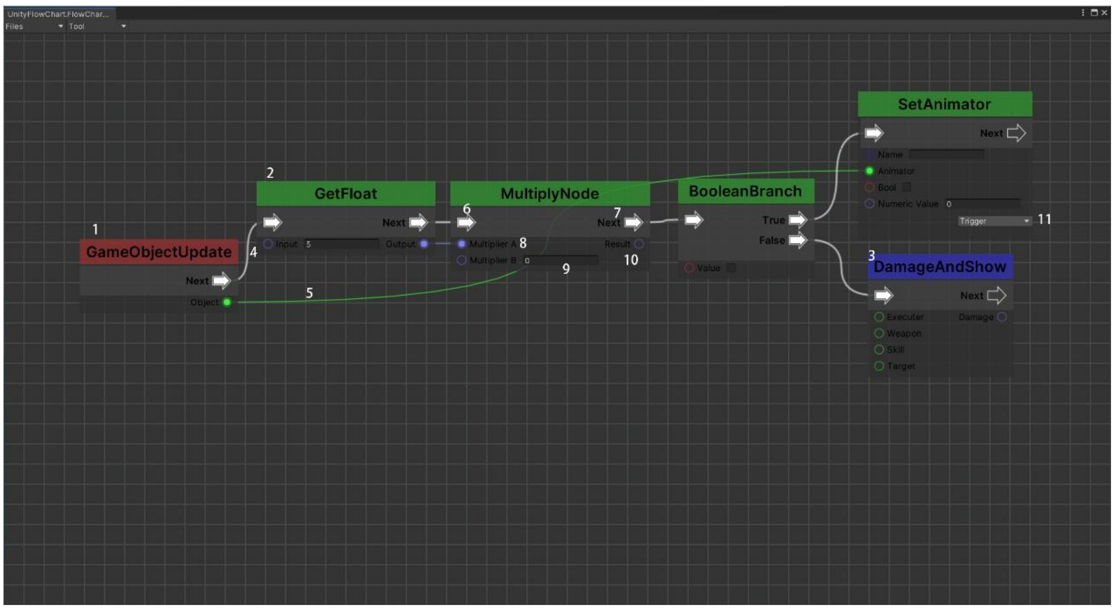
Flowchart editor

directory

Concept note

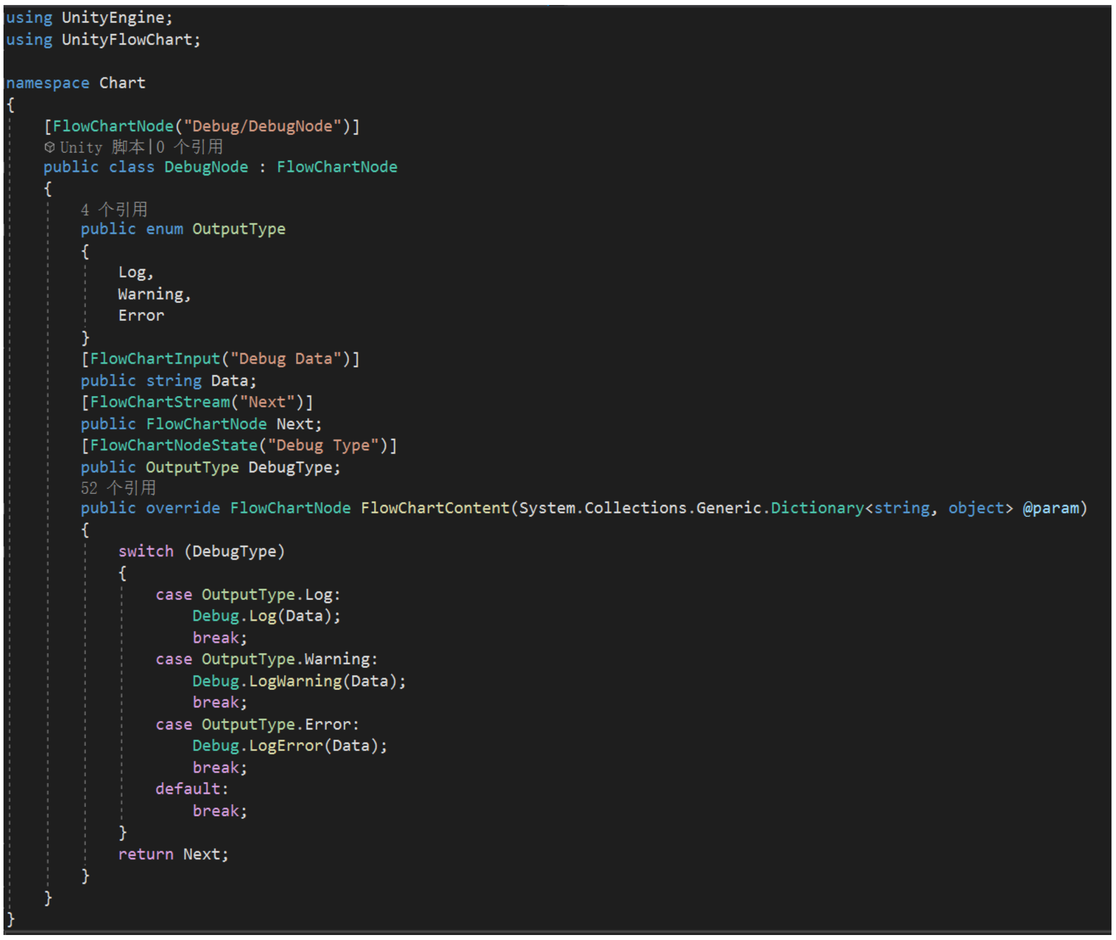


1、 The root node
2、 Normal node
3、 The subplot node
4、 The process is wired
5、 The parameters are wired
6、 Process input point
7、 The process output point
8、 The parameter input point
9、 The parameter input box
10、 The parameter output point
11、 Node state selection

Open and save

You can select any of the prefabricated flowcharts generated and then right-click to select the Flow Chart button to open the prefab, or you can click on a blank space or open a blank flowchart through Assets/Flow Chart.
You can save edited flowcharts in the editor state through File/Save or Ctrl s. If the flowchart is opened from a prefab, the original prefab is overwritten directly, and if it is a new blank flowchart, a path selection box is pop-up to select save the directory.

Custom node



```
using UnityEngine;
using UnityFlowChart;

namespace Chart
{
    [FlowChartNode("Debug/DebugNode")]
    ⚙Unity 脚本|0 个引用
    public class DebugNode : FlowChartNode
    {
        4 个引用
        public enum OutputType
        {
            Log,
            Warning,
            Error
        }
        [FlowChartInput("Debug Data")]
        public string Data;
        [FlowChartStream("Next")]
        public FlowChartNode Next;
        [FlowChartNodeState("Debug Type")]
        public OutputType DebugType;
        52 个引用
        public override FlowChartNode FlowChartContent(System.Collections.Generic.Dictionary<string, object> @param)
        {
            switch (DebugType)
            {
                case OutputType.Log:
                    Debug.Log(Data);
                    break;
                case OutputType.Warning:
                    Debug.LogWarning(Data);
                    break;
                case OutputType.Error:
                    Debug.LogError(Data);
                    break;
                default:
                    break;
            }
            return Next;
        }
    }
}
```

A class inherited from FlowChartNode represents a node

FlowChartNodeAttribute

Each node must have the label
String editorPath: The node appears at the level and name of the list
FlowChartNodeType   type: indicates whether the node is a root node or a normal node
Bool hidden: Whether the node is hidden in the list

parameter

Both the input and output parameters must have a FlowChartinput / FlowChartOutput label, and their string parameters represent the name of the variable displayed on the editor.

Process output

The process output must have a FlowChartStream label, and the string parameter is the name it appears on the editor. A node allows multiple output nodes, and the return value of the FlowChartNode.FlowChartContent method determines which node to execute next.

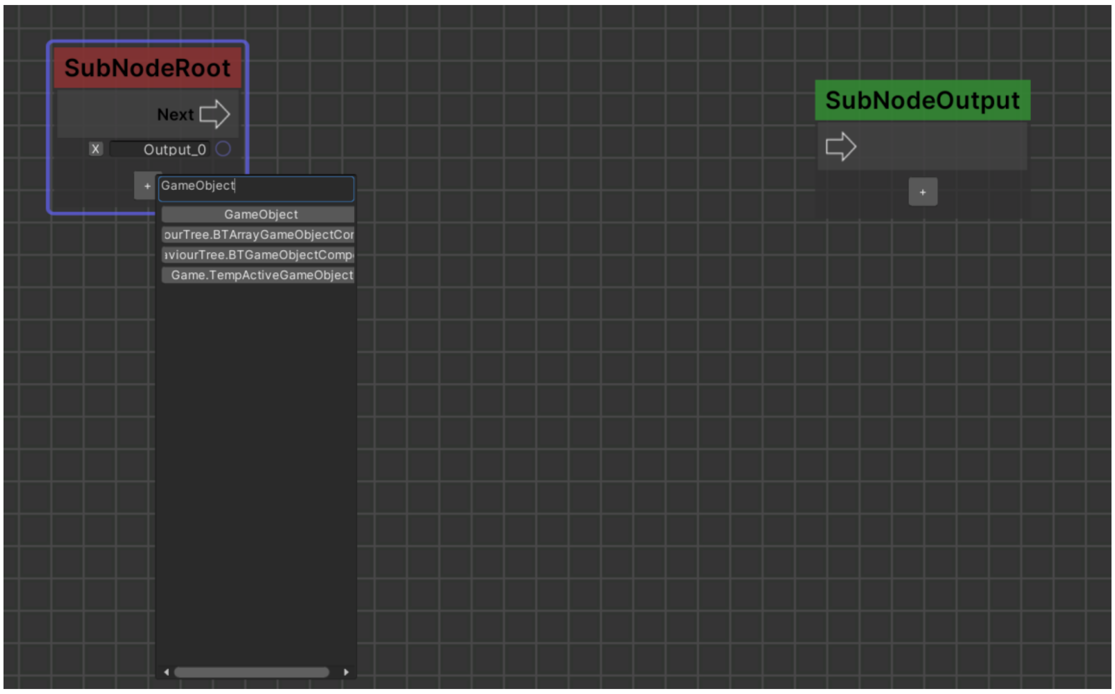
Status selection

The status selection must have the FlowChartNodeState label, and the string parameter is the name it appears on the editor. This field must be an enumerated type.

Abstract method FlowChartNode.FlowChartContent

Each node class must implement a method that determines the specific behavior of the node, @param parameter is a dictionary that holds all the parameters in the flowchart execution. Input parameters can be used directly in the method. The output parameter *must be assigned (similar* to the *out* parameter used) *before* the *method returns, otherwise it* will cause an empty reference error that the next node will not be able to access the parameter.
The return value of the node determines which node is executed next.

Subplot

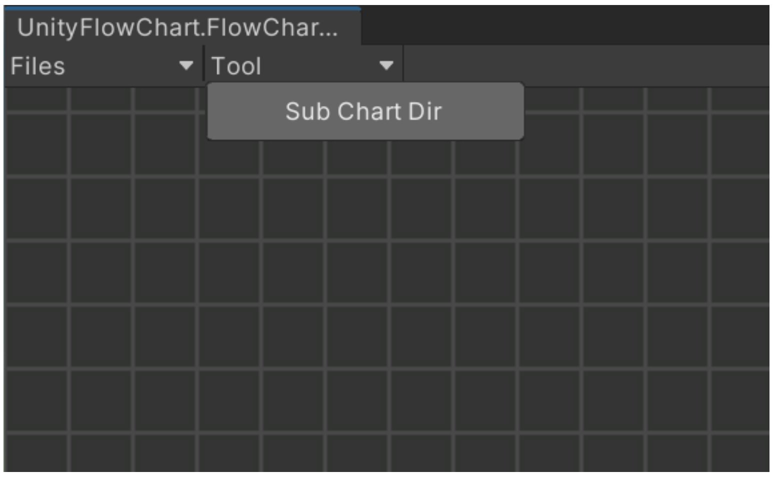

SubNodeRoot
Next
X
Output_0
+
GameObject
GameObject
ourTree.BTArrayGameObjectCor
iviourTree.BTGameObjectComp
Game.TempActiveGameObject
SubNodeOutput
+

Multiple nodes can be made into a subplot for modular editing. The subplot must have and have only one SubNodeRoot node and one SubNodeOutput node. Both nodes can add or remove input and output parameters.

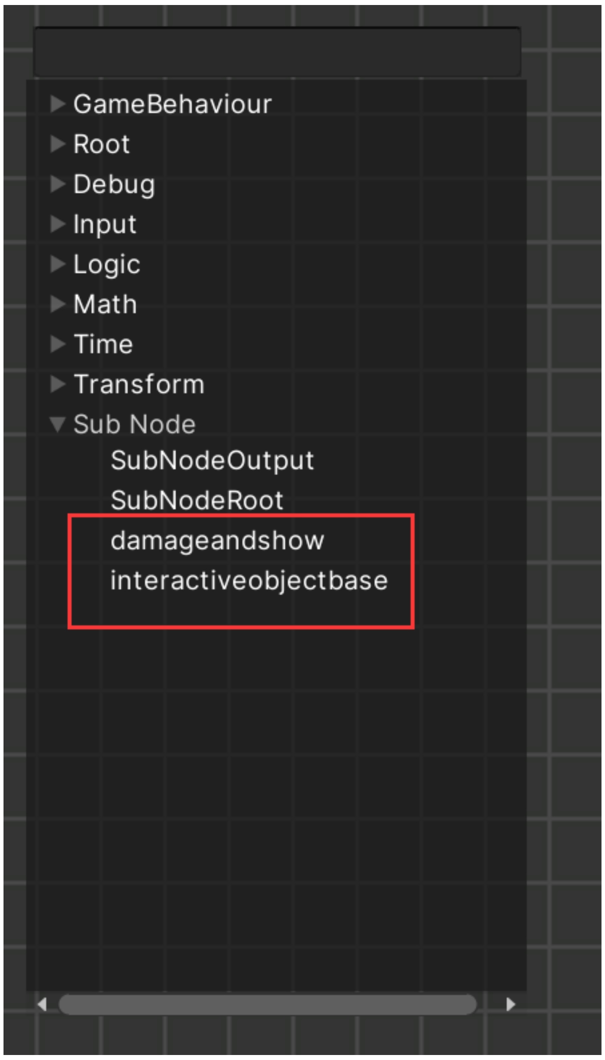
Add a delete parameter

You can add parameters by clicking on the number in the node, and the X button can delete the parameters.
When you add parameters, the panel pops up, you can enter the parameter type in the search box and then click the corresponding button to add parameters.
Because there are too many classes, the panel displays buttons only after the search parameters have been filtered to less than1 00 results.

Use subplots



To use subplots, you must save the prefabricated results generated by the edited subplots in the specified directory (which does not contain subdirectories). The directory can be selected viathe Tool->Sub Chart Dir button.



Once selected, you can search the list of nodes or find the created subplot nodes in the Sub Node subdirectories.

Parameter input

Parameters can be entered directly in the editor.
Currently, only the following types of input are supported:

1、 Base type(int, float, double, string, bool).

2、 Vector2,Vector2Int, etc.).

3、 Color, Rect

4、 The type inherited from UnityEngine.Object

5、 Enumerate the type

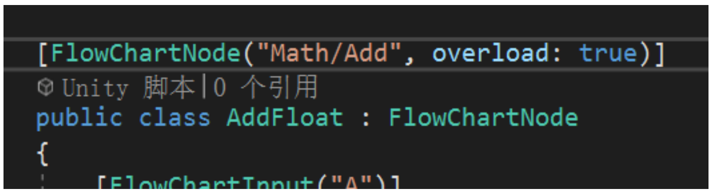
No type parameters of the subplot node currently support parameter input because Unity cannot serialize any parameters that are not of type at compile time.

The parameters are paired

When connecting input and output parameters, certain rules must be met, such as not being able to pair the output of int with the input of GameObject.
When you connect, you can match as long as any of the following rules are met.

1、 The same type can be matched.
2、 Any type of output can match the input of thes tring type.
3、 The output of the derived class can match the input of the base class.
4、 The numeric type (int, float, double)can be matched arbitrarily with the enumerated type.
5、 Vector (2, 3, 4) types and floats can be converted to each other.
6、 UnityEngine.GameObject output can match any input derived from UnityEngine.Component, but the Component must be available through GameObject.GeComponent or an empty reference error will occur.
7、 Types derived from UcityEngine.Component can also be converted to each other, but the previous rule must also be met.
8、 Types derived from UcityEngine.Component can be converted directly to theUnityEngine.GameObject type.

The node is overloaded



```
[FlowChartNode("Math/Add", overload: true)]
Unity 脚本|0 个引用
public class AddFloat : FlowChartNode
{
    [FlowChartInput("A")]
```

function:

The nodes with the same Path in the node label, and nodes with the overload parameter set to True can enable node overloading. When overloading is enabled, *the first* parameter that connects to the node will be used as the basis for the overload, selecting the node that is most suitable for the overload to replace.
(For example, if you connect the last Vector3 input to the AddFloat node, the node becomes AddVector3,at which pointconnecting the second parameter does not change the node regardless of type.) ）

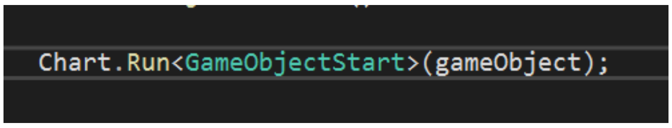
request:

Overloaded nodes must comply with the following requirements:

1、The number of node process output points, parameter input points, and parameter output points must be the same.

2、The corresponding parameters between the overloaded nodes, including input and output, must be adaptable.

Trigger the flowchart



```
Chart.Run<GameObjectStart>(gameObject);
```

Gets the prefabricated generated by the flowchart editor and then calls GetComponent<FlowChart>(). Run, you can run the flowchart, where generic or Type variables represent the triggered root node, and if the root node is not in the flowchart, nothing will be done. The parameters of the method are the output parameters required by the root node.