DEVELOPING AN INDIVIDUAL BASED MODEL OF PHYTOPLANKTON USING A SIZE-BASED APPROACH

Nontsasa Princess Tobi

Thesis submitted as partial requirement for the degree of
MASTER OF SCIENCE in the Department of Zoology, Faculty of Science,
University of Cape Town.

June 2005

Supervisor: Dr. C. L. Moloney

Declaration

This dissertation reports the results of research that I carried out in the Marine Biology Research Institute, Zoology Department, at the University of Cape Town. It has not been submitted for any other degree or examination at any other University. All opinions expressed, unless otherwise acknowledged, are my own.

signature removed

.....

Nontsasa Princess Tobi

08/06/2005

Date

Acknowledgements

I would like to say thanks to National Research Foundation (NRF), UCT scholarship office, Marine Biology Research Institute of the Zoology department and J.W. Johnston scholarship for making this project possible. The beginning abd the completion of this project would have not been possible without the planning and contribution of my supervisor: Dr Coleen Moloney; Coleen thank you very much for giving all your support, guidance and love during the time of my studies, may God richly bless you. Nangamso!!

I would like to thank my biological mother for her support, encouragements and love; Mama I made it by the grace of God, undithwele kwaze kwaba lapha. How can I forget my spiritual mothers: Mrs Morrison, Chikwata and Magudya; I am not forgetting their husbands. I would also like to thank my colleages in the Moloney laboratory for their smiles and encouragements and for creating such a lovely environment for everybody, keep it up guys! I would like to thank my sisters in the Lord for their prayers: Berverly, Samantha, Nozuko, Lebogang, Ranolang and Desiree. Last but not least I would like to thank Mr X. Tonjeni our little boy Mfezeko for being my ladder for me to reach masters level; may God bless you abundantly.

Developing an individual-based model of phytoplankton using a size-based approach

Nontsasa Princess Tobi

Marine Biology Research Institute, Zoology Department, University of Cape Town, Rondebosch, 7701, South Africa, E-mail addresses: ntobi@egs.uct.ac.za / nontsasat@yahoo.com
Tel: +27 21 650 3613; Fax: +27 21 650 3301 and Cell: +27 82 063 8364

An individual-based model (IBM) is created using java programming to examine three size classes of a phytoplankton community in the southern Benguela region. A size-based simulation approach has been developed previously using state variable models to investigate the dynamics of carbon and nitrogen flows, but the approach has not been applied using IBMs. For this study the IBM modeled phytoplankton and nutrients. All parameters were dependent on body size. The parameters describe nitrogen uptake, carbon fixation, respiration, growth, reproduction, and mortality (include grazing and sinking). The simulations have three size classes of phytoplankton, namely, pico-, nano- and net-phytoplankton. The model represents a cm^3 of water in the euphotic zone; this small volume was required because of computer constraints. In the model each phytoplankton individual is modeled as an object, and each object has a set of values and a set of behaviors. The nutrient-phytoplankton IBM (**NP-IBM**) used a time step of 12 hours over a total of 250 days to represent relevant processes that occur in each individual phytoplankton cell. Pico-phytoplankton were too numerous to be modeled effectively, and were deliberately constrained at unrealistically small numbers. Two mortality models (quadratic and linear functions) for calculating probability of dying of individuals for each size class were developed, and the numbers, nitrogen pool and biomass of each size class were calculated. The quadratic function resulted in greater numbers of nano- and net-phytoplankton, and a reduced bloom duration compared with the linear function, and it was used in further experiments. The model is very simple, only including phytoplankton with no zooplankton. Large phytoplankton cells (nano-and net-phytoplankton) had greater biomass but fewer individuals than small phytoplankton (pico-phytoplankton). The size-based IBM approach warrants further investigation, but is constrained by the number of individuals that can be handled.

Author Keywords: Individual-based model; NP-IBM; Size-based approach; Nutrients; State-variable models; Java programming language; phytoplankton; mortality probability

Introduction

The upwelling region of the Benguela ecosystem is situated on the west coast of southern Africa. There are three main factors that govern upwelling events in the region: 1) climatic forcing via the south-east Atlantic high pressure anticyclone, which causes equator-ward winds, 2) the topography and 3) the influence of the Agulhas Current (Touratier et al. 2003). Upwelling of nutrient-rich water supports large stocks of phytoplankton in the west coast region with large primary production (Moloney et al. 1991, Brown et al. 1999), that in turn supports important commercial fisheries (Moloney et al. 1991). The largest fishery in the region is the pelagic fishery, based on sardine *Sardinops sagax* and anchovy *Engraulis encrasicolus*. Coastal upwelling systems are also seen as areas where excess atmospheric carbon is used by phytoplankton in the euphotic zone and exported to the deep ocean by rapid sinking after surface waters have been depleted of available nutrients (Touratier et al. 2003). The transfer of carbon and other nutrients in these ecosystems is of interest.

A number of studies have been done in the southern Benguela to understand the trophic processes affecting pelagic fish, and processes occurring in the plankton, including studies by Moloney and Field (1991), Cochrane (2000) and Touratier et al. (2003). Simulation models are used to investigate factors affecting community structure and dynamics and serve as a basis for understanding the functioning of marine planktonic ecosystems. One approach that has been used for simulations of plankton communities after upwelling events uses a size basis (Moloney and Field 1991, Moloney et al. 1991, Moloney 1992). Marine pelagic food webs are largely structured on the basis of organism size. More importantly, rates of many processes occurring in plankton

ecosystems depend on body size (Moloney and Field 1989). This means that body size can be used to estimate almost all parameters in a size-based plankton model. Moloney et al. (1991) showed how such a model, based on ecological principles and not on a specific ecosystem, could be used to simulate interactions in plankton communities of any ecosystem. Moloney and Field's (1991) model was a specific application of a traditional category of model known as a nutrient-phytoplankton-zooplankton (NPZ) model. NPZ models are ecological models of lower trophic level processes in the upper ocean, and include at least one limiting nutrient (N) e.g. nitrogen, one or more phytoplankton producers (P), and one or more zooplankton consumers (Z) (Batchelder et al. 2002). NPZ models have been designed to answer questions about nutrient flow and how it is linked in phytoplankton and zooplankton populations, and factors affecting the vertical distribution of chlorophyll and nutrients i.e. the interaction of upper ocean physics and biology (Evans and Garcon 1995). Traditionally, NPZ models have assumed that all individuals within a trophic compartment are identical (Batchelder et al. 2002), and the models typically estimate total biomass in each compartment. For this reason, NPZ models fall into the category of "state variable" models.

Individual based models (IBMs) use a "bottom-up" approach that focuses on individuals (Grimm 1999), which are allowed to interact. The modeller assesses how population characteristics emerge when the point of view is shifted towards the individual 'upwards' (Grimm 1999; Grimm 2002). IBMs contrast with state variable models in the way in which they handle organisms (Gross et al. 1992). State variable models treat all individuals as the same, using one "state" variable to represent population biomass or numbers e.g. Moloney and Field (1991), and assuming that each individual in the population is equal to an 'averaged' individual (Grimm 1999). IBMs have been used in

the southern Benguela upwelling region to interpret fish population functioning (Miller et al. submitted). IBMs of ichthyoplankton have been coupled to circulation models (Huggett 2003; Parada 2003) and particle tracking including behavior has been used to examine sources, sinks and retention in dynamically changing flow fields. To date I know of no IBM of phytoplankton and nutrients. State variable models have been developed to simulate phytoplankton communities using size-based approaches (e.g. Moloney and Field 1991) but never such an approach in an IBM.

There are a number of reasons why an IBM of phytoplankton would be useful. The first is to compare such a model with more traditional state variable models. Secondly, phytoplankton behaviour (e.g. vertical movements) and species-dependent characteristics might be important for understanding phytoplankton blooms, and these are easily addressed using IBMs. This project aims to develop an IBM of a phytoplankton community, using size as a basis to differentiate among groups of individuals (as for Moloney and Field 1991). It has been assumed that the physical environment is kept as simple as possible, meaning that light is assumed to be non-limiting, temperature effects are kept constant and the physical structure is stable. Advantages and disadvantages of the IBM approach for phytoplankton will be assessed.

Methods

The model consists of a phytoplankton community occurring in the euphotic zone just above the thermocline. In this study an IBM was created using Java programming. The IBM is used to examine carbon transfer and nitrogen absorption for three size classes of a phytoplankton community in the Benguela region: pico-phytoplankton (0.2-20μm

ESD, equivalent spherical diameter), nano-phytoplankton (2-20μm ESD) and net-phytoplankton (20-200μm ESD).

An average size is calculated for each size class using a geometric mean (Moloney and Field 1991). Because of computer constraints related to the maximum number of individuals, the model represents a cubic centimeter (cm^3) of water. The flows of carbon and nitrogen for individual cells are represented in **Fig. 1**. Carbon is absorbed by each phytoplankton cell through carbon-fixation during photosynthesis and each cell takes up nitrogen from solution (Moloney et al. 1991). Cells lose carbon through respiration and accumulate carbon and nitrogen as growth. They die through natural mortality, which includes grazing and sinking. Reproduction is done through replication whereby cells divide mitotically, producing two identical replicates (same size) (see **Fig. 1**).

There are many important processes and parameters that are associated with population dynamics of phytoplankton. The following are simulated in this model based on the calculations in Moloney and Field (1991): nitrogen uptake, carbon fixation, respiration, growth, reproduction and mortality.

Nitrogen uptake

The uptake rate of nitrogen is calculated as:

$$N\ uptake_{ind}\ (\text{ng N d}^{-1}) = \mu_i\ (\text{d}^{-1}) \times N_{ind}\ (\text{ng N}) \qquad (1)$$

where N_{ind} is the nitrogen mass of each individual cell (ind) and μ_i is the mass-specific uptake rate for each size class *i*. *Nitrogen* is the ambient dissolved nitrogen (NO_3) concentration. The uptake rates are calculated using Michaelis-Menten models with size-dependent maximum uptake rates ($\mu_i max$) and half-saturation constants (K_i).

$$\mu_i\,(\mathrm{d}^{-1}) = \mu_i max\,(\mathrm{d}^{-1}) \frac{Nitrogen\,(\mathrm{ng\ N\ cm}^{-3})}{Nitrogen\,(\mathrm{ng\ N\ cm}^{-3}) + K_i\,(\mathrm{ng\ N\ cm}^{-3})} \quad (2)$$

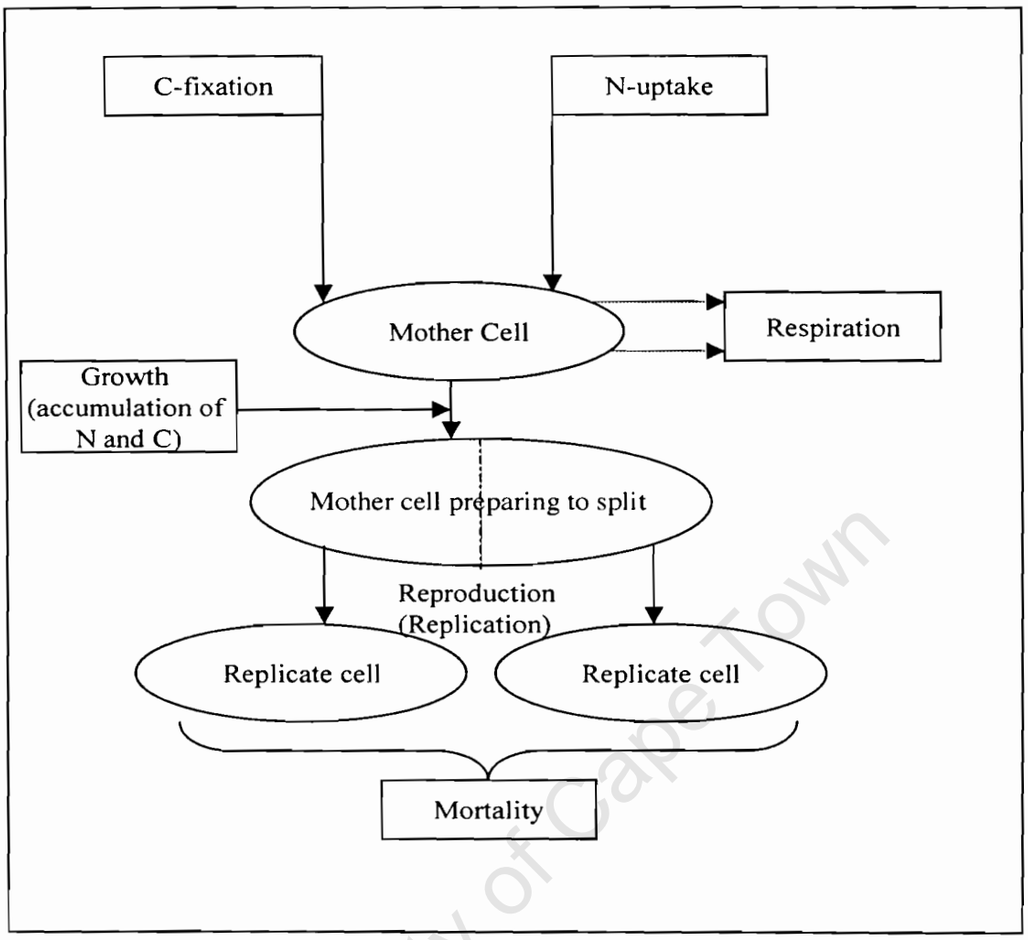


Fig. 1 Diagrammatic representation of the flows of carbon and nitrogen for each individual phytoplankton cell

Carbon fixation

Carbon fixation is determined by nitrogen availability and is calculated as follows:

$$C\,fixation_{ind}\,(\mathrm{ng\ C\ d}^{-1}) = \mu_i\,(\mathrm{d}^{-1}) \times C_{ind}(\mathrm{ng\ C}) \quad (3)$$

where C_{ind} is the carbon mass of each individual cell.

Respiration

Respiration rate in the model is calculated as a fraction of each individual carbon mass:

$$Respiration_{ind}\ (\text{ng C d}^{-1}) = R_i\ (\text{d}^{-1}) \times C_{ind}\ (\text{ng C}) \quad (4)$$

where R_i is the weight-specific size-dependent respiration rate.

Growth

Growth is calculated as follows:

$$Growth_{ind}\ (\text{ng C d}^{-1}) = C\,fixation_{ind}\ (\text{ng C d}^{-1}) - Respiration_{ind}(\text{ng C d}^{-1}) \quad (5)$$

Reproduction

When the C weight of a cell is double the initial mass, it divides into two, the two cells each being assigned half the mass of the "mother" cell (**Fig. 1**).

Mortality

It was necessary to develop a function to represent mortality. It was assumed that the probability of dying would increase as the total biomass increased, and that small cells would have a greater probability of dying than large cells per unit of time. The probability of dying for each cell in a size class *i* has been calculated as:

$$Probability_i = a_i Biomass\ (\text{pg C})^b \quad (6)$$

where a_i is different for each size class: 1, 0.01 and 0.0001 for pico-phytoplankton, nano-phytoplankton and net-phytoplankton respectively. The exponent b determines whether the relationship between the increase in biomass and phytoplankton mortality is linear (b=1) or quadratic (b=2). These assumed relationships are illustrated in **Fig. 2**.

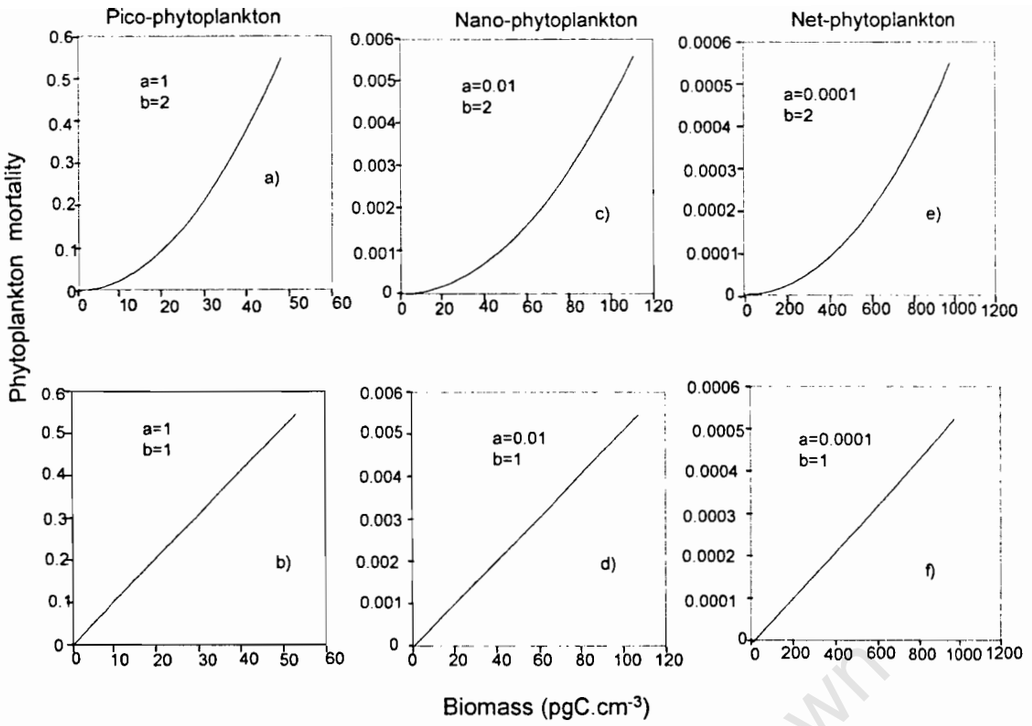


Fig. 2 Models for calculating the daily probability of dying for different size classes of phytoplankton at different biomasses (equation 6). The *a* and *b* values are indicated on the graphs

Another form of mortality is when C and N are lost through sinking of phytoplankton. Sinking losses are calculated as:

$$\textit{Sinking}\ (\text{ng C or N d}^{-1}) = S_i\ (\text{m d}^{-1})/D\ (\text{m}) \times C_{ind}\ (\text{or } N_{ind}) \qquad (7)$$

where S_i is the sinking velocity of size class i and D is the depth of the water mass.

Model Parameters

Regression models in **Table 1** were used to calculate values of size-based parameters used in this IBM simulation model (**Table II**), as was done by Moloney and Field (1991).

Table I. Regression models used to calculate values of size-dependent parameters in the simulation model, where M represents carbon mass (pg):

Process/ variable	Models	Parameters	Values	Units
Nitrogen uptake	$\mu_i max\ (d^{-1}) = VMAXA * M^{SIZEEXP}$	VMAXA	3.6	$pg\ C^{0.25}\ d^{-1}$
		SIZEEXP	-0.25	-
	$K_i (mg\ N\ m^{-3}) = VKS1 * M^{VKS2}$	VKS1	2	$pg\ C^{-0.38} mg\ N\ m^{-3}$
		VKS2	0.38	-
Respiration	$R_i\ (d^{-1}) = RV * M^{SIZEEXP}$	SIZEEXP	-0.25	-
		RV	1.7	$pg\ C^{0.25} d^{-1}$
Sinking	$S_i\ (m\ d^{-1}) = SP1 * M^{SP2}$	SP1	0.029	$pg\ C^{-0.42}\ m\ d^{-1}$
		SP2	0.42	-

The calculated parameter values for K_i and sinking rates differ from those presented in Moloney and Field (1991); the values here are correct (**Table II**).

Table II. The phytoplankton sizes, masses and model parameters used in this model

	Pico-phytopl.	Nano-Phytopl.	Net-Phytop.
ESD (μm)	0.63	6.3	63
Volume (μm^3)	0.132	132	132461
Carbon-weight (pg)	0.088	16	2800
Nitrogen-weight (pg)	0.015	2.6	463
$\mu_i max$ (d^{-1})	6.6	1.8	0.5
*K_i (ng N cm^{-3})	0.8	5.7	40.7
Respiration (d^{-1})	3.1	0.9	0.23
*Sinking (m d^{-1})	0.01044	0.09	0.81044
Mortality *a*	1	0.01	0.0001
Mortality *b*	1 or 2	1 or 2	1 or 2
C:N ratio	6	6	6
Starting numbers	500	32	18
Starting biomass (ng C)	0.044	0.512	50.4

* : parameter values differ from those presented in Moloney and Field (1991)

The program

The model has been named as a nutrient-phytoplankton-individual-based model (NP-IBM) (**Fig. 3**). It was developed using the Java programming language in Jbuilder X (http:://www.Borland.com/, last accessed 20 January 2005).

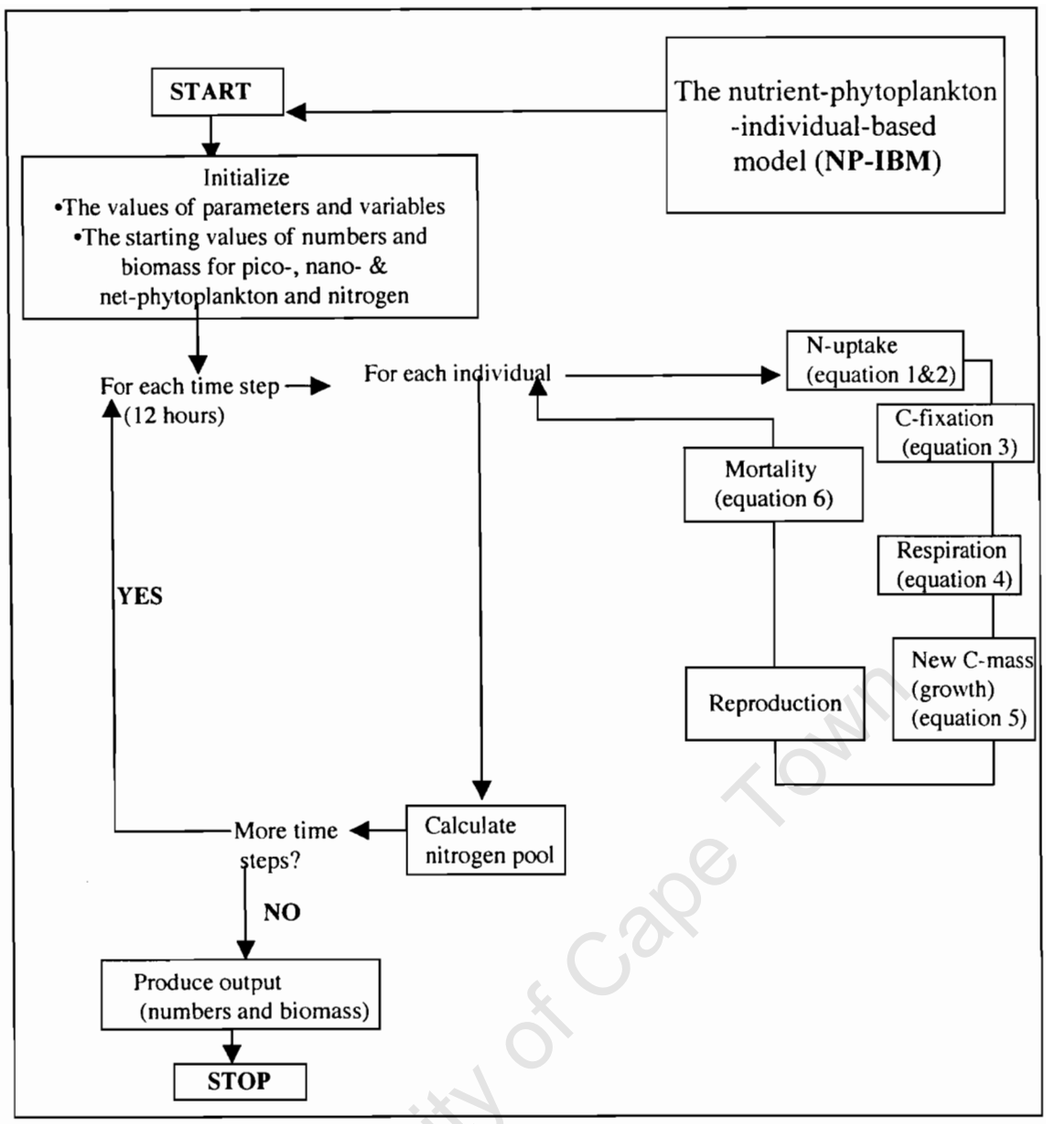


Fig. 3 A summary representation of the program

The java program (Plankton.jpx, **Appendix B**) is based on a template developed by Christian Mullon (IRD, France, pers. comm.) and which uses libraries created by him, and was adapted by myself and Coleen Moloney (UCT, pers. comm.). I developed some algorithms for the program, and adapted some of the java language. I carried out the

simulations myself including capturing and exporting the results. The program consists of five classes that are described below.

SimulationPlankton class

This is the main class of the program. In this class all the required graphics controllers of the program and output graphs are set up and the procedure to be followed in each simulation is laid out. Time steps are set in hours which are converted to fractions of one day. The numbers of individuals in the model are set with a maximum initial number of 6000 pico-phytoplankton, 600 nano-phytoplankton and 60 net-phytoplankton. The nitrogen pool is set to a maximum initial value of 60 ng N cm^{-3}. The probability of dying for each size class is calculated, varying between zero and one. The total initial biomass of phytoplankton is calculated, based on the numbers that are input. In the first time step, individuals are created for each size class, and are assigned parameter values. For each subsequent time step, the individuals in each size class are allowed to grow, reproduce and die, and population variables (total biomass and total numbers) are calculated.

Phytoplankton24 class

All the values of the size dependent parameters are calculated in this class using regression models (**Table II and Appendix A**).

Pico-phytoplankton, Nano-phytoplankton and Net-phytoplankton classes

These classes set the attributes of the individuals in each size class, e.g. carbon mass.

Experiment 1

Experiment 1 was performed to check which time step would be appropriate for the model to simulate all three size classes at the same time. Four different time steps were tested: 4, 6, 12 and 24 hours. The N pool was initialized at 490 mg N cm^{-3}.

Experiment 2

Because of the computer constraints, the model was unable to handle very large numbers of individuals, which affected mainly pico-phytoplankton. As a consequence mortality was made very high and was possibly unrealistic. This experiment compared the linear and quadratic models (equation 6) for calculating the daily probability of dying for each of the different size class of individuals (**Fig. 2**).

Experiment 3

In this experiment, the model was executed with all three-size classes in the community, using a time step of 12 hours and the quadratic model to calculate mortality probability for each size-class.

Results

Experiment 1

Time steps of 4 and 6 hours consumed a lot of time to complete the simulation. A 24-hour time step resulted in negative values, and was discounted as being too large. A 12-hour time step was found to be stable and did not consume a lot of time; this time step was used for all subsequent simulations (see **Appendix A**).

Experiment 2

The model was used to simulate and calculate the probability of dying for each individual pico-, nano- and net-phytoplankton cell. Simulation outputs are shown in **Fig. 4**. Pico-phytoplankton and nano-phytoplankton grow faster in numbers than net-phytoplankton, which has the largest biomass (**Table III**). All three populations persist at some maximum value until the individuals start to die because all the nitrogen has been consumed (**Fig. 4**) and none of phytoplankton cells can grow and reproduce without nitrogen.

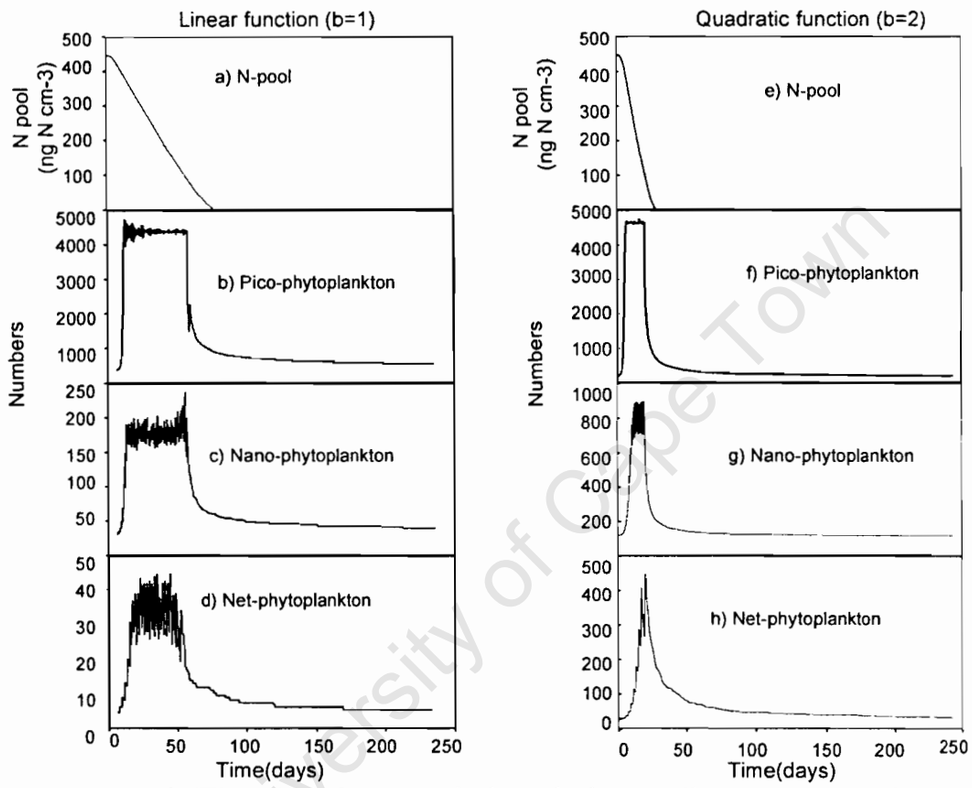


Fig. 4 The output of the IBM simulation for experiment 2: Dissolved nitrogen pool outputs and change with time of phytoplankton numbers using both linear and quadratic functions to calculate mortality.

Table III. Results of experiment 2 comparing the two mortality models (b=1 or 2) in terms of the approximate maximum values of biomass and numbers of individuals, and the durations of blooms for each phytoplankton size class.

Mortality model	Size classes	Maximum biomass (ng C cm^{-3})	Maximum number	Duration (days)
b=1	Pico-phytoplankton	0.4	4500	35
	Nano-phytoplankton	4	250	35
	Net-phytoplankton	126	45	35
b=2	Pico-phytoplankton	0.4	5800	12
	Nano-phytoplankton	15	900	12
	Net-phytoplankton	1260	60	30

When the model is run with a linear mortality function (b=1) phytoplankton individuals grow slower than with a quadratic function (Fig. 4). The number of individuals for each size-class reaches a maximum value where reproduction and growth are balanced by mortality, but only persists at these levels for a short time for the quadratic mortality function (**Table III**). The numbers then decrease rapidly. The nitrogen pool is totally consumed in approximately 25 days when using the quadratic function as compared to 60 days when using the linear function.

Experiment 3

The outputs of the IBM for the whole community show that all size class biomasses reach a maximum value (**Fig. 5**) and decrease with a decrease in the nitrogen pool (**Fig. 4**). Net-phytoplankton have the biggest biomass whereas pico-phytoplankton and nano-phytoplankton have lower biomass (**Fig. 5**).

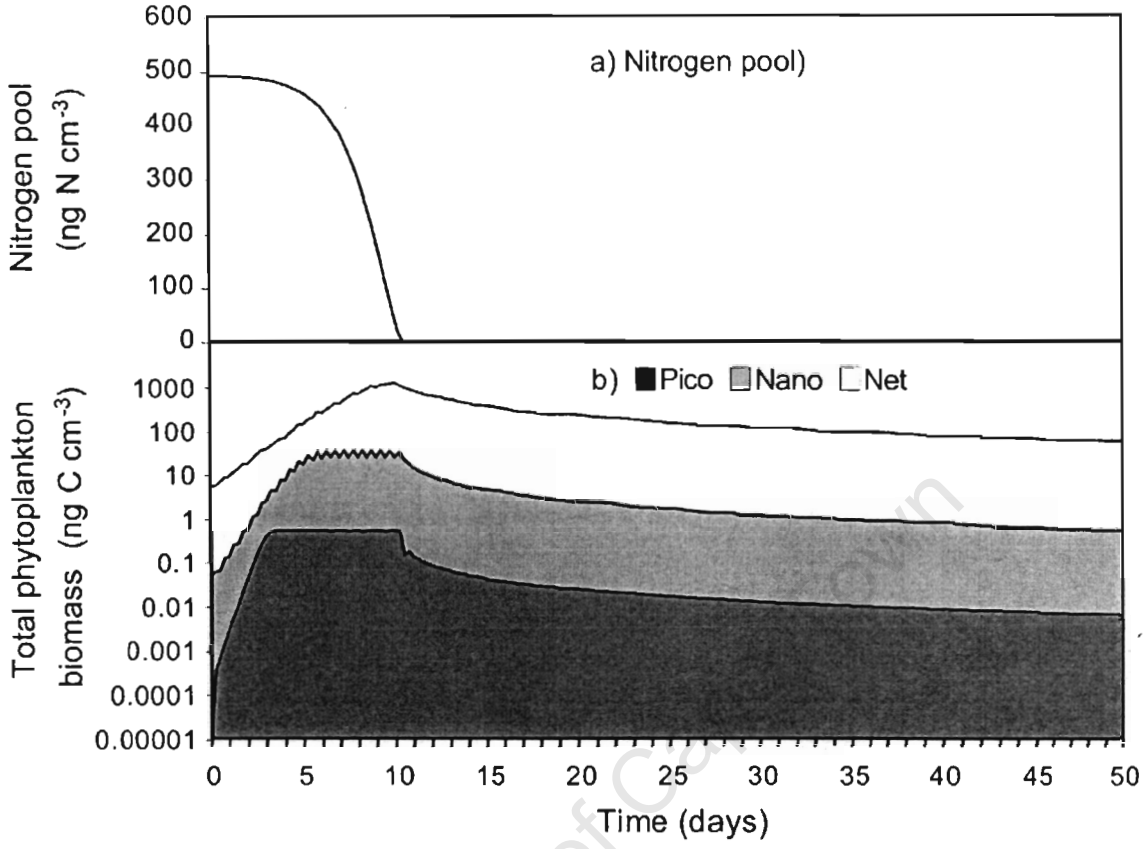


Fig. 5 The output of the IBM simulations for experiment 3: change with time of a) nitrogen pool concentration and b) total phytoplankton biomass using the quadratic mortality (b=2). Relative contributions by different sizes of phytoplankton are shown. Note: a logarithmic axis is used because small phytoplankton have very small biomasses.

Discussion

In developing the model, a choice had to be made between using a large time step, which would speed up the calculations but which might result in negative numbers that give unrealistic results, and a small time step which consumes a lot of computer time. A 24-hour time step gave unrealistic results, because the nitrogen pool became negative. The 12-hour time step that was used for the model is probably not ideal for pico-phytoplankton but was appropriate for large phytoplankton. The pico-phytoplankton numbers were kept artificially low by large mortality rates to suit the model, and were

therefore unrealistic. Nevertheless the NP-IBM was able to simulate different size classes in a phytoplankton community, with relevant processes and parameters.

The model only includes phytoplankton, with no grazers (zooplankton). A mortality parameter for the phytoplankton individuals was therefore needed. For the southern Benguela it has been shown that a lack of N reduces growth of phytoplankton, whereas a decrease in numbers is mainly caused by grazing, sinking having little effect (Pitcher et al. 1991). The two mortality models that were tested, linear (b=1) and quadratic (b=2), gave different results for nano- and net-phytoplankton but not for pico-phytoplankton. Using both models, pico-phytoplankton reached a maximum number of approximately 4500 individuals cm^{-3} but the duration of blooms differed, being 25 days when using the quadratic and 60 days for the linear function. A maximum number of 4500 individuals cm^{-3} is much smaller than found in the field (Li 1995). In the North Atlantic, Li (1995) found that small phytoplankton reached a maximum number of the order of 100 000 cells ml^{-1}. In this study, the very large mortality probability suppresses the numbers of pico-phytoplankton. In the 'real world', pico-phytoplankton are grazed by protists, which have short response time scales, similar to the generation times of pico-phytoplankton (Franks 2001). The protists impose heavy mortality rates on pico-phytoplankton, and as a result they represent a constant background in the ocean, probably controlled by grazing. The mortality probability for pico-phytoplankton acted as a grazing control in the present study; pico-phytoplankton were kept very low in numbers and also in biomass, in agreement Franks (2001).

The quadratic mortality function resulted in a large number of individuals of nano- and net-phytoplankton. These large numbers consumed large amounts of N relatively quickly. Using the quadratic mortality function, the phytoplankton had reduced periods of blooming, because as soon as the N was fully consumed the numbers dropped drastically. When using the linear function, the numbers did not increase to as many as when using the quadratic mortality function. The phytoplankton bloom persisted for approximately 60 days, and when the N was totally consumed the numbers decreased. The difference in bloom duration when using these two mortality models is caused by the fact that they have different total numbers of individuals. The quadratic mortality function resulted in greater numbers than the linear mortality function because the quadratic mortality function had a smaller probability of dying at low biomass values than the linear mortality function (**Fig. 2**). The larger the number of individuals, the larger the amount of N being consumed, and as soon as the N is finished, growth ceases, and numbers decrease.

Models of N and C flows were done using nutrient-phytoplankton-zooplankton (NPZ) models by Steele and Henderson (1992) for different mortality terms, including linear and quadratic terms. Their major conclusion was that the form of the function used for predicting mortality plays a major role in determining the response of all models. They found it most appropriate to use quadratic mortality functions to simulate different plankton systems in the coastal environment. Even though this present study is dealing with individuals, it was found that the term of the mortality function was important in determining the results of the simulations. Following the recommendations of Steele and

Henderson (1992), the quadratic function was used in further experiments to model the probability of dying.

IBMs represent individual properties and processes (Batchelder et al. 2002) but allow population and community properties to emerge (Grimm 1999). IBMs can be very complex and can be difficult to interpret, however they have some advantages over state variable models (Batchelder et al. 2002). State variable models ignore many important details, whereas IBMs attempt to include as many processes and properties as possible (Batchelder et al. 2002). The total duration of the bloom in this model took approximately 20 days, which is similar to that found in Moloney and Field's (1991) model. The net-phytoplankton biomass reached 1200 ng C cm^{-3}, the nano-phytoplankton had a maximum value of about 35 ng C cm^{-3} and pico-phytoplankton 0.5 ng C cm^{-3}, which is different from Moloney and Field (1991). In their model net-phytoplankton had a maximum value equivalent to 250 ng C cm^{-3}, nano-phytoplankton 550 ng C cm^{-3} and pico-phytoplankton approximately 750 ng C cm^{-3}. The values in the present model are more realistic; net-phytoplankton had the highest biomass followed by nano-phytoplankton, with pico-phytoplankton having the lowest. Generally, large cells would be expected to dominate autotroph biomasses (Timmermans 2005), constituting about 90% of the total biomass and small cells about 10% (Moloney 1992). Further research is needed on IBMs of phytoplankton size classes using the java programming language.

Java is an object-oriented programming language that resembles a simplified form of C++ (Slack 2000). In object-oriented languages each 'real world' entity is modeled as an object and each object has a set of values and a set of behaviors. Java was developed in

1991 and incorporated in Internet programming in 1995 (Slack 2000). Although Java has become well known for its ability to do Internet programming, it is also a good general programming language (Slack 2000). Java is an example of a high-level language and has its own syntax for describing "classes" (a class represents a grouping of similar objects) and "methods". It is particularly well suited to individual-based modeling (Budd 2000).

This model represented here is a stochastic model; mortality of individuals was determined using a random number generator. Therefore the simulations represented in the graphs do not show smooth curves. There were a number of limitations in this model: no recycling of nitrogen (ammonia), no predators (zooplankton). There were also no real individual differences among phytoplankton, apart from size. Bottom-up simulation models (IBMs) are powerful tools in biology, ecology and natural resource management (Grimm 2002). They can be used to interpret, predict and identify individual processes and behaviours (Grimm 2002; Gross et al. 1992). Nevertheless, the NP-IBM could not be effectively used for pico-phytoplankton. However, it would be worth examination for further application to large phytoplankton

References

BATCHELDER, H. P., CHRISTOPHER, A. E. and T. M. POWELL 2002 – Individual-based models of copepod populations in coastal upwelling regions: implications of physiologically and environmentally influenced diel vertical migration on demographic success and near shore retention. *Prog. Oceanogr.* **53**: 307-333.

BROWN, S. L., LANDRY, M. R., BARBER, R. T., CAMPBELL, L., GARRISON, D. L. and M. M. GOWING 1999 – Pico-phytoplankton dynamics and production in the Arabian Sea during the 1995 Southwest Monsoon. *Deep-Sea Res.* **46**: 1745 – 1768.

BUDD, T. 2000 – Understanding Object-Oriented Programming with Java. Addison-Wesley Longman, Inc. New York

COCHRANE, K. L. 2000 – Reconciling sustainability, economic efficiency and equity in fisheries: the one that got away? *Fish Fisheries.* **1**: 3-21.

EVANS, G. T. and V. C. GARCON 1997 – One dimensional models of water column biogeochemistry. In: Joint Global Ocean Flux Study (JGOFS); a core project of the international geosphere-biosphere programme. *JGOFS Report* **23**: 4-12.

FRANKS, P. J. S. 2001 – Phytoplankton blooms in a fluctuating environment: the roles of plankton response time scales and grazing. *J. Plankton Res.* **23**: 1433 –1441.

GRIMM, V. 2002 – Visual debugging: A way of analyzing, understanding and communicating bottom-up simulation models in ecology. *Nat. Res. Model.* **15**: 23-38.

GRIMM, V. 1999 – Ten years of individual-based modeling in ecology: what have we learned and what could we learn in the future? *Ecol. Model.* **115**: 129-148.

GROSS, L. J., ROSE, K. A., RYKIEL, E. J., VAN WINKLE, W. §Jr. and E. E. WERNER 1992 – Individual-based modeling: Summary of a workshop. In: Individual-based models and approaches in ecology: Populations, communities and ecosystems. DeAngelis, D. L. and L. J. Gross (Eds.). New York, London; Chapman & Hall publishers: 511-522.

HUGGETT, J. A. 2003 – Comparative ecology of the copepods *Calanoides carinatus* and *Calanus agulhensis* in the southern Benguela and Agulhas Bank ecosystems. PhD thesis, University of Cape Town, 232 pp + 2 Appendices.

LI, W. K. W. 1995 – Composition of ultraphytoplankton in the central North Atlantic. *Mar. Ecol. Prog. Ser.* **122**: 1-8.

MILLER, D. C. M., MOLONEY, C. L., VAN DER LINGEN, C., LETT, C., MULLON, C. and J. G. FIELD Submitted – Modelling the effects of physical-biological interactions and spatial variability in spawning and nursery areas on recruitment of sardine in the southern Benguela ecosystem *J. mar. Sys.*

MOLONEY, C. L. 1992 – Simulation studies of trophic flows and nutrient cycles in Benguela upwelling foodwebs. *S. Afr. J. mar. Sci.* **12**: 457-476.

MOLONEY, C. L., FIELD, J. G. and M. L. LUCAS 1991 – The size-based dynamics of plankton food webs. II. Simulations of three contrasting southern Benguela food webs. *J. Plankton Res.* **13**: 1039-1092.

MOLONEY, C. L. and J. G. FIELD 1991 – The size-based dynamics of plankton food webs. I. A simulation model of carbon and nitrogen flows. *J. Plankton Res.* **13**:1003-1038.

MOLONEY, C. L. and J. G. FIELD 1989 – General allometric equations for rates of nutrient uptake, ingestion and respiration in plankton organisms. *Limnol. Oceanogr.* **34**: 1290-1299.

PARADA, C. E. 2003 – Modeling the effects of environmental and ecological processes on the transport, mortality, growth and distribution of early stages of Cape anchovy (*Engraulis encrasicolus*) in the Benguela system. PhD thesis, University of Cape Town, 125 + (iii) pp.

PITCHER, G. C., WALKER, D. R., MITCHELL-INNES, B. A. and C. L. MOLONEY 1991 – Short term variability during an anchor station study in the southern Benguela upwelling system: Phytoplankton dynamics. *Prog. Oceanogr.* **28**: 39-64.

STEELE, J. H. and E. W. HENDERSON 1992 – The role of predation in planktonic models. *J. Plankton Res.* **14**: 157-172.

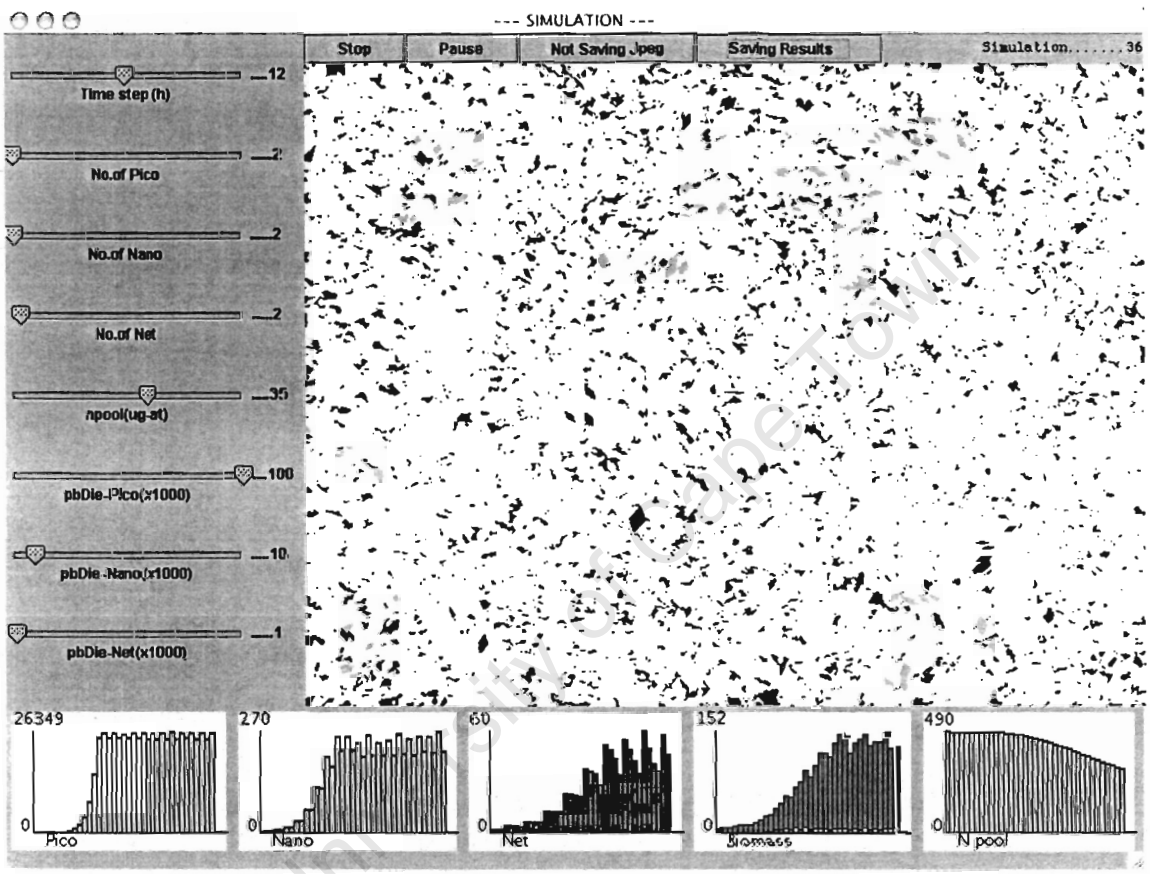
SLACK, J. M. 2000 – Programming and problem solving with java. Library of Congress Cataloging-in-Publication Data; United States of America.

TIMMERMANS, K. R., VAN DER WAGT, B., VELDHUIS, M. J. W., MAATMAN, A. and H. J. W. DE BAAR 2005 –Physiological responses of three species of marine pico-phytoplankton to ammonium, phosphate, iron and light limitation. *J. Sea Res.* **53:** 109-120.

TOURATIER, F., FIELD, J. G. and C. L. MOLONEY 2003 – Simulated carbon and nitrogen flow of the planktonic food web during an upwelling relaxation period in St Helena Bay (southern Benguela ecosystem). *Prog. Oceanogr.* **58**: 1-41.

Appendices

Appendix A:



Appendix B:

Phytoplankton24 class

```
package plankton;

import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
import javax.swing.*;

public class Phytoplankton24 {
public int nPico;
  public int nNano;
  public int nNet;
  public static double nPool;
  Vector thePicoPhytoCells = new Vector();
  Vector theNanoPhytoCells = new Vector();
  Vector theNetPhytoCells = new Vector();
  static double pbDie, pbPicoDie, pbNanoDie, pbNetDie;
  static double pbGraze, pbPicoGraze, pbNanoGraze, pbNetGraze;
  static double biomassPico, biomassNano, biomassNet;
  double[] nb = new double[3];
  static double[] biomass = new double[3];
  static double[] dissolvedPools = new double[2];
  public static final double SIZEEXPONENT = -0.25;
  public static final double VMAXA = 3.6;
  public static final double VKS1 = 2.0;
  public static final double VKS2 = 0.38;
  public static final double RV = 1.7;
  public static final double SP1 = 0.029;
  public static final double SP2 = 0.42;
  public static final double Q10 = 2.0;
  public static final double CN_RATIO = 6.0;
  public static final double EXPONENT = 2;
//-------------------------------------------------------------------------
  public Phytoplankton24() {
  }
//-------------------------------------------------------------------------
  public static String setName(String phytoName) {
    String name = phytoName;
    return name;
  }
//-------------------------------------------------------------------------
  public static double setESD(double min, double max, double ESD) {
    ESD = Math.sqrt(min * max);
    return ESD;

  }
//-------------------------------------------------------------------------
```

```
  public static double setVolume(double ESD, double volume) {
    volume = 4. / 3. * Math.PI * Math.pow(ESD / 2., 3.);
    return volume;
  }
  //---------------------------------------------------------------------------
  public static double setCweight(double volume, double cweight) {
    cweight = 0.4 * Math.pow(volume, 0.75);
    return cweight;
    //else cweight = 0.07 * volume;
  }
//---------------------------------------------------------------------------
  public static double setNweight(double cweight, double nweight) {
    nweight = cweight / CN_RATIO;
    return nweight;
  }
  //---------------------------------------------------------------------------
  public static double setVmax(double cweight, double vmax) {
    vmax = VMAXA * Math.pow(cweight, SIZEEXPONENT);
    return vmax;
  }
//---------------------------------------------------------------------------
  public static double setKs(double cweight, double Ks) {
    Ks = VKS1 * Math.pow(cweight, VKS2);
    return Ks;
  }
//---------------------------------------------------------------------------
  public static double setResp(double cweight, double resp) {
    resp = RV * Math.pow(cweight, SIZEEXPONENT);
    return resp;
  }
//---------------------------------------------------------------------------
  public static double setSink(double cweight, double sink) {
    sink = SP1 * Math.pow(cweight, SP2);
    return sink;
  }

//---------------------------------------------------------------------------
  public void init(int nb, double pb, String ID) {
    pbDie = pb;
    int nCells  = nb;
    //System.out.println(ID +" nCells = " + nCells);
    //nPool = nb;
    if (ID == "pico") {
      nPico = nCells;
      //System.out.print( "nPico" +nPico);
      pbPicoDie = pbDie;
      pbPicoGraze = 0.01f;
      for (int i = 0; i < thePicoPhytoCells.size(); i++) {
        PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
            elementAt(i);
        Grid2d.removeFromACell(cell);
      }
      thePicoPhytoCells.removeAllElements();
```

```
    for (int i = 0; i < nPico; i++) {
      PicoPhytoplankton cell = new PicoPhytoplankton();
      thePicoPhytoCells.add(cell);
    }
   }
   if (ID == "nano") {
    nNano = nCells;
    pbNanoDie = pbDie;
    pbNanoGraze = 0.01f;
    for (int i = 0; i < theNanoPhytoCells.size(); i++) {
      NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
          elementAt(i);
      Grid2d.removeFromACell(cell);
    }
    theNanoPhytoCells.removeAllElements();
    for (int i = 0; i < nNano; i++) {
      NanoPhytoplankton cell = new NanoPhytoplankton();
      theNanoPhytoCells.add(cell);
    }
   }
   if (ID == "net") {
    nNet = nCells;
    pbNetDie = pbDie;
    pbNetGraze =0.01f;
    for (int i = 0; i < theNetPhytoCells.size(); i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
          elementAt(i);
      Grid2d.removeFromACell(cell);
    }
    theNetPhytoCells.removeAllElements();
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = new NetPhytoplankton();
      theNetPhytoCells.add(cell);
    }
   }
  }
//-------------------------------------------------------------------------
  public void move(double timeFactor, String ID) {
   if (ID == "pico") {
    for (int i = 0; i < nPico; i++) {
      PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
          elementAt(i);
      cell.move(timeFactor);
    }
   }
   if (ID == "nano") {
    for (int i = 0; i < nNano; i++) {
     NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
          elementAt(i);
      cell.move(timeFactor);
    }
   }
   if (ID == "net") {
```

```
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
          elementAt(i);
      cell.move(timeFactor);
    }
   }
  }
//---------------------------------------------------------------------------
  public double growth(double nitrogen, double timeFactor, String ID, double totNOut) {
    //double NO3 = NO3.getConc();
    //double NH4 = NH4.getConc();
    //double nitrogen = NO3 + NH4;
    //double nitrogen = 150.0f;
    //PicoPhytoplankton
    nPool = nitrogen;
    //System.out.println( "nPico " + nPico);
    if (ID == "pico") {
      double vmax = timeFactor * PicoPhytoplankton.vmax;
      double Ks = PicoPhytoplankton.Ks;
      double resp = timeFactor * PicoPhytoplankton.resp;
      double v = vmax * (nitrogen) / (nitrogen + Ks);
      for (int i = 0; i < nPico; i++) {
        PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
            elementAt(i);
        double cmass = cell.cellCmass;
        double nmass = cell.cellNmass;
        double respiration = resp * cmass;
        double cuptake = v * cmass + respiration;
        double nuptake = v * nmass;
        cmass = cmass + cuptake - respiration;
        cell.cellCmass = cmass;
        cell.cellNmass = cmass/CN_RATIO;
        //System.out.println("cell" + i + " cmass:" +cmass + " cellCmass " +
cell.cellCmass);
        totNOut = totNOut + (nuptake)/1000.0f;
        //System.out.println("picototNOut " + totNOut );
      }
    //return totNOut;
    }
    if (ID == "nano") {
      double vmax = timeFactor * NanoPhytoplankton.vmax;
      double Ks = NanoPhytoplankton.Ks;
      double resp = timeFactor * NanoPhytoplankton.resp;
      double v = vmax * (nitrogen) / (nitrogen + Ks);
      for (int i = 0; i < nNano; i++) {
        NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
            elementAt(i);
        double cmass = cell.cellCmass;
        double nmass = cell.cellNmass;
        double respiration = resp * cmass;
        double cuptake = v * cmass + respiration;
        double nuptake = v * nmass;
        cmass = cmass + cuptake - respiration;
```

```
      cell.cellCmass = cmass;
      cell.cellNmass = cmass/CN_RATIO;

      //System.out.println("nanototNOut " + totNOut );
    }
  //return totNOut;
  }
   if (ID == "net") {
     double vmax = timeFactor * NetPhytoplankton.vmax;
     double Ks = NetPhytoplankton.Ks;
     double resp = timeFactor * NetPhytoplankton.resp;
     double v = vmax * (nitrogen) / (nitrogen + Ks);
     for (int i = 0; i < nNet; i++) {
       NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
           elementAt(i);
       double cmass = cell.cellCmass;
       double nmass = cell.cellNmass;
       double respiration = resp * cmass;
       double cuptake = v * cmass + respiration;
       double nuptake = v * nmass;
       cmass = cmass + cuptake - respiration;
       cell.cellCmass = cmass;
       cell.cellNmass = cmass/CN_RATIO;

       //System.out.println("net Nuptake " + nuptake );
       totNOut = totNOut + (nuptake)/1000.0f;
       //System.out.println("nettotNOut " + totNOut );
     }
     //return totNOut;
   }
  return totNOut;
  }

//-------------------------------------------------------------------------
  public void reproduce(String ID) {
    if (ID == "pico") {
      Vector picoNewBorn = new Vector();
      double cweight = PicoPhytoplankton.cweight;
      for (int i = 0; i < nPico; i++) {
        PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
            elementAt(i);
        //System.out.println("cell "+i +" cellCmass :" +cell.cellCmass);
        if (cell.cellCmass >= (2.0f * cweight)) {
          Vector picoSpawn = cell.picoSpawn();
          picoNewBorn.addAll(picoSpawn);
          cell.setCellCmass(cweight);
        }
      }
      thePicoPhytoCells.addAll(picoNewBorn);
      nPico = thePicoPhytoCells.size();
    }
    if (ID == "nano") {
      double cweight = NanoPhytoplankton.cweight;
```

```
    for (int i = 0; i < nNano; i++) {
      NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
          elementAt(i);
      if (cell.cellCmass >= (2.0f * cweight)) {
        NanoPhytoplankton replicate = new NanoPhytoplankton(cell);
        theNanoPhytoCells.add(replicate);
        cell.cellCmass = cweight;
      }
    }
    nNano = theNanoPhytoCells.size();
  }
  if (ID == "net") {
    double cweight = NetPhytoplankton.cweight;
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
          elementAt(i);
      if (cell.cellCmass >= (2.0f * cweight)) {
        NetPhytoplankton replicate = new NetPhytoplankton(cell);
        theNetPhytoCells.add(replicate);
        cell.cellCmass = cweight;
      }
    }
    nNet = theNetPhytoCells.size();
  }

 }

//-------------------------------------------------------------------------
  public void sink(double timeFactor, String ID) {
    if (ID == "pico"){
      for (int i = 0; i > nPico; i++) {
        PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
            elementAt(i);
        double pbSink = cell.sink/2000.*timeFactor; // 20m =200decimeter= 2000ml
        if (Math.random() < pbSink) {
          cell.living = false;
        }
      }
    }
    if (ID == "nano"){
      for (int i = 0; i > nNano; i++) {
        NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
            elementAt(i);
        double pbSink = cell.sink/2000.*timeFactor;
        if (Math.random() < pbSink) {
          cell.living = false;
        }
      }
    }
    if (ID == "net"){
      for (int i = 0; i > nNet; i++) {
        NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
            elementAt(i);
```

```
        double pbSink = cell.sink/2000.*timeFactor;
        if (Math.random() < pbSink) {
          cell.living = false;
        }
      }
    }
  }
//---------------------------------------------------------------------------
  public void eval(String ID) {
    if (ID == "pico") {
      nb[0] = nPico;
      biomass[0] = Phytoplankton24.biomassPico/ 1000.0f;//ng C
    }
    if (ID == "nano") {
      nb[1] = nNano;
      biomass[1] = Phytoplankton24.biomassNano / 1000.0f;//ng C
    }
    if (ID == "net") {
      nb[2] = nNet;
      biomass[2] = Phytoplankton24.biomassNet / 1000.0f;//ng
    }
    dissolvedPools[0] = nPool;
  }
//---------------------------------------------------------------------------
  public void mortality(double timeFactor, String ID) {
    //double pbDie = timeFactor * PicoPhytoplankton.pbDie;
    if (ID == "pico") {
      double pbDie = PicoPhytoplankton.pbDie * Math.pow
(Phytoplankton24.biomassPico/1000.0f,EXPONENT);
      for (int i = 0; i < nPico; i++) {
        PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
          elementAt(i);
        System.out.println("pbDiePico" + pbDie);
        if (Math.random() < pbDie) {
          cell.living = false;
          System.out.println();
        }
      }
    }
    if (ID == "nano") {
      double pbDie = NanoPhytoplankton.pbDie * Math.pow
(Phytoplankton24.biomassNano/1000.0f,EXPONENT);
      for (int i = 0; i < nNano; i++) {
        NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
          elementAt(i);
        System.out.println( "pbDieNano " + pbDie);
        if (Math.random() < pbDie) {
          cell.living = false;
        }
      }
    }
    if (ID == "net") {
```

```
    double pbDie = NetPhytoplankton.pbDie * Math.pow
(Phytoplankton24.biomassNet/1000.0f,EXPONENT);
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.elementAt(i);
      //System.out.println( "pbDieNet " + pbDie);
      if (Math.random() < pbDie) {
        cell.living = false;

      }
    }
  }
 }
//---------------------------------------------------------------------------
 public void remove(String ID) {
  double indMass;
  if (ID == "pico") {
   Vector survivals = new Vector();
   for (int i = 0; i < nPico; i++) {
    PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
       elementAt(i);
    if (cell.living) survivals.add(cell);
    else Grid2d.removeFromACell(cell);
   }
   thePicoPhytoCells = survivals;
   nPico = thePicoPhytoCells.size();
   biomassPico = 0.0f;
   for (int i = 0; i < nPico; i++) {
    PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
       elementAt(i);
     indMass = cell.cellCmass;
     biomassPico = biomassPico + indMass;
   }
  }
  if (ID == "nano") {
   Vector survivals = new Vector();
   for (int i = 0; i < nNano; i++) {
    NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
       elementAt(i);
    if (cell.living) survivals.add(cell);
    else Grid2d.removeFromACell(cell);
   }
   theNanoPhytoCells = survivals;
   nNano = theNanoPhytoCells.size();
   biomassNano = 0.0f;
   for (int i = 0; i < nNano; i++) {
    NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
       elementAt(i);
     indMass = cell.cellCmass;
     biomassNano = biomassNano + indMass;
   }
  }
  if (ID == "net") {
   Vector survivals = new Vector();
```

```
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.elementAt(i);
      if (cell.living) survivals.add(cell);
      else Grid2d.removeFromACell(cell);
    }
    theNetPhytoCells = survivals;
    nNet = theNetPhytoCells.size();
    biomassNet = 0.0f;
    for (int i = 0; i < nNet; i++) {
      NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
          elementAt(i);
       indMass = cell.cellCmass;
       biomassNet = biomassNet + indMass;
    }
   }
 }
 //---------------------------------------------------------------------------
 public void trace(Graphics G, int w, int h, String ID) {
   if (ID == "pico") {
     for (int i = 0; i < nPico; i++) {
       PicoPhytoplankton cell = (PicoPhytoplankton) thePicoPhytoCells.
          elementAt(i);
       cell.trace(G, w, h);
     }
   }
   if (ID == "nano") {
     for (int i = 0; i < nNano; i++) {
       NanoPhytoplankton cell = (NanoPhytoplankton) theNanoPhytoCells.
          elementAt(i);
       cell.trace(G, w, h);
     }
   }
   if (ID == "net") {
     for (int i = 0; i < nNet; i++) {
       NetPhytoplankton cell = (NetPhytoplankton) theNetPhytoCells.
          elementAt(i);
       cell.trace(G, w, h);
     }
   }
 }
}
//---------------------------------------------------------------------------
```

SimulationPlankton class

```
package plankton;

import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
```

```
import javax.swing.*;

public class SimulationPlankton extends Simulation{
   static MatrixFrame matrixFrame;
   static Grid2d theGrid = new Grid2d(10);
   Phytoplankton24 phytoplankton24;
   public static int time;
   public static double timeFactor, npoolIncre;
   public double totNOut;
   public double NO3, NH4;
   public boolean calcNpool;
   public double[] npool = new double[500];
   public double[] thePicoParms = new double[10];
   public double[] theNanoParms = new double[10];
   public double[] theNetParms = new double[10];

   Phytoplankton24 PicoPhytoplankton = new Phytoplankton24();
   Phytoplankton24 NanoPhytoplankton = new Phytoplankton24();
   Phytoplankton24 NetPhytoplankton = new Phytoplankton24();
   InputFileReader inputFileReader = new InputFileReader();

   //-------------------------------------------------------------------------
   public SimulationPlankton() {
     matrixFrame = new MatrixFrame(this);
     matrixFrame.addSliderInPanel("Time step (h)", 1, 0, 24);
     matrixFrame.addSliderInPanel("No.of Pico", 2, 0, 6000);
     matrixFrame.addSliderInPanel("No.of Nano", 2, 0, 600);
     matrixFrame.addSliderInPanel("No.of Net", 2, 0, 60);
     matrixFrame.addSliderInPanel("npool(ug-at)", 35, 0, 60);
     matrixFrame.addSliderInPanel("pbDie-Pico(x1000)", 1, 0, 100);
     matrixFrame.addSliderInPanel("pbDie-Nano(x1000)", 1, 0, 100);
     matrixFrame.addSliderInPanel("pbDie-Net(x1000)", 1, 0, 100);
     matrixFrame.addGraphicInPanel("Pico", PicoPhytoplankton.nb);
     matrixFrame.addGraphicInPanel("Nano", NanoPhytoplankton.nb);
     matrixFrame.addGraphicInPanel("Net", NetPhytoplankton.nb);
     matrixFrame.addGraphicInPanel("Biomass", Phytoplankton24.biomass);
     matrixFrame.addGraphicInPanel("N pool", Phytoplankton24.dissolvedPools);
     matrixFrame.endFrame();
     U.initRainbow();
     npool = InputFileReader.nitrogen;
     npoolIncre = 0.1;
   }

  //-------------------------------------------------------------------------
  public void init(int[] theValues) {
    int nPico = theValues[1];
    int nNano = theValues[2];
    int nNet = theValues[3];
    timeFactor = (double) theValues[0] / 24.0f;
// Use the data file if no npool input from the graphics panel
    if (theValues[4] > 0) {
      npool[0] = 14.0f * (double) theValues[4];
      calcNpool = true;
```

```
  }
  else {
    double[] npoolNew = new double[1000];
    double[] newX = new double[1000];
    double[] oldX = new double[1000],
    boolean[] arrayMatch = new boolean[1000];
    npoolNew[0] = npool[0];
    newX[0] = 0;
    oldX[0] = 0;
    int iNewMax = (int) ( (20 / timeFactor) + 1);
    for (int iNew = 1; iNew <= iNewMax; iNew++) {
      newX[iNew] = newX[iNew - 1] + timeFactor;
      arrayMatch[iNew] = false;
    }
    for (int iOld = 1; iOld <= 200; iOld++) {
      oldX[iOld] = oldX[iOld - 1] + npoolIncre;
    }
    for (int iNew = 1; iNew <= iNewMax; iNew++) {
      do {
        for (int iOld = 1; iOld <= 200; iOld++) {
          if (newX[iNew] >= oldX[iOld - 1]) {
            if (newX[iNew] < oldX[iOld]) {
              npoolNew[iNew] = ( (npool[iOld] - npool[iOld - 1]) / 0.1) *
                (newX[iNew] - oldX[iOld - 1]) + npool[iOld - 1];
              arrayMatch[iNew] = true;
            }
          }
        }
      }
      while (arrayMatch[iNew] = false);
    }
    npool = npoolNew;
  }
  double pbPicoDie = (double) theValues[5] / 100.0f;
  double pbNanoDie = (double) theValues[6] / 1000.0f;
  double pbNetDie = (double) theValues[7] / 10000.0f;
  PicoPhytoplankton.init(nPico, pbPicoDie, "pico");
  NanoPhytoplankton.init(nNano, pbNanoDie, "nano");
  NetPhytoplankton.init(nNet, pbNetDie, "net");
  time = 0;
}
//---------------------------------------------------------------------------
public void step(int[] theValues) {
  totNOut = 0.0f;

  PicoPhytoplankton.move(timeFactor, "pico");
  NanoPhytoplankton.move(timeFactor, "nano");
  NetPhytoplankton.move(timeFactor, "net");

  //System.out.println("totNOut 1 " + totNOut );
  totNOut =+ PicoPhytoplankton.growth(npool[time], timeFactor, "pico", totNOut);
  //System.out.println("totNOut 2 " + totNOut );
  totNOut =+ NanoPhytoplankton.growth(npool[time], timeFactor, "nano", totNOut);
```

```
  //System.out.println("totNOut 3 " + totNOut );
  totNOut =+ NetPhytoplankton.growth(npool[time], timeFactor, "net", totNOut);
  //System.out.println("totNOut final " + totNOut );
  if (calcNpool = true) {
    npool[time+1] = npool[time] - totNOut;
  }

  PicoPhytoplankton.reproduce("pico");
  NanoPhytoplankton.reproduce("nano");
  NetPhytoplankton.reproduce("net");

  PicoPhytoplankton.sink(timeFactor, "pico");
  NanoPhytoplankton.sink(timeFactor, "nano");
  NetPhytoplankton.sink(timeFactor, "net");

  PicoPhytoplankton.mortality(timeFactor, "pico");
  NanoPhytoplankton.mortality(timeFactor, "nano");
  NetPhytoplankton.mortality(timeFactor, "net");

  PicoPhytoplankton.remove("pico");
  NanoPhytoplankton.remove("nano");
  NetPhytoplankton.remove("net");

  PicoPhytoplankton.eval("pico");
  NanoPhytoplankton.eval("nano");
  NetPhytoplankton.eval("net");

  time++;
 }
 //-------------------------------------------------------------------------
 public void end() {
 }
//-------------------------------------------------------------------------
 public void trace(Graphics G, int w, int h) {
  try {
   G.setColor(new Color(0, 0, 0));
   G.fillRect(0, 0, w, h);
   PicoPhytoplankton.trace(G, w, h,"pico");
   NanoPhytoplankton.trace(G, w, h, "nano");
   NetPhytoplankton.trace(G, w, h, "net");
  }
  catch (java.util.NoSuchElementException e) {
  }
 }
//-------------------------------------------------------------------------
 public static void main(String[] args) throws java.io.IOException,
    java.text.ParseException {
  try {

UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
  }
  catch (Exception e) {}
  java.util.Date today;
```

```
    today = new Date();
    System.out.println(today);
    SimulationPlankton simulationPlankton = new SimulationPlankton();
  }
}
//--------------------------------------------------------------------------
```

Pico-Phytoplankton class

```
package plankton;
import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
import javax.swing.*;
import apple.awt.*;

public class PicoPhytoplankton extends Individual2d{
  public String PicoPhytoplankton;
  public String name;
  public static double[] thePicoParms = new double [10];
  public static double ESD, volume, cweight, nweight;
  public static double vmax, Ks, resp, sink, pbDie;
  double cellCmass, cellNmass;
  boolean living;
  //-------------------------------------------------------------------------
  public PicoPhytoplankton() {
    name = Phytoplankton24.setName ("Pico");
    ESD = Phytoplankton24.setESD (0.2,2., ESD);            // (um)
    volume = Phytoplankton24.setVolume(ESD, volume);       // (cubic um)
    cweight = Phytoplankton24.setCweight(volume, cweight); // (pg C)
    nweight = Phytoplankton24.setNweight(cweight, nweight);// (pg N)
    vmax = Phytoplankton24.setVmax(cweight, vmax);         // (per day)
    Ks = Phytoplankton24.setKs(cweight, Ks);               // (ug.N per cubic cm)
    resp = Phytoplankton24.setResp(cweight, resp);         // (per day)
    sink = Phytoplankton24.setSink(cweight, sink);         // (cm per day)
    pbDie = Phytoplankton24.pbPicoDie;                     // (value between 0&1)
    x = Math.random();
    y = Math.random();
    vx = 0.01f * U.alea0();
    vy = 0.01f * U.alea0();
    this.setCellCmass(cweight); // carbon biomass (pg C)
    this.getCellCmass();
    cellNmass = nweight; // nitrogen biomass (pg N)
```

```
  living = true;
  type = "PicoPhytoplankton";
  Grid2d.putInACell (this);
 }
 //-------------------------------------------------------------------------
 public PicoPhytoplankton (PicoPhytoplankton mother){

  x = mother.x + 0.01f * U.alea0();
  y = mother.y + 0.01f * U.alea0();
  vx = mother.vx;
  vy = mother.vy;
  this.setCellCmass(cweight); // carbon biomass (pg C)
  this.getCellCmass();
  cellNmass = nweight; // nitrogen biomass (pg N)
  living = true;
  vx += 0.02f * U.alea0();
  vy += 0.02f * U.alea0();
  double xa = x;
  double ya = y;
  x += 0.2f * vx;
  y += 0.2f * vy;
  rebord();
  Grid2d.changeCell(xa, ya, this);
 }
 //-------------------------------------------------------------------------

 public Vector picoSpawn() {
  Vector children=new Vector();
  PicoPhytoplankton child = new PicoPhytoplankton(this);
  children.add(child);
  return (children);
  }
//-------------------------------------------------------------------------
  public void setCellCmass(double cweight) {
   this.cellCmass = cweight;
  }
  public double getCellCmass() {
   return cellCmass;
  }
//-------------------------------------------------------------------------
  public void move(double timeFactor) {
   vx += 0.02f * U.alea0();
   vy += 0.02f * U.alea0();
   double xa = x;
   double ya = y;
   x += 0.2f * vx;
   y += 0.2f * vy;
   rebord();
   Grid2d.changeCell(xa, ya, this);
  }
  //-------------------------------------------------------------------------
  public void trace(Graphics G, int w, int h) {
   int ix = (int) (w * x);
```

```
    int iy = (int) (h * y);
    double n = U.norm(vx, vy);
    double c = 0.02f;
    int dx = (int) (c * w * vx * 4.0f);
    int dy = (int) (c * h * vy * 4.0f);
    int ddx = (int) (c * w * vy * 2.0f);
    int ddy = (int) ( -c * h * vx * 2.0f);
    Polygon p = new Polygon();
    p.addPoint(ix - dx, iy - dy);
    p.addPoint(ix + ddx, iy + ddy);
    p.addPoint(ix + dx, iy + dy);
    p.addPoint(ix - ddx, iy - ddy);
    G.setColor(Color.yellow);
    G.fillPolygon(p);
    G.setColor(Color.yellow);
    G.drawPolygon(p);
   }
}

//---------------------------------------------------------------------------
```

Nano-Phytoplankton class

```
package plankton;
import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
import javax.swing.*;
import apple.awt.*;

public class NanoPhytoplankton extends Individual2d{
  public String NanoPhytoplankton;
  public String name;
  public static double[] theNanoParms = new double [10];
  public static double ESD, volume, cweight, nweight;
  public static double vmax, Ks, resp, sink, pbDie;
  double cellCmass, cellNmass;
  boolean living;
  //---------------------------------------------------------------------------
  public NanoPhytoplankton() {
    name = Phytoplankton24.setName ("Nano");
    ESD = Phytoplankton24.setESD (2.,20., ESD);              // ESD (um)
    volume = Phytoplankton24.setVolume(ESD, volume);         // (cubic um)
    cweight = Phytoplankton24.setCweight(volume, cweight);  // (pg C)
    nweight = Phytoplankton24.setNweight(cweight, nweight); // (pg N)
    vmax = Phytoplankton24.setVmax(cweight, vmax);           // (per day)
    Ks = Phytoplankton24.setKs(cweight, Ks);                 // (ug N cubic cm)
    resp = Phytoplankton24.setResp(cweight, resp);           // (per day)
```

```
  sink = Phytoplankton24.setSink(cweight, sink);          // (cm per day)
  System.out.println();
  pbDie = Phytoplankton24.pbNanoDie;                      // (value between 0&1)
  x = Math.random();
  y = Math.random();
  vx = 0.01f * U.alea0();
  vy = 0.01f * U.alea0();
  this.setCellCmass (cweight);     // carbon biomass (pg C)
  this.getCellCmass();
  cellNmass = nweight;             // nitrogen biomass (pg N)
  living = true;
  type = "NanoPhytoplankton";
  Grid2d.putInACell (this);
 }
 //-------------------------------------------------------------------------

 public NanoPhytoplankton (NanoPhytoplankton mother){
   x = mother.x + 0.01f * U.alea0();
   y = mother.y + 0.01f * U.alea0();
   vx = mother.vx;
   vy = mother.vy;
   this.setCellCmass(cweight); //carbon biomass (pg C)
   this.getCellCmass();
   living = true;
   type = "NanoPhytoplankton";
   Grid2d.putInACell(this);
 }
//-------------------------------------------------------------------------
 public Vector nanoSpawn(){
   Vector children = new Vector();
   NanoPhytoplankton child = new NanoPhytoplankton (this);
   children.add(child);
   return (children);
 }
 //-------------------------------------------------------------------------
 public void setCellCmass(double cweight){
   this.cellCmass = cweight;
 }
 public double getCellCmass(){
   return cellCmass;
 }
 //-------------------------------------------------------------------------
  public void move(double timeFactor){
    vx += 0.02f * U.alea0();
    vy += 0.02f * U.alea0();
    double xa = x;
    double ya = y;
    x += 0.2f * vx;
    y += 0.2f * vy;
    rebord();
    Grid2d.changeCell(xa, ya, this);
  }
  //-------------------------------------------------------------------------
```

```
 public void trace (Graphics G, int w, int h){
   int ix = (int) (w*x);
   int iy = (int) (h*y);
   double n=U.norm(vx, vy);
   double c=0.02f;
   int dx = (int) (c * w * vx * 4.0f);
   int dy = (int) (c * h * vy * 4.0f);
   int ddx = (int) (c * w * vy * 2.0f);
   int ddy = (int) (-c * h * vx * 2.0f);
   Polygon p = new Polygon();
   p.addPoint (ix - dx, iy - dy);
   p.addPoint (ix + ddx, iy + ddy);
   p.addPoint (ix + dx, iy + dy);
   p.addPoint (ix - ddx, iy - ddy);
   G.setColor (Color.green);
   G.fillPolygon(p);
   G.setColor(Color.green);
   G.drawPolygon(p);
 }
}
//---------------------------------------------------------------------------
```

Net-Phytoplankton class

```
package plankton;

import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
import javax.swing.*;
import apple.awt.*;


 public class NetPhytoplankton extends Individual2d{
 public String NetPhytoplankton;
 public String name;
 public static double[] theNetParms = new double [10];
 public static double ESD, volume, cweight, nweight;
 public static double vmax, Ks, resp, sink, pbDie;
 double cellCmass, cellNmass;
 boolean living;
 //---------------------------------------------------------------------------
 public NetPhytoplankton() {
   name = Phytoplankton24.setName ("Net");
   ESD = Phytoplankton24.setESD (25.,125., ESD);          // (ESD um)
   volume = Phytoplankton24.setVolume(ESD, volume);       // (cubic um)
   cweight = Phytoplankton24.setCweight(volume, cweight); // (pg C)
   nweight = Phytoplankton24.setNweight(cweight, nweight); // (pg N)
   vmax = Phytoplankton24.setVmax(cweight, vmax);         // (per day)
```

```
  Ks = Phytoplankton24.setKs(cweight, Ks);              // (ug C per cubic cm)
  resp = Phytoplankton24.setResp(cweight, resp);        // (per day)
  sink = Phytoplankton24.setSink(cweight, sink);        // (cm per day)
  pbDie = Phytoplankton24.pbNetDie;                     // ( value between 0&1
  x = Math.random();
  y = Math.random();
  vx = 0.01f * U.alea0();
  vy = 0.01f * U.alea0();
  this.setCellCmass(cweight); //carbon biomass (pg C)
  this.getCellCmass();
  cellNmass = nweight;
  living = true;
  type = "NetPhytoplankton";
  Grid2d.putInACell (this);
 }
 //---------------------------------------------------------------------------
 public NetPhytoplankton (NetPhytoplankton mother){
  x = mother.x + 0.01f * U.alea0();
  y = mother.y + 0.01f * U.alea0();
  vx = mother.vx;
  vy = mother.vy;
  this.setCellCmass(cweight); //carbon biomass (pg C)
  this.getCellCmass();
  living = true;
  type = "NetPhytoplankton";
  Grid2d.putInACell(this);
 }
//---------------------------------------------------------------------------
 public void move(double timeFactor){
  vx += 0.02f * U.alea0();
  vy += 0.02f * U.alea0();
  double xa = x;
  double ya = y;
  x += 0.2f * vx;
  y += 0.2f * vy;
  rebord();
  Grid2d.changeCell(xa, ya, this);
 }
 //---------------------------------------------------------------------------
 public Vector netSpawn(){
   Vector children = new Vector();
   NetPhytoplankton child = new NetPhytoplankton (this);
   children.add(child);
   return (children);
 }
 //---------------------------------------------------------------------------
 public void setCellCmass(double cweight){
  this.cellCmass = cweight;
 }
 public double getCellCmass(){
  return cellCmass;
 }
```

```
 public void trace (Graphics G, int w, int h){
  int ix = (int) (w*x);
  int iy = (int) (h*y);
  double n=U.norm(vx, vy);
  double c=0.02f;
  int dx = (int) (c * w * vx * 4.0f);
  int dy = (int) (c * h * vy * 4.0f);
  int ddx = (int) (c * w * vy * 2.0f);
  int ddy = (int) (-c * h * vx * 2.0f);
  Polygon p = new Polygon();
  p.addPoint (ix - dx, iy - dy);
  p.addPoint (ix + ddx, iy + ddy);
  p.addPoint (ix + dx, iy + dy);
  p.addPoint (ix - ddx, iy - ddy);
  G.setColor (Color.blue);
  G.fillPolygon(p);
  G.setColor(Color.blue);
  G.drawPolygon(p);
 }
 }
//--------------------------------------------------------------------------
```

InputFileReader class

```
package plankton;

import matrix.*;       //Christian Mullon's libraries
import java.awt.*;     //standard Java libraries
import java.util.*;
import java.lang.*;
import java.io.*;
import java.text.*;
import javax.swing.*;

public class InputFileReader {
 public static double[] nitrogen = new double[500];
 public InputFileReader(){
 String nextInput;
  double nextValue;
  int lineNumber = 0;
  NumberFormat aNumberFormatter = NumberFormat.getInstance();
  File inFile = new File("/Users/ntobi/jbproject/plankton/plankton/plankton/nitrogen.txt");
  System.out.println("Input file exists? " +inFile.exists());
  System.out.println("Can read input file?" +inFile.canRead());
  if (inFile.exists() && inFile.canRead()) {
   try { //Create an input stream and attach it to the file
    BufferedReader fileInStream = new BufferedReader (new FileReader(inFile));
    nextInput = fileInStream.readLine();
    while (nextInput!= null) {
     try {
      StringTokenizer tokenizer = new StringTokenizer(nextInput);
      nextValue = aNumberFormatter.parse(tokenizer.nextToken()).doubleValue();
```

```
        nitrogen[lineNumber] = nextValue;
        lineNumber++;
        nextInput = fileInStream.readLine();
      }
      catch (java.text.ParseException e) {
        System.out.println("can't parse - please check InputFilereader");
      }
    }
      fileInStream.close();
    }
    catch (java.io.IOException e) {
      System.out.println("IO error - please check InputFilereader");
    }
  }
 }

}
```