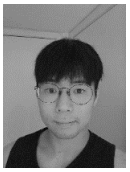
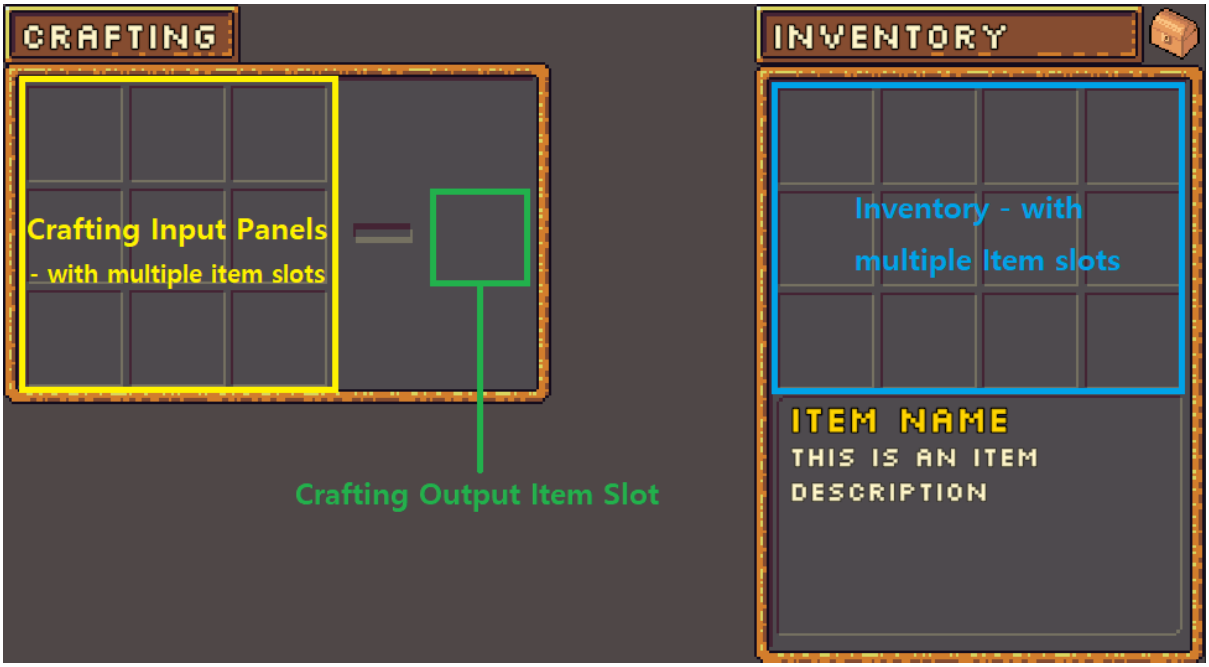


Minecraft-like Crafting System Package Documentation

GAME3023 – Game Engines 3.

Chaewan Woo (#101354291).

- **Scene Overview**



- **Item Behaviors-(for all item slots)**

```
public class ItemSlot : MonoBehaviour, IPointerClickHandler
```

```
public virtual void OnPointerClick(PointerEventData eventData)
{
    if (eventData.button == PointerEventData.InputButton.Left)
    {
        if (SlotAndCursor())
        {
            if (!SlotAndCursorSameItem()) // if not same item
                SwapItem();
            else
                StackAllItemCursorToSlot(); // put cursor item and stack them

        }
        else if (SlotAndNoCursor())
        {
            PickItem(); // pick up on item from the slot to the cursor
        }
        else if (NoSlotAndCursor())
        {
            PlaceItem(); // drop item from the cursor to the slot
        }
    }
    else if (eventData.button == PointerEventData.InputButton.Right)
    {
        if (SlotAndCursor())
        {
            if (!SlotAndCursorSameItem()) // if not same item
                SwapItem();
            else
                StackOneItemCursorToSlot(); // place one by one and stack them
        }
        else if (SlotAndNoCursor())
        {
            PickHalfOfItem(); // pick up half of the item from the slot to the cursor
        }
        else if (NoSlotAndCursor())
        {
            DropOneItemCursorToSlot(); // drop each item from the cursor to the slot
        }
    }
}
```

```
    OnPointerClick Events

    OnPointerClick Conditions
```

Most cases for the "**Events functions**", and "**Conditions functions**" are already **pre-made**. So, it can **just be re-arranged** like the above image **to create different item behaviors**.

- Item Behaviors-(for a specific item slot)

```
public class CraftingOutputSlot : ItemSlot, IPointerClickHandler
```

```
public override void OnPointerClick(PointerEventData eventData)
{
    if (SlotAndCursor())
    {
        if (SlotAndCursorSameItem())
        {
            StackAllItemSlotToCursor();
            UpdateInputPanel();
        }
    }
    else if (SlotAndNoCursor())
    {
        PickItem();
        UpdateInputPanel();
    }
}
```

To create a distinct item slot with unique behavior, simply set the ItemSlot class as a parent class, and use the provided "**Event functions**" and "**Condition functions**" that were made from the parent class. **It's as easy as that**.

- **Pre-made Recipe Samples**

- ## Recipe

```
public class Recipe : ScriptableObject
{
    public Item[] input = new Item[9];  // 9 = 3x3 crafting input pannel
    public Item output;
    public int outputAmount = 1;

    Unity Message | 0 references
    private void OnValidate()
    {
        // force the outputAmount always greater than 0
        outputAmount = Mathf.Max(outputAmount, 1);
    }
}
```

The Recipe is a Scriptable Object, adding new recipes to the game project is a straightforward process, and they **can be immediately used**.

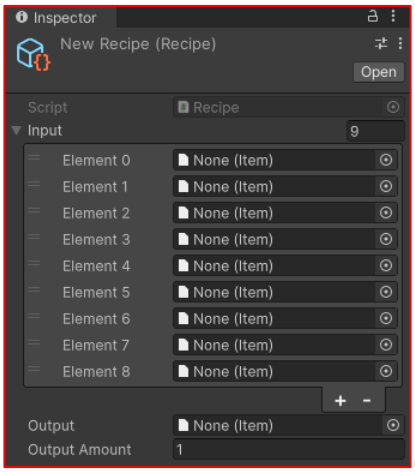
The upcoming Crafting algorithm for the Recipe **doesn't require a fixed 3x3 crafting table**. It's **adaptable to any n x n** configuration.

Additionally, **this package was initially made to have difficult to make a mistake**. For example, under the OnValidate(), the outputAmount cannot be accidentally set to less than 1, automatically resetting it to 1, while allowing values higher than 1.

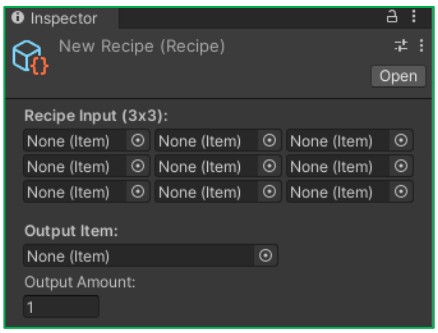
- **RecipeEditor**

```
[CustomEditor(typeof(Recipe))]
Unity Script | 0 references
public class RecipeEditor : Editor
{
    0 references
    public override void OnInspectorGUI()
    {
        Recipe recipe = (Recipe)target; // cast the target to the Recipe type

        EditorGUI.BeginChangeCheck();

        int gridSize = Mathf.CeilToInt(Mathf.Sqrt(recipe.input.Length));

        EditorGUILayout.LabelField("Recipe Input (" + gridSize + "x" + gridSize + "):", EditorStyles.boldLabel);
        for (int i = 0; i < gridSize; i++) // display the nxn grid of input items
        {
            EditorGUILayout.BeginHorizontal();
            for (int j = 0; j < gridSize; j++)
            {
                int index = i * gridSize + j;
                EditorGUILayout.PropertyField(serializedObject.FindProperty("input").GetArrayElementAtIndex(index), GUIContent.none, GUILayout.Width(100));
            }
            EditorGUILayout.EndHorizontal();
        }

        GUILayout.Space(10);

        // Display the output item field
        EditorGUILayout.LabelField("Output Item:", EditorStyles.boldLabel);
        EditorGUILayout.PropertyField(serializedObject.FindProperty("output"), GUIContent.none, GUILayout.Width(200));

        EditorGUILayout.PrefixLabel("Output Amount:", EditorStyles.boldLabel);
        EditorGUILayout.PropertyField(serializedObject.FindProperty("outputAmount"), GUIContent.none, GUILayout.Width(60), GUILayout.ExpandWidth(false));

        if (EditorGUI.EndChangeCheck())
            serializedObject.ApplyModifiedProperties(); // apply change if change occurs on the end
    }
}
```

The **RecipeEditor** is **designed to simplify the setup of crafting recipes for newcomers**. It **provides an intuitive interface** to **easily configure** **input item slot panels** and the **output slot**, including specifying the desired **output amount**.

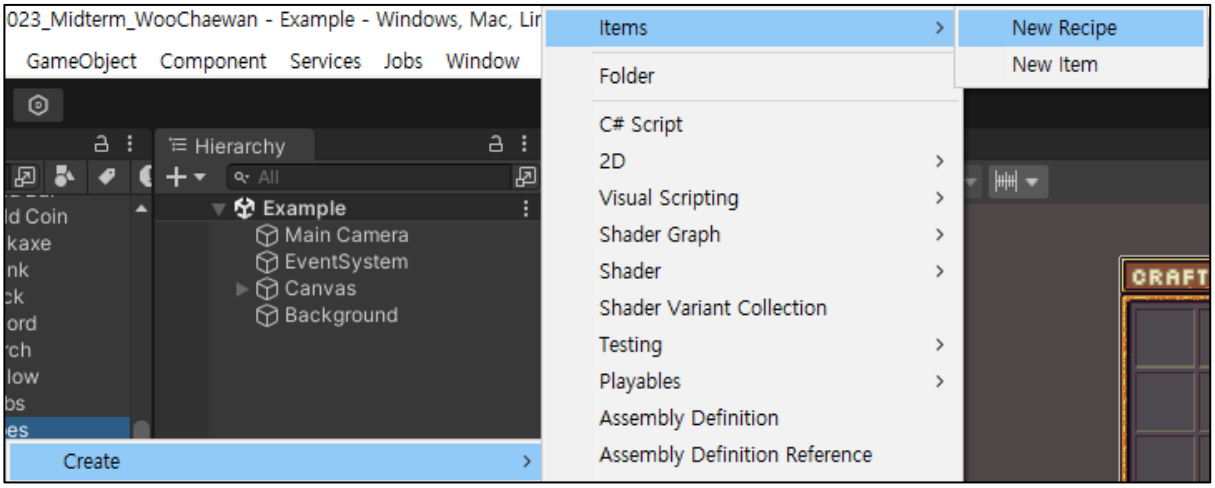


(Before)

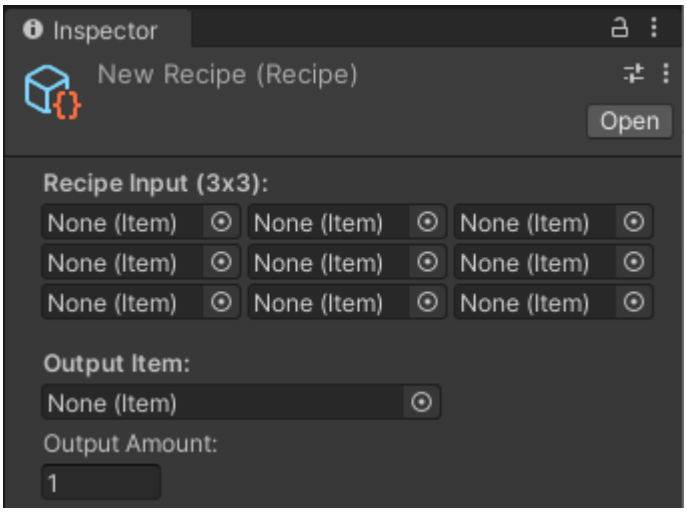


(After)

- **How to make Recipe (Scriptable Object)?**

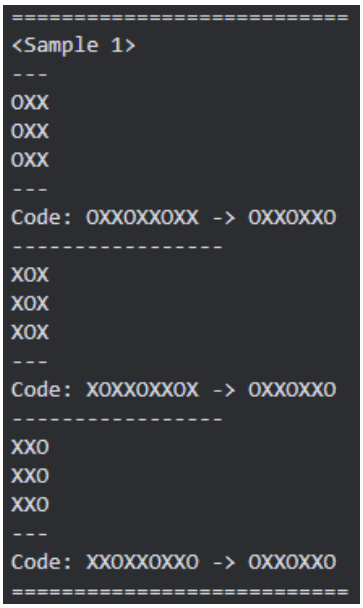


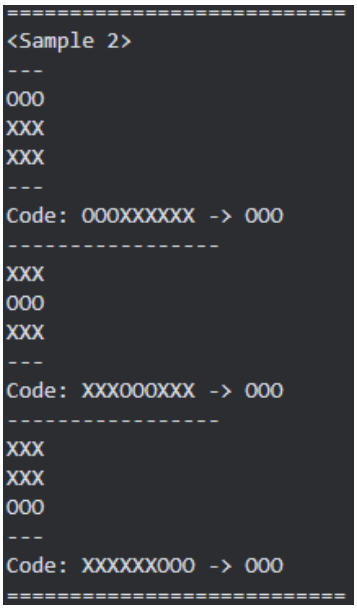
1. Create -> Items -> New Recipe

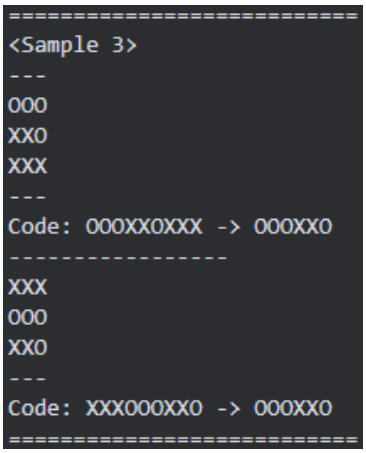


2. Easily assign a Scriptable Object by dragging and dropping it to either the Recipe Input or Output, and then specify the desired Output Amount.

- # Crafting Algorithm







The crafting algorithm is straightforward:

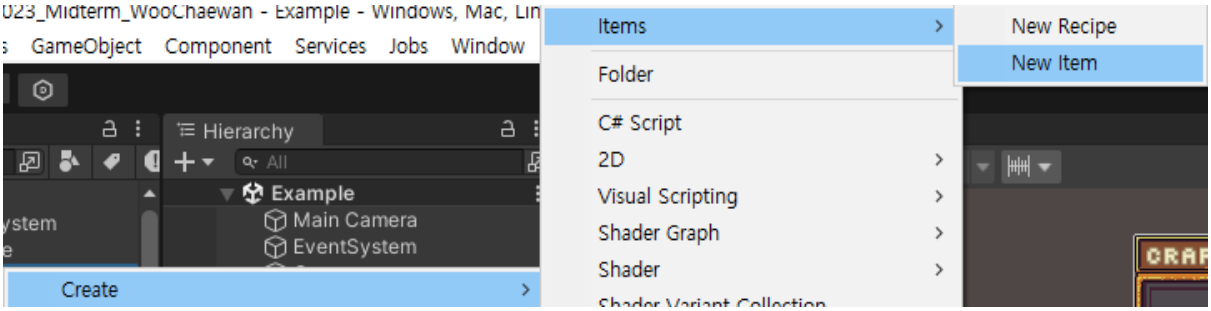
1. Combine all “item types” from the craft input panels into a single string.
2. Remove the outer “Empty slot item type” chars from the created string.
3. You'll notice that they all align correctly, even if placed in different locations, as long as their shapes match.

- # Item Script

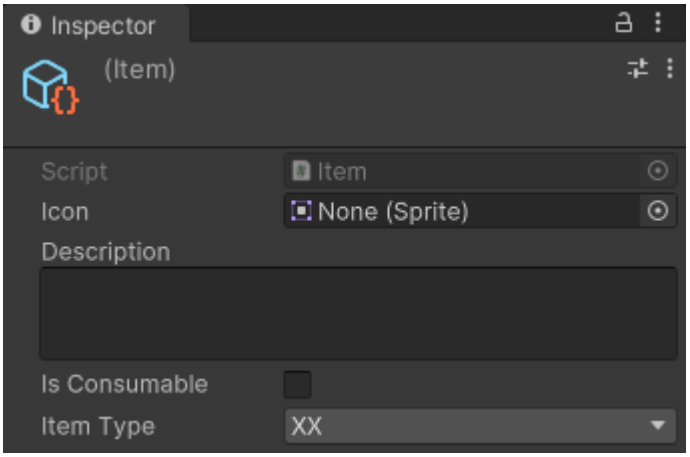
```
public enum eItemType   /* !WARNING! - When generating Recipe Code, it exclude some of the X chars. So, Don't use any X char for Itemtypes!!! - !WARNING! */
{
    XX, // empty
    B1, C1, C2, C3,
    C4, C5, C6, F1,
    G1, G2, G3, O1,
    P1, P2, S1, S2,
    T1, Y1
}
//Attribute which allows right click->Create
[CreateAssetMenu(fileName = "New Item", menuName = "Items/New Item")]
Unity Script | 19 references
public class Item : ScriptableObject //Extending SO allows us to have an object which exists in the project, not in the scene
{
    public Sprite icon;
    [TextArea]
    public string description = "";
    public bool isConsumable = false;
    public eItemType itemType;

    // public void Use() ...
}
```

The eItemType enum represents the "item types" mentioned above, and it **must be assigned** to each item ScriptableObject; **additional enums are required as more item Scriptable Objects are created.**

- # How to make Item (Scriptable Object)?



1. Create -> Items -> New Item



- Easily assign an item sprite to the icon, provide a brief item description, and configure the appropriate enum eItemType for the item type.

- # Crafting Script

```
public class Crafting : MonoBehaviour
{
    [SerializeField]
    List<Recipe> recipeList = new List<Recipe>();
    [SerializeField]
    List<string> recipeCode = new List<string>();

    [SerializeField]
    GameObject craftingInputPanel;
    [SerializeField]
    CraftingOutputSlot craftingOutputSlot;

    GenerateInputCodeFromRecipes before start

    Code Handler

    Craft System
}
```

The Crafting Script is highly efficient and optimized. [GenerateInputCodeFromRecipes before start], and [Craft System] **sharing common** [Code Handler] **functions**.

[GenerateInputCodeFromRecipes before start]

```
#region GenerateInputCodeFromRecipes before start
Unity Message | 0 references
private void OnValidate()   // calls whenever the asset is modified in the Unity Editor.
{
    GenerateInputCodeFromRecipes();
}
1 reference
void GenerateInputCodeFromRecipes() // add all Recipe input patterns to the recipeCode List
{
    recipeCode.Clear();
    for (int i=0; i< recipeList.Count; i++)
    {
        if (recipeList[i] == null) // error notifier
        {
            Debug.LogError("recipeList["+ i +"]" + "was null");
            recipeList.RemoveAt(i);
            return;
        }

        Item[] input = recipeList[i].input;
        string inputCode = GenerateInputCode(input);
        recipeCode.Add(inputCode);
    }
}
#endregion
```

Uses GenerateInputCode function from [Code Handler]

[**Code Handler**]

```
#region Code Handler
2 references
string GenerateInputCode(Item[] input)
{
    string inputCode = "";
    foreach (Item item in input)
    {
        inputCode += (item != null) ?
            item.itemType.ToString() : ((eItemType)0).ToString();
    }
    return ModifyInputCode(inputCode);
}

1 reference
string ModifyInputCode(string inputCode)
{
    int firstCharIndex = 0;
    int lastCharIndex = inputCode.Length - 1;

    while (firstCharIndex < inputCode.Length && inputCode[firstCharIndex] == 'X') // front
        firstCharIndex++;
    while (lastCharIndex >= 0 && inputCode[lastCharIndex] == 'X') // back
        lastCharIndex--;

    if (firstCharIndex < lastCharIndex) // combine
        return inputCode.Substring(firstCharIndex, lastCharIndex - firstCharIndex + 1);

    return "";
}
#endregion
```

These 2 functions are the "**crafting algorithm**" that was explained previously.

[**Craft System**]

```
#region Craft System
1 reference
public void InteractInputPanel()    // function calls from the CraftingInputSlot.cs only if any change on InputPanel
{
    craftingOutputSlot.DestroyCurrentItem();    // to reset output item

    List<ItemSlot> craftInputList =
        new List<ItemSlot>(craftingInputPanel.transform.GetComponentsInChildren<ItemSlot>());

    Item[] input = new Item[craftInputList.Count];
    for (int i = 0; i < craftInputList.Count; i++) // stores the 'Item' objects from the craft input slots
    {
        input[i] = craftInputList[i].GetItem();
    }

    string inputCode = GenerateInputCode(input);
    if (inputCode == "") return;

    Debug.Log(inputCode);

    int foundRecipeIndex = recipeCode.FindIndex(code => code == inputCode);
    if (foundRecipeIndex == -1) return; // -1 means not found

    CreateOutputItem(foundRecipeIndex, craftInputList);
}
```

Uses GenerateInputCode function from [**Code Handler**]

. . . Any missing scripts, as well as those already covered in the documentation, will receive more comprehensive explanations in the video.