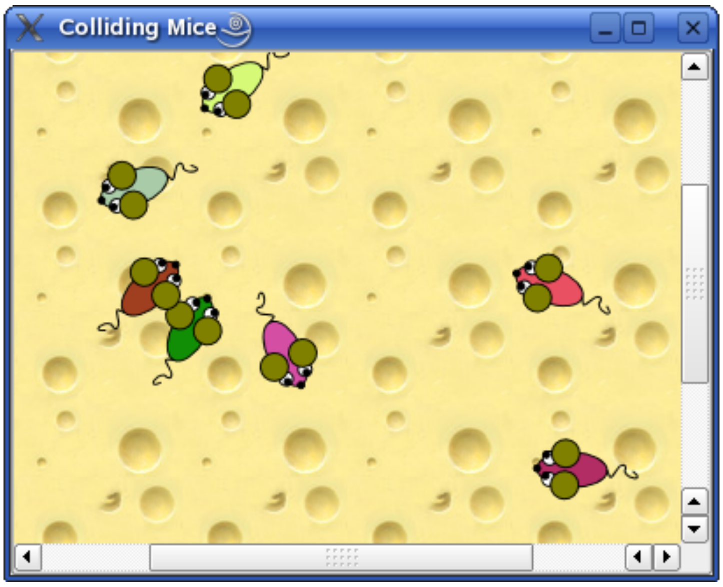
Project IT3709 Fuzzy logic reasoner :“Competing Mice”

The figure below shows a screen dump for the running example code and is the starting point for the project.



A number of mice are moving around a cheese area within a radius of 150 (pixels) from the center. They try to avoid colliding with each other. When two mice collide, their ears become red.

The main goal of the project is to implement a **fuzzy logic reasoner** to give the mice intelligent behavior. This behavior is related to trying to **control the cheese area** (circle) and includes actions like: attack a mouse, wait (noaction) and flee. The reasoning consists of four steps: fuzzifaction, rule evaluation, aggregation of rule output, and defuzzication. You should also consider use of hedges in the fuzzy rules (less, more, very, …). Before starting programming you have to understand Mamdani (and Sugeno) inference. There will be in a fuzzy logic lecture at the end of September (start of the second part of the course). The theory is described in the corresponding hand-outs.

In this subproject you are free to use any programming language and do all the graphics yourself, but there exists code for all the graphics in C++ using Qt. The code is found at: http://qt-project.org/doc/qt-5.0/qtwidgets/graphicsview-collidingmice.html. Note that this gives easy access to simple behavior and graphics needed, so you can concentrate on the fuzzy logic part. The Qt programming environment uses C++ and runs on Windows, Linux, Mac etc. Qt can be downloaded from http://qt-project.org/downloads. You should also read through the graphics view framework found at at http://qt-project.org/doc/qt-5.0/qtwidgets/graphicsview.html . Study the example code and be sure that you understand how it works.

From the colliding mouse example code, you just need to add a fuzzy logic reasoner and some attributes to the mice. The intelligent mice behavior is described by a set of fuzzy rules. You must also specify these rules with corresponding membership functions.

How is the animation generated

Qt handles the animation steps as described in the previously mentioned graphics view framework. The animation is taken care of by Qt by using a timer. But between each animation step you must run your fuzzy reasoner to decide the behavior of each mouse. The fuzzy rule set must be parsed into an internal representation that the fuzzy reasoner can run in-between each animation step.

Detailed description of what to do and some hints

Since the example code just let a set of mice move around a cheese, you now shall let them compete for having control over the cheese area. The mice are all of different sizes, have different max movement speed and power (strength). The linguistic variables (attributes) in the fuzzy reasoning are **health, rate, distance and action**. Health may decrease when fighting with another mouse. Rate describes the size and power of a mouse. Distance is the distance to a competitor. Action is the the action you want to do (flee, no action, attack). You must define the membership functions of the fuzzy sets you want to use here.

Mouse behavior

A mouse has the following behavior:

- looks forward to see if there are any competitors. The view angle is 120 degrees and a mouse only worries about the two closest competitors.
  Hints: For each mouse in the scene:
  - Determin the distance to the others and evaluate the “power” of the two nearest and within view. Here you may use the built-in functions for item position: QGraphicsItem::scenePos(), collision and bounding rectangle of the scene: QGraphicsItem::sceneBoundingRect()
    - If the fuzzy evaluation results in the action:
      - **“flee”** then move in the opposite direction of the two competitors. This may result in detecting new enemies that must be handled in the next animation frame.
      - **“attack”** then determin which one of the two to attack. Note that the speed you approach the enemy comes from the fuzzy logic inference. Use random numbers to decide which one that attacks first and how the health is changed during fighting (the degree of how each attack hurts you)
      - **“no action”** just move in the current direction with the speed determined by the fuzzy reasoning.

- If two mice **collide**, i.e are close enough to fight each other, then:
    - Choose a random number of the time to prepare the fight action
    - Color the mouse red when it fights a competitor (after preparation time above)
    - When a mouse is hurt, reduce its health with a random number and show its health using color.

Rules and variables should be read from a file. Here is an example of a possible format:

```
define lingvar health: injured, good, execellent;
define lingvar rate:close,near,far;
………
define fuzzyset injured:triangle=[0,0,10,1,20,0];
define fuzzyset far:trapez=[0,0,10,1,20,0];


if (health is good) AND (rate is far) then action is noaction;
……….
……….
```

You should "compile" this file into a proper run-time structure in order to the fuzzy reasoning ☺ Hint: you could implement functions for computing the membership based on different shapes of the fuzzy sets (triangle, ..) as well as the the operators NOT, AND and OR.

Linear membership functions to implement: triangle, trapezoid, grade, reverse grade: (put in formulas and pictures here from Powerpoint).

Similary you need to handle the hedges (very, little, much, more….). The membership function can be reused in the Mamdani defuzzification part . How?

Subtasks in project:

1) Choose linguistic variables and their fuzzy set and define at least three fuzzy rules for mouse behavior.
2) Design a **run-time data structure** for the rules and instantiate the rules in 1). Note that the this part should be reused in 3d) so **no** hardcoding of rules :-) but make functions/methods that are general for other rule-sets.
3) Implement inference machine based on Mamdani
   a. Fuzzifaction (membership functions)
   b. Rule evaluation (with AND, OR, NOT)
   c. Mamdani aggregation and defuzzifaction
   d. **Parse** fuzzy set and input rules into the **run-time structure** in 2) (define lvar health:bad, good, execellent) define fset bad: triangle=[0,0,10,1,20,0]
   e. Add hedges
   f. Change c) to Sugeno infrerence
4) Make a short report and "explain" what you see in 20 runs
5) Demo
6) Code readability and structure