Session 6: Visualization and Mobility

Tobias Baumgartner

Shawn Winter School 2008, 17.-21.11.08, Lübeck, Germany

November 19th, 2008

Outline

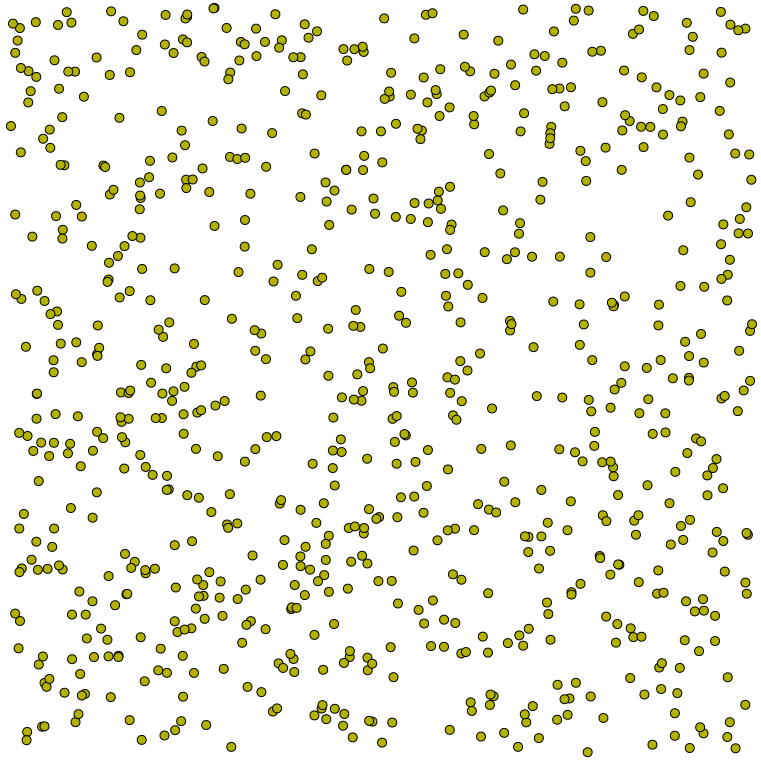
Outline

Basic Functionality

- Optional application in `src/apps/vis`
- Must be explicitely enabled with `ccmake ../src`
  $\longrightarrow$ Set `MODULE_APPS_VIS` to `ON`
- Library `cairo` required
- Easy to configure
- Flexible
- Extensible

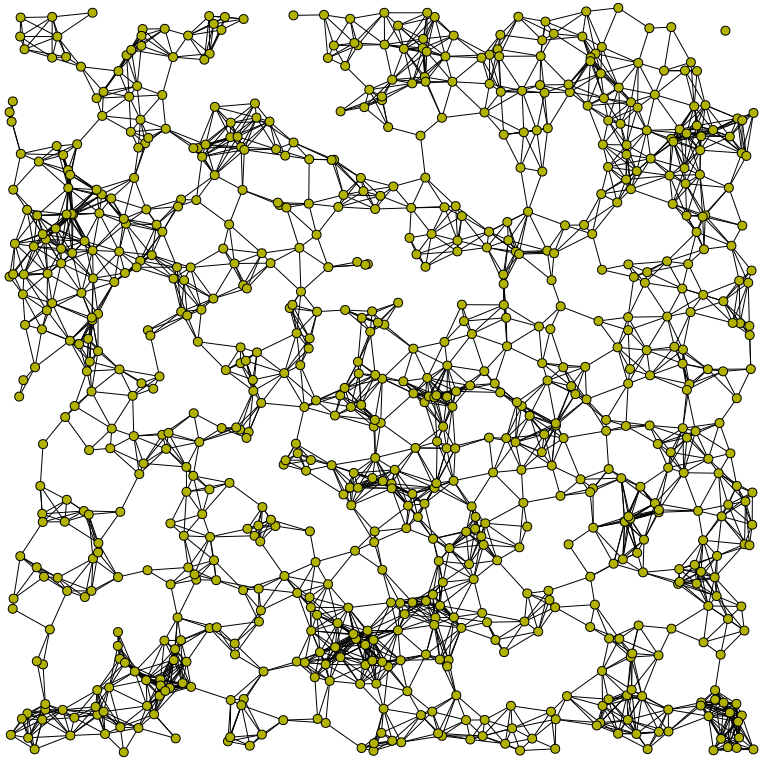
Basic Visualization

```
# Set name of visualization
# that is used in ongoing
# tasks
vis=my_visualization
# Initialization, create
# underlying structures
vis_create

# Camera setup (simple setup)
vis_simple_camera

# Create output image
# (default: snapshot.pdf)
vis_single_snapshot
```



Basic Visualization with Edges

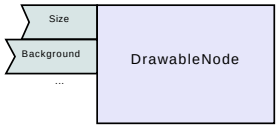
```
# Initialize Vis:
vis=my_visualization
vis_create

# Create Edges between Nodes
vis_create_edges \
  source_regex=.* \
  target_regex=.*

# Camera setup (simple setup)
vis_simple_camera

# Create output image:
vis_single_snapshot
```

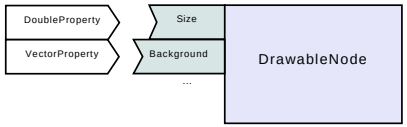


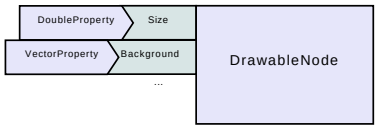
Using Visualization for Creating Videos

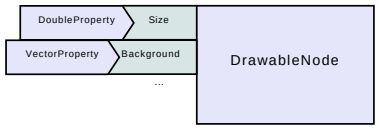
Outline

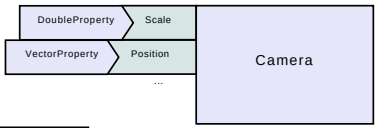
Objects and Properties

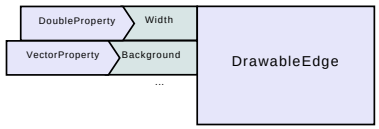












Properties

- Used by other vis objects to set options, e.g.
  - Width of the generated image
  - Color of Nodes
  - ...
- Basic properties `int`, `bool`, `double`, `Vec`
- Priority, if there are multiple Properties per Object
- Configure via Tasks
  - `vis_constant_double value=0.2 \`
    `elem_regex=node.default.* prop=size prio=1`
  - `vis_constant_double value=0.3 \`
    `elem=node.default.v23 prop=size prio=2`
  - `vis_constant_double value=38 elem=cam prop=scale prio=1`

Drawables (1 of 3)

- **DrawableNode**
  - Draw a `Node` on the image
  - One implemenation: `DrawableNodeDefault` that sets "node.default" as prefix for elements, and draws the node as a circle
  - E.g., change color of a node:

```
vis_constant_vec x=1 y=0 z=0 \
  elem_regex=node.default.* prop=background prio=1
```

  - Parameters are, for example, `background`, `foreground`, or `size`
  - If other drawings like shapes or small battery symbols are needed, one can implement an own `Drawable`

Drawables (2 of 3)

- **DrawableEdge**
  - Draw connection between two nodes
  - `DrawableEdgeDefault` with task `vis_create_edges`
  - E.g., change color of an edge:

```
vis_create_edges source_regex=.* target_regex=.*
vis_constant_vec x=0 y=0 z=0.8 \
  elem_regex=edge.default.* prop=color prio=1
```

  - Parameter `line_width` provided

Drawables (3 of 3)

- **DrawableGraphics**
  - Take image as background
  - Use task `vis_graphics` with paramters `name` and `file`

Camera

- Used to define clipping of visualization
- Configure via properties, e.g.

```
vis_constant_vec x=18.75 y=18.75 z=0 \
  elem=cam prop=position
```

- More properties: `background`, `width`, `height`, `scale`
- Task **vis_simple_camera** that creates simple view

Writer

- Final Step: Create the Image
- Different writers available: `ps`, `pdf`, `png`
- Task **vis_single_snapshot**
  - Create single image
  - Parameter `filename`
  - Parameter `writer` can be `ps`, `pdf`, `png`
- Additional Tasks
  - **vis_external_animation** that creates a sequence of one point in time
  - **vis_internal_animation** that provides a live update of image: Movie. **Sorry, still under development.**

Advanced Configuration

- Set color related to Tag value
  - Tag value contains `double` value 0..1
  - Color varies from red for 0 to blue for 1
  - Use task `vis_tag_color_vec` and parameter `dynamictag`

```
vis_tag_color_vec elem_regex=node.* \
  dynamictag=temperature prop=background
```

- Create Tree with Edges: Task `vis_create_edges_tree`
  - Draw exactly one edge per node to its predecessor
  - Node must create `StringTag` with label of predecessor
  - Tag must be named `predecessor`

Outline

Extend Visualization

- For special features, Visualization must be extended
- Example: Color related to special tag values
  - Write your own property derived from `PropertyTask`
  - Property returns your own ColorVec

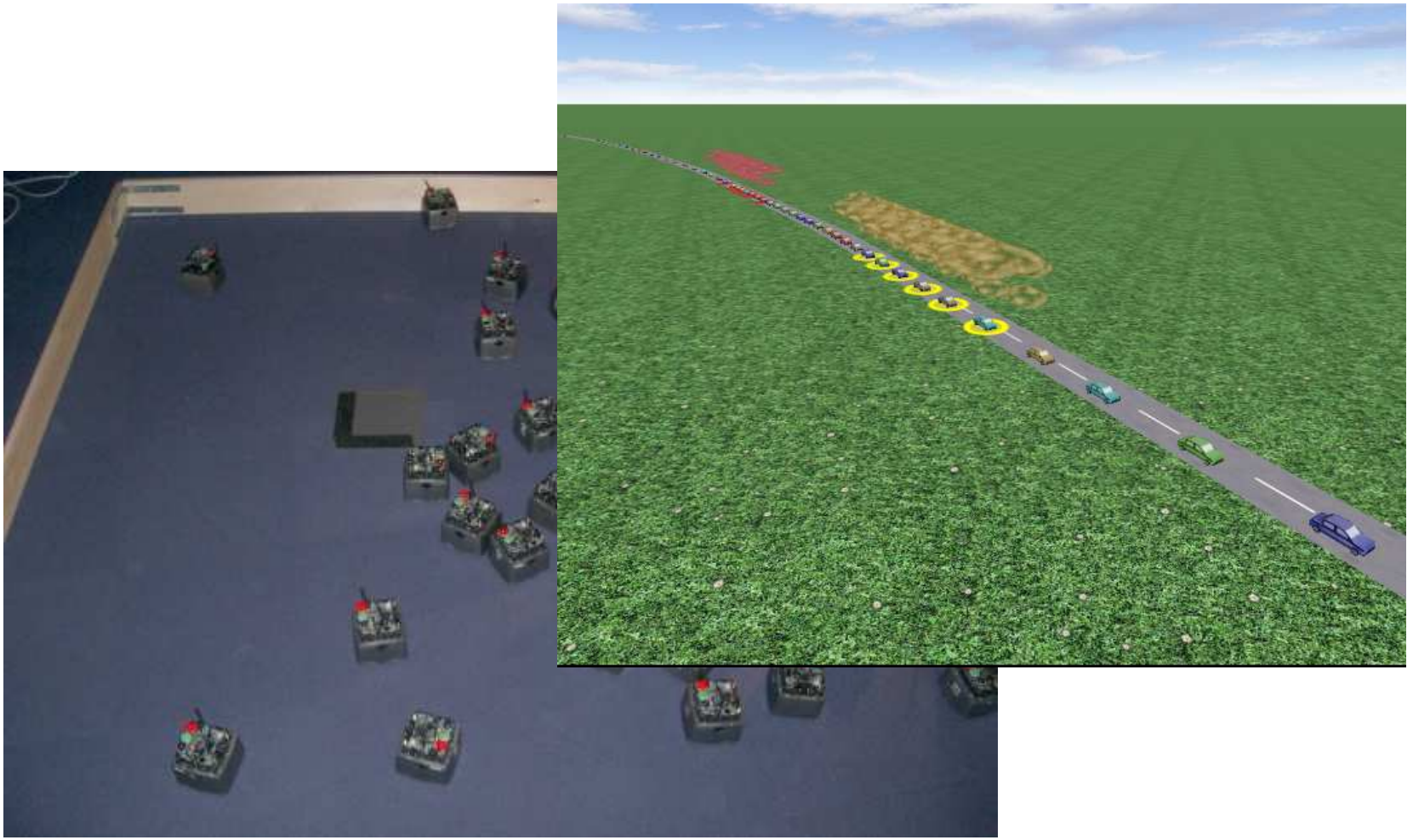
```
1  class PropertyMySpecialColorVecTask
2      : public PropertyTask {
3    ...
4    class PropertyMySpecialColorVec
5        : public Property<shawn::Vec>
6    {...};
7    ...
8    /// Return own color defined by PropertyMySpecialColorVec
9    virtual ConstPropertyHandle
10       create_property( shawn::SimulationController& sc )
```

  - Add to simulation task keeeper

```
1  sc.simulation_task_keeper_w().add(
2          new PropertyMySpecialColorVecTask );
```

Outline

Active Mobility vs. Passive Mobility

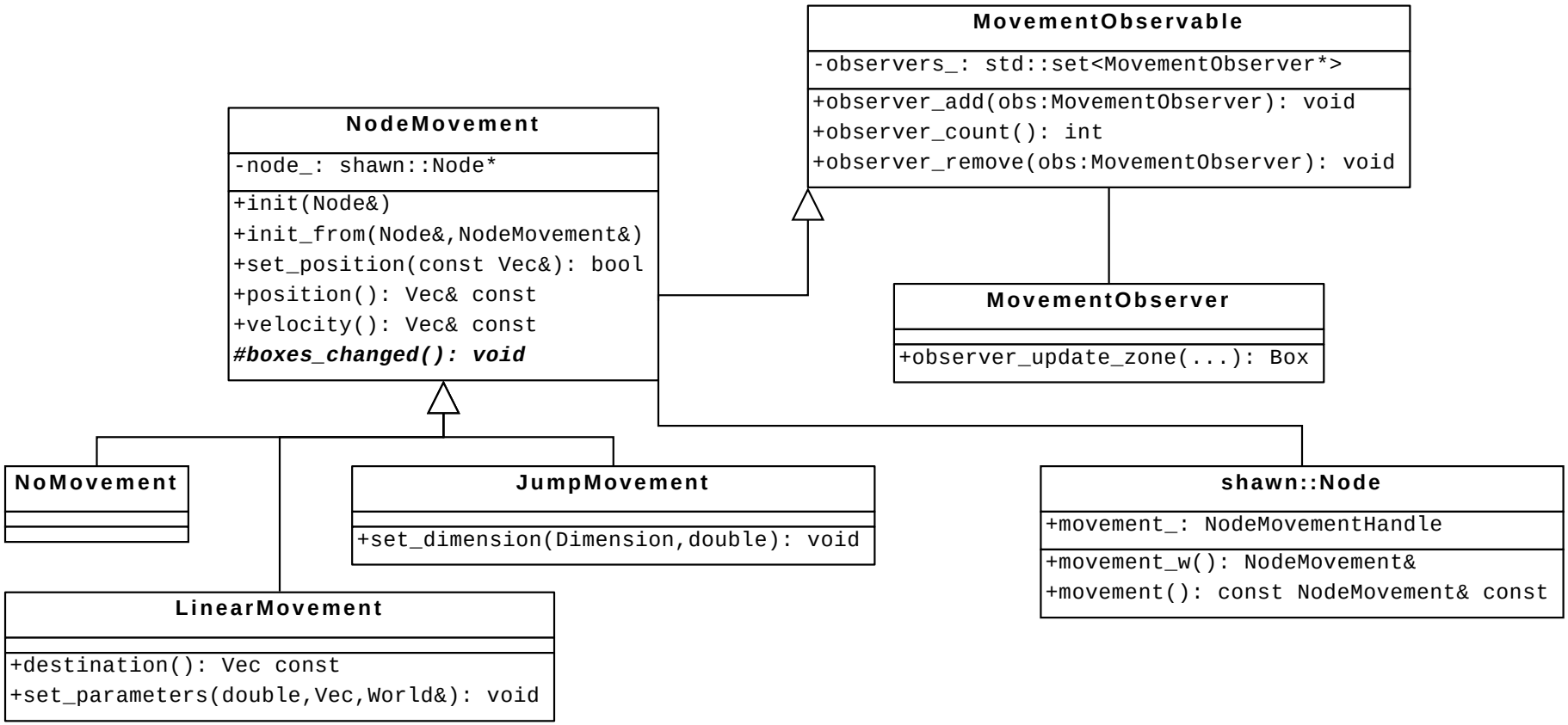


Outline

Basic Functionality

- Mobility allows a Node to change position
- Allows for **continuous** movements
  - Exact Position at **any** point in time
  - Current velocity provided at **any** point in time
- Different kinds of mobility models
  - Playback of Files
  - Implement own Mobility
  - Static Network
- Playback of Movements: Allows for complex net behavior

Base Class Diagram



Outline

Basic Node Movement

- Each Node is connected to exactly one NodeMovement
- Provide position for Node when `real_position()` is called
  (Node asks Movement for position)
- Access via Node to change or update positions:
  - `node.movement_w()`
  - `node.set_movement()`
- Access via Node to get additional values like velocity:
  - `node.movement().velocity()`

MovementObserver

- Movement is observable
- Observers can register themselves to get informed on updates
- Each registered observer adds an observed zone (`Box`)
- If Movement leaves at least one of the boxes, the observers are notified and provide a new observed zone
- Main Observer is EdgeModel:
  GridModel runs update if Node it leaves observed zonde

Movement Implementations (1 of 2)

- **NoMovement**
  - Standard movement for static networks
  - Allows exactly **one** call of `set_position`
- **JumpMovement**
  - Jumps directly to given position
  - Not a continuous Movement
  - No velocity provided
  - Example

```
1  shawn::JumpMovement *jm =
2    dynamic_cast<shawn::JumpMovement*>(
3      &node.movement_w() );
4  jm->set_position( new_pos );
```

Movement Implementations (2 of 2)

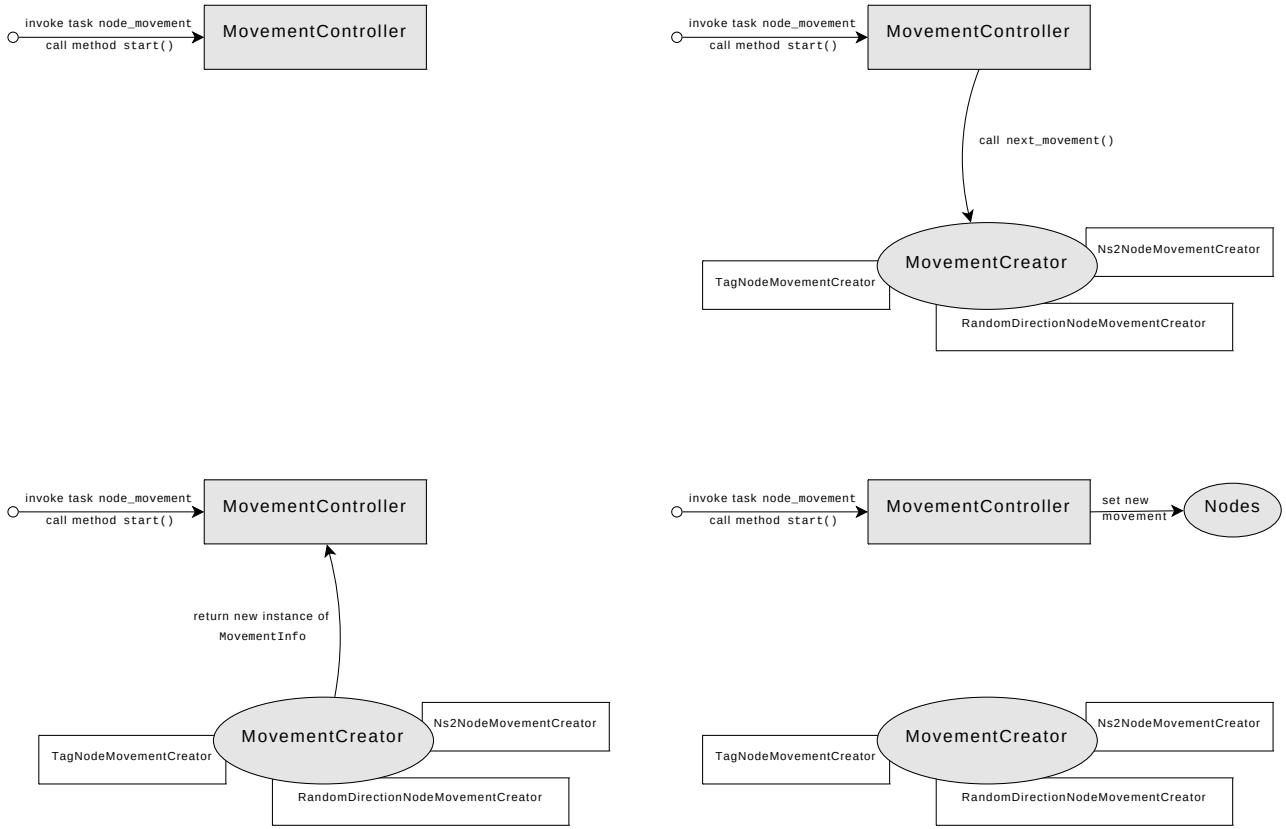
- **LinearMovement**
  - Continuous Movement
  - Set target position and velocity
  - Nodes move automatically and continuously
  - Position is recalculated on each request
    => Node has position in any point in time
  - Example

```
1  lm = new shawn::LinearMovement();
2  lm->set_parameters( velocity, pos,
3     simulation_controller_w().world_w() );
4  node.set_movement( lm );
5  // ...
6  // for updates, call set_parameters again
```

  - Complex paths by chaining Movements

Outline

Playbacks: Basic Functionality



Playback Implementations (1 of 2)

- **NS2NodeMovementCreator**
  - Parser for ns2 mobility files
  - Set initial position

```
$node_(0) set X_ 10.45
$node_(0) set Y_ 14.67
$node_(0) set Z_ 0.0
```

  - Set ongoing positions (X, Y, velocity)

```
$ns_ at 1 "$node_(0) setdest 12.38 14.99 30.79"
```

- **RandomDirectionNodeMovementCreator**
  - Nodes move randomly in World
  - Random velocity from `random_direction_v_min` to `-v_max`
  - Random direction $0 \leq dir \leq 2\pi$
  - Move `random_direction_t_move_min...-max` rounds
  - Then pause `random_direction_t_stop_min...-max` rounds

Playback Implementations (2 of 2)

- **TagNodeMovementCreator**
  - Read movement information over tags (e.g. set via XML file)
  - Group must be either `linear` or `jump`
  - Destination via Double-Tags `x-pos`, `y-pos`, `z-pos`
  - If `linear`: Additional tags `velocity`, `arrival_time`, and `duration` are read in
  - Each visited node gets the Bool-Tag `visited` attached
- **Direct Connection to SUMO**
  - SUMO[1] - Simulation of Urban MObility
  - Optional Application in `shawn/src/apps/traci_client`
  - Set `MODULE_APPS_TRACI_CLIENT` to `ON` with `ccmake`
  - Set parameters `remote_host` and `remote_port`

[1] `http://sumo.sourceforge.net`

Outline

Exercises

- Use `http://wisebed.eu/ws08/ws08-session06.conf` that is preconfigured to create an image of the tree
- Implement a simple mobility that is used for the gateway (example provided on next slide)
- Adapt your tree routing algorithm to deal with mobile gate
  - Gate floods the network periodically
  - Nodes accept greater hopcount even if it is greater after given time (e.g., three times the flooding period)
  - Node are only allowed to accept messages from nodes with greater hopcount

Hints

```
#include "sys/node_movements/linear_movement.h"
...
void
Ws08Processor::
init_movement( void )
    throw()
{
    double width  = owner().world().upper_right().x()
       - owner().world().lower_left().x();
    double height = owner().world().upper_right().y()
       - owner().world().lower_left().y();

    shawn::Vec new_pos = shawn::Vec(
          owner().world().lower_left().x() + width / 2,
          owner().world().lower_left().y() + height / 2,
          0.0 );

    shawn::LinearMovement *lm = new shawn::LinearMovement();
    lm->set_parameters( (new_pos - owner().real_position()) / 20.0,
                        new_pos,
                        owner_w().world_w() );
    owner_w().set_movement( lm );
}
```

Example Output of Visualization