Converting A Coroutine To An Async

Preface

The original content is from the author Michael Quinn from Medium.com. It has been modified and updated for use in this wiki.

> This is an **advanced** topic! You should have an understanding of Object Oriented Programming in C# as well as an understanding behind C# Delegates.

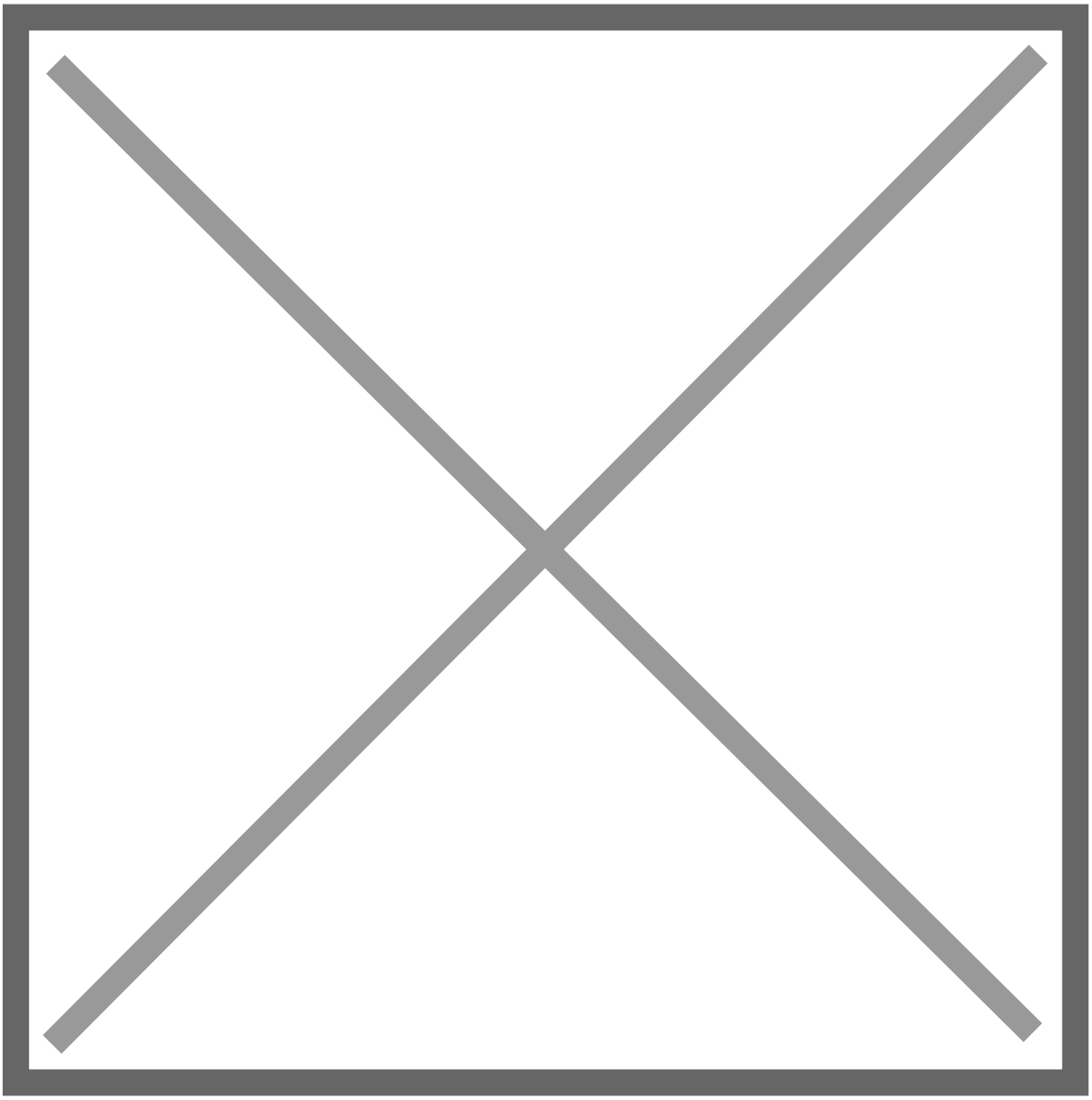
> I could have translated the code below into code snippets, but I found it more important to physically type what's needed for the sake of understanding.

About

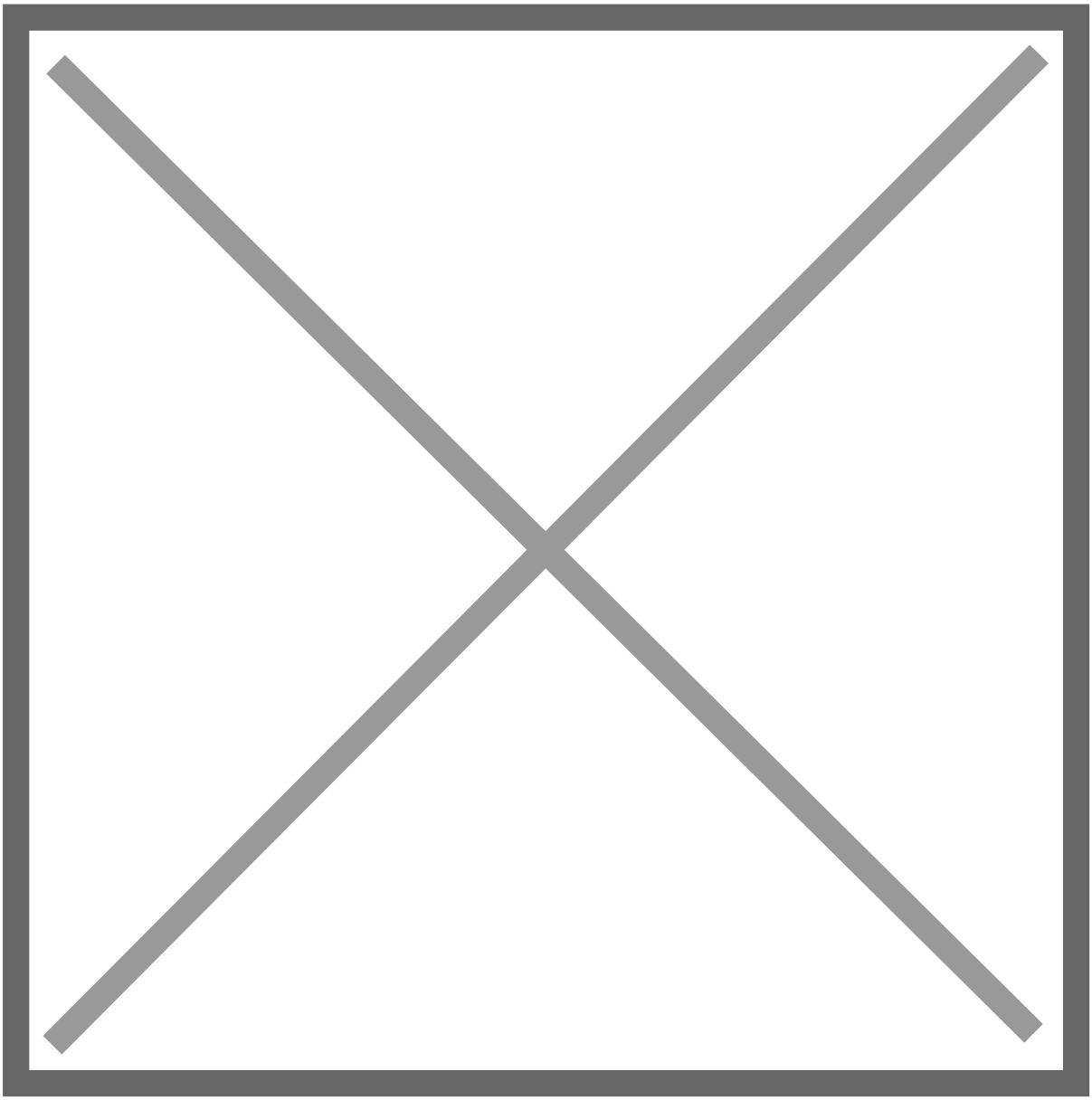
Coroutines are amazing ways to sequence logic together over many frames. Asynchronous programming, or **async**, allows the code to similarly be split over multiple frames, but allows for multithreading which has the benefit of logic being concurrently executed instead of sequentially.

The Coroutine Way

I have a very simple example that rotates cubes for a certain amount of time.

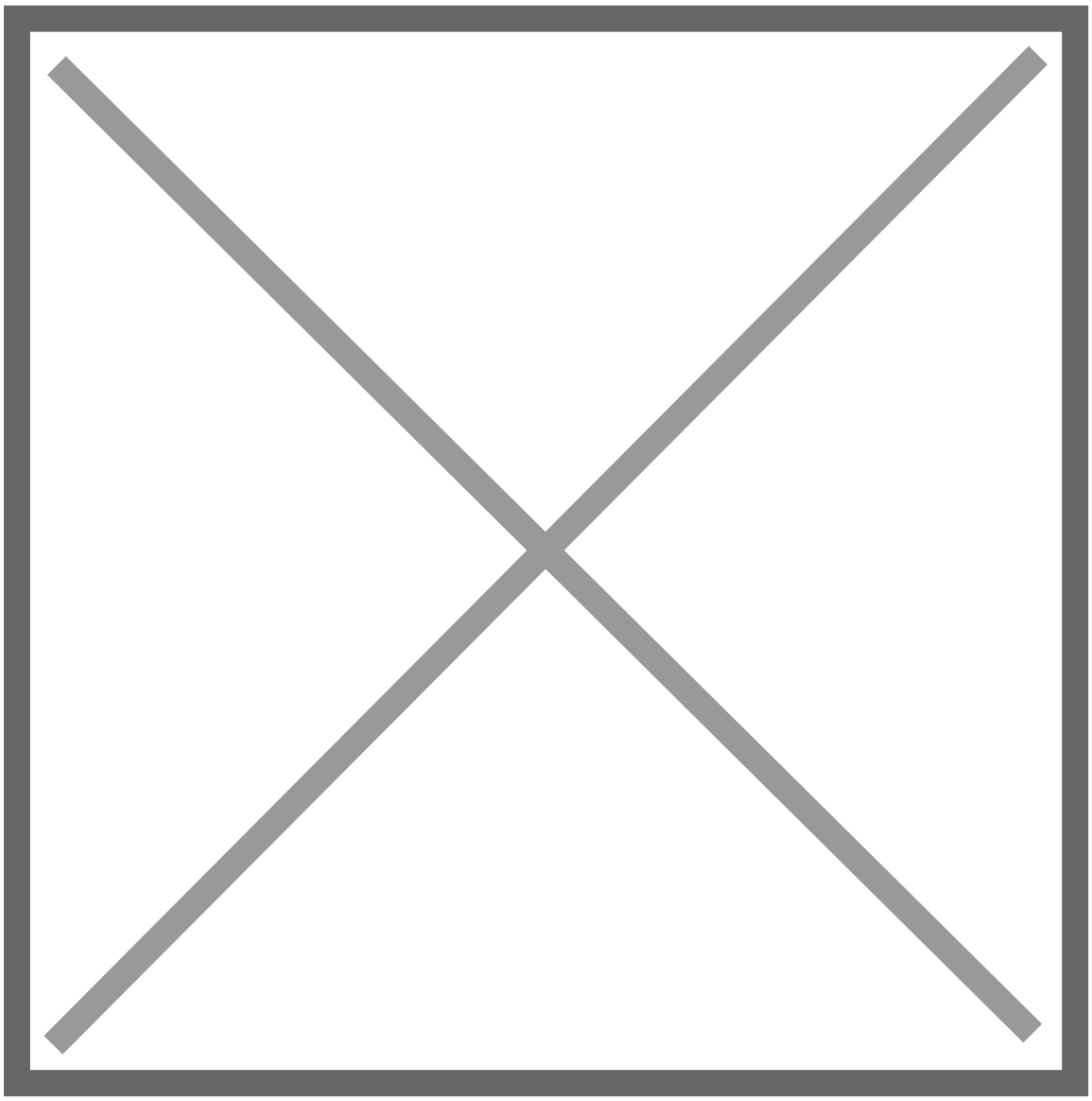


This script will do a loop over all the cubes and trigger a coroutine that will rotate them. Setting this function to run when a button is pressed and we have the following effect.

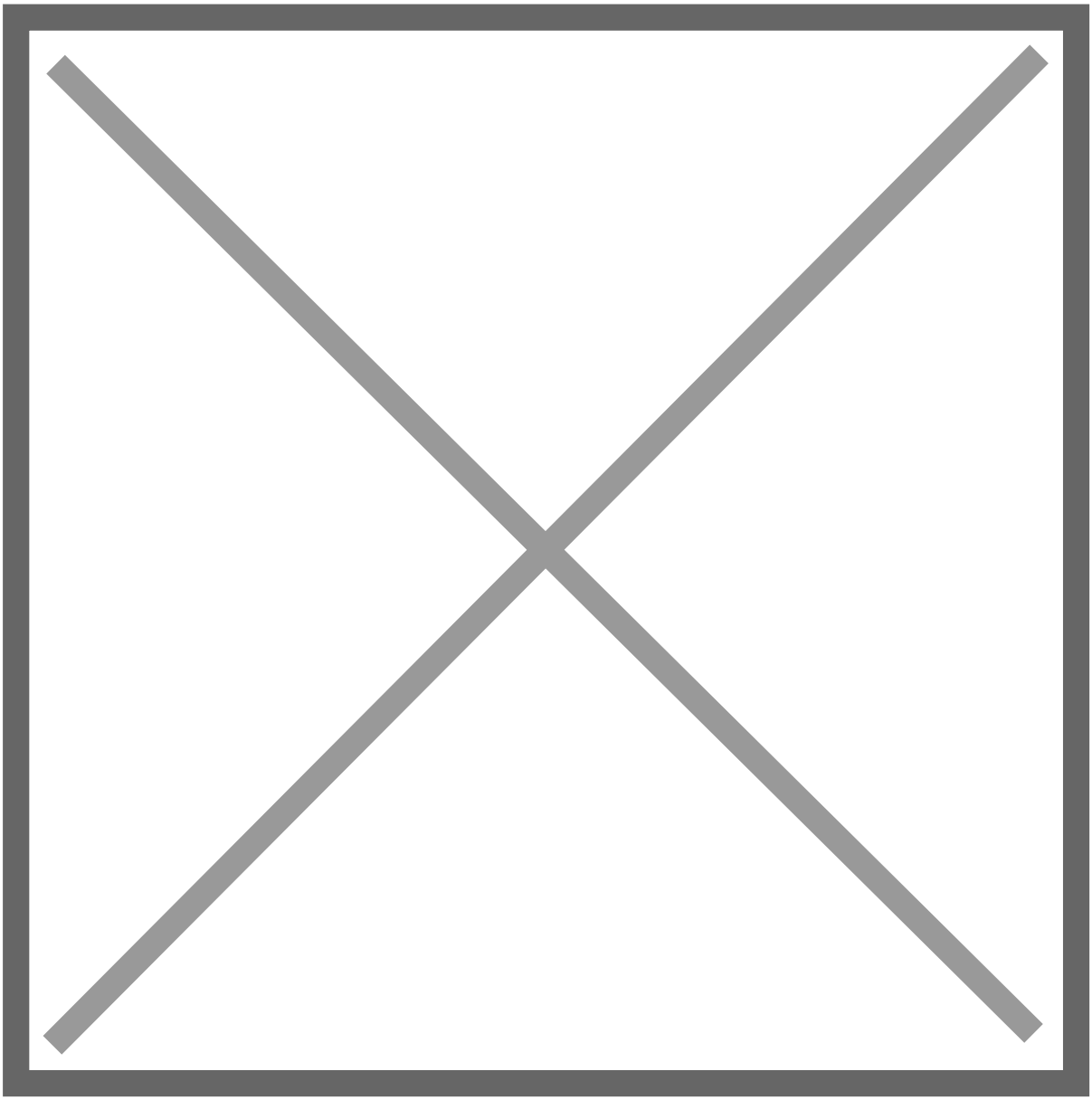


The Async Way

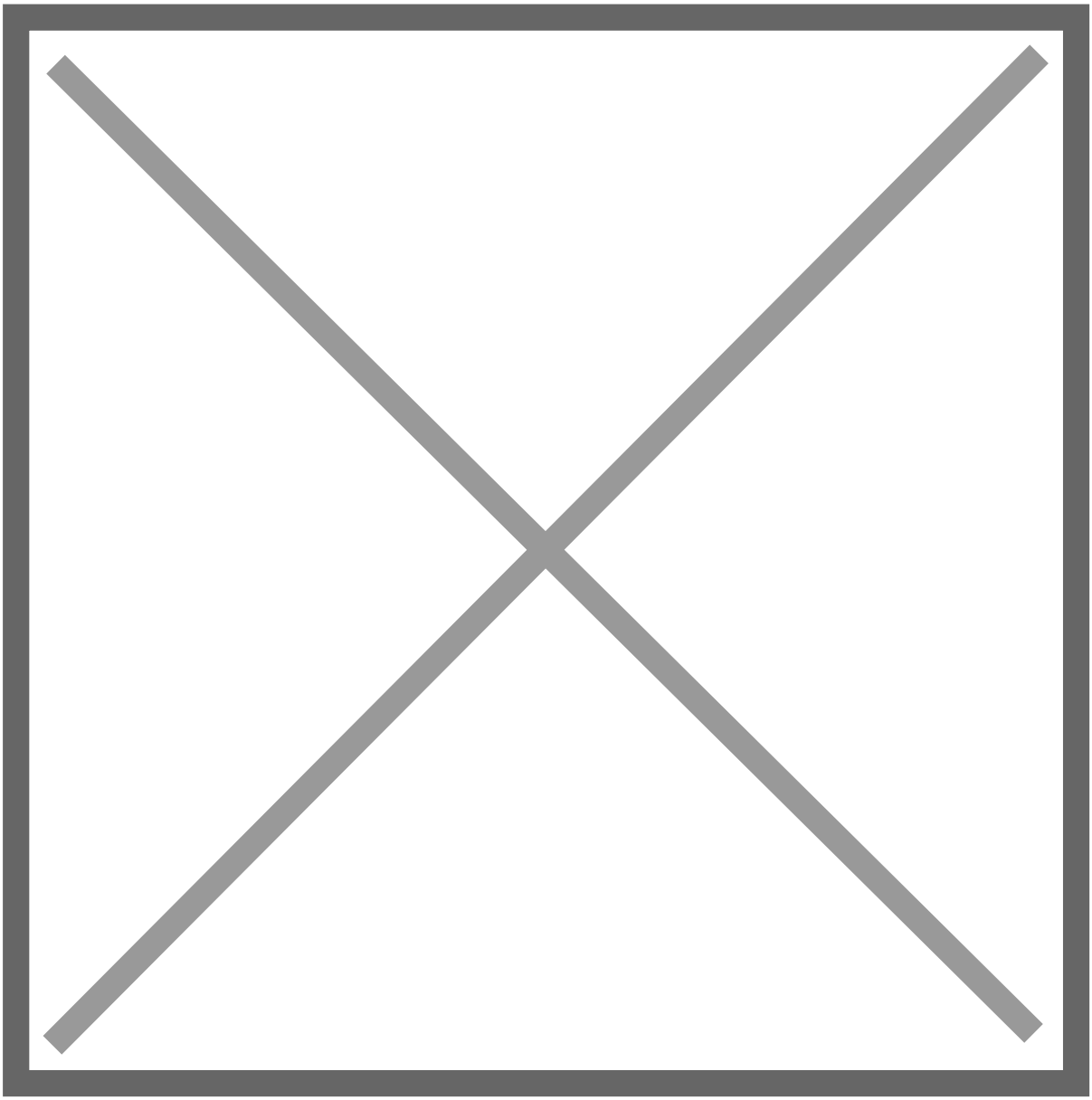
To convert the current method we are going to swap **IEnumerator** to **async** and then change the *yield* to an *await task*.



With the proper using statement, we can simplify the naming and finish with the following code.

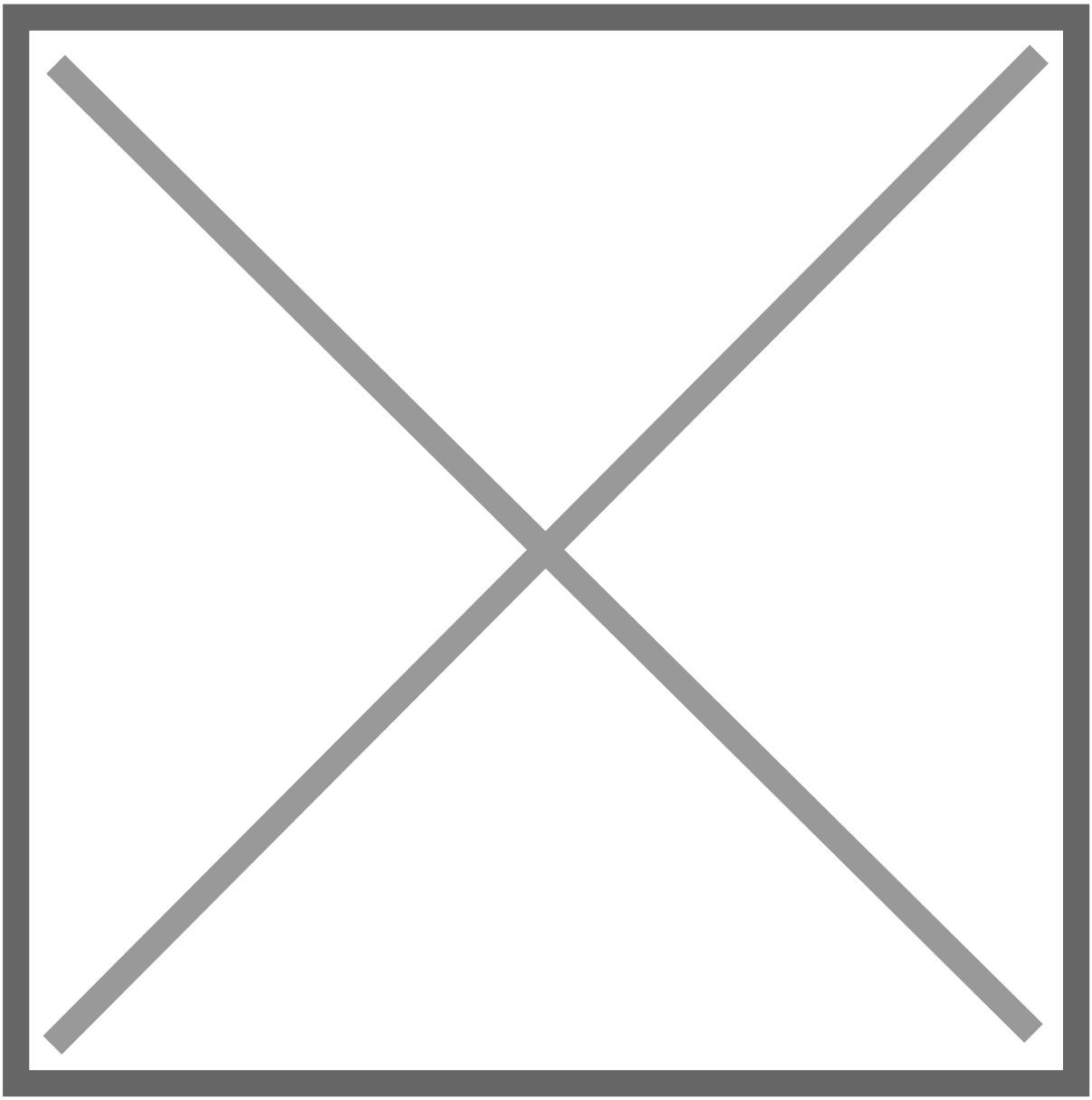


Get ready for the results! They are … exactly the same.



Okay so where is the power in this? The benefits are immense. The very first we may be able to see was that I had to add a return type when converting the function from a coroutine to an async method. Coroutines don’t return values like that.

To get around this shortcoming, I’ve had to do crazy things like this before.



Ugh, yuck. Who wants to read that? With async, everything is far more readable and easier when it comes to return values.

The strongest immediate benefit of async is the power of multithreading. Coroutines run on what is called the **main thread**. The longer it takes the main thread to move, the more lag your application will experience. Async, however, can run on threads besides the main thread. This benefit from async can give a major benefit to an applications performance.

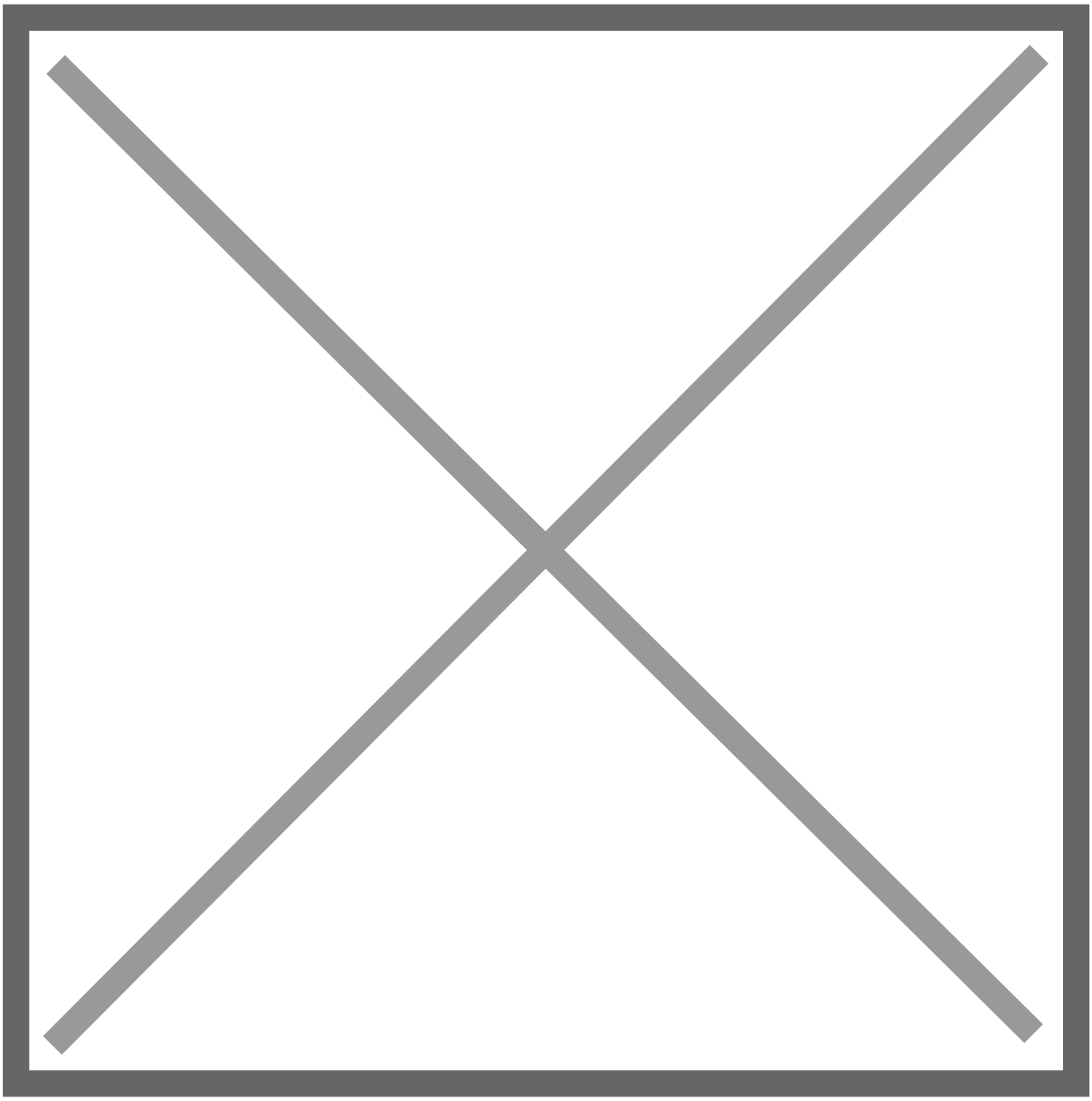
Multithreading is not totally noticeable in this example, but this next benefit is. Async has the ability to run sequentially using the keyword **await**.

But if Coroutines are made for sequential logic, then how is this a benefit? Coroutines *are* made for sequential logic, but they are not made for sequentially executing coroutines.

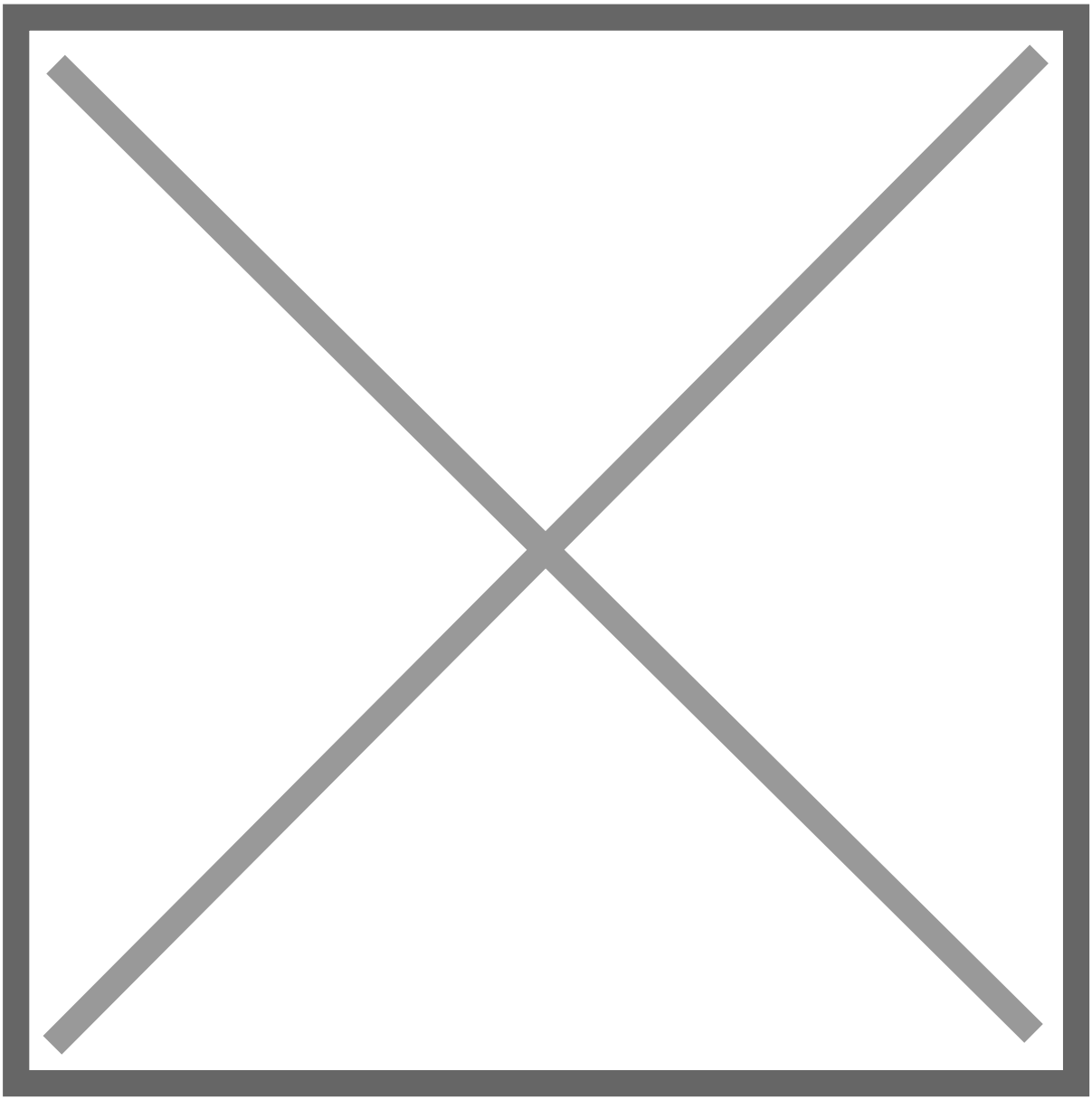
Now, there are ways to sequentially execute coroutines. The absolute simplest way would be to have a **StartCoroutine()** function at the end of a coroutine. This chaining however, will lead to

very difficult code to unpick when it comes time to refactor. A cleaner way to handle this with async methods is using **await.**

Our first step is to change the return type, from **void** to **Task**. Then we can use the await functionality by making the original function also async.

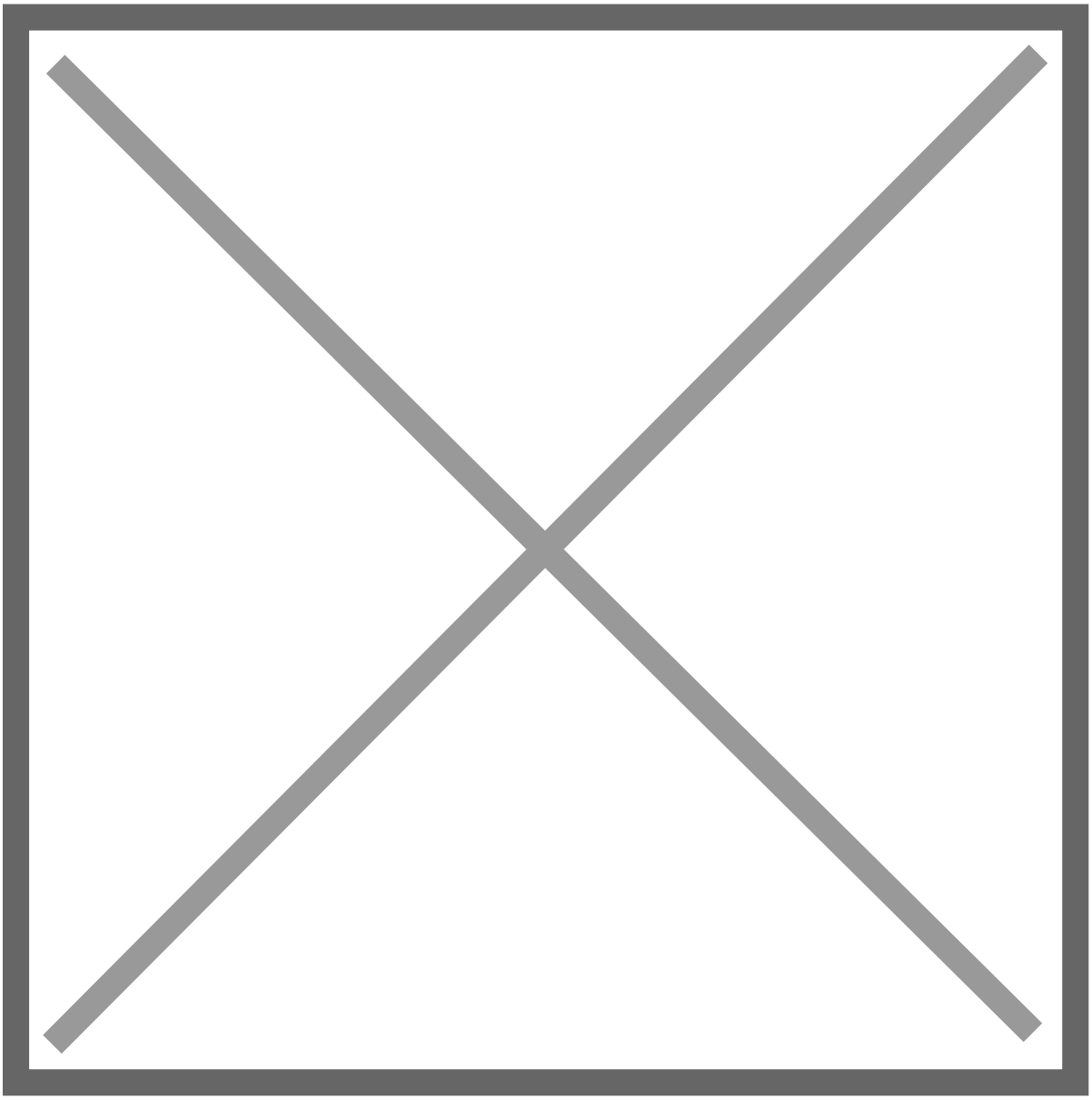


Now when we test this logic, we will see the cubes only rotate after the one before it is finished.

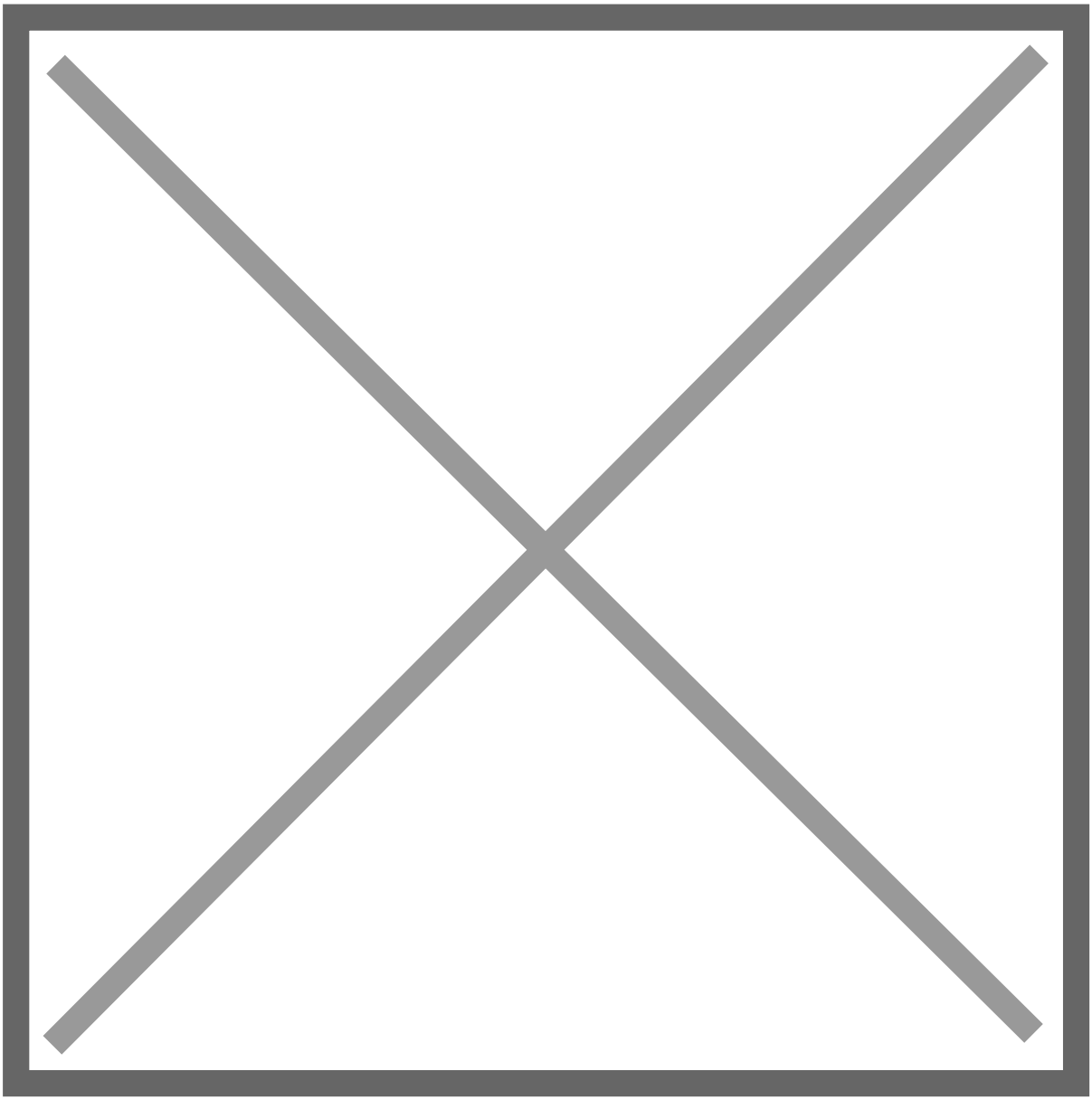


Another benefit, that might convince you to never use coroutines again, is that with async, you can make logic wait until a series of tasks have completed.

To demonstrate this, I’m going to create a simple array of colors, then have the button change ONLY after all the cubes have finished moving.

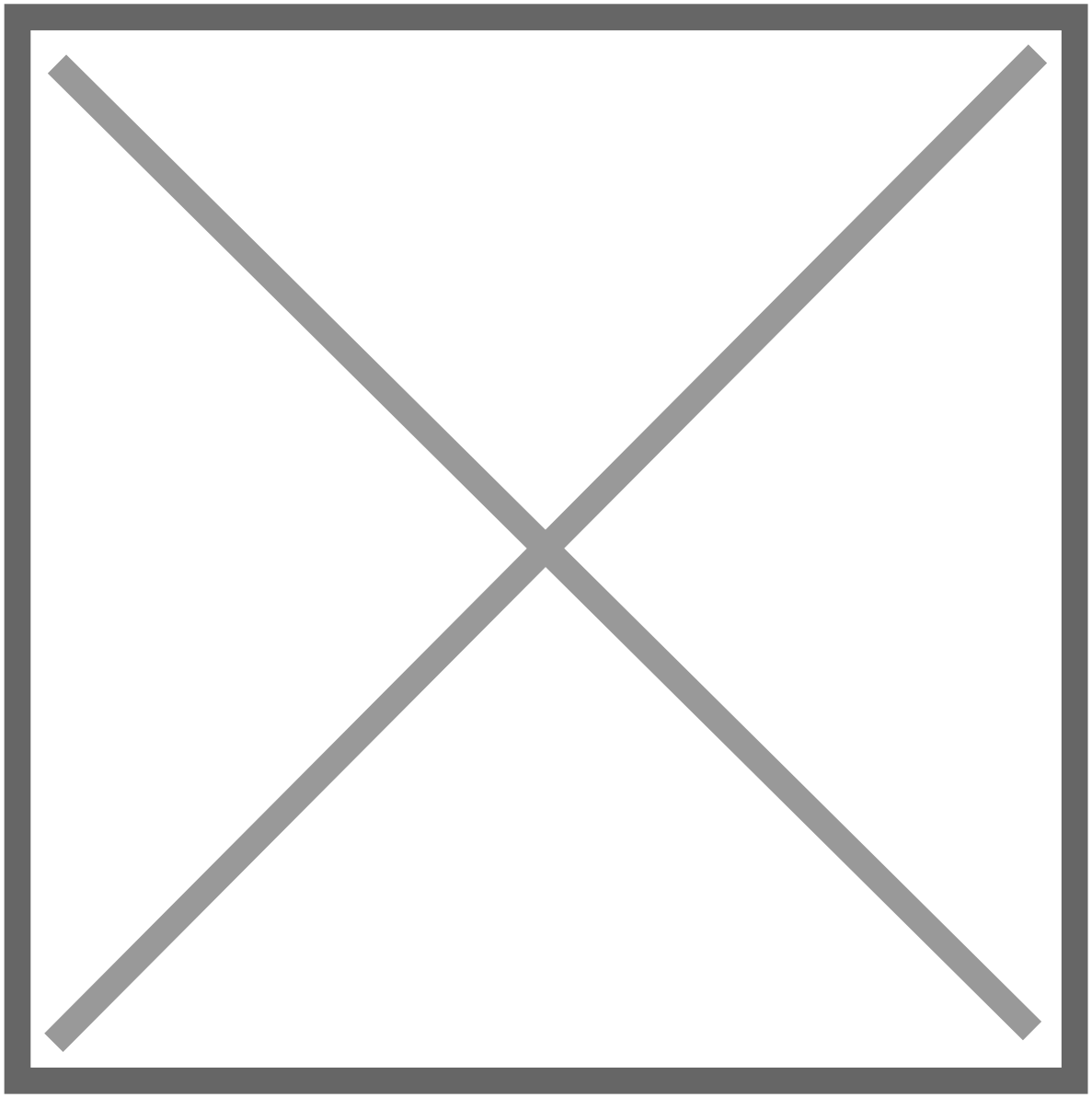


Before we change this logic, let’s see what we have currently.

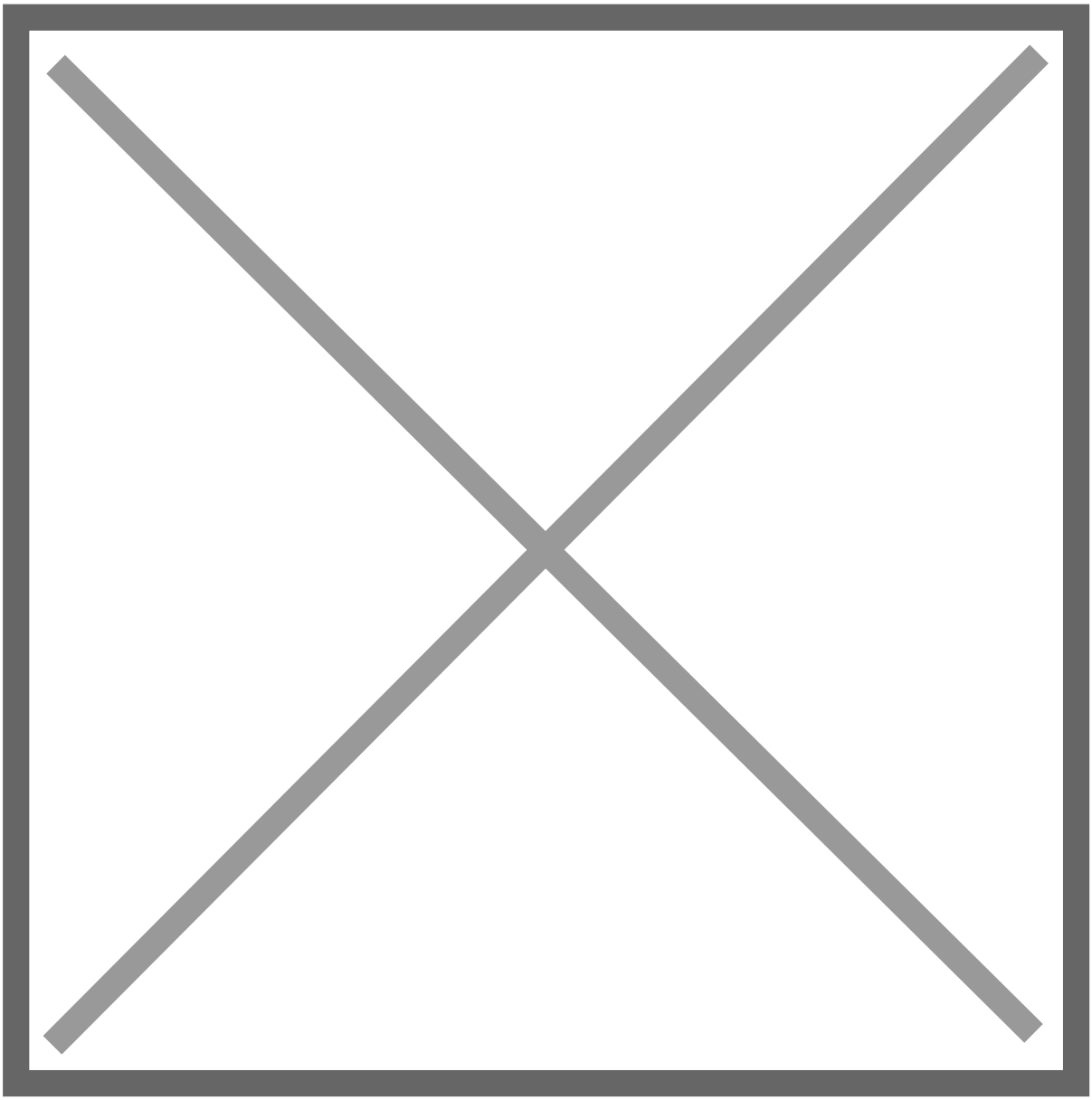


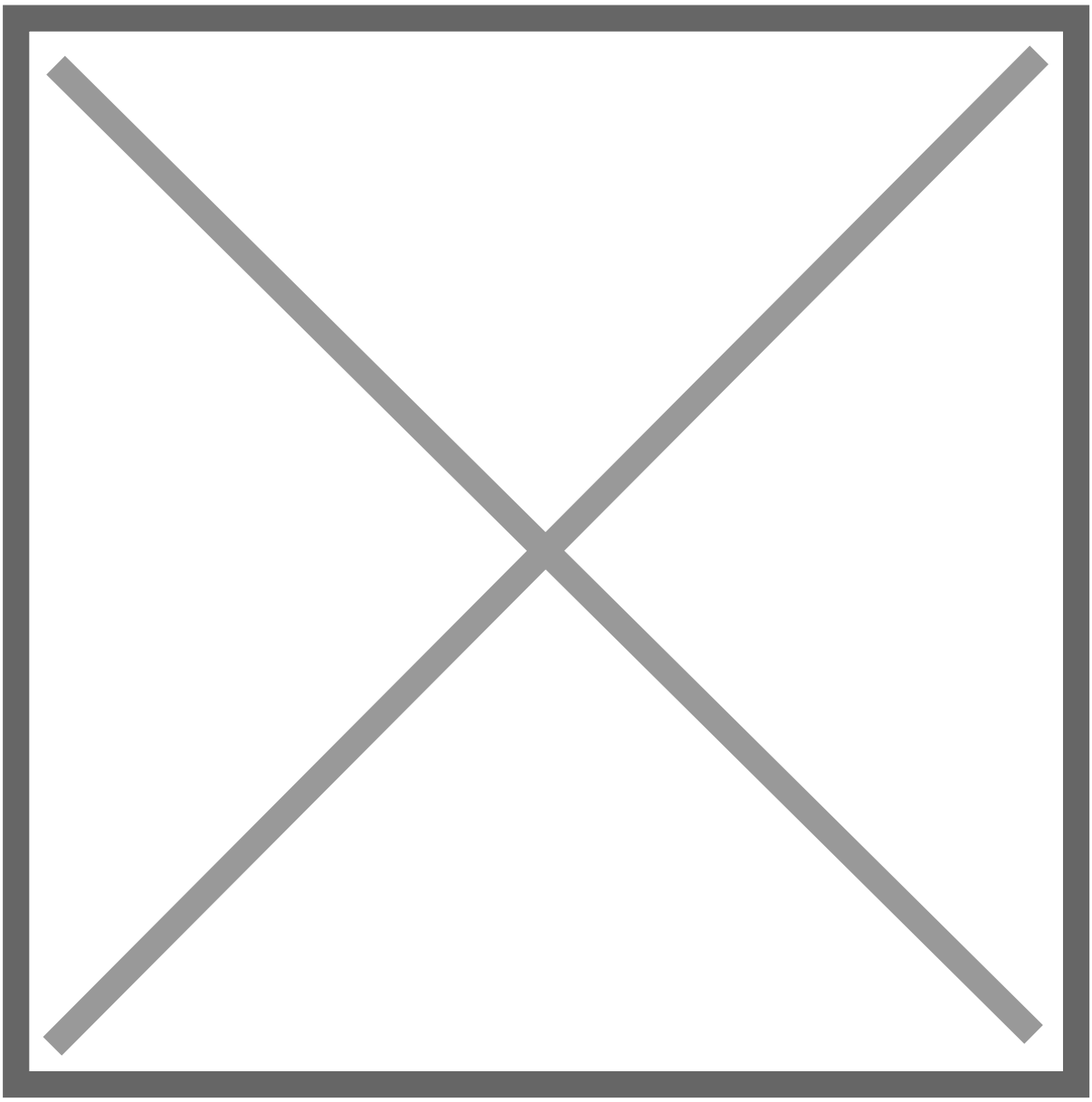
Okay, so now with async, we will be able to change the behavior around so that all the cubes will rotate at the same time, but the button will still only change after they are all finished.

We are going to want to create a list of tasks that we can add to, then check to see if all those tasks have finished before changing the color of the button.



The entire code will now look like the following.





That’s pretty cool right there.

Takeaway

This article has only scratched the surface of the power of asynchronous programming but it has definitely convinced me to make the switch from coroutines to async.

Revision #1
Created 15 July 2024 14:39:48 by Wes
Updated 15 July 2024 14:46:45 by Wes