Graphical user interface, website

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

**Mainpage.xaml.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices.WindowsRuntime;

using Windows.Foundation;

using Windows.Foundation.Collections;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

using Windows.UI.Xaml.Controls.Primitives;

using Windows.UI.Xaml.Data;

using Windows.UI.Xaml.Input;

using Windows.UI.Xaml.Media;

using Windows.UI.Xaml.Navigation;

/\*

\* Daniel Bruce

\* 5/3/2022

\* This program tracks the distance a jellyfish has moved at consistent time intervals (the distance is actually a randomly generated number)

\* A short history of collected data, and unit conversion features are available.

\*/

namespace DataCollector

{

public sealed partial class MainPage : Page

{

private MeasureLengthDevice jellyfishMovement = new MeasureLengthDevice(Unit.Imperial);//default unit is imperial

private Queue<string> measureHistory = new Queue<string>();

public MainPage()

{

this.InitializeComponent();

//the initial data binding

//historyDataBinding();

this.jellyfishMovement.timeData.Tick+= measurmentTextUpdateHandler;

historyListView.ItemsSource = this.measureHistory;

historyListView.Visibility= Visibility.Collapsed;

}

//enables & disables collection respectively

private void collectToggleButton\_Checked(object sender, RoutedEventArgs e)

{

this.jellyfishMovement.StartCollecting();

}

private void collectToggleButton\_Unchecked(object sender, RoutedEventArgs e)

{

this.jellyfishMovement.StopCollecting();

}

//toggle that makes the history visible

private void displayHistoryToggleButton\_Checked(object sender, RoutedEventArgs e)

{

historyListView.Visibility = Visibility.Visible;

}

private void displayHistoryToggleButton\_Unchecked(object sender, RoutedEventArgs e)

{

historyListView.Visibility = Visibility.Collapsed;

}

//Update measurements

private void measurmentTextUpdateHandler(object sender, object e)

{

//recent measurement

recentMeasurementTextBox.Text = this.jellyfishMovement.mostRecentMeasure.ToString();

//history

historyDataBinding();

//converted measurment

if (jellyfishMovement.unitsToUse == Unit.Imperial)

{

convertedMeasurementTextBox.Text = Math.Round(this.jellyfishMovement.MetricValue(),4).ToString();

}

else {

convertedMeasurementTextBox.Text = Math.Round(this.jellyfishMovement.ImperialValue(),4).ToString();

}

}

//format raw data and bind historylistview to the formatted data

private void historyDataBinding() {

int qLength = this.measureHistory.Count + 1;

string record = "";

//dequeue so that things don't break

if (qLength > 10)

{

this.measureHistory.Dequeue();

}

//add raw data to record string

record += this.jellyfishMovement.GetRawData()[this.measureHistory.Count].ToString();

//add units of measurement

if (jellyfishMovement.unitsToUse == Unit.Imperial)

{

record += '″';

}

else {

record += " cm";

}

//add time stamp

record += (" Time: " + DateTime.Now);

//add record to queue

this.measureHistory.Enqueue(record);

//update display with queue data

historyListView.ItemsSource = this.measureHistory.ToArray<string>().Reverse();

}

private void inchesRadioButton\_Checked(object sender, RoutedEventArgs e)

{

this.jellyfishMovement.setUnit(Unit.Imperial);

}

private void centimetersRadioButton\_Checked(object sender, RoutedEventArgs e)

{

this.jellyfishMovement.setUnit(Unit.Metric);

}

}

}

**MeasureLengthDevice.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using Windows.UI.Xaml;//for dispatch timer

//This "device" tracks various length measurments in a table

namespace DataCollector

{

class MeasureLengthDevice : Device, IMeasuringDevice

{

//Constants

private const int MAX\_QUEUE\_LENGTH = 10;

private const decimal INCH\_CENTIMETER\_CONVERSION = 2.54m;

//collections

private Queue<int> dataCaptured = new Queue<int>();

//properties

private int \_mostRecentMeasure;

public int mostRecentMeasure

{

get => this.\_mostRecentMeasure;

private set => this.\_mostRecentMeasure = value;

}

private DispatcherTimer \_timeData = new DispatcherTimer(); //this will be used in mainpage.xaml.cs, but we need to control collection through the object, so it's here

public DispatcherTimer timeData {

get => \_timeData;

private set => \_timeData = value;

}

private Unit \_unitsToUse;//determines base units

public Unit unitsToUse {

get => \_unitsToUse;

private set => \_unitsToUse = value;

}

//constructor

public MeasureLengthDevice(Unit u) {

this.unitsToUse = u;

this.timeData.Tick += captureData;//subscribe tick event to captureData

}

//methods

public void setUnit(Unit u)

{

this.unitsToUse = u;

}

//Adds data to the dataCaptured queue when the timer ticks

public void captureData(object sender, object e) {

int d = GetMeasurement();

this.mostRecentMeasure = d;

//add data to queue and remove old data from queue if needed (max length 10)

dataCaptured.Enqueue(d);

if (dataCaptured.Count > MAX\_QUEUE\_LENGTH) {

dataCaptured.Dequeue();

}

}

//interface-loaned methods

public int[] GetRawData()

{

//this passes the array by reference, but we aren't changing any values when we use this, so this should be fine

return this.dataCaptured.ToArray<int>();

}

//displays imperial and metric values respectively, regardless of what the base units are

public decimal ImperialValue()

{

if (unitsToUse == Unit.Imperial)

{

return this.mostRecentMeasure;

}

else {

return this.mostRecentMeasure \* INCH\_CENTIMETER\_CONVERSION;

}

}

public decimal MetricValue()

{

if (unitsToUse == Unit.Metric)

{

return this.mostRecentMeasure;

}

else

{

return this.mostRecentMeasure / INCH\_CENTIMETER\_CONVERSION;

}

}

//activates and deactivates the timer respectively

public void StartCollecting()

{

TimeSpan measInterv = new TimeSpan(0,0,3); //tick every 3 seconds. You can pretend this is the fastest 15 seconds of your life if you want.

this.timeData.Interval = measInterv;

this.timeData.Start();

}

public void StopCollecting()

{

this.timeData.Stop();

}

}

}

**Device.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

public class Device

{

//this generates a random number between 1 and 10

public int GetMeasurement() {

Random rNum = new Random();

return rNum.Next(10)+1;

}

}

}

**UnitsEnumeration.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

//for designating what kind of units of measurement should be used

public enum Unit { Metric, Imperial};

}

**IMeasuringDevice.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataCollector

{

public interface IMeasuringDevice

{

decimal MetricValue();

decimal ImperialValue();

void StartCollecting();

void StopCollecting();

int[] GetRawData();

}

}