Nodges - Technische Dokumentation

1. Einleitung

Nodges ist eine webbasierte 3D-Netzwerkvisualisierungsanwendung, die mit `three.js` entwickelt wurde. Sie ermöglicht das Laden, Anzeigen und Interagieren mit komplexen Graphen und Netzwerken. Die Anwendung zeichnet sich durch eine modulare Architektur aus, die auf spezialisierten Managern basiert, um verschiedene Aspekte wie Layout, Interaktion, Zustandsverwaltung und Darstellung zu handhaben.

Das Kernkonzept der Anwendung ist die Entkopplung der einzelnen Verantwortlichkeiten in separate Module, die über ein zentrales Event-System und einen State-Manager miteinander kommunizieren. Dies ermöglicht eine hohe Flexibilität und Erweiterbarkeit.

2. Architekturübersicht

Die folgende Grafik zeigt die Hauptkomponenten der Nodges-Anwendung und ihre Beziehungen zueinander.

```
graph TD
    subgraph User Interface
        A[index.html]
        B[UIManager]
        C[LayoutGUI]
    end

    subgraph Core Logic
        D[main.js / NodgesApp]
        E[StateManager]
        F[CentralEventManager]
        G[InteractionManager]
        H[LayoutManager]
    end

    subgraph Rendering & Objects
        I[three.js Scene]
        J[Node Objects]
        K[Edge Objects]
        L[HighlightManager]
        M[NodeMeshFactory]
    end

    subgraph Data & Workers
        N[JSON Data]
        O[FutureDataParser]
        P[layout-worker.js]
    end

    A --> D
    D --> E
```

```
    D --> F
    D --> G
    D --> H
    D --> B
    D --> C
    D --> L

    F --> G
    G --> E
    G --> L
    G --> B

    E --> B
    E --> L

    H --> P
    P --> H
    H --> J
    H --> K

    D -- loads --> N
    N -- parsed by --> O
    O --> D

    D -- creates --> J
    D -- creates --> K
    J -- created by --> M
    J -- added to --> I
    K -- added to --> I
    L -- affects --> J
    L -- affects --> K

    B -- manipulates --> A
    C -- controls --> H
```

3. Datenfluss

Das Laden und Verarbeiten der Netzwerkdaten folgt einem klaren Prozess, der im folgenden Diagramm dargestellt ist.

```
sequenceDiagram
    participant User
    participant NodgesApp
    participant FileHandler
    participant FutureDataParser
    participant LayoutManager
    participant Scene

    User->>NodgesApp: Klickt auf "Load Data"
    NodgesApp->>FileHandler: loadData(url)
    FileHandler-->>NodgesApp: Gibt JSON-Daten zurück
```

```
    alt Daten im Zukunftsformat
        NodgesApp->>FutureDataParser: parseData(data)
        FutureDataParser-->>NodgesApp: Gibt geparste Knoten & Kanten zurück
    end
    NodgesApp->>NodgesApp: clearScene()
    NodgesApp->>NodgesApp: createNodes()
    NodgesApp->>NodgesApp: createEdges()
    NodgesApp->>LayoutManager: applyLayout()
    LayoutManager-->>NodgesApp: Aktualisierte Knotenpositionen
    NodgesApp->>Scene: Fügt Knoten- und Kanten-Meshes hinzu
```

4. Ereignisbehandlung (Interaktionsfluss)

Benutzerinteraktionen werden durch ein zentrales System verarbeitet, das Low-Level-DOM-Ereignisse in anwendungsspezifische Aktionen umwandelt.

```
sequenceDiagram
    participant User
    participant DOM
    participant CentralEventManager
    participant InteractionManager
    participant StateManager
    participant HighlightManager
    participant UIManager

    User->>DOM: Führt Mausbewegung aus
    DOM->>CentralEventManager: handleMouseMove(event)
    CentralEventManager->>CentralEventManager: Raycast auf 3D-Szene
    CentralEventManager->>InteractionManager: publish('hover_start',
{object})
    InteractionManager->>HighlightManager: highlightHoveredObject(object)
    InteractionManager->>StateManager: setHoveredObject(object)
    StateManager->>UIManager: notifySubscribers()
    UIManager->>DOM: Aktualisiert Cursor & Tooltip

    User->>DOM: Klickt auf Objekt
    DOM->>CentralEventManager: handleClick(event)
    CentralEventManager->>InteractionManager: publish('click',
{clickedObject})
    InteractionManager->>StateManager: setSelectedObject(object)
    StateManager->>HighlightManager: notifySubscribers()
    HighlightManager->>HighlightManager: Wendet Glow-Effekt an
    StateManager->>UIManager: notifySubscribers()
    UIManager->>DOM: Zeigt Info-Panel mit Objektdetails an
```

5. Komponentenbeschreibung

5.1 `main.js` (NodgesApp)

Die `NodgesApp`-Klasse in `main.js` ist der zentrale Orchestrator der Anwendung. Sie ist verantwortlich für:

- **Initialisierung:** Einrichten der `three.js`-Szene, Kamera, Renderer und Lichter.
- **Manager-Instanziierung:** Erstellen von Instanzen aller Kern- und Hilfsmanager.
- **Datenladung:** Laden der initialen und vom Benutzer ausgewählten Netzwerkdaten.
- **Szenen-Management:** Erstellen, Aktualisieren und Entfernen von Knoten- und Kantenobjekten in der `three.js`-Szene.
- **Animations-Loop:** Steuerung der Haupt-Render-Schleife (`requestAnimationFrame`), die für die kontinuierliche Aktualisierung der Szene und der Animationen sorgt.

5.2 `StateManager.js`

Der `StateManager` ist das "Gehirn" der Anwendung. Er hält den gesamten Anwendungszustand in einem einzigen Objekt und implementiert ein Publisher/Subscriber-Muster, um andere Komponenten über Zustandsänderungen zu informieren.

- **Zentraler Zustand:** Verwaltet wichtige Informationen wie das aktuell ausgewählte (`selectedObject`) oder überfahrene (`hoveredObject`) Objekt.
- **Benachrichtigungen:** Andere Manager (z. B. `HighlightManager`, `UIManager`) abonnieren den `StateManager`, um auf Zustandsänderungen zu reagieren und die Benutzeroberfläche oder visuelle Effekte entsprechend zu aktualisieren.
- **Entkopplung:** Dient als zentrale Wahrheit und entkoppelt die Manager voneinander, da sie nicht direkt miteinander kommunizieren müssen.

5.3 `CentralEventManager.js`

Diese Komponente ist die primäre Schnittstelle für Benutzereingaben. Sie bündelt alle DOM-Event-Listener an einem Ort.

- **Ereigniserfassung:** Lauscht auf Low-Level-DOM-Ereignisse wie `mousemove`, `click`, `keydown` etc.
- **Raycasting:** Verwendet einen `RaycastManager`, um festzustellen, welche 3D-Objekte von Mausereignissen betroffen sind.
- **Ereignis-Abstraktion:** Wandelt die rohen DOM-Ereignisse in semantisch reichere, anwendungsspezifische Ereignisse um (z. B. `hover_start`, `selection_start`) und veröffentlicht diese für andere Komponenten.
- **Performance:** Nutzt Techniken wie Throttling, um die Anzahl der verarbeiteten Ereignisse zu begrenzen und die Leistung zu optimieren.

5.4 `InteractionManager.js`

Der `InteractionManager` ist der Abonnent der vom `CentralEventManager` veröffentlichten Ereignisse. Er interpretiert die Absicht des Benutzers und leitet die entsprechenden Aktionen ein.

- **Ereignis-Interpretation:** Verarbeitet High-Level-Ereignisse wie `hover_start` oder `click`.
- **Zustandsänderung:** Löst Zustandsänderungen im `StateManager` aus (z. B. durch Aufruf von `stateManager.setSelectedObject(obj)`).
- **Visuelles Feedback:** Arbeitet eng mit dem `HighlightManager` zusammen, um visuelles Feedback für Interaktionen (z. B. Hervorhebungen, Glüheffekte) zu steuern.
- **UI-Steuerung:** Verwaltet die Anzeige von UI-Elementen wie dem Info-Panel, basierend auf der Benutzerinteraktion.

5.5 `LayoutManager.js`

Der `LayoutManager` ist für die algorithmische Anordnung der Knoten im 3D-Raum zuständig.

- **Algorithmen-Vielfalt:** Stellt eine Sammlung verschiedener Layout-Algorithmen zur Verfügung (z. B. Force-Directed, Circular, Grid).
- **Web Worker:** Lagert rechenintensive Layout-Berechnungen in separate Web Worker aus, um den Haupt-Thread nicht zu blockieren und die UI reaktionsfähig zu halten.
- **Normalisierung:** Skaliert und zentriert die resultierenden Knotenpositionen, um sicherzustellen, dass das Netzwerk eine konsistente Größe hat und gut sichtbar ist.
- **GUI-Integration:** Wird von der `LayoutGUI` gesteuert, die es dem Benutzer ermöglicht, verschiedene Layouts und deren Parameter auszuwählen.