```csharp
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// Logical representation of an NAND (NOT AND) gate.<br/><br/>
/// The NAND gate is also a universal gate.
/// </summary>
public class NAndGate : Circuit
{
    public NAndGate() : this(Vector2.zero) { }

    public NAndGate(Vector2 startingPos) : base("NAND", 2, 1, startingPos) { }

    /// <summary>
    /// Returns an output to update if the output has changed due to alterations in input power statuses.
    /// </summary>
    /// <returns>The list of outputs that should have their connections called.</returns>
    protected override List<Output> UpdateOutputs()
    {
        bool outputStatus = Outputs[0].Powered;
        List<Output> outputs = new List<Output>();

        // NAND gate representation
        Outputs[0].Powered = !(Inputs[0].Powered && Inputs[1].Powered);

        if (outputStatus != Outputs[0].Powered || MaterialNotMatching()) outputs.Add(Outputs[0]);

        return outputs;
    }

    /// <summary>
    /// Checks all outputs to determine if the output node material is not matching its power status.<br/><br/>
    /// This is utilized within custom circuits to force update calls that would normally not occur due to the nature of UpdateOutputs().
    /// </summary>
    /// <returns>Whether any output material does not match its power status.</returns>
    private bool MaterialNotMatching()
    {
        if (Outputs[0].StatusRenderer == null) return false;

        return (Outputs[0].Powered && Outputs[0].StatusRenderer.sharedMaterial != CircuitVisualizer.Instance.PowerOnMaterial) ||
               (!Outputs[0].Powered && Outputs
                 [0].StatusRenderer.sharedMaterial !=
                 CircuitVisualizer.Instance.PowerOffMaterial);
    }
}
```