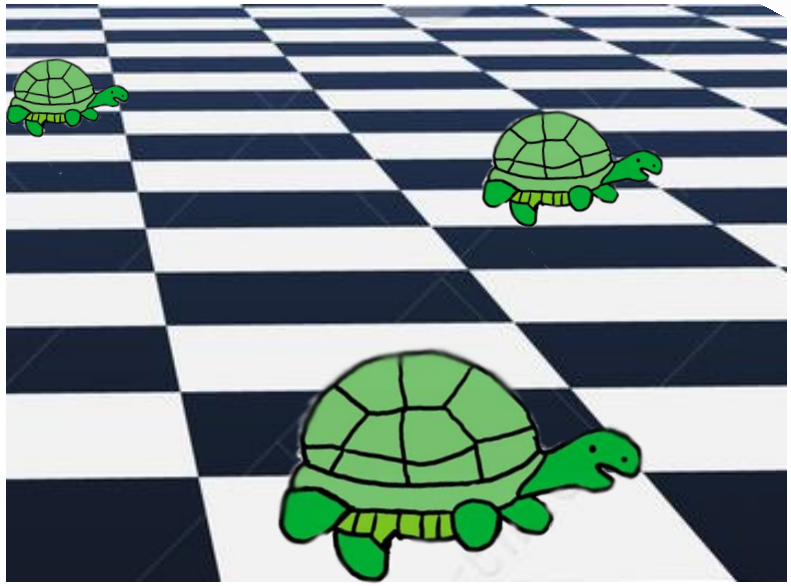
Übung 4

Institut für Technische Informatik

10.12.2018

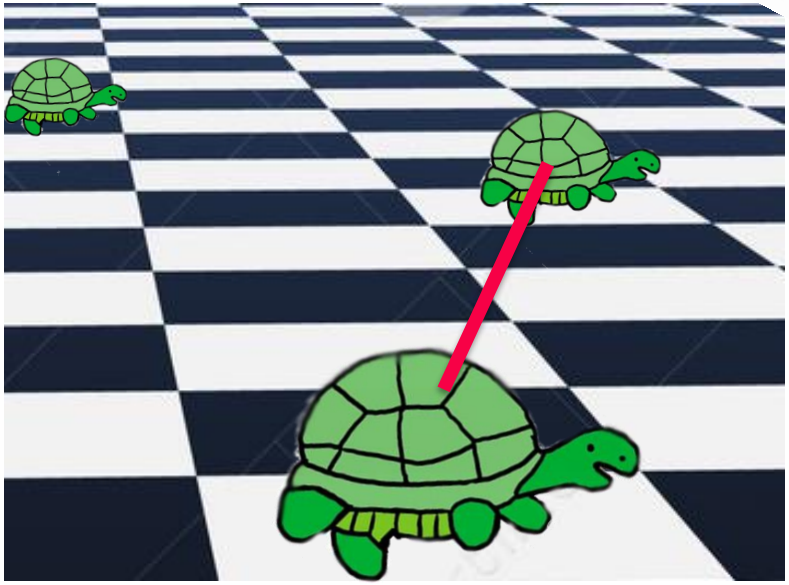
NetLogo Agents

- NetLogo is a world made of four kinds of agents
  - Patches: make up the background or landscape
  - Turtles: move around on top of the patches
  - Observer: oversees everything going on in the world
  - …



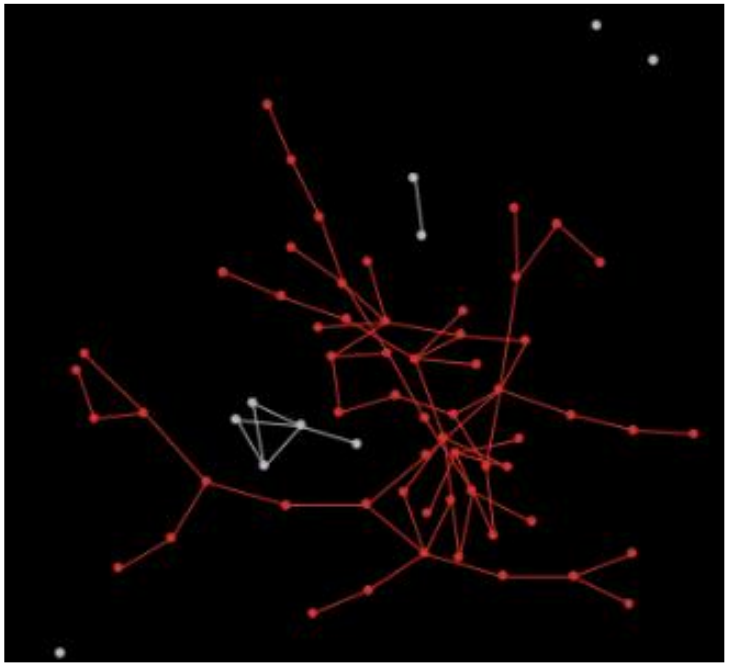
NetLogo Agents

- NetLogo is a world made of four kinds of agents
  - Patches: make up the background or landscape
  - Turtles: move around on top of the patches
  - Observer: oversees everything going on in the world
  - Links: connection between two turtles



Links

- Agents that connect turtles
  - A link appears as a line corresponding to the shortest path between the two turtles
  - Very useful to build networks and graphs



Links

- Different types of links
  - Unidirectional
  - Bidirectional

- Creating links
  - Unidirectional links
    - **create-link-to agent**,
      create-link-from agent
    - **create-links-to agentset**,
      create-links-from agentset
  - Example

```
crt 2
ask turtle 0 [
  create-link-to turtle 1
]
```

Links

- Creating links
  - Bidirectional links:
    - **create-link-with agent**, **create-links-with agentset**
  - You cannot have more than one bidirectional link (or more than one unidirectional link going in the same direction) between the same two nodes
    - If you try to create a link where one (of the same breed) already exists, nothing happens
- A node cannot be linked to itself!
  - Creating a link from a turtle to itself generates a runtime error

```
to setup
  ca
  crt 3
  ask turtle 0 [ create-link-with turtle 0 ]
end
```

Links

- Retrieving links
  - who number of the two turtles it links
  - **link end1 end2** returns the link connecting end1 and end2
    - Returns *nobody* if there is no link connecting the 2 ends
  - **links** returns an agentset of all existing links

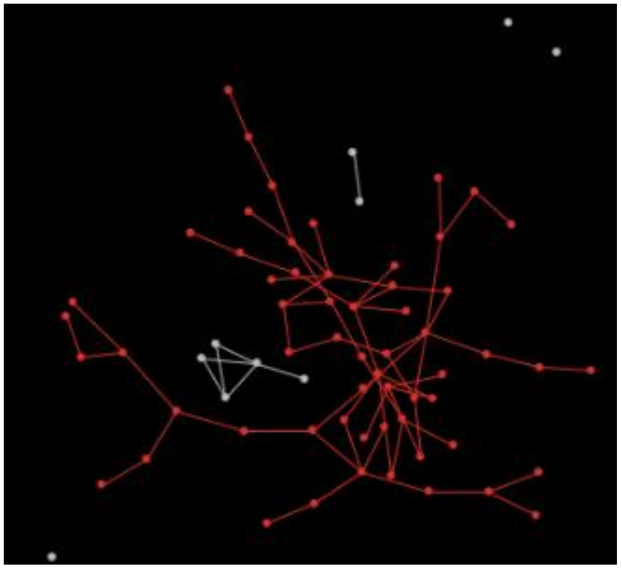
```
to setup
  ca
  crt 3
  ask turtles [fd 10]
  ask turtle 0 [
    create-link-with turtle 1
    create-link-with turtle 1 ;; does nothing
  ]
end
```

Links

- Agents have own variables
  - **links-own**

```
links-own [traffic]
```

  - Built-in variables
    - **end1**
    - **end2**
    - **color**, **shape**, **thickness**, …



- One can specify breeds of links
  - Example

```
directed-link-breed [streets street]
streets-own [cars bikes]
```

Links

- Get information about a link
  - **in-link-neighbor?**, **out-link-neighbor?** report true if there is a directed link going from turtle to the caller
  - **in-link-neighbors?**, **out-link-neighbors?** reporting the agentsets that have ingoing/outgoing links to/from the caller

```
crt 2
ask turtle 0 [
  create-link-to turtle 1 ;; unidirectional link
  show in-link-neighbor? turtle 1 ;; false
  show out-link-neighbor? turtle 1 ;; true
]
```

Links

- Get information about a link
  - **both-ends** returns the two turtles connected by a link

```
crt 2
ask turtle 0 [
        create-link-with turtle 1
]
ask link 0 1 [
        ask both-ends [
                set color red ;; turtles 0 and 1 turn red
        ]
]
```

Links

- Returning an agentset of all neighboring turtles that:
  - **in-link-neighbors**: that have directed links coming from them to the caller
  - **out-link-neighbors**: that have directed links coming from the caller

```
crt 4
ask turtle 0 [ create-links-to other turtles ]
ask turtle 1 [ ask in-link-neighbors [ set color blue ] ] ;; turtle 0 turns blue
```

  - **link-neighbors**: that are at the other end of undirected links connected to this turtle

```
crt 3
ask turtle 0 [
  create-links-with other turtles
  ask link-neighbors [ set color red ] ;; turtles 1 and 2 turn red
]
ask turtle 1 [
  ask link-neighbors [ set color blue ] ;; turtle 0 turns blue
]
end
```

Links

- Location of a link
  - Links do not have a location as turtles do
  - They are not considered to be on any patch and you cannot find the distance from a link to another point
  - But you can get the distance between connected turtles using **link-length**
- Hide links
  - **hide-link** set the link variable **hidden?** to true
  - **show-link** makes the link visible again
- Destroy links
  - **die** kills one link
  - **clear-links** destroys every link

Tables and Arrays

- NetLogo supports arrays and hash tables
  - Can be included with **extensions[array]** and **extensions[table]** at the beginning of the procedure tab
  - Include both with **extensions[array table]**
- Arrays
  - Arrays have fixed size and are generated from lists with **array:from-list *list***
  - **array:set *array pos value*** sets the given *value* at the position *pos* in the *array*
  - To get an item from an array use **array:item *array pos***
  - The array length can be retrieved via **array:length *array***
  - To get a list from an array one can use **array:to-list *array***

Tables and Arrays

- Tables
  - A table is a pair of (key,value), where a key could be a string, number, Boolean or list
  - Create a table:
    - **table:make**
    - **table:from-list**
  - To set entries:
    - **table:put table key value**
  - To get an entry:
    - **table:get table key**

Replaces the entry if there was one with that key before

Causes an error if there is no such key

Tables and Arrays

- Tables
  - To remove an entry:
    - **table:remove *table key***
  - To remove all entries:
    - **table:clear *table***
  - To return the number of entries:
    - **table:length *table***
  - To report a list of keys
    - **table:keys *table***
  - To check if the table has a specific key
    - **table:has-key? *table key***

Tables and Arrays

- Tables
  - Example (i)

```
let dict table:make
table:put dict "turtle" "Schildkröte"
table:put dict "node" "Knoten"

print table:length dict ;; 2
print table:keys dict ;; [turtle node]

if table:has-key? dict "turtle"
   [table:remove dict "turtle"]

print table:keys dict ;; [node]
```

Tables and Arrays

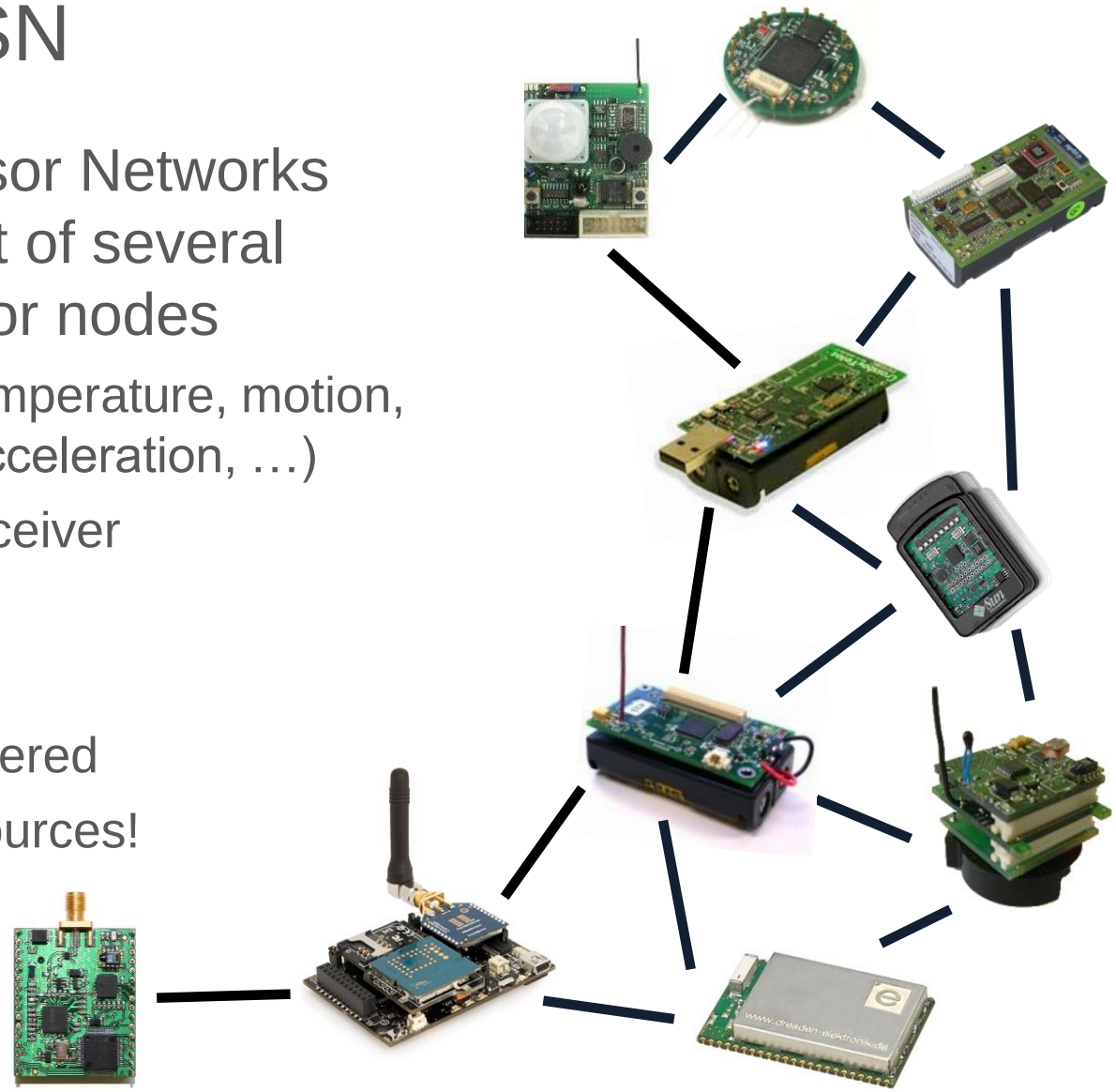
- Tables
  - Example (ii)

```
extensions[table]
turtles-own [ props ]

to setup
  clear-all
  create-turtles 2
  ask turtles [
    set props table:make
    let dummy_variable 0
    set dummy_variable random 2 ;; either 0 or 1
    ifelse dummy_variable = 0
       [ table:put props "energy" "low" ]
       [ table:put props "energy" "high" ]
  ]
    print count turtles with [table:get props "energy" = "high"]
end
```

Exercise 4: WSN

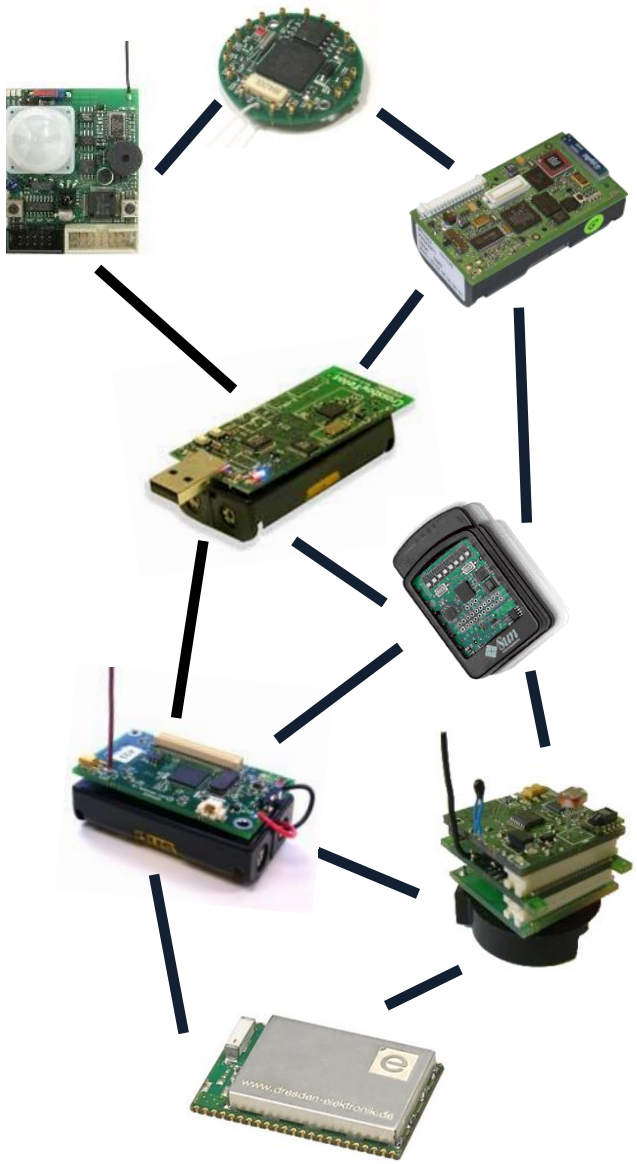
- Wireless Sensor Networks (WSN) consist of several wireless sensor nodes
  - Sensors (temperature, motion, pressure, acceleration, …)
  - Radio transceiver
  - Small-size
  - Low cost
  - Battery-powered
  - Limited resources!



Exercise 4: WSN

- Typical scenario
  - High amount of nodes randomly dispersed in the field (no topology information)

→ The WSN has to self-organize (topology, role assignment) using distributed algorithms



Exercise 4: WSN

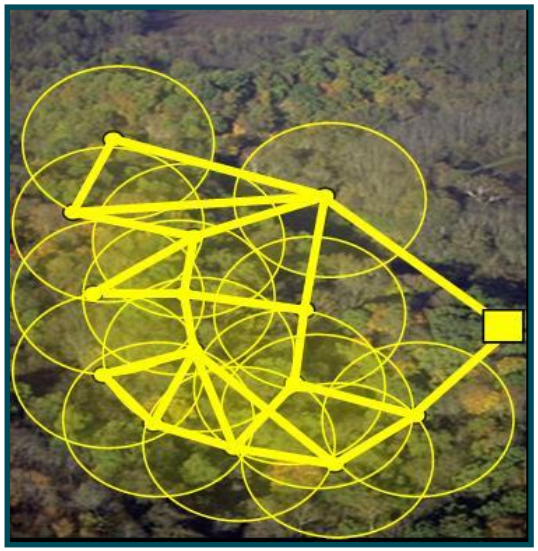
- Role Assignment
  - Every node chooses a role
  - Each role corresponds to a task (e.g., sensing, forwarding, inactive)
  - Example applications are coverage, clustering, in-network aggregation
  - References

(1) Kay Römer, Christian Frank, Pedro Jose Marron, Christian Becker. Generic Role Assignment for Wireless Sensor Networks. *In Proceedings of the 11th workshop on ACM SIGOPS European workshop*, Leuven, Belgium, 2004.

(2) Christian Frank, Kay Römer. Algorithms for Generic Role Assignment in Wireless Sensor Networks. *In Proceedings of the 3rd international conference on Embedded networked sensor systems, pag. 230-242.* San Diego, California, USA. 2005

Exercise 4: Coverage Problem in WSN

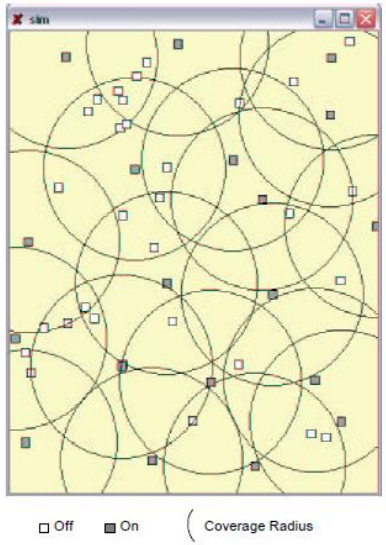
- Your Task
  - Build a NetLogo model of a WSN with turtles being sensor nodes connected through links
  - Sensor nodes have a given sensing radius and a given battery level
  - Cover a given area with as few nodes as possible (redundant nodes can replace nodes with depleted batteries)
    - Maximum coverage of the area
    - Minimum number of nodes that actively sense the environment



Exercise 4: Coverage Problem in WSN

- Cover a given area with as few nodes as possible
  - Abstract definition of the role assignment algorithm:

```
ON :: {
  battery-level > 0 &&
  count(1 hop) {
  role == ON
} <= 0 }
OFF :: else
```

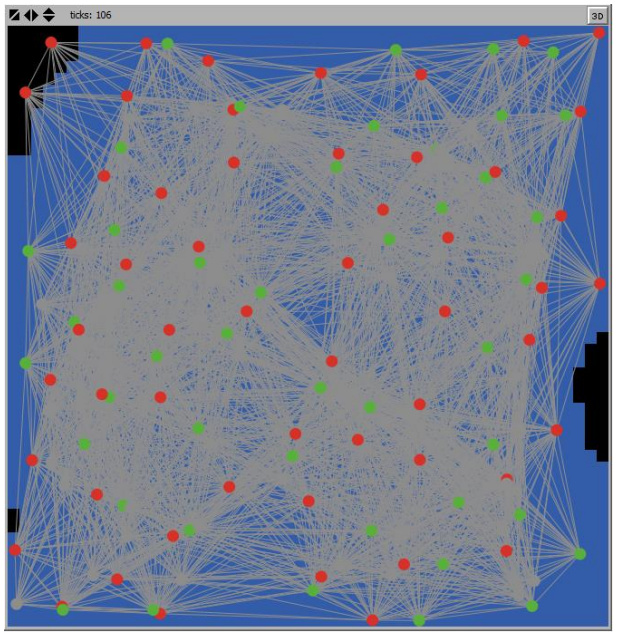


  - Each node that has a sufficient battery level…
  - ... and no neighbors with role ON should get role ON
  - Every time that a value changes, the node evaluates the predicates and chooses a role

Exercise 4: Coverage Problem in WSN

- Cover a given area with as few nodes as possible
  - Abstract definition of the role assignment algorithm:

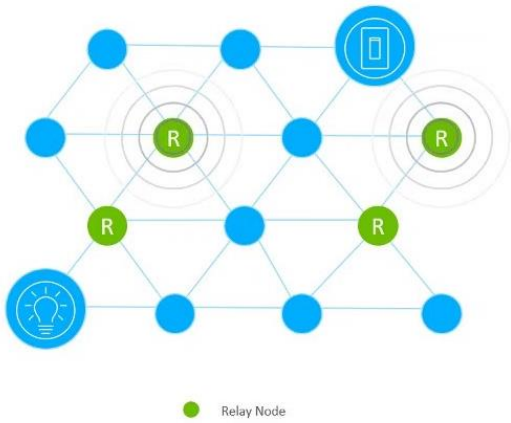
```
ON :: {
  battery-level > 0 &&
  count(1 hop) {
  role == ON
} <= 0 }
OFF :: else
```



  - Nodes with depleted battery (red)
  - Nodes with role OFF (gray)
  - Nodes with role ON (green)

Exercise 4: WSN

- Relay node selection in sensor networks
- Flooding
  - BLE and Zigbee
  - Used instead of routing
- Managed flooding
  - Time to live message counter
  - Relay nodes
  - Use role assignment



https://www.youtube.com/watch?v=TJSQzQz5CWA

- Deadline is Sunday, 06.01.2019, at 23:59 CET
  - Follow the instructions carefully!

Questions?