Tutorials

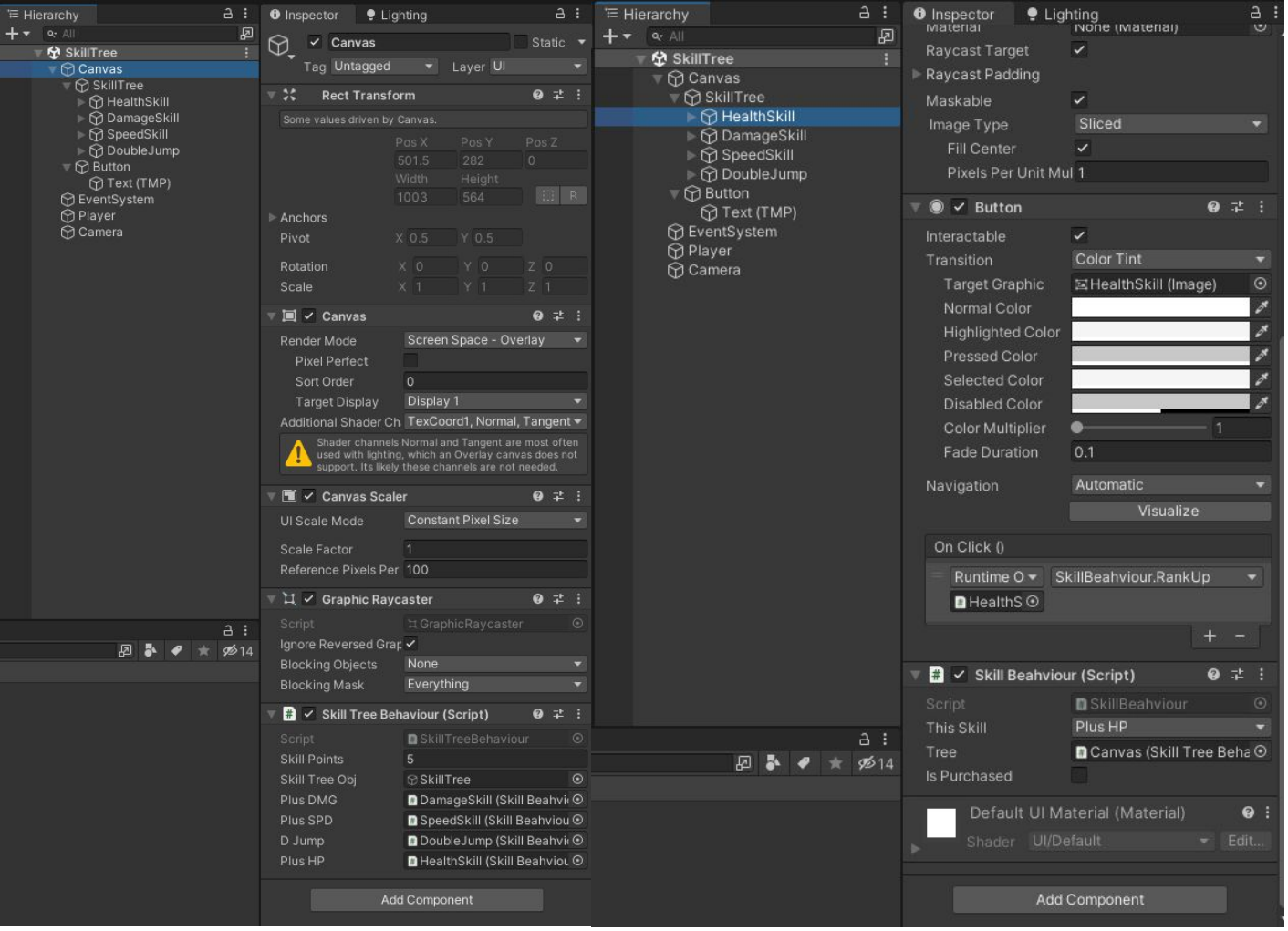
Skill Tree Behaviour

This script controls how the window for purchasing skill works, so in this case it simply turns on and off when the button is clicked or when, it also contains the players' skill points and the main object that holds the skill tree visual, this is how it turn the window on and off

```
private bool skillTreeBool = false;
public int skillPoints = 0;
public GameObject skillTreeObj;
public SkillBeahviour plusDMG, plusSPD, DJump, plusHP;
```

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Tab))
    {
        SkillTree();
    }
    if (skillTreeBool)
    {
        skillTreeObj.SetActive(true);
    }
    else { skillTreeObj.SetActive(false); }
}

public void SkillTree()
{
    skillTreeBool = !skillTreeBool;
}
```

Window activation code on the skill tree behaviour script


Hierarchy
SkillTree
Canvas
SkillTree
HealthSkill
DamageSkill
SpeedSkill
DoubleJump
Button
Text (TMP)
EventSystem
Player
Camera
Inspector
Lighting
Canvas
Static
Tag Untagged
Layer UI
Rect Transform
Some values driven by Canvas.
Pos X 501.5
Pos Y 282
Pos Z 0
Width 1003
Height 564
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas
Render Mode Screen Space - Overlay
Pixel Perfect
Sort Order 0
Target Display Display 1
Additional Shader Ch TexCoord1, Normal, Tangent
Shader channels Normal and Tangent are most often used with lighting, which an Overlay canvas does not support. Its likely these channels are not needed.
Canvas Scaler
UI Scale Mode Constant Pixel Size
Scale Factor 1
Reference Pixels Per 100
Graphic Raycaster
Script GraphicRaycaster
Ignore Reversed Grap
Blocking Objects None
Blocking Mask Everything
Skill Tree Behaviour (Script)
Script SkillTreeBehaviour
Skill Points 5
Skill Tree Obj SkillTree
Plus DMG DamageSkill (Skill Beahvi
Plus SPD SpeedSkill (Skill Beahviou
D Jump DoubleJump (Skill Beahvi
Plus HP HealthSkill (Skill Beahviou
Add Component
Hierarchy
SkillTree
Canvas
SkillTree
HealthSkill
DamageSkill
SpeedSkill
DoubleJump
Button
Text (TMP)
EventSystem
Player
Camera
Inspector
Lighting
Raycast Target
Raycast Padding
Maskable
Image Type Sliced
Fill Center
Pixels Per Unit Mul 1
Button
Interactable
Transition Color Tint
Target Graphic HealthSkill (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
Visualize
On Click ()
Runtime O
SkillBeahviour.RankUp
HealthS
Skill Beahviour (Script)
Script SkillBeahviour
This Skill Plus HP
Tree Canvas (Skill Tree Beha
Is Purchased
Default UI Material (Material)
Shader UI/Default
Edit...
Add Component

```
public enum Skills
{
    plusDMG, plusSPD, DJump, plusHP
}
```

The skill behaviour script has the main functionality, first whether or not the skill has been purchased, and a required skill that needs to be purchased before this current skill is available

```
public class SkillBeahviour : MonoBehaviour
{
    public Skills thisSkill;
    public SkillTreeBehaviour tree;
    public bool isPurchased = false;

    private SkillBeahviour requiredSkill;
    private TextMeshProUGUI textSkill;
```

In the start function we can set which skill need a prerequisite skill before they can be acquired

```
private void Start()
{
    textSkill = GetComponentInChildren<TextMeshProUGUI>();

    if (thisSkill.Equals(Skills.plusDMG)) { requiredSkill = tree.plusHP; }
    if (thisSkill.Equals(Skills.DJump)) { requiredSkill = tree.plusSPD; }
}
```

```
public void RankUp()
{
    if(tree.skillPoints > 0 && !isPurchased)
    {
        if (thisSkill.Equals(Skills.plusHP))
        {
            tree.skillPoints--;
            isPurchased = true;
            textSkill.text = "Purchased!";
            PlusHealth();
        }
        else if (thisSkill.Equals(Skills.plusSPD))
        {
            tree.skillPoints--;
            isPurchased = true;
            textSkill.text = "Purchased!";
            PlusMovement();
        }
        else if (requiredSkill.isPurchased)
        {
            tree.skillPoints++;
            isPurchased = true;
            textSkill.text = "Purchased!";
            if (thisSkill.Equals(Skills.plusDMG)) { PlusDamage(); }
            if (thisSkill.Equals(Skills.DJump)) { DoubleJump(); }
        }
    }
}
```

The rank up method is called when the player clicks the skill in the skill tree, first we check if there is enough skill points then we check which skill is being purchased and add call a function on the player that add the new upgrade to them.

```
private void PlusMovement()
{
    Debug.Log("Movement Upgraded");
}

private void PlusDamage()
{
    Debug.Log("Damage Upgraded");
}

private void PlusHealth()
{
    Debug.Log("Health Upgraded");
}

private void DoubleJump()
{
    Debug.Log("Jump Upgraded");
}
```

These are the methods that are called on the player as need, it can contain anything so this is just some boiler code

Inventory System

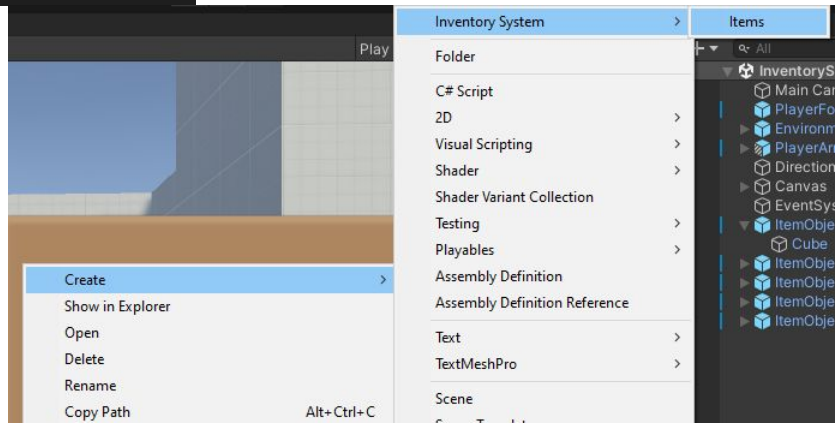
```
public class InventoryDisplay : MonoBehaviour
{

    public RectTransform backGround;

    public GameObject itemDisplay;

    private List<GameObject> currentObjects;
    private PlayerInventory pInventory;

public void AddNewItem(InventoryItem item )
{
    GameObject objInstance = Instantiate(itemDisplay);
    RectTransform instTransform = objInstance.GetComponent<RectTransform>();
    instTransform.SetParent(backGround);
    TextMeshProUGUI objectDisplay = objInstance.transform.Find("ItemName").GetComponent<TextMeshProUGUI>();
    UseableItem useableItem = objInstance.GetComponent<UseableItem>();
    useableItem.item = item;
    objectDisplay.text = item.itemName;

}
```

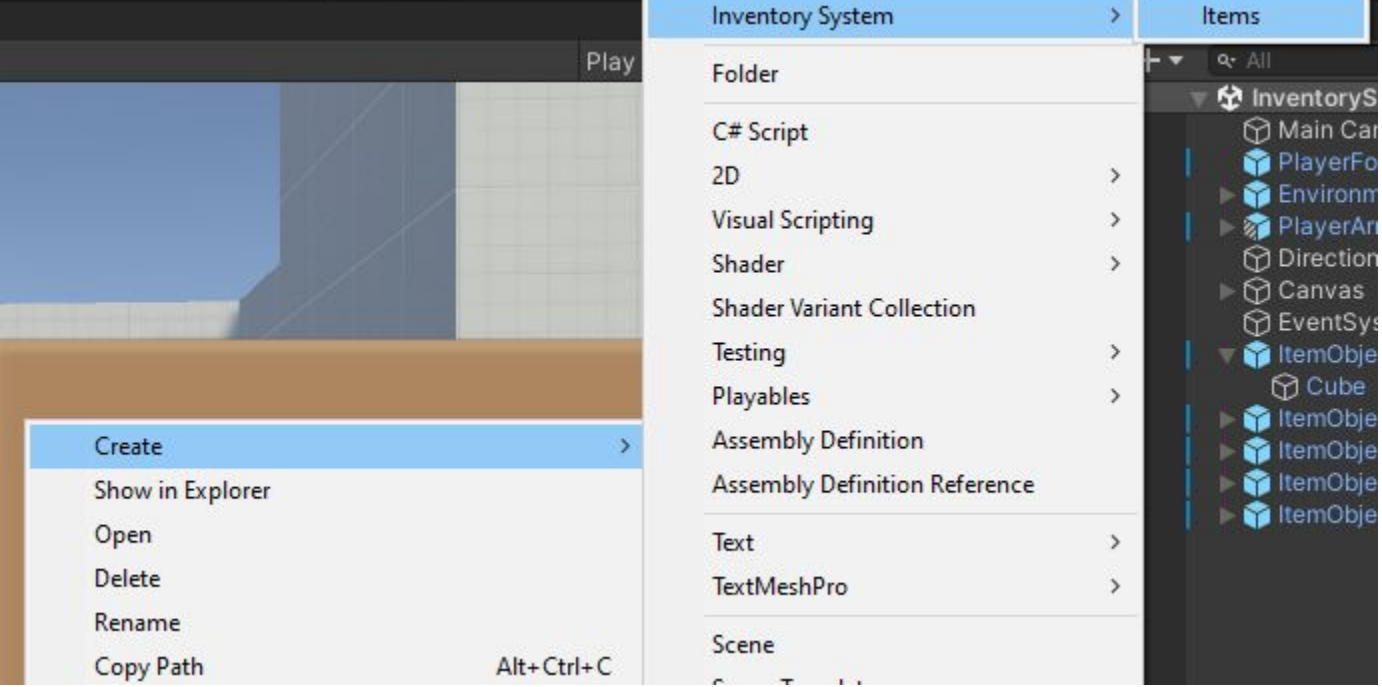
The inventory display script that sits on the canvas gameobject and adds new item to the players' UI to show them what items they own
The Add new item creates a new itemDisplay prefab and sets the value for it based on the inventory item class that has been passed to the function.

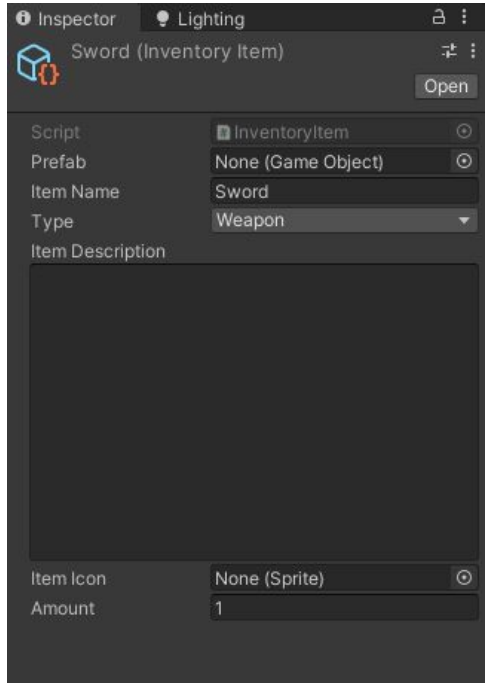
```
[CreateAssetMenu(fileName = "New Inventory Object", menuName = "Inventory System/Items")]
public class InventoryItem : ScriptableObject
{
    public GameObject prefab;
    public string itemName;
    public enum ItemType
    {
        Weapon,
        Potion,
        Coin,
    }
    public ItemType type;
    [TextArea(15,20)]
    public string itemDescription;
    public Sprite itemIcon;
    public int amount;
```

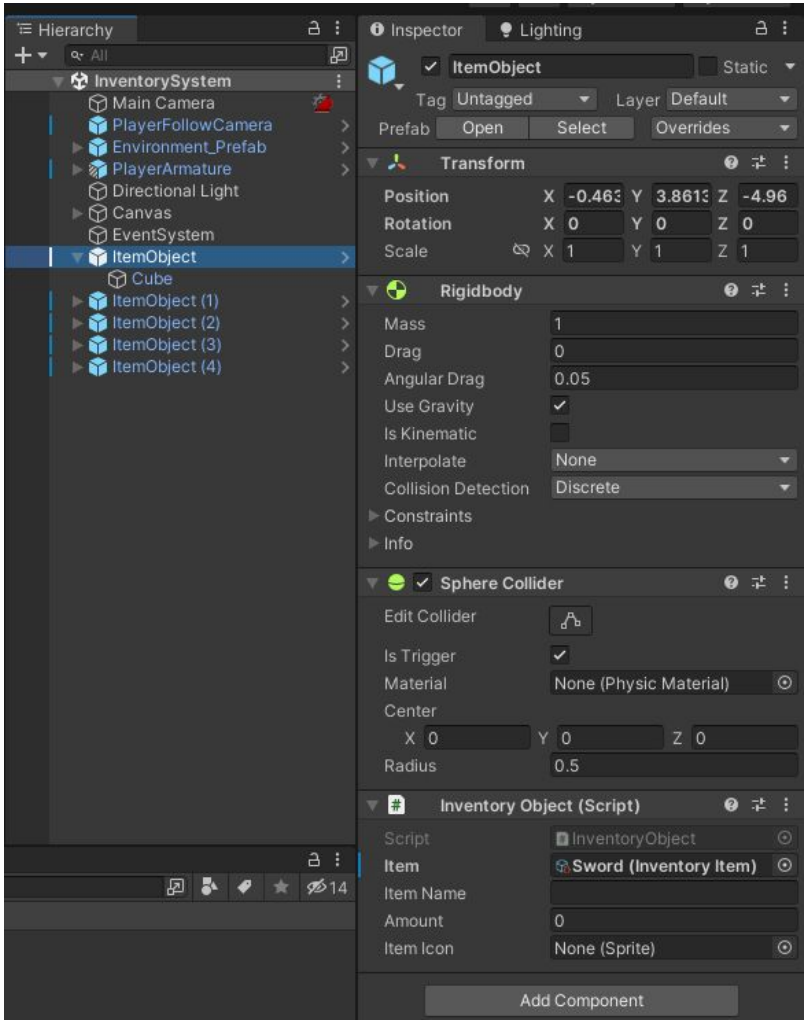
```
public void UseItem()
{
    switch (type)
    {
        case InventoryItem.ItemType.Weapon:
            Debug.Log("Weapon");
            break;
        case InventoryItem.ItemType.Potion:
            Debug.Log("Potion");
            break;
        case InventoryItem.ItemType.Coin:
            Debug.Log("Coin");
            break;

    }
}
```

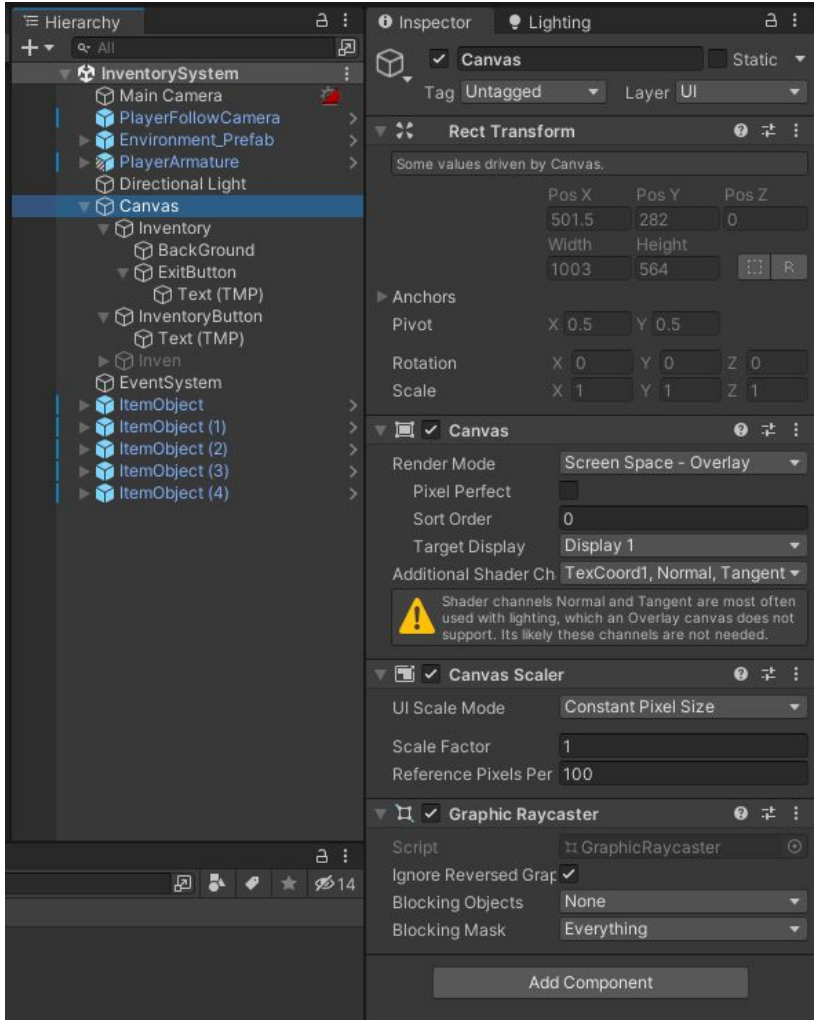
This is the inventory item class which is a scriptable object that has a Create asset menu keyword at the top, this allows scriptable object to be reused easily by creating it from inside the engine and then assigning the data for the asset.
This asset also contains the code for using the item based on what ItemType it is.

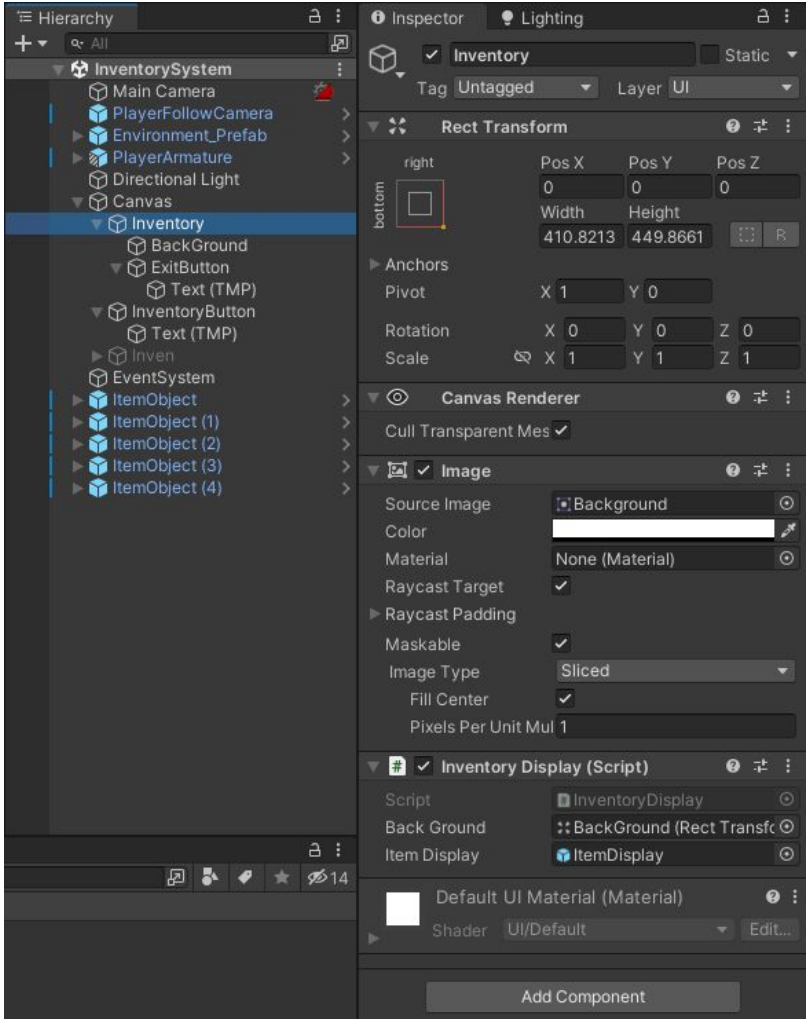


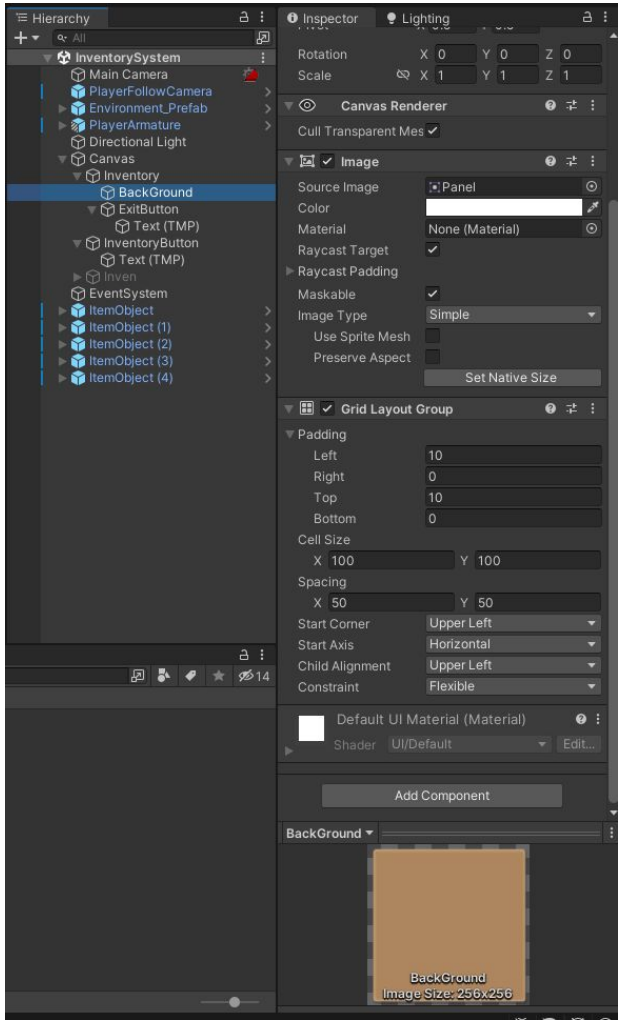

Play
Create
Show in Explorer
Open
Delete
Rename
Copy Path
Alt+Ctrl+C
Inventory System
Folder
C# Script
2D
Visual Scripting
Shader
Shader Variant Collection
Testing
Playables
Assembly Definition
Assembly Definition Reference
Text
TextMeshPro
Scene
Items
All
InventoryS
Main Ca
PlayerFo
Environm
PlayerAr
Direction
Canvas
EventSy
ItemObje
Cube
ItemObje
ItemObje
ItemObje
ItemObje

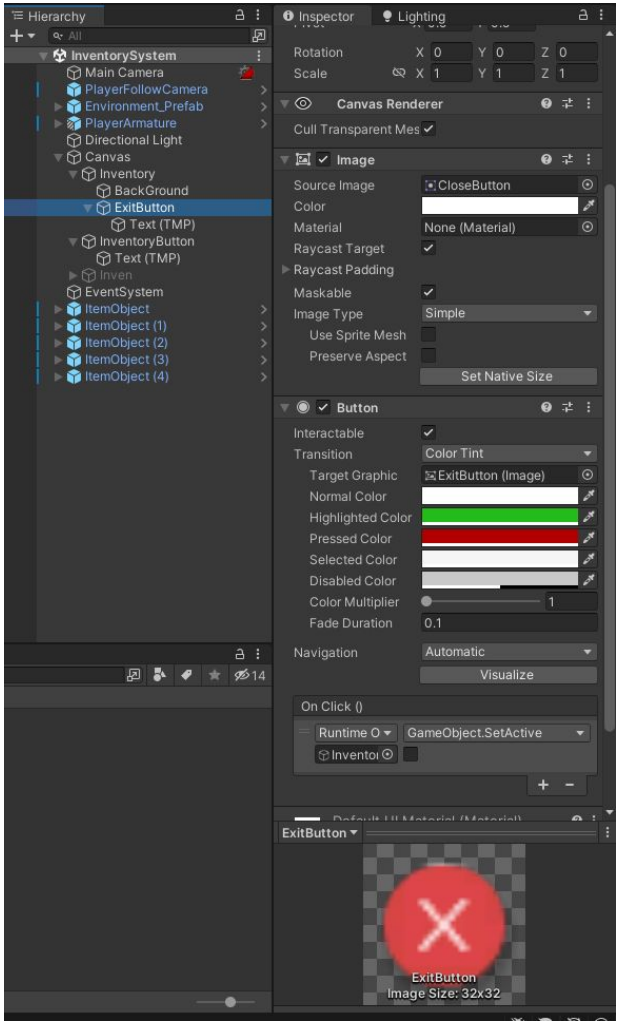

Inspector
Lighting
Sword (Inventory Item)
Open
Script
InventoryItem
Prefab
None (Game Object)
Item Name
Sword
Type
Weapon
Item Description
Item Icon
None (Sprite)
Amount
1

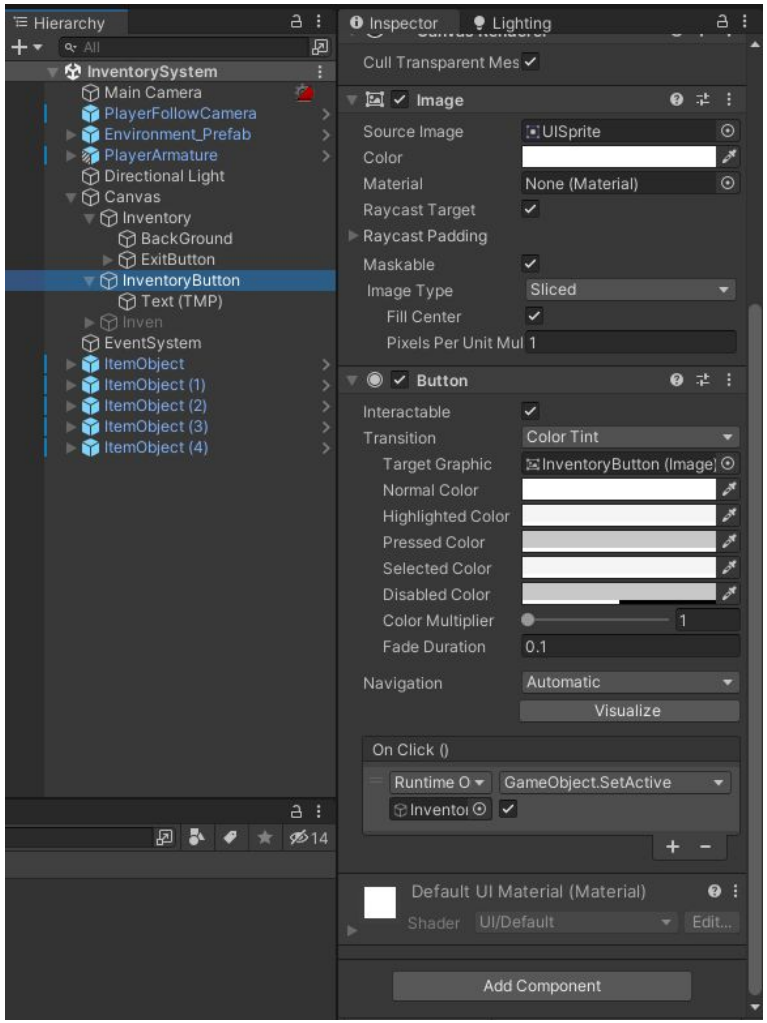

Hierarchy
All
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
EventSystem
ItemObject
Cube
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
ItemObject
Static
Tag Untagged
Layer Default
Prefab Open Select Overrides
Transform
Position X -0.463 Y 3.8613 Z -4.96
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Info
Sphere Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center
X 0 Y 0 Z 0
Radius 0.5
Inventory Object (Script)
Script InventoryObject
Item Sword (Inventory Item)
Item Name
Amount 0
Item Icon None (Sprite)
Add Component

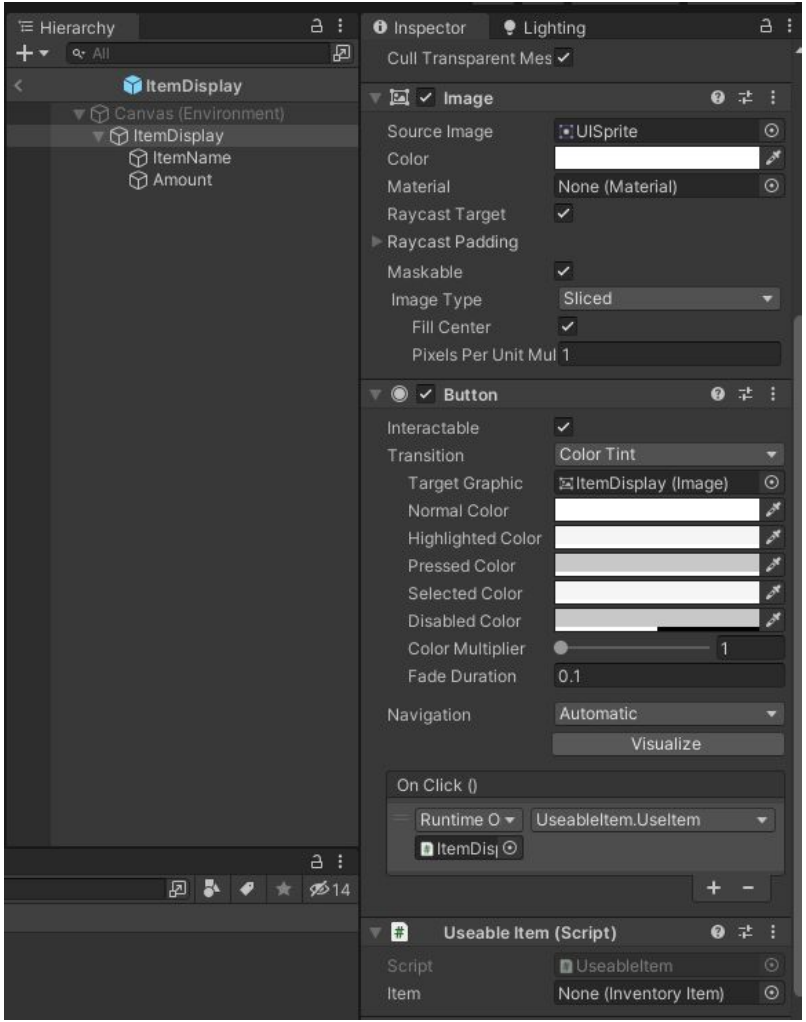

Hierarchy
All
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
Inventory
BackGround
ExitButton
Text (TMP)
InventoryButton
Text (TMP)
Inven
EventSystem
ItemObject
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
Canvas
Static
Tag Untagged
Layer UI
Rect Transform
Some values driven by Canvas.
Pos X Pos Y Pos Z
501.5 282 0
Width Height
1003 564
Anchors
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas
Render Mode Screen Space - Overlay
Pixel Perfect
Sort Order 0
Target Display Display 1
Additional Shader Ch TexCoord1, Normal, Tangent
Shader channels Normal and Tangent are most often used with lighting, which an Overlay canvas does not support. Its likely these channels are not needed.
Canvas Scaler
UI Scale Mode Constant Pixel Size
Scale Factor 1
Reference Pixels Per 100
Graphic Raycaster
Script GraphicRaycaster
Ignore Reversed Grap
Blocking Objects None
Blocking Mask Everything
Add Component


Hierarchy
All
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
Inventory
BackGround
ExitButton
Text (TMP)
InventoryButton
Text (TMP)
Inven
EventSystem
ItemObject
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
Inventory
Static
Tag Untagged
Layer UI
Rect Transform
right
bottom
Pos X
Pos Y
Pos Z
0
0
0
Width
Height
410.8213
449.8661
Anchors
Pivot
X 1
Y 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Canvas Renderer
Cull Transparent Mes
Image
Source Image
Background
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Sliced
Fill Center
Pixels Per Unit Mul 1
Inventory Display (Script)
Script
InventoryDisplay
Back Ground
BackGround (Rect Transfc
Item Display
ItemDisplay
Default UI Material (Material)
Shader
UI/Default
Edit...
Add Component


Hierarchy
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
Inventory
BackGround
ExitButton
Text (TMP)
InventoryButton
Text (TMP)
Inven
EventSystem
ItemObject
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
Rotation
Scale
Canvas Renderer
Cull Transparent Mes
Image
Source Image
Panel
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size
Grid Layout Group
Padding
Left
10
Right
0
Top
10
Bottom
0
Cell Size
X 100
Y 100
Spacing
X 50
Y 50
Start Corner
Upper Left
Start Axis
Horizontal
Child Alignment
Upper Left
Constraint
Flexible
Default UI Material (Material)
Shader
UI/Default
Edit...
Add Component
BackGround
BackGround
Image Size: 256x256


Hierarchy
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
Inventory
BackGround
ExitButton
Text (TMP)
InventoryButton
Text (TMP)
Inven
EventSystem
ItemObject
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
Rotation
Scale
Canvas Renderer
Cull Transparent Mes
Image
Source Image
CloseButton
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size
Button
Interactable
Transition
Color Tint
Target Graphic
ExitButton (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime O
GameObject.SetActive
ExitButton
Image Size: 32x32


Hierarchy
InventorySystem
Main Camera
PlayerFollowCamera
Environment_Prefab
PlayerArmature
Directional Light
Canvas
Inventory
BackGround
ExitButton
InventoryButton
Text (TMP)
Inven
EventSystem
ItemObject
ItemObject (1)
ItemObject (2)
ItemObject (3)
ItemObject (4)
Inspector
Lighting
Cull Transparent Mes
Image
Source Image
UISprite
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Sliced
Fill Center
Pixels Per Unit Mul
1
Button
Interactable
Transition
Color Tint
Target Graphic
InventoryButton (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime O
GameObject.SetActive
Default UI Material (Material)
Shader
UI/Default
Edit...
Add Component


Hierarchy
All
ItemDisplay
Canvas (Environment)
ItemDisplay
ItemName
Amount
14
Inspector
Lighting
Cull Transparent Mes
Image
Source Image
UISprite
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Sliced
Fill Center
Pixels Per Unit Mul
1
Button
Interactable
Transition
Color Tint
Target Graphic
ItemDisplay (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
1
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime O
UseableItem.UseItem
ItemDis
Useable Item (Script)
Script
UseableItem
Item
None (Inventory Item)

```
public class InventoryObject : MonoBehaviour
{

    public InventoryItem item;

    public string itemName;
    public int amount;
    public Sprite itemIcon;
```

```
private void Awake()
{
    BeginInventoryObject();
}

public void BeginInventoryObject()
{
    itemName = item.itemName;
    amount = item.amount;
    itemIcon = item.itemIcon;
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        PlayerInventory playerInventory = other.GetComponent<PlayerInventory>();
        playerInventory.AddNewItem(this);
        Destroy(this.gameObject);
    }
}
```

This is just a script that reacts to the player picking it up, it simply call the function on the players' Player Inventory script to add the item to their inventory then the gameobject that owns this Inventory object destroys it self.

This is the player inventory class it adds a new item to the inventory items list. Adding a new item is done by first check if the item already exists in the list, if so then it will simply increment the amount of the item by the amount that has been picked up, if not then it will create a new item in the list and also create a communicate with the inventory display scripts to create a new item.

```
public class PlayerInventory : MonoBehaviour
{
    public InventoryDisplay display;

    public List<InventoryItem> inventoryItems;

    public List<InventoryObject> inventoryObjects;

    void Start()
    {
        inventoryItems = new List<InventoryItem>();
    }


    public void UseItem(InventoryItem item)
    {
        switch (item.type)
        {
            case InventoryItem.ItemType.Weapon:
                break;
            case InventoryItem.ItemType.Potion:
                break;
            case InventoryItem.ItemType.Coin:
                break;

        }
        RemoveItem(item);
    }
```

```
public void AddNewItem(InventoryObject item)
{
    int newIndex = 0;
    if (inventoryItems.Count > 0)
    {
        foreach (InventoryItem items in inventoryItems)
        {
            newIndex--;
            if (items.itemName == item.itemName)
            {
                items.amount += item.amount;
                break;
            }
            else
            {
                inventoryItems.Add(item.item);
                display.AddNewItem(item.item);
                break;
            }
        }
    }
    else
    {
        inventoryItems.Add(item.item);
        display.AddNewItem(item.item);
    }
}

public void RemoveItem(InventoryItem item)
{
    inventoryItems.Remove(item);
}
```

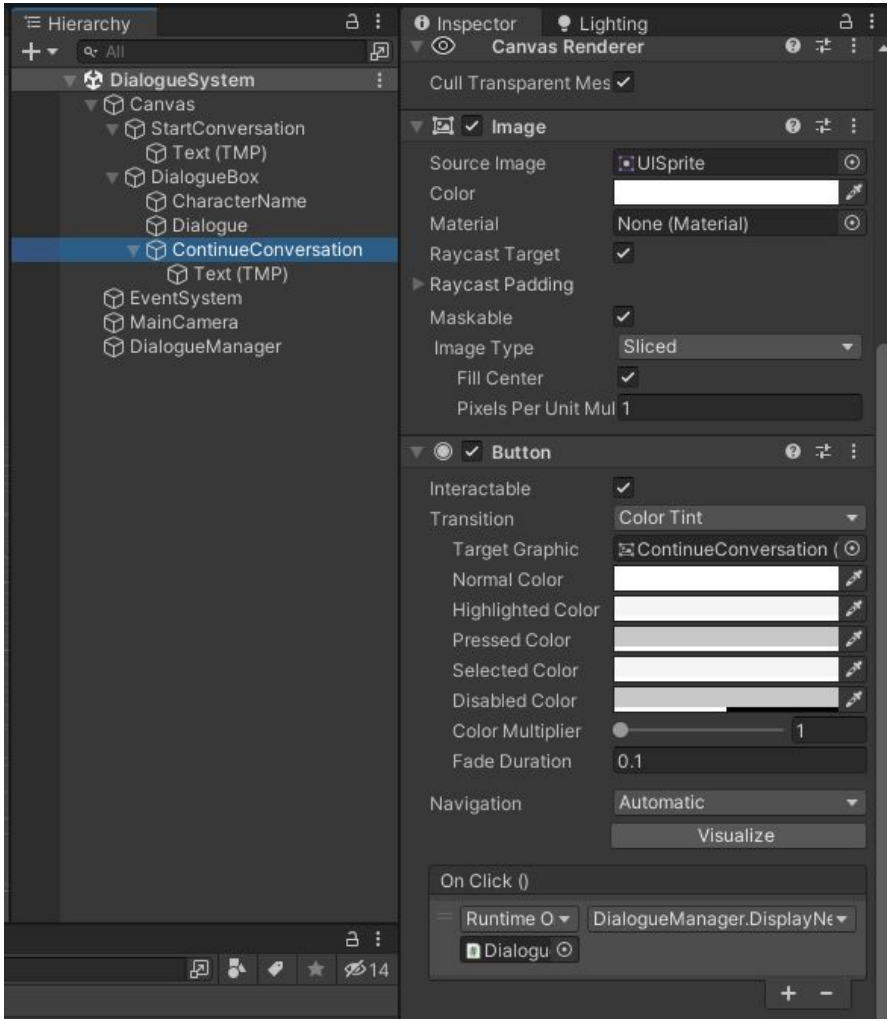
```
public class UseableItem : MonoBehaviour
{
    public InventoryItem item;

    public void UseItem()
    {
        item.UseItem();
    }
}
```

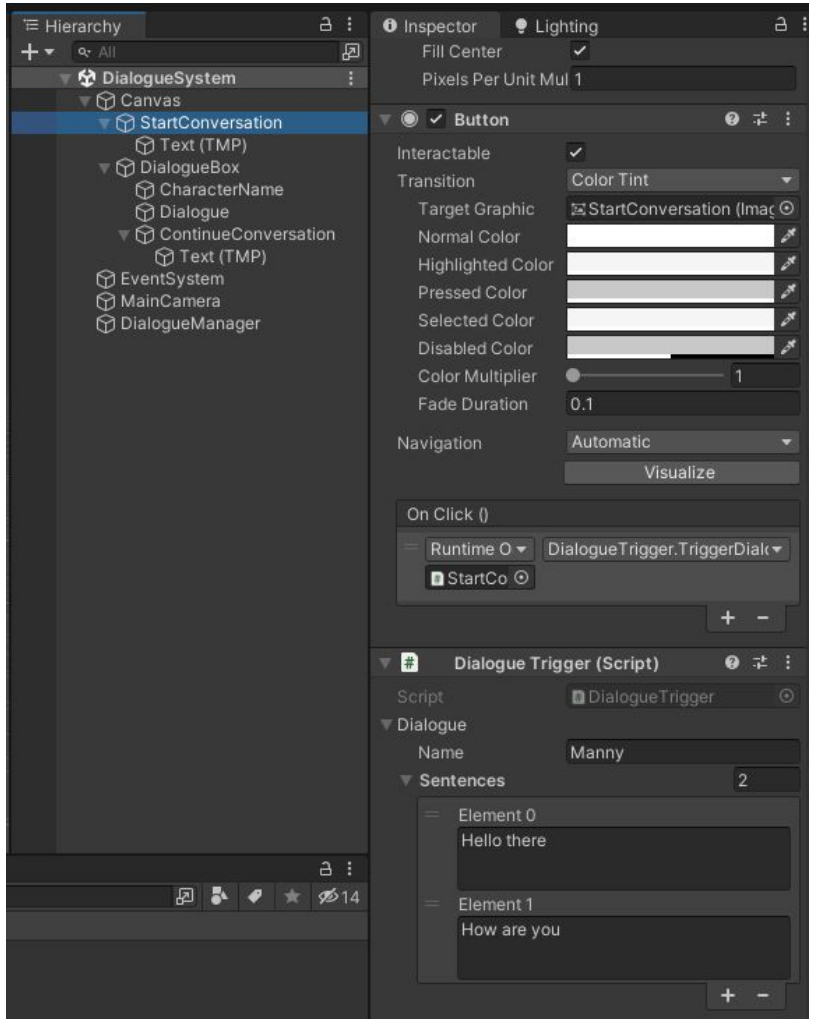
This sits on the item display gameobject, use this with a button to use the item that has been selected

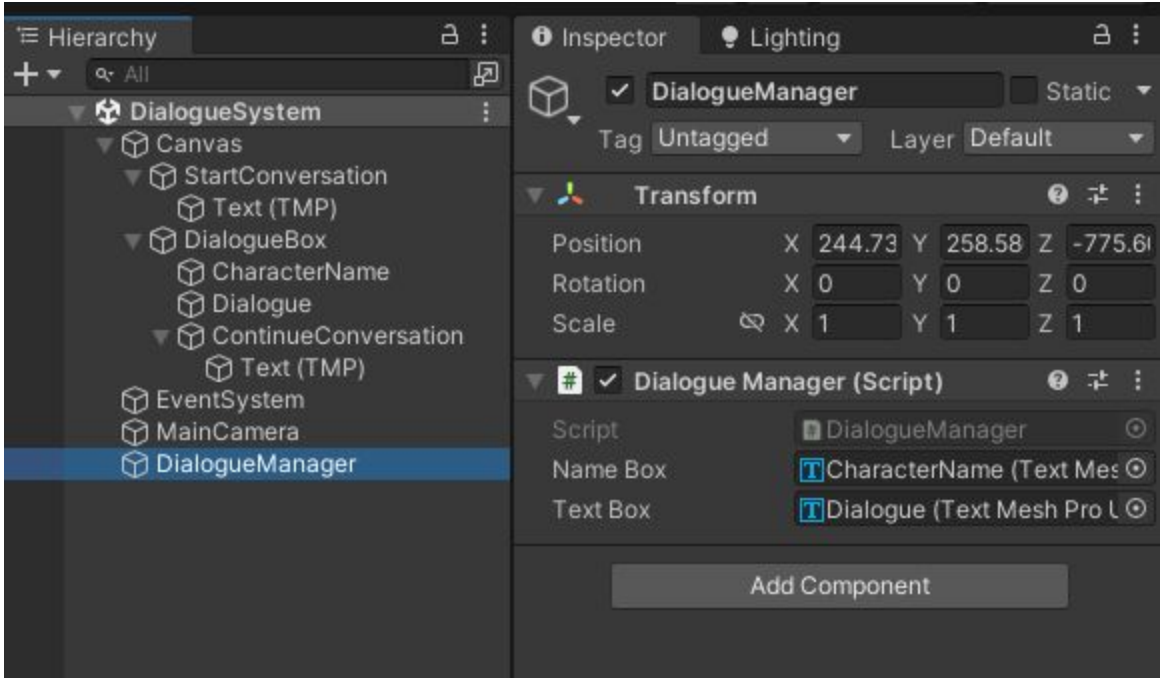
Dialogue System

```
[System.Serializable]
public class Dialogue
{
    public string name;
    [TextArea(3,10)]
    public string[] sentences;
}
```

A Dialogue class, this holds an array of sentences that a character will say as well as their name.
Making it serializable with make sure it can be instanced in the editor for easy editing.


Hierarchy
Inspector
Lighting
DialogueSystem
Canvas
StartConversation
Text (TMP)
DialogueBox
CharacterName
Dialogue
ContinueConversation
Text (TMP)
EventSystem
MainCamera
DialogueManager
Canvas Renderer
Cull Transparent Mes
Image
Source Image
UISprite
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Sliced
Fill Center
Pixels Per Unit Mul
1
Button
Interactable
Transition
Color Tint
Target Graphic
ContinueConversation (
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
1
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime O
DialogueManager.DisplayNe
Dialogu
14


Hierarchy
Inspector
Lighting
DialogueSystem
Canvas
StartConversation
Text (TMP)
DialogueBox
CharacterName
Dialogue
ContinueConversation
Text (TMP)
EventSystem
MainCamera
DialogueManager
Fill Center
Pixels Per Unit Mul
1
Button
Interactable
Transition
Color Tint
Target Graphic
StartConversation (Imag
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
1
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime O
DialogueTrigger.TriggerDialo
StartCo
Dialogue Trigger (Script)
Script
DialogueTrigger
Dialogue
Name
Manny
Sentences
2
Element 0
Hello there
Element 1
How are you
14


Hierarchy
All
DialogueSystem
Canvas
StartConversation
Text (TMP)
DialogueBox
CharacterName
Dialogue
ContinueConversation
Text (TMP)
EventSystem
MainCamera
DialogueManager
Inspector
Lighting
DialogueManager
Static
Tag
Untagged
Layer
Default
Transform
Position
X 244.73
Y 258.58
Z -775.6
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Dialogue Manager (Script)
Script
DialogueManager
Name Box
CharacterName (Text Mes
Text Box
Dialogue (Text Mesh Pro L
Add Component

```
public class DialogueManager : MonoBehaviour
{
    private Queue<string> sentences;
    public TextMeshProUGUI nameBox;
    public TextMeshProUGUI textBox;
```

The dialogue manage uses a queue to handle all of the sentences in the dialogue, this is because the Enqueue and Dequeue methods allow for easy removal and procedure through all of the strings in the queue.

The start dialogue starts when the player clicks a button or whatever else you decide and first sets the name of the character speaking and clears previous dialogue if there is any, then adds all of the sentences from the dialogue class to the queue using the enqueue, and starts the Display Next Sentence method. This method first check if there are still sentences in the dialogue then remove the first item in the queue and displays, this method can be called using a button to carry on the conversation. This continues until the sentence count is equal to 0 then the conversation is ended.

```
void Start()
{
    sentences = new Queue<string>();
}

public void StartDialogue(Dialogue dialogue){

    nameBox.text = dialogue.name;

    sentences.Clear();

    foreach(string sentence in dialogue.sentences){
        sentences.Enqueue(sentence);
    }
    DisplayNextSentence();
}

public void DisplayNextSentence(){
    if (sentences.Count == 0){
        EndDialogue();
        return;
    }
    string sentence = sentences.Dequeue();
    textBox.text = sentence;
}

void EndDialogue(){
    Debug.Log("End Conversation");
}
```

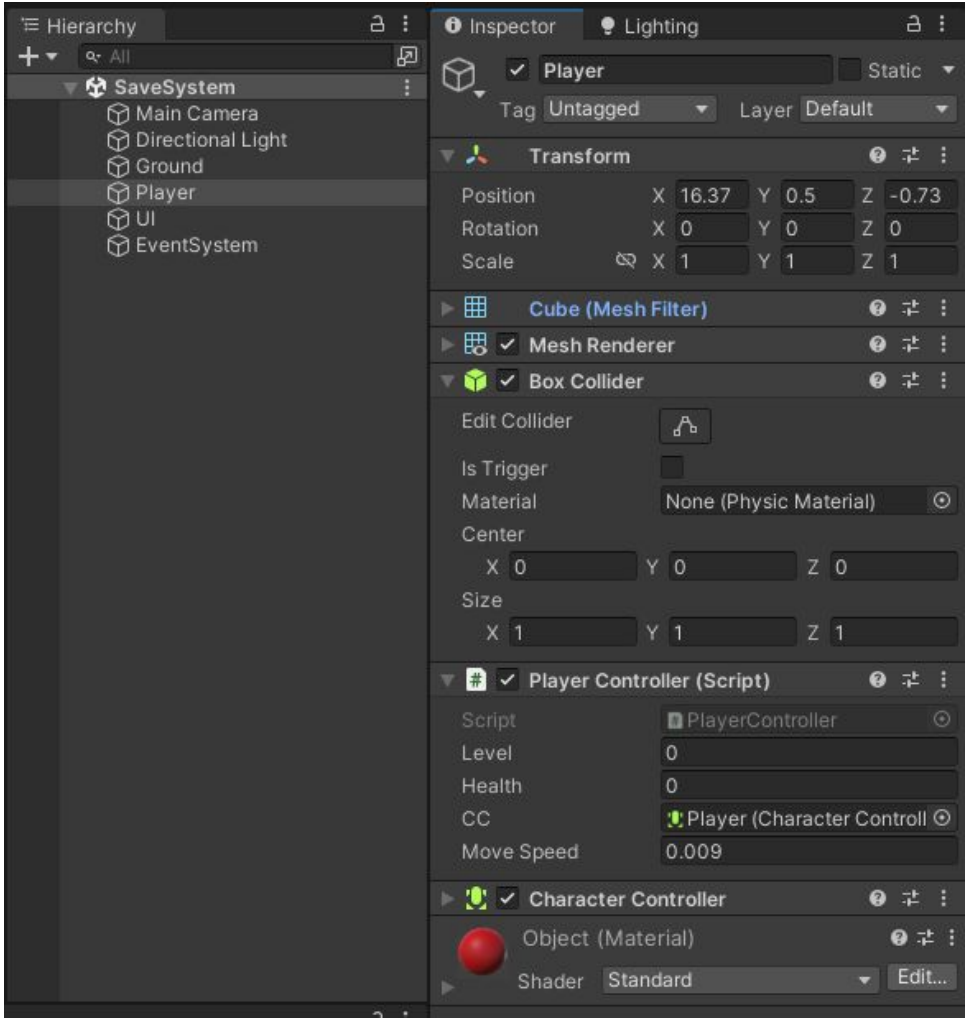
```
public class DialogueTrigger : MonoBehaviour
{
    public Dialogue dialogue = new Dialogue();

    public void TriggerDialogue(){
        FindObjectOfType<DialogueManager>().StartDialogue(dialogue);
    }

}
```

This is what is used to start the dialogue from the dialogue manager

Save System

```
public class PlayerController : MonoBehaviour
{

    public int level = 0;
    public int health = 0;
```

This is just an example of data that can be held in a stats class and saved


Hierarchy
All
SaveSystem
Main Camera
Directional Light
Ground
Player
UI
EventSystem
Inspector
Lighting
Player
Static
Tag Untagged
Layer Default
Transform
Position X 16.37 Y 0.5 Z -0.73
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Cube (Mesh Filter)
Mesh Renderer
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center
X 0 Y 0 Z 0
Size
X 1 Y 1 Z 1
Player Controller (Script)
Script PlayerController
Level 0
Health 0
CC Player (Character Controll
Move Speed 0.009
Character Controller
Object (Material)
Shader Standard
Edit...

```
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
```

```
public static PlayerStats LoadPlayer() {
    string path = Application.persistentDataPath + "/player.noz";
    if (File.Exists(path))
    {
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        PlayerStats stats = formatter.Deserialize(stream) as PlayerStats;

        return stats;
    }
    else
    {
        Debug.LogError("Save file not found in" + path);
        return null;
    }
}
```

```
public static class SaveSystem
{
    public static void SavePlayer(PlayerController player)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/player.noz";
        FileStream stream = new FileStream(path, FileMode.Create);

        PlayerStats stats = new PlayerStats(player);

        formatter.Serialize(stream, stats);
        stream.Close();


    }
```

The save system class doesn't need a monobehaviour as it won't be spawned into the scene.
This class has 2 method Load and Save
The Save method uses a binaryformatter to serialize data
The path is where on the user's system this save file will be located, persistent Data path is a general path that all operating systems have and is common path for saving data, this is followed by the extension which is custom to what you want.
The stream is used to either open or create a file path
And the stats is the data we want to save
We use the formatter to save the stats through the filestream then we close the stream as this can stay in memory
Load player is similar to this but we first have to check whether the save file exist first then we use file mode open to open up a pre existing file path to access its data.
Make use to the add the using statements at the top to be able to have access to the binaryformatter class and file stream class.

```
public class PlayerStats
{
    public string Name;
    public int Health;
    public int Level;
    public float[] Position;


    public PlayerStats(PlayerController player)
    {
        Health = player.health;
        Level = player.level;

        Position = new float[3];
        Position[0] = player.transform.position.x;
        Position[1] = player.transform.position.y;
        Position[2] = player.transform.position.z;

    }
}
```