Programming Assignment 4.3

**Assignment Description**

In this module, you learned about developing Unity classes; of course, you've been adding functionality to Unity classes (scripts) lots in the previous modules and courses in the Specialization!

In this assignment, you'll implement a Mover class that moves a game object back and forth.

**Why do we care?**

This assignment gives you more practice implementing functionality in Unity, and it's also the first programming assignment that actually uses Unity in the autograder.

**Getting Started**

Download the appropriate zip file for your OS below and unzip the file somewhere on your computer.

**Windows Programming Assignment 4.3 Materials** ZIP File

**Mac Programming Assignment 4.3 Materials** ZIP File

The contents of the zip file is different from what you've seen in the past, because it contains both Unity materials and a console app project for the autograder.

Unzip the Unity materials zip file somewhere on your computer and open the Unity project in Unity.

**Starting in Unity**

All your work for this assignment is in the Mover.cs file; don't change ANY of the code in the other files I've provided to you.

For this assignment, you'll be adding functionality to the **Start** and **Update** methods in the provided **Mover** class stub to implement the required functionality.

What is the required functionality? When the game starts, the game object the Mover script is attached to starts moving to the right. After (approximately) 1 second, the game object starts moving to the left instead. After another (approximately) 1 second, the game object starts moving to the right again instead. The game object keeps changing direction every second until the game is stopped.

The Unity materials I provided to you contains a stub for the **Mover** class, but you still have lots of work to do to finish implementing that class (script).

In the Mover script I provided, add code to the **Start** method to add a Timer component, set its duration to the **TimerDuration** constant I provided, and run the timer. Add code to the **Update** method to check if the timer has finished; if it has, multiply the **directionMultiplier** field by -1 and run the timer again. After the code you just added, add code to move the game object using the **directionMultiplier** field, the **MoveAmountPerSecond** constant I provided, and **Time.deltaTime**.

Note: The big idea behind the **directionMultiplier** field is that if it's set to 1 the game object will move to the right and if it's -1 the game object will move to the left. Given that idea, you should be able to figure out how to use the **directionMultiplier** field when you're moving the game object.

Run the game. The game object the Mover script is attached to should move back and forth on the screen, changing direction approximately every second.

**Moving Over to the Console App**

Now that you have your code working in Unity, it's time to move it over into the console app and make sure it will all work fine in the automated grader. Unzip the autograder materials zip file somewhere on your computer and open the Visual Studio solution.

The Visual Studio project I provided to you has a stub for the **Mover** class; you'll replace that file with the one you developed in Unity soon. The project I provided also includes a fully-implemented **Program** class that tests the classes you've implemented.

Although you can build and run the Visual Studio solution at this point, it will crash on all the test cases except for Test Case 1. Following the steps in the next paragraph will fix that.

Copy the following file, AND ONLY THE FOLLOWING FILE, into the same folder as the Program.cs file in the console app project: Mover.cs. Confirm that you want to overwrite the Mover.cs file when prompted to do so. Do NOT copy the Timer script over; doing that will break the automated grader.

Build the solution in Visual Studio. You should now be able to run the solution as discussed below.

**Required Output Format**

The **Program** class I provided handles all the output correctly. The output is simply Passed or FAILED for each test case that's selected by the user input.

**Running Your Code**

Because of the code I included to work with the automated grader on Coursera, when you run your program the command prompt window will open and it will sit there doing nothing. To make your code run, type in a test case number (1 through 10, inclusive) and press the <Enter> key; your code should then run so you can check your output.

For example, your input could be

**3**

to run the third test case.

You can actually run your code again if you want to by typing in a test case number and pressing the <Enter> key again. When you’re ready to stop running your code, type q (for quit).

Here's what running the code multiple times with different inputs should look like (though none of your test cases should fail!). The first line is the test case number, the second line is your output line, and so on:

**1**

**Passed**

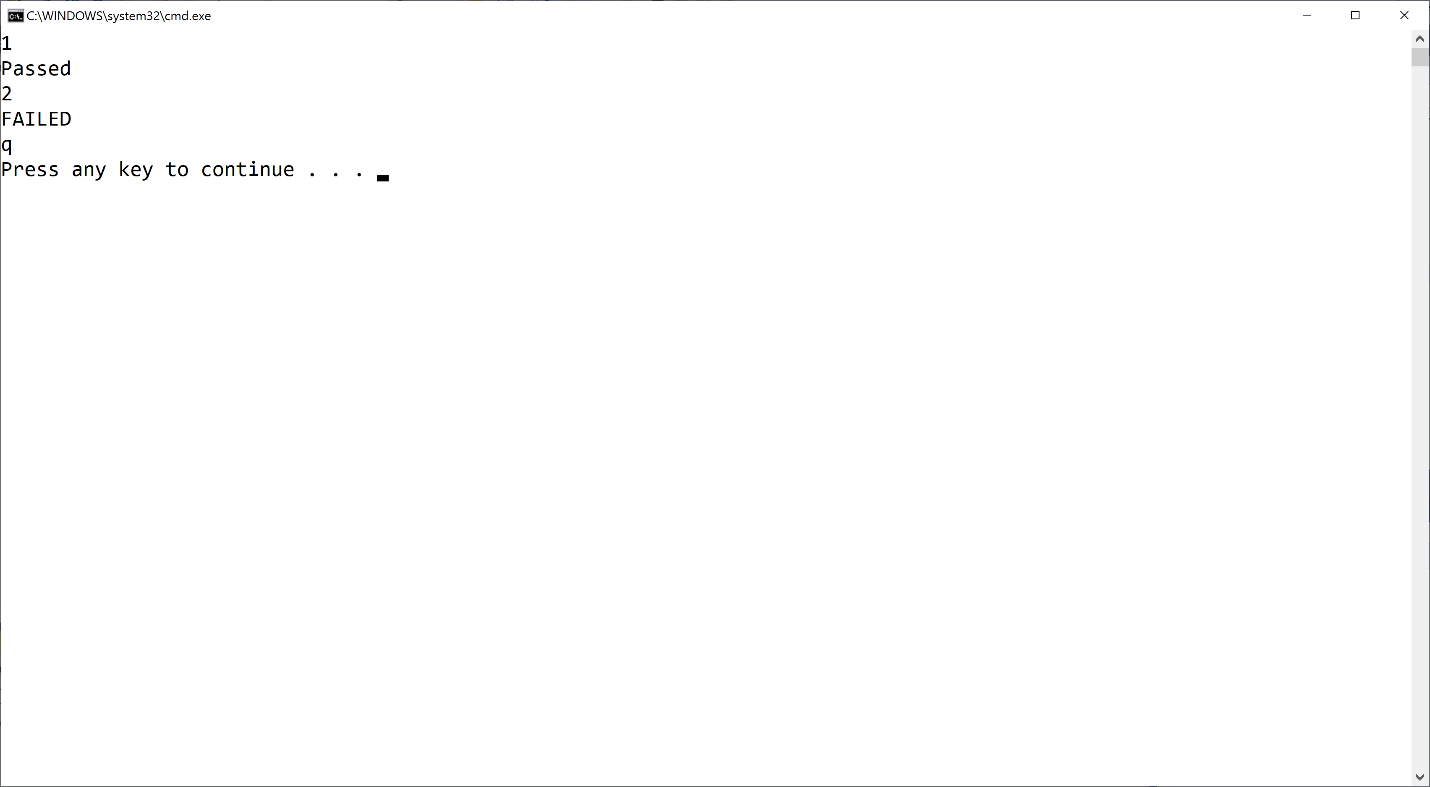
**2**

**FAILED**

**q**

**Important Note:** The Coursera formatting makes it look like there's a blank line between each of the lines above, but there's not. The above output should be exactly 5 lines of input/output.

The image below shows my console window when I run the code multiple times as described above:



If your output doesn't match the image above EXACTLY (no extra words, characters, spaces, or blank lines) you'll fail all the test cases in the automated grader. Of course, you shouldn't get any FAILED output when you run your code!

Mac users: You may or may not have an extra blank line after the q. That's fine.

**Test Case Inputs**

The automated grader uses the set of test cases that are built into the Program.cs file I provided to you.

**Common Problems**

Historically, lots of people post about grading errors on this assignment (because their code is wrong). Here are a couple things you should check:

1. You have to start with the assignment materials I gave you and add your code as described above. The code I gave you includes the appropriate structure for the automated grader to work. If you don't do this, you'll almost definitely fail all the test cases in the automated grader
2. I give you an example in this assignment (and all assignments) for what output you should get when you run the code multiple times. Be sure to try that example; if your code doesn't generate output exactly as shown in the example, your code isn't working properly

**Submitting Your Solution**

Go to the LMS and submit your project to **Assignment 4.3**