Interaktions-Guide für Nodges

Dieses Dokument beschreibt die technische Umsetzung und Funktionsweise der Interaktionsmöglichkeiten in Nodges: Hovering, Selektion, Pfadsuche und allgemeine Suchfunktionen.

1. Hover-Interaktion (Mouse Over)

Die Hover-Funktionalität ermöglicht es dem Nutzer, Informationen über Knoten (Nodes) und Kanten (Edges) zu erhalten, indem er den Mauszeiger darüber bewegt.

Ablauf

1. **Raycasting**: Der `RaycastManager` (in `src/utils/RaycastManager.ts`) führt kontinuierlich (gedrosselt) Raycasts durch, um Schnittpunkte zwischen dem Mauszeiger und 3D-Objekten zu finden.
    - **Kanten**: Werden als `THREE.Mesh` mit `userData.type === 'edge'` erkannt. Für gekrümmte Kanten wird die Kurvengeometrie berücksichtigt.
    - **Knoten**: Werden als Instanzen eines `THREE.InstancedMesh` erkannt (`userData.type === 'node_instanced'`). Der Manager extrahiert die ID der Instanz.
2. **Status-Update**: Wenn ein Objekt gefunden wird, informiert der `RaycastManager` den `StateManager`.
    - `StateManager.setHoveredObject(object)` wird aufgerufen.
3. **Visuelles Feedback**: Der `HighlightManager` (`src/effects/HighlightManager.ts`) reagiert auf Statusänderungen.
    - **Knoten**: `applyNodeHoverHighlight` wird ausgeführt. Es wird typischerweise ein "Glow"-Effekt oder eine Farbänderung angewendet.
    - **Kanten**: `applyEdgeHoverHighlight` wird ausgeführt. Die Kante wird oft in ihrer Originalfarbe oder einer Highlight-Farbe hervorgehoben und ggf. verdickt.
4. **UI-Info**: Der `UIManager` zeigt das `HoverInfoPanel` (definiert in `src/styles/main.css` als `#hoverInfoPanel`) an. Der `CentralEventManager` füllt dieses Panel mit Daten aus `object.userData`.

Wichtige Dateien

- `src/utils/RaycastManager.ts`
- `src/effects/HighlightManager.ts`
- `src/core/StateManager.ts`
- `src/ui/HoverInfoPanel.ts` (bzw. Logik im UIManager)

2. Selektion (Auswahl)

Nodges unterstützt sowohl Einzel- als auch Mehrfachauswahl (Box-Selection).

Einzelauswahl

- **Trigger**: Klick (Maus-Down/Up innerhalb kurzer Zeit) auf ein Objekt.
- **Verarbeitung**: `InteractionManager` oder `SelectionManager` erkennt den Klick.
- **Logik**: `SelectionManager.setSingleSelection(object)` wird aufgerufen.
- **Status**: Das Objekt wird im `SelectionManager.selectedObjects` Set gespeichert und im `StateManager` als `selectedObject` gesetzt.

- **Visualisierung**:
  - Der `HighlightManager` wendet `applySelectionEffect` an (oft ein permanenteres Highlight als beim Hover).
  - Bei Knoten werden oft auch verbundene Kanten hervorgehoben (`NeighborhoodHighlighter`).

Mehrfachauswahl (Box Selection)

- **Trigger**: Ziehen der Maus mit gedrückter Taste (oft Shift oder Ctrl + Drag) oder in einem speziellen Auswahlmodus.
- **Logik**:
  - `SelectionManager.performBoxSelection` berechnet ein 2D-Rechteck auf dem Bildschirm.
  - Es werden alle Objekte ermittelt, deren Bildschirmposition innerhalb dieses Rechtecks liegt.
- **Ergebnis**: Alle gefundenen Objekte werden zur Auswahl hinzugefügt (`addToSelection`).

Wichtige Dateien

- `src/utils/SelectionManager.ts`
- `src/core/InteractionManager.ts`

3. Pfadsuche (Path Finding)

Die Pfadsuche ermittelt Verbindungen zwischen zwei Knoten im Netzwerk.

Funktionsweise

- **Algorithmen**: Implementiert in `src/utils/PathFinder.ts`.
  - **BFS (Breitensuche)**: `findShortestPath`. Findet den Pfad mit den wenigsten Hops (Kanten).
  - *A (A-Stern)**: `findAStarPath`. Findet den kürzesten Pfad basierend auf räumlicher Distanz (euklidisch) als Heuristik.
- **Start & Ziel**: Werden über die API (`setStartNode`, `setEndNode`) oder UI-Interaktion festgelegt.
- **Visualisierung**:
  - `createPathVisualization`: Erstellt `THREE.TubeGeometry` oder Linien entlang des gefundenen Pfades.
  - `startPathAnimation`: Lässt ein Objekt (z.B. eine Kugel) entlang des Pfades wandern (`animatePathTraversal`), um den Fluss zu visualisieren.

Wichtige Dateien

- `src/utils/PathFinder.ts`

4. Suche & Filterung (Neighborhood Search)

Diese Funktion dient dazu, spezifische Knoten zu finden oder die lokale Umgebung eines Knotens zu erkunden.

Funktionsweise

- **Nachbarschaftssuche**:
  - Klasse: `NeighborhoodHighlighter` (`src/utils/NeighborhoodHighlighter.ts`).
  - Methode: `findNeighborhood(centerNode, hops)`.

    - Ablauf: Ausgehend von einem "Zentral-Knoten" werden rekursiv alle verbundenen Knoten und Kanten bis zu einer gewissen Tiefe (`hops`) gesammelt.
- **Highlighting ("Search Effect")**:
    - `HighlightManager.applySearchEffect`: Hebt gefundene Objekte hervor.
    - `NeighborhoodHighlighter.dimOthersInScene`: Dimmt alle *nicht* beteiligten Objekte ab (reduziert Opazität), um den Fokus auf das Suchergebnis zu lenken.
- **Text-Suche (UI)**:
    - Wird typischerweise über Eingabefelder im `VisualMappingPanel` oder `EnvironmentPanel` gesteuert.
    - Filtert Knoten basierend auf Metadaten (Name, ID, Typ).
    - Gefundene Knoten werden an den `NeighborhoodHighlighter` oder `SelectionManager` übergeben.

Wichtige Dateien

- `src/utils/NeighborhoodHighlighter.ts`
- `src/effects/HighlightManager.ts`
- `src/ui/VisualMappingPanel.ts` (UI-Eingabe)