目录

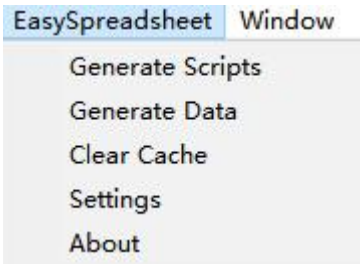
1 Introduction

In game development, we often use xlsx files to edit a lot of game constants, such as missions, equipment configurations, Tools are needed to convert the xlsx data into data that is usable at runtime.

EasySpreadsheet is a plugin that helps you easily use xlsx data in Unity games. It is easy to use, rich in features, flexible in setup, and does not rely on third-party libraries when running.

It parses xlsx files in the Unity editor and generates code and data for the game runtime.
It generates one ScriptableObject code and one data file for each sheet in the excel file.
When the game runs, EasySpreadsheet loads the generated data file, providing an API to query the data.

Once installed, its menu will appear in the Unity Editor menu bar:



When you click Generate scripts, a ScriptableObject type code file is generated for each sheet in the xlsx file.

When you click Generate data, you generate a ScriptableObject data file for each sheet in the xlsx file.

Features:

- Quick code and data generation with one click
- Support for all Unity versions, cross-platform
- Visual Scripting support
- Support types int, long, float, string, bool, List, Dictionary
- Support engine types Vector2, Vector3, Rect...
- Support for custom types
- Flexible Settings
- Rich examples

2 Files

（1）Xlsx files

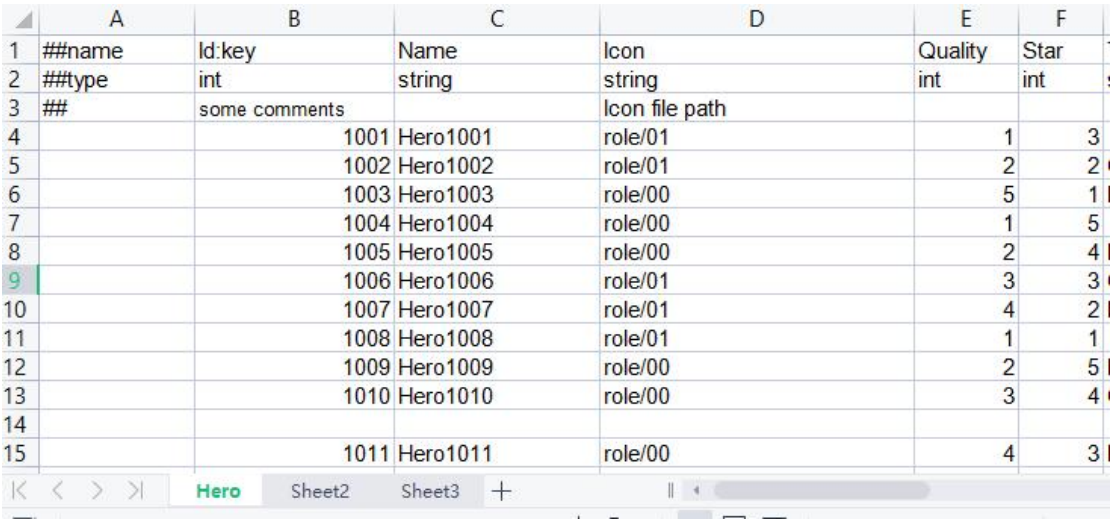
Use Hero.xlsx as an example, as shown below.

The first three lines of the first Hero page are specially marked,
The first line is the variable name (##name tag),
The second line is the variable type (##type tag),
The third line is the comment (## tag),
The fourth row and the following are the data.

Therefore, the Hero page conforms to the convention and is imported to generate code (hero.cs and HeroTable.cs) and data (hero.asset).

The first three lines of the other two sheets, Sheet2 and Sheet3, were unmarked, so they were ignored.

Note: The data page name does not have to be the same as the file name and can contain multiple data
pages.



	A	B	C	D	E	F
1	##name	Id:key	Name	Icon	Quality	Star
2	##type	int	string	string	int	int
3	##	some comments		Icon file path		
4		1001	Hero1001	role/01	1	3
5		1002	Hero1002	role/01	2	2
6		1003	Hero1003	role/00	5	1
7		1004	Hero1004	role/00	1	5
8		1005	Hero1005	role/00	2	4
9		1006	Hero1006	role/01	3	3
10		1007	Hero1007	role/01	4	2
11		1008	Hero1008	role/01	1	1
12		1009	Hero1009	role/00	2	5
13		1010	Hero1010	role/00	3	4
14						
15		1011	Hero1011	role/00	4	3

Hero Sheet2 Sheet3

（2）The generated code file

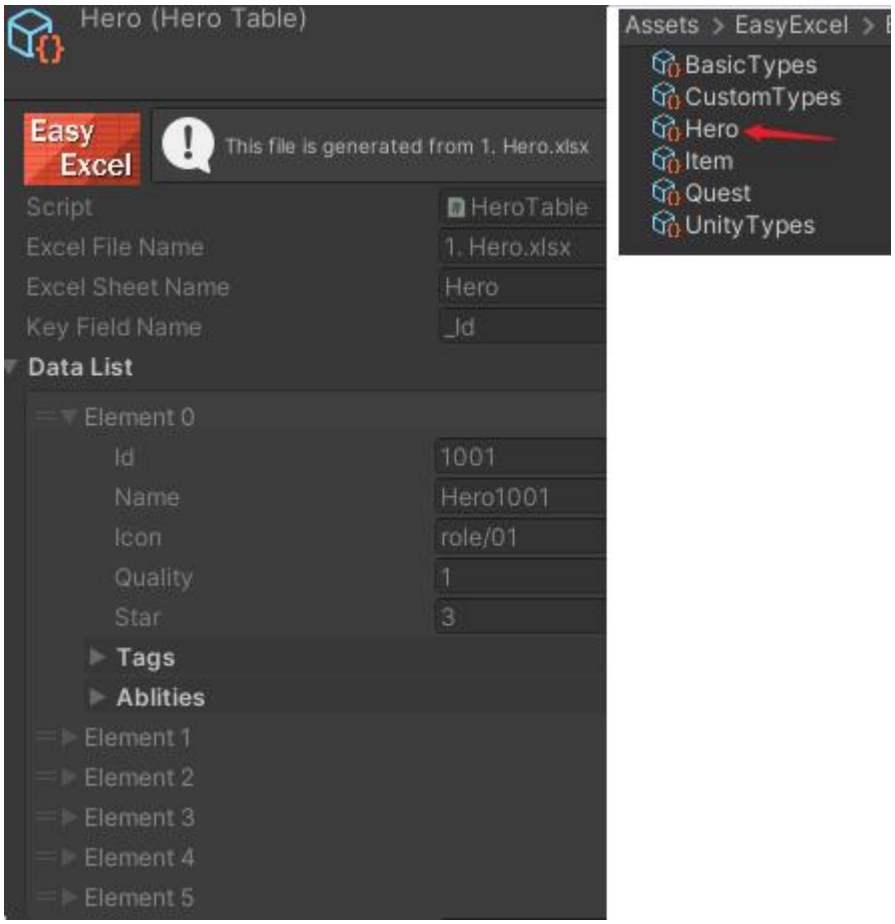
Because only the Hero page in hero.xlsx has data, only hero.cs and HeroTable.cs are generated.
The Hero class represents a row of data in the Hero page.
The HeroTable is a collection of Hero data that contains all the data imported from the Hero page.

```cs
[Serializable]
public sealed partial class Hero : EERowData
{
    [EEKeyField]
    [SerializeField]
    private int _Id;
    public int Id => _Id;

    [SerializeField]
    private string _Name;
    public string Name => _Name;

    [SerializeField]
    private string _Icon;
    public string Icon => _Icon;

    [SerializeField]
    private int _Quality;
    public int Quality => _Quality;

    [SerializeField]
    private int _Star;
    public int Star => _Star;
```

```cs
public sealed partial class HeroTable : EERowDataTable
{
    [SerializeField]
    private List<Hero> _dataList = new List<Hero>();

    private Dictionary<int, Hero> _dataMap;
    public Dictionary<int, Hero> DataMap => _dataMap;
    public List<Hero> DataList => _dataList;

    public Hero Get(int key) => _dataMap.TryGetValue(ke
    public Hero this[int key] => _dataMap.TryGetValue(k

    protected override void OnDeserialized()
    {
        _dataMap = new Dictionary<int, Hero>(_dataList.
        foreach (var data in _dataList)
        {
            _dataMap.Add(data.Id, data);
            (data as ISerializationCallbackReceiver).On
        }
        PostInit();
    }
```

（3）The generated data file

The hero. asset file is generated, which is a ScriptableObject file of type HeroTable and contains all the Hero data.



（4）The generated class DataTables

A data management class, DataTables.cs, is also generated each time the code is generated. It is responsible for loading data and querying data.

DataTables contain both synchronous and asynchronous Load methods

DataTables generate static methods for each type of data, such as:

DataTables.GetHero(1001)

DataTables.GetHeroList()

Note that it is important to initialize the load data before calling these static methods.

```
public partial class DataTables
{
    public static DataTables Instance => s_instance;

    private Dictionary<Type, EERowDataTable> _cfgs;
    public Dictionary<Type, EERowDataTable> Cfgs => _
    public BasicTypesTable BasicTypesTable {get; priv
    public CustomTypesTable CustomTypesTable {get; pr
    public HeroTable HeroTable {get; private set;}
    public ItemTable ItemTable {get; private set;}
    public QuestTable QuestTable {get; private set;}
    public UnityTypesTable UnityTypesTable {get; priv


    public void Load(IEEDataLoader loader)
    {
    public IEnumerator LoadAsync(IEEDataLoader loader
    {

    public static Hero GetHero(int id)
    {
        return s_instance.HeroTable.Get(id);
    }

    public static List<Hero> GetHeroList()
    {
        return s_instance.HeroTable.DataList;
    }
```

3 Runtime API

（1）Initialize load data

Create a DataTables instance and call Load to complete the initialization.

```
public class InitializeEasyExcel : MonoBehaviour
{
    private readonly EE.DataTables _dataTables = new EE.DataTables();

    private void Awake()
    {
        // Load all data when app starts.
        _dataTables.Load(new EasyExcel.EEDataLoaderResources());
    }
}
```

（2）Get Data

In the Hero example, obtain the hero configuration with id 1001

```
// Get hero with id 1001.
EE.Hero cfg1 = EE.DataTables.GetHero(1001);

Debug.Log("Hero's name is " + cfg1.Name);
```

Get the all hero configuration

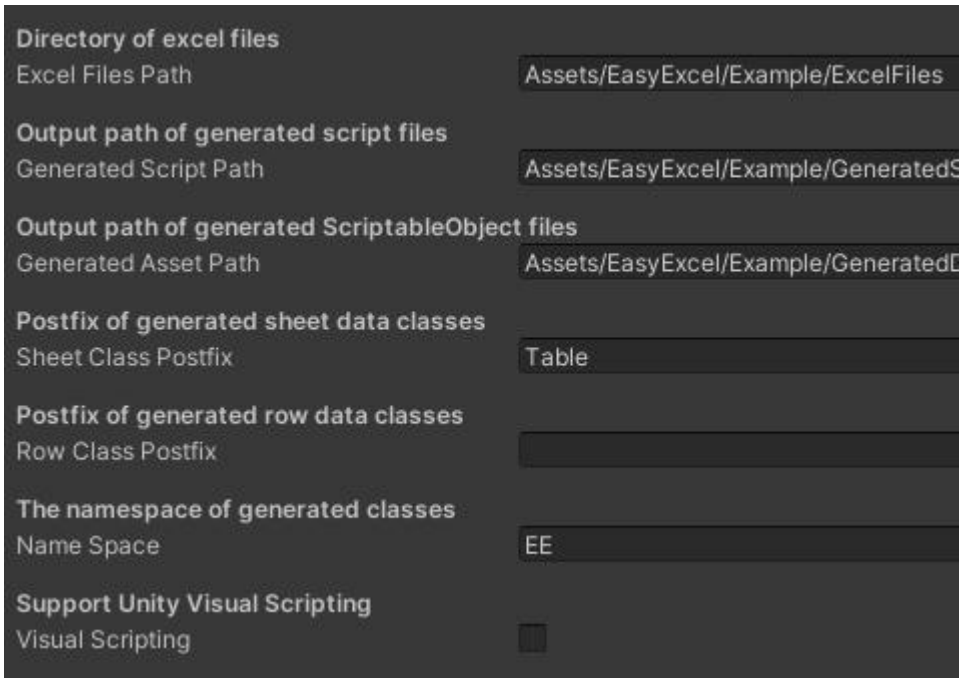
```
// Get all heros'data.
List<EE.Hero> heros = EE.DataTables.GetHeroList();
```

4 Settings

（1）Settings UI

Click EasySpreadsheet/Settings to open the screen.

Here you can set input and output paths, class name rules, and Unity's Visual Scripting.



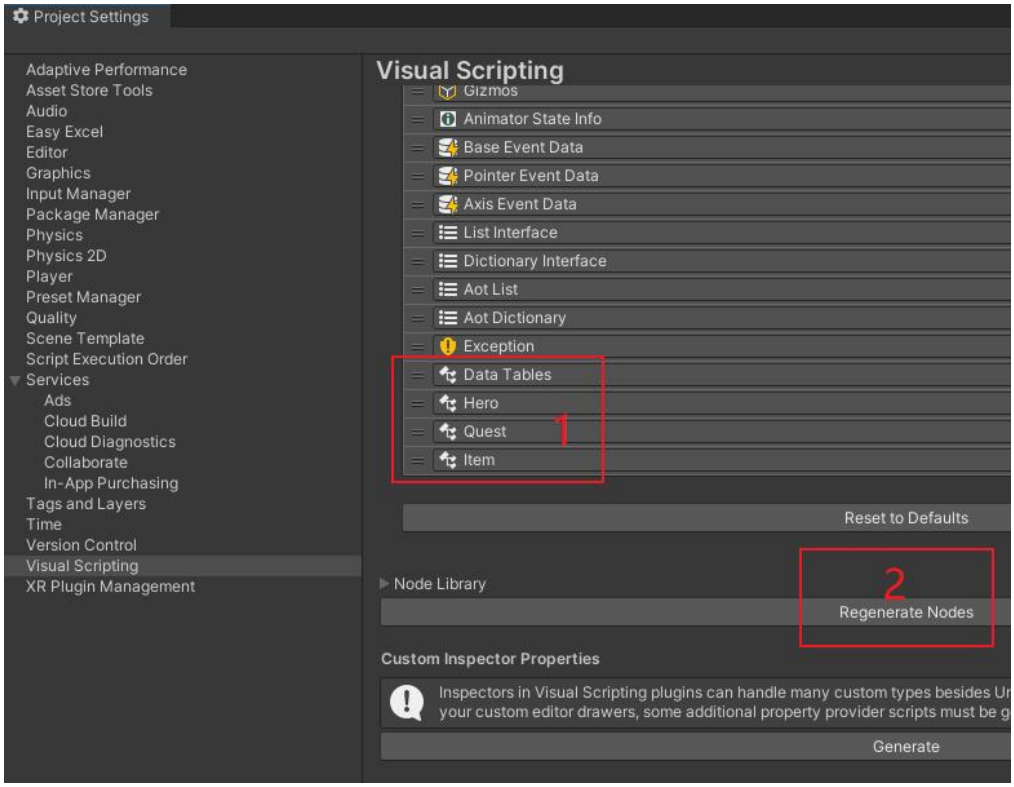
（2）How to Enable Visual Scripting

Check Visual Scripting in the interface

Click Generate Scripts to regenerate the code

In the ProjectSettings/VisualScripting, join the TypeOptions the generated type

Click Regenerate Nodes



In the Example/Scene/VisualScripting can view the sample:

5 Examples

（1）PredefinedTypes.xlsx

_PredefinedTypes.xlsx is a fixed file that you can use to add custom types.

Notice you can't delete it.

A	B	C	D	E	F
##var	full_name	separator	comment	fields	
##var				name	type
##	全名(包含模块和名字)	分割符		字段名	类型
	UnityEngine.Vector2	-		x	float
				y	float
	UnityEngine.Vector3	-		x	float
				y	float
				z	float
	UnityEngine.Rect	-		x	float
				y	float
				width	float
				height	float
	Int3	-		x	int
				y	int
				z	int
	Int2	-		x	int
				y	int
	Int4	-		x	int

（2）BasicTypes.xlsx

The BasicTypes page of BasicTypes.xlsx demonstrates the base types supported: int, long, float, string, bool, List, Dictionary.

For List, Dictionary, you can write (sep=;) after the type. To specify the element separator, Use such as (sep=;) , (sep=#), (sep=,). If omitted (sep=;) , Easy Excel uses the default delimiter '; '.

D	E	F	G	H	I	J	K
num1	num2	num3	bool1	nums	strs	nums2	dict1
int	float	long	bool	int[](sep=;)	string[](sep=;)	float[](sep=;)	< int , int >
20	3.1	1	TRUE	1;2;3;4;5	a123;b456;c78	3.1;3.2;3.3	1 : 11 ; 2 : 22; 3: 33 ;
80	4.2	2	FALSE	1;2;3;4	eeee	4.2;4.3	1:11;2:22;3:33;
100	5	3	TRUE	1;2;3	a;bbbb	5	1:11;2:22
1000	50.8	4	FALSE	1;2	cccc	50.8	1:11
120	25	5	TRUE	1	ddddd	25	
160		6	FALSE				
200		7	TRUE				

（3）CustomTypes.xlsx

The CustomTypes page in CustomTypes.xlsx shows how to use CustomTypes that need to be defined first in _PredefinedTypes.xlsx.

C	D	E	F
point2d	point3d	point3ds	point2dDict
Int2	Int3	Int3[]	<int,Int2>
5-6	5-6-7	5-6-7;2-3-4	111:5-6
3-9	2-3-4	2-3-4;	222:3-9
1-2	3-4-5		333:1-2
4-7	4-5	4-5-6	444:4-7
			222:3-9 ; 333:1-2 ; 501:6-7

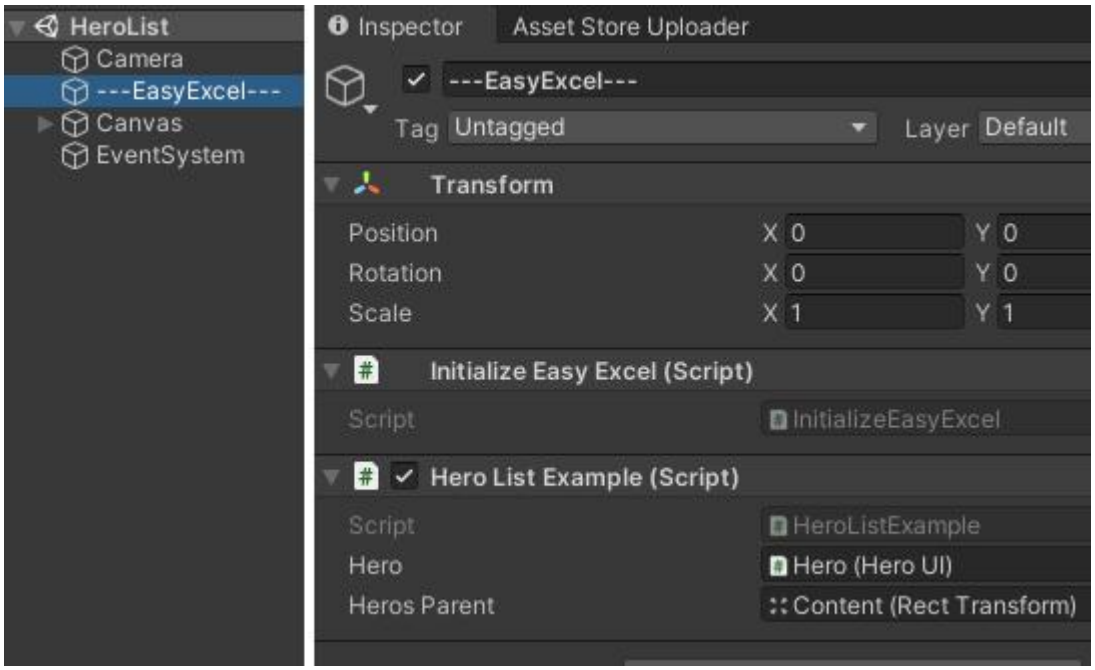
（4）UnityEngineTypes.xlsx

The UnityEngineTypes.xlsx page shows how to use Unity's built-in types, which need to be defined in _PredefinedTypes.xlsx first.

C	D	E	F
vec2	vec3	vec3s	vec3Dict
UnityEngine.Vector2	UnityEngine.Vector3	UnityEngine.Vector3[]	<int,UnityEngine.Vector3>
5.1-6.2	5.1-6.2-7.9	5.1-6.2-7.9 ; 2-3-4	111 : 5.1-6.2-7.9 ; 402 : 2-3-4
1.3-2.5	3.2-4.2-5.7		
4.3-7.1	4-5	4-5-6	402 : 2-3-4

（5）Hero、Quest、Item

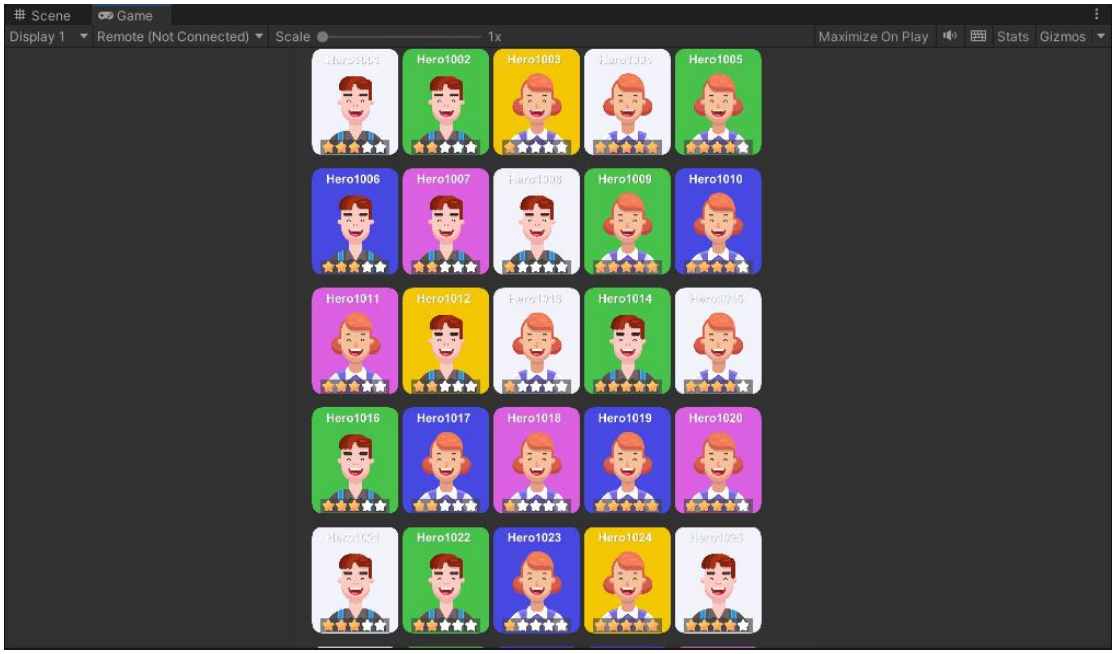
The Hero page in hero.xlsx demonstrates how to configure Hero character data,
Check out the scene file
EasySpreadsheet\Example\Scenes\HeroList.unity



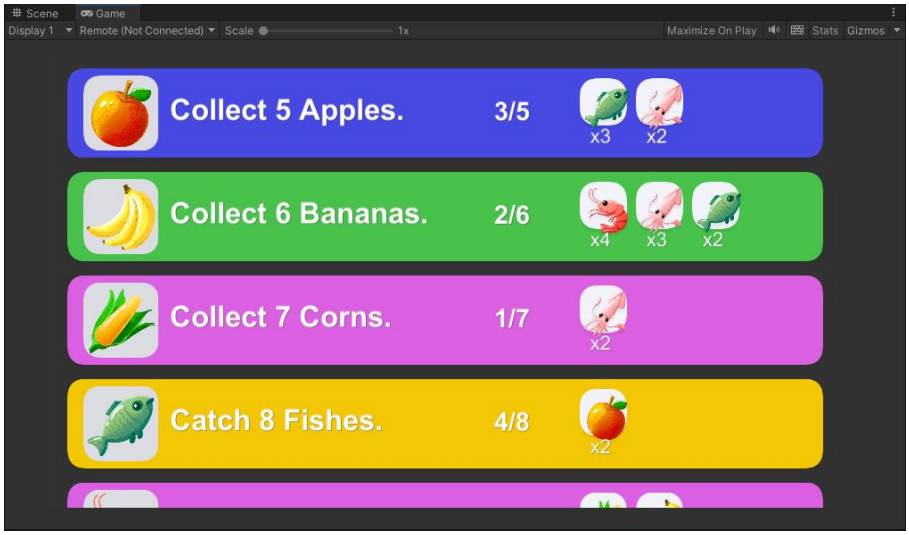
InitializeEasySpreadsheet.cs script shows how to initialize and load data.

HeroListExample.cs Get the list of heroes and display it.

The runtime screenshot is as
follows:



The Quest page in quest.xlsx shows how to configure the task data, and the Item page in item.xlsx shows how to configure the items. Where Quest references the Item table, look in the scene file EasySpreadsheet\Example\Scenes\QuestList.unity.



（6）Visual Scripting

Scene file EasySpreadsheet Example Scenes visualScript.unity

See [How to enable the Visual Scripting].

6 Support

QQ：1534921818

Mail：1534921818@qq.com