```csharp
using System;
using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;

/// <summary>
/// BehaviorManagerPreview handles game state actions transitions within a preview scene.
/// </summary>
public class BehaviorManagerPreview : MonoBehaviour
{
    // Singleton state reference
    private static BehaviorManagerPreview instance;

    /// <summary>
    /// The material utilized for empty inputs or outputs the user is currently hovered on.
    /// </summary>
    [SerializeField]
    Material selectedMaterial;

    /// <summary>
    /// Displays the current world position.
    /// </summary>
    [SerializeField]
    TextMeshProUGUI coordinatesText;

    /// <summary>
    /// Displays the current label of the empty input or output hovered on, if applicable.
    /// </summary>
    [SerializeField]
    TextMeshProUGUI labelText;

    /// <summary>
    /// Whether the user is currently hovered onto an empty input or output.
    /// </summary>
    private bool isInput;

    /// <summary>
    /// Whether the user is currently hovered onto a UI element.
    /// </summary>
    private bool isUILocked;

    /// <summary>
    /// The current GameObject raycasted to; guaranteed to be an input or output.
    /// </summary>
    private GameObject currentHitObject;

    /// <summary>
    /// The current index of the empty input or output that the user is hovered on.
    /// </summary>
    private int labelIndex;

    /// <summary>
    /// The global grid height that all raycasts are cast on.
    /// </summary>
    private Plane coordinatesPlane;

    /// <summary>
    /// Default text utilized when the user is not hovered onto an empty input or output.
    /// </summary>
    private readonly string defaultHoverText = "hover over and select inputs/outputs to view their order & label";

    // Enforces a singleton state pattern and establishes the grid plane.
    private void Awake()
    {
        if (instance != null)
        {
            Destroy(this);
            throw new Exception("BehaviorManagerPreview instance already established; terminating.");
        }

        instance = this;
        coordinatesPlane = new Plane(Vector3.down, GridMaintenance.Instance.GridHeight);
    }

    private void Start() { labelText.text = defaultHoverText; }

    private void Update()
    {
        // If hovered onto UI, reset to default values
        if (EventSystem.current.IsPointerOverGameObject()) { isUILocked = true; State(null); return; }

        isUILocked = false; // Otherwise, game is not paused.

        bool raycastHit = Physics.Raycast(CameraMovementPreview.Instance.PlayerCamera.ScreenPointToRay(Input.mousePosition), out RaycastHit hitInfo);

```

```csharp
        // If raycast invalid, reset to default values
        if (!raycastHit) { State(null); return; }

        // If raycast GameObject is not an input or output, reset to default values
        if (hitInfo.transform.gameObject.layer != 9 && hitInfo.transform.gameObject.layer != 10) { State(null); return; }

        State(hitInfo.transform.gameObject);
    }

    /// <summary>
    /// Exhibits different text states based on hit object properties.<br/><br/>
    /// This text is then written to <seealso cref="labelText"/>.
    /// </summary>
    /// <param name="hitObject"></param>
    private void State(GameObject hitObject)
    {
        // Already completed calculations for the same hit object.
        if (currentHitObject == hitObject) return;

        // Otherwise, restore previoous hit object to default values.
        if (currentHitObject != null) currentHitObject.GetComponent<MeshRenderer>().material = currentHitObject.layer == 9 ? PreviewManager.Instance.InputMaterial : PreviewManager.Instance.OutputMaterial;

        // UpdateLabelIndex(hitObject) != -1 implies it is an empty input or outrput.
        if (hitObject != null && UpdateLabelIndex(hitObject) != -1)
        {
            // Obtains the label text
            string newLabelText = isInput ? MenuLogicManager.Instance.CurrentPreviewStructure.InputLabels[labelIndex] : MenuLogicManager.Instance.CurrentPreviewStructure.OutputLabels[labelIndex];

            // Sets text values
            hitObject.GetComponent<MeshRenderer>().material = selectedMaterial;
            labelText.text = (isInput ? "input" : "output") + " #" + (labelIndex + 1) + (newLabelText != "" ? " - " + newLabelText : "");
            currentHitObject = hitObject;
        }

        // Is null and/or invalid input/output; restore default values.
        else
        {
            labelText.text = defaultHoverText;

            if (hitObject == null) currentHitObject = null;
        }
    }

    /// <summary>
    /// Obtains the current mouse to world position.
    /// </summary>
    /// <returns>The current world position.</returns>
    public Vector3 UpdateCoordinates()
    {
        Ray raycastRay = CameraMovementPreview.Instance.PlayerCamera.ScreenPointToRay(Input.mousePosition);

        if (coordinatesPlane.Raycast(raycastRay, out float distance))
        {
            Vector3 point = raycastRay.GetPoint(distance);

            // If the UI is not locked, also update the coordinates UI text.
            if (!isUILocked) coordinatesText.text = "(" + point.x.ToString("0.0") + ", " + point.z.ToString("0.0") + ")";

            return new Vector3(point.x, GridMaintenance.Instance.GridHeight, point.z);
        }

        throw new Exception("Unable to obtain new mouse position -- raycast failed.");
    }

    /// <summary>
    /// Obtains the index of the empty input or output belonging to the given hit object.
    /// </summary>
    /// <param name="hitObject">The raycasted game object.</param>
    /// <returns>The index of the empty input or output.</returns>
    private int UpdateLabelIndex(GameObject hitObject)
    {
        if (hitObject.layer == 9)
        {
            isInput = true;
            labelIndex =
                MenuLogicManager.Instance.CurrentPreviewStructure.InputOrders[PreviewManager.Instance.Inputs.IndexOf(hitObject.GetComponent<CircuitVisualizer.InputReference>().Input)];
        }

        else
        {
            isInput = false;
            labelIndex = MenuLogicManager.Instance.CurrentPreviewStructure.OutputOrders[PreviewManager.Instance.Outputs.IndexOf(hitObject.GetComponent<CircuitVisualizer.OutputReference>().Output)];
        }

        return labelIndex;
    }

    // Getter methods
    public static BehaviorManagerPreview Instance { get { return instance; } }

    public bool IsUILocked { get { return isUILocked; } }
}
```