**Kristian Recio**

**Michelle Garcia**

**Anjanelle Hernandez**

**Adventure Game Tutorial:**

This tutorial only focuses on the features we added to the Unity Adventure Game since the tutorials for that game are already online.

**To Add a Dialogue with the Security Guard:**

· **The Pictures**

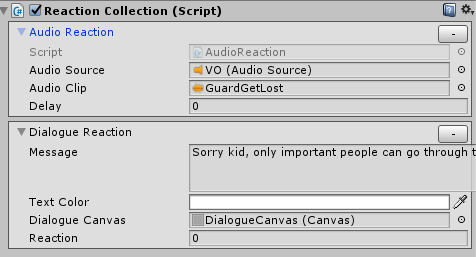
The security guard’s portrait and the dialogue screen background are the only pictures involved in creating the dialogue. For the security guard’s portrait, I opened the Security Room scene and moved the scene view to face the security guard. I then took a screenshot and cropped the photo. For the dialogue screen background, I created an image in GIMP with the size of 744x419. I then chose a color that was similar to the colors of the environment and moved the transparency to 50%. This allows for the background to be seen while the dialogue is being showed.

· **The Canvas**

For the dialogue, I created a canvas and called it DialogueCanvas. I then created an empty object called Background as a child of it. I set the background image I created as the image for Background and as a child of Background created a MessageText Text game object. I then created 5 buttons as children of DialogueCanvas and named then Option1, Option2, Option3, Option4, and Option5. I also created a 6th button and named it Close. For each of them I set the Source Image as UISprite and the color with the hex color of #33455AFF. For the close button, I added an “Outline (Script)” by finding it in the Add Component option. The Effect Color is set to black, the Effect Distance is X: 1 and Y: -1, and the Use Graphic Alpha box is checked. The reactions of the buttons, I put in an empty Game Object I called Reactions and as children I set other empty Game Objects called Option1Reactions, Option2Reactions, etc. To create a Reaction just create an empty game object and add a Reaction component to it. Also Option4 and Option5 are deactivated and then DialogueCanvas is deactivated.

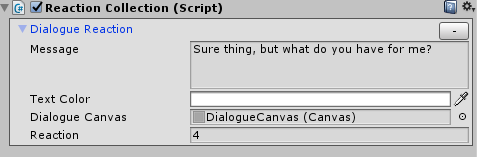
· **Option 1 Reactions, Interactable, and OnClick()**

**Default Reaction:**

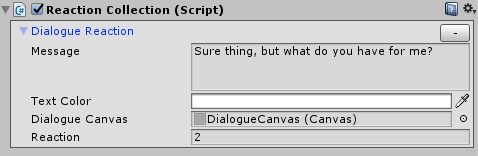


“Sorry kid, only important people can go through the gate.”

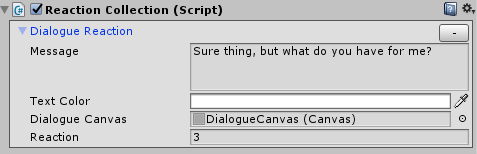
**GlassesCoffeeFishReaction:**



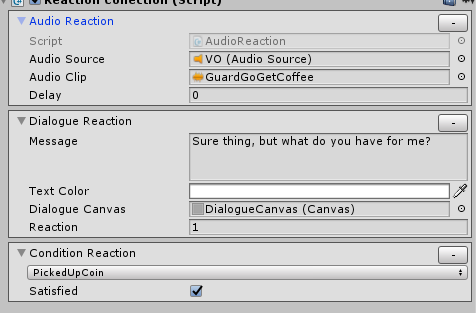
**GlassesAndCoffeeReaction:**

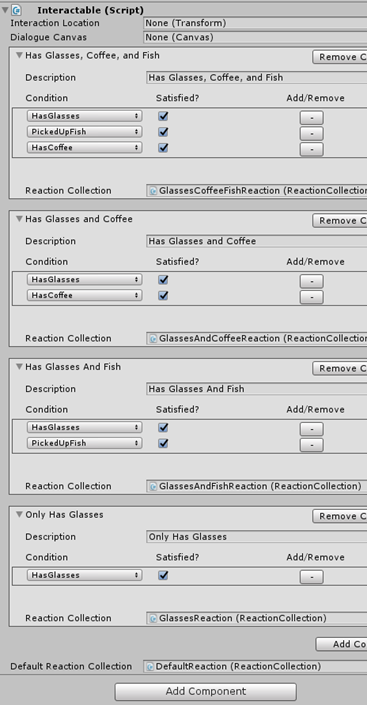


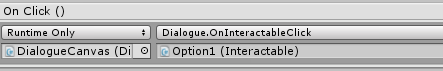
**GlassesAndFishReaction:**



**GlassesReaction:**

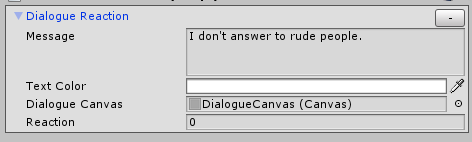


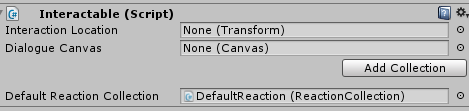




· **Option2 Reactions, Interactable, OnClick()**

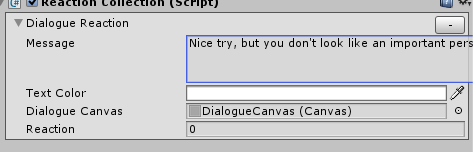
**Default Reaction:**





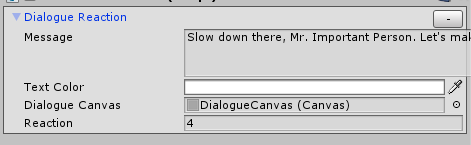
· **Option3 Reactions, Interactable, and OnClick()**

**Default Reaction:**



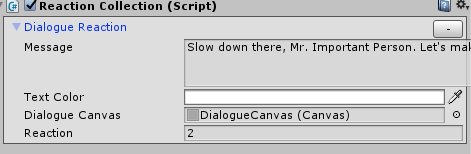
“Nice try, but you don't look like an important person.”

**GlassesCoffeeFishReaction:**



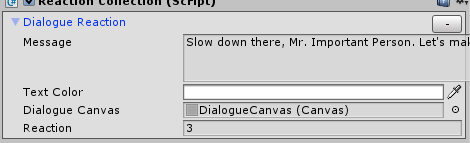
“Slow down there, Mr. Important Person. Let's make a little exchange..”

**GlassesAndCoffeeReaction:**



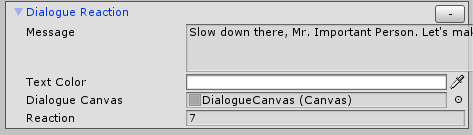
“Slow down there, Mr. Important Person. Let's make a little exchange..”

**GlassesAndFishReaction:**

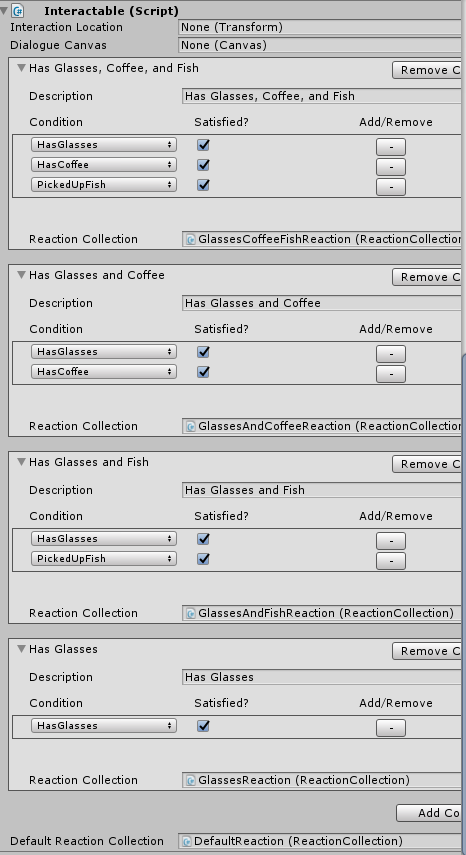


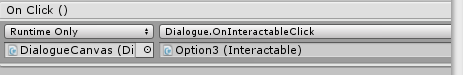
“Slow down there, Mr. Important Person. Let's make a little exchange..”

**GlassesReaction:**



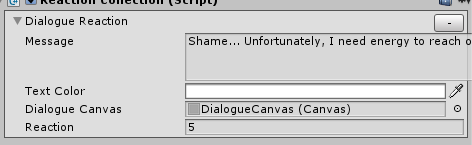
“Slow down there, Mr. Important Person. Let's make a little exchange..”





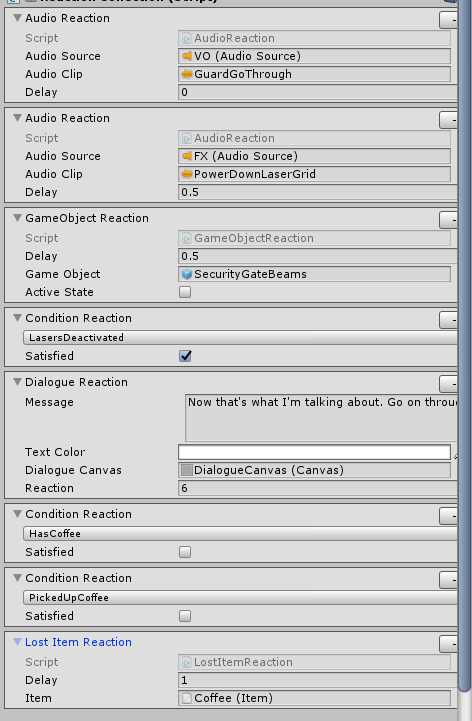
· **Option 4 Reactions, Interactable, and OnClick()**

**DefaultReaction:**



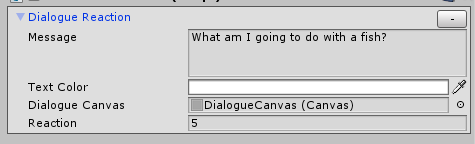
“Shame... Unfortunately, I need energy to reach over and push the button to open the gate...”

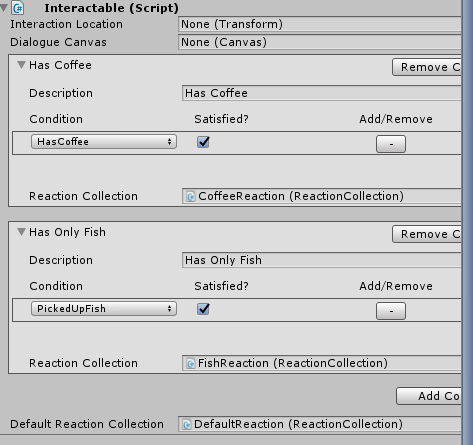
**CoffeeReaction:**

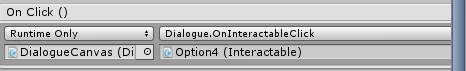


“Now that's what I'm talking about. Go on through.”

**FishReaction:**

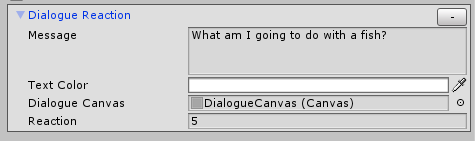


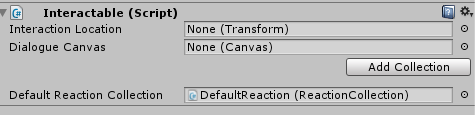


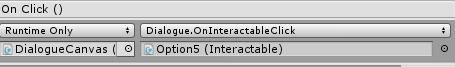


· **Option 5 Reactions, Interactable, and OnClick()**

**DefaultReaction:**

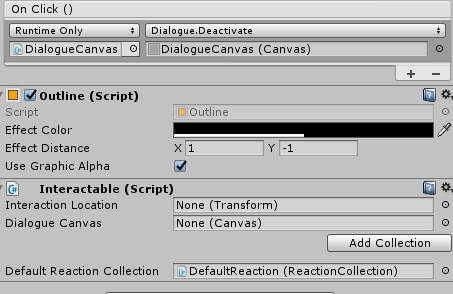






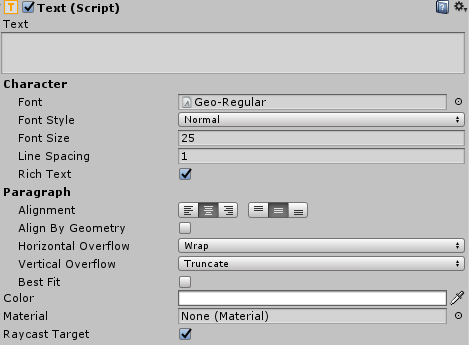
· **Close Components and Reaction**

**Default Reaction:** Has nothing



· **Text Components on Text Game Objects in DialogueCanvas**

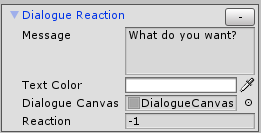
The were all set to this:



Except, the Close button that had “X” in Text and the font size as 14.

· **The GuardInteractable**

I removed whatever was in the Interactable Script except the Interaction Location. The Dialogue Canvas option is set the DialogueCanvas I created and the Default Reaction is set to the DefaultReaction that is a child of GuardInteractable. For the DefaultReaction, I removed all the reactions except the Audio Reaction and added a Dialogue Reaction shown below:



**Scripts:**

**DialogueReactionEditor:**

using UnityEditor;

using UnityEngine;

[CustomEditor(typeof(DialogueReaction))]

public class DialogueReactionEditor : ReactionEditor

{

private SerializedProperty messageProperty;

private SerializedProperty textColorProperty;

private SerializedProperty dialogueCanvasProperty;

private SerializedProperty reactionProperty;

private const float messageGUILines = 3f;

private const float areaWidthOffset = 19f;

private const string dialogueReactionPropMessageName = "message";

private const string dialogueReactionPropTextColorName = "textColor";

private const string dialogueReactionPropDialogueCanvasName = "dialogueCanvas";

private const string dialogueReactionPropReactionName = "reaction";

protected override void Init()

{

messageProperty = serializedObject.FindProperty(dialogueReactionPropMessageName);

textColorProperty = serializedObject.FindProperty(dialogueReactionPropTextColorName);

dialogueCanvasProperty = serializedObject.FindProperty(dialogueReactionPropDialogueCanvasName);

reactionProperty = serializedObject.FindProperty(dialogueReactionPropReactionName);

}

protected override void DrawReaction()

{

EditorGUILayout.BeginHorizontal();

EditorGUILayout.LabelField("Message", GUILayout.Width(EditorGUIUtility.labelWidth - areaWidthOffset));

messageProperty.stringValue = EditorGUILayout.TextArea(messageProperty.stringValue, GUILayout.Height(EditorGUIUtility.singleLineHeight \* messageGUILines));

EditorGUILayout.EndHorizontal();

EditorGUILayout.PropertyField(textColorProperty);

EditorGUILayout.ObjectField(dialogueCanvasProperty);

EditorGUILayout.PropertyField(reactionProperty);

}

protected override string GetFoldoutLabel()

{

return "Dialogue Reaction";

}

}

**InteractableEditor:**

using UnityEngine;

using UnityEditor;

[CustomEditor(typeof(Interactable))]

public class InteractableEditor : EditorWithSubEditors<ConditionCollectionEditor, ConditionCollection>

{

private Interactable interactable;

private SerializedProperty interactionLocationProperty;

private SerializedProperty dialogueCanvasProperty;

private SerializedProperty collectionsProperty;

private SerializedProperty defaultReactionCollectionProperty;

private const float collectionButtonWidth = 125f;

private const string interactablePropInteractionLocationName = "interactionLocation";

private const string interactablePropDialogueCanvas = "dialogueCanvas";

private const string interactablePropConditionCollectionsName = "conditionCollections";

private const string interactablePropDefaultReactionCollectionName = "defaultReactionCollection";

private void OnEnable ()

{

interactable = (Interactable)target;

collectionsProperty = serializedObject.FindProperty(interactablePropConditionCollectionsName);

interactionLocationProperty = serializedObject.FindProperty(interactablePropInteractionLocationName);

dialogueCanvasProperty = serializedObject.FindProperty(interactablePropDialogueCanvas);

defaultReactionCollectionProperty = serializedObject.FindProperty(interactablePropDefaultReactionCollectionName);

CheckAndCreateSubEditors(interactable.conditionCollections);

}

private void OnDisable ()

{

CleanupEditors ();

}

protected override void SubEditorSetup(ConditionCollectionEditor editor)

{

editor.collectionsProperty = collectionsProperty;

}

public override void OnInspectorGUI ()

{

serializedObject.Update ();

CheckAndCreateSubEditors(interactable.conditionCollections);

EditorGUILayout.PropertyField (interactionLocationProperty);

EditorGUILayout.PropertyField(dialogueCanvasProperty);

for (int i = 0; i < subEditors.Length; i++)

{

subEditors[i].OnInspectorGUI ();

EditorGUILayout.Space ();

}

EditorGUILayout.BeginHorizontal();

GUILayout.FlexibleSpace ();

if (GUILayout.Button("Add Collection", GUILayout.Width(collectionButtonWidth)))

{

ConditionCollection newCollection = ConditionCollectionEditor.CreateConditionCollection ();

collectionsProperty.AddToObjectArray (newCollection);

}

EditorGUILayout.EndHorizontal ();

EditorGUILayout.Space ();

EditorGUILayout.PropertyField (defaultReactionCollectionProperty);

serializedObject.ApplyModifiedProperties ();

}

}

**Dialogue:**

using UnityEngine;

public class Dialogue : MonoBehaviour {

private Interactable currentInteractable;

public void OnInteractableClick(Interactable interactable)

{

currentInteractable = interactable;

currentInteractable.Interact(currentInteractable);

}

public void Deactivate(Canvas canvas)

{

canvas.gameObject.SetActive(false);

}

}

**DialogueManager:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class DialogueManager : MonoBehaviour {

public Text text;

public void DisplayMessage(string message, Color color)

{

text.text = message;

text.color = color;

}

}

**Interactable:**

using UnityEngine;

using UnityEngine.UI;

public class Interactable : MonoBehaviour

{

public Transform interactionLocation;

public Canvas dialogueCanvas;

public ConditionCollection[] conditionCollections = new ConditionCollection[0];

public ReactionCollection defaultReactionCollection;

public void Interact (Interactable interactable)

{

if (interactable.gameObject.name == "GuardInteractable")

{

dialogueCanvas.gameObject.SetActive(true);

Button[] btn = GameObject.Find("DialogueCanvas").GetComponentsInChildren<Button>(true);

btn[0].gameObject.SetActive(true);

btn[1].gameObject.SetActive(true);

btn[2].gameObject.SetActive(true);

btn[3].gameObject.SetActive(false);

btn[4].gameObject.SetActive(false);

btn[0].gameObject.GetComponentInChildren<Text>().text = "I'd like to go through the gate, please.";

btn[1].gameObject.GetComponentInChildren<Text>().text = "Open the gate!";

btn[2].gameObject.GetComponentInChildren<Text>().text = "I'm an important person, let me through.";

}

for (int i = 0; i < conditionCollections.Length; i++)

{

if (conditionCollections[i].CheckAndReact ())

return;

}

defaultReactionCollection.React ();

}

}

**DialogueReaction:**

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class DialogueReaction : Reaction

{

public string message;

public Color textColor = Color.white;

public float delay;

public Canvas dialogueCanvas;

public int reaction;

private Button[] button;

private DialogueManager dialogueManager;

protected override void SpecificInit()

{

dialogueManager = FindObjectOfType<DialogueManager>();

button = dialogueCanvas.gameObject.GetComponentsInChildren<Button>(true);

}

protected override void ImmediateReaction()

{

dialogueManager.DisplayMessage(message, textColor);

switch (reaction)

{

case 0:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

break;

case 1:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(true);

button[3].gameObject.GetComponentInChildren<Text>().text = "Nothing...";

break;

case 2:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(true);

button[3].gameObject.GetComponentInChildren<Text>().text = "[Give Coffee]";

break;

case 3:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(true);

button[3].gameObject.GetComponentInChildren<Text>().text = "[Give Fish]";

break;

case 4:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(true);

button[4].gameObject.SetActive(true);

button[3].gameObject.GetComponentInChildren<Text>().text = "[Give Coffee]";

button[4].gameObject.GetComponentInChildren<Text>().text = "[Give Fish]";

break;

case 5:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(false);

button[4].gameObject.SetActive(false);

break;

case 6:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(false);

button[4].gameObject.SetActive(false);

break;

case 7:

button[0].gameObject.SetActive(false);

button[1].gameObject.SetActive(false);

button[2].gameObject.SetActive(false);

button[3].gameObject.SetActive(true);

button[3].gameObject.GetComponentInChildren<Text>().text = "I have nothing to exchange...";

break;

default:

break;

}

}

public IEnumerator wait()

{

yield return new WaitForSeconds(0.6f);

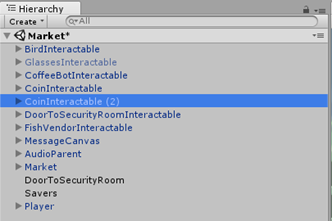
}

}

**Adding a coin to the game[1]**

1. In the project folder, go to the scenes folder and open up the market scene.

2. In the hierarchy tab, right click “CoinInteractable” and duplicate it.

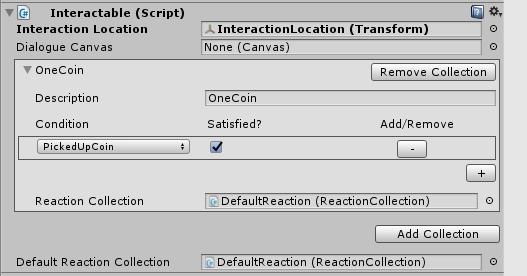


**Both CoinInteractable and CoinInteractable (1)**

3. Click both CoinInteractable and CoinInteractable (1), and go to the inspector tab.

4. Under the tab Interactable (Script), click “Add Collection”.

5. In the description box, type OneCoin, and change the condition to PickedUpCoin.

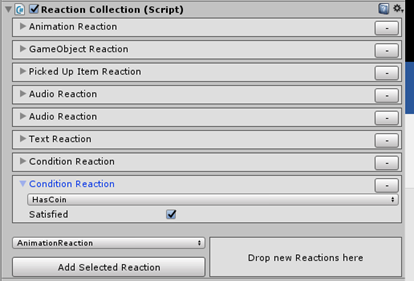


6. Under the CoinInteractable and CoinInteractable (1) tabs, click DefaultReaction.

7. Under the Reaction Collection (Script), click on the drop-down list above the Add Selected Reaction button and select ConditionReaction.

8. Click the Add Selected Reaction button.

9. In the new Condition Reaction box, choose “HasCoin”.



**CoinInteractable (1)**

10. Make sure you are in Scene view, and then click on CoinInteractable (1).

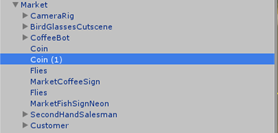
11. In the scene, move the CoinInteractable (1) from the small slot in the soda machine to the small slot in the other soda machine.



**Adding in the coin**

12. In the hierarchy tab, click on the drop-down list on the market.

13. Right click on the Coin and duplicate it.

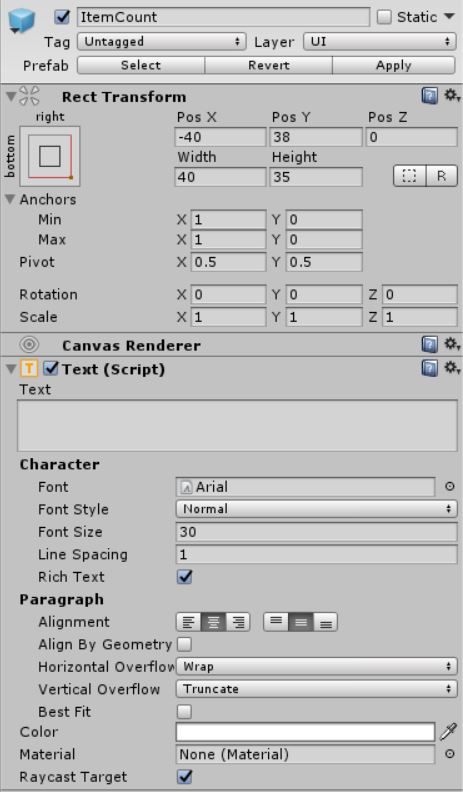


14. Move the second coin and move it to the same spot that the CoinInteractable (1) was moved to earlier.

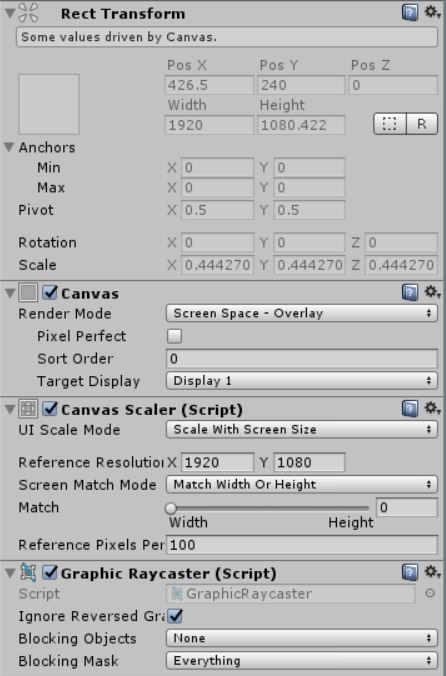
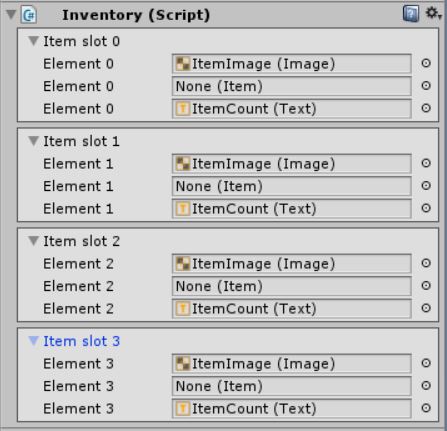


**Inventory Counter:**

All of this work will be done in the Persistent scene. To create the item counter on the lower right part of an item's image we must first delete the ItemSlot prefab in the Prefab folder under the Project tab and all other ItemSlots in the Persistent scene except ItemSlot0. With ItemSlot0 selected, create a new Text UI gameobject child and name it ItemCount. It is important that this gameobject is at the very end in the list of children. Make sure that the Text field within the Text UI's Text component is empty, that the font size is set to 30, the color is set to white, middle and center are selected for alignment, and that it is selected as a raycast target. The inspector tab will be shown below. After doing all this, create a new prefab in the Prefabs folder previously mention with ItemSlot0 and name it ItemSlot. Duplicate ItemSlot0 three more times, each time renaming the new gameobject with a different number at the end so that you end with four ItemSlot gameobjects numbered 0-3.



At this point, the scripts InventoryEditor and Inventory must both be changed to reflect the scripts shown below in order to continue. After modifying both of these scripts, select the PersistentCanvas gameobject under the Hierarchy tab. Under the Inventory script component, each item slot drop down should include an extra element. This is where each ItemCount gameobject created before must be dragged to, relevant to their assigned ItemSlot number. If the drop downs have been reset, the first element in each must be linked to an ItemSlot's ItemImage, the second element can be left empty, and the third element must be linked to an ItemSlot's ItemCount. Again, the inspector tab after completion will be shown below.

**Scripts - InventoryEditor:**

using UnityEngine;

using UnityEditor;

[CustomEditor(typeof(Inventory))]

public class InventoryEditor : Editor

{

private bool[] showItemSlots = new bool[Inventory.numItemSlots];

private SerializedProperty itemImagesProperty;

private SerializedProperty itemsProperty;

private SerializedProperty itemCountProperty;

private const string inventoryPropItemImagesName = "itemImages";

private const string inventoryPropItemsName = "items";

private const string inventoryPropItemCountName = "itemCount";

private void OnEnable()

{

itemImagesProperty = serializedObject.FindProperty(inventoryPropItemImagesName);

itemsProperty = serializedObject.FindProperty(inventoryPropItemsName);

itemCountProperty = serializedObject.FindProperty(inventoryPropItemCountName);

}

public override void OnInspectorGUI()

{

serializedObject.Update();

for (int i = 0; i < Inventory.numItemSlots; i++)

{

ItemSlotGUI(i);

}

serializedObject.ApplyModifiedProperties();

}

private void ItemSlotGUI(int index)

{

EditorGUILayout.BeginVertical(GUI.skin.box);

EditorGUI.indentLevel++;

showItemSlots[index] = EditorGUILayout.Foldout(showItemSlots[index], "Item slot " + index);

if (showItemSlots[index])

{

EditorGUILayout.PropertyField(itemImagesProperty.GetArrayElementAtIndex(index));

EditorGUILayout.PropertyField(itemsProperty.GetArrayElementAtIndex(index));

EditorGUILayout.PropertyField(itemCountProperty.GetArrayElementAtIndex(index));

}

EditorGUI.indentLevel--;

EditorGUILayout.EndVertical();

}

}

**Scripts - Inventory:**

using UnityEngine;

using UnityEngine.UI;

public class Inventory : MonoBehaviour

{

public Image[] itemImages = new Image[numItemSlots];

public Item[] items = new Item[numItemSlots];

public Text[] itemCount = new Text[numItemSlots];

private int iCount = 0;

public const int numItemSlots = 4;

public void AddItem(Item itemToAdd)

{

for (int i = 0; i < items.Length; i++)

{

if (items[i] == null)

{

items[i] = itemToAdd;

itemCount[i].text = "1";

iCount = 1;

itemImages[i].sprite = itemToAdd.sprite;

itemImages[i].enabled = true;

return;

}

else if ((items[i] != null) && (items[i] == itemToAdd))

{

iCount = iCount + 1;

itemCount[i].text = iCount.ToString();

}

}

}

public void RemoveItem(Item itemToRemove)

{

for (int i = 0; i < items.Length; i++)

{

if (items[i] == itemToRemove)

{

items[i] = null;

itemCount[i].text = "";

itemImages[i].sprite = null;

itemImages[i].enabled = false;

return;

}

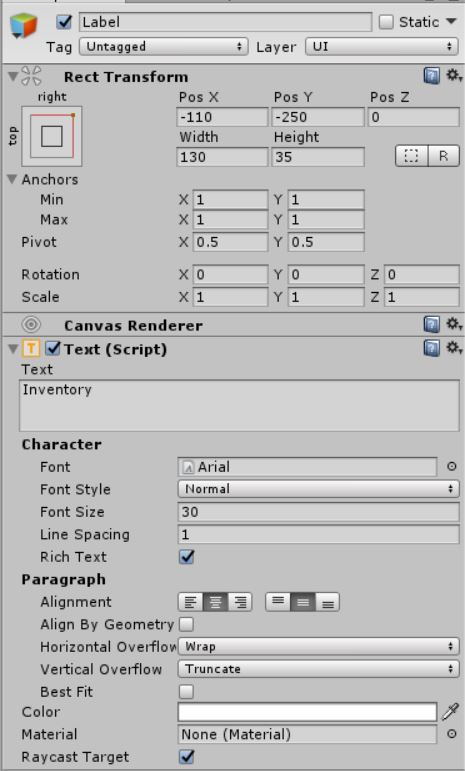
}

}

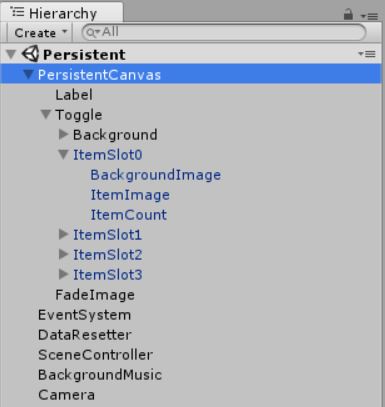
}

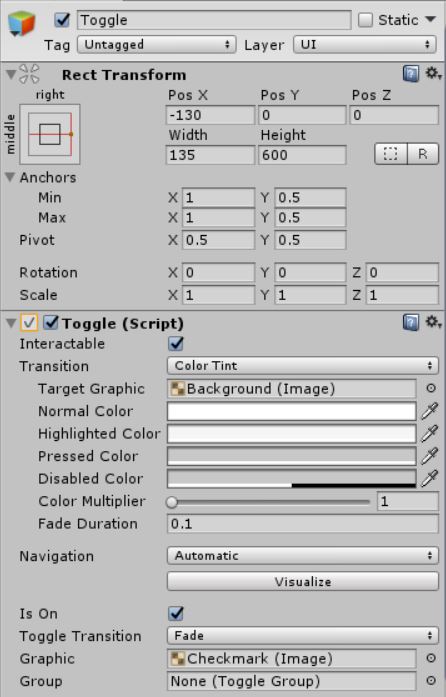
**Toggleable Inventory:**

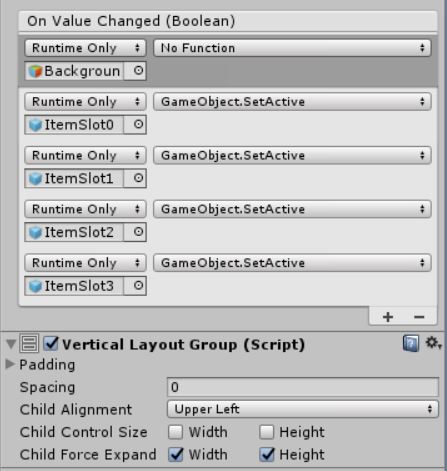
All of this work will also be done in the Persistent scene. To make the inventory toggleable, we must first create a Text UI child and a Toggle UI child both under the PersistentCanvas gameobject. Make sure that the Text UI child is placed above the Toggle UI child in their order under the PersistentCanvas gameobject. Rename the Text UI child Label and set the text field of the text component to Inventory. Set the font size to 30, the alignment to middle and center, and make sure it is selected as a raycast target. Set the text color to white.



For the Toggle gameobject, we must move everything that is under the Inventory gameobject into the Toggle gameobject. That is, the Background, ItemSlot0, ItemSlot1, ItemSlot2, ItemSlot3, and FadeImage gameobjects must all be moved as children of the Toggle gameobject. After moving everything over, it is then safe to delete the Inventory Game object. A picture of the Hierarchy tab will be provided below. Once this is complete, select the Toggle gameobject and add four more rows to the On Value Changed portion of the Toggle script component so that there are now five rows. The first should be set to No Function in the drop down menu to the right while the rest will all be set to GameObject.SetActive as their function. Use the circle menu or drag and drop the Background and each ItemSlot gameobject in the hierarchy tab into each of the object fields respectively. Create a Vertical Layout Group component for the Toggle gameobject. After making sure that everything in Child Control Size is unselected and everything in Child Force Expand is selected under the Vertical Layout Group component, everything is complete.







[1] This is only a partial tutorial. It doesn’t work completely.