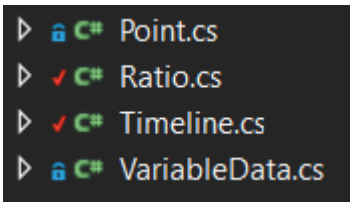
Introduction

This project was made using c# in the unity engine. This was chosen due to the easy ability to not only produce gui and interactive 2d environments but the amount of places that it can be used within a web browser application. I also personally as a programmers am most comfortable working with c# so it is the best IDE for me because it relies on that ability.

Summary of all techniques

- parameter passing
- for loop
- method returning a value
- encapsulation of private methods that work on public attribute of a "template" class
- making an list of objects
- simple and compound selection (if/else)
- error handling (for example catching a divide by 0 error, or a null pointer while using an array of object…)
- Option pane generation for communicating with the user
- GUI tabs
- use of some other specialized library you imported

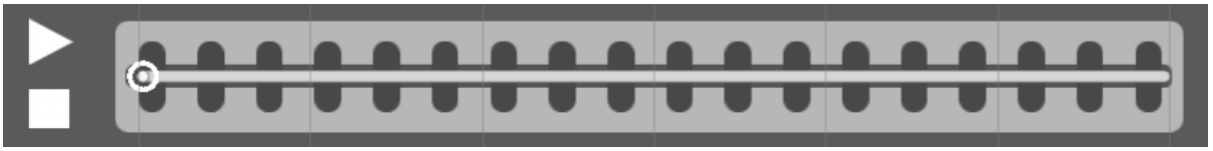
Structure of the the program



This was broken down into a few different classes in order to make the coding process as simple and efficient as possible. The two main sections of the gui - the timeline and the variables - was split in order to make the code easy to read and in order to separate the different logic that is done between them.

The other two classes in the architecture are the ratio and point objects which are used regularly to track data for the user. This makes it quite easy to use already calculated data and just apply it to wherever I need it to go.

The image above displays part of the overall program which holds the input and the output data. The information is taken using the input velocity slider bar and the two gear teeth amounts and this information is finally calculated into the bottom set. Through the calculations and the input techniques such as parameter passing, for loop, simple/compound selection, and etc are used.



Above is the timeline that has its own dedicated class like the data calculations. This was not initially in the clients requirements for the program but it turned out to be something useful in practice and was actually kept because it was actually a convenience. Not all the code for the class is shown in the figure below but the main logic of it is there.

```
private void Update()
{
    if(active)
    {
        timelineSlider.value += Time.deltaTime;
    }
    else if(play && pointsList.Count != 0)
    {
        timelineSlider.value += Time.deltaTime;

        if(Time.time - startTime >= pointsList[index].GetTime() && index < pointsList.Count) //at each time stamp updates the pivot
        {
            pivot.GetComponent<Rigidbody2D>().angularVelocity = pointsList[index].GetVel();
            index++;
        }
        if(index >= pointsList.Count)
        {
            play = false;
            index = 0;
        }
    }
}
```

The overall logic of it is simple in the larger scheme of things but it works efficiently and is consistent with replaying things based on time despite the computer the user is on. This is because of it taking advantage of Unity's Time.time information which gives the time since the program has activated and allows for time based calculations rather than frame dependent. This also makes it accurately copy what it recorded previously.

It's also important to note that once the timeline is activated or deactivated the variable data class is essentially put on pause in order to avoid the two fighting over the motor section.

Data structures used

Lists were used in order to hold data dynamically along the timeline since there was an inconsistent amount of times where the speed could be changed and could be added at any time. This also consisted of objects since each slice contained multiple different variables to be held instead of just a single float.

Unique algorithms

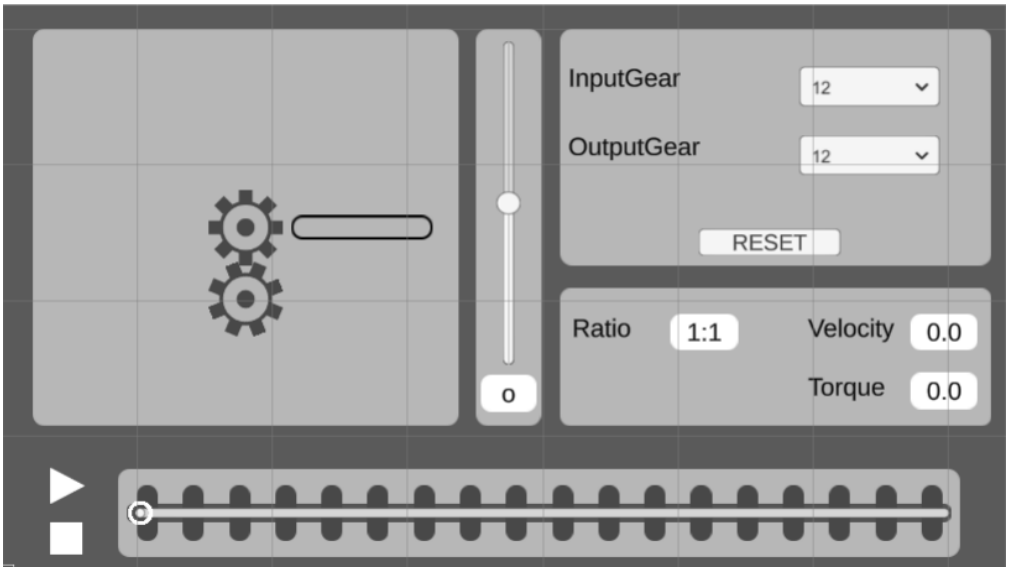
This program has changed dramatically from the start of production and planning due to some of the requests form the client. THe most unique algorithm that is used within the program is the ability to create ratios based on integer values. Though not that unique or difficult to understand the process itself was not taught in class and the outcome works quite well due to the circumstances it is used in.

```
public void Simplify()
{
    // Simplify the sign.
    if (denominator < 0)
    {
        numerator = -numerator;
        denominator = -denominator;
    }

    int gcd_ab = BruteForce(numerator, denominator);
    numerator = numerator / gcd_ab;
    denominator = denominator / gcd_ab;

    ratioString = numerator.ToString() + ":" + denominator.ToString();
}

1 reference
int BruteForce(int num, int den)
{
    int gcd = 1;
    for (int i = 1; i <= num && i <= den; i++)
    {
        if (num % i == 0 && den % i == 0)
        {
            gcd = i;
        }
    }
    return gcd;
}
```

As seen above all that this function does is essentially brute force to see the greatest common factor of the numerator and denominator. Although, as stated before, not terribly complicated as an algorithm as a whole this brute force method was effective for the size of the values being put in. Since they were quite small and not few in numbers this simple solution was optimal and a more complicated algorithm would be too much for what it would be needed for.

GUI

Below we can see how my GUI choice was used in order to make it as easy to understand as possible and quickly usable by its users. The main purpose of the product is to be used to teach those new to robotics the use of motors and gear ratio affects so in order to do so it needed to be visually clean - so to be easy to absorb - but also to allow for easy adjustments to the variables and see the effect live.



Software used

Visual studio → Easy to use with C# withUnity because of built in auto finishing code which makes the process much faster
Unity → All the gui was easily done using this program and allowed for a system that can quickly be built and put online. Also comes with many built in tools and components that makes the process much more streamlined

grade