ALGAE

Programmierpraktikum II SS 2000

Clemens Bauer
cbauer@cosy.sbg.ac.at
Peter EisenmANN
peisen@cosy.sbg.ac.at
Thomas Frommwald
tfromm@cosy.sbg.ac.at
Daniel Kadir
dkadir@cosy.sbg.ac.at
Michael Liedlgruber
mliedl@cosy.sbg.ac.at

Project manager: Roland Schwaiger

August 8, 2000

Abstract

In our project for the course *Programmierpraktikum II* we simulate a multiple agent system. The agents are algaes which live in an aquarium. They are controlled via an artificial neural network that determines the actions of the agent based on the inputs it gets through the sensors an agent is equipped with.
Agents may reproduce either sexually or a-sexually, and we got a certain mutation rate so the descendants are not exactly like their parent.
Agents are able to spread pheromones to attract partners. Actions reduce the energy level of an agent, so it has to regain life force either through photo-synthesis or by consuming nutrition substrate. A light source resides on top of the aquarium and most of the substrate is at the bottom.

Contents

Chapter 1

Introduction

ALGAE is a followup system of elc (evolution, learning and communication), which is an implementation of a multi agent system (MAS) conceptualized and implemented by Roland Schwaiger and Konrad Lang. They investigate the evolution of wandering behavior of simple, reactive agents in a highly dynamic environment. Those simple agents wander based on the outcomes of feed forward ANNs (multi layer perceptrons).
The world where our simulation will take place is an two-dimensional aquarium. On the top resides a uniform light source and at the bottom there is some nutrition substrate. Algae (agents) live in the world between. They move around consuming light and substrate (to regain energy), and may produce pheromones to find reproduction partners. The reproduction cycle takes place either sexual through meiosis or a-sexual through mitosis. There is a certain mutation rate when new agents are "produced".
All actions an agent can perform except photosynthesis and substrate accomodation reduce the agent's energy level. Once it reaches zero, the agent dies.

The simulation may be configured in every detail by the user before it is started, and a huge amount of statistics is provided once a simulation ends.

Chapter 2

Explanation of used symbols

2.1 Simulation

- d Duration of the simulation in time-steps
- $worldx, worldy$ Horizontal and vertical dimensions of the aquarium
- $state_L$ Light state - `all the time, never, function`.
- i_L Light intensity at light source position
- d_L Light distribution function
- c_L Light cycle function
- a_S Total initial substrate amount
- d_S Substrate distribution function
- ren_S Substrate renewal rate
- dec_P Amount of time a pheromone particle will exist
- $randSeed$ This is the global randomseed value.
- t Timestep

2.2 Algaes

- ini_a Initial number of algaes
- ini_e Initial energy of an agent
- τ Transparency factor of algaes
- a An agent
- e Actual energy level of a

- x, y Position of a
- C An agent's chromosome
- m *Modified Miller Matrix*, used for the ANN

2.2.1 Sensors

- f_P Field of view of pheromone sensor (in degrees)
- r_P Range of pheromone sensor (in world units)
- f_L Field of view of light sensor
- r_L Range of light sensor
- f_S Field of view of substrate sensor
- r_S Range of substrate sensor
- f_B Field of view of bumper sensor
- r_B Range of bumper sensor

2.2.2 Actions

- e_M Necessary energy for movement
- e_I Energylevel necessary for mitosis
- e_E Energylevel necessary for meiosis
- e_P Energy used for pheromone output
- e_S Amount of energy gathered by consuming substrate
- e_L Amount of energy gathered by photosynthesis (at standard light intensity)

Chapter 3

World of ALGAE

3.1 The Simulation

Our task is to provide a simulation environment that sets the input to the ANN and performs the actions, shown by the graphical output. It also should take care of reproduction methods and the death of an agent when out of energy. At last the simulation should be visualized.
At the beginning a certain amount of agents is randomly initialized. This includes the position and the heading as well as the neural network associated with each agent. This is the point, where the aquarium (the concrete matrix) gets initialized too. This includes the substrate distribution and the light, which are both set to a starting value. The pheromone level is set to zero at each point in the matrix.
After this global initialization, the main simulation loop is started. In that loop each agent performs some actions, which result from influences of its neighborhood.

3.2 Aquarium

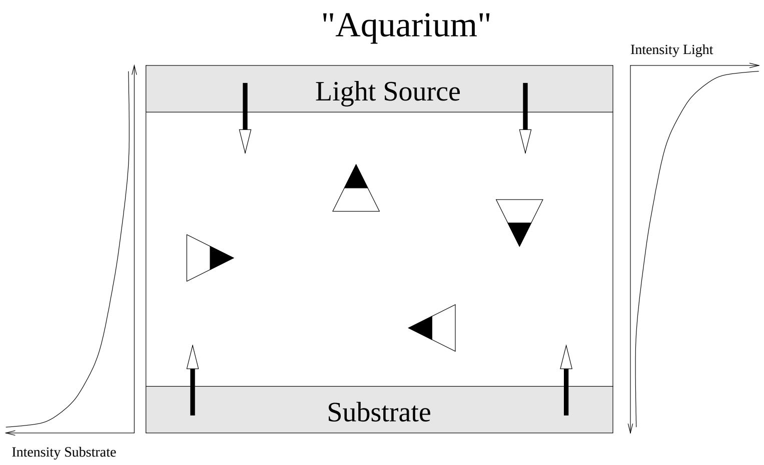


Figure 3.1: The world where our simulation takes place.

The aquarium has a two dimensional extension, specified by $(worldx, worldy)$. The borders on the left and right have a transparency factor of zero, i.e. no light may shine through the borders.

3.3 World

Basically, the "world" is responsible for controlling the experiment, some of its tasks are:

1. Set the output of the experiment in the GUI, i.e. show the agent, light distribution, substrate distribution, show details of the agents, s.a. energy level, genotype, phenotype.

2. Perform all calculations from one time-step t_i to time-step t_{i+1}, new agent positions[1], set sensor values of the agents[2]

3. Insert new agents in the world once reproduction has taken place.

4. Remove agents with e of zero (or below).

3.4 Light and Substrate

The light source is completely uniform, rays are sent from top to bottom in a straight line. The light source can be set to simulate a day/night cycle or another function, but may also be set to a specific value during the simulation, or even turned off completely by the user. Light has a higher intensity at the top of the aquarium than at the bottom. Per default,the intensity is modeled by the function 1/y, where y represents the vertical world axis, but an other function may be chosen by the user.
Agents reduce the light intensity for every spot below them, but they do not block the light completely due to a certain transparency value.
Similarly conditions apply for the substrate which diffuses into the water. We only have a certain amount of substrate particles which difuse into the water over time. The distribution of substrate is based on a mathematical function. If a substrate particle is consummated, it is rebuilt after a specific amount of time, until the supply is used up completely. At bottom, substrate is rebuilt faster than above because the source resides there.

Let us denote the light distribution by $d_L(x)$ and the substrate distribution by $d_S(x)$. An example for both distribution functions would be $d(x) = \frac{1}{x}$.
im Aquarium
In figure 3.2 the distribution of the substrate is given. In each location (x, y) the substrate has concentration 1 or 0. Once a substrate is consumed by an agent the substrate will be rebuilt according to the given $d_S(x)$. In the given figure we have chosen $d_S(x) = y^2$, for

[1] The agents perform their steps and request from the world their new positions, since only the world knows about the positions of the other agents. Furthermore, just the world knows about the concentrations (light, substrate, pheromone).

[2] This has to happen in the first phase of interval $[t_i, t_{i+1}]$ since all further actions of the agents rely on the sensors values.

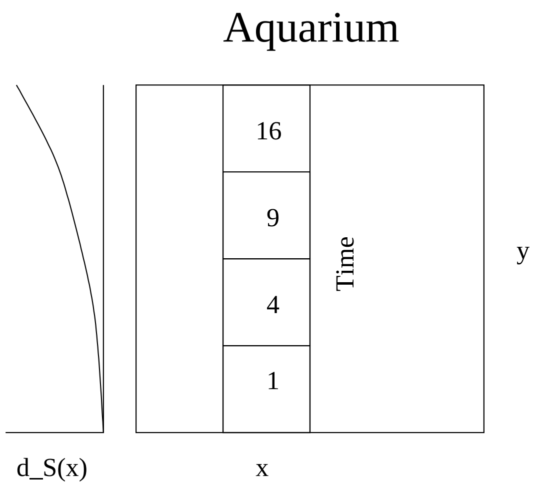


Figure 3.2: The concentration and distribution of the substrate, according to $d_S(x) = x^2$.

$y = 3$ we get $d_S(3) = 9$ which are the time-steps the system needs to rebuild the substrate concentration[3].
Next let's consider the light distribution $d_L(x)$. Consider that an agent a_i in spot (x, y) blocks part of the light for all the agents a_j in locations $(x, y'), y' < y$. The light intensity will degrade according to a certain transparency factor τ_i of agent a_i.

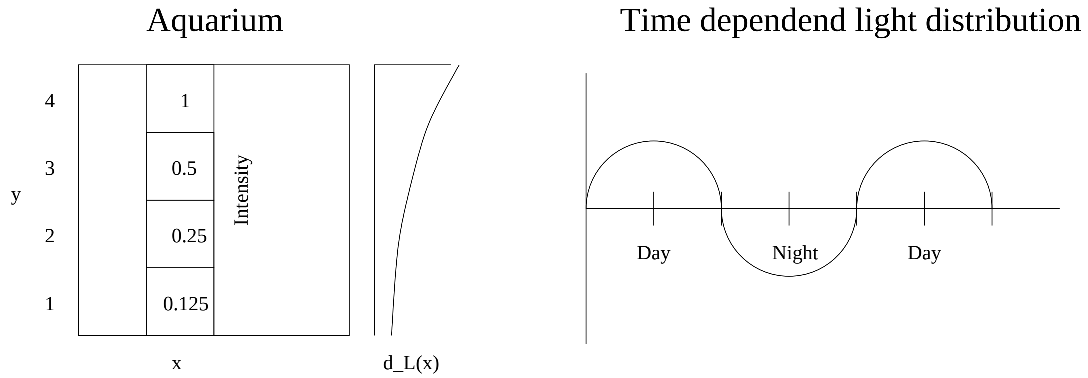


Figure 3.3: The concentration and distribution of the light, according to $d_L(x) = 1/x$.

In order to model day and night in the simulation $d_L(x)$ may depend on the time. As an example consider the sin function, as seen in figure 3.3. Further the time dependent modeling of the substrate concentration may be considered. The light intensity can also be changed manually by the user during the simulation.

3.5 Agents: Energy

Every agent is "born" with the same amount of energy. Energy is used for every action the agent may perform. If the energy level e reaches zero, the algae dies. Energy may be regained through photo-synthesis or substrate accomondation. It should be stated out

[3]In a future release of the simulation the substrate concentration in (x, y) only may be rebuilt if there is a substrate concentration in locations $\{(x, y')|\forall y' : y' < y\}$

clearly that our agents do not have a fixed life span. They live as long as they got an amount of energy greater than zero.

3.6 Agents: Sensors

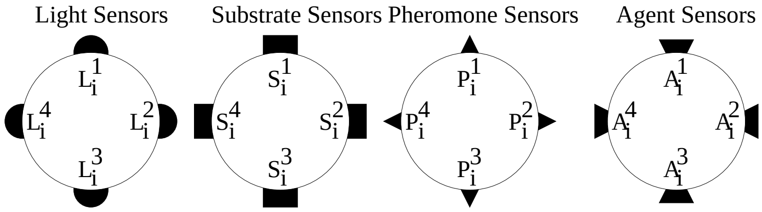


Figure 3.4: The various sensors of agents a_i.

To enable the agents to get information about their environment, we equip them with several sensors. An agent got four different sensors, where each points into the main directions forward, right, left and backward. Therefore, the agent gets 16 different sensor input (4 sensors * 4 directions) every time-step.
All sensors have a certain field of view (FOV) and a specific range. Due to the fact the strength of the input values from the various positions of the sensor directions differ, the agent should be able to detect where the source of signal is.

3.6.1 Light sensor

This sensor gets the summed light intensity of its FOV within its range.

3.6.2 Substrate sensor

This sensor gets the value of the substrate particles in its FOV. The value each particle adds to the sensor input depends on its distance.

3.6.3 Pheromone sensor

This sensor delivers the pheromon density in its FOV (Pheromons are released by the agents in order to find reproduction partners). The value each particle adds to the sensor input depends on its distance.

3.6.4 Bumper sensor

This sensor detects obstacles (other agents or the world's border) around the algae.

3.7 Agents: Actions

An agent is able to perform an arbitrary amount of *different* actions every time-step of the simulation. All of these actions consume/produce a specific amount of energy. Of course, the agent won't perform actions that would use a higher amount of energy than it currently has.

3.7.1 Movement

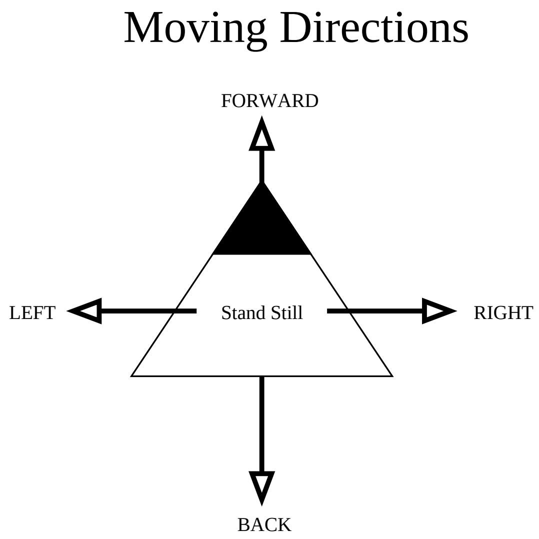


Figure 3.5: The possible moving directions of agent a_i.

Moving is probably the most important task an agent has to perform, in order to navigate to reproduction partners, towards substrate or higher light intensity. The agent may move one unit per time-step.
The agent always turns towards the new direction if necessary - it cANNot make some kind of sidestep.

3.7.2 Photo-synthesis

One way for an agent to regain energy is photo-synthesis. The agent directly converts the light intensity at its current location into life energy. Higher light intensity produces a higher amount of energy.

3.7.3 Substrate acommomdation

If an agent is at the same location as a substrate particle, it consumes the particle and regains a specific amount of energy. The substrate is rebuilt after a certain number of time-steps, depending on its y position.

3.7.4 Reproduction

If the energy level e of an algae reaches a certain amount the algae may reproduce either sexually through meiosis or a-sexually (mitosis).

- Meiosis, Pheromones
  Once an agent a_i has collected enough energy points it may reach a critical limit e_E for sexual reproduction. From that moment on the agent may produce pheromone in order to signal other agents its ability for sexual reproduction. The agent will release

the pheromone in its actual location[4]. The pheromone will be apparent in the spot for a given amount of time-steps, let's denote the amount of time by dec_P.

For the case e of an agent is higher than e_E and therfore it may start producing pheromone, we discussed several possibilities:

1. To distinguish between the pheromones of different agents we may introduce two genders, male and female. The two genders release different kinds of pheromone. As can be seen easily, we reduce the probability of reproduction significantly.
2. Another way to consider pheromone of different agents we may intro two agent types. The first type just releases pheromone and the second agent type just senses pheromone. Apply the argument above.
3. If one agent releases pheromone and meets another agent it will force the agent to reproduce, no matter if the other agent has an energy level greater than e_E. There is no way for an agent to distingiush between its own or other pheromone.
4. Finally, we introduce a flag for pheromone for each agent. With this method we get the possibility to differentiate between the pheromone of the various agents. Unfortunately not a biological plausible proposal.

In our simulation we will use on the third possibility: If one agent is willing to reproduce and meets another agent it will force the other one to reproduce, without taking care for the other's current energy level.

- Mitosis
  If an agent reaches an energy level e_I before it finds a partner for meiosis, it can reproduce itself a-sexually through the way of mitosis. This means the agent parts itself so we get two nearly identical individuals (they do not have exactly the same genotype because there is a certain mutation rate).

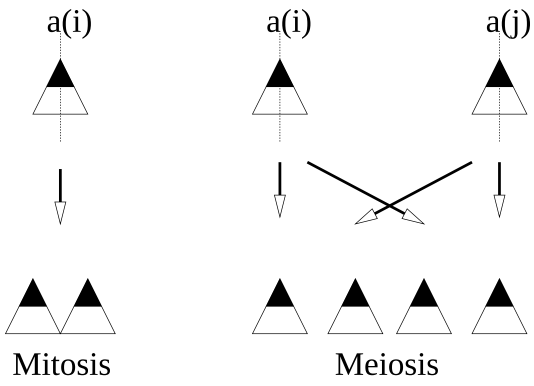


Figure 3.6: Reproduction of the agents a_i and a_j .

Figure 3.7: The ANN of agent a_i .

3.8 Agents: Neural Network

The various actions of an agent are partially determined by its mapping from input of the environment to output, i.e. reactions. This type of agent may be called reactive agent or stimulus/response agent. In our multi agent system we model this stimulus/response relationship by means of an artificial neural network – multi layer perceptron – (ANN). The input to the ANN consists of the sensor inputs and the e of the agent. Various hidden layers are used to perform transformations of the signals. Finally at the output level the calculated moving directions are shown.

Figure 3.8: The detail view of one neuron in the ANN.

[4] In future releases of the simulation we may consider some distribution function for pheromone release in the neighborhood of the agent

3.8.1 How the ANN works in detail

The following steps describe how the output of the ANN is calculated in detail:

1. First of all the input neurons of the ANN are set to their apropriate values. During this initialization the input neurons' output values are set to the sensors' values and the energy level. The output values of the outputneurons are reset to zero in this step too.

2. Now all neurons of the ANN are traversed to calculate the outputvalue for each of them. To simplify things a bit, the neurons can't be connected to themselves (no loops) and a connection from a neuron with a higher index to a neuron with a lower index isn't possible (except for connections from the neurons in the hidden layer to the neurons in the output layer). These few rules are important to avoid recursive calculations.
   To calculate the output for a neuron, the parent neurons for each neuron have to be found. Then the following formula is used to calculate the output of the current neuron j:

   $$o_j = tf\left(\sum_i w_{ij} o_i\right)$$

   The used variables:

   - o_j: This is the output for the current neuron j.
   - i: This is the number of parent neurons of neuron j.
   - o_i: This is the output of the parent neuron i.
   - w_{ij}: This is the weightvalue between neuron i and neuron j.
   - $tf(x)$: This is the transfer function used in the ANN. Currently for this transferfunction Cubic Splines are used.

3. Corresponding to the "The winner takes all" rule for the used ANN it's important now to determine which neuron has to strongest output (respectively the winner neuron).
   The index of the neuron with the highest outputvalue is then the final result of the ANN.

3.9 Evolutionary Cycle and Reproduction

As pointed out above we want to build our system based on the steady state approach, i.e. no generations are built and definite end is proposed but reproduction occurs on the fly. Basically all relevant information has to be encoded into various chromosomes (bit-strings). In our simulation we want to use three chromosomes per agent.

- The first chromosome C_1 encodes the topology (neurons and connections) of the ANN in a Modified Miller Matrix (MMM) mANNer. Basically the main diagonal of the connectivity matrix where an entry $m_{ii} = 1$ states that the neuron is present. Whereas entry $m_{ii} = 0$ states that the neuron is not present. If the entry is not in the main diagonal m_{ij}, where $i <> j$, then the entry states the existence of the connection from neuron i to neuron j.
- The second chromosome C_2 encodes the weights of the connections and the biases of the neurons, two Bytes are used for encoding.
- The third chromosome C_3 encodes the cubic splines, where each point of the spline is encoded with its x and y coordinate, two Bytes respectively.

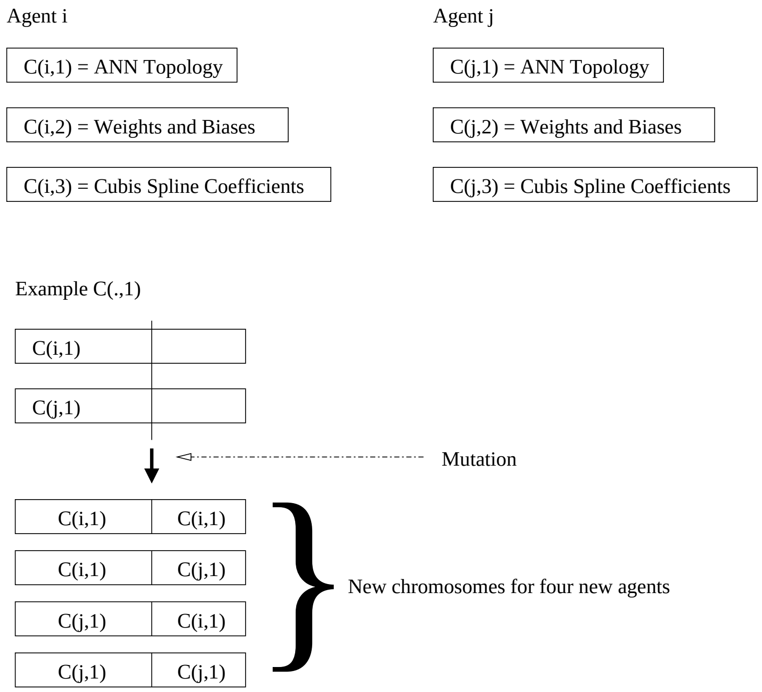


Figure 3.9: The details for crossing over between chromosome C_1 of agent a_i and a_j.

In order to reproduce we have to consider two different cases, meiosis and mitosis. In figure 3.9 we give an example for meiosis. In the mitosis case we duplicate the chromosomes and perform the same steps as in the meiosis case.

3.9.1 The birth process of a new agent in detail

In ALGAE an agent can be born in three different ways: Random initialization, mitosis and meiosis. The first method is used at startup to fill our aquarium. The latter two ones are used only for reproduction.

The randomly initialized agent

The randomly initialized agent is only used at startup of a simulation. Each agent is randomly placed into the world and gets a random ANN. To get different agents with different behaviours, we use random numbers as random seeds for the different agents.

Mitosis

Mitosis is the asexual way an agent could be born. If a new agent is born this way, the new agent will get all its data copied 1:1 from the agent, who initiated the mitosis.
But since we don't want to get exact copies, we at least do some slight mutations (depending on the mutation rate) on the ANN of the new agent (the details about mutation can be read in the next subsection).

Meiosis

Meiosis is the sexual way an agent could be born and involves always two agents. The ANN of an agent born this way is a crossing over of the two parents' ANNs. After the crossing over is done, an at least slight mutation (depending on the mutation rate) is done on the new agent's ANN (the details about crossing over can be read in the next subsection).

3.9.2 How mutation and crossing over work in detail

Since in ALGAE the genotype is represented by simple bitstrings, mutation and crossing over are simply modifications of these.

Mutation

During a mutation of a bitstring, some of the bits in this bitstring simply get switched. Which bits get switched depends on the mutationrate. How many bits of a bitstring get switched depends on the mutationrate too.
The mutation is done in the following steps:

1. For each bit a random number (double) is calculated.
2. This random number is then compared against the used mutationrate and if the random double value is lower than the mutationrate the current bit is switched.
3. Steps 1 and 2 are repeated until the tail of the bitstring is reached.

Crossover

Crossing over needs always two bitstrings $b1$ and $b2$ as source. How many bits the new bitstring gets from $b1$ or from $b2$ depends on the crossingover rate
The crossingover is done in the following steps:

1. For each bit a random number (double) is calculated.
2. This random number is then compared against the used crossingover rate. If the random double value is lower than the crossingover rate an internal flag, showing which bitstring to use as source, is toggled.
3. If this internal flag equals 1, the value for the current bit is taken from $b1$. Otherwise the bit is taken from $b2$.
4. Steps 1 and 2 are repeated until the tail of the bitstring is reached.

3.9.3 The different ways an agent may die

During a simulation an agent may die in three different ways:

- The agent runs out of energy because of actions he made. An agent does only try to reproduce if he has enough energy. But since putting pheromones and movement consume energy too, the agent may die if there is not enough energy available.
- The agent runs out of energy because he got involved in meiosis, although he has not enough energy. And although in that case one of the parents dies, the new child stays alive.
- If a new agent should be born, but there is no place around the parents of that agent to place him at, then the new agent will die.

Chapter 4

ALGAE File Format Specifications

The formats in which information in ALGAE will be saved to disk.

4.1 Configuration files

4.1.1 Introduction

The ALGAE-settings for different simulation environment presets are saved using the XML file-format.
This format was chosen because of the flexibility and the fact, that one doesn't have to read or edit any binary file-format. Therefore the files can easily be created or modified with any text editor. The restrictions within the creation or modification process of such a settings file are donated in the file *ElementDefinitions.DTD*. This file holds some simple guidelines for creating or modifying the ALGAE-settings and shouldn't be changed unless the format is to be extended or modified.

4.1.2 Section `Settings`

The settings-block is the main section and holds all subsections with information about the initial simulation environment. This time only one Settings-block per file is used, but it is also possible for future development of the file-format to extend the format in such a way, that more than one settings-block could be stored in one file.
This time, for each preset (or initial ALGAE-settings) a separate file is created.

The Settings-block has the following attributes:

- Version - this attribute stores the file-format version
- Creator - the program (editor) which created this file (in most cases this would be ALGAE itself)
- Name - the name of the settings-file
- Date - date and time of file-creation (or file-modification) in the format DD.MM.YY HH:MM:SS

7

4.1.3 Section `Simulation`

This section holds some general information about the initial state of a simulation.

- Duration d
- Width $worldx$
- Height $worldy$
- Light_enabled $state_L$, may be

  `all the time`

  `never`

  `function`
- Light_intensity i_L
- Light_distribution_func d_L
- Light_cycle_func c_L
- Substrate_amount a_S
- Substrate_renew_rate ren_S
- Pheromone_dec_rate dec_P
- Randseed $randSeed$

4.1.4 Section `ALGAES`

This section stores information about the algaes and their initial states.

- Initial_algaes ini_a
- Initial_energy ini_e

4.1.5 Section `Sensors`

This section stores information about the algaes' sensors and their initial states.

- Pheromone_fov f_P
- Pheromone_range r_P
- Substrate_fov f_S
- Substrate_range r_S

- Light_fov f_L
- Light_range r_L
- Bumper_fov f_B
- Bumper_range r_B

4.1.6 Section `Action`

The attributes of this section store information about the amount of energy which is used respective collected by specific actions.

- Energy_movement e_M
- Energy_mitose e_I
- Energy_meiose e_E
- Energy_pheromone_output e_P
- Energy_substrate e_S
- Energy_light e_L

4.2 Statistic Files

4.2.1 Introduction

An ALGAE statistics file stores very detailed informations about a simulation process. *NOTE: Statistics are only written if the program was started in Batch-Mode.*

4.2.2 File Structure

- Section `Setting`

  Subsection `Simulation`

  Subsection `Agent #0`

  Subsection `Agent #1`

  .

  .

  Subsection `Agent #n`

- Section `Step 1`

  Subsection `General Stats`

  Subsection `Agent #1`

  Subsection `Agent #2`

.

.

Subsection `Agent #n`

- Section `Step 2`

    Subsection `General Stats`

    Subsection `Agent #1`

    Subsection `Agent #2`

    .

    .

    Subsection `Agent #n`

- Section `Step n`

    Subsection `General Stats`

    Subsection `Agent #1`

    Subsection `Agent #2`

    .

    .

    Subsection `Agent #n`

- Section `Summary`

    Subsection `General Stats`

    Subsection `Agent #0`

    Subsection `Agent #1`

    .

    .

    Subsection `Agent #n`

Following: The different sections described more detailed

4.2.3 Section `Settings`

This section stores most of the information defined in the settingsfile used for the simulation which generated this statistics file.
These settings are stored in the following subsection:

Subsection `Simulation`

- steps d(Integer value)
- random_seed (Integer value) - This is the number which is used as an initialization value for the random number generator. Since this number is used forthroughout the whole simulation, the simulation may be repeated again and again, with the same results every time)
- world_width $worldx$ (Integer value)
- world_height $worldy$ (Integer value)
- neurons_used (integer value) - Total number of neurons used in the ANNs of the agents
- input_neurons_used (integer value) - Number of input neurons used for an ANN of an agent (this time the input values for an ANN of an agent are his energylevel and the 16 values of his sensors).
- output_neurons_used (integer value) - Number of output neurons used for an ANN of an agent.
- initial_agents ini_a (integer value)
- initial_energy ini_e (float value)
- substrate a_S (integer value)
- substrate_renew_rate ren_S (float value)
- pheromone_decrease_rate dec_P (float value)
- pheromone_sensor_fov f_P (integer value)
- pheromone_sensor_range r_P (integer value)
- substrate_sensor_fov f_S (integer value)
- substrate_sensor_range r_S (integer value)
- light_sensor_fov f_L (integer value)
- light_sensor_range r_L (integer value)
- bumper_sensor_range r_B (integer value)

Additionally this section includes information for each agent, initialized at the startup of the simulation.
As a consequence this section stores n subsections of the following type (where n is the number of initial agents)

Subsection `Agent #i`

- agent_index (integer value) - This is the index of the index in the list of all agents in the simulation.
- position_x x (integer value)
- position_y y (integer value)
- energy_level e (float value) - This is the energy-level of an agent. In this section it's always the same value as initial_energy ini_e in the SUBSECTION SIMULATION since we see the initial state of an agent here

4.2.4 Section `Step i`

A section of this type is stored for each step of the simulation. The following subsection is stored once per step and stores some general information of step i.

Subsection `General Stats`

- agents_alive (integer value)- How many agents were alive at the current step
- agents_died (integer value) - How many agents died at this step
- mitosis_count (integer value) - How often mitose occured at this step (indicates the number of new born agents in this step, too, when taken in addition with meiosis_count)
- meiosis_count (integer value) - How often did meiose occur in this step (indicates the number of new born agents in this step, too, when taken in addition with mitosis_count)
- light_intensity i_L (float value)
- substrate_amount a_S (float value)
- substrate_consumed (integer value) - How much substrate was consumed by the agents during this step
- substrate_renewed (integer value) - How much substrate was renewed in this step
- pheromone_dissolved (integer value)- How many pheromone particles dissolved in this step
- pheromone_renewed (integer value) - How much pheromone was renewed in this step.
- light_consumed (integer value) - How much light was consumed by the agents during this step.

The following n sections of the following type store informations about the agents and their state in step i.
NOTE: Only information of living agents is stored here.

Subsection `Agent #a`

- agent_index (integer value) - Same as in subsection Subsection `Agent #i` in Section `Settings`
- heading (integer value) - This value stores the direction, the agent is currently heading towards
- position_x x (integer value), position_y y (integer value) - Same as in subsection `Agent #i` in section `Settings`, but updated during actions made by the agent
- energy_level e (integer value) - Same as in subsection `Agent #i` in Section `Settings`, but updated during actions made by the agent.
- ANN_output (integer value) - This value is the numer of the "winner" (e.g. the one with the highest value) neuron of the ANN. Respectively it's in most cases the same value as then heading (except for zero: the agent stands still, but shows into a direction not equal zero.
- mitosis (boolean value) - Did the agent do mitosis during this step?
- meiosis (boolean value) - Was the agent involved into meiose during this step?
- used_light (boolean value) - Did the agent consume light in this step?
- used_substrate (boolean value) - Did the agent consume substrate in this step?
- birth (boolean value) - Was this the "birthdate" of the agent?
- sensors_values - These are the values for the agent's sensors in step i. They are stored in a single line. The values are stored in the following order:

  bumper sensor values - one float value per sensor (4 values)

  pheromone sensor values - one float value per sensor (4 values)

  substrate sensor values - one float value per sensor (4 values)

  light sensor values - one float value per sensor (4 values)

4.2.5 Section `Summary`

This section is a short summarization of the agents who lived and died during the simulation.
The first subsection stores some general information about the ALGAE population.

Subsection `General Stats`

- dead_agents (integer value) - How many agents of the entire population died?
- living_agents (integer value) - How many agents of the entire population are still alive at the end of the simulation?

Then a short summary for each agent follows (it doesn't matter wether an agent is dead or alive, this list contains the whole population).
The list consists of n subsections, where n is the total number of agents born during the simulation.

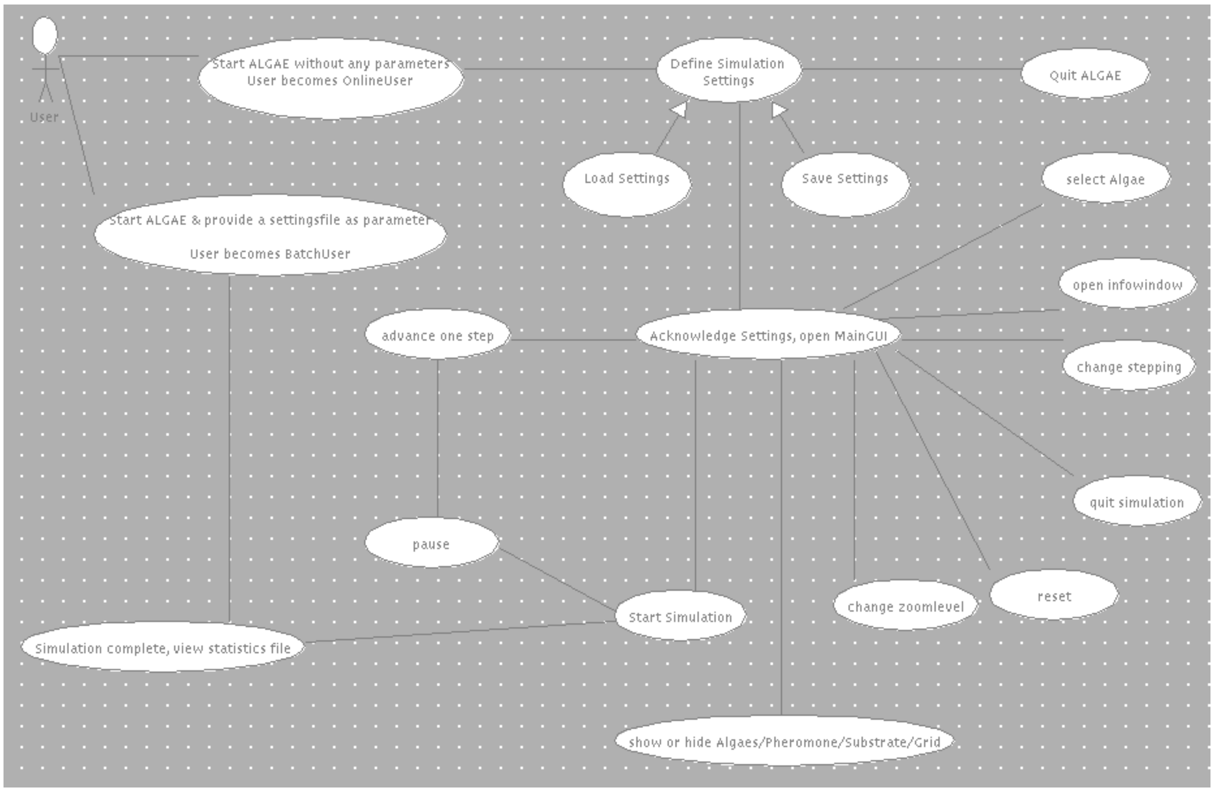
Subsection `Agent #i`

- alive (boolean value) - Indication whether the agent is dead or if he survived the whole simulation
- birth_step (integer value) - During which timestep the agent was born
- alife_steps (integer value) - This is the time the agent was alive
- mitosis_count (integer value) - This counter stores how often the agent initiated a mitose
- meiosis_count integer value) - This counter stores how often the agent was involved into a meiose
- used_light (float value) - How much light did the agent consume?
- used_substrate (float value) - How much substrate did the agent consume?
- number_of_parents (integer value) - If the agent was initialized at the beginning of the simulation, this value is 0. Otherwise, if the agent was born during mitosis/meiosis this value is 1/2
- parent_1 (integer value) - This parameter is only stored, if the agent has at least one parent. In that case it specifies the first parent (in case of mitosis the one and only parent)
- parent_2 (integer value) - This parameter is only stored, if the agent has two parents. In that case it specifies the second parent (only used in case of meiosis)
- number_of_children (integer value) - This value stores the number of children of agent i. If the agent never was involved into mitosis or meiosis, this number is zero
- children_list - This is a list of number_of_children integer values. It stores the index for each child of agent a_i. This list is only stored if number_of_children is greater than zer
- phenotype_miller_matrix (2-dimensional array of integer values) - This is a 2-dimensional with a size of n*n, where n is the number of total neurons used in an ANN of an agent. Ir is stored in n lines with n values (1 or 0) per line
- phenotype_weight_matrix (2-dimensional array of float values) - This is a 2-dimensional with a size of n*n, where n is the number of total neurons used in the ANN of an agent. It is stored in n lines with n values per line
- phenotype_spline_matrix (2-dimensional array of float values) - This is a 2-dimensional with a size of 2*n, where n is the number of x/y-pairs used for the cubic splines in the ANNs

Chapter 5

Use Cases and Class Diagrams

5.1 Use Cases of the Program



These are the possibilities that a user of ALGAE has during the simulation. The first decision is wether (s)he wants to act as a `batch`- or an `online`-user.

- A `Batch User` starts Algae providing a filename of a pre-defined settingsfile as parameter. The simulation starts without any further inputs. The batch user doesn't see the graphical output or any part of the user interface (GUI) of the program.

  Once the simulation ran for a certain number of timesteps (defined in the settingsfile), it ends and the user may view the statistical output the simulation produced. The statistical file generated got the name of the used settingsfile with *.stat* at the end - for example `set1.asf.stat`

- The `Online User` is able to watch the whole simulation in real-time. (S)he can enter any settings and may load or save a configuration file. Once the simulation is started, the main GUI is visible and the user can manipulate the sim in various ways - see the graph above and the next chapter

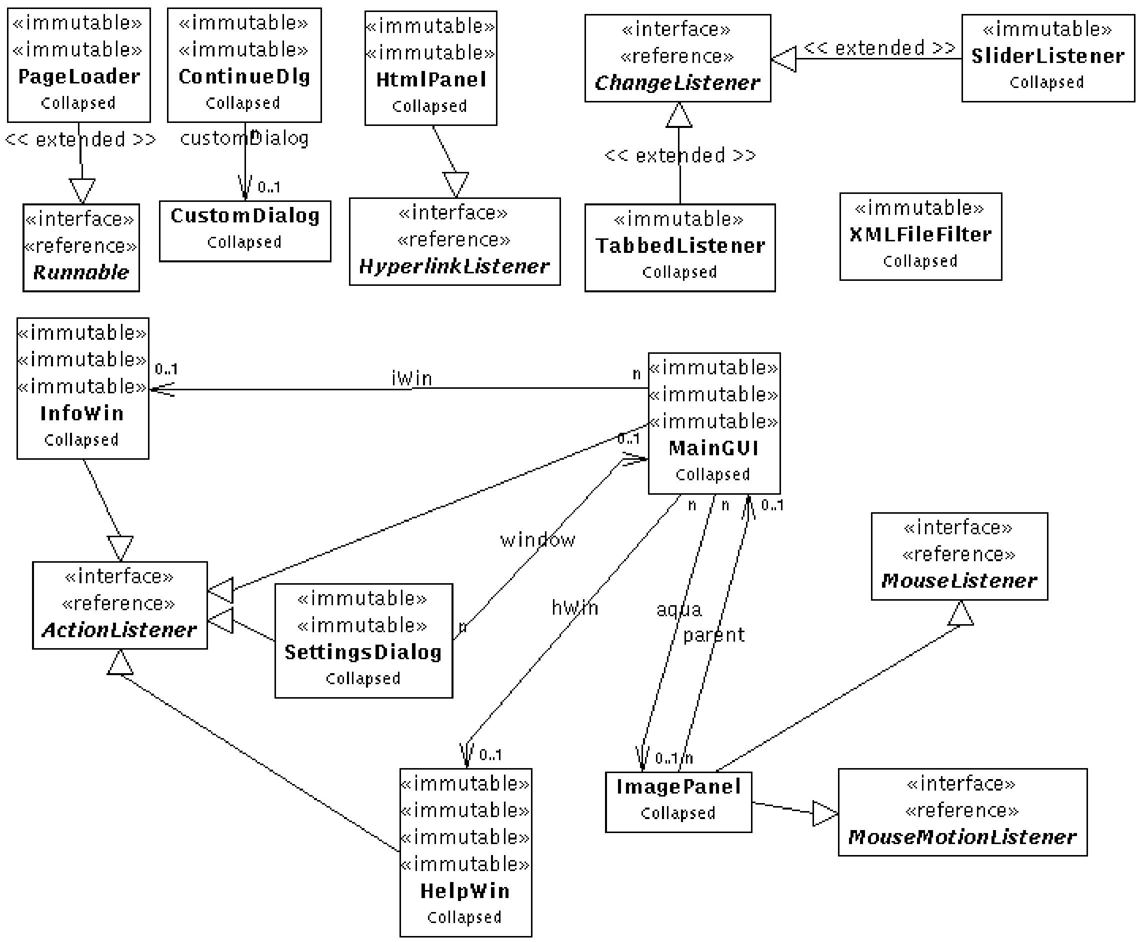
Class Diagrams

Attributes and methods of the classes are shown in a standardized schemata:

Type Attribute *AttributeName : DataType*
Type Method *MethodName (Parameter1: ParameterType1, Parameter2...) : ReturnDataType*

If *Type* is omitted, the Attribute/Method is `public`. Other possible types are `private, package, protected`.
Classes may optional be `immutable`.

5.2 Package `GUI`



This package represents every function that the program's user interface provides.

5.2.1 Class `CustomDialog`

The class CustomDialog is a derivation of JDialog, which implements a dialog with custom controls. This dialog provides a label for status informations, a textfield for entering a number and a button to confirm the entered value (although pressing *Enter* is enough too).

- Private Attribute `retVal:Integer`
  This is the return value of the dialog.

- Private Attribute `typedText:String`
  The text entered into the dialog. It is used for validation for a correct entered value.

- Method `actionPerformed(e:ActionEvent):Void`
  The actionlistener for the CustomDialog. All actions performed within the CustomDialog are checked and handled in here.

- Method `customDialog(aFrame:Frame,steps:Integer,parent:ContinueDlg)`
  The constructor for CustomDialog.

- Method `getResult():integer`
  This method returns the value entered in the CustomDialog. In case of an abnormal termination of the CustomDialog, the return value equals -1.

- Method `getValidatedText():String`
  This method returns the entered text as a String-object.

- Method `propertyChange(e:PropertyChangeEvent):Void`
  The main event dispatcher for the CustomDialog.

- Method `windowClosing(we:WindowEvent):Void`
  Method called, if the CustomDialog is to be closed.

5.2.2 Immutable Class `SettingsDialog`

The class SettingsDialog is a derivation of JFrame, which provides a dialog to enter all settings and initialization values for a single simulation.

- Package-Attribute `presets:String[] = {"-----"}`
  This is the list of ALGAE-settingsfiles found during startup.

- Method `actionPerformed(e:ActionEvent):Void`
  The main event dispatcher for the SettingsDialog.

- Method `addASFFilesToCombo():Void`
  This method checks the current working directory for ALGAE settingsfiles and adds them (if any found) to the presets-list.

- Method `dialogImportSettingsFile():Void`
  Call the settingsfile-import with a JFileChooserDialog.
- Method `dm(s:String):Void`
  This method is used for debugging-purposes only. It throws a message out to the console.
- Method `exportActionSettings(xmldoc:XmlDocument,XML>ElementNode):Void`
  Export the actions-settings out to an ALGAE-settingsfile.
- Method `exportAlgaesSettings(xmldoc:XmlDocument,XML:ElementNode):Void`
  Export the agents-settings out to an ALGAE-settingsfile.
- Method `exportSensorSettings (xmldoc:XmlDocument,XML: :ElementNode):Void`
  Export the sensors-settings out to an ALGAE-settingsfile.
- Method `exportSimulationSettings(xmlDoc:XmlDocument,XML:ElementNode):Void`
  Export the simulation-settings out to an ALGAE-settingsfile.
- Method `exportSettingsFile():Void`
  Export all settings to an ALGAE-settingsfile.
- Method `getSettingsVar():Settings`
  Return a variable of type Settings, which contains all settings specified in the dialog.
- Method `importActionsSettings(xn:org.w3c.dom.Node):Void`
  Import the actions-settings from an ALGAE-settingsfile.
- Method `importAlgaesSettings(xn:org.w3c.dom.Node):Void`
  Import the agents-settings from an ALGAE-settingsfile.
- Method `importSensorSettings(xn:org.w3c.dom.Node):Void`
  Import the sensors-settings from an ALGAE-settingsfile.
- Method `importSimulationSettings(xn:org.w3c.dom.Node):Void`
  Import the simulation-settings from an ALGAE-settingsfile.
- Method `importSettingsFile(s:String):Void`
  Import all settings from an ALGAE-settingsfile.
- Method `isPreset(s:String):Boolean`
  Checks wether a specific filename has the right extension to be regarded as an ALGAE-settingsfile.
- Method `makeActionsPanel():void`
  Build up the actions-settings panel.
- Method `makeAlgaesPanel():Void`
  Build up the agents-settings panel.
- Method `makeSensorsPanel():Void`
  Build up the sensors-settings panel.

- Method `makeSimulationPanel():Void`
  Build up the simulation-settings panel.

- Method `msgErr(S:String):Boolean`
  This method is used to bring up an error dialog with a specific error message.

- Method `settingsDialog()`
  The constructor for the SettingsDialog.

- Method `storeSettingsToVar():Settings`
  This method stores the settings from the dialog into a variable of type Settings.

- Method `verifySettings():Boolean`
  This method verifies all settings to be correct.

- Method `windowClosing(e:WindowEvent):Void`
  This is called, when the dialog is being closed.

5.2.3 Class `ImagePanel`

The class ImagePanel is a derivation of JPanel, which provides all drawing functions used in the simulation. These actions are collected and when the ImagePanel is updated, all drawing actions are performed on the canvas inherited by the parent class JPanel.

- Package-Attribute `algaePos:Integer = null`
  This is a 2-dimensional array storing the positions of the agents positions to draw.

- Package-Attribute `lightPos:Integer = null`
  This is a 2-dimensional array storing the positions of the different light fields and intensities to draw.

- Package-Attribute `pheroPos:Integer = null`
  This is a 2-dimensional array storing the positions of the pheromones and their intensities.

- Package-Attribute `subsbstratePos:Integer = null`
  This is a 2-dimensional array storing the positions of the substrates and their intensities.

- Package-Attribute `doubleBuffer:Boolean = true`
  This flag is set when it's possible to use doublebuffering.

- Package-Attribute `drawGrid:Boolean = true`
  This flag is set if drawing the grid is desired.

- Package-Attribute `firstTime:Boolean = true`
  A flag, which is set, if it's the first time the doublebuffer (when enabled) is used (time of creation of doublebuffer).

- Package-Attribute `height:Integer`
  Height of the aquarium.

- Package-Attribute `width:Integer`
  Width of the aquarium.
- Package-Attribute `mouseX:Integer = -1`
  Position along the x-axis within the ImagePanel.
- Package-Attribute `mouseY:Integer = -1`
  Position along the y-axis within the ImagePanel.
- Static Package-Attribute `scale:Float = 0.0f`
  Scale used for displaying the world.
- Static Package-Attribute `scaleHalf:Float = 0.0f`
  Experimental scale value (not used yet).
- Package-Attribute `selX:Integer = -1`
  x-Position of selected grid element (measured in grid squares).
- Package-Attribute `selY:Integer = -1`
  y-Position of selected grid element (measured in grid squares).
- Package-Attribute `selected:Integer = -1`
  Number of selected agent (-1 if none selected).
- Package-Attribute `x:Integer`
  Temporary variable for drawing world elements.
- Package-Attribute `y:Integer`
  Temporary variable for drawing world elements.
- Method `clearAList():Void`
  Method used to clear the list of agents to view (in case there is no agent in the world).
- Method `clearLList():Void`
  Method used to clear the list of light elements to view (in case there is no light in the world).
- Method `clearPList():Void`
  Method used to clear the list of pheromones to view (in case there is no pheromone in the world).
- Method `clearSList():Void`
  Method used to clear the list of substrate to view (in case there is no substrate in the world).
- Method `getGridX():Integer`
  Get x-component of mouse position within the aquarium (measured in grid squares)
- Method `getGridY():Integer`
  Get y-component of mouse position within the aquarium (measured in grid squares)

- Method `imagePanel(par:MainGUI)`
  The constructor of an ImagePanel.
- Method `mouseClicked(e:MouseEvent):Void`
  Eventhandler for mouse-click events.
- Method `mouseDragged(e:MouseEvent):Void`
  Eventhandler for mouse-dragging events (not used).
- Method `mouseEntered(e:MouseEvent):Void`
  Eventhandler which is called whenever the mouse enters the aquarium.
- Method `mouseExited(e:MouseEvent):Void`
  Eventhandler which is called whenever the mouse exits the aquarium.
- Method `mouseMoved(e:MouseEvent):Void`
  Eventhandler which is called whenever the mouse is moved within aquarium.
- Method `mousePressed(e:MouseEvent):Void`
  Eventhandler which is called whenever the mouse is pressed within aquarium.
- Method `mouseRealeased(e:MouseEvent):Void`
  Eventhandler which is never used, but must be defined.
- Method `paintAlgae(marked:Integer,pos:Integer):Void`
  Paint all agents listed in algaePos.
- Method `paintAll():Void`
  This method is called to redraw all the contents of the aquarium.
- Method `paintComponent(g:graphics):Void`
  Overridden method of JPanel. Used for redrawing the component.
- Method `paintGrid():Void`
  Paint the grid (if enabled) with specified scale.
- Method `paintLight(pos:Integer):Void`
  Paint the light (if enabled) using lList.
- Method `paintPheromone(pos:Integer):Void`
  Paint all pheromones listed in pList.
- Method `paintSubstrate(pos:Integer):Void`
  Paint all the substrate listed in pList.
- Method `paintRefresh():Void`
  Update the contents of the imagepanel.
- Method `setAlgae(sel:Integer,pos:Integer):Void`
  Set the list of agents for later drawing. The index of the selected agent (if any) is speficied too (index into aList).

- Method `setLight(pos:Integer):Void`
  Set the list for light for later drawing.

- Method `setPheromones(pos:Integer):Void`
  Set the list of pheromone for later drawing.

- Method `setSubstrates(pos:Integer):Void`
  Set the list of sustrate for later drawing.

- Method `turnOnOffDB(dbswitch:Boolean):Void`
  Method used to turn on/off the usage of the doublebuffer.

- Method `turnOnOffGrid(gSwitch:Boolean):Void`
  Method to turn on/off the grid.

- Method `update(g:Graphics):Void`
  Update the contents of the imagepanel.

5.2.4 Immutable Class `MainGUI`

This class is a derivation of JFrame and is the main window, where the simulation takes place. The user is able to change display settings, get informations about the current simulation state and watch the simulation in progress.

- Package-Attribute `alreadyDrawing:Boolean = false`
  Flag showing if the MainGUI is already redrawing the contents of the ImagePanel.

- Package-Attribute `aquaScale:Float = 1.0f`
  This is the scale used for drawing the world in ALGAE (this value is sent to the ImagePanel and then used for drawing).

- Package-Attribute `dispAlgaes:Boolean = true`
  Flag showing wether agents should be drawn or not.

- Package-Attribute `dispPhero:Boolean = true`
  Flag showing wether pheromone should be drawn or not.

- Package-Attribute `dispSubstrate:Boolean = true`
  Flag showing wether substrate should be drawn or not.

- Package-Attribute `drawGrid:Boolean = true`
  Flag showing wether the grid should be drawn or not.

- Package-Attribute `drawing:Boolean = true`
  The same as alreadyDrawing.

- Package-Attribute `running:Boolean = false`
  This flag shows if the simulation is running.

- Attribute `selection:Integer = -1`
  This is the number of the selected agent (if none selected this value is set to -1).

- Package-Attribute `simuPaused:Boolean = false`
  This flag shows, if the simulation is currently paused.

- Package-Attribute `simulationDelay:Integer = stepping-1`
  This is used for the timer, which is controlling the simulation.

- Package-Attribute `stepCount:Integer = 1`
  This value is decreased every simulation step. When this value equals 0, the world is redrawn and the value is set to stepping.

- Package-Attribute `stepping:Integer = 1`
  The number of steps, which are simulated, before the next screen update occurs.

- Method `actionPerformed(e:ActionEven):Void`
  The main event dispatcher for MainGUI.

- Method `addCBS():Void`
  Method to add the checkboxes for display options to the GUI.

- Method `addCanvas():Void`
  This method adds the imagepanel (the aquarium) to the GUI.

- Method `addMenuItems:Void`
  Inserts all needed menuitems into the menu.

- Method `addSliders():Void`
  This method adds the slider for speed controlling to the GUI.

- Method `addToolBar():Void`
  This method adds the toolbar to the GUI.

- Method `addToolBarButtons():Void`
  Adds the buttons to the toolbar.

- Method `likeToContinue(doneSteps:Integer):Integer`
  This method is used to determine, wether the user likes to continue the simulation, after it has finished.

- Method `mainGUI(set:Settings)`
  The constructor.

- Method `paintAll(w:WorldState,onlyStatus:Boolean):Void`
  This method is used to redraw the aquarium and its contents.

- Method `paused():Boolean`
  Method returning wether the simulation is paused or not.

- Method `stopped():Boolean`
  Method returning wether the simulation is stopped or not.

- Method `selected():Integer`
  Method returning the number of the selected agent.

- Method `setNewAreaDimensions():Void`
  In case of zooming in/out the GUI has to recalculate the dimension of the scrollpane.

- Method `setState(stat:String):Void`
  This method sets the state of the simulation (paused, running,...).

- Method `setStatus(stat:String,light:Float,subAm:Float,numAlg:Integer, step:Integer,stepsToDo:Integer):Void`
  With this method the content of the statusbar is updated.

- Method `setupGUI():Void`
  This is the method which calls all other methods for building up he GUI.

- Method `viewAlgae():Boolean`
  Method returning wether the agents should be drawn or not.

- Method `viewPheromone():Boolean`
  Method returning wether the pheromone should be drawn or not.

- Method `viewSubstrate():Boolean`
  Method returning wether the substrate should be drawn or not.

- Method `windowClosing(e:WindowEvent):Void`
  This method is called when the window is being closed.

5.2.5 Immutable Class `SliderListener`

This little class is used, to get notified if a slider component in the GUI is changing its state.

- Method `stateChanged(e:ChangeEvent):Void`
  This method is called every time the position of the speed slider is changed.

5.2.6 Immutable Class `ContinueDlg`

The `ContinueDlg` is a derivation of CustomDialog. It provides the functionality needed if a simulation is finished and the program asks the user wether he wants to continue the simulation or not.

- Method `continueDlg(frame:JFrame,donwSteps:Integer)`
  The constructor for the ContinueDlg.

- Method `getEnteredValue():Integer`
  This method returns the number of steps to append to the finished simulation (if a valid value entered).

5.2.7 Immutable Class `HelpWin`

This class is used to display a help window with a HTML help.

- Method `actionPerformed(e:ActionEvent):Void`
  The handler for any action performed within the help window.
- Method `helpWin(s:String)`
  The constructor for the HelpWin.
- Method `windowClosing(e:WindowEvent):Void`
  Method called when the help window is being closed.

5.2.8 Immutable Class `HtmlPanel`

The class HtmlPanel is derivated from JPanel and provides all functions needed in order to display a HTML file.

- Method `htmlPanel(fileName:String)`
  Constructor for the HtmlPanel.
- Method `hyperLinkUpdate(e:HyperlinkEvent):Void`
  This is the notification method, if a hyperlink within the HtmlPanel is selected.
- Protected Method `linkActivated(u:URL):Void`
  This method follows a link in a hyperlink.

5.2.9 Immutable Class `XMLFileFilter`

This class is used to decide wether a selected file in the open dialog of the SettingsDialog should be regarded as a valid selection or not. Depending on the validation it's accepted or not.

- Method `accept(f:File):Boolean`
  Method which checks wether a selected file should be accepted or not.
- Method `getDescription():String`
  Returns a short description of the fileextension ASF.

5.2.10 Immutable Class `InfoWin`

This class is a derivation of JFrame and is used to display informations about a specific agent (the selected one).

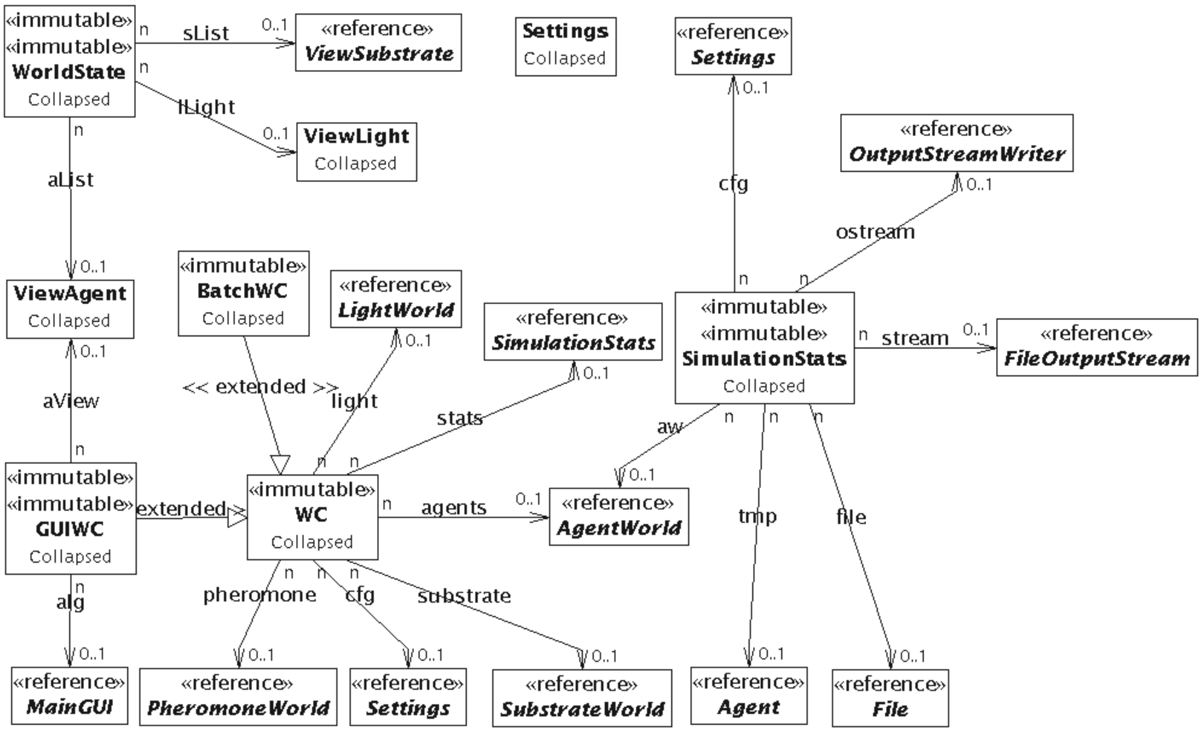
- Method `actionPerformed(e:ActionEvent):Void`
  The handler for any action performed within the info window.
- Method `clearValues():Void`
  If no agent is selected this method should be called to clear the contents of the info window.

- Method `infoWin()`
  The constructor of the info window.

- Method `setValues(val:Float,sel:Integer):Void`
  If an agent is selected, this method is used to set the appropriate values in the info window.

- Method `windowClosing(e:WindowEvent):Void`
  Method called when the info window is being closed.

5.3 Package `Simulation`



5.3.1 Class `Settings`

This class is used to store all settings for a single simulation. These are the same settings entered into the settings dialog.

- Attribute `duration:Integer`
  The length of the simulation measured in steps.

- Attribute `width:Integer`
  The width of the aquarium.

- Attribute `height:Integer`
  The height of the aquarium.

- Attribute `initial_algaes:Integer`
  Number of initial agents in the aquarium.

- Attribute `initial_energy:Float`
  Initial energylevel for an agent.
- Attribute `light_cycle_function:Integer`
  Not implemented yet
- Attribute `light_enabled:Integer`
  Not implemented yet
- Attribute `light_function:Integer`
  Not implemented yet
- Attribute `light_intensity:Integer`
  The base intensity of the light source.
- Attribute `pheromone_dec_rate:integer`
  The program uses the reciprocal value of this value to calculate the number of steps, it takes until a piece of pheromone is dissolved.
- Attribute `substrate_amount:Integer`
  Initial amount of substrate.
- Attribute `substrate_renew_rate:Float`
  The program uses the reciprocal value of this value to calculate the number of steps it takes until a new piece of substrate is generated.
- Attribute `energy_light:Float`
  Amount of energy an agent gets for collecting light.
- Attribute `energy_mitose:Float`
  Amount of energy required for mitose.
- Attribute `energy_meiose:Float`
  Amount of energy required for meiose.
- Attribute `energy_movement:Float`
  Amount of energy used for movement.
- Attribute `energy_pheromone_output:Float`
  Amount of energy used for pheromoneoutput.
- Attribute `energy_substrate:Float`
  Amount of enegery an agent get for substrate.
- Attribute `light_fov:Integer`
  FOV of a light sensor.
- Attribute `light_range:Integer`
  Range of a light sensor.
- Attribute `pheromone_fov:Integer`
  FOV of a pheromone sensor.

- Attribute `pheromone_range:Integer`
  Range of a pheromone sensor.
- Attribute `substrate_fov:Integer`
  FOV of a substrate sensor.
- Attribute `substrate_range:Integer`
  Range of a substrate sensor.
- Attribute `bumper_range:Integer`

  Range of a bumper sensor.
- Attribute `randSeed:Integer`
  This is a randomseed value, which is used for the entire program, to be able to repeat exactly the same simulation again and again.

5.3.2 Class `WC`

This is the basic WorldController, which is used to control the state of a simulation.

- Attribute `stepsRunning:Integer = 0`
  Number of steps already done.
- Attribute `stepsToRun:Integer = 0`
  Total number of steps to do in the simulation.
- Method `finish():Void`
  Clean up the worldcontroller.
- Method `step():Void`
  Do a single step of the simulation.
- Method `WC(set:Settings)`
  The constructor for the worldcontroller.

5.3.3 Immutable Class `GUIWC`

This is an extended derivation of WC. The GUIWC has some more attributes and methods than the basic WC, which are needed for drawing the state of the simulation and displaying informations.

- Method `gUIWC(simulationSettings:Settings,p:MainGUI)`
  The constructor for the GUIWC.
- Method `getAgentAt(px:Integer,py:Integer):Integer`
  Get index of agent at a certain position (if there is an agent).
- Method `getAgentInfos(n:Integer):Float[]`
  Get all informations about an agent (used for info window).

- Method `getLightAt(px:Integer,py:Integer):Float`
  Get light intensity at a certain position in the aquarium.

- Method `getPheromoneAt(px:Integer,py:Integer):Float`
  Get pheromone amount at a certain position in the aquarium (if there is any pheromone at this position).

- Method `getState():WorldState`
  Get state of the world.

- Method `getSubstrateAt(px:Integer,py:Integer):Float`
  Get substrate amount at a certain position in the aquarium (if there is any substrate at this position).

- Method `run():WorldState`
  Make a single step of the simulation.

5.3.4 Immutable Class `BatchWC`

This is an extended derivation of WC. The BatchWC has some more attributes and methods than the basic WC. This WorldController is used for a simulation in batchmode. Additionaly this WorldController writes a so called statisticsfile to disk.

- Attribute `fileName:String`
  The filename of the ALGAE settingsfile to use for the simulation.

- Package-Attribute `Initialized:Boolean = false`
  Flag showing if the BatchWC was initialized correctly.

- Method `batchWC(set:Settings,fn:String)`
  The constructor for the BatchWC.

- Method `run():Void`
  Method to run the simulation.

5.3.5 Class `WorldState`

This class is used to temporary store the state of the current simulation. These informations are then used later for writing the statisticsfile or for displaying the current simulation with all its worlds.

- Attribute `dimx:Integer`
  The width of the aquarium.

- Attribute `dimy:Integer`
  The height of the aquarium.

- Attribute `lightIntensity:Float`
  The current lightintensity in the lightworld.

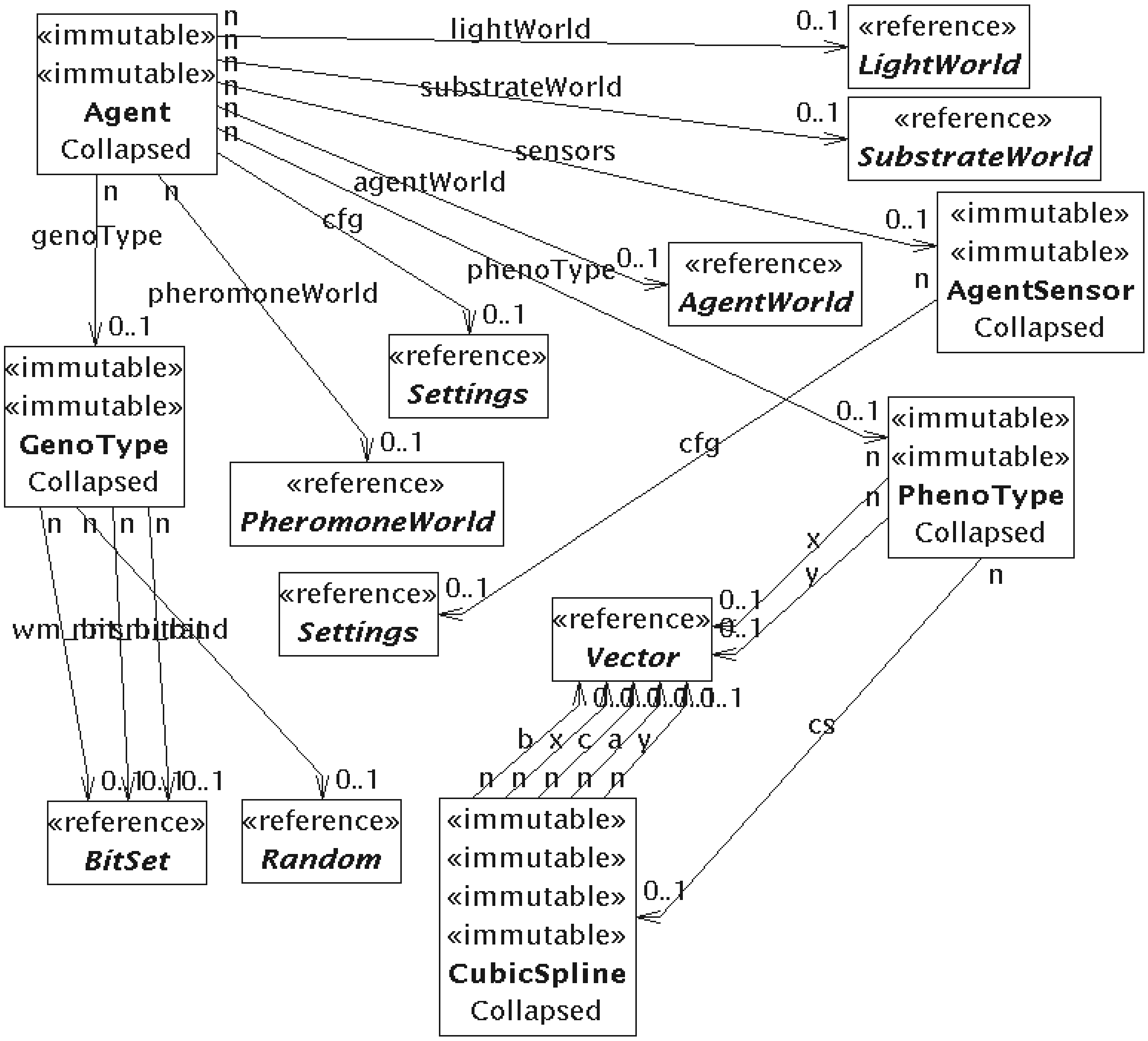
- Attribute `pList:ViewPheromone[]`
  List of pheromones to show.
- Attribute `aList:ViewPheromone[]`
  List of agents to show.
- Attribute `sList:ViewPheromone[]`
  List of substrate to show.
- Attribute `steps:Integer`
  Steps already done.
- Attribute `stepsToRun:Integer`
  Total steps to run.
- Attribute `substrateAmount:Float`
  Current substrateamount in the substrateworld.
- Method `worldState(wc:WC)`
  The constructor for this class.

5.3.6 Immutable Class `SimulationStats`

This class provides all methods, which are needed to write statistics to disk (this class is only used in batchmode).

- Package-Attribute `fileName:String`
  The filename of the statistics file.
- Method `openStream(set:Settings,agw:AgentWorld):Boolean`
  This method opens a filestream for statistiscoutput to a file.
- Method `closeStream():Boolean`
  This method closes the stream for the statisticsfile.
- Method `simulationStats(s:String)`
  Constructor for this class.
- Method `writeStats(stepNo:Integer):Boolean`
  Write simulationstate for a specific step.
- Method `exportSNNS(idx:Integer,a:Agent):Boolean`
  Export the SNNS-file for a specific agent.
- Method `writeSummary():Boolean`
  Write the summary of the whole simulation into the statisticsfile.

5.4 Package Agents



With this package, the core of the program is built: The agents and their neural networks.

5.4.1 Immutable Class Agent

This class stores all informations about one agent. There are also methods implemented in this class, which are needed to move the agent, to calculate the agent's ANNoutput. An also very important part of ALGAE, the reproduction (either sexual or asexual) of the agents, is implemented in this class too (the repdroduction is fullfilled with the different constructors of Agent).

- Attribute `ANN_output:integer = 0`
  This variable stores the ANNoutput for later use.

- Attribute `born:Boolean = false`
  Was the agent born during this step?

- Attribute `dead:Boolean = false`
  Shows if the agent is alive.
- Attribute `died:Boolean = false`
  This flag shows, if the agent died during the current step.
- Attribute `dir:integer`
  The current heading direction of the agent.
- Attribute `energy_level:Float = 0.0`
  The current energylevel.
- Attribute `madeMeiosis:Boolean = false`
  This flags shows, if the agent was involved into meiosis during the current step.
- Attribute `madeMitosis:Boolean = false`
  This flags shows, if the agent was involved into mitosis during the current step.
- Attribute `sensor_values:Float[] = new float[SENSORCOUNT]`
  These are the sensor values for the current step.
- Attribute `usedLight:Boolean = false`
  Did the agent consume light during the current step.
- Attribute `usedSubstrate:Boolean = false`
  Did the agent consume substrate during the current step.
- Attribute `x:Integer`
  The current position of the agent (x-coordinate).
- Attribute `y:Integer`
  The current position of the agent (y-coordinate).
- Method `agent(a:Agent,b:Agent,crossOverRate:Float,mutationRate:Float, c:Settings,rand:Random)`
  Constructor for meiosis.
- Method `agent(a:Agent,mutationRate;Float,c:settings,rand:Random)`
  Constructor for mitosis.
- Method `agent( sx:Integer,sy;Integer,c:Settings,rand:Random,aw:AgentWorld, sw:SubstrateWorld,pw:PheromonWorld,lw;LightWorld)`
  Constructor for a random initialized agent.
- Method `calcStep():Integer`
  Calculate the ANN output and the new position of an agent (Mitosis and meiosis are done in this method too).
- Method `getInfos():Float[]`
  Get all infos of an agent (used for agent information window).
- Method `isAlive():Boolean`
  This method returns if the agent is alive.

5.4.2 Immutable Class `AgentSensor`

This class provides all the methods, which are used by the agents' sensors.

- Attribute `fov:Integer`
  The field of view for a sensor (measured in degrees).
- Attribute `range:Integer`
  The range of an agent (measured in grid elements).
- Method `agentSensor(c:Settings,iRange:Integer,iFov:Integer)`
  Constructor for an agentSensor.
- Method `getValuesA(x:Integer,y:Integer,w:AgentWorld):float[]`
  This method returns the values of an agent's bumper sensors.
- Method `getValuesL(x:Integer,y:Integer,w:LightWorld):float[]`
  This method returns the values of an agent's light sensors.
- Method `getValuesP(x:Integer,y:Integer,w:PheromoneWorldWorld):float[]`
  This method returns the values of an agent's pheromone sensors.
- Method `getValuesS(x:Integer,y:Integer,w:SubstrateWorld):float[]`
  This method returns the values of an agent's substrate sensors.
- Method `inBounds(x:Integer,y:Integer):Boolean`
  This method checks if the current sensor detection position is in bounds.

5.4.3 Immutable Class `Phenotype`

This class stores the ANN of an agent represented with matrices. The core function of the ANN (calculation of the ANNoutput) is also placed in here.

- Package-Attribute `hiddenCount:Integer`
  Number of neurons in hidden layer.
- Package-Attribute `inOutCount:Integer`
  Number of neurons without hidden neurons.
- Attribute `inCount:Integer`
  Number of input neurons.
- Attribute `nCount:Integer`
  Total number of neurons used.
- Attribute `neuronOutput:Float`
  This array stores the ouput of the ANN.
- Attribute `neuron_matrix:Integer`
  This is the modified miller matrix.

- Attribute `outCount:Integer`
  Number of output neurons.

- Attribute `spline_matrix:Float`
  This array stores the splines used as transformation function in a neuron.

- Attribute `visited:Boolean`
  Flag showing wether a specific neuron was already visited during calculation process or not.

- Attribute `weight_matrix:Float`
  The weights for the ANN.

- Method `calculateNeuron(n:Integer):Float`
  This method is used to calculated the output of neuron n.

- Method `connected(i:integer,j:integer):Boolean`
  This method is used to check if two neurons are connected.

- Method `getPrev(i:integer,n:Integer):Integer`
  Get nth parent neuron of neuron i.

- Method `getPrevNum(n:Integer):Integer`
  Get number of parent neurons of neuron n.

- Method `phenotype(g:Genotype)`
  The constructor of phenoType.

- Method `printNeuronMatrix():Void`
  Print the neuron_matrix to the console (for debugging purposes).

- Method `printoutput():Void`
  Print the output of the ANN to the console (for debugging purposes).

- Method `printWeightMatrix():Void`
  Print the weight_matrix to the console (for debugging purposes).

- Method `processNet(inVec:Float):Integer`
  This method processes the ANN and calculates the output.

- Method `transferFunc(funcSel:Integer,val:Float):Float`
  This method is used to transform the output of the ANN.

5.4.4 Immutable Class `GenoType`

This is pretty much the same as PhenoType, but instead of matrices GenoType uses only bitstrings to represent the ANN of an agent. Through the different constructors the reproduction of the ANN is controlled (meiosis, mitosis, randomly initialized agent)

- Package-Attribute `crippled:Integer`
  This attributes stores the strengh of crippleness of an agent.

- Package-Attribute `randSeed:Integer`
  This random seed is used for random initializations.
- Method `crippleness():Integer`
  Returns the cripplenes of an agent.
- Method `crossOver(a:GenoType,b:Genotype,corssOverRate:Float, mutationRate:Float):Void`
  This method creates a new GenoType by crossing over two other genotypes.
- Method `float2IEEE(f:Float):Bitset`
  This method converts a single precision float to a 32-bit IEEE bitset.
- Method `iEEE2float(f:BitSet):Float`
  This method converts a given bitset to a single precision float.
- Method `genoType(rs:Integer)`
  This is the constructor for a randomly initialized genoType.
- Method `genoType(rs:Integer,a;GenoType,b:genotype,crossOverRate:Float, mutationRate:Float)`
  The constructor for a genotype which is a result of crossing over two others.
- Method `genoType(rs:Integer,a:GenoType,mutationRate:Float)`
  The constructor for a mutated genotype.
- Method `getFloatOuttaSm(idx:Integer):Float`
  Extracts a float out of the spline bitset.
- Method `getFloatOuttaWm(idx:Integer):Float`
  Extracts a float out of the weight matrix bitset.
- Method `initRandomNm():Void`
  Initializing the neuron matrix with random values.
- Method `initRandomSm():Void`
  Initializing the splines with random values.
- Method `initRandomWm():Void`
  Initializing the weight matrix with random values.
- Method `isCrippled():Boolean`
  Returns wether the agent is crippled or not.
- Method `mutate(mutationRate:Float):Void`
  This method makes a mutation on the genotype.
- Method `putFloatIntoSm(idx:Integer,f:Float):Void`
  Stores a float into the spline matrix bitset.
- Method `putFloatIntoWm(idx:Integer,f:Float):Void`
  Stores a float into the weight matrix bitset.

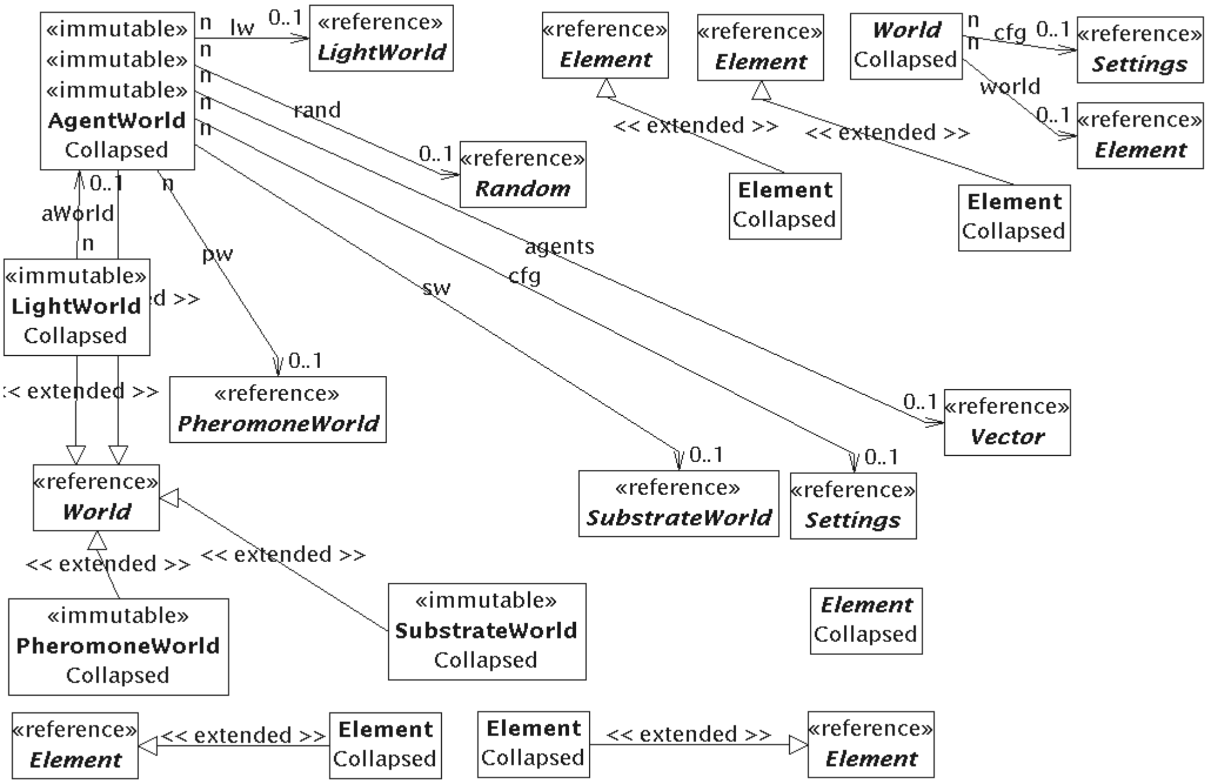
5.4.5 Immutable Class `CubicSpline`

This class provides an interpolation function for internal use in PhenoType (used as the transfer function for a neuron).

- Package-Attribute `nPoints:Integer`
  Number of points used for the splines.
- Protected Method `cubicSpline(n:Integer,px:Vector,py:Vector)`
  Constructor for a CubicSpline.
- Method `getValue(px:Double):Double`
  Get the value of the CubicSpline at position px.

5.5 Package `Worlds`

The world where our simulation takes place is built with these classes.



5.5.1 Class `World`

This is the basic class for all worlds used in ALGAE.

- Static Attribute `OUTOFWORLD:Double = 0.0`
  This value indicates, that the sensor is out of bounds.
- Attribute `renew:Integer = 0`
  How much world elements were renewed during the last step?

- Attribute `used:Integer = 0`
  How much world elements were used during the last step?

- Abstract Method `step(step:Integer):Void`
  Abstract method for the calculation of one step.

- Method `world()`
  Constructor for World.

- Method `world(y:Integer,x:Integer)`
  Constructor for World with given dimension.

5.5.2 Immutable Class `AgentWorld`

AgentWorld is a derivation of World and stores all informations all agents, which were born during a simulation (the agents which already died, are stored in here too for later use in the statistics file).

- Attribute `agentcount:Integer`
  Number of total agents in the agent world.

- Attribute `mAX_AGENTS:integer`
  Maximum number of agents in the world.

- Attribute `stepDeadCount:Integer = 0`
  How many agents died during the last step?

- Attribute `stepLightUsed:integer = 0`
  How much light was consumes during the last step?

- Attribute `setpMeiosisCount:Integer = 0`
  How often did meiosis occur during the last step?

- Attribute `stepMitosisCount:Integer = 0`
  How often did mitosis occur during the last step?

- Attribute `stepPheromoneNew:integer = 0`
  How much pheromones were set during the last step?

- Attribute `stepPheromoneUsed:Integer = 0`
  How much pheromones were dissolved during the last step?

- Attribute `stepSubstrateNew:Integer = 0`
  How much substrate was renewed during the last step?

- Attribute `setpSubstrateUsed:Integer = 0`
  How much substrate was used during the last step?

- Method `agentWorld(set:Settings,s:SubstrateWorld,p:PheromoneWorld, l:LightWorld):Boolean`
  The constructor for the agent world.

- Method `aliveAgents():Integer`
  Return the number of agents alive.

- Method `doPheromoneOutput(tmp:Agent):Integer`
  Let an agent ouput new pheromone.

- Protected Method `findNeighbour(nOf:Integer,a:Agentempty:Boolean):Integer`
  Find a direct neighbour of an agent.

- Method `getList():ViewAgent[]`
  Returns a list of ViewAgent for later drawing.

- Method `initAgents():Void`
  Randomly initialized the agents at startup.

- Method `move(tmp:Agent,sel:integer):Integer`
  Move all agents to their new positions (according to the ANN output).

- Method `step(steps:Integer):Void`
  Calculate one step in the agent world.

5.5.3 Class `SubstrateWorld`

This class is a derivation of World. It stores information about the current substrate distribution and the substrate amount at each position in the aquarium.

- Attribute `amount:Double`
  Total amount of substrate available in the aquarium.

- Method `getList():ViewSubstrate[]`
  This method returns a list of ViewSubstrate for later drawing.

- Method `getSubstrateAmount():Double`
  Returns the amount of available substrate.

- Method `step(step:Integer):Void`
  This method calculates one step in the substrate world.

- Method `substrateWorld(set:Settings)`
  The constructor for the substrate world.

5.5.4 Class `LightWorld`

This class is a derivation of World. It stores information about the current light distribution and the light amount at each position in the aquarium.

- Attribute `intensity:Double`
  The current intensity of the lightsource.

- Method `consume(x:Integer,y:integer):Double`
  This method is called whenever an agent consumes light.

- Method `getList():ViewLight[]`
  This method returns a list of ViewLight for later drawing.

- Method `lightWorld(set:Settings)`
  Constructor for the light world.

- Method `setAgentWorld(aWorld:AgentWorld):Void`
  This method is used to set the agentworld from outside of this class.

- Method `step(step:Integer):Void`
  This method calculates one step in the light world.

- Method `update():Void`
  This method is used to update the state of the light world.

5.5.5 Class `PheromoneWorld`

This class is a derivation of World. It stores the pheromone amount at each position in the aquarium.

- Method `getList():ViewPheromone[]`
  This method returns a list of ViewPheromone for later drawing.

- Method `pheromoneWorld(set:Settings)`
  The constructor of the pheromone world.

- Method `set(x:integer,y:Integer)`
  This method is used to set a new pheromone.

- Method `step(step:Integer):Void`
  This method calculates one step in the pheromone world.

5.5.6 Class `Element` abstract class for World

This is the basic class for all types of elements used in the worlds of ALGAE (substrate, pheromone, agent and light).

- Attribute `amount:Double`
- Private Method `add(step:Integer):Boolean`
- Method `element()`
- Private Method `element()`
- Private Method `exists():Boolean`
- Protected Method `update():Boolean`
- Method `value():Double`
- Method `value(x:Integer,y:Integer):Double`

5.5.7 Class `Element` for LightWorld

This is the class used for a lightelement. It holds the light intensity at a certain position in LightWorld.

- Attribute `x:Integer`
- Attribute `y:Integer`
- Method `add(s:Integer):Boolean`
- Method `element(x:Integer,y:Integer)`
- Method `exists():Boolean`
- Method `remove(s:Integer):Boolean`
- Protected Method `update():Boolean`

5.5.8 Class `Element` for PheromoneWorld

This is the class used for a piece of pheromone stored in PheromoneWorld. It holds the pheromone amount at a certain position in PheromoneWorld.

- Attribute `stepsToResolve:Integer`
- Method `add(steps:Integer):Boolean`
- Method `element()`
- Method `element(amount;Double,dec:Integer)`
- Method `exists():Boolean`
- Method `remove(steps:Integer):Boolean`
- Protected Method `update():Boolean`

5.5.9 Class `Element` for SubstrateWorld

This is the class used for a piece of substrate stored in SubstrateWorld. It holds the substrate amount at a certain position in SubstrateWorld.

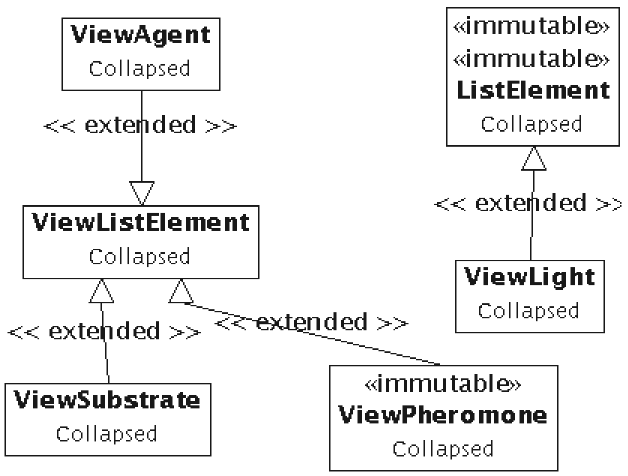
- Attribute `stepsToGo:Integer`
- Attribute `stepsToRegenerate:Integer`
- Method `add(steps:Integer):Boolean`
- Method `element()`
- Method `element(amount;Double,dec:Integer)`
- Method `exists():Boolean`
- Method `remove(steps:Integer):Boolean`
- Protected Method `update():Boolean`

5.5.10 Class `Element` for AgentWorld

This is the class used for an agent places in AgentWorld.

- Method `add(steps:Integer):Boolean`
- Method `element()`
- Method `exists():Boolean`
- Method `remove(steps:Integer):Boolean`
- Protected Method `update():Boolean`

5.6 Package `View`



5.6.1 Class `ListElement`

A class which simply stores positions.

- Attribute `x:Integer`
  x-coordinate of ListElement.
- Attribute `y:Integer`
  y-coordinate of ListElement.
- Method `listElement(x:Integer,y:Integer)`
  Constructor for a ListElement.

5.6.2 Class `ViewListElement`

A class which simply stores positions for later drawing.

- Attribute `x:Integer`
  x-coordinate of ViewListElement.

- Attribute `y:Integer`
  y-coordinate of ViewListElement.

- Method `viewListElement(x:Integer,y:Integer)`
  Constructor for a ViewListElement.

5.6.3 Class `ViewAgent`

This class is a derivation of ViewListElement and stores the position of a single agent for later drawing.

- Attribute `direction:Integer`
  Direction of the agent to be drawn.

- Attribute `indexOf:Integer`
  Index of the agent to be drawn in the agent list.

- Method `viewAgent(x:Integer,y:Integer)`
  Constructor for ViewAgent.

5.6.4 Class `ViewLight`

This class is a derivation of ViewListElement and stores the position of a single lightelement for later drawing.

- Attribute `amount:Float`
  Amount of LightElement used to calculate transparency for drawing.

- Method `viewLight(x:Integer,y:Integer,amount:Float)`
  Constructor for ViewLight.

5.6.5 Class `ViewSubstrate`

This class is a derivation of ViewListElement and stores the position of a single piece of substrate and its concentration for later drawing.

- Attribute `amount:Double`
  Amount of SubstrateElement used to calculate transluency for drawing.

- Method `viewSubstrate(x:Integer,y:Integer,amount:Double)`
  Constructor for ViewSubstrate.

5.6.6 Class `ViewPheromone`

This class is a derivation of ViewListElement and stores the position of a single piece of pheromone and its concentration for later drawing.

- Attribute `amount:Double`
  Amount of PheromoneElement used to calculate transluency for drawing.

- Method `viewPheromone(x:Integer,y:Integer,amount:Double)`
  Constructor for ViewPheromone.

Chapter 6

GUI Description

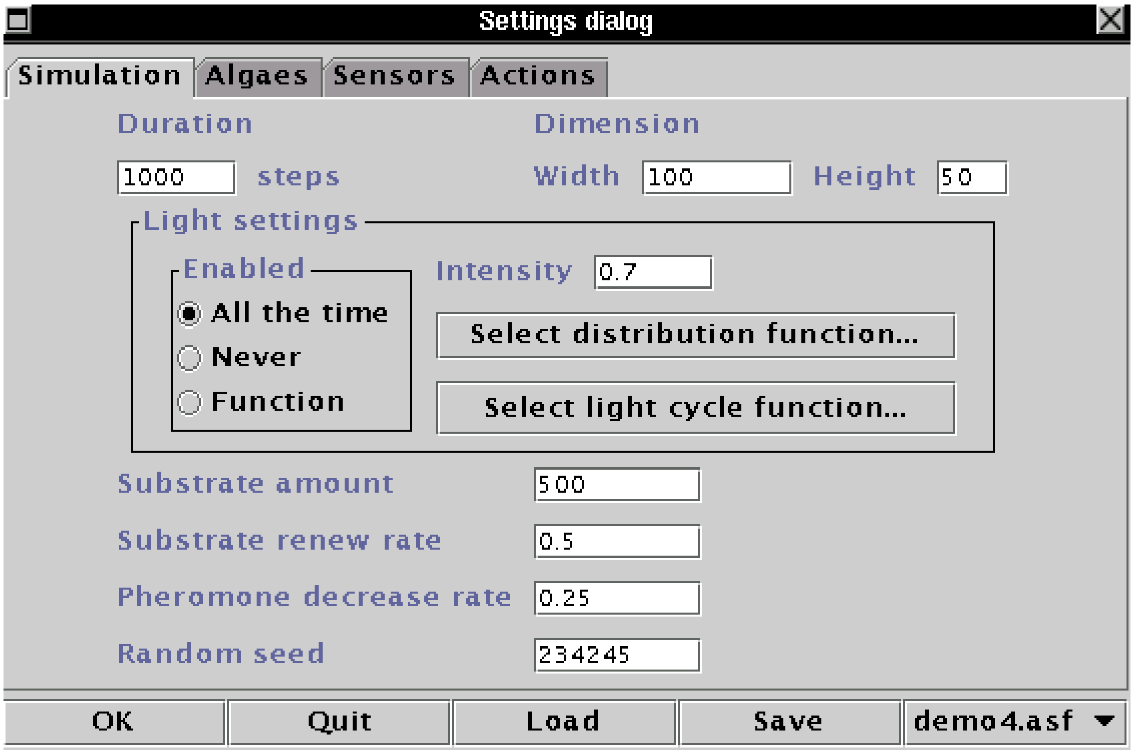
In this chapter, we explain how the program is controlled and how it outputs the information. We use a **G**raphical **U**ser **I**nterface for input and output.

6.1 Configuration Window

The program starts with the configuration window, where all parameters relevant for the simulation are adjusted.
With the `OK` button the user accepts the adjustment; the Simulation window opens.
With the `Quit` button the user may abort the program
When the button `Save` is clicked, a dialog asking for a filename pops up. The current configuration is saved to this file
All saved configuration files with the appendix .asf are shown when the user clicks the dropdown list. When one of the filenames is selected, the configuration is loaded from this file.
There are four tabstrips in the configuration window:

1. Simulation
2. Algaes
3. Sensors
4. Actions

6.1.1 Simulation



- The text field labeled `Duration` determines the duration d of the simulation. If the entered time is exceeded, the user may enter any number of additional timesteps, and the simulation is resumed; otherwise, the simulation finally stops.
- The text fields labeled `Width`, $worldx$ and `Height`, $worldy$ determine the size of the aquarium.
- The four text fields on the bottom determine:

  the amount of substrate a_S at the beginning of the simulation. (`Substrate amount`)

  after how many time-steps ren_S the substrate particles are renewed on bottom of the aquarium(`Substrate renewal rate`)

  after how many steps dec_P a pheromone disapears (`Pheromone decrease rate`)

  the `random seed` that the program uses. To eliminate any differences based on coincidence, the user can enter the same random seed for every simulation.
- In the frame `Light settings`, $state_L$ there are
  - radio buttons labeled `Enabled` with the following possibilities:

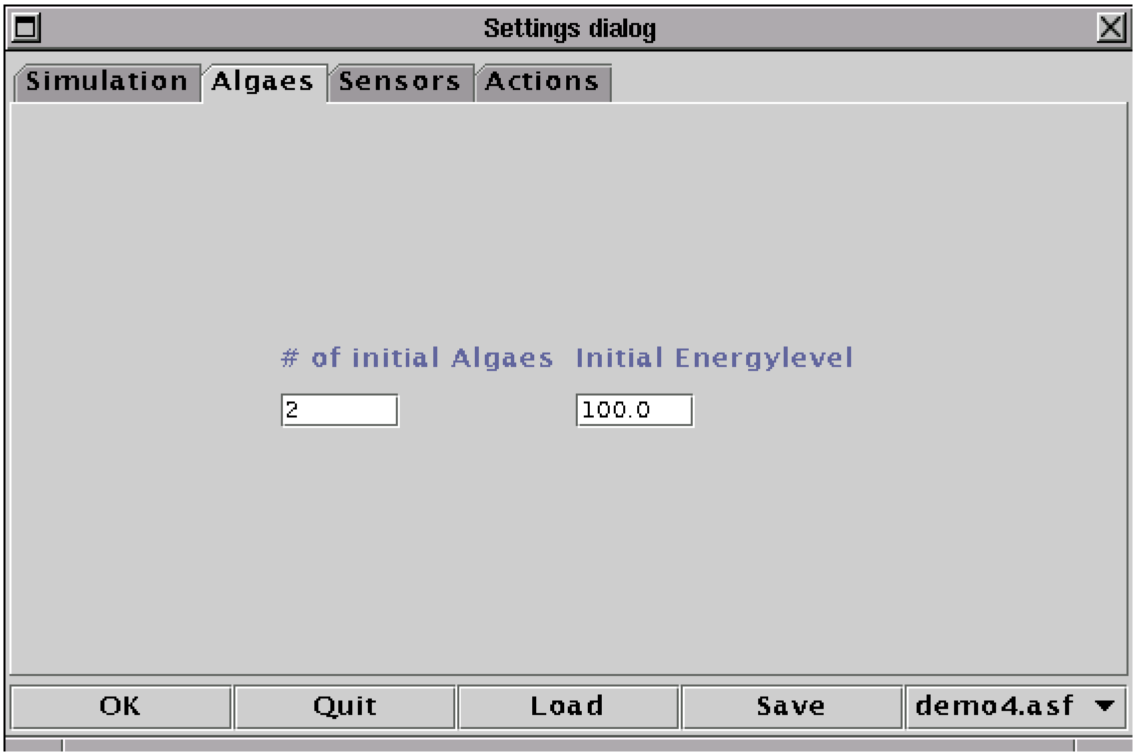
    `All the time`, which means the light shines the whole simulation

    `Never`, which means the light doesn't shine at any time of the simulation

    `Function`, the light intensity is determined by a function. *Remark: this is not fully implemented yet. At this stage, the light always shines based on a function.*

- a text field labeled `Intensity`, which determines the light intensity i_L on the top of the aquarium.
- a button labeled `Select distribution function`, where the user can select one function d_S of several light distribution functions *Remark: Not implemented, the distribution uses a fixed function at this stage.*
- a button `Select light cycle function`, which determines how the light intensity changes over time. *Remark: Now, this is always a sinus function to simulate a day/night cycle.*

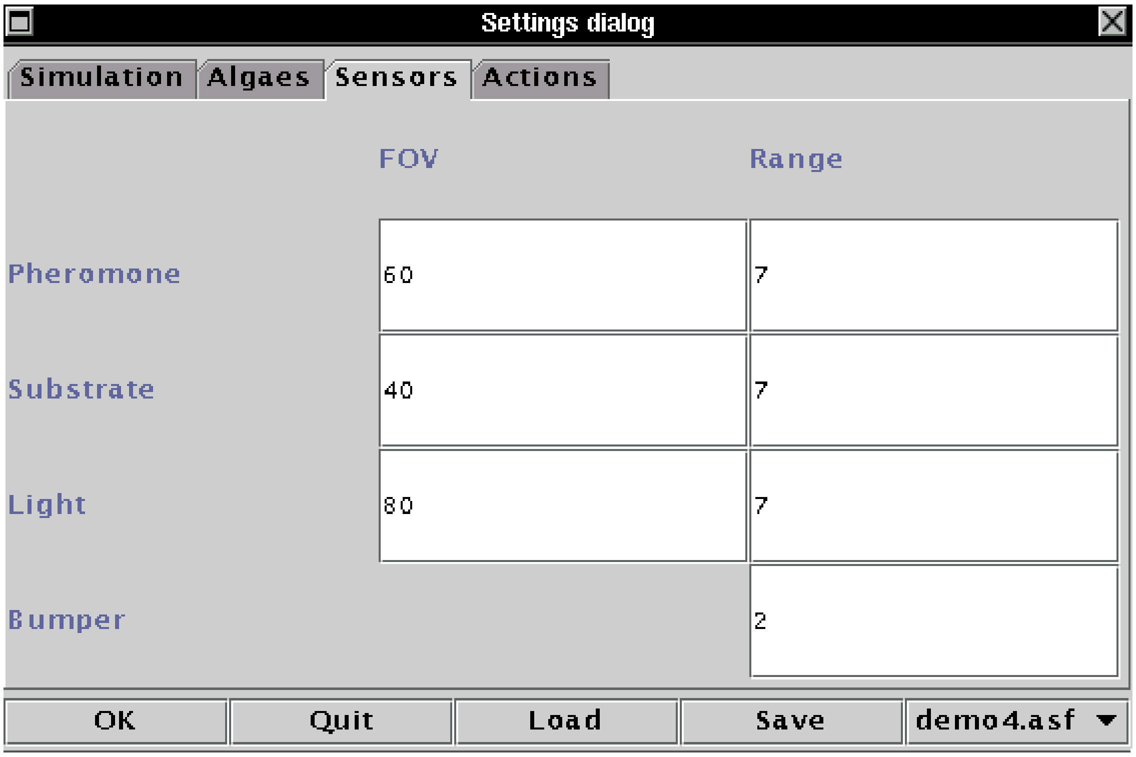
6.1.2 Algaes



In this tabstrip there are two textfieldes.

- The first field labeled `# of initial Algaes` determines the number of algae ini_a the simulation should begin with..
- The second textfield labeled `Initial Energylevel` determines the energylevel ini_e which all algaes have at the beginning of the simulation.

6.1.3 Sensors

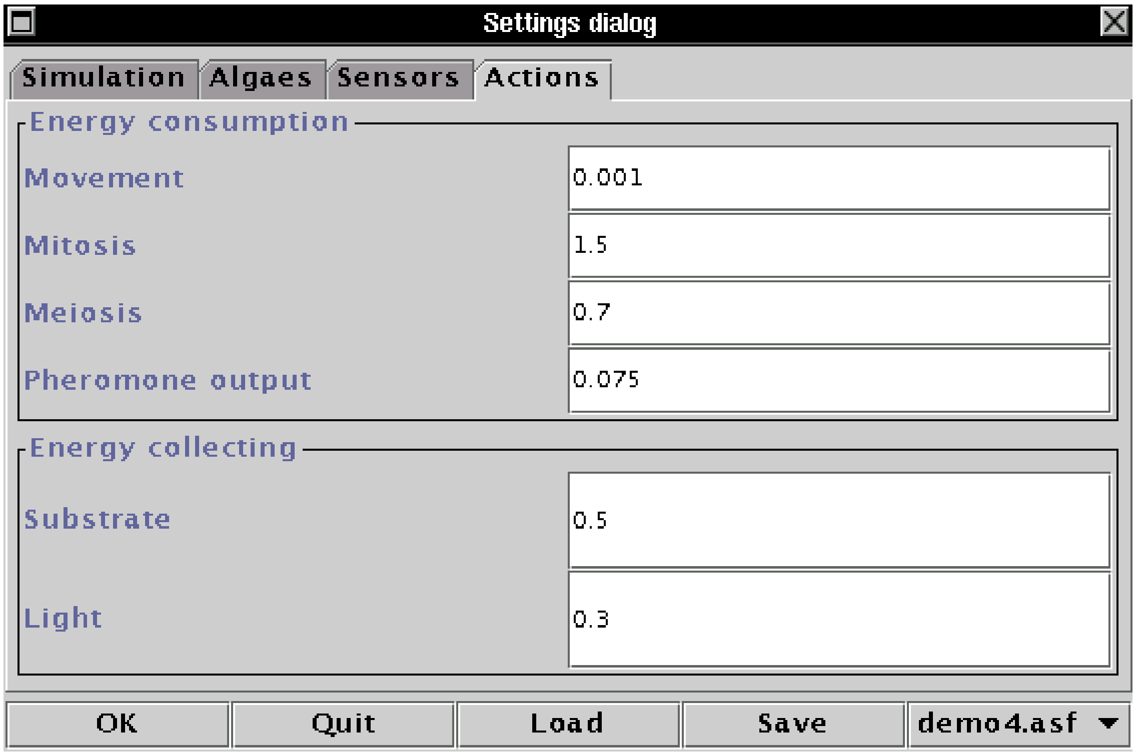


In this tabstrip the user determines the adjustment of the four sensors (`Pheromone` P, `Substrate` S,`Light` L,`Bumper` B)

- `FOV` Field of View f , this is the angle that the sensor field covers in degrees. Note that the bumper sensor always has a f of 1. It only detects obstacles in a straight line.

- `Range` the range r of the sensor in world units

6.1.4 Actions



In this tabstrip, there are two frames:

- In the first frame labeled `Energy consumption` the user can determine

  How much energy e_M an agent needs to move around (`Movement`).

  How much energy e_I an agent needs to reproduce via mitoses (`Mitoses`).

  How much energy e_E an agent uses to reproduce via meiose (`Meioses`).

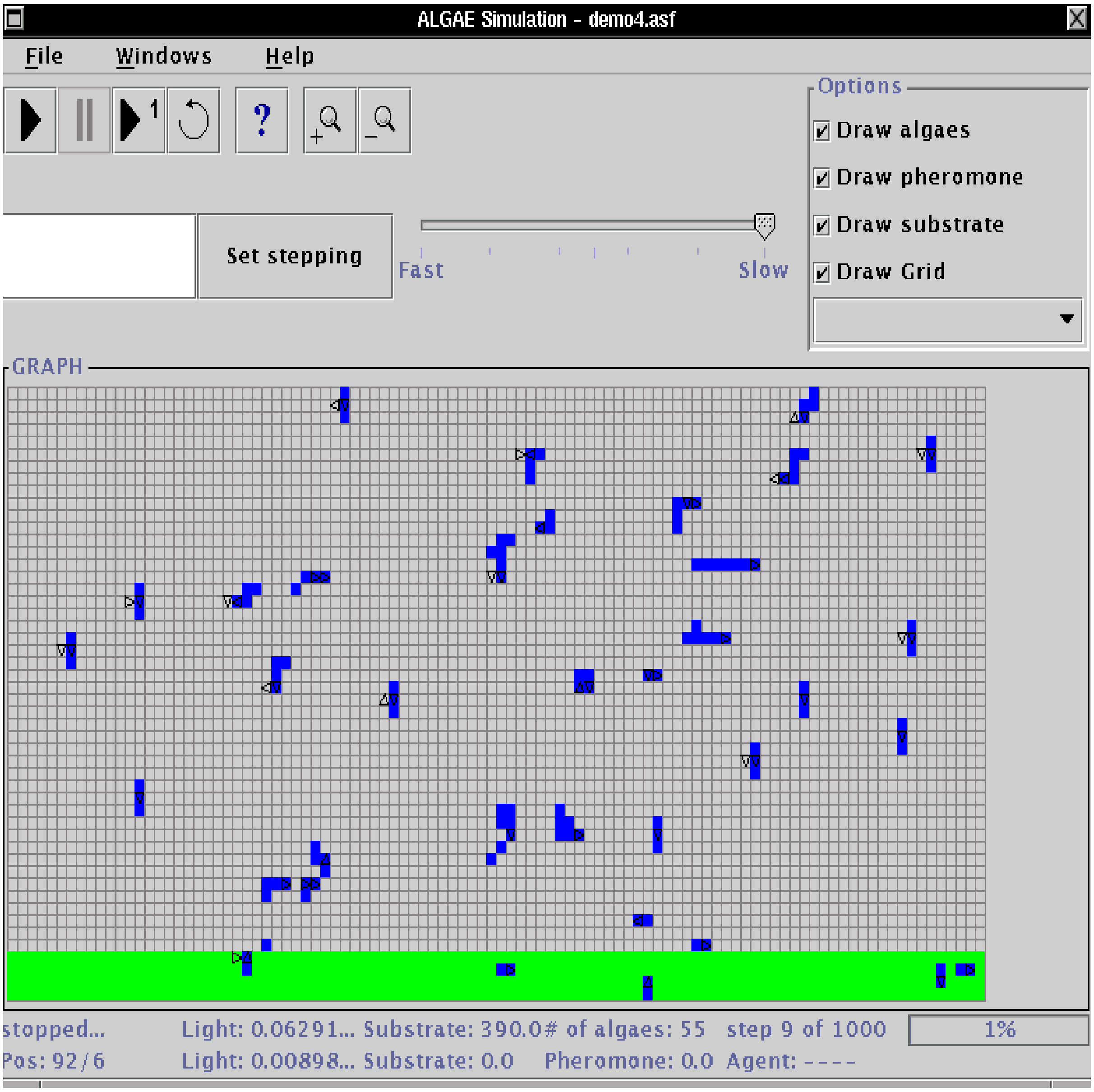
  How much energy e_P an agent uses to discharge one pheromone prticle (`Peromone output`).

- in the second frame labeled (`Energy collecting`) the user determines

  how much energy e_S an agent gets by consuming one (`Substrate`) particle

  and how much energy e_L can an agent can take from (`Light`) `(this is a relative value, because the amount of energy is also based on the light intensity)`

6.2 Main GUI



6.2.1 Aquarium

This is the world where our simulation takes place. It shows the agents, pheromon tracks and substrate parts.

- Agents are shwon in black
- Selected agents are red
- Pheromone tracks are blue and cover substrate particles on the same position
- Substrate is green

If an agent is selected via a click of the left mouse button, it changes its color and information about this agent is shown in the **Agent Info Window**(see below) in the case it is visible.

6.2.2 MenuBar

File

- `Quit` Ends the application after a safety query

Windows

- `Show/hide agent info window` (see below)

Help

- `Topics` (shows the online help)
- `About ALGAE` (displays the program's credits)

6.2.3 ToolBar

The most important controls of the program art contained in the toolbar.

Buttons

- The first button starts or resumes the simulation
- The second button pauses the simulation
- The third button is used to proceed one step
- The fourth button restarts the simulation, using the current parameters
- With the sixth button it is possible to raise the zoom factor of the aquarium so the details can be watched more easiely.
- The ninth button is used to lower the zoom factor to improve the overview over the sim

  (The possible zoom factors reach from 1 pixel for each algae to a 10*10-pixel-per-agent view.)

- `Set stepping`: If a number s is entered into the text field besides this button and `Set Stepping` is clicked, the output of the aquarium is updated every s steps instead of after every step (default)

Slider

There is one slider in the program window:

- The speed sp_Sim of the simulation, e.g. the delay is controlled by the second slider (from `Fast` to `Slow`).

Check buttons (`Options`)

- `Draw algaes/pheromone/substrate/grid`: The visible objects in the aquarium may be shown or hidden using this four check buttons

6.2.4 StatusBar

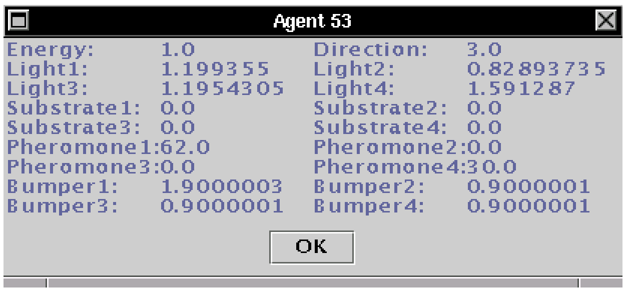
By default, it shows the following information (from left to right):

- The simulation status (running, stopped, etc)
- The light intensity on top of the aquarium
- The current agent count
- The current and total number of timesteps

If the mouse pointer resides at a specific top in the aquarium, the second line of the statusbar shows the following information:

- The vertical and horizontal positions
- Light intensity at this place
- Is there substrat at this place?
- How many phermomon particles, if any, reside at this place?
- If an agent is at the position, what's its number?

6.3 Agent Information Window



If the menu point `Windows/Info Window` is chosen in the Main GUI, this window opens. It can be closed with the `Okay` button. The following information about the selected agent is given:

- Current energy level e
- Moving direction of the agent
- Sensor values of all four sensors

Appendix A

Session logs

A.1 March 15, 1999

Time: 17:00 - 17:30
Place: Room 202.2, Institute for Computer Sciences, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EisenmANN, T. Frommwald, M. Liedlgruber and D. Kadir

Treated points:

- Formalities
- General introduction to the problem

Tasks to perform until March 22,2000

- Get photos for the project-homepage
- Have a careful look at the document
- Have a look on JAVA

A.2 March 22, 2000

Time: 17:00 - 17:30
Place: Room 202.2, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EisenmANN, T. Frommwald, M. Liedlgruber and D. Kadir

Treated points:

- We have a concrete matrix where the simulation should take place
- Substrate

  Distribution of substrate is based on a mathematical function

  There is a specific amount of the substrate at every point in our matrix

The regeneration of the substrate depends on the distribution function and a basic time period (determined by some parameters)

- Sensors

  The sensors of our agents have a specific range and an angle of 'sight'. These things are variabel.

  We got a new sensor that can detect contact (with the wall or other agents)

- Pheromons disintegrate after a while

- Possible agent actions: Power consumption, movement, pheromon output, reproduction

  All actions consume different amounts of energy

- Energy consumption through photosynthesis or substrat accomondation

- Movement

  Speed ?

  Agents may move one unit in one time unit

- Reproduction

  Meiose / Mitose

  If the agent wants to reproduce through meiose he has to have a certain amount of energy. The agent follows the pheromon tracks of other agents and puts out pheromons himself.

- Pheromon output

  During movement

  The agent should not follow his own pheromon track

- Simulation steps

  The current status of the sim is displayed after each simulation step

  During one step the agents may perform an arbitrary amount of actions (exception: movement !)

Tasks to fullfill until March 29, 2000

- Prepare further questions

A.3 March 29, 2000

Time: 17:00 - 17:30
Place: Room 202.2, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EisenmANN, T. Frommwald and D. Kadir

Treated points:

- Agent input

  We need another input for our neural network: the current energy level of the agent

  It is probably necessary that agents know whether the detected pheromon is their own or not

- The neural network

  There are units in the ANN which are connected through weighted lines

  The more important the connection of two unit becomes, the thicker the connecting line.

  Output consists of the movement direction and everything concerning reproduction

- The light

  Probably there will be some sort of 'fight' for the positions that are near the light

  Agents that are not on top level will get less light (shadows)

  For this reason they may try to consume more of the substrate - the according connecting line of the ANN gets thicker

- How could the agent learn that sexual reproduction is better than the mitose ?

Tasks to fullfill until April 12, 2000:

- Revise the paper:

  Exact problem definition

  Session logs

  Discussions

A.4 April 12, 2000

Time: 17:00 - 17:30
Location: Room 202.2, Institute for Computer Sciences, University of Salzburg
Present persons: R. Schwaiger, P. EisenmANN, T. Frommwald, M. Liedlgruber and D. Kadir

Treated points:
(document revised:)

- Light/Substrate:

  Diffusion: Substrate regenerates faster at the bottom than above.

  Agent has transparency factor.

- Pheromon, Mitose, Meiose:

  Pheromon is placed only at the point, were agent is situated;

  time-delayed pheromon reduction.

  4 various ways to consider meiosis - we choose the 3rd one: forced reproduction.

- World(object):

  Output over all partial aspects.

- ANN:

  Processing of signals with 'Cubic Splines-Methode'(set points to calculate a function)

  In the ANN 'the winner takes all' - the maximum evaluated response is selected

Tasks until Mai 2, 2000

- first GUI (/World)-prototype
- create UML-model
- ev. other questions

A.5 May 2, 2000

Time: 17:00 - 17:30
Location: Room 202.2, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EinsenmANN, T. Frommwald and D. Kadir

Treated points:

GUI Interface

- Saving

  Possibility of saving from intermediate stages

  To draft on a particular step of the simulation

- Buttons, menu bar and tabstrip

  "Settings" in the menu bar is problematic

  Button for gradual running

  Rename "Main" by "Aquarium"

New tabstrip which shows a list of all algae and the possibility to select one of them

Move the "ANN Sensors" tabstrip into the algae tab writer

A new menu point "Windows" which shows the details of the agents.

- others

  Show light and substrate in the aquarium.

  Possibility to hold back indication for a certain amount of steps

Tasks until May 24, 2000

- Description of the GUI Interface
- Some agents should move in the aquarium
- Think about in which information we would be interested
- Make treated changes of the GUI

A.6 May 24, 2000

Time: 17:00 - 17:30
Location: Room 202.2, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EinsenmANN, T. Frommwald and D. Kadir

Treated points:

- Document

  The document has to contain the class diagrams and use cases

  The source code should not be included

- GUI

  We will include the agent selection list discussed in a former session

- ANN

  The ANN is initialized randomly

  Visualization of the ANN is done by another project group, so we can omit that part

- We should use the same random seed for each start
- The maximum number of neurons may be entered
- Our agents "learn through reproduction"
- We save information using the XML format

Tasks until June 6, 2000

- Revise the paper, insert case diagrams and use cases
- Finish the GUI
- Make multiple agents move around in the world

A.7 June 6, 2000

Time: 17:00 - 17:30
Location: Room 0.09, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EisenmANN, T. Frommwald and D. Kadir

Treated points:

- GUI - Settingsdialog

  Simulation: change substrate-settings to be similar to light-settings (with function)

  Sensors: add bumper-sensor

  add button to save settings

- Simulation:

  output the sim state only every x-steps (definable)

  make the duration of simulation (number of timesteps) changeable during runtime

  Statistics: protocol the simulation state on harddisk and make sure functionality can be added later easily

- There should also be a slider to change the amount of substrate (similar to light)
- Colored agents: color depends on their amount of energy

Next meeting: June 21, 2000

A.8 June 21, 2000

Time: 17:00 -17:30
Location: Room 0.09, Institute for Computer Science, University of Salzburg
Present persons: R. Schwaiger, C. Bauer, P. EisenmANN, T. Frommwald and D. Kadir

Treated points:

- Progress of implementation
- Improvement of paper

  Write the section "abstract" (150 words)

  Extend and revise the Introduction

  Add new (latest) screenshots to the paper

  Formalize and generalize the whole paper.

Next meeting: July 5, 2000 14:15 (Presentation)