IMPLEMENTIERUNG EINER PROLOG-BASIERTEN NAVIGATION FÜR DEN CAMPUS DER RUHR-UNIVERSITÄT

- **Wissensbasierte Methoden -**

Corinna Müller, Jan Kanowski, Jan Ulbrich, Moritz Sagurna

Inhalt

- Vorstellung des Projektes „Navigation in PROLOG“
- Anforderungen
- Arbeitspaket
  - Grundlagenbildung
  - Die Implementierung in JAVA
  - PROLOG
- Evaluierung
- Ausblick

Quelle: https://www.ruhr-uni-bochum.de/bilder/luftbilder/index.htm

RUHR UNIVERSITÄT BOCHUM RUB

Anforderungen (1)

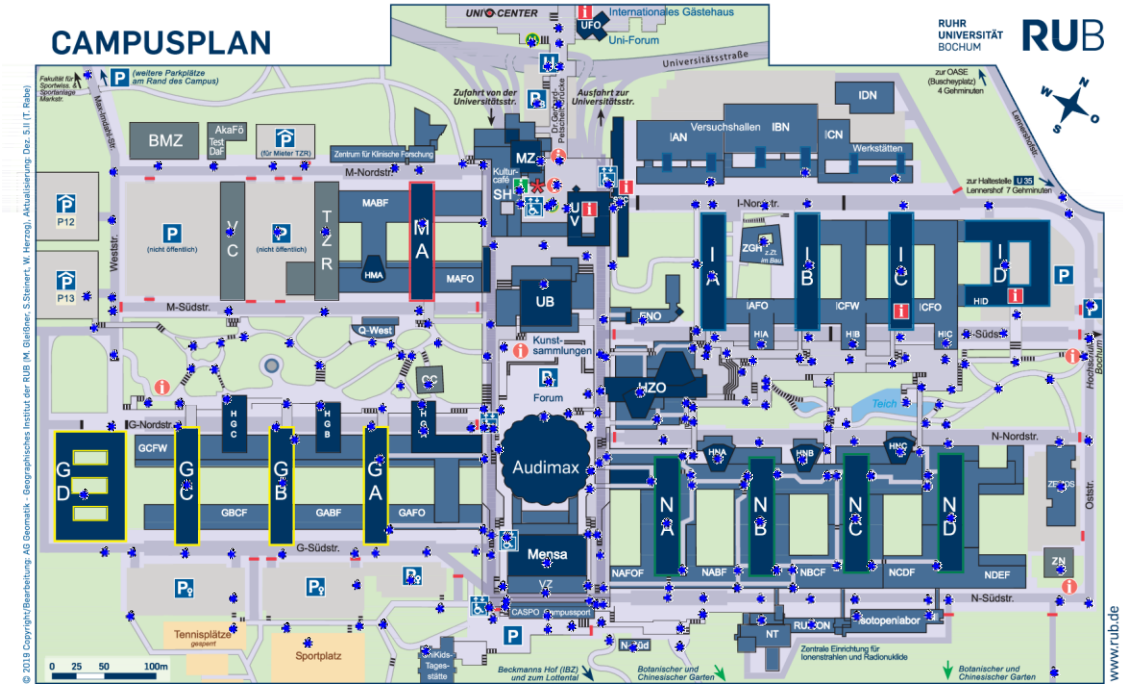
- Wegberechnung anhand von Start- und Zielpunktwahl
- Netzwerk aus definierten Wegen
- Punktauswahl anhand von Netzwerkkarte und blinder Auswahl
- Pfadermittlung auf dem Campusgelände mit zwei definierten Attributen
  - Ohne Beschränkung, mit Barrierefreiheit
- Modularer Aufbau der Software

Anforderungen (2)

- Pfadermittlungszeit <=5 Sekunden
- Visualisierung des Weg-Netzwerkes und gefundener Pfade
- Benutzerfreundlichkeit der GUI
- Implementierung als Java-Programm
- Abarbeitung der Suche durch PROLOG

Arbeitspaket: Grundlagenbildung

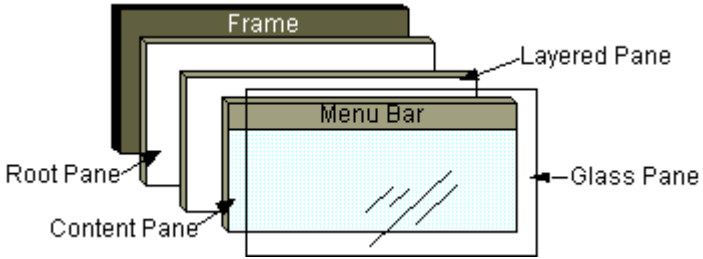
- Pfade erstellen:
  - Dreidimensionalität
  - Barrierefreiheit
- Drei Schritte:
  - Zeichnen der Knoten
  - MATLAB für Koordinaten
  - Kanten definieren (Excel als Schnittstelle)
- Viel Handarbeit
  - ➢ Anfällig für Fehler



RUHR UNIVERSITÄT BOCHUM RUB

Arbeitspaket: Java GUI

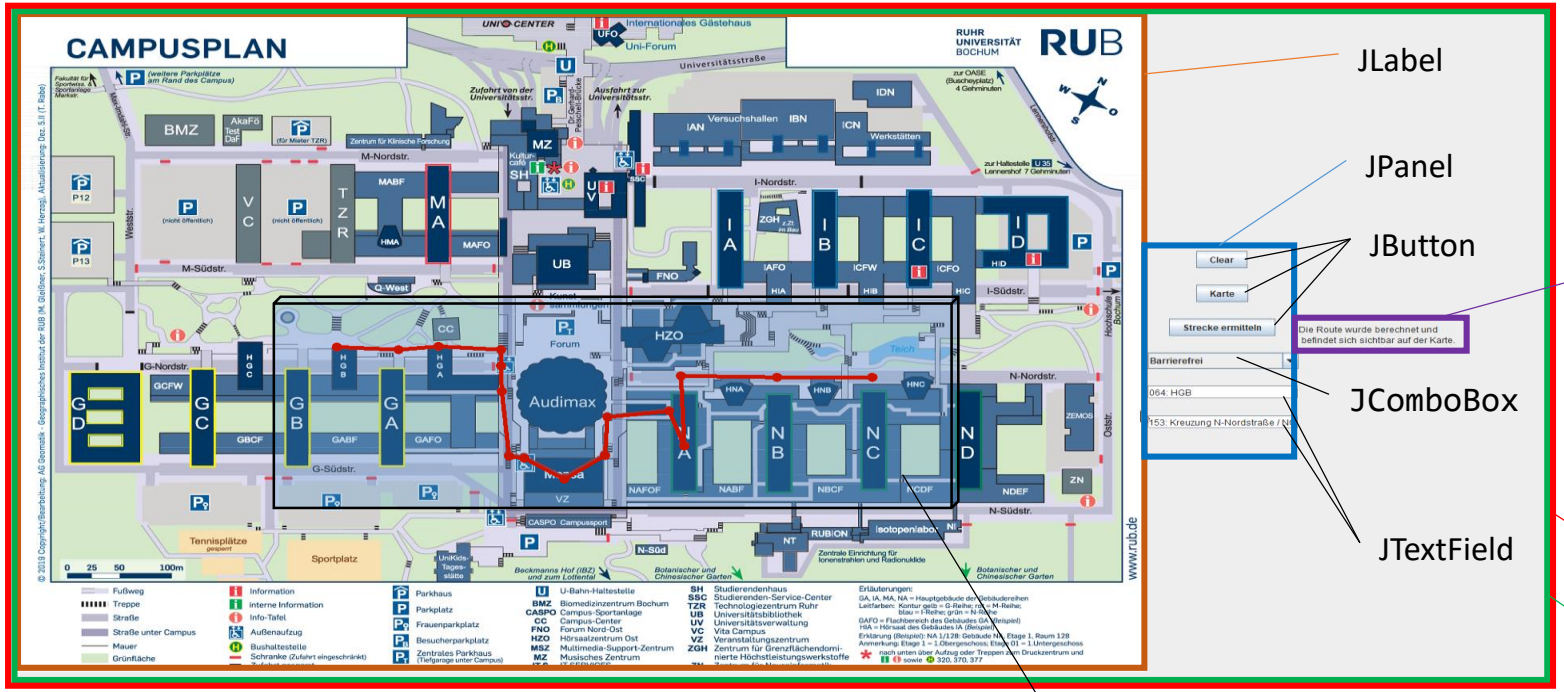
- Java Bibliothek JRE 1.8
- Swing Library
  - JComponents
  - Content Pane
  - Glass Pane
- Layout
  - FlowLayout
  - BoxLayout



- JPopupMenu
  - JMenuItem
  - Koordinatenabgleich
  - setLightWeightPopupEnabled

RUHR UNIVERSITÄT BOCHUM RUB

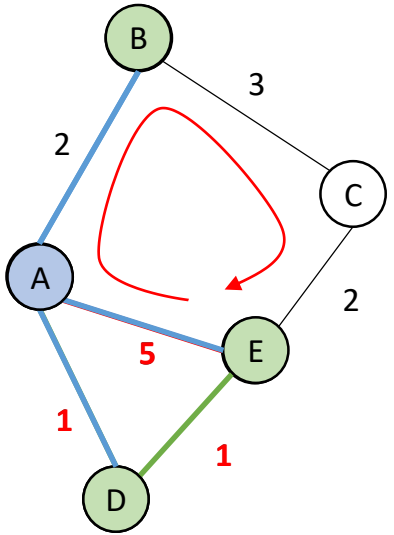
Arbeitspaket: Java GUI



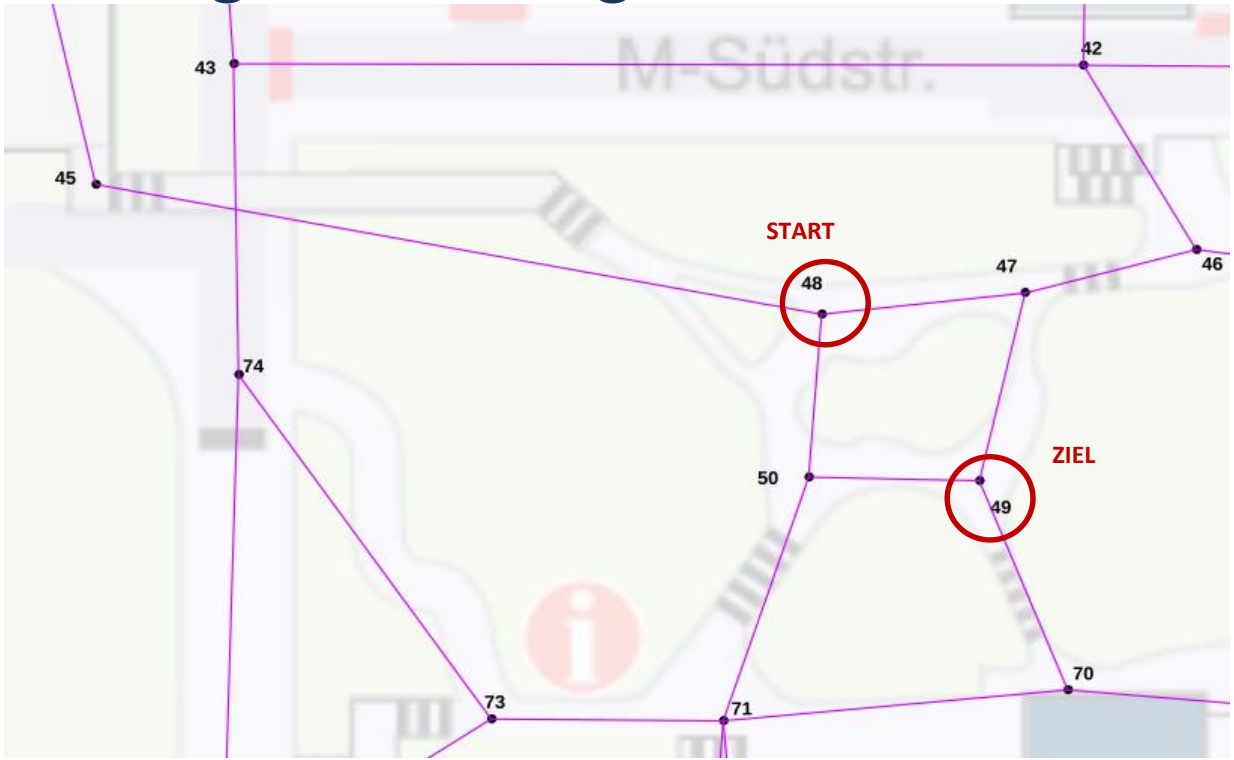
RUHR UNIVERSITÄT BOCHUM RUB

Arbeitspaket: PROLOG

- **TuProlog vs. SWI-Prolog:**
  - SWI-Prolog hat kürzere Laufzeiten
- Zu berücksichtigen:
  - „Merken“, wo man schon war, sonst Endlosschleife
  - Verkürzung der Suche: Berücksichtigung von Distanzen
- **Verschiedene Möglichkeiten der Implementierung:**
  - Breite-Zuerst Suche:
    - Stack-Overflow, da Listen zu lang werden
  - Tiefe-Zuerst-Suche:
    - Läuft lange Wege



Beispiel Pfadgenerierung



RUHR UNIVERSITÄT BOCHUM RUB

Beispiel Pfadgenerierung

go(48, 49, X).

go(From, To, [Path,Distance]), traverse(From), rpath([To|RPath], Dist)-> […] **σ1 = From|48, To|49, [Path, Distance]|X**

traverse(48), rpath([49|RPath], Dist)-> reverse([49|RPath], Path), Distance is round(Dist), write([Path, Dist, Distance]); write('There is no route')

traverse(48), rpath([49|RPath], Dist, […]

traverse(From), retractall(rpath(_,_)), traverse(From,[],0) **σ2 = From|48**

retractall(rpath(_,_)), traverse(48,[],0), rpath([49|RPath], Dist)-> reverse([49|RPath], Path), Distance is round(Dist), write([Path, Dist, Distance]); write('There is no route')

traverse(48,[],0), rpath([49|RPath], Dist), […]

traverse(From, Path, Dist) , path(From, T, D), not(memberchk(T, Path)), shorterPath([T,From|Path], Dist+D), traverse(T,[From|Path],Dist+D) **σ3 = From|48, Path|[],Dist|0**

path(48, T, D), not(memberchk(T, [])), shorterPath([T,48|[]], 0+D), traverse(T,[48|[]],0+D), rpath([49|RPath], Dist), […]

path(48, T, D), not(memberchk(T, [])), shorterPath([T,48|[]], 0+D), traverse(T,[48|[]],0+D), rpath([49|RPath], Dist), […]

path(From,To,Dist), edge(To,From,Dist) **σ4 = From|48, To|T,Dist|D**

edge(T,48,D), not(memberchk(T, [])), shorterPath([T,48|[]], 0+D), traverse(T,[48|[]],0+D), rpath([49|RPath], Dist), […]

edge(T,48,D), not(memberchk(T, [])), shorterPath([T,48|[]], 0+D), traverse(T,[48|[]],0+D), rpath([49|RPath], Dist), […]

edge(45,48,442) **σ5 = 45|T, D|442**

not(memberchk(45, [])), shorterPath([45,48|[]], 0+442), traverse(45,[48|[]],0+442), rpath([49|RPath], Dist), […]

shorterPath([45,48|[]], 0+442), traverse(45,[48|[]],0+442), rpath([49|RPath], Dist), […]

shorterPath([H|Path], Dist), rpath([H|_], D), !, Dist < D, retract(rpath([H|_], _)), assert(rpath([H|Path], Dist)) **σ6 = H|45, Path|[48], Dist|0+442**

rpath([45|_], D), !, 442 < D, retract(rpath([45|_], _)), assert(rpath([45|[48]], 442)), traverse(45,[48|[]],0+442), rpath([49|RPath], Dist , […]

Ist false, da nicht definiert, daher wird „shorterPath([H|Path], Dist)“ abgebrochen → Backtracking → Finden von „shorterPath(Path, Dist)“

shorterPath([45,48|[]], 0+442), traverse(45,[48|[]],0+442), rpath([49|RPath], Dist), […]

shorterPath(Path, Dist), assert(rpath(Path,Dist)) **σ6 = Path|[45,48], Dist|442, rpath([45,48], 442)**

traverse(45,[48|[]],0+442), rpath([49|RPath], Dist), […]

traverse(45,[48|[]],0+442), rpath([49|RPath], Dist), […]

traverse(From, Path, Dist) , path(From, T, D), not(memberchk(T, Path)), shorterPath([T,From|Path], Dist+D), traverse(T,[From|Path],Dist+D) **σ7 =From|45, Path|[48], Dist|0+442**

path(45, T, D), not(memberchk(T, [48])), shorterPath([T,45|[48]], 0+442+D), traverse(T,[45|[48]],0+442+D), rpath([49|RPath], 0+442), […]

path(45, T, D), not(memberchk(T, [48])), shorterPath([T,45|[48]], 0+442+D), traverse(T,[45|[48]],0+442+D), rpath([49|RPath], 0+442), […]

path(From,To,Dist), edge(To,From,Dist) **σ8 =From|45, To|T, Dist|D**

edge(T,45,D), not(memberchk(T, [48])), shorterPath([T,45|[48]], 0+442+D), traverse(T,[45|[48]],0+442+D), rpath([49|RPath], 0+442), […]

Blaue Prädikate sind Metaprädikate und werden nicht in dem Sinne zur Resolution gebracht, sondern sind quasi immer true

Evaluierung

- Erfüllung aller gesetzten Anforderungen
  - Laufzeit < 5 (Tmax_ohnebarriere = 2s, Tmax_mitbarriere = 0,5s)
  - Wegberechnung für anhand von uns definierter Pfade und Attributzuweisungen ergibt plausible Ergebnisse
  - Benutzer erhält Rückmeldungen über die grafische Oberfläche
- SWI-Prolog rechnet schneller als TU-Prolog
- Beschränkungen durch den Suchalgorithmus
    - Je nach Bezeichnung der Knoten (aufsteigend / absteigend) wird der Aufwand deutlich größer.
    - Es wird immer die erste gefundene Kante genommen und von dort aus weiter gesucht, ohne darauf zu achten, ob dies in die richtige Richtung führt

Ausblick

- Erweiterung der Attributliste
- Detailliertere Netzgestaltung
- Verbesserungsvorschläge der PROLOG-Implementierung:
  - Berücksichtigung des Abstands zum Endpunkt (Heuristik)
  - Aufteilung der Karte in verschiedene Sektoren, sodass „falsche“ Sektoren ausgeschlossen werden können