Code Structure & Standards

▼ Referencing

▼ How to do it Properly

- Follow the rules of the License that comes with the code you are copying
  - Give Credit ALWAYS regardless if the creator says so or not
  - Follow the rules of the License
  - For more information visit the links below:
    - https://www.patrick-wied.at/blog/how-to-correctly-use-code-you-didnt-write
- Credit the original developer properly by referencing the developer and source in your ReadMe
  - You can also add comments in your code where you use the algorithm or feature you researched

▼ Referencing Guidance

▼ Referencing an Algorithm from GitHub, etc

- Make sure you follow the rules of the License if there is one
- You can always give credit to the developer by adding a comment above the class that uses the algorithm or a variation there of

```
//  Title: GraphicsDrawer source code
//  Author: Smith, J
//  Date: 2011
//  Code version: 2.0
//  Availability: https://www.graphicsdrawer.com/
```

▼ Referencing Code from StackOverflow, Blogs, etc

▼ Example 1 - Method Comment

```
/// <summary>
/// Rotate Tile on Z Axis using Lerp<br/>
/// Implementation for Lerp taken from:<br/>
/// https://gamedevbeginner.com/the-right-way-to-lerp-in-unity-with-examples/#right_way_to_use_lerp<br/>
/// </summary>
/// <param name="a">Start Value</param>
/// <param name="b">Target Value</param>
/// <returns></returns>
private IEnumerator RotateTileRoutine(float a, float b)
{
    // PRE COROUTINE - SETUP FOR COROUTINE
    float elapsedTimed = 0f;
    float zRotation = 0f;

    while (elapsedTimed < TimeToLerp)
    {
        // COROUTINE
        float t = elapsedTimed / TimeToLerp;
        t = t * t * (3f - 2f * t); // Easing Function
        zRotation = Mathf.LerpAngle(a, b, t);

        transform.rotation = Quaternion.Euler(0f, 0f, zRotation);

        elapsedTimed += Time.deltaTime;

        yield return null;
    }

    // POST COROUTINE - AFTER THE COROUTINE EXECUTED
    transform.rotation = Quaternion.Euler(0f, 0f, b);
    yield return _delay;

    StartCoroutine(RotateTileRoutine(b, a));
}
```

Example 2 - Inline Comment

```
/// <summary>
/// Rotate Tile on Z Axis using Lerp<br/>
/// </summary>
/// <param name="a">Start Value</param>
/// <param name="b">Target Value</param>
/// <returns></returns>
private IEnumerator RotateTileRoutine(float a, float b)
{
    // The Right Way to Lerp in Unity with Examples by John
    // https://gamedevbeginner.com/the-right-way-to-lerp-in-unity-with-examples/#right_way_to_use_lerp<br/>

    // PRE COROUTINE - SETUP FOR COROUTINE
    float elapsedTimed = 0f;
    float zRotation = 0f;

    while (elapsedTimed < TimeToLerp)
    {
        // COROUTINE
        float t = elapsedTimed / TimeToLerp;
        t = t * t * (3f - 2f * t); // Easing Function
        zRotation = Mathf.LerpAngle(a, b, t);

        transform.rotation = Quaternion.Euler(0f, 0f, zRotation);

        elapsedTimed += Time.deltaTime;

        yield return null;
    }

    // POST COROUTINE - AFTER THE COROUTINE EXECUTED
    transform.rotation = Quaternion.Euler(0f, 0f, b);

    yield return _delay;

    StartCoroutine(RotateTileRoutine(b, a));
}
```

Example 3 - Inline Comment

```
// (c) 2022 [copyright holder]
// This code is licensed under [license] (see LICENSE.txt for details)
// (source name)[link]
```

Referencing in the Readme

```
# Project Name
Short Description

# References
## Lerping in Unity - by John
[The Right Way to Lerp in Unity with Examples](https://gamedevbeginner.com/the-right-way-to-lerp-in-unity-with-examples/#ri
```

Referencing your own work

```
// (c) [year] by [copyright holder]
// This code is licensed under [license] (see LICENSE.txt for details)
// Link
```

Code Standards and Structure

Standards

▼ Personal Standard of Uee

```
using System.Collections;
using System.Collections.Generic;
using System.Data.SqlClient;
using UnityEngine;

namespace Adventure
{
    /// <summary>
    /// Interfaces should always Start with the Letter I
    /// </summary>
    public interface IMyInterface
    {
    }

    /// <summary>
    /// You can suffix abstract classes with the word Base
    /// </summary>
    public abstract class SomethingBase
    {
    }

    /// <summary>
    /// Class names should always be PascalCase
    /// </summary>
    public class PersonalStandard : MonoBehaviour
    {
        #region VARIABLES

        // Public fields could be either PascalCase or camelCase. Choose one design and stick with it
        public string MyPublicField;
        // Private fields should be either _camelCase or just camelCase
        private string _myPrivateField;

        // Creating Constant Fields
        public const int MIN_AGE = 18;

        #endregion

        #region PROPERTIES

        /// <summary>
        ///
        /// </summary>
        public string MyPrivateField
        {
            get => _myPrivateField;
            set => _myPrivateField = value;
        }

        #endregion

        #region UNITY METHODS

        #endregion

        #region METHODS

        /// <summary>
        /// Method names should always be PascalCase
        /// </summary>
        public void MyMethod()
        {
        }

        /// <summary>
        /// Parameter names should always be camelCase
        /// </summary>
        /// <param name="parameterA"></param>
        /// <param name="parameterB"></param>
        public void MyMethodWithParameters(string parameterA, string parameterB)
        {
        }

        public void MyMethodWithDefaultParameters(string parameterA, string parameterB = "ParameterB")
        {
        }

        public void OneLinerMethod() => print("Hello");

        public void OnLinerMethodWithParameters(string parameterA) => print(parameterA);

        public void WorkingWithDisposableTypes(string connectionString)
        {
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                string query = "";
                // use the connection and the stream
                using (SqlCommand command = new SqlCommand(query))
                {
                }
            }
        }

        #endregion
    }
}
```

▼ Structure

```
using System;
using System.Collections;
using UnityEngine;

namespace Adventure.Traps
{
    public class RotateTrap : MonoBehaviour
    {
        #region FIELDS

        // As the region name suggests, this is where you would put all of your
        // public, private, protected Fields.
        // You should also separate your Public fields from your Private and
        // Protected Fields.
        // For example by placing your public Fields first,
        // then your private/protected there after.

        [Header("Rotation Properties")]
        public Vector2 Rotation = new Vector2(0f, 180f);

        [Header("Lerp Properties")]
        public float TimeToLerp = 3f;
        public float DelayBetweenRotations = 1f;

        private WaitForSeconds _delay;

        #endregion

        #region PROPERTIES

        // This is where all of your Properties for this class would be placed.
        public string Name
        {
          get => gameObject.name;
          set => gameObject.name = value;
        }

        #endregion

        #region UNITY METHODS

        // In the UNITY METHODS region, you would include all of the Unity Event
        // Methods your class would use.

        private void Awake()
        {
            _delay = new WaitForSeconds(DelayBetweenRotations);
        }

        private void Start()
        {
            StartCoroutine(RotateTileRoutine(Rotation.x, Rotation.y));
        }

        #endregion

        #region METHODS

        // The METHODS region is where you would put the main logic for your class.
        // These METHOD regions can be separated into different regions that handle
        // certain tasks, such as BUTTON METHODS, INPUT PROCESSING METHODS, etc

        /// <summary>
        /// Rotate Tile on Z Axis using Lerp<br/>
        /// </summary>
        /// <param name="a">Start Value</param>
        /// <param name="b">Target Value</param>
        /// <returns></returns>
        private IEnumerator RotateTileRoutine(float a, float b)
        {
            // The Right Way to Lerp in Unity with Examples by John
            // https://gamedevbeginner.com/the-right-way-to-lerp-in-unity-with-examples/#right_way_to_use_lerp<br/>

            // PRE COROUTINE - SETUP FOR COROUTINE
            float elapsedTimed = 0f;
            float zRotation = 0f;

            while (elapsedTimed < TimeToLerp)
            {
                // COROUTINE
                zRotation = Mathf.LerpAngle(a, b, SimpleEasing(elapsedTimed / TimeToLerp));

                transform.rotation = Quaternion.Euler(0f, 0f, zRotation);

                elapsedTimed += Time.deltaTime;

                yield return null;
            }

            // POST COROUTINE - AFTER THE COROUTINE EXECUTED
            transform.rotation = Quaternion.Euler(0f, 0f, b);

            yield return _delay;

            StartCoroutine(RotateTileRoutine(b, a));
        }

        #endregion

        #region HELPER METHODS

        // Helper Methods Compliment the methods inside of your METHODS region.
        // They are useful when you want to breakdown a larger method
        // into smaller readable pieces.

        private float SimpleEasing(float t)
        {
            return t * t * (3f - 2f * t);
        }

        #endregion
    }
}
```