Required Dependencies:

The Roguelike Map System requires the Newtonsoft JSON(Json .NET) package to work, if you do not have it on your Unity project already, you can install it by the following steps:

1. Open your Unity Package Manager.
2. Click Install package by name.
3. Use the following to install the Json .NET package: “com.unity.nuget.newtonsoft-json”

Tutorial:

To build the sample scene from scratch:

1. Setup the Roguelike Map System
    - Add the "RoguelikeMapScrollView" to the Canvas
        - In the Prefab folder, drag the RoguelikeMapScrollView prefab into your main Canvas. This object will serve as the container for the map.
    - What’s Inside the "RoguelikeMapScrollView" Prefab?
        - Inside the prefab, you’ll find a MapGenerator object. This object is responsible for:
            1. Creating a new map or loading existing map data based on the map config index.
            2. Populating the map and displaying it as the content within the scroll view.
2. Customizing the Path Node
    - If you’d like to customize your path nodes, you can modify the PathNode Prefab used in the MapGenerator.
    - Additionally, you can adjust the size and spacing of the nodes by modifying the corresponding settings in the MapGenerator component.
3. Set Up the MapInfoManager
    - Create an empty GameObject in the Scene and add the MapInfoManager script to it. This manager:
        - Loads the saved map data when the game starts.
        - Saves the map data when needed.
    - The map data is saved at: Application.persistentDataPath + mapDataSavePath
4. Canvas Settings
    - These are the only settings required to set up the RoguelikeMap system within the Canvas. Now you’re ready to generate and customize maps!

Creating Dummy Data to Test the System

1. Create MapConfigs

- MapConfigs are the files that store all the information about your maps.
- For more details about what's included in a MapConfig, refer to the relevant sections in the documentation.
- You can also use the two Sample MapConfigs provided in the Sample Folder.

2. Set Up Buttons for Map Generation

- Create UI buttons and attach the MapGenerationTester script to them.
- For each button, assign the OnClick callback to the OnClick function in MapGenerationTester.

3. Generate Maps

- Once you click each button, a map will be generated based on the corresponding MapConfig file that you've assigned.

Understanding the Map Structure

1. MapNode

- MapNode is an abstract class that serves as the foundation for all interactable nodes on the map. To create different types of nodes, you should extend this class and override their behaviours as needed.
- Each MapNode contains a NodeData object specific to its node type.
- The sample scene includes two examples:
  - EliteMapNode
  - MinionMapNode
- When interacted with:
  - Clicking on an EliteMapNode will print:
    "Elite Map Node: {MapLevel Name} Cleared"
  - Clicking on a MinionMapNode will print:
    "Minion Map Node: {MapLevel Name} Cleared"
- To customize your own MapNode, override the OnClick function in your subclass.

2. NodeData

- NodeData is a base class that holds fundamental node-related information, such as:
  - Node position
  - Connected nodes
  - Node index
- To add data specific to certain node types, extend this class.
- The sample scene provides two specialized NodeData classes:

        - EliteNodeData
        - MinionNodeData
    - Both of these include an additional field:
        - MapLevel Name → Identifies the level associated with the node.
    - When an EliteMapNode prints:
        - "Elite Map Node: {MapLevel Name} Cleared"
        - It retrieves the MapLevel Name from its EliteNodeData. The same applies to MinionMapNode with MinionNodeData.

System Components:

Map:

- List<List<NodeData>> mapNodes
    - All nodes included in this layer, sorted by layer.
- List<PathNodeData> pathNode
    - The position and direction data for all pathNodes in this map.
- String MapName
    - Name of this Map
- Int MapIndex
    - Index of this map, it would be the same as the Index of the MapConfig used to create this map.

MapNode:

- Image nodeImage
    - The Unity GUI Element placeholder for display NodeIcons.
- Button nodeButton
    - The Unity GUI Element placeholder for trigger OnClick Function.
- Sprite lockedImage
    - The Icon of the MapNode when it is locked.
- Sprite attendableImage
    - The Icon of the MapNode when it is unlocked.
- Sprite clearedImage
    - The Icon of the MapNode when it is cleared.
- NodeData nodeData
    - NodeData specific to its node type.

NodeData:

- Vector2 position
    - On Screen Position of the Node.
- List<NodeData> Incoming

    - List of Incoming connections to this Node.
- List<NodeData> outgoing
    - List of outgoing connections to this Node.
- NodeType nodeType
    - NodeType of current Node
- Int nodeIndex
    - Node Index
- NodeState nodeState
    - The current state of this Node(Locked, visited, cleared etc.)
- Int layerCount
    - The number of the layer this Node belongs to.

MapConfig:

Map config includes every detail of your map including:

- List<NodeTemplateData> MapNodeTemplates:
    - This is the list to store all the templates of each type of map node that will be used in this map config. If there are any types of nodes with no template stored in this list, they will not be populated.
- List<NodeTypeWeightData> NodeTypeWeights
    - This list stores the weight data for each type of map node, basically, when populating paths, node types with higher weights will be more likely to have more paths connected to it, if there is any type that does not have its according weight stored in this list, it will have weight 0 by default. Weights can not be negative.
- Int MapConfigIndex:
    - This index will be used as the Index for the generated maps, the MapInfoManager will use this index to store map Data, if there is a match in MapInifoManager saved data, the map data will be loaded, otherwise a new map will be generated based on the MapConfig settings.
- Int nodeWidth;
    - The Width of each node in pixel (Nodes are defaulted to have a Square shape so Width = Height
- Int nodeSpacing;
    - The space between nodes in the same Layer, depending on the orientation settings
- Int layerWidth;
    - The width of each layer is in pixels, depending on the orientation settings.
- Int nodeOffset;
    - The position offset range for each Node
- List<MapLayer> layers:
    - This List stores the layer settings for each layer in this map.

Map Layer:

Map Layer stores settings for each layer in Map including:

- Bool randomizedLayer
    - If this is checked, then this layer is randomized, all nodes in this layer will be created randomly using the settings in this layer.
- List<PredefinedNodeData> layerNodes:
    - This list will only be used if the "randomizedLayer" is not checked, and if this layer is not randomized, then all of the nodes in this layer will be created using the data saved in this list.
- List<NodeTypeWeightData> weightedPotentialNodeTypes:
    - If this layer is randomized, then weights in this list will be used to determine the chances to populate the node of each type, types with higher weight will be more likely to be created in this layer.
- Int minNodeSize;
- Int maxNodeSize
    - If this layer is randomized, then these two variables will be used to determine the total number of nodes in this Layer.

FAQs:

1. Why my NodeData is not deserialized properly?
    a. RoguelikeMap System using NewtonSoft.Json Package to serialize/deserialize Map object to achieve Save/Load functions, which include a List of NodeDatas for each Map Node in the current Map, so if you have a list of objects that may contain a looped reference, or read-only fields which are not primitive types, then save and load system may not work for you. To avoid potential errors:
        i. Always save Object indexes instead of references, and load the real Object reference after deserialization.
        ii. Try not to use ReadOnly unless it is a primitive type(int, float, enums etc.)