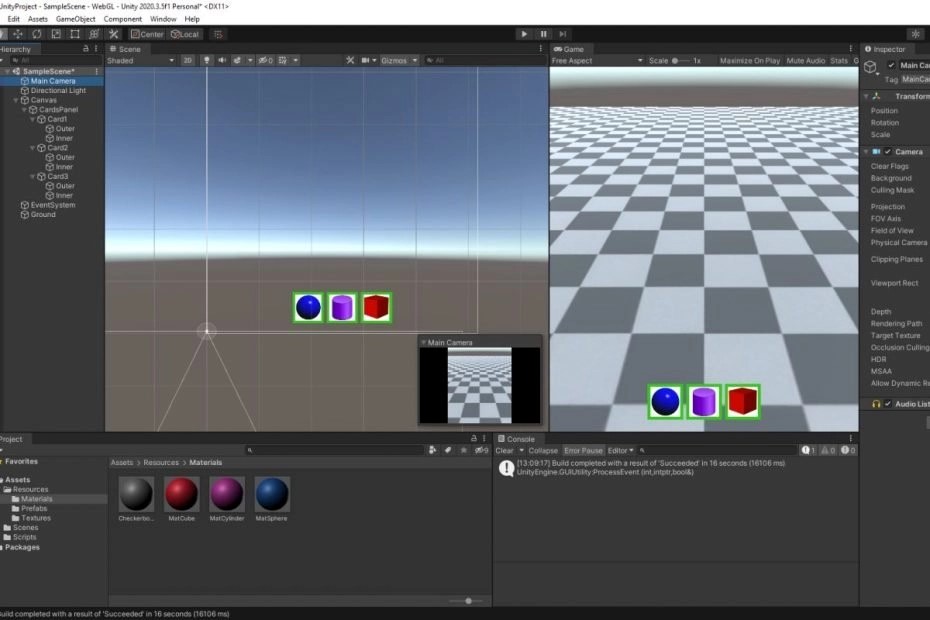
Implement Drag and Drop Item in Unity



**Shamim Akhtar**

This tutorial guides students through implementing a drag-and-drop item functionality in Unity. It covers creating a scene with a ground plane, dummy prefabs, canvas, and UI items. The provided C# script enables dragging UI elements onto the scene to instantiate corresponding prefabs. Through step-by-step instructions and code explanations, learners will be able to associate UI elements with prefabs, handle drag events, and instantiate objects in response to user interaction.

This tutorial also serves as a practical introduction to utilising C# interfaces. By implementing the **IBeginDragHandler**, **IDragHandler**, and **IEndDragHandler** interfaces in the provided script, students will gain hands-on experience in working with interfaces in C#.

They will understand how interfaces facilitate the implementation of specific behaviours, such as handling drag events in this case, by enforcing a contract that defines required methods.

Through this tutorial, users not only learn how to implement drag-and-drop mechanics but also grasp the concept and practical application of interfaces in C#, enhancing their understanding of object-oriented programming principles within the Unity environment.

Click to try the WebGL version below. NOTE: The WebGL version might not work for mobile devices.

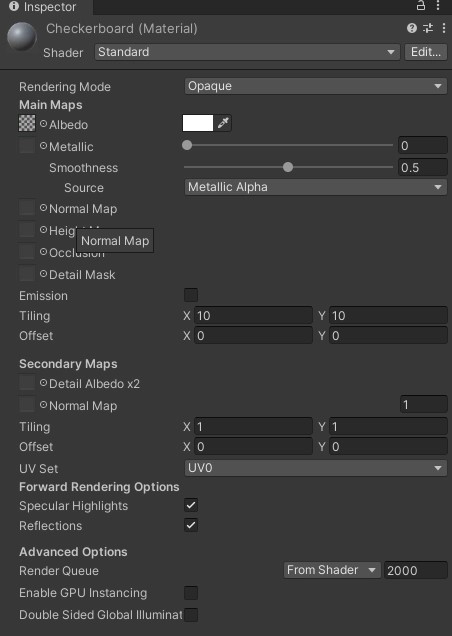
<https://faramira.com/downloads/dragdrop/WebGL/>

# The Unity Project

Create a new Unity 3D project.

## The Ground

* Right-click on the Hierarchy window and add a plan. Name it Ground.
* Create a folder in Assets and call it Resources. Create another folder in Resources and name it Textures. Download the [checkerboard](https://github.com/shamim-akhtar/drag-and-drop/blob/main/UnityProject/Assets/Resources/Textures/Checkerboard.jpg)image and put it in the folder Assets/Resources/Textures. Drag and drop this image onto the ground.
* Set the Ground’s scale to be 10, 1, 10.
* Double-click on the checkerboard material and open the material file. Set the tile factor to 10, 10 as shown below.
* Checkerboard material for the ground.



* Add a Box Collider component to the ground.
* The Ground’s Inspection should look like below.

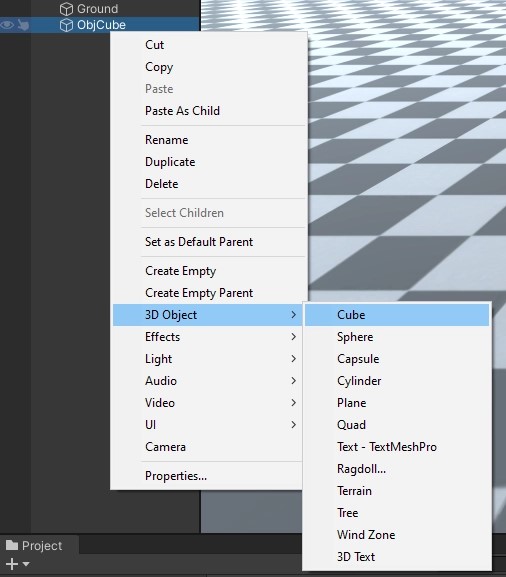
Inspector view of the Ground game object

## The Dummy Prefabs

Next, we will create some dummy prefabs to initialise by dragging and dropping UI items. We will associate each UI item with one such prefab so that when we drop the UI item, we instantiate the correct prefab associated with that UI item.

For our demo, we will create a cube, a sphere, and a cylinder. Go ahead and carry out the following steps.

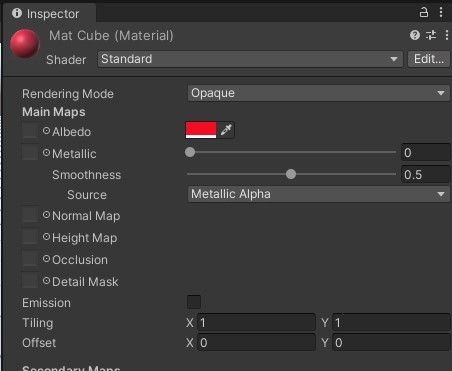
* Right-click on the Hierarchy window and create a new empty game object. Name it **ObjCube** and reset the transform.
* Select **ObjCube** and add a cube game object as shown below.



* Add a cube game object.

Create a material and name it **MatCube**. Set a colour to the material by selecting the Albedo, as shown below.

* Set this material to the Cube.



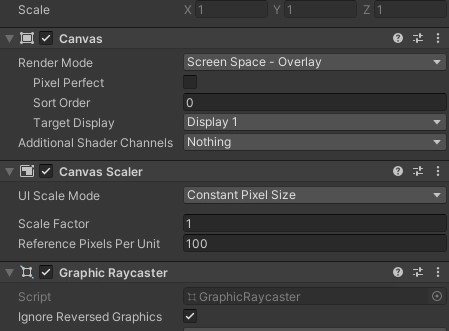
* Create a folder in the Resources directory; name it Prefabs. Drag and drop ObjCube to the Prefabs folder and make it a prefab.
* Delete ObjCube from the Hierarchy.

Repeat the above steps for a Sphere and a Cylinder. You can create a different material for the Sphere and the Cylinder. Your Prefabs folder should now have three prefabs.

* ObjCube
* ObjSphere
* ObjCylinder

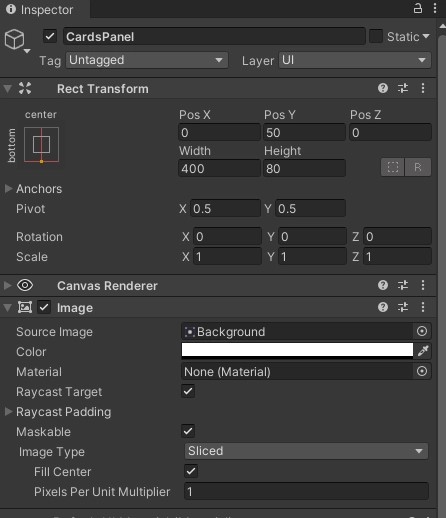
## The Canvas and the UI Item

Next, we create the Canvas and the UI items that we will drag.

* Right-click on the Hierarchy window and create a new Canvas. Make sure that we have the settings as shown in the diagram below.

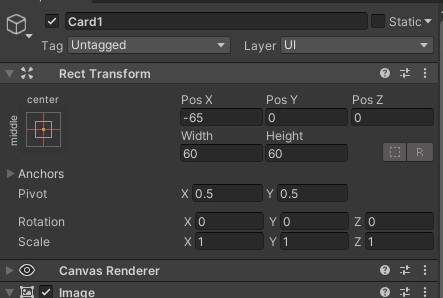
Settings for the canvas

* Select the canvas and add a new UI Panel. Name this Panel as **CardsPanel**. Set the **CardPanel’s** parameters as shown below.

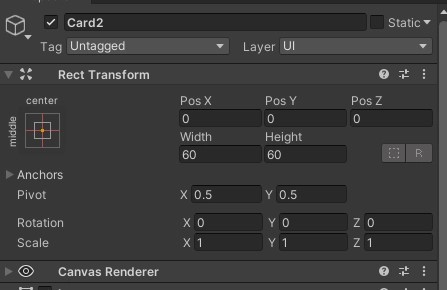


The UP-Panel parameters

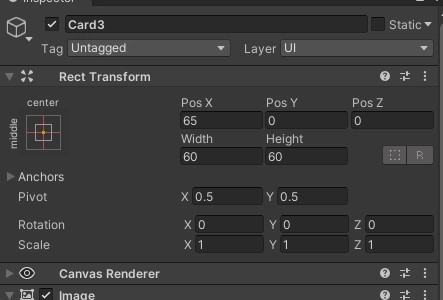
* Create three subpanels within this **CardPanel**. Name the first sub-panel as **Card1**, the second as **Card2** and the third as **Card3**.



Position of Card1 Panel

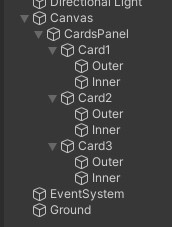


Position of Card2 Panel



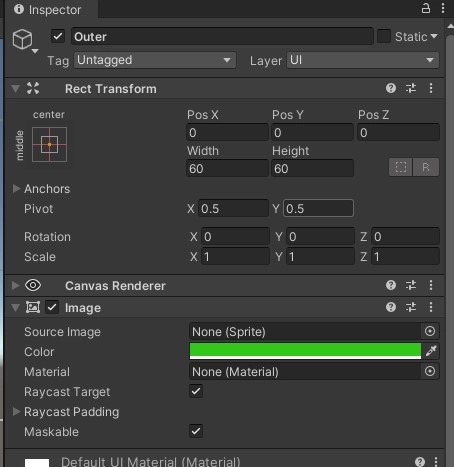
Position of Card3 Panel

* To each of these sub-panels, add two sprites. Name them Inner and Outer.

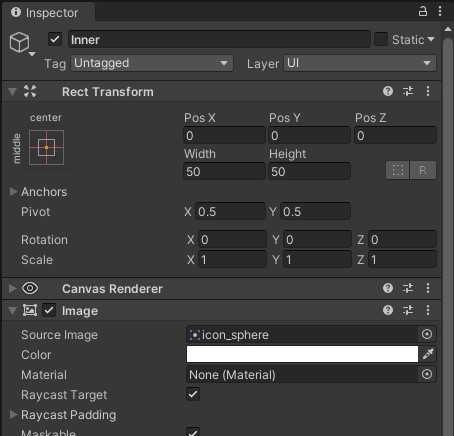


Inner and Outer sprites in each sub-panel

* Set the Outer sprite’s parameter as shown in the picture below.

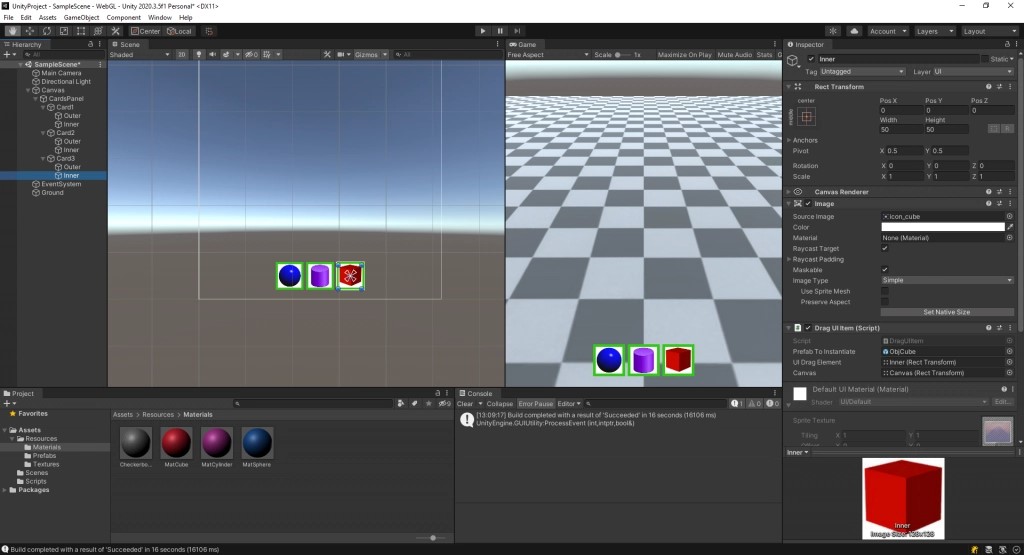


* The Outer sprite’s parameters.
* Download the [icon for the cube](https://github.com/shamim-akhtar/drag-and-drop/blob/main/UnityProject/Assets/Resources/Textures/icon_cube.jpg), the [icon for the sphere](https://github.com/shamim-akhtar/drag-and-drop/blob/main/UnityProject/Assets/Resources/Textures/icon_sphere.jpg) and the[icon for the cylinder](https://github.com/shamim-akhtar/drag-and-drop/blob/main/UnityProject/Assets/Resources/Textures/icon_cylinder.jpg). Put these images in the Textures folder. Convert them to 2D sprite type.
* Select the Inner sprite for Card1 and then set the width and height to be 50, 50. Drag and drop **icon\_sphere** as set as the sprite for the Inner (as shown below).



Set the width, height, and the sprite for the Inner for **Card1**.

* Similarly, for Card2 Inner, set the **icon\_cylinder** and for Card3 Inner, set **icon\_cube**.
* Your UI should like the below picture.



This picture shows the final UI for our demonstration.

* Select the Camera and set the transform values as below. These values will make the camera position and orientation a bit better for us to view the ground.



* Set the camera position and orientation to these values for a better view of the scene.

## The Drag UI Item Script

So far, we have only created our scene for testing the drag and drop of UI items to instantiate prefabs on the scene. We will now begin the necessary script for our main feature.

* Go ahead and create a new C# script file named **DragUIItem**.
* Double-click this file and open it in Visual Studio.
* Make the following change to the class declaration.

public class DragUIItem :

MonoBehaviour,

IBeginDragHandler,

IDragHandler,

IEndDragHandler

{

}

* We want to implement three interfaces for our drag and drop to work. These interfaces are **IBeginDragHandler**, **IDragHandler**, and **IEndDragHandler**.
* Add the following variables to the class.

[SerializeField]

GameObject PrefabToInstantiate;

[SerializeField]

Retransforms UIDragElement;

[SerializeField]

RectTransform Canvas;

private Vector2 mOriginalLocalPointerPosition;

private Vector3 mOriginalPanelLocalPosition;

private Vector2 mOriginalPosition;

* Line 2 from the above code stores the reference to the prefab that we want to instantiate.
* The variable in line 4 holds the reference to the UI element that we will drag. The variable in line 6 contains the reference to the canvas.

### The Start Method

* In the Start method, we store the initial position of the draggable UI item.

private void Start()

{

mOriginalPosition = UIDragElement.localPosition;

}

### Implement Interface Functions

* Next, we implement the interface functions. These are:

#### OnBeginDrag

public void OnBeginDrag(PointerEventData data)

{

mOriginalPanelLocalPosition = UIDragElement.localPosition;

RectTransformUtility.ScreenPointToLocalPointInRectangle(

Canvas,

data. Position,

data.pressEventCamera,

out mOriginalLocalPointerPosition);

}

#### OnDrag

public void OnDrag(PointerEventData data)

{

Vector2 localPointerPosition;

if (RectTransformUtility.ScreenPointToLocalPointInRectangle(

Canvas,

data.position,

data.pressEventCamera,

out localPointerPosition))

{

Vector3 offsetToOriginal =

localPointerPosition -

mOriginalLocalPointerPosition;

UIDragElement.localPosition =

mOriginalPanelLocalPosition +

offsetToOriginal;

}

}

#### OnEndDrag

public void OnEndDrag(PointerEventData eventData)

{

StartCoroutine(

Coroutine\_MoveUIElement(

UIDragElement,

mOriginalPosition,

0.5f));

RaycastHit hit;

Ray ray = Camera.main.ScreenPointToRay(

Input.mousePosition);

if (Physics.Raycast(ray, out hit, 1000.0f))

{

Vector3 worldPoint = hit.point;

//Debug.Log(worldPoint).

CreateObject(worldPoint);

}

}

* In the above code, the highlighted section uses a coroutine to reset the draggable UI’s position back to its initial position.

public IEnumerator Coroutine\_MoveUIElement(

RectTransform r,

Vector2 targetPosition,

float duration = 0.1f)

{

float elapsedTime = 0;

Vector2 startingPos = r.localPosition;

while (elapsedTime < duration)

{

r.localPosition =

Vector2.Lerp(

startingPos,

targetPosition,

(elapsedTime / duration));

elapsedTime += Time.deltaTime;

yield return new WaitForEndOfFrame();

}

r.localPosition = targetPosition;

}

* Then, we do a raycast to determine where the ray intersects on the ground. If there is a hit, then we call the CreateObject method.
* Finally, the CreateObject method is as below.

public void CreateObject(Vector3 position)

{

if (PrefabToInstantiate == null)

{

Debug.Log("No prefab to instantiate");

return;

}

GameObject obj = Instantiate(

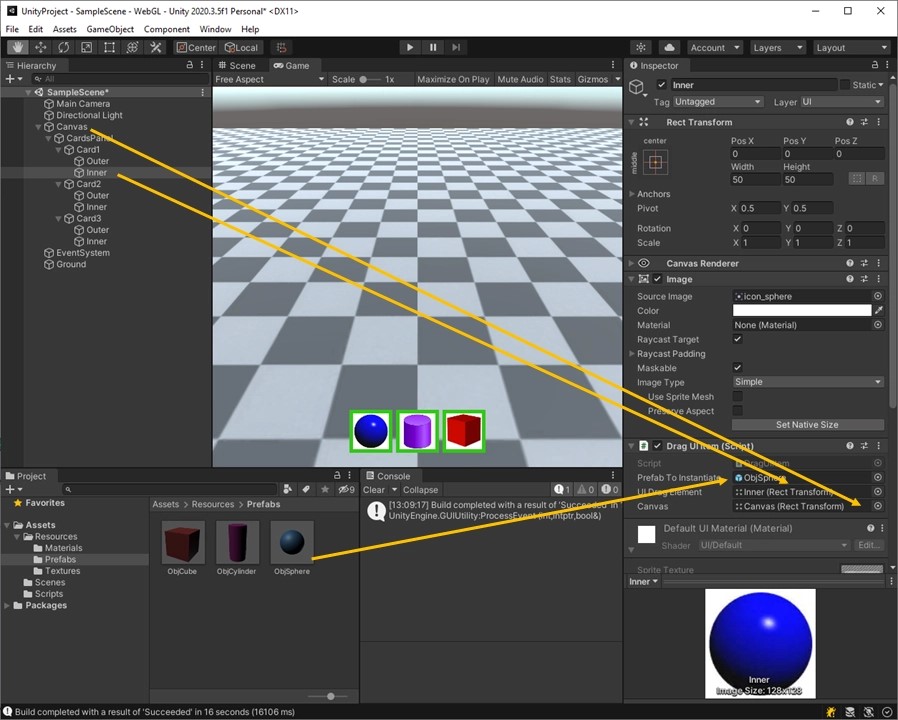
PrefabToInstantiate,

position,

Quaternion.identity);

}

* Attach the Script and Configure
* Go back to Unity editor and attach the DragUIItem script to each of the Inner images of Card1, Card2 and card3.
* Select Inner from Card1 and associate the serialisable fields as below.

Associate the prefab, the draggable UI item and the canvas to the serialisable fields.

* Similarly, do the same process for Inner from card2 and Inner from Card3 (with different prefabs).

Click Play and run the application. You should now drag the UI items onto the Ground and see the instantiated prefabs on the ground, as shown below.

<https://youtu.be/DsrYejNt5O0>