Beads Visualization Showcase

Status: Planned **Location:** `showcases/beads-viz/` (after HATS migration complete) **Dependencies:** psd3-selection (HATS), psd3-graph, psd3-simulation

Overview

Visualize beads issue dependencies as an interactive force-directed graph. Helps answer:

- "What's blocking what?"
- "Why can't I work on X?"
- "What should I close to unblock the most work?"

Data Source

Reads `.beads/issues.jsonl` - one JSON object per line:


```
{"id":"bd-a1b2","title":"Fix
bug","status":"open","priority":1,"dependencies":[{"dependsOnId":"bd-
c3d4","depType":"blocks"}]}
```


Visual Encoding

Nodes (Issues)

Attribute	Encoding
Status	Fill color: green=open, gray=closed, orange=blocked, red=P0
Priority	Size: P0 largest → P4 smallest
Title	Label (truncated)
ID	Tooltip

Edges (Dependencies)

Relationship	Encoding
A blocked by B	Arrow from A → B
Type: blocks	Solid line
Type: parent-child	Dashed line
Type: related	Dotted line (optional, maybe hide)

Layout

Force-directed with:

- Blockers pulled upward (dependency direction)
- Clusters by connectivity
- Closed issues can be filtered out or dimmed

Interactions

Core

- **Drag**: Reposition nodes
- **Zoom/Pan**: Navigate large graphs
- **Hover**: Highlight node + all dependencies (upstream blockers, downstream blocked)

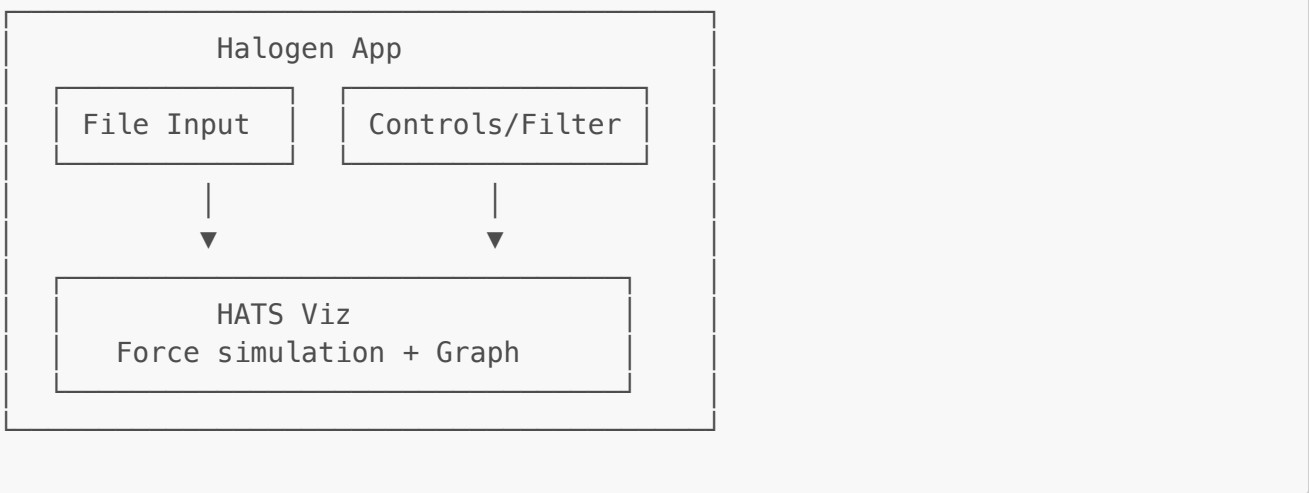
Filters

- Show/hide closed issues
- Filter by priority (P0-P1 only, etc.)
- Filter by label (if labels used)

Analysis

- **Critical path**: Highlight longest dependency chain
- **Bottlenecks**: Size nodes by "how many issues does closing this unblock?"
- **Ready view**: Only show open issues with no open blockers (matches `bd ready`)

Architecture



Modules

```
src/
├── Main.purs              -- Entry point
├── Component/
│   ├── App.purs           -- Main Halogen component
│   ├── Graph.purs         -- HATS force-directed graph
```

```
│   ├── Controls.purs      -- Filter controls
│   └── FileLoader.purs    -- JSONL file input
├── Data/
│   ├── Issue.purs         -- Issue type (can import from beads-purs or
redefine)
│   └── Graph.purs         -- Convert issues to graph structure
└── Viz/
    └── IssueGraph.purs    -- HATS tree builder for issue graph
```

Graph Conversion

```
type IssueNode =
  { id :: IssueId
  , title :: String
  , status :: Status
  , priority :: Int
  , blocked :: Boolean  -- has open blockers?
  }

type IssueEdge =
  { source :: IssueId
  , target :: IssueId  -- source depends on target
  , depType :: DependencyType
  }

issuesToGraph :: Array Issue -> { nodes :: Array IssueNode, edges :: Array
IssueEdge }
```

HATS Implementation

Uses tick-driven force simulation (like LesMis demo):

```
issueGraph :: forall r. Array IssueNode -> Array IssueEdge -> Simulation -
> HATSTree r
issueGraph nodes edges sim =
  H.container "g" [ H.class_ "issue-graph" ]
    [ H.container "g" [ H.class_ "edges" ] (map renderEdge edges)
    , H.container "g" [ H.class_ "nodes" ] (map renderNode nodes)
    ]
  where
  renderNode node =
    H.elem "circle"
      [ H.key node.id
      , H.attr "r" (priorityToRadius node.priority)
      , H.attr "fill" (statusToColor node.status node.blocked)
      , H.onSimulation sim node.id  -- position from simulation
      , H.onDrag sim node.id        -- draggable
      , H.onHighlight                -- hover highlighting
        { identify: node.id
```

```
        , related: \id -> isConnected id node.id edges
        }
      ]

renderEdge edge =
  H.elem "line"
    [ H.key (edge.source <> "-" <> edge.target)
    , H.class_ (depTypeToClass edge.depType)
    , H.onSimulationEdge sim edge.source edge.target
    ]
```

File Input Options

1. **File picker**: User selects `.beads/issues.jsonl` via `<input type="file">`
2. **URL param**: `?path=/path/to/.beads/issues.jsonl` (if running locally with file access)
3. **Paste**: Textarea to paste JSONL content directly
4. **Demo data**: Built-in sample for showcase

Stretch Goals

- **Live reload**: Watch file for changes (if Electron/Tauri wrapper)
- **Edit from graph**: Click node to open issue, right-click to close
- **Timeline view**: Animate issue creation/closure over time
- **Swimlanes**: Group by assignee or label
- **Export**: SVG/PNG export of current view

Implementation Order

1. **Static graph**: Load JSONL, render force simulation, basic styling
2. **Interactions**: Drag, zoom, hover highlighting
3. **Filters**: Status filter, priority filter
4. **Analysis**: Bottleneck sizing, critical path
5. **Polish**: Nice controls, responsive layout, demo data

Related

- `beads-purs` - CLI tool this visualizes
- `site/website/src/Component/Tour/LesMis*.purs` - Force simulation reference
- `psd3-graph` - Graph algorithms
- `psd3-simulation` - Force simulation bindings