Aesthetic Graph Drawing of Hierarchical State Diagrams

Guided Research

Maximilian Rudolf Hotter, B.Sc.

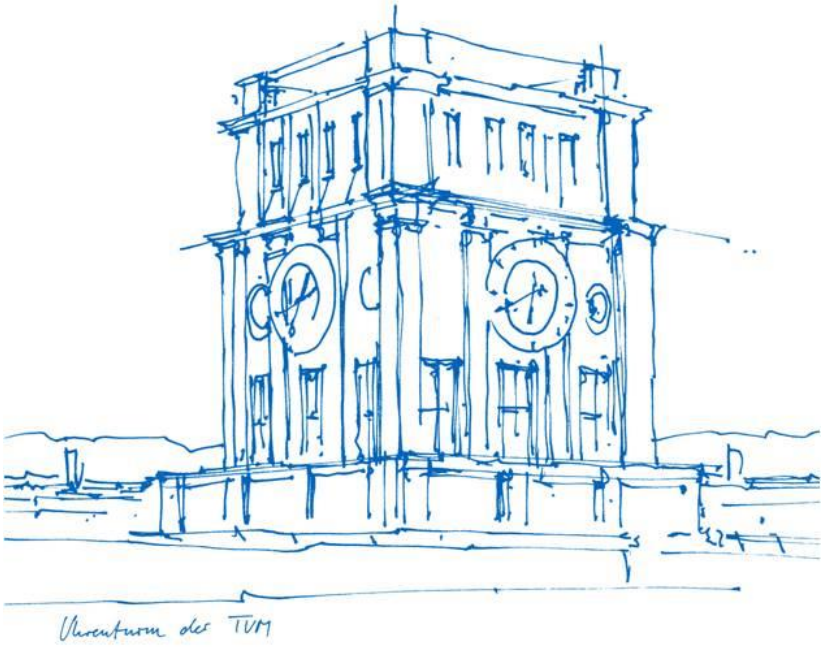
maximilian.hotter@tum.de

Advisor: Daniel Dyrda, M.Sc.

Supervisor: Prof. Gudrun Klinker, Ph.D.

Technische Universität München

Munich, April 30, 2020



1. Motivation

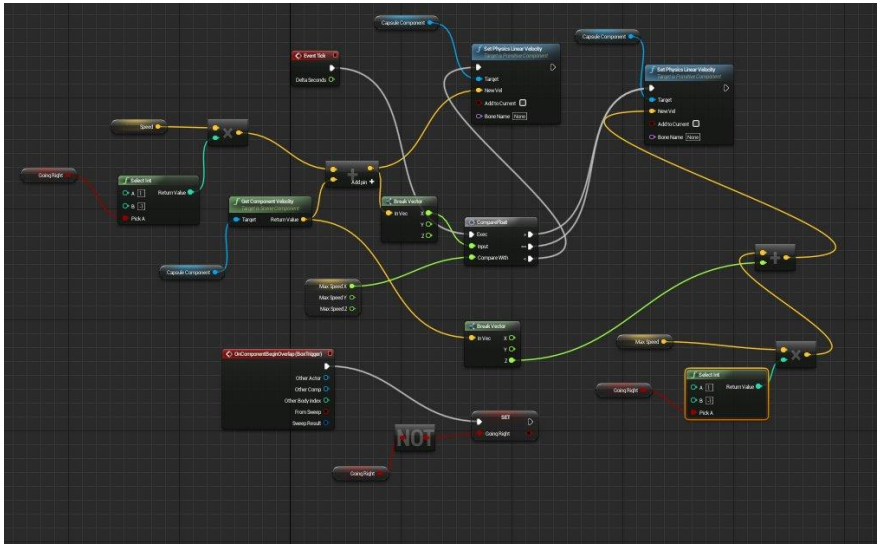
- State diagrams are widely used

→ Visualization is important

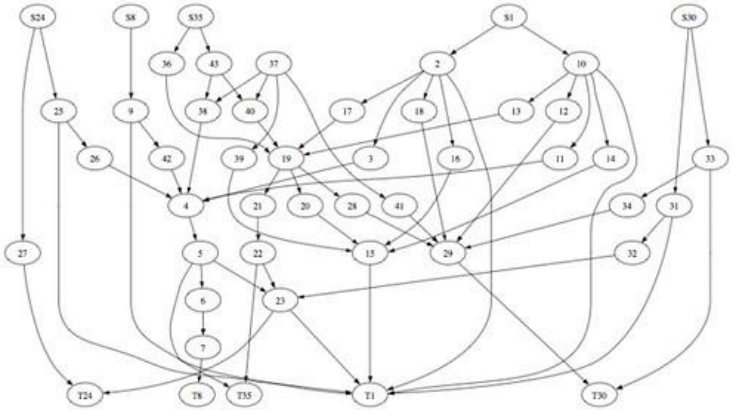
Existing solutions:

- Existing solutions like visual programming editors (Unreal or Unity Engine)
  - Mainy manual
  - Edges do not avoid collisions
- Graph visualization tools like Graphviz or PlantUML
  - Draw nodes and edges as they want
  - Can avoid collisions
  - No freedom for the user
  - Changes when adding nodes

→ Finding better solutions: **Graph Drawing**



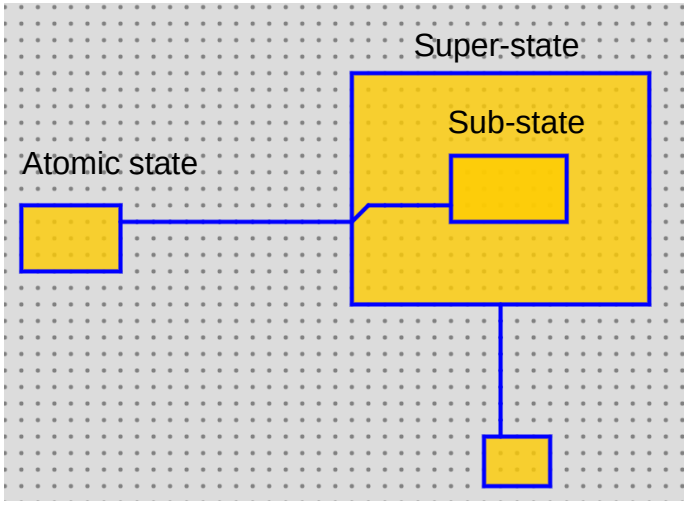
Source: https://davikingcode.com/blog/from-unity-2d-to-unreal-paper-2d/



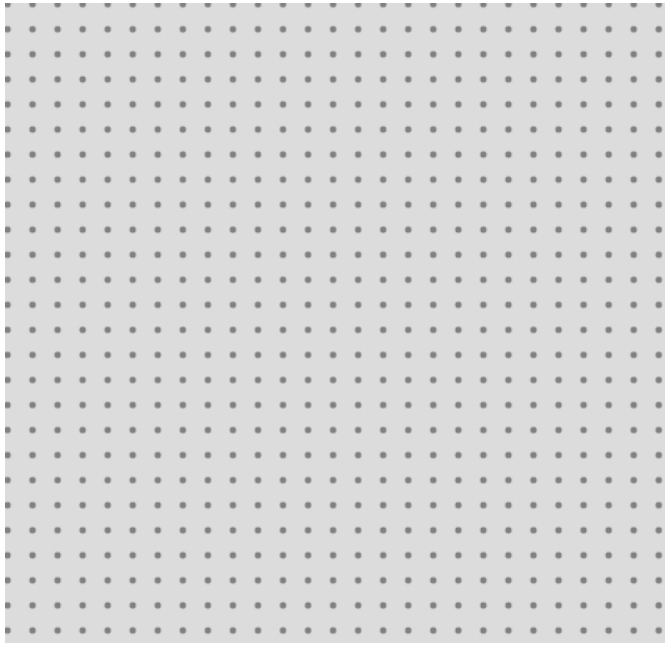
Source: Kent By, https://www.flickr.com/photos/kentbye/1156412902

2. Requirements
2.1. Hierarchical state diagram

- Planarity is **not guaranteed**!
  - The user can decide where to place his states
- **Location of states** (nodes) is fixed. The algorithm should not move them.
- **Hierarchical order** of states.
  - Super-states and sub-states
  - Edges must be able to **cross state borders**.



2. Requirements
2.2. Grid

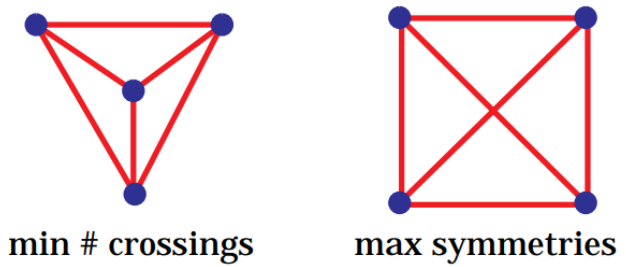


2. Requirements
2.3. Aesthetic Criteria (1)

- Small **number of crossings**
- Small **drawing area**
- Small total **edge length**
- Uniform edge length
- Reduce **number of straight edge segments**
- Small **number of edge bends**
- Large **angular resolution** (maximize smallest angle)
- As **symmetric** as possible

→ In general, it is not possible to simultaneously optimize all aesthetic criteria (e.g. number of crossings versus maximum symmetries)

→ Complexity issues: Mostly NP-hard in general.



Source: Isabel F. Cruz; Roberto Tamassia. How to Visualize a Graph: Specification and Algorithms. 1994

2. Requirements
2.3. Aesthetic Criteria (2)

1. Edges should **not come too close to state nodes**.
2. The **number of crossings and overlaps** should be as small as possible.
3. Edges should **not use unnecessary paths** (e.g. into and out of dead ends) and should not move too far from the direct path.
4. Edge lengths should not become **unnecessary long**.

3. Approach
3.1. Graph search algorithm

- **Breadth First search**
  - Explores equally in all directions
  - Expensive/No movement costs

- **Dijkstra‘s Algorithm** (Uniform Cost Search)
  - Prioritization which path to explore
  - Favors lower cost paths

- **A* Algorithm**
  - Modification of Dijkstra’s Algorithm
  - Optimized for a single destination
  - Different costs
  - Prioritizes paths that seem to be leading closer to a goal

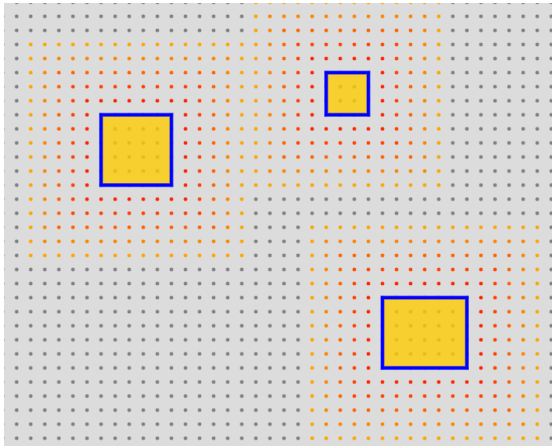
Source: Amet Patel, https://www.redblobgames.com/pathfinding/a-star/introduction.html

- ➢ We use A*
- ➢ **Criteria met: Shortest paths** and thus not **unnecessary long**

3. Approach
3.2. Distance Fields and Voronoi

- Avoid coming too close to other states

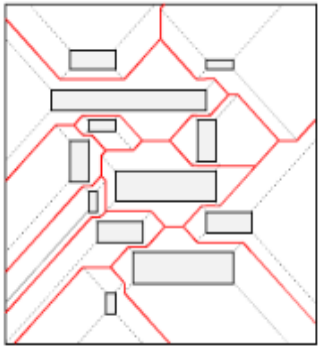
→ **Distance Fields**



- **Voronoi** could be also used

→ But distance fields behave similar to Voronoi when big enough

→ „Automatic“ path corridors in the center between nodes



Source: Papadopoulou, Evanthia; Lee, D. Critical Area Computation -- A New Approach. 1999.

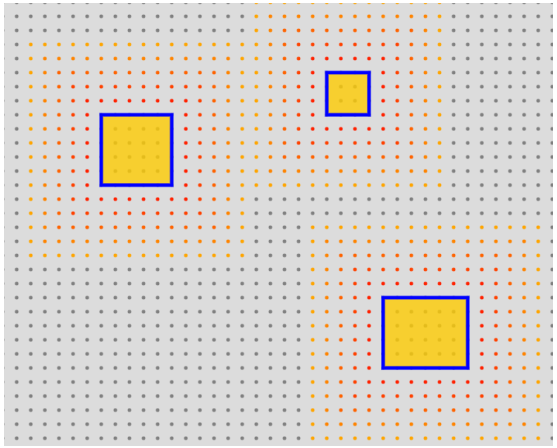
➢ Criterion met: Edges should **not come too close to state nodes**.

3. Approach
3.3. Different path costs

- Horizontal, vertical, and diagonal movement
  - Diagonal movement should have different **multiplier**
  - E.g. **Normal**: 1, **Diagonal**: ~1,414

Path costs

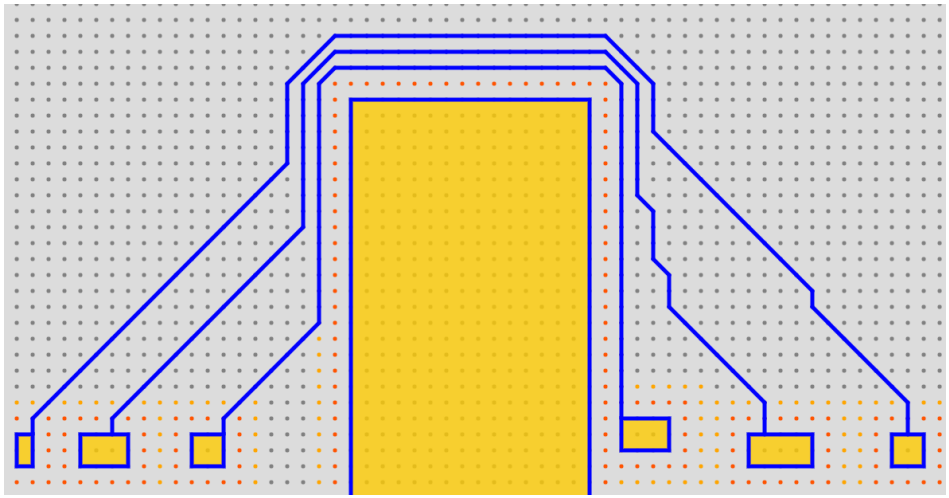
- High costs for **crossing state nodes**.
- Staggered costs for **crossing distance fields**, if they do not belong to target or start node.
- High costs for **crossing other edges**



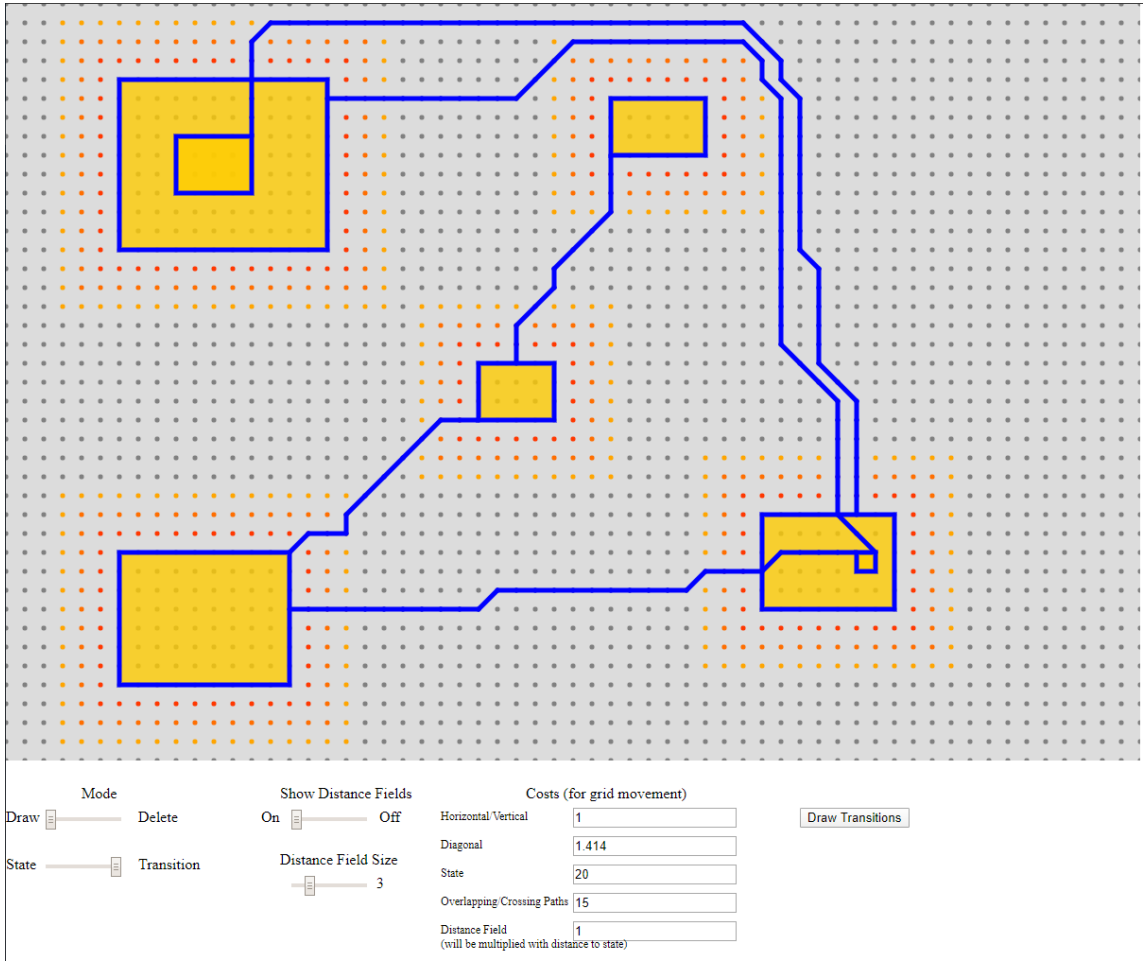
→ **Costs = Multiplier * Path costs**

3. Approach
3.4. Prioritization

- Edges that have lower path costs should be prioritized to avoid crossings.



4. Implementation



5. Results

- Distance Field: Edges do **not come too close to state nodes**.
- Different path costs: The **number of crossings and overlaps** should be as small as possible.
- A* algorithm**: Shortest paths** and thus not **unnecessary long**

6. Outlook

- Paths could be curved (using bezier curves or similar)
  - Avoiding crossings becomes more difficult

- Computed paths often bend in unwanted places. Could be more straightend afterwards.

- Use post-processing algorithms