“Tools 23 – QR Code Generator” Document

Step by step guide to use this package:

After importing the package, you will see a folder named “Tools 23 – QR Code Generator” inside your “Assets” folder. In the Figure 2 you can see the third-party plugins that have been used in this project.

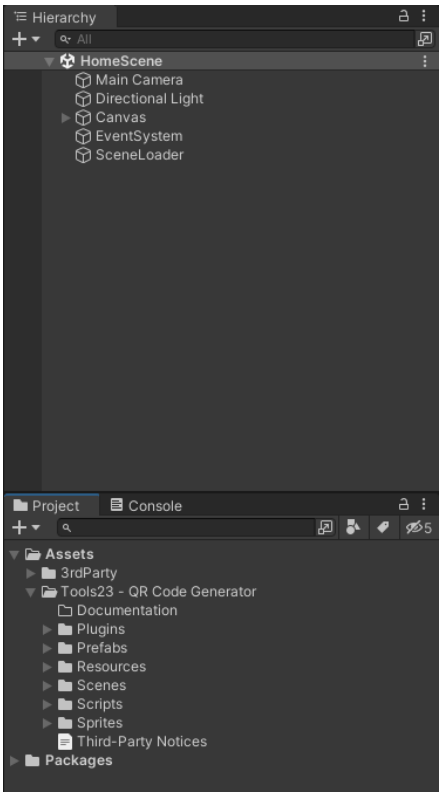

Figure 1

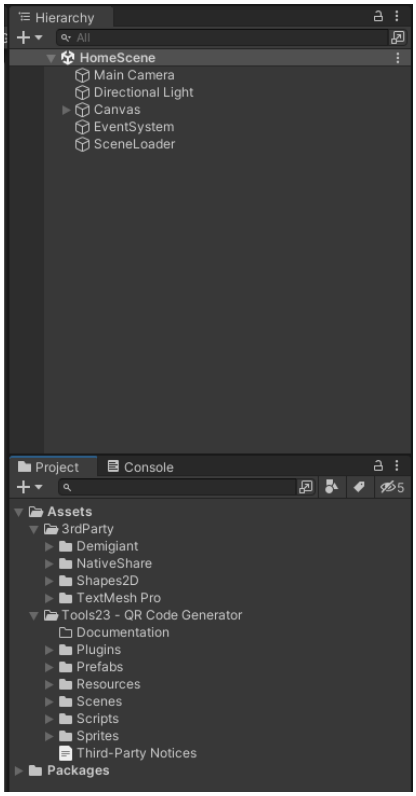

Figure 2

The links to these third-party plugins are given below:

DOTween (HOTween v2): [DOTween (HOTween v2) | Animation Tools | Unity Asset Store]

Shapes2D – Procedural sprites and UI: [Shapes2D - Procedural sprites and UI | Sprite Management | Unity Asset Store]

Native Share for Android & iOS: [Native Share for Android & iOS | Integration | Unity Asset Store]

Rest of the package you can download and import from the Package Manager of Unity editor.

In the Figure 3, you can see that our package has only two sprites that's been used. "backButton" sprite is used for both the back button and the home button. "close" sprite is used as the error sign.

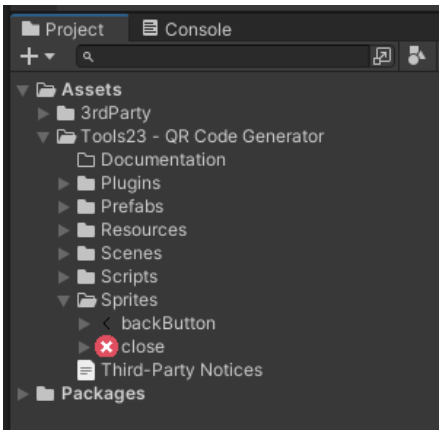

Figure 3

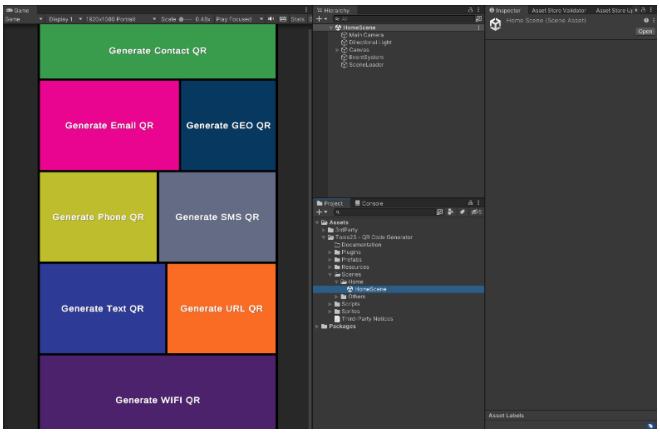

Figure 4

To open the "HomeScene" follow the Figure 4. As shown in the Figure 5, ButtonPanel can be expanded and you can see all the buttons responsible to switch from "HomeScene" to all the other QR Code generating scenes.

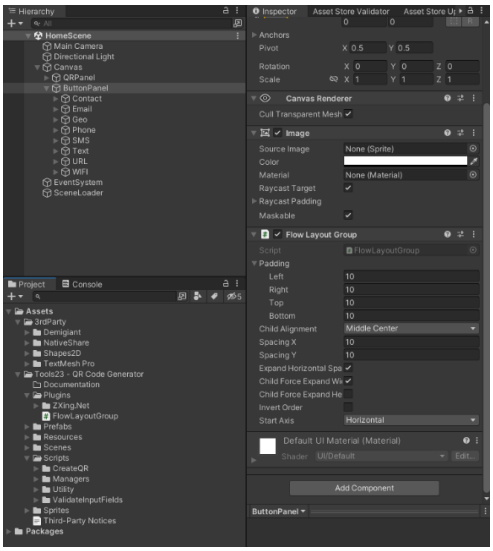

Figure 5

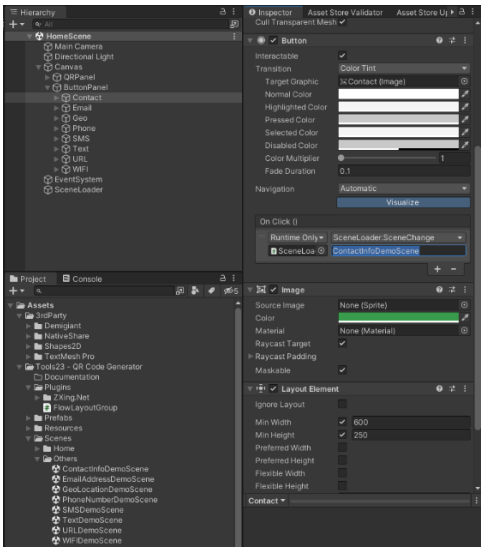

Figure 6

For example, you can select the "Contact" button, add "OnClick()", then drag and drop "SceneLoader" and choose "SceneChange" method. "SceneChange" method takes parameter which is a string that represents scene name. (Figure 5 – 7)

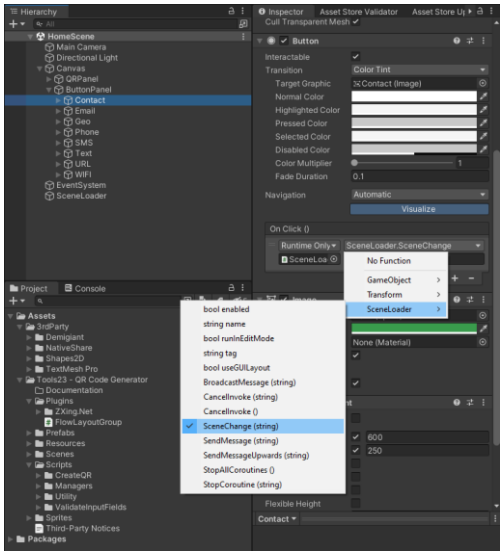

Figure 7

If you want to setup a Contact QR Generator UI, then go to the “Others” folder and drag “ContactInfoDemoScene” to the “Canvas” (Figure 8).

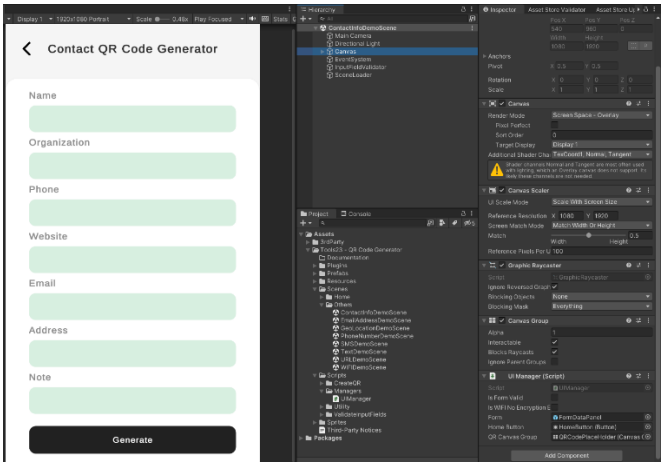

Figure 8

As shown in the Figure 9, add “UIManager” component in the “Canvas” (or you can just create an empty object and do the same in your scene). Drag “FormDataPanel”, “HomeButton” and “QRCodePlaceHolder” to the “Form”, “HomeButton” and “QR Canvas Group” field of the UIManager script respectively.

According to the Figure 10, you can create an empty gameobject called “InputFieldValidator” and add “ContactInfoInputFieldValidations” script as a component.

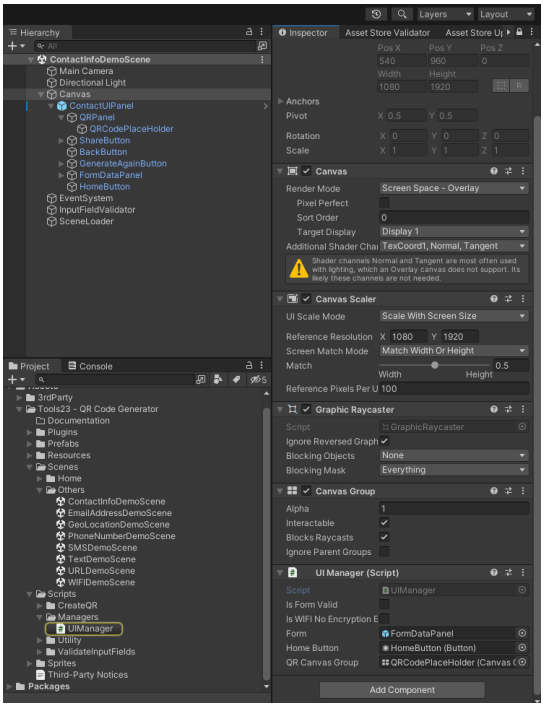

Figure 9

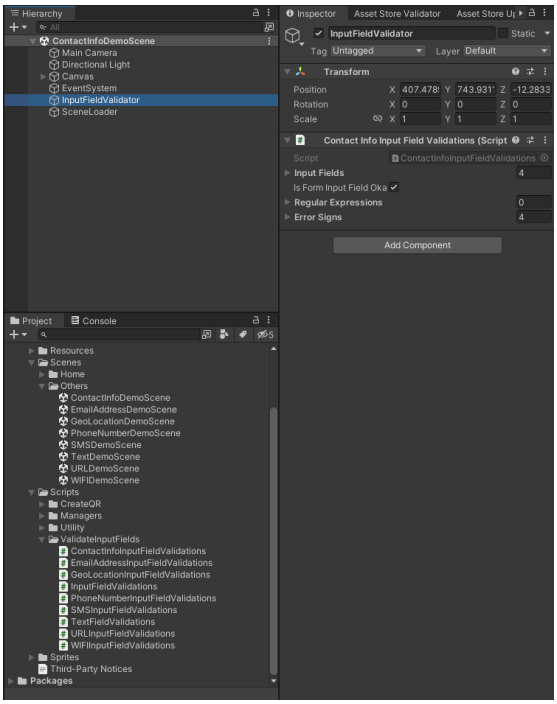

Figure 10

As shown in the Figure 11, drag and drop the “Name”, “Telephone”, “URL” and “Email” input field to the “Input Fields” list under the “Contact Info Input Field Validations” script. The mentioned input field has their corresponding “Label” with an image used as Error sign. Drag those images to the “Error Signs” list.

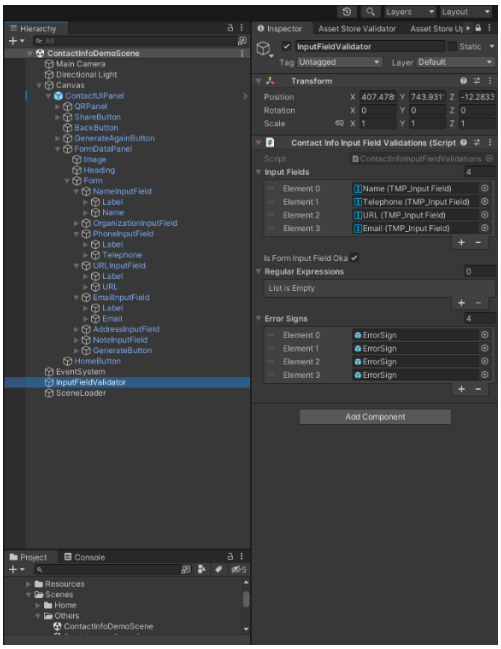

Figure 11

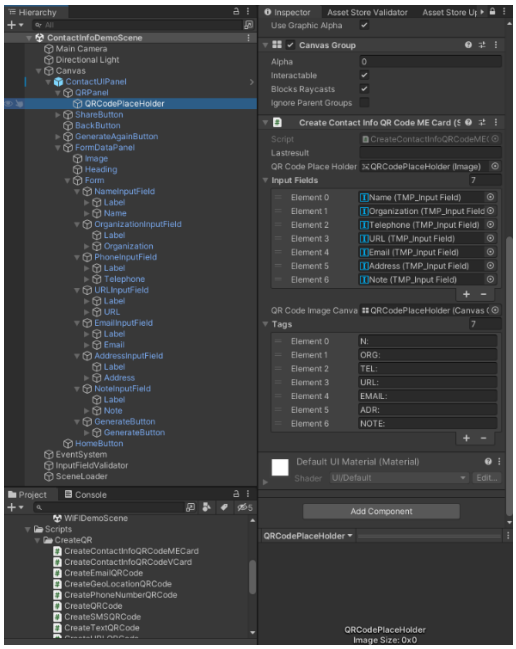

Figure 12

You can set the functionality of the other buttons as shown in the (Figure 13-17).

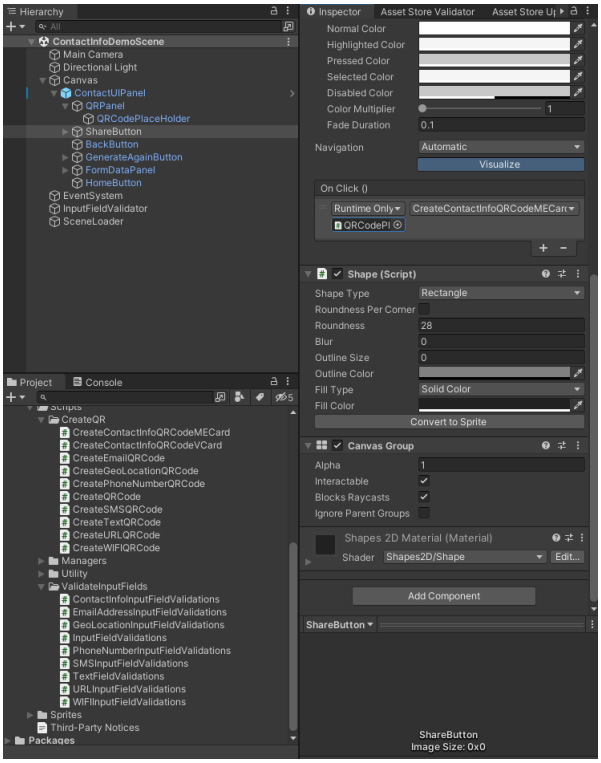

Figure 13

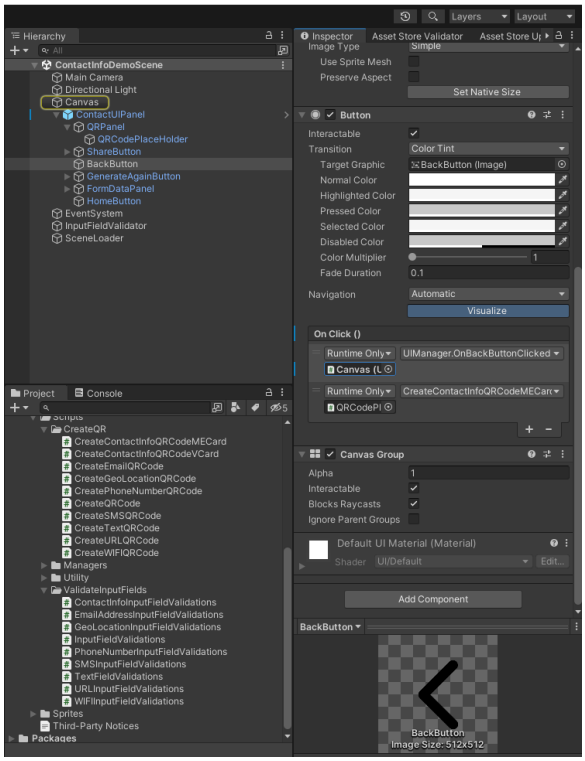

Figure 14

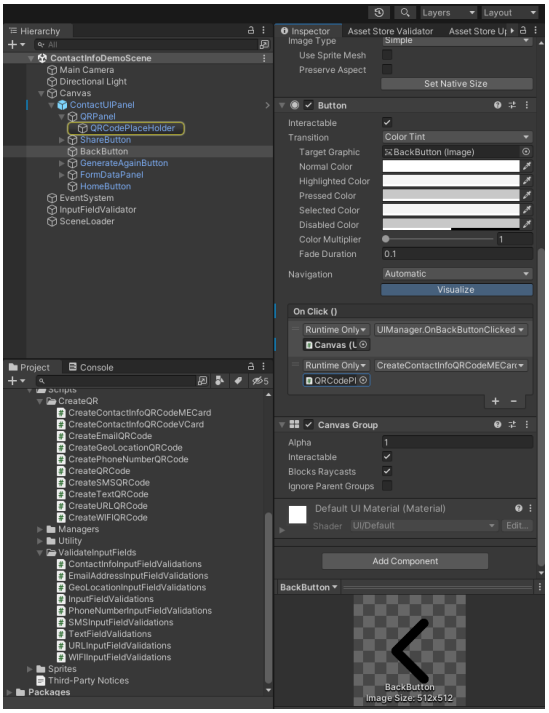

Figure 15

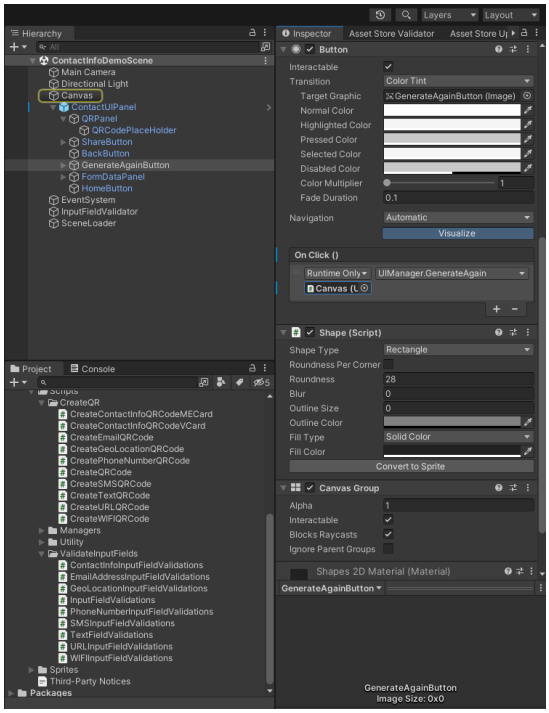

Figure 16

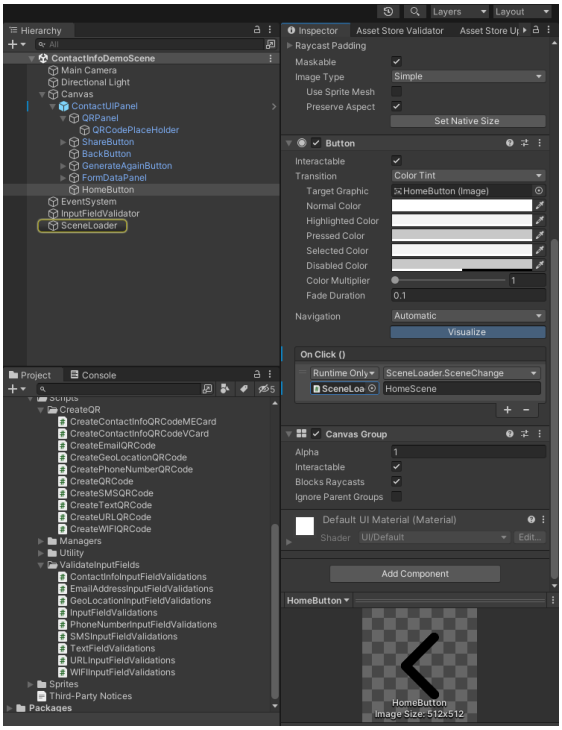

Figure 17

Scripts:

CreateQRCode: This is the main script that is responsible for creating the actual QR code (Figure 18 - 20).

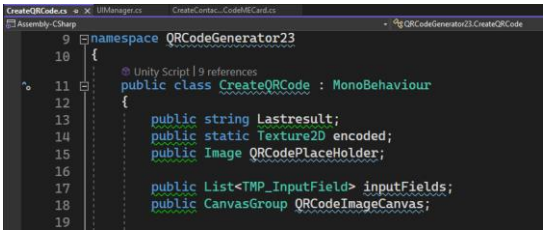

Figure 18

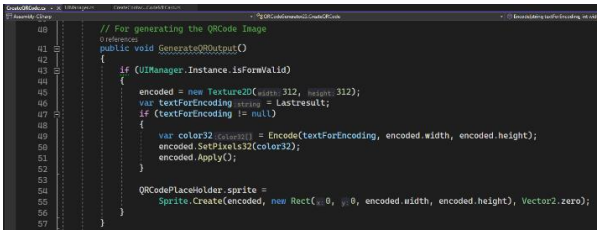

Figure 19

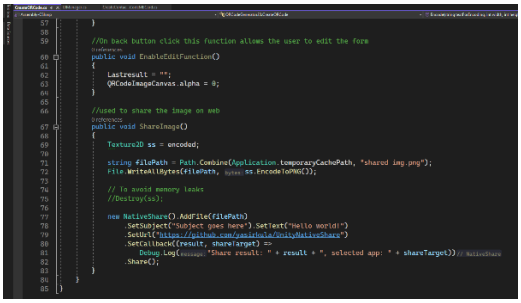

Figure 20

The “GenerateQROutput” creates the QR code. The “EnableEditFunction” enables the user to edit the form on clicking the back button. The “ShareImage” uses the “NativeShare” to allow the user to share the QR image on available platforms.

CreateContactInfoQRCodeMECard: This script is inherited from the “CreateQRCode” script.

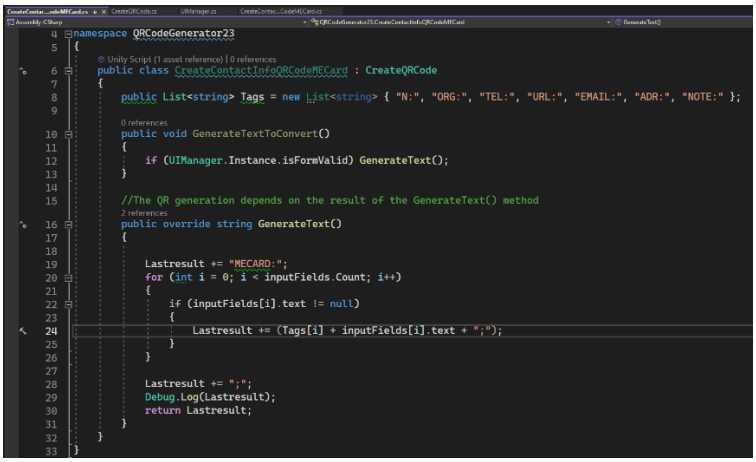

Figure 21

The “GenerateTextToConvert” function uses the “GenerateText” method and returns a string that is further used to create the QR Code.

InputFieldValidation: This is an abstract class and responsible for validating the input field values.

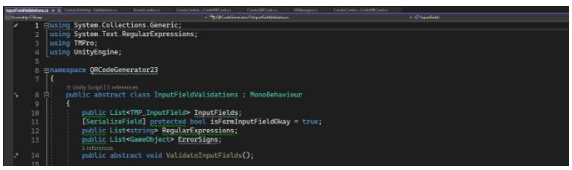

Figure 22

```
5 references
protected void ValidateFields()
{
    for (int i = 0; i < InputFields.Count; i++)
    {
        isFormInputFieldOkay =
            (isFormInputFieldOkay) && Regex.IsMatch(input: InputFields[i].text, pattern: RegularExpressions[i]);
        if (!isFormInputFieldOkay)
        {
            ErrorSigns[i].gameObject.SetActive(true);
            Handheld.Vibrate();
            break;
        }
        else
        {
            ErrorSigns[i].gameObject.SetActive(false);
        }
    }

    if (isFormInputFieldOkay) UIManager.Instance.isFormValid = true;
    isFormInputFieldOkay = true;
}
}
```

Figure 23

ContactInputFieldValidation: This class is inherited from the “InputFieldValidations” class.

```
namespace QRCodeGenerator23
{
    Unity Script (1 asset reference) | 0 references
    public class ContactInfoInputFieldValidations : InputFieldValidations
    {
        // Regex pattern for validating contact information
        private string namePattern = @"^[A-Za-z]+((\s)?((\'|\-|\.)?([A-Za-z])+))*$";
        private string phonePattern = @"^\+?[0-9]{0,3}[ -]?\(?[0-9]{3}\)?[ -]?[0-9]{3}[ -]?[0-9]{4}$";

        private string urlPattern =
            @"^((https?|ftp)://)?(www\.)?([a-zA-Z0-9_\-]+)(\.[a-zA-Z]{2,})(\.[a-zA-Z]{2,})?(/[\w\-\./\?\%\&=]*)?$";

        private string emailPattern = @"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";

        1 reference
        public override void ValidateInputFields()
        {
            RegularExpressions.Add(namePattern);
            RegularExpressions.Add(phonePattern);
            RegularExpressions.Add(urlPattern);
            RegularExpressions.Add(emailPattern);
            ValidateFields();
        }
    }
}
```

Figure 24

There are other scripts similar to the above mentioned scripts. You also need to follow the same process to setup the different QR code generating UI and get your desired output.