# **Nav Link Proxy:**

# **Troubleshooting and Best Practices**

In my recent work with the **Nav Link Proxy**, I have encountered and fixed several issues related to AI navigation, such as:

* **AI Move To** node returning “Aborted”
* AI falling off the Nav Mesh
* Target Actor vs Target Vertex distinction
* Issues with **GetRandomPointInRadius** and **StopMovement**
* Problems with **Nav Proxy** not firing, misbehaving, or issues with transforming, rotating, and scaling
* Data monitoring inconsistencies

I aim to contribute this knowledge to the community to aid in understanding and improving **Nav Link Proxy** behavior. While I don’t have complete insight into the underlying systems, I’ve developed methods to gain control over **Nav Proxy** functionality to achieve the desired outcomes.

## **Initial Setup Process:**

1. **Drop a new Nav Link:**
   1. Place a new **Nav Link** in the level.
2. **Smart Link Adjustments:**
   1. Drag the two Smart Link polygons to the desired locations.
   2. In the **Details > Smart Link** area, press the **Copy End Points** button at the top.
   3. Slightly move the Smart Link in the viewport to perfectly align the four floating cylinders.
   4. Enable **Smart Link Is Relevant** in the **Details** section.
3. **Agent Settings:**
   1. Navigate to **Edit > Project Settings > Navigation System > Agents > Supported Agents** and create a custom agent name.
   2. Copy this custom agent name and paste it into **Default Agent Name** in the **Navigation System** settings.
4. **Associate Smart Link with Agent:**
   1. Select the Smart Link, go to **Details > Supported Agents**, and click the custom name created in the project settings.
5. **Smart Link Blueprint:**
   1. In the Smart Link's **Details** panel, click the **Blueprint** button at the top right.
   2. Inside the Blueprint, right-click and search for **EventReceiveSmartLinkReached**.
   3. This event will trigger when AI attempts to use the Nav Link over a gap in the Nav Mesh.
6. **Reusable Blueprint Logic:**
   1. Use this Blueprint to define your Smart Link's logic.
   2. Once programmed, ensure all Smart Links use this Blueprint, allowing you to easily drag-and-drop the Blueprint from your Content folder into any level.

## **Troubleshooting Common Issues:**

* **Simple Link Settings:**
  + Modify the following settings in **Details > Simple Link > Point Links**:
    - **Left Project Height**: Set to 1000
    - **Snap Height**: Enable and set to 50
    - **Snap Radius**: Set to 30 to prevent link failures when characters slide past.
* **Validating Nav Proxies:**
  + To check if a Proxy connects two pieces of green Nav Mesh, press P in the editor or press ' in-game to reveal debug mode. Then press 0 to highlight valid Nav Proxies.
  + Trust the arrows you see in-game, as the editor’s arrows may not always be valid.
* **AI Behavior with Failed Nav Proxies:**
  + If the Nav Proxy fails, the AI will run to the nearest location against the Nav Mesh's boundary. If out of bounds, the AI may incorrectly run toward the player.
  + This can be mitigated by ensuring Nav Proxy Smart Links are placed close to the borders of connected Nav Mesh areas.

## **AI and Nav Proxy Fixes:**

1. **Sliding AI:**
   1. AI physics and movement often interfere with Nav Proxy functionality, especially at speeds above 150. Use variables to control and monitor the AI’s movement near the Nav Proxy.
2. **Proximity and Jump Calculations:**
   1. Ensure Nav Proxy Smart Links are placed close to the border of the green Nav Mesh. AI will attempt to walk towards the straight green lines connecting disconnected Nav Mesh islands.
3. **Reliable Blueprint Variables:**
   1. Inside the Nav Proxy Blueprint, the **Agent** and **Destination** variables within the **EventReceiveSmartLinkReached** event are reliable, but the AI may fire a Nav Proxy every tick instead of just once. This can create errors, particularly when using multiple Nav Proxies close together. Ensure your logic accounts for this.

## **Additional Fixes for AI Move To Node:**

1. **Optimizing Nav Mesh Settings:**
   1. Edit the **Project Settings > Navigation Mesh** for better performance:
      1. **Tile Size UU**: 1500-3000
      2. **Cell Size**: 10
   2. These adjustments can reduce AI recalculating their path unnecessarily.
2. **Navigation Generation:**
   1. Set **Runtime Generation** to **Static** while debugging. **Dynamic Nav Mesh** tends to “re-think” frequently, leading to navigation issues.
3. **Scaling and Rotation Caution:**
   1. Avoid scaling or rotating any Navigation-related elements, as this can introduce calculation errors that are difficult to diagnose.
4. **Handling AI Move To Failures:**
   1. If **AI Move To** fails, try looping multiple **AI Move To** nodes using various methods:
      1. **Destination Actor**
      2. **Destination Vector**
      3. **GetRandomPointInNavigableRadius**
5. **Branch Logic:**
   1. Use conditional logic to handle **AI Move To** failures:
      1. **AI Move > Movement Result > Branch** – If **Aborted**, handle it with appropriate logic.

## **Nav Proxy Event Programming Tips:**

1. **Pause AI Movement:**
   1. Set up a 0.5-second pause at the beginning and end of a Nav Proxy transition to reduce errors during the handover from Nav Mesh to Nav Link navigation.
2. **Jump Logic:**
   1. Program the AI to jump based on the Z-axis difference between the AI and the Nav Proxy, scaling this logic using the **SmartLinkReached** event data.
   2. Use the Z-axis and distance to determine whether the AI should jump or fall, depending on the angle and proximity of the Nav Proxy.

By following these steps and employing these best practices, you can create a robust and reliable Nav Link Proxy system that works seamlessly in your game environment.