Coding Test

This test is used to get a feel for your coding experience and style. There are no points to be earned, however, a time estimate is given to provide you with the means to measure how much time and effort you should invest in a single task.
If anything is unclear or if you're uncertain about a specific aspect, don't hesitate to ask rather than working towards an uncertain goal.
You will need a development environment running .Net as well as a Unity Version of your choice (preferably Unity 2020.2 or above).

Refactor

Goal

Investigate the project "RefactorTest". Run it in your .Net Environment and try it out (remember that you might have to rebuild the solution).

1. Describe what the goal of the code is.
2. Describe potential issues that you see with that implementation
3. Describe possible solutions on how to solve some or all of the aforementioned issues
4. Refactor the given code and provide a solution to some or all of your proposed changes.

Time Estimate

6h

Additional

If interested you can try to also investigate the Singletons used in the project and apply solutions of how to solve any issues you might perceive there (if not already done so).

Bezier

Goal

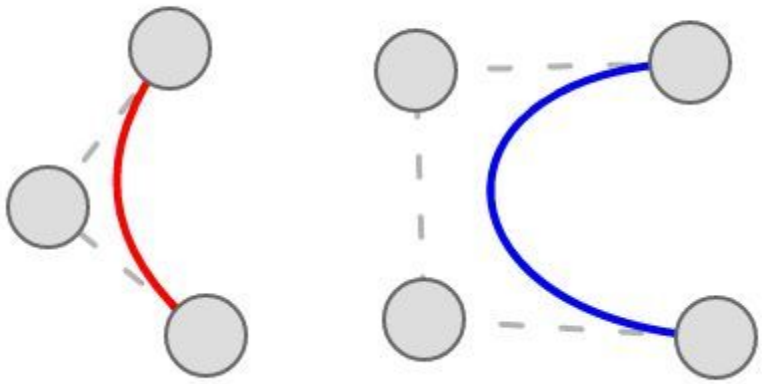
Create a Unity Project "Bezier" as well as a Scene that includes the following elements:

- A root Object
- On the root Object should be a LineRenderer
- 3 - 4 GameObjects on a 2D plane as children of the Root object

Write a script (placed on the Root) that evaluates a Bezier-curve between all the points given in the following fashion:

1. The first and last points are the Start and End points of your curve
2. The points in between are intermediate points influencing your resulting curve
3. Create distinct points describing your resulting curve and apply them to your LineRenderer.

As a reminder here are some examples of a quadratic and a cubic Bezier-curve. Your solution should look similar to this:



Time Estimation

6h

Additional

Instead of doing a quadratic or cubic interpretation you can instead try to do a recursive version that evaluates 3 - X GameObjects and creates a Bezier curve between them.

The Ticker

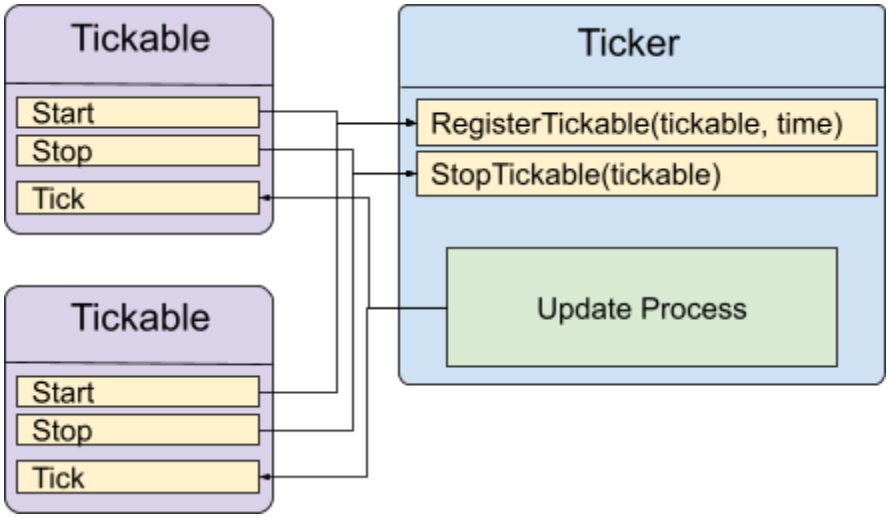
Goal

Write a Ticker-System. Other systems should be able to register themselves with a given time-amount. The Ticker then calls a “tick”-function on a registered system every time the given amount of time has passed.
The idea behind this is to basically have an update-system which doesn’t call it’s update-functions every frame and is also independent of Unity-Update().

Time estimate

6h

Implementation



System Overview

The Ticker-System should make use of Coroutines to wait before calling registered Tickables.

Additional

Make a test: Write a MonoBehaviour that simply adds Time.deltaTime in its Update()-function to a timer-variable. Also, start a Coroutine that does the same by waiting for x-seconds and adding that seconds to its own timer-variable (in a loop).
Observe the variables (maybe in a Debug.Log() or whatever you prefer).
Describe your observation and discuss why that specific output is created.

Mover

Time estimate

8h

Goal

Design and implement a system that allows for a given unit to handle different type of movements. A designer or artist should be able to simply set a specific movement type and Movespeed and the unit should act accordingly.
Your system should support the MoveTypes:

- Walk
- Fly

Implementation

Make use of the provided project labelled **Entity Movement**.
The MainScene contains a simple level, a Unit and a Target. On play, the unit should go through some sort of setup and then follow its movement strategy to try to reach the target. The actual movement implementation is not that important, however, you can use the following table which defines the movement behaviour (you can ignore the Teleport-type for now). Since the actual implementation is not important, it is not necessary that each of these move types actually reaches the target (as long as they at least try to get there). Meaning, it is ok for the Walk-Type to continuously get blocked by the obstruction in the middle and never reach the target. If you want you can still implement some proper pathfinding (it will not factor into any sort of evaluation, however).
If the unit reaches the target it should stop.

Walk	Tries to reach the target with the shortest path possible. Is blocked by collision with elements on the ground.
Fly	Tries to reach the target with the shortest path possible. Is blocked by elements in the air.
Teleport	Waits a short amount of time and then immediately transitions to target.

There is no base-code given in this case so feel free to come up with a system of your own, note however that the scene setup is deliberate (collider setup, physics layers).

Additional

Now imagine that after some time, the team decides to add a new move-type called “Teleport” (as described above).
Could your system handle adding this new type of movement, complete with the actual movement handling, without having to change much to none of your original code?
If your code can handle that, describe how it achieves that. How could you refactor your code to be able to do sth like that in the future?