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

/// <summary>
/// Coordinates keeps track of the world position as well as the grid snapping mode.
/// </summary>
public class Coordinates : MonoBehaviour
{
    // Singleton state reference
    private static Coordinates instance;

    /// <summary>
    /// Dictates how the current world position within the editor scene should be interpreted.<br/>
    /// This modified position is utilized for several actions within the scene, such as placing wires and moving circuits.<br/><br/>
    /// <seealso cref="GRID"/>: snap current mouse position to the visual grid.<br/>
    /// <seealso cref="NONE"/>: keep the current mouse position as is.
    /// </summary>
    public enum SnappingMode { GRID, NONE }

    /// <summary>
    /// The transparency value of <seealso cref="gridStatus"/> when <seealso cref="SnappingMode.NONE"/> is enabled.
    /// </summary>
    [SerializeField] float gridTransparencyConstant;

    /// <summary>
    /// In-scene icon that visualizes the status of <seealso cref="snappingMode"/>.
    /// </summary>
    [SerializeField] Image gridStatus;

    /// <summary>
    /// Toggles the <seealso cref="SnappingMode"/> currently not in use.
    /// </summary>
    [SerializeField] KeyCode snapToggleKey;

    /// <summary>
    /// Displays the current world coordinates to the user.
    /// </summary>
    [SerializeField] TextMeshProUGUI coordinateText;

    /// <summary>
    /// Stores the inspector-assigned color of <seealso cref="gridStatus"/
```

```
>.
    /// </summary>
    private Color gridStatusColor;

    /// <summary>
    /// Utilized to perform a raycast to calculate <seealso cref="mousePos"/>.
    /// </summary>
    private Plane raycastPlane;

    /// <summary>
    /// The current <seealso cref="SnappingMode"/>.
    /// </summary>
    private SnappingMode snappingMode;

    /// <summary>
    /// Stores the calculated mouse to world position.
    /// </summary>
    private Vector3 mousePos;

    private void Update()
    {
        // If the snap toggle key is pressed at a valid time, switch states.
        if (Input.GetKeyDown(snapToggleKey) && BehaviorManager.Instance.CurrentStateType != BehaviorManager.StateType.PAUSED)
        {
            snappingMode = snappingMode == SnappingMode.GRID ? SnappingMode.NONE : SnappingMode.GRID;
            CurrentSnappingMode = snappingMode; // Ensures the UI is also updated.
        }
    }

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

        instance = this;

        // Initializes private values
        raycastPlane = new Plane(Vector3.down,
              GridMaintenance.Instance.GridHeight);
        gridStatusColor = gridStatus.color;
    }

    /// <summary>
    /// Snaps the specified position to the grid.
    /// </summary>
    /// <param name="normalPos">The position that should be snapped to the grid.</param>
    /// <returns>The grid position.</returns>
    public static Vector3 NormalToGridPos(Vector3 normalPos) { return new Vector3((int)(normalPos.x + 0.5f * Mathf.Sign(normalPos.x)), GridMaintenance.Instance.GridHeight, (int)(normalPos.z + 0.5f * Mathf.Sign(normalPos.z))); }

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

    /// <summary>
    /// Returns a new ray from the camera to the current mouse position.
    /// </summary>
    private Ray CameraRay { get { return CameraMovement.Instance.PlayerCamera.ScreenPointToRay(Input.mousePosition); } }

    /// <summary>
    /// Calculates and returns the current grid position.
    /// </summary>
    public Vector3 GridPos { get { return NormalToGridPos(mousePos); } }

    /// <summary>
    /// Calculates and returns the current mouse position.
    /// </summary>
    public Vector3 MousePos
    {
        get
        {
            Ray ray = CameraRay;

            if (raycastPlane.Raycast(ray, out float distance))
            {
                mousePos = ray.GetPoint(distance);

                // Updates the coordinates UI if the game is not currently paused
                if (BehaviorManager.Instance.CurrentStateType != BehaviorManager.StateType.PAUSED) coordinateText.text = "(" + mousePos.x.ToString("0.0") + ", " + mousePos.z.ToString("0.0") + ")";
```

```

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

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

        /// <summary>
        /// Returns a modified version of <seealso cref="mousePos"/> based on
          <seealso cref="snappingMode"/>.
        /// </summary>
        public Vector3 ModePos { get { return snappingMode ==
          SnappingMode.GRID ? GridPos : MousePos; } }

        /// <summary>
        /// Serves as a getter method as well as a setter method for both
          <seealso cref="snappingMode"/> and <seealso cref="gridStatus"/>.
        /// </summary>
        public SnappingMode CurrentSnappingMode { get { return snappingMode; }
            set
            {
                snappingMode = value;

                if (value == SnappingMode.GRID)
                {
                    gridStatus.color = gridStatusColor;
                }

                else
                {
                    Color temp = gridStatusColor;

                    temp.a = gridTransparencyConstant;
                    gridStatus.color = temp;
                }
            }
        }
    }
```