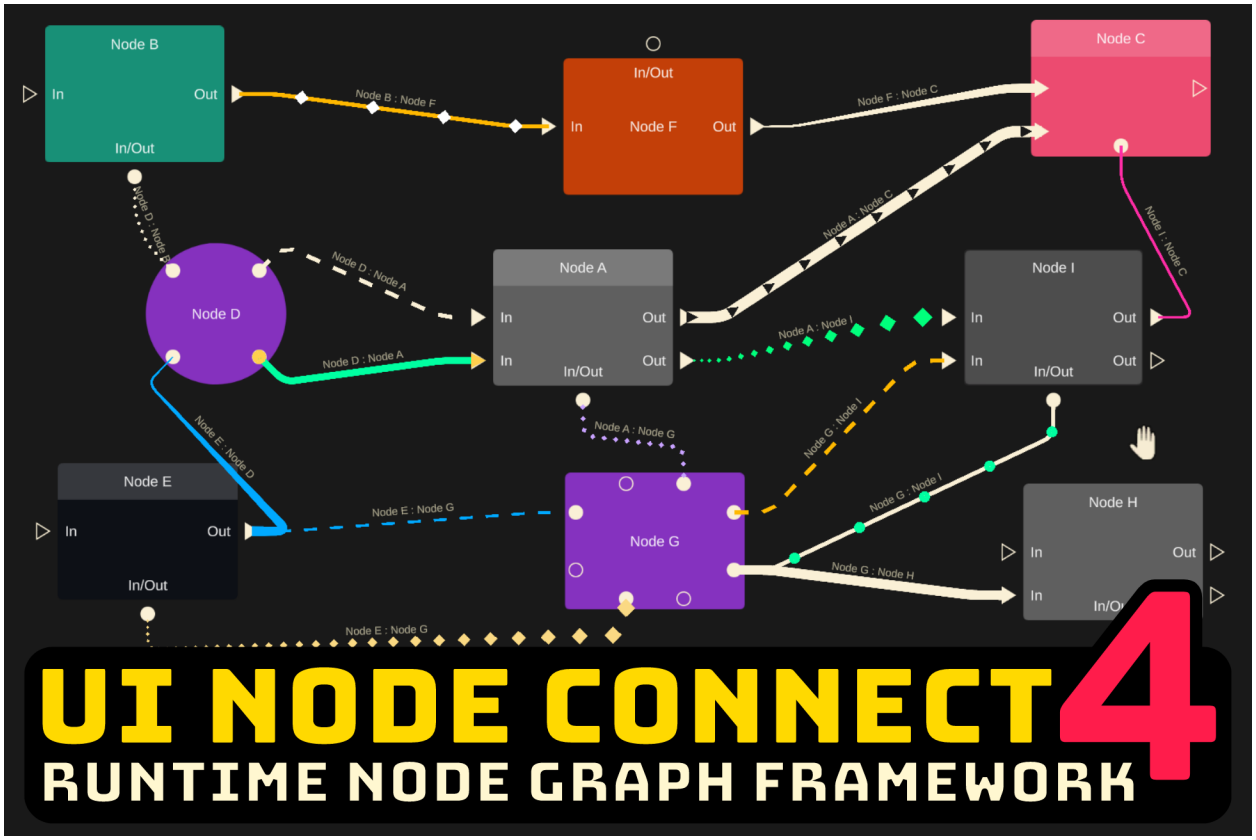


The UI Node Connect 4 is a **runtime UGUI Node Graph Framework** with Nodes, Ports and Connections, drag and drop, a custom UI line renderer and useful features to create node graphs, flowcharts, circuit boards, connection puzzles, among other creative ways for the users to interact with your game or app.

This documentation contains information on the implemented features and code, examples and how to customize.

[View the Code Reference]

Contact

[Asset Store Page]

[support@meadowgames.com]

MeadowGames
[MeadowGames.com]

Contents

1. OVERVIEW

The **UI Node Connect 4 (UIC4)** is a framework for UI Canvas that facilitates and speeds up the creation of node graphs and similar mechanics for your Game or App, enabling you to easily create node graphs, flowcharts, circuit boards, connection puzzles, among others.

It is possible to customize the asset with your own sprites, texts, colors, buttons and expand it by adding new features using the API to hook up to events and analyze the Node's Ports and Connections.

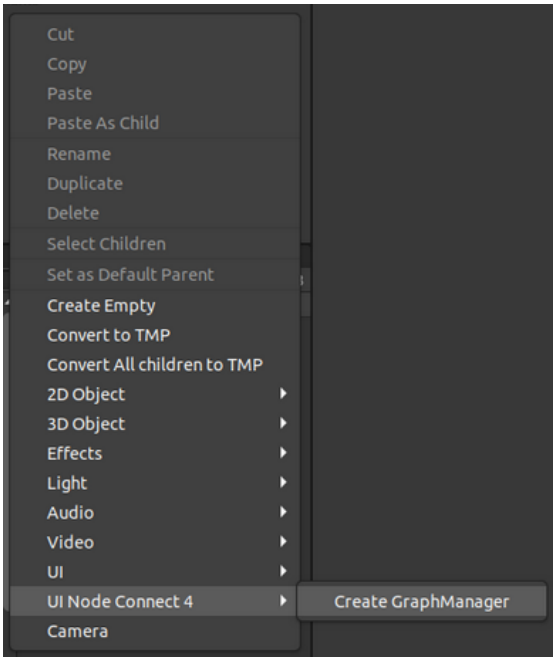
The main visual elements of the system are the Nodes, Ports and Connections, that you can access via the Graph Manager. The UIC4 also has an event system for a more organized and decoupled interaction of your scripts with the system.

In the following sections, the system's modules and example scenes are explained.

2. Getting Started

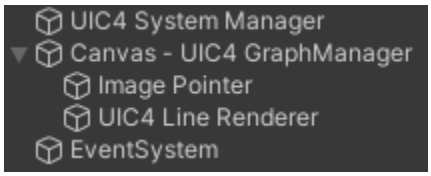
With the package imported into the scene a new option will be available in the Hierarchy's context menu (access it by clicking in the hierarchy or a GameObject with the right mouse button), "UI Node Connect 4". From there, you can create a Graph from scratch by selecting "UI Node Connect 4 > Create a GraphManager".

Note that you will also have access to the sample scenes, they are a great learning source and can be used as templates for your projects.

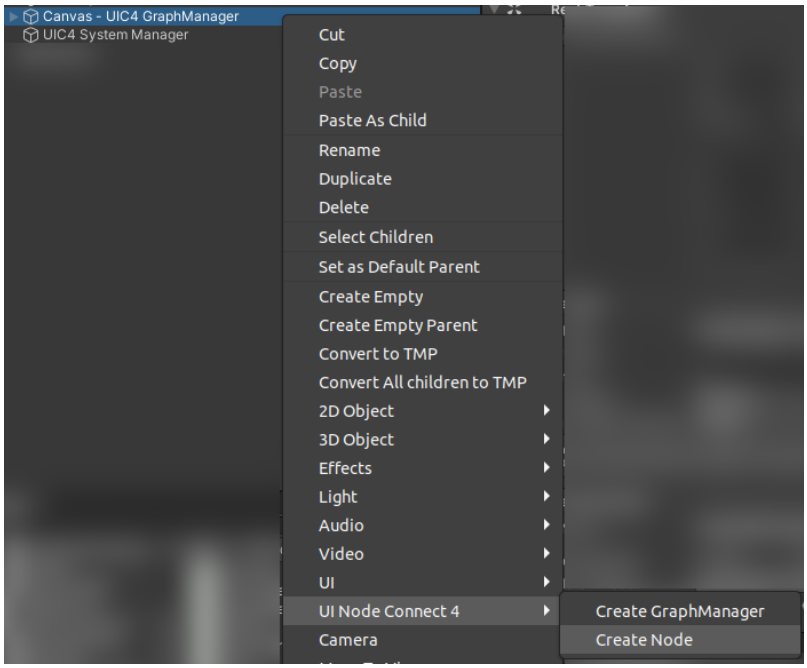


If you create a Graph Manager in a blank project, this button will add two GameObjects, besides the EventSystem needed for the Canvas interactions, those are:

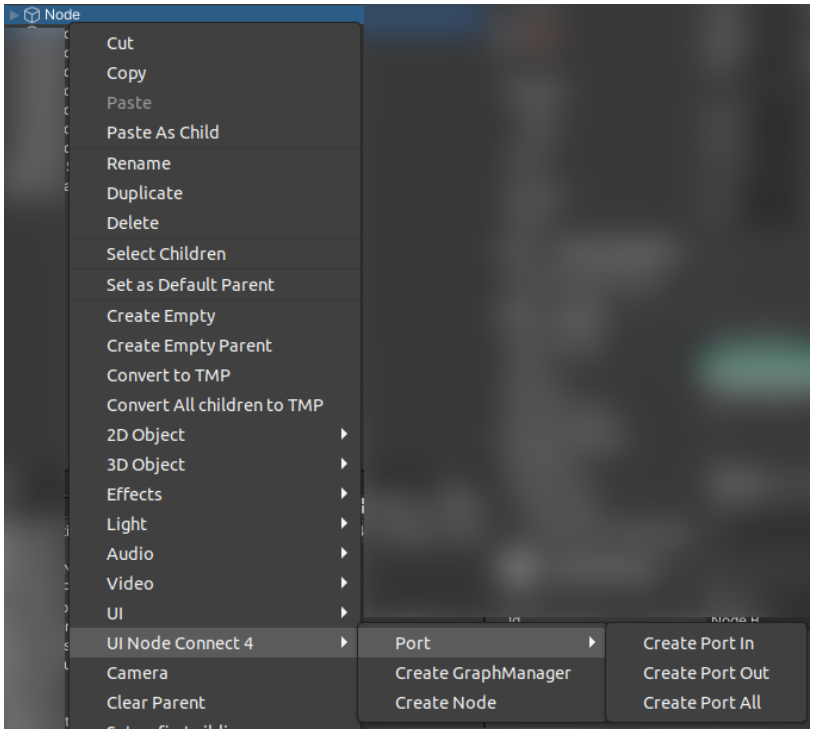
- **Canvas - UIC4 Graph Manager:** object with a Canvas and a Graph Manager components, already setup with a custom image for a pointer and a child UIC Line Renderer
- **UIC4 System Manager:** object with an UIC System Manager and Input Manager components



After the Graph Manager is added to the scene you can start adding Nodes. Opening the context menu from the Graph Manager GameObject the “UI Node Connect 4” menu will also show the option “Create Node”.



The added Nodes come with one Port of each type as an example, at this point, the system already has all the needed components to work. You can also add new Ports to the Nodes using the context menu.



Notes:

- If you have multiple Canvas, it is recommended to set the variable “Compensate Scale Of Parents” to true in the UIC Line Renderer’s inspector
- The Graph Manager **can** be a child of other canvases

Examples of possible hierarchy	
- Canvas ↳Graph Manager	- Canvas ↳Canvas ↳… ↳Graph Manager

- The Nodes **must** be, in any hierarchical level, under a Graph Manager

Examples of possible hierarchy	
- Graph Manager ↳Node	- Graph Manager ↳GameObject ↳… ↳Node

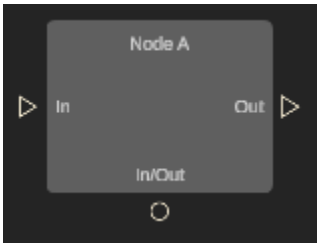
- It is not mandatory for the UIC Line Renderer to be a child of the Graph Manager, however, it is recommended that it is a child of a Canvas with the same Scale and Render Mode settings as the Graph Manager

3. Graph Elements

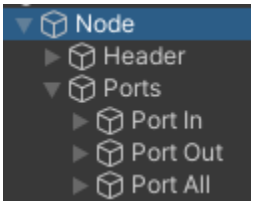
The Graph Elements are the main visual elements of the system, those are the Nodes, Ports and Connections. While Node and Port are MonoBehaviours, the Connection is a non-MonoBehaviour object stored by the GraphManager and rendered by the UIC Line Renderer.

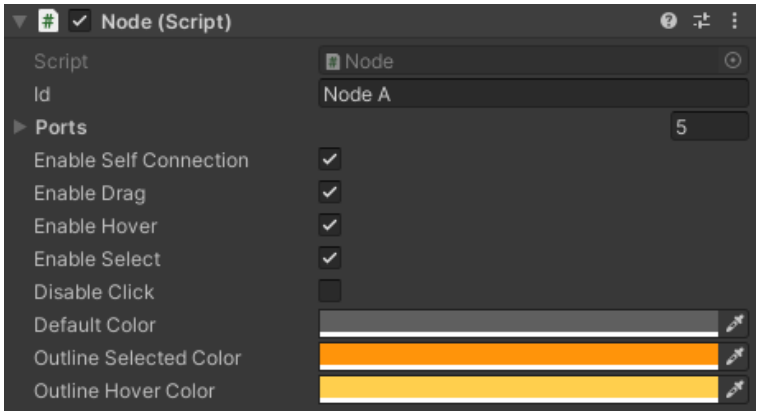
3.1. Node

The Node is a visual element that serves as a hub for Ports. It can have any number of Ports of any type and its visual is given by an Image component that can be customized freely.



The Image component is used for the sprite and Raycast. It also contains an Outline component, enabled when the node is hovered or selected.

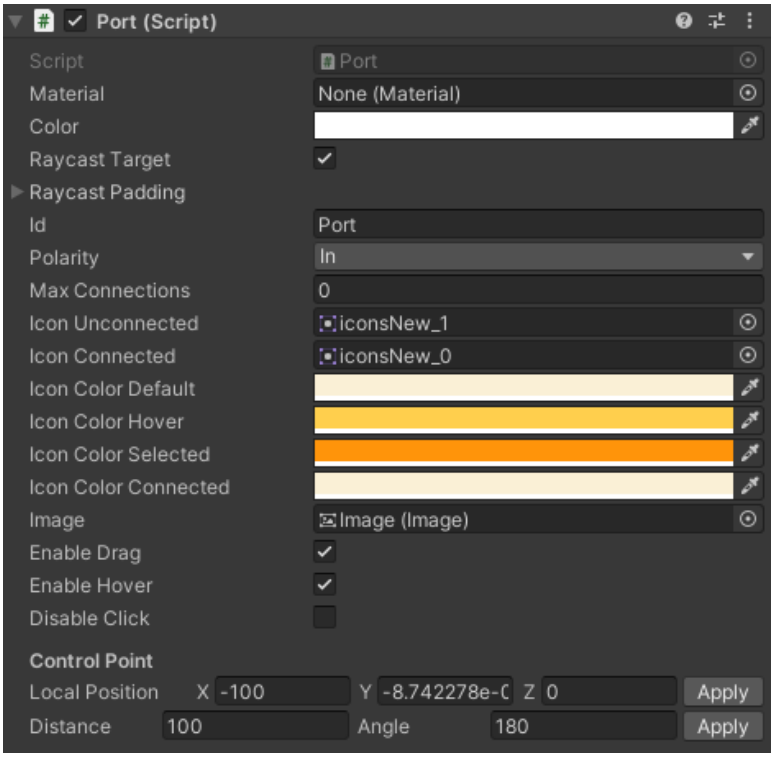




3.2. Port

The Port is a spot for connections. From it, a connection can be dragged to another Port, creating a link between them. This component holds a list of its own connections, which makes it possible to know the nodes connected to this particular port.

The Image is the visual component of the Port. It is not a necessary component for the port to work but if present, its icon and color will be updated based on the Port's colors settings.



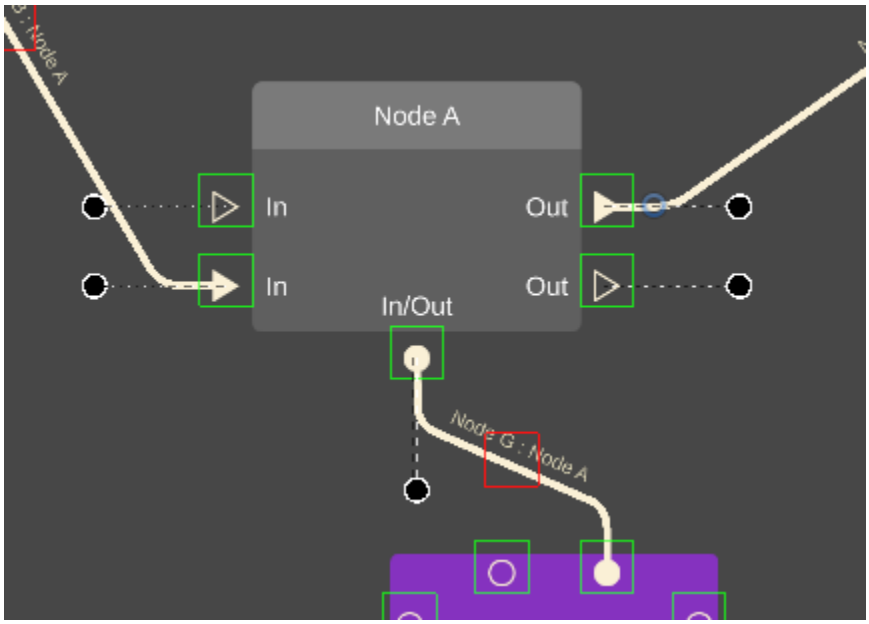
3.2.1. Control Point

Each Port contains a child called Control Point, it is used by the connection line as a control point for its curvature, making the line be directed to a set position in relation to the port.

It can be adjusted in 3 ways:

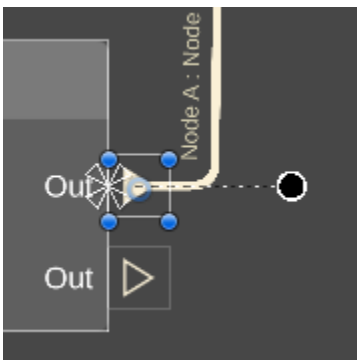
- manually by dragging the Control Point GameObject to the desired position
- from the Port inspector by setting the local position or its distance from the port and direction or
- via script with the methods below:

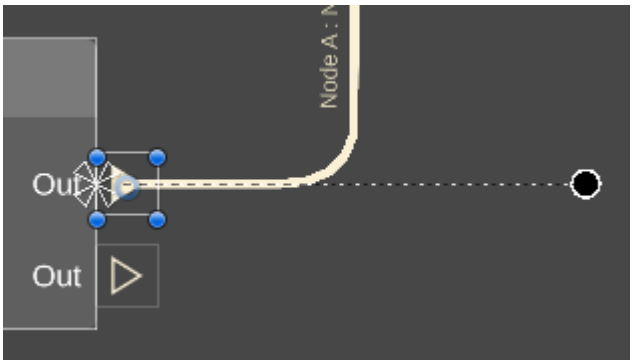
```
port.SetControlPointLocalPosition(Vector3 position)
port.SetControlPointDistanceAngle(float distance, float angle)
```



The Control Point affects the SpLine, Z Shape and Soft Z Shape curve styles. Since this GameObject is a child of the Control Point, this position and direction will be relative to the Port.

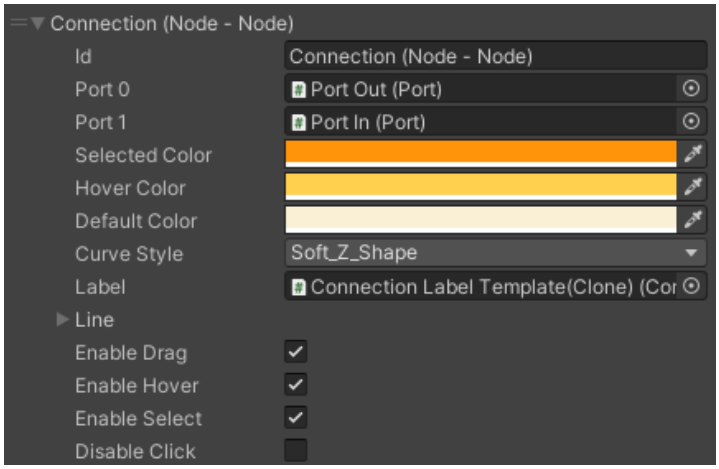
The distance from the connection point will also influence the line curvature.





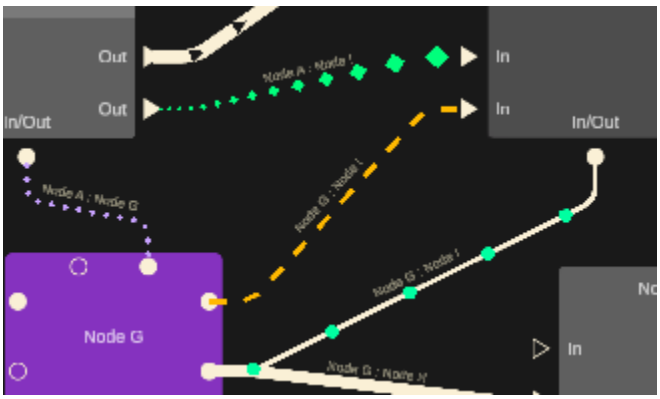
3.3. Connection

The Connection links the ports visually and logically. It holds a reference for the Line, rendered by the UIC Line Renderer, and references for each port it connects.



3.3.1. Line

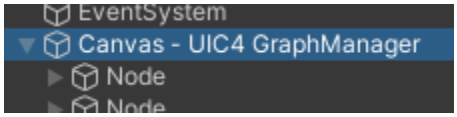
The Connection Line is rendered by the UIC Line Renderer. It also can have its style customized to represent different types of connections.



The Line Animation can have different styles as well and can be activated to represent, for example, data flow.

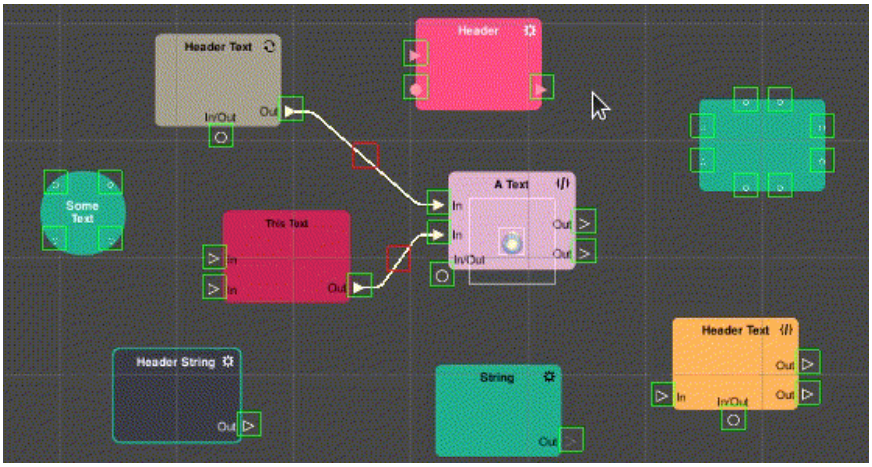
3.3.2. Connections in Edit Mode

If you need to create a scene where some connections are already made between nodes, you are able to create those connections in edit mode by selecting the Graph Manager in the Hierarchy.



In the Scene view, each Port will have a green square around and each Connection a red square in the middle of it.

To create connections, click on the first port then click on the second port. To remove connections, click on the red square in the middle of the connection.



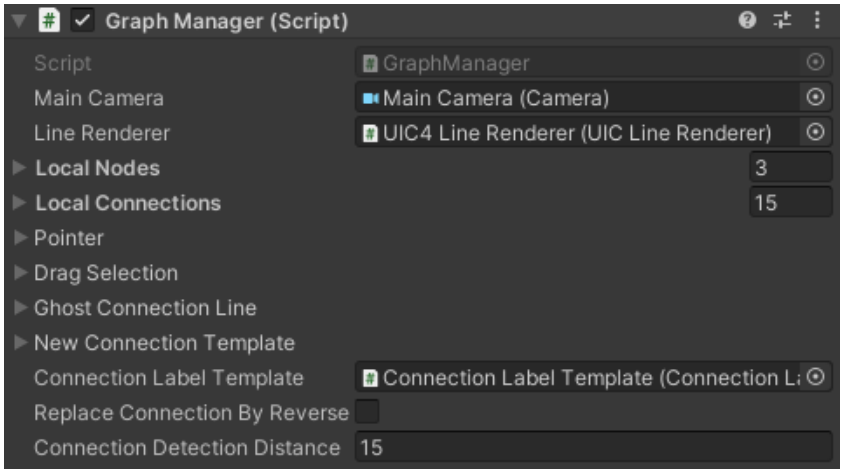
4. Core System Modules

There are 4 core modules that should be in the scene so the system works, those are the Graph Manager, UIC Line Renderer, Input Manager and UIC System Manager. Each will be detailed in the next sections.

4.1. Graph Manager

The Graph Manager is the module that has the Canvas reference, it is a component that should be in a Canvas, parent (at any above hierarchical level) to the Nodes. It contains references for all the local Nodes and Connections and serves as a hub of references, for the Raycaster, Drag and Drop, UIC Line Renderer and Connections and Lines visual style and settings.

It is possible to create a Graph Manager in a project from the Editor's context menu, doing so will automatically add an UIC Line Renderer linked to this Graph and a UIC System Manager if there is none in the scene.

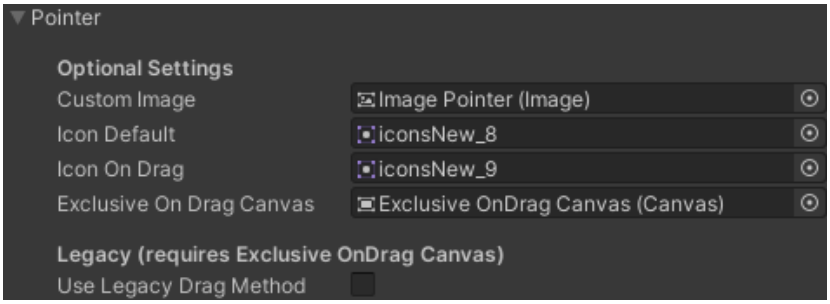


4.1.1. Pointer

The Graph Manager's Pointer is responsible for managing the click and drag/drop actions.

One optional setting is to add a custom Image component as reference for the pointer, it will follow the pointer. You can define the "default" and "on drag" icons for the pointer.

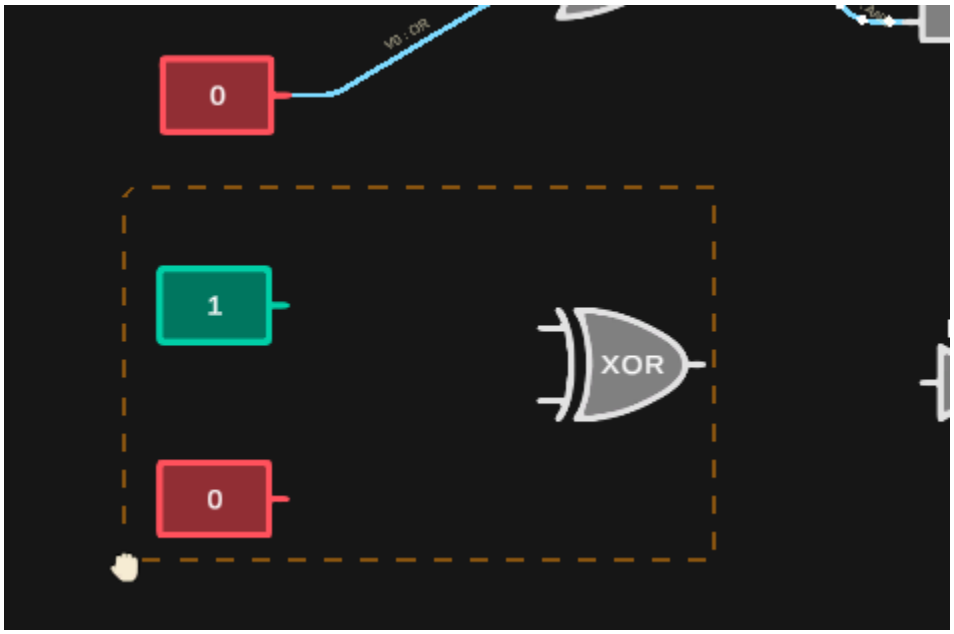
If you need the dragged object to be placed in a separate canvas during the drag, for example to adjust its layer position or add some different effect on it, you can set this canvas at the "Exclusive On Drag Canvas" variable.



4.1.2. Drag Selection

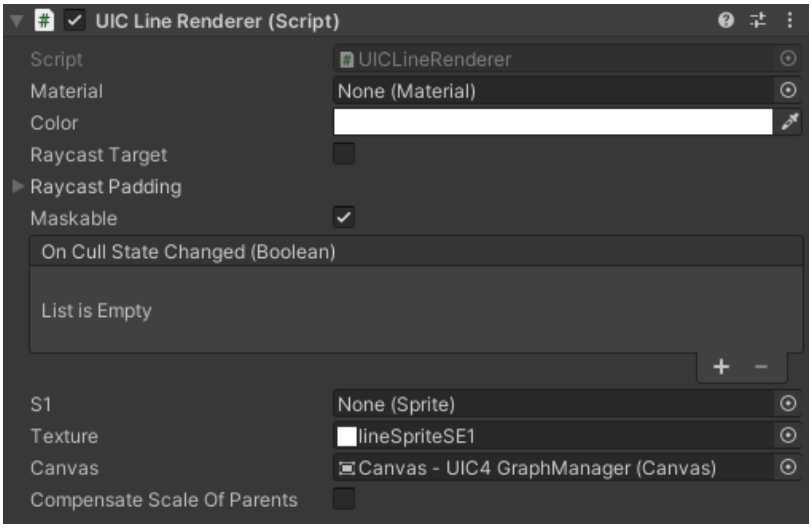
The Drag Selection is the module that manages the selection of elements by dragging the mouse. It creates a rectangle to indicate the objects that will be selected.

You can adjust the line style and, optionally, set a different UIC Line Renderer to render the drag selection line.



4.2. UIC Line Renderer

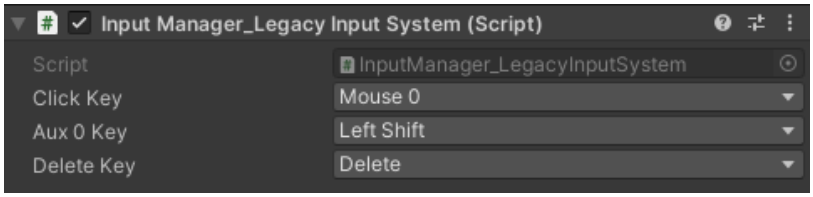
The UIC Line Renderer is a UI MaskableGraphic component responsible for managing and rendering the lines.



4.3. Input Manager

The Input manager is a component, by default attached to the same GameObject as the UIC System Manager, that is responsible for indicating the inputs used by the system.

The default Input Manager uses the legacy Unity's input system and its inputs can be configured via the Inspector.



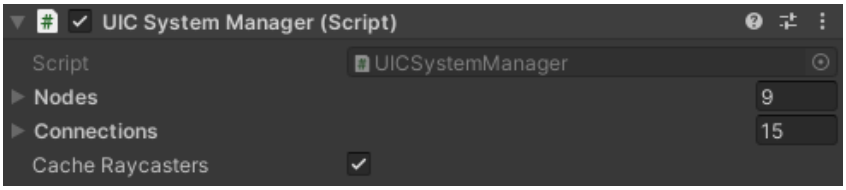
4.3.1. Replacing the Input Manager

It is possible to create a new Input Manager adjusted for your needs, such as using inputs from VR, by creating a new Component that inherits from UINodeConnect4.InputManager and overriding the needed methods and properties (use the InputManager_LegacyInputSystem as a template).

After creating the new component, replace the current one in the scene by your custom one.

4.4. UIC System Manager

The UIC System Manager is responsible for managing all the system's executions and events. It is a singleton that can be accessed by other scripts from its static methods and variables.



4.4.1. Event Manager

The Event Manager enables the possibility of subscribing to UIC system's events. It returns a IElement as parameter and can be accessed from the UIC System Manager by calling UICSystemManager.UICEvents.

The default events are:

- OnPointerDown, OnDrag, OnPointerUp, OnDeleteKeyPressed, OnPointerHoverEnter, OnPointerHoverExit, OnConnectionCreated, OnConnectionRemoved, NodeAdded, NodeRemoved, ConnectionAdded, ConnectionRemoved, OnElementSelected.

5. Secondary Modules

Other optional modules are fully decoupled from the UIC core, they can be used to expand the system's capabilities, those are the Context Menu and Extension Components.

5.1. Context Menu

To add other UI elements, such as buttons, that can be clicked without deselecting the ISelectable graph elements (Nodes and Connections), it is required that they have a component that inherits from IElement.

In the project, there is a predefined set of objects, managed by the Context Menu Manager, that can be used to change some settings and selected elements settings during the runtime, those are the Context Items.

The Context Menu Manager is responsible for refreshing the Context Items when an element is selected or deselected.

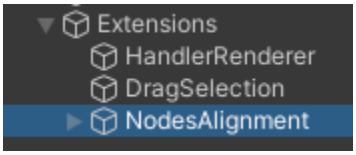
5.1.1. Context Item

The Context Item is a component that has an action related to the global or the selected graph elements settings. Some of the default items are buttons, sliders and dropdowns that are updated with the related value once an element is selected or unselected and can be set without unselecting the elements on the graph.

5.2. Extension Components

5.2.1. Nodes Alignment

The Nodes Alignment module adds methods to align and space nodes vertically and horizontally. An example of its use is the SetNodesAlignemtn Context Item, which has child buttons that call the align and space methods for organizing the selected Nodes.




```
// The methods without passing nodes as parameter aligns and spaces the
```

```
selected nodes
```

```
AlignVertical()
AlignVertical(List<UIC3_Node> nodes)
```

```
AlignHorizontal()
AlignHorizontal(List<UIC3_Node> nodes)
```

```
DistributeEvenCenterVertical()
DistributeEvenCenterVertical(List<UIC3_Node> nodes)
```

```
DistributeEvenCenterHorizontal()
DistributeEvenCenterHorizontal(List<UIC3_Node> nodes)
```

```
DistributeEvenSpaceVertical()
DistributeEvenSpaceVertical(List<UIC3_Node> nodes)
```

```
DistributeEvenSpaceHorizontal()
DistributeEvenSpaceHorizontal(List<UIC3_Node> nodes)
```

5.2.2. Connection Label Rule

The connection Label Rule is an abstract MonoBehaviour class that can be overridden to create rules for automatic labeling the newly created connections. It also can be called from script by calling ExecuteRule(connection) to label a passed Connection.

There are 3 Rule examples in the project that can be used: Fixed Label, Start Node ID (label the connection with the ID of the start node) and Start Plus End Node IDs (label the connection with the format “[start node ID] : [end node ID]”).

5.2.3. Port Match Rule

The Port Match Rule is a feature to facilitate the implementation of custom rules to enable or disable the connection between a “draggedPort” with a “foundPort” when creating a connection by dragging.

The rule returns a bool value that is tested before the default Port Polarity rules.

To create custom port match rules you need to create a component that inherits from Extension.PortMatchRule (an abstract singleton), implement the ExecuteRule method and add this component in the scene.

Only one component that inherits from PortMatchRule should be in the scene, since it is a singleton.

An example of a simple port match rule is the CustomPortMatchRule component of the Serialization Sample scene:

```
public class CustomPortMatchRule : PortMatchRule
   {
       public override bool ExecuteRule(Port draggedPort, Port foundPort)
       {
           return draggedPort.ID == foundPort.ID ? true: false;
       }
   }
```

6. SERIALIZATION

The UICSerialization namespace has classes to help on the serialization of the Graph Elements.

Note that each project has its own specificiness and, while it is possible to use the serialization system as it is for most projects, you might need to serialize specific data for your project. In that case you can use the built in SerializationEvents to call additional methods or create your own serialization system using the basic serializable types (SerializableNode, SerializablePort and SerializableConenction).

The Serialization Sample scene in the package is an usage example and can be used as a template for building more complex projects.

6.1. SERIALIZABLE TYPES

Each Graph Element has a respective serializable type containing only the needed data for the serialization process, those are: **SerializableNode, SerializablePort** and **SerializableConnection**.

In addition to that, there is a **SerializableGraph** type that contains a list of serializable Nodes and serializable Connections and methods to facilitate the serialization of all elements of a graph manager or the selected ones.

Those types are used by the UIC Serializers during the conversion to JSON and can also be used in custom serialization systems to transform data in any other data to be saved and loaded.

6.2. SERIALIZER HELPER CLASSES

There are four types of Serializer classes containing helpful methods to directly transform between elements, serializable and JSON.

There is one serializer class dedicated for each graph element type, **NodeSerializer, PortSerializer** and **ConnectionSerializer**, plus a **GraphSerializer**, to facilitate the serialization of all elements of a graph manager or the selected ones.

To be able use the serializers to deserialize data directly from JSON to scene objects, it is required to set templates for the Nodes and Ports based on their IDs, so the objects are correctly instantiated. This is done by having a **DeserializationTemplates** component in the scene with the fields for Nodes and Ports setup correctly.

The Serialization Sample scene uses the serializers to save to file and load elements to the scene. This system can be enough for most projects, however, if you need more data to be serialized or other actions to take place before on save and load, you can subscribe to the **SerializationEvents**.

6.2.1. Deserialization Templates

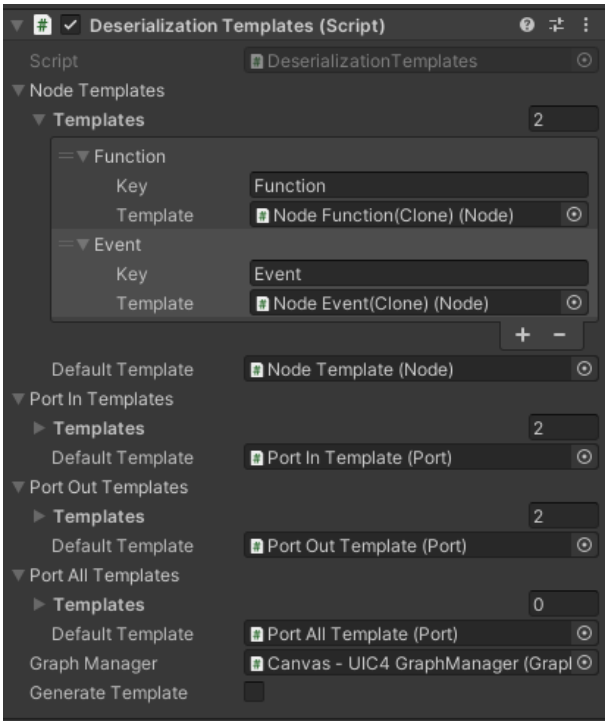
The Deserialization Templates component is used by the serializers to instantiate nodes and ports based on its ID. It facilitates the serialization when there are different types of nodes and ports with different layouts, sprites and components.

You can manually add templates to the component by creating items to the Node, Port In, Port Out and Port All templates lists. Each template must have an unique Key that is searched based on the IGraphElement.ID.

If you have a premade scene, you can use the checkbox “Generate Template” to automatically fill the templates.

Mind that:

- The scene elements’ IDs must be correctly set up and
- The templates are generated as childs of the DeserializationTemplates component, so you can, if needed, make adjustments and create prefabs.



6.2.2. Serialization Events

The serialization events are:

```
UICEvent<SerializableGraph> OnGraphSerialize
UICEvent<GraphManager> OnGraphDeserialize

UICEvent<SerializableNode> OnNodeSerialize
UICEvent<Node> OnNodeDeserialize

UICEvent<SerializablePort> OnPortSerialize
UICEvent<Port> OnPortDeserialize

UICEvent<SerializableConnection> OnConnectionSerialize
UICEvent<Connection> OnConnectionDeserialize
```

7. CUSTOMIZING

It is possible to customize the system in a variety of ways depending on the project's specificity. In this section, there are starter examples on how to customize parts of the system.

More examples on how to use the system are the actual semple scenes.

7.1. CREATING A NEW CONTEXT ITEM

To create a new context item, you will need to add a component that inherits from UICContextMenu.ContextItem, this will enable the possibility of interacting with this UI component while graph elements are selected.

To be able to update your component when a graph element is selected or unselected, override the OnChangeSelection method and implement it as you need.

The example below is the ButtonRemoveSelected, that removes the selected elements.

```
public class RemoveElementsButtonItem : ContextItem
   {
       Button _button;

       public void RemoveSelected()
       {
           for (int i = UICSystemManager.selectedElements.Count - 1; i >= 0; i--)
           {
               UICSystemManager.selectedElements[i].Remove();
           }
       }

       void OnEnable()
       {
           _button = GetComponent<Button>();
           _button.onClick.AddListener(RemoveSelected);
       }

       void OnDisable()
       {
           _button.onClick.RemoveAllListeners();
       }

       public override void OnChangeSelection()
       {
           gameObject.SetActive(UICSystemManager.selectedElements.Count > 0);
       }
   }
```

7.2. Using the New Input System

It is possible to use the new input system along with the legacy without any changes, however, if you want a **project only with the New Input System enabled** (check out the Unity's New Input System Installation Guide).

Two steps are needed to make UI Node Connect 4 support it.

1) Extract all the components of "***Assets/UI Node Connect 4/New Input System/UIC_NewInpuSystemComponents.zip***" to the same folder.

2) Locate and replace the ***InputManager_LegacyInputSystem*** component (normally located at the "UIC4 System Manager" GameObject) by the newly extracted ***InputManager_NewInputSystem*** component.

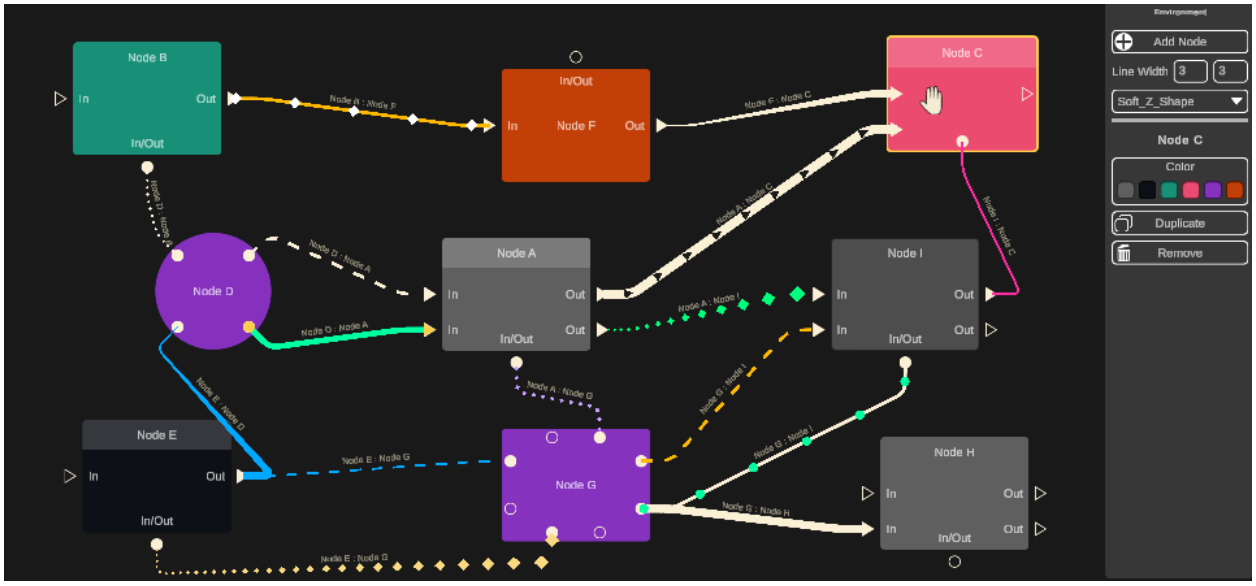
8. Sample Scenes

The sample scenes are good learning sources that you can also use as templates for your projects.

8.1. Flowchart

The Flowchart scene contains multiple Nodes and predefined Connections with different styles to show a variety of possibilities in terms of setting the visual style of Nodes, Ports and Connections.

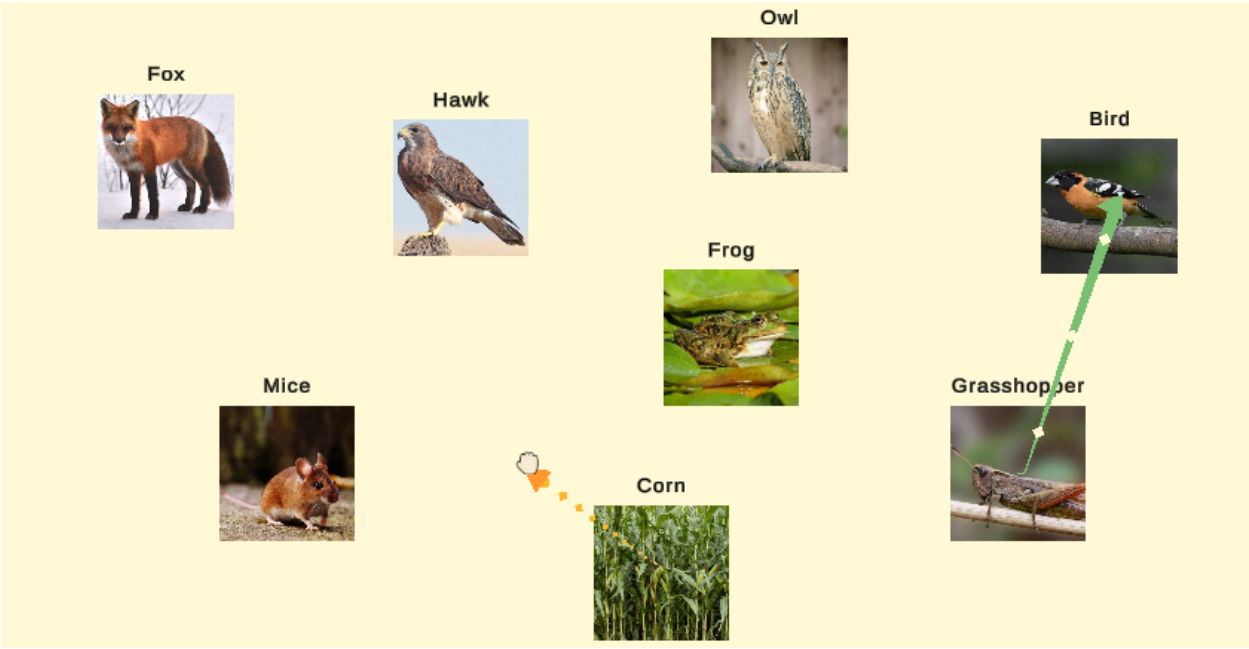
It uses the Connection Label Rule extension and the default Context Menu.



8.2. Connect the Food Chain

This example is a simple educational puzzle where the player needs to connect the item from below the food chain to the correct ones above it.

Its manager has a method that is called every time a new connection is created. The new connection is validated using a list of possible correct connections, if correct the line color is set to green.

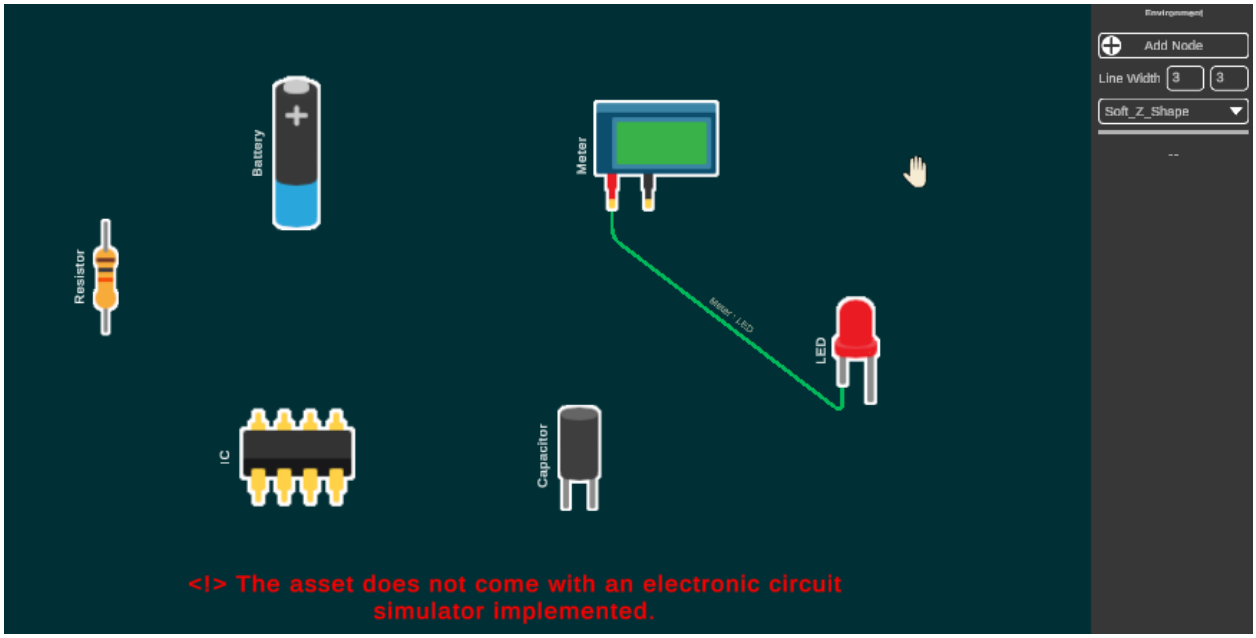


8.3. Circuit Board

This scene is an idea of use for the system, where it was customized to represent a circuit board with Nodes as components, and Connections as wires.

Observation: The asset does not come with an electronic circuit simulator implemented.

This implementation idea can be achieved with some work on the integrating or coding of a circuit simulator library to the project and modeling of the needed components, while a parser can be written using UICSystemManager.Nodes or graphManager.localNodes to get all the circuit elements and their connection points.

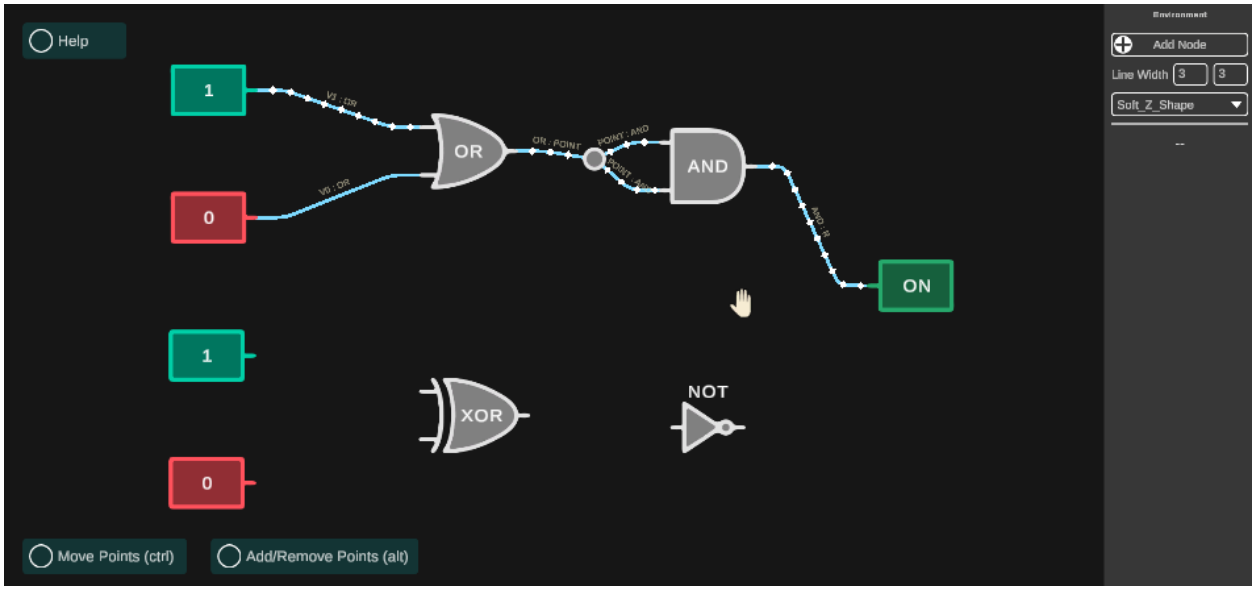


8.4. Logic Gates

The Logic Gates scene is a working logic gates solver where various schemes can be tested, including gate locks and latches.

This scene is a good UIC usage example, it contains:

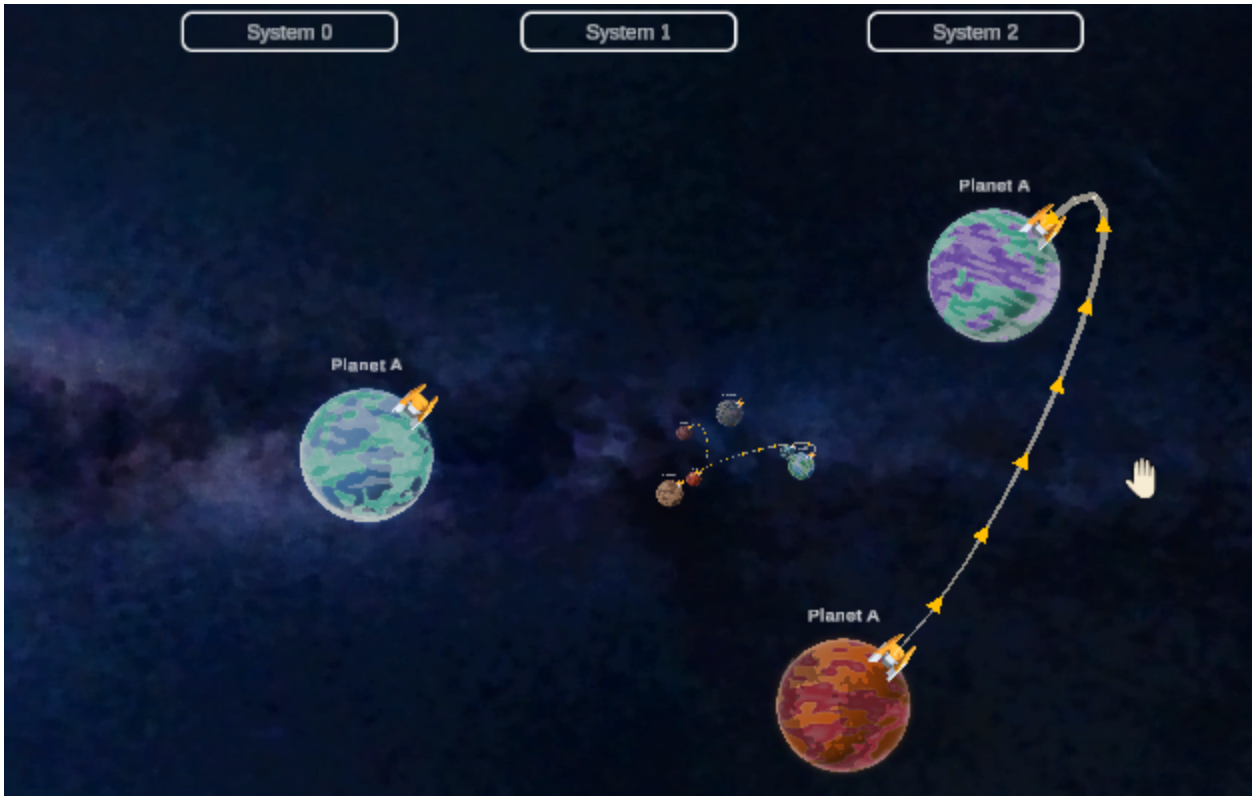
- A solver that recursively runs through the nodes in a backwards way and solves the gates in a forward way;
- An implementation of a way to split connections with a click on the connection by adding a node in the pointer position between the two new connections;
- An example of classes that model the Nodes into objects that can be used by the solver;
- Connections with different styles to represent different states and data flow.

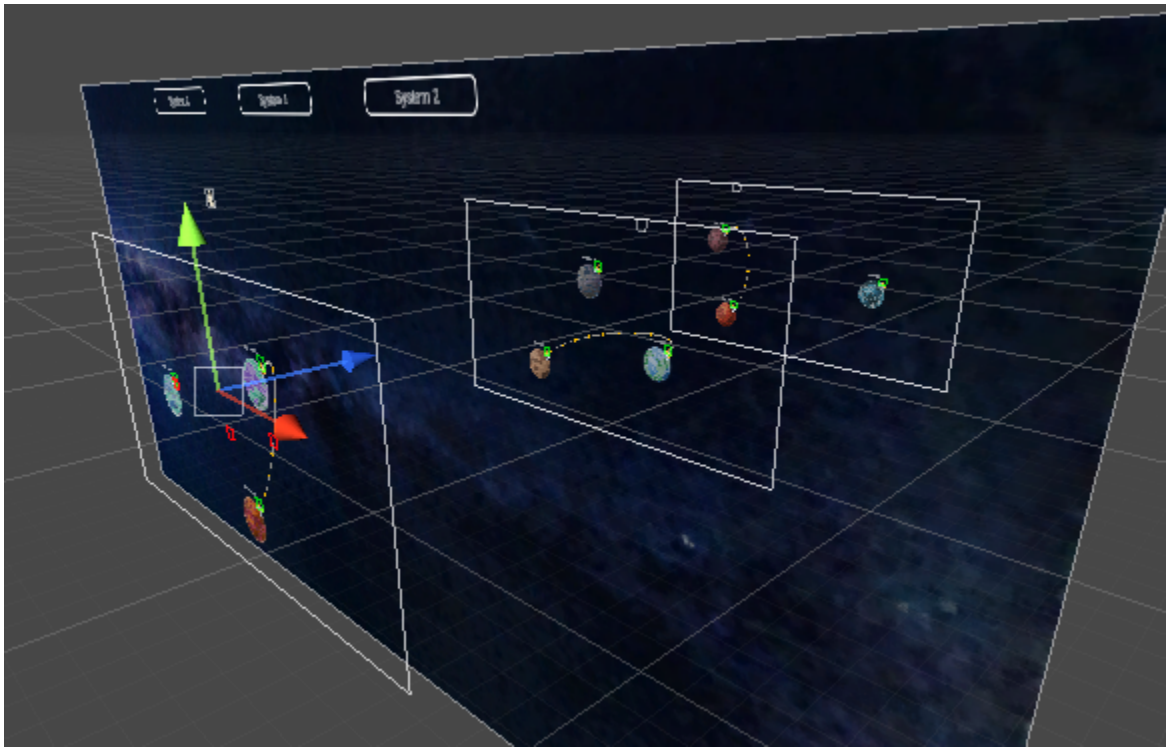


8.5. Multiple Charts

This scene shows the use of 3 Graph Managers in the scene and the Canvases set to World Space Render Mode.

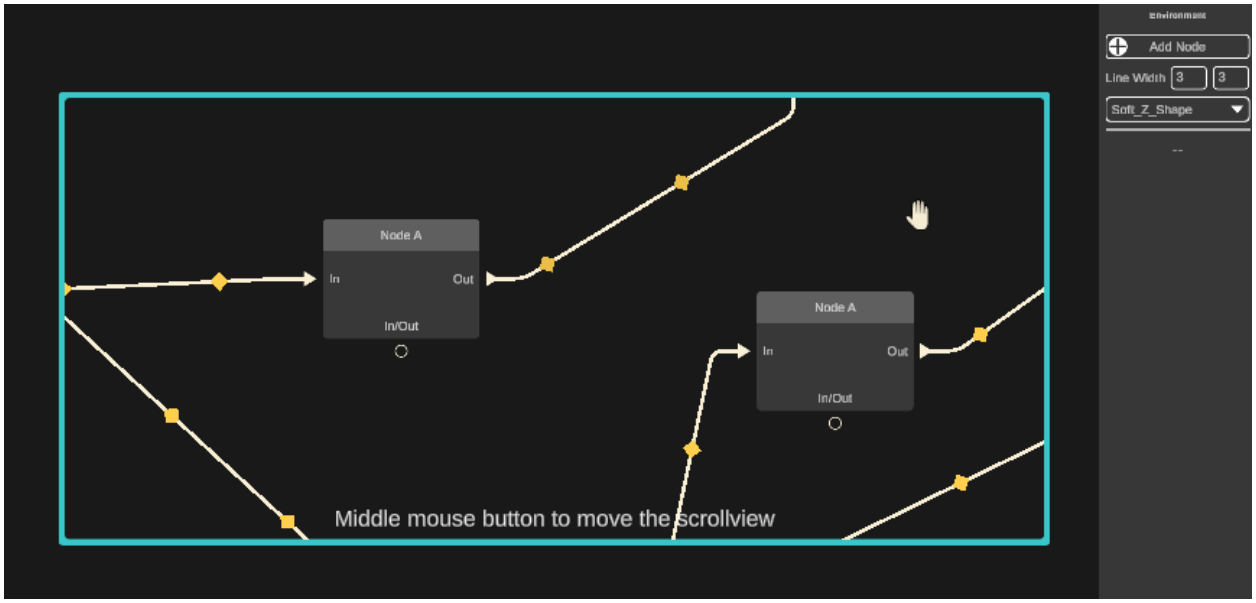
Each top button (System 0 to 2) makes the camera move to be in front of the respective graph, enables its Graph Manager component and disables the other ones.


System 0
System 1
System 2
Planet A
Planet A
Planet A



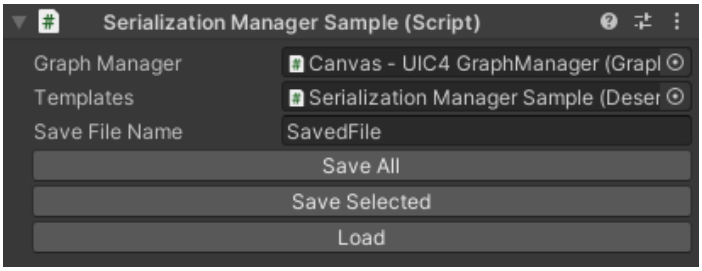
8.6. ScrollView Graphs

This scene has an example of a graph inside a ScrollView component where the Connection lines are also masked. You can move the ScrollView with the middle mouse button and it has a very basic zoom by scrolling the mouse wheel.

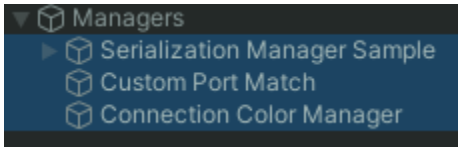


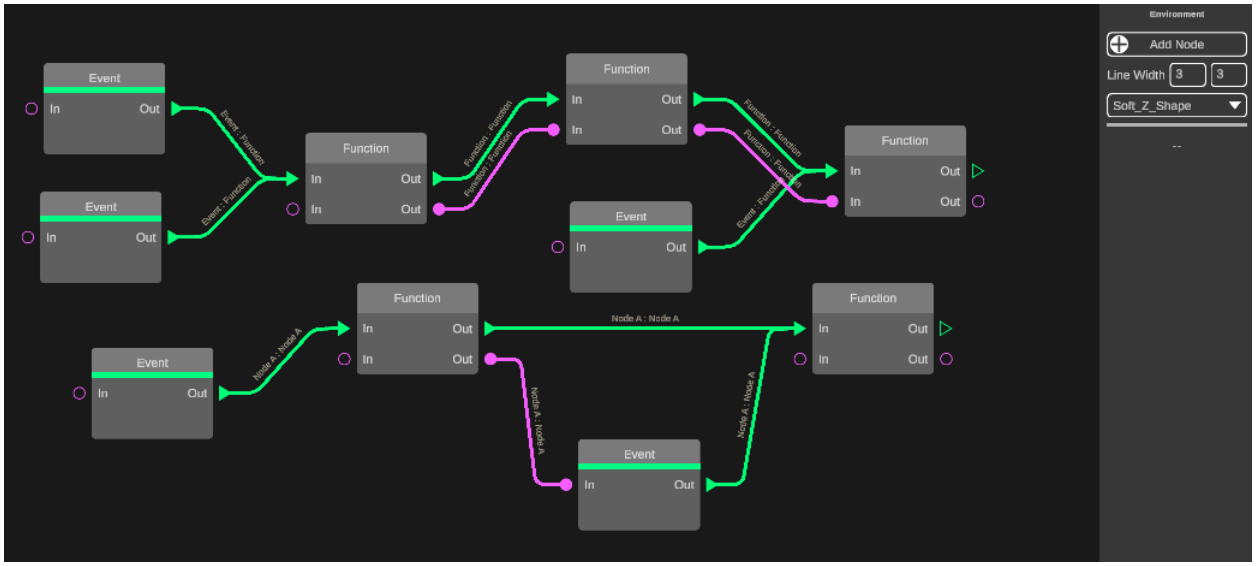
8.7. Serialization Sample

This scene is an usage example for the UICSerialization namespace. It is possible to save the whole graph or the selected elements to a file in JSON format and load the file into scene elements.



The scene also contains a Custom Port Match component that defines that only ports with the same ID can be connected to each other and a Connection Color Manager that have methods subscribed to SerializationEvents.




Environment
Add Node
Line Width
3
3
Soft_Z_Shape
--
Event
In
Out
Function
In
Out
Event : Function
Function : Function
Node A : Node A