Introduction

This document explains how to use the Simple Line Graph asset. It is compatible with **Unity 2020.x or later**.

The prefab samples are present in Prefabs/UI/Graph folder.

Basic Usage

The line graph is represented by the *LineGraph* class. To draw the graph, a dataset (*ILineGraphData*) is required as the first parameter of the *PlotGraph* method. **Before drawing,** the graph is automatically cleared to ensure no overlap with previous data.

Example:

```
ILineGraphData selectedData = new
SimpleLineGraphData(
    new List<Vector2>() { new
Vector2(0, 25), new Vector2(10, 50) },
    new List<Vector2>() { new
Vector2(0, 100), new Vector2(25, 75) }
);
lineGraph.PlotGraph(selectedData);
```

Customizing Dots and Lines

By default, the line graph draws uniform dots and lines. You can customize these elements by passing the optional *dotCreated* and *lineCreated* delegate arguments to the `Plot` method.

Example:

```
lineGraph.PlotGraph(selectedData, DotCreated, LineCreated,
CriticalValuesFound);

private void LineCreated(int lineId, Image line)
{
    line.color = _colorPick[lineId].Color;
}

private void DotCreated(int lineId, Image dot)
{
    dot.color = _colorPick[lineId].Color;
}
```

- The *lineId* parameter refers to the index of the line in the dataset, ranging from **0** to **number of lines - 1**.

Performing Calculations Before Plotting

If you need to perform calculations after identifying the **minimum and maximum values** in the dataset, you can use the optional *criticalValuesFound* delegate argument in the *PlotGraph* method.

Performing Calculations After Plotting

If you need to perform calculations after identifying the **line, particularly dots relative positions to graph**, you can use the optional *normalizedLineDotsCalculated* delegate argument in the *PlotGraph* method.

Additional Visual Effects

- DrawHighlights() draws the highlight under the curve using the normalized dot positions respectively to graph container size, color, and highlight prefab itself.

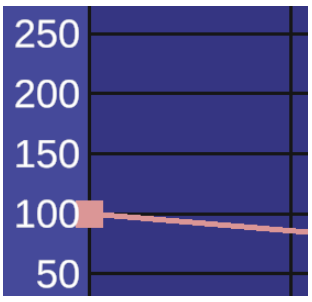
Clearing the Graph

To reset the graph, including gridlines, text, dots, and lines, you can call the *Clear* method. This ensures that all previous content is removed before drawing a new graph. Moreover, the graph utilizes pooling system to decrease the number of allocations and *Destroy* function calls during the plotting process after cleaning.

Gridlines and Discretization

- **Discretization** refers to the unit step used for spacing between grid lines.

- For example, if the vertical discretization is set to **50 units**gridlines will be drawn every 50 units.



Data Structures and Formats

The dataset for the graph can be represented in different forms. The following built-in formats are available:

- **SimpleLineGraphData**: Requires multiple *IList<Vector2>* objects, where each list represents a separate line to be drawn on the graph.

Example:

```
IList<Vector2> line1 = new
List<Vector2>() { new Vector2(0, 25),
new Vector2(10, 50) };
IList<Vector2> line2 = new
List<Vector2>() { new Vector2(0, 100),
new Vector2(25, 75) };
var dataset = new
SimpleLineGraphData(line1, line2);
```

- **FunctionLineGraphData**: Uses multiple *Function* structs, where each struct contains:

  - A **function** (*Func<float, float>*)

  - A dataset of **independent variables** (*x*).

Optional Features

- You can **disable the rendering** of horizontal and/or vertical grid lines as well as text labels.

- **Note**: Logarithmic scale is **currently not supported**.