STATEMENT OF ORIGINALITY

I certify that the following used in the creation of this prototype are my own original work:

- Design concept, testing plan, and overall interaction design.
- Unity project setup and integration.

I also acknowledge that I used example code provided (PlayerMovement.cs) on **Blackboard (DECO7230 Digital Prototyping, The University of Queensland, Semester 2, 2025)**. These code snippets were adapted by me to suit my prototype's requirements.

I acknowledge that I used OpenAI ChatGPT (2025) to generate and assist with parts of the Unity C# scripts (DragObject.cs, HoverEnlarge.cs, TrimVideo.cs). These scripts were adapted and modified by me to fit the requirements of my project.

References for AI assistance and any other external sources are provided in the reference section.

REFERENCE

Code Snippets / Unity Scripts:

1. (Drag Object) retrieved from OpenAI ChatGPT: How to drag and place objects in Unity using mouse click and hold (https://chat.openai.com/) Last accessed 29/8/2025

2. (Hover Enlarge) retrieved from OpenAI ChatGPT: How to enlarge a Unity object on mouse hover and reset when mouse leaves (https://chat.openai.com/) Last accessed 29/8/2025

3. (Trim Video) retrieved from OpenAI ChatGPT: How to simulate video trimming in Unity using mouse click and drag on a cube (https://chat.openai.com/) Last accessed 29/8/2025

Images / Assets:

1. Wedding Photos: retrieved from Brides (https://www.brides.com/gallery/must-have-wedding-photos) Last accessed 29/8/2025

2. Wood Wall Texture: retrieved from Vecteezy (https://www.vecteezy.com/free-photos/wood-wall-texture) Last accessed 29/8/2025

Identifying External Sources in my code:

Code snippet used as it appears in the external source, with no changes/adaptations:

//DragObject//

```
using UnityEngine;


[RequireComponent(typeof(Collider))]
public class DragObject : MonoBehaviour
{
    private Camera cam;
    private bool isDragging = false;
    private Vector3 offset;
    private float distanceToCamera;


    void Start()
    {
        cam = Camera.main; // grab main camera
    }


    void OnMouseDown()
```

```
    {
        // Calculate distance from camera to object
        distanceToCamera = Vector3.Distance(transform.position, cam.transform.position);

        // Convert mouse position to world space
        Vector3 mouseWorld = cam.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, distanceToCamera));

        // Save offset so object doesn't snap to mouse
        offset = transform.position - mouseWorld;

        isDragging = true;
    }

    void OnMouseUp()
    {
        isDragging = false;
    }

    void Update()
    {
        if (isDragging)
        {
            // Convert mouse to world position and apply offset
```

```
            Vector3 mouseWorld = cam.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, distanceToCamera));

            transform.position = mouseWorld + offset;

        }

    }

}
```

//HoverEnlarge//

```
using UnityEngine;


public class HoverEnlarge : MonoBehaviour

{

    private Vector3 originalScale;

    public Vector3 hoverScale = new Vector3(1.5f, 1.5f, 1.5f); // Size when hovered


    void Start()

    {

        // Save the original scale of the object

        originalScale = transform.localScale;

    }


    void OnMouseEnter()

    {

        // When mouse hovers over the object, enlarge it
```

```
        transform.localScale = hoverScale;
    }

    void OnMouseExit()
    {
        // When mouse leaves, reset to original size
        transform.localScale = originalScale;
    }
}

//TrimVideo//
using UnityEngine;

[RequireComponent(typeof(BoxCollider))]
public class TrimVideoCube : MonoBehaviour
{
    private Camera cam;
    private bool isTrimming = false;
    private Vector3 offset;
    private float initialXScale;

    public enum Edge { Right, Left } // choose which edge to trim
    public Edge trimEdge = Edge.Right;
```

```
    void Start()
    {
        cam = Camera.main;
        initialXScale = transform.localScale.x;
    }

    void OnMouseDown()
    {
        // Start trimming
        isTrimming = true;

        // Store offset between mouse and cube edge
        Vector3 mouseWorld = GetMouseWorldPos();
        if (trimEdge == Edge.Right)
            offset = transform.position + new Vector3(transform.localScale.x / 2f, 0, 0) -
mouseWorld;
        else
            offset = transform.position - new Vector3(transform.localScale.x / 2f, 0, 0) - mouseWorld;
    }

    void OnMouseUp()
    {
        isTrimming = false;
    }
```

```
    void Update()
    {
        if (isTrimming)
        {
            Vector3 mouseWorld = GetMouseWorldPos() + offset;

            float newXScale = transform.localScale.x;

            if (trimEdge == Edge.Right)
            {
                // Compute new scale from left side fixed
                float leftX = transform.position.x - transform.localScale.x / 2f;
                newXScale = Mathf.Max(0.1f, mouseWorld.x - leftX);
                transform.localScale = new Vector3(newXScale, transform.localScale.y,
transform.localScale.z);
                // Adjust cube center
                transform.position = new Vector3(leftX + newXScale / 2f, transform.position.y,
transform.position.z);
            }
            else
            {
                // Trim left edge
                float rightX = transform.position.x + transform.localScale.x / 2f;
                newXScale = Mathf.Max(0.1f, rightX - mouseWorld.x);
                transform.localScale = new Vector3(newXScale, transform.localScale.y,
transform.localScale.z);
```

```
            // Adjust cube center
            transform.position = new Vector3(rightX - newXScale / 2f, transform.position.y,
transform.position.z);
        }
    }
}

Vector3 GetMouseWorldPos()
{
    Vector3 mousePos = Input.mousePosition;
    mousePos.z = cam.WorldToScreenPoint(transform.position).z;
    return cam.ScreenToWorldPoint(mousePos);
}
}
```