easyGrid documentation

Table of content

easyGrid generator

easyGrid is a tool that allows the easy creation and configuration of grids in a 3D space. The user can create various grid configurations to be later used in different game-like contexts. The software was developed in Unity 2020.2.3f1 (64-bit).

Setup

You need to do the following steps:

1. Import unity package.
2. Open the GridGenerator scene.
3. Change the parameters of the grid and click the "Generate Grid" button. The UI has:
    1. Set up cell grid parameter section that modifies individual cells of your grid, and;
    2. Manipulate grid section that changes the position and rotation of the whole grid.
4. When satisfied with the grid, click "Save Grid."
    1. You can save as many different types of grids as you want! All your parameters will be kept in the *GridSaveFile.json* file (*GridSystem/Model/GridSaveFile.json*)
5. Open your current game scene where you want to apply the grid to the following:
    1. Create an empty game object (this object will contain your items) in your scene.
    2. Add the *"SaveLoadGridConfig"* file anywhere in your scene.
    3. To use your grid (and its parameters), you need three lines of code that you should add to the file that is responsible for executing the grid:

```
private IGenerator gridGenerator;
//Reference of your empty game object
[SerializeField]
private GameObject _gameContainer;
//List of game objects that you want to display in your scene
[SerializeField]
private List<GameObject> _myGameItems = new List<GameObject>();

// Start is called before the first frame update
void Start()
{
//Create a grid object and add an empty object as a parameter
gridGenerator = new GridGenerator(GameContainer);

//Setup grid parameters by adding the array index position in the
JSON FILE
gridGenerator.SetUpJsonGrid(0);

//Add your list items to the new grid!
gridGenerator.AddItemsToGrid(MyGameItems);
}
```

    4. And that's it!

NOTE: you can do a quick test by opening the GridTestScene in GridSystem/UnityDemo. Also, you can see a YouTube tutorial on the Grid Generator: https://www.youtube.com/watch?v=Z2FCzNWLjeU

Generate Grids manually

If you do **not** want to create a grid based on the parameters saved in the JSON file, you can call another gridGenerator constructor in which you add parameters you see fit. The constructor has the following parameters:

```
/// <summary>
        /// Call this constructor if you want to create a Grid manually
        /// </summary>
        /// <param name="columnLenght"></param>
        /// <param name="rowLenght"></param>
        /// <param name="xSpace"></param>
        /// <param name="ySpace"></param>
        /// <param name="xStart"></param>
        /// <param name="yStart"></param>
        /// <param name="zStart"></param>
        /// <param name="scale"></param>
        /// <param name="GameObjectRotation"></param>
        /// <param name="emptyGameObject"></param>

public GridGenerator (int columnLenght, int rowLenght, float xSpace, float ySpace, float xStart, float yStart, float zStart, float scale, Vector3 rotation, GameObject emptyGameObject = null)
{
            this._columnLenght = columnLenght;
            this._rowLenght = rowLenght;
            this._xSpace = xSpace;
            this._ySpace = ySpace;
            this._xStart = xStart;
            this._yStart = yStart;
            this._zStart = zStart;
            this._scale = scale;
            this._rotation = rotation;
            this._emptyGameObject = emptyGameObject;

            _instanciateGameObjects = new List<GameObject>();
   }
```

Here is a script example using the constructor to generate grids manually:

```
 private IGenerator gridGenerator;
//Reference of your empty game object
[SerializeField]
private GameObject _gameContainer;
[SerializeField]
//List of game objects that you want to display in your scene
private List<GameObject> _myGameItems = new List<GameObject>();

// Start is called before the first frame update
public void Start()
{
    gridGenerator = new GridGenerator(
            5, //column Lenght
            3, //RowLenght
            2.0f, //x_space
            2.0f, //y_space
            0, //x_start
            0, //y_start
            0, //z_start
            1, //scale
            new Vector3(0f, 0f, 0f), //empty_GO_rotation)
            _gameContainer); //empty_GO

    //Add your list items to the new grid!
    gridGenerator.AddItemsToGrid(_myGameItems);
}
```

Generator parameters description

The grid generator UI (when opening GridGenerator scene) has a variety of parameters that the user can manipulate. Each parameter is described below:

Set up cell parameters:

- **Column** Length: Define the number of cell columns of your grid.
- **Row Length**: Define the number of cell rows of your grid.
- **XSpace**: Define the space between cells on the “x” axis.
- **YSpace**: Define the space between cells on the “y” axis.
- **XStart**: Define the beginning of the cells in relation to the empty game object on the “x” axis.
- **YStart**: Define the beginning of the cells in relation to the empty game object on the “y” axis.
- **ZStart**: Define the beginning of the cells in relation to the empty game object on the “z” axis.
- **Scale**: Define the scale of all cells.
- **Cell RotX:** Define the rotation of all cells on the “x” axis.
- **Cell RotY:** Define the rotation of all cells on the “y” axis.
- **Cell RotZ:** Define the rotation of all cells on the “z” axis.

Manipulate grid parameters:

- **Grid Pos X:** Define the position of the entire grid in the “x” axis.
- **Grid Pos Y:** Define the position of the entire grid in the “y” axis.
- **Grid Pos Z:** Define the position of the entire grid in the “z” axis.
- **Grid Rot X:** Define the rotation of the entire grid in the “x” axis.
- **Grid Rot Y:** Define the rotation of the entire grid in the “y” axis.
- **Grid Ros Z:** Define the rotation of the entire grid in the “z” axis.