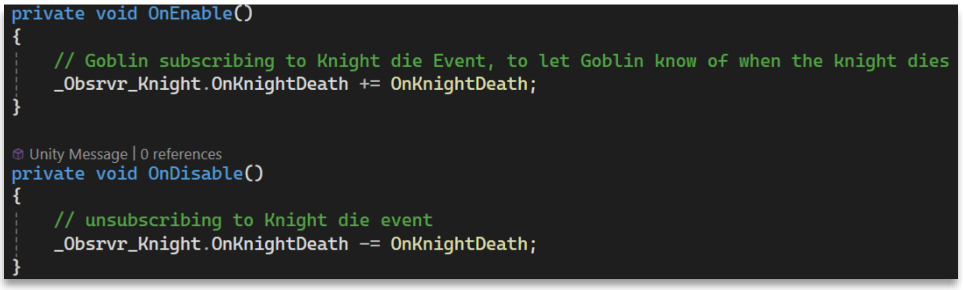
To avoid these problems, we can use the Observer Pattern. With this pattern, we can set up an OnDie event in the Knight class that the King Goblin class subscribes to (and vice versa). When the Knight dies, we simply invoke the event, and all the classes that have subscribed to it will be notified. They can then perform any actions they need. This helps keep the code clean and efficient.

We can set up the events like this:



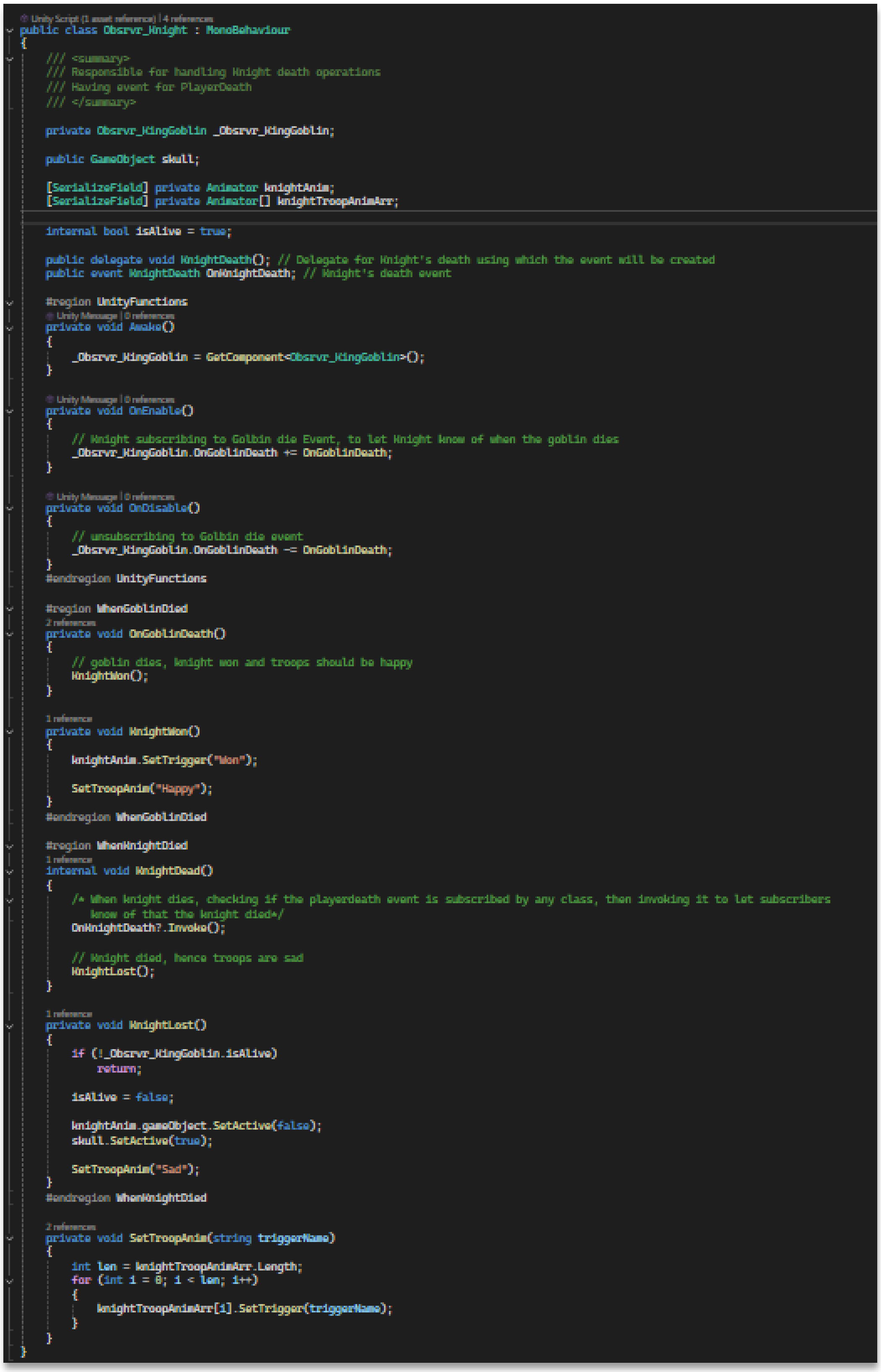
```csharp
public delegate void KnightDeath(); // Delegate for Knight's death using which the event will be created
public event KnightDeath OnKnightDeath; // Knight's death event
```

And let the classes to subscribe like this:



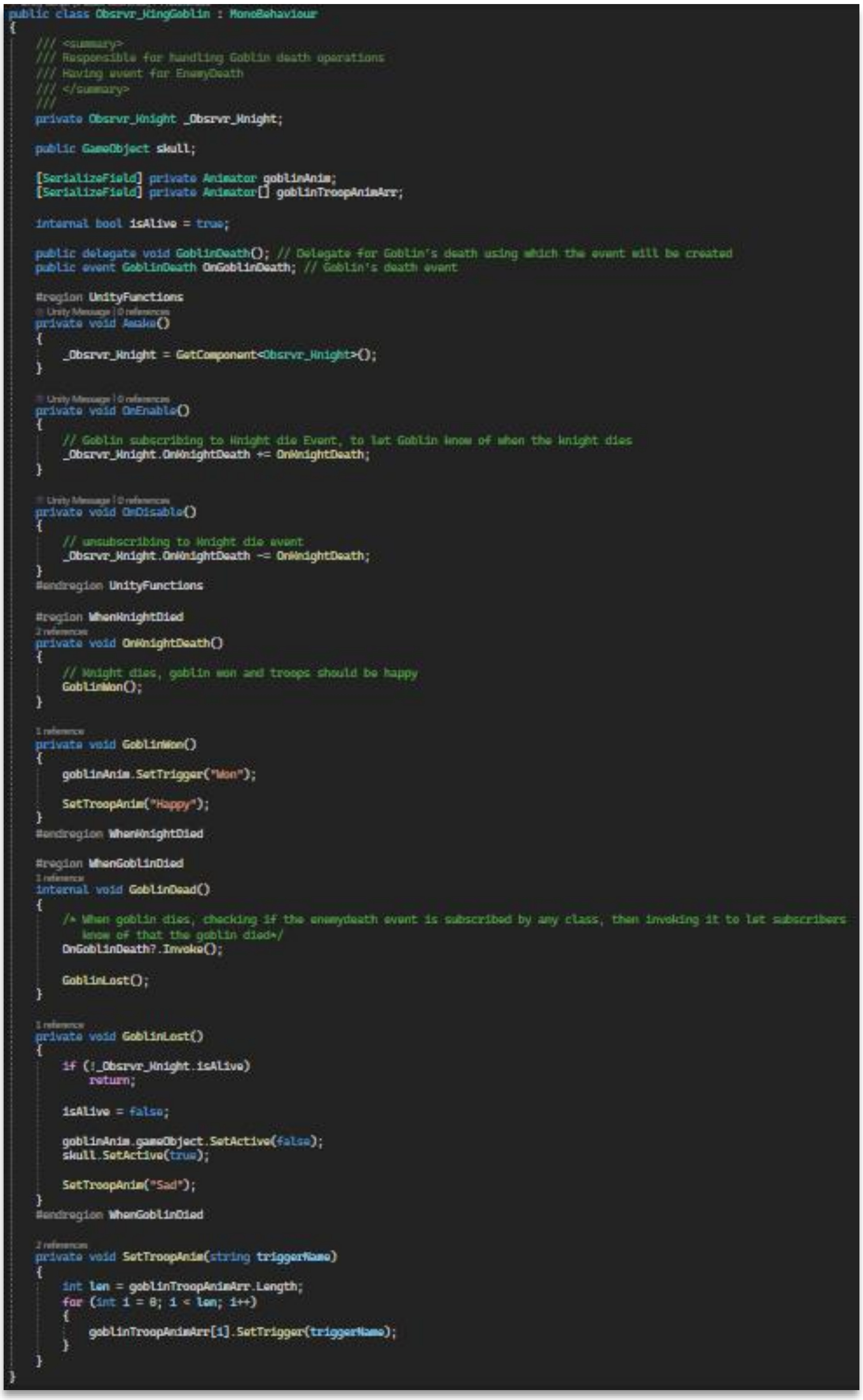
```csharp
private void OnEnable()
{
    // Goblin subscribing to Knight die Event, to let Goblin know of when the knight dies
    _Obsrvr_Knight.OnKnightDeath += OnKnightDeath;
}

Unity Message | 0 references
private void OnDisable()
{
    // unsubscribing to Knight die event
    _Obsrvr_Knight.OnKnightDeath -= OnKnightDeath;
}
```

Thus, the knight class will have the OnKnightDeath event that will be subscribed by the goblin class. When this event is triggered the goblin will be notified that the knight is dead, so be happy.



```
Unity Script (1 asset reference) | 4 references
public class Obsrvr_Knight : MonoBehaviour
{
    /// <summary>
    /// Responsible for handling Knight death operations
    /// Having event for PlayerDeath
    /// </summary>

    private Obsrvr_KingGoblin _Obsrvr_KingGoblin;

    public GameObject skull;

    [SerializeField] private Animator knightAnim;
    [SerializeField] private Animator[] knightTroopAnimArr;

    internal bool isAlive = true;

    public delegate void KnightDeath(); // Delegate for Knight's death using which the event will be created
    public event KnightDeath OnKnightDeath; // Knight's death event

    #region UnityFunctions
    Unity Message | 0 references
    private void Awake()
    {
        _Obsrvr_KingGoblin = GetComponent<Obsrvr_KingGoblin>();
    }

    Unity Message | 0 references
    private void OnEnable()
    {
        // Knight subscribing to Golbin die Event, to let Knight know of when the goblin dies
        _Obsrvr_KingGoblin.OnGoblinDeath += OnGoblinDeath;
    }

    Unity Message | 0 references
    private void OnDisable()
    {
        // unsubscribing to Golbin die event
        _Obsrvr_KingGoblin.OnGoblinDeath -= OnGoblinDeath;
    }
    #endregion UnityFunctions

    #region WhenGoblinDied
    2 references
    private void OnGoblinDeath()
    {
        // goblin dies, knight won and troops should be happy
        KnightWon();
    }

    1 reference
    private void KnightWon()
    {
        knightAnim.SetTrigger("Won");

        SetTroopAnim("Happy");
    }
    #endregion WhenGoblinDied

    #region WhenKnightDied
    1 reference
    internal void KnightDead()
    {
        /* When knight dies, checking if the playerdeath event is subscribed by any class, then invoking it to let subscribers
           know of that the knight died*/
        OnKnightDeath?.Invoke();

        // Knight died, hence troops are sad
        KnightLost();
    }

    1 reference
    private void KnightLost()
    {
        if (!_Obsrvr_KingGoblin.isAlive)
            return;

        isAlive = false;

        knightAnim.gameObject.SetActive(false);
        skull.SetActive(true);

        SetTroopAnim("Sad");
    }
    #endregion WhenKnightDied

    2 references
    private void SetTroopAnim(string triggerName)
    {
        int len = knightTroopAnimArr.Length;
        for (int i = 0; i < len; i++)
        {
            knightTroopAnimArr[i].SetTrigger(triggerName);
        }
    }
}
```

Similarly, the Goblin class will have the OnGoblinDeath event that will be subscribed by the knight class. When this event is triggered the knight will be notified that the goblin is dead, so be happy.



```
public class Obsrvr_KingGoblin : MonoBehaviour
{
    /// <summary>
    /// Responsible for handling Goblin death operations
    /// Having event for EnemyDeath
    /// </summary>
    ///
    private Obsrvr_Knight _Obsrvr_Knight;

    public GameObject skull;

    [SerializeField] private Animator goblinAnim;
    [SerializeField] private Animator[] goblinTroopAnimArr;

    internal bool isAlive = true;

    public delegate void GoblinDeath(); // Delegate for Goblin's death using which the event will be created
    public event GoblinDeath OnGoblinDeath; // Goblin's death event

    #region UnityFunctions
    Unity Message | 0 references
    private void Awake()
    {
        _Obsrvr_Knight = GetComponent<Obsrvr_Knight>();
    }

    Unity Message | 0 references
    private void OnEnable()
    {
        // Goblin subscribing to Knight die Event, to let Goblin know of when the knight dies
        _Obsrvr_Knight.OnKnightDeath += OnKnightDeath;
    }

    Unity Message | 0 references
    private void OnDisable()
    {
        // unsubscribing to Knight die event
        _Obsrvr_Knight.OnKnightDeath -= OnKnightDeath;
    }
    #endregion UnityFunctions

    #region WhenKnightDied
    2 references
    private void OnKnightDeath()
    {
        // Knight dies, goblin won and troops should be happy
        GoblinWon();
    }

    1 reference
    private void GoblinWon()
    {
        goblinAnim.SetTrigger("Won");

        SetTroopAnim("Happy");
    }
    #endregion WhenKnightDied

    #region WhenGoblinDied
    1 reference
    internal void GoblinDead()
    {
        /* When goblin dies, checking if the enemydeath event is subscribed by any class, then invoking it to let subscribers
           know of that the goblin died*/
        OnGoblinDeath?.Invoke();

        GoblinLost();
    }

    1 reference
    private void GoblinLost()
    {
        if (!_Obsrvr_Knight.isAlive)
            return;

        isAlive = false;

        goblinAnim.gameObject.SetActive(false);
        skull.SetActive(true);

        SetTroopAnim("Sad");
    }
    #endregion WhenGoblinDied

    2 references
    private void SetTroopAnim(string triggerName)
    {
        int len = goblinTroopAnimArr.Length;
        for (int i = 0; i < len; i++)
        {
            goblinTroopAnimArr[i].SetTrigger(triggerName);
        }
    }
}
```

Now, go and check the scripts and play the project and try out yourself!