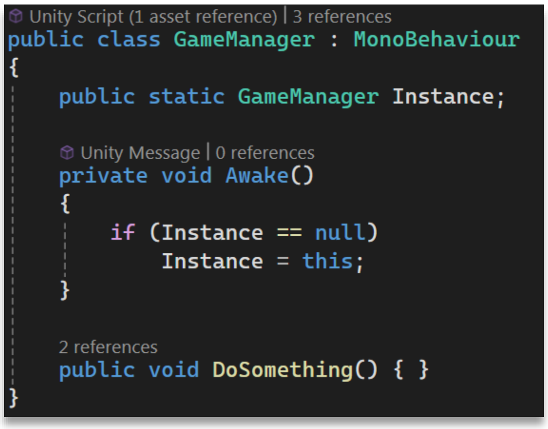
Singleton Pattern:

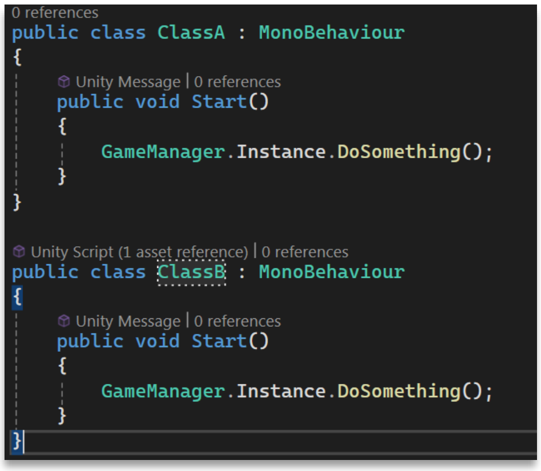
The singleton pattern is all about making sure there's only one instance of a class. In simple terms, it's like creating one object that is used everywhere in the scene, and there are no copies or duplicates of it. It is one of the most important Design Pattern. Programmers use this when they need just one object for things that need to be shared across the whole program, like a game manager, audio manager, or multiplayer connection class etc.

Let's say we have a game manager class with some code. If class A and class B both try to use the game manager, but for some reason, two separate instances of the game manager are created, then class A and class B could behave differently. To avoid this and make sure both classes always use the same game manager, we can turn the game manager into a singleton. This way, both class A and class B will always reference the exact same instance, ensuring consistent results. In Unity C#, we can easily create a singleton like this:



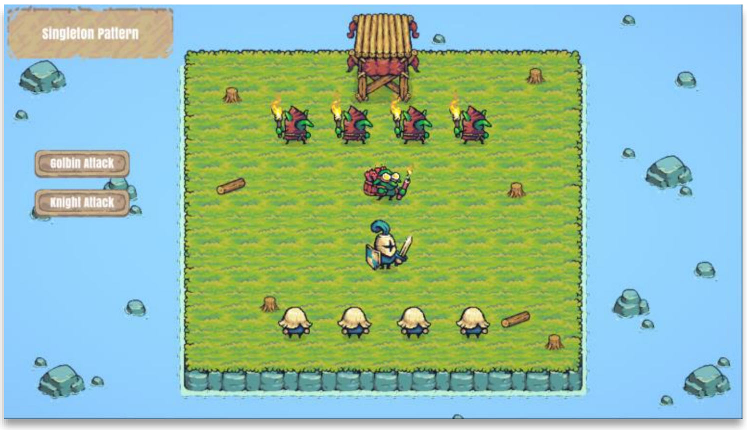
```
Unity Script (1 asset reference) | 3 references
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    Unity Message | 0 references
    private void Awake()
    {
        if (Instance == null)
            Instance = this;
    }

    2 references
    public void DoSomething() { }
}
```

The instance should be declared as public static so that it can be accessed from anywhere without needing to create a new object of the class. This code ensures that only one instance of the Game Manager class is created. Then, class A and class B can simply call the Game Manager instance like this:

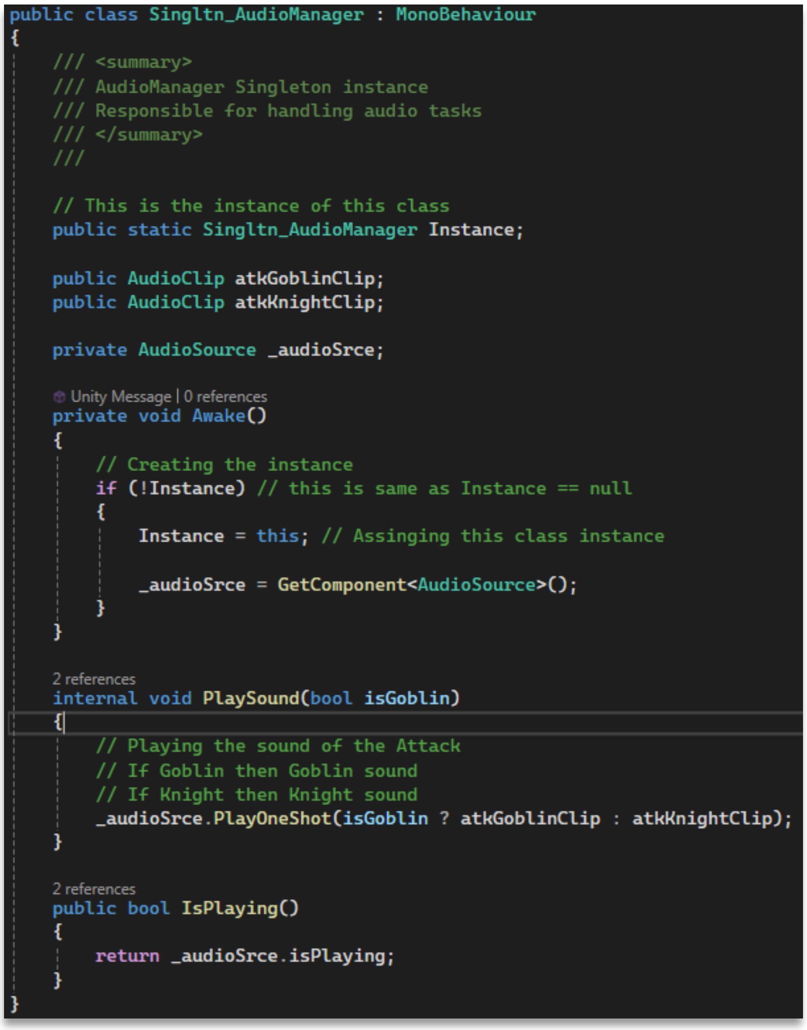


```
0 references
public class ClassA : MonoBehaviour
{
    Unity Message | 0 references
    public void Start()
    {
        GameManager.Instance.DoSomething();
    }
}

Unity Script (1 asset reference) | 0 references
public class ClassB : MonoBehaviour
{
    Unity Message | 0 references
    public void Start()
    {
        GameManager.Instance.DoSomething();
    }
}
```

For our example, we have taken a classic knight vs goblin example in a simple 2D setup.

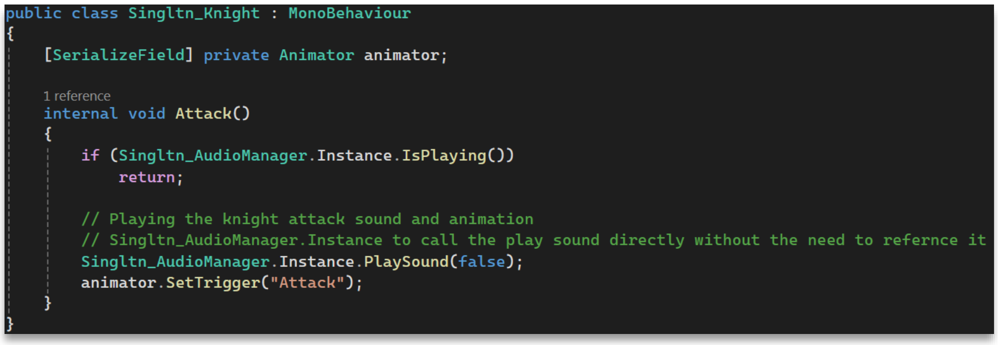


What we want to achieve here is that when the Goblin Attack button is pressed, it should play a goblin sound and animation, and when the Knight Attack button is pressed, it should play the relevant knight sound and animation. To do this, we can use an Audio Manager and make it a singleton to ensure the same instance of the class is used.

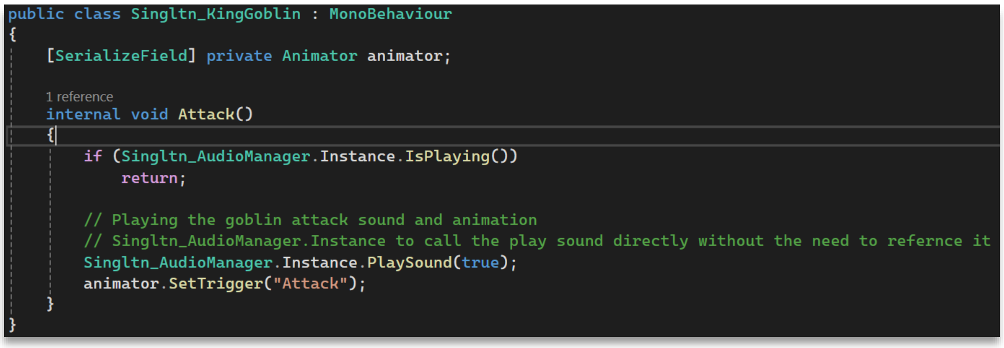


```
public class Singltn_AudioManager : MonoBehaviour
{
    /// <summary>
    /// AudioManager Singleton instance
    /// Responsible for handling audio tasks
    /// </summary>
    ///

    // This is the instance of this class
    public static Singltn_AudioManager Instance;

    public AudioClip atkGoblinClip;
    public AudioClip atkKnightClip;

    private AudioSource _audioSrce;

    Unity Message | 0 references
    private void Awake()
    {
        // Creating the instance
        if (!Instance) // this is same as Instance == null
        {
            Instance = this; // Assinging this class instance

            _audioSrce = GetComponent<AudioSource>();
        }
    }

    2 references
    internal void PlaySound(bool isGoblin)
    {
        // Playing the sound of the Attack
        // If Goblin then Goblin sound
        // If Knight then Knight sound
        _audioSrce.PlayOneShot(isGoblin ? atkGoblinClip : atkKnightClip);
    }

    2 references
    public bool IsPlaying()
    {
        return _audioSrce.isPlaying;
    }
}
```

Then this singleton Instance can be accessed by the class like this:



```
public class Sing1tn_Knight : MonoBehaviour
{
    [SerializeField] private Animator animator;

    1 reference
    internal void Attack()
    {
        if (Sing1tn_AudioManager.Instance.IsPlaying())
            return;

        // Playing the knight attack sound and animation
        // Sing1tn_AudioManager.Instance to call the play sound directly without the need to refernce it
        Sing1tn_AudioManager.Instance.PlaySound(false);
        animator.SetTrigger("Attack");
    }
}
```



```
public class Sing1tn_KingGoblin : MonoBehaviour
{
    [SerializeField] private Animator animator;

    1 reference
    internal void Attack()
    {
        if (Sing1tn_AudioManager.Instance.IsPlaying())
            return;

        // Playing the goblin attack sound and animation
        // Sing1tn_AudioManager.Instance to call the play sound directly without the need to refernce it
        Sing1tn_AudioManager.Instance.PlaySound(true);
        animator.SetTrigger("Attack");
    }
}
```

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