448.057 Context-Aware Computing (UE)
Exercise 5 – Growing Point Language

Task description

The task for the fifth and last assignment of the Context-Aware Computing exercises consists in simulating the *ortho+* and *ortho-* tropism of the Growing Point Language (GPL) using NetLogo. The GPL is a programming language that facilitates the self-organization of complex pre-specified patterns, such as the interconnection structure of an arbitrary electrical circuit.

The three main abstractions in the GPL are pheromones, materials, and growing points (GP). On a two-dimensional medium, processors are spread with a high density. Growing points are processes that can move from processor to processor according to a tropism (movement of an organism due to external stimuli). They additionally can secrete long-range pheromone concentrations to establish a gradient field. In other words, a growing point can connect a set of user-defined points by moving from one point to the next along a straight line and by laying down material along its path. As growing points travel, they label processors they visit with a material. Every material has its own color, i.e., a processor changes its color if material is dropped on it. A tropism is a growing points' rule to decide which processor it has to visit next.

Use the provided skeleton code "`Uebung5_template.nlogo`" containing buttons for `setup`, `go` and one each to create a GP with one of the aforementioned tropisms. These two buttons call empty stub-functions `create-ortho-plus` and `create-ortho-minus`, which you will have to implement. Additionally we provide two buttons `visualize-radius` and `hide-radius` for switching the visualization of pheromone concentrations on and off, respectively. When the `setup` button is pressed, the NetLogo program creates `number-processors` agents of breed `processors`. The method `shuffle-processors` randomizes the positions of these processors.

Extend this template to: (*i*) allow the user to place GPs in the world and define a pheromone radius; (*ii*) simulate the propagation of the first GP to connect the other GPs with lines by laying down material. Users will place the initial, moving GP first. After that by the press of the according `create`-button further GPs, with their respective tropism can be created. These GPs are fixed and will secret pheromones in a radius, entered by the user. The first GP shall move according to the tropism of the acting GP, e.g., for the *ortho+* tropism, the first GP shall move towards the second point along the gradient of increasing pheromone concentration and lay down material on its way. When it reaches the second point, it shall move according to the third point and repeat this operation until no tropism applies anymore. For *ortho-* GPs this would mean as soon as the moving GP leaves the pheromone radius, the point has been visited, therefore it focuses on the next point. Note that a point must be located within the radius of pheromone secretion of the next point to visit. If this is not the case, the algorithm should stop and an appropriate message should be displayed to the user. Ideally the algorithm stops after all points have been visited.

Placement of points. The `setup` button shall ask the user to place the moving GP. It should be placed by clicking somewhere into the world. The processor located nearest to the position of the mouse click is then selected as the location of the GP and assigned a name by settings its label. For the moving GP no radius is needed. Afterwards, the user can place additional GPs by clicking the according `create` button and similarly to the first GP, the location of the GP. The user should then be asked for the radius of the GP. Make sure to color GPs with different tropisms differently. All processors located within that radius shall then be assigned a pheromone concentration that is inversely proportional to the distance from the point.

Michael Stocker, Robert Thomann <michael.stocker@tugraz.at, r.thomann@tugraz.at>

Note that all GPs have to secrete a different pheromone, i.e., processors need to store the concentrations of multiple pheromones (for convenience, you can create pre-defined patterns of GP).

Pheromone visualization. Use the `hide-radius` and `visualize-radius` buttons to switch respectively off and on the pheromone visualization. When implementing this functionality, make sure to use various colors to represent different pheromones and to vary the gradation of a color to highlight different concentrations. Please note that `hide-radius` does not hide the materials laid down by the moving GP.

Simulate the propagation of the GP. Upon pressing the `go` button, the first GP shall move according to the active GP. This is achieved by implementing the needed tropism.

- *ortho+*: the moving GP moves in the direction of the maximum **increase** of pheromone concentration secreted by the active point. The active GP counts as visited when the moving GP moves onto the processor containing the active GP.
- *ortho-*: the moving GP moves in the direction of the maximum **decrease** of pheromone concentration secreted by the active point. The active GP counts as visited when the moving GP leaves the pheromone radius.

Along the way, the GP shall drop material on each visited processor by changing its color to red. As soon as one GP has been visited the moving GP focuses the GP which has been placed next by the user. Visiting in this context means that the visited GP will be disabled for the remainder of the run. If all GPs placed by the user have been visited the program shall stop.

Hard-coded test program(s). In addition to the manual placement of growing points by the user, use an additional button or slider to let your program draw "Das Haus vom Nikolaus"[1] using hard-coded growing points (i.e., you do not need to ask the user to manually enter the position of the GPs) and take a screen-shot of your solution. Draw this picture twice, once with only *ortho+* and once with only *ortho-*. Add at least one more hard-coded, well known shape which can be drawn by the press of a button. It must use both tropisms. Feel free to test also other arbitrary shapes.

Basic functionality *(10 points)*. Create a NetLogo model that follows the specifications above and make sure that the program does not suddenly break or enter an infinite loop. The submitted code should be properly commented and each team member should be able to explain it. Please write a short description including any remark about your implementation in the space below.

[1] `http://de.wikipedia.org/wiki/Haus_vom_Nikolaus`

Michael Stocker, Robert Thomann <michael.stocker@tugraz.at, r.thomann@tugraz.at>

Bonus task: *diag+* and *diag-* tropisms *(+2 points)*. Students have also the chance to collect additional points by extending their NetLogo program with the *diag+* (counterclockwise) and *diag-* (clockwise) tropisms. Add two more `create`-buttons for these tropisms. The *diag* tropisms move orthogonal to the gradient of the pheromone concentration. Find a useful way to include these tropisms into your drawings. This means that you should define yourself when a *diag*-GP counts as visited. Add another hard-coded drawing showing the new tropisms.

Please submit your NetLogo source code including screen-shots to the TeachCenter as illustrated in the tutorial within **Sunday, 20.01.2019, 23:59 CET**.

Group ID:

Stud. Names:

Finalize Form

Michael Stocker, Robert Thomann <michael.stocker@tugraz.at, r.thomann@tugraz.at>