Code Examples

Key Points

- **Coroutines** allow you to split a function up so that it takes place over a number of frames.
- They are a unique implementation of the **IEnumerator** type in C#, which normally iterates over a collection but, in this case, is used to return different parts of a function instead.
- Coroutines can be suspended with the **yield keyword**, usually until the next frame. Doing this inside of a while loop creates a mini-update loop.
- It's also possible to wait for a number of seconds (using the **Wait for Seconds** class) or to yield while another coroutine runs.
- **Break** is used instead of return to end a coroutine early.
- If you don't need to yield a coroutine (as in, suspend it until at least the next frame) then it doesn't need to be a coroutine.

Code Examples

Invoke

```
1. public class InvokeExample : MonoBehaviour
2. {
3.     void Start()
4.     {
5.         // Calls MyFunction in 5 seconds time.
6.         Invoke(nameof(MyFunction), 5);
7.
8.         // Calls MyFunction in 1 second, and then every 2 seconds.
9.         InvokeRepeating(nameof(MyFunction), 1, 2);
10.    }
11.
12.    void MyFunction()
```

```
13.  {
14.      // Do Something!
15.  }
16. }
```

Basic Coroutine

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class CoroutineExamples : MonoBehaviour
5.  {
6.      bool running;
7.
8.      private void Start()
9.      {
10.         StartCoroutine(MyCoroutine());
11.     }
12.
13.     IEnumerator MyCoroutine()
14.     {
15.         // Do this first,
16.         running = true;
17.
18.         while (running)
19.         {
20.             // Do this every frame,
21.             yield return null;
```

```
    }

    // Then wait five seconds

    yield return new WaitForSeconds(5);

    // Then do this last.
  }
}
```

Wait for Seconds

```
using UnityEngine;
using System.Collections;

public class CoroutineExamples : MonoBehaviour
{
  private void Start()
  {
    StartCoroutine(MyCoroutine());
  }

  IEnumerator MyCoroutine()
  {
    WaitForSeconds waitForSeconds = new WaitForSeconds(5);

    // Wait for five seconds

```

```
        yield return waitForSeconds;

        // Then wait for five more seconds

        yield return waitForSeconds;

        // Then do this.
    }
}
```

Wait for End of Frame / Wait for Fixed Update

```
using UnityEngine;
using System.Collections;

public class CoroutineExamples : MonoBehaviour
{
    private void Start()
    {
        StartCoroutine(MyCoroutine());
    }

    IEnumerator MyCoroutine()
    {
        // Waits until the end of the frame.
        yield return new WaitForEndOfFrame();

        // Waits until Fixed Update is called.
```

```
17.       yield return new WaitForFixedUpdate();
18.    }
19. }
```

Yield for another Coroutine

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class CoroutineExamples : MonoBehaviour
5.  {
6.     private void Start()
7.     {
8.        StartCoroutine(MyCoroutine());
9.     }
10.
11.    IEnumerator MyCoroutine()
12.    {
13.       // Do something
14.
15.       // Wait for this coroutine to finish...
16.       yield return StartCoroutine(MyOtherCoroutine());
17.
18.       // And then wait for this one
19.       yield return StartCoroutine(AnotherCoroutine());
20.
21.       // Do something else
22.    }
```

```
23.
24.    IEnumerator MyOtherCoroutine()
25.    {
26.        yield return new WaitForSeconds(5);
27.    }
28.
29.    IEnumerator AnotherCoroutine()
30.    {
31.        yield return new WaitForSeconds(5);
32.    }
33. }
```

Demonstration example

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class CoroutineExamples : MonoBehaviour
5.  {
6.      [SerializeField] AudioSource audioSource;
7.
8.      private void Start()
9.      {
10.         FadeAudioSource(0);
11.     }
12.
13.     public void FadeAudioSource(float targetVol)
14.     {
```

```
15.       StopAllCoroutines();
16.       StartCoroutine(FadeVolume(targetVol));
17.    }
18.
19.    IEnumerator FadeVolume(float targetVol)
20.    {
21.       while (audioSource.volume != targetVol)
22.       {
23.          audioSource.volume = Mathf.MoveTowards(audioSource.volume, targetVol,
       0.5f * Time.deltaTime);
24.          yield return null;
25.       }
26.    }
27. }
```