NTNU
Innovation and Creativity

Reduction of search space using group-of-experts and RL.

Tore Rune Anderson

Master of Science in Computer Science
Submission date: January 2007
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

This is a novel approach in trying to reduce the search space by having the input signals being worked upon, and hence reduced, before being presented to the problem solver. Standard Reinforcement Learning systems get bogged down in the huge search spaces created by fine-resolution input states. But, unfortunately, it is not enough to just manually reduce the resolution, since it is not always clear exactly what details need to be abstracted away and what details are necessary for proper action selection. I will try the group-of-experts approach to finding the proper abstraction level.

The reduction will be achieved by having different experts work on the raw input signals, which can be of great magnitude, and give their suggestions on what actions to perform to an action selector. In this way, all the input signals are being predigested by some experts, and the search area in which the agent must learn how to react is greatly reduced to the output signals of these experts.

This diploma builds on the project I delivered this spring with the focus on action selection combined with reinforcement learning.

Assignment given: 28. August 2006
Supervisor: Keith Downing, IDI

NTNU
Innovation and Creativity

Reduction of search space using group-of-experts approach

Tore Rune Anderson

Master of Science in Computer Science
Submission date: 22nd of January, 2007
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

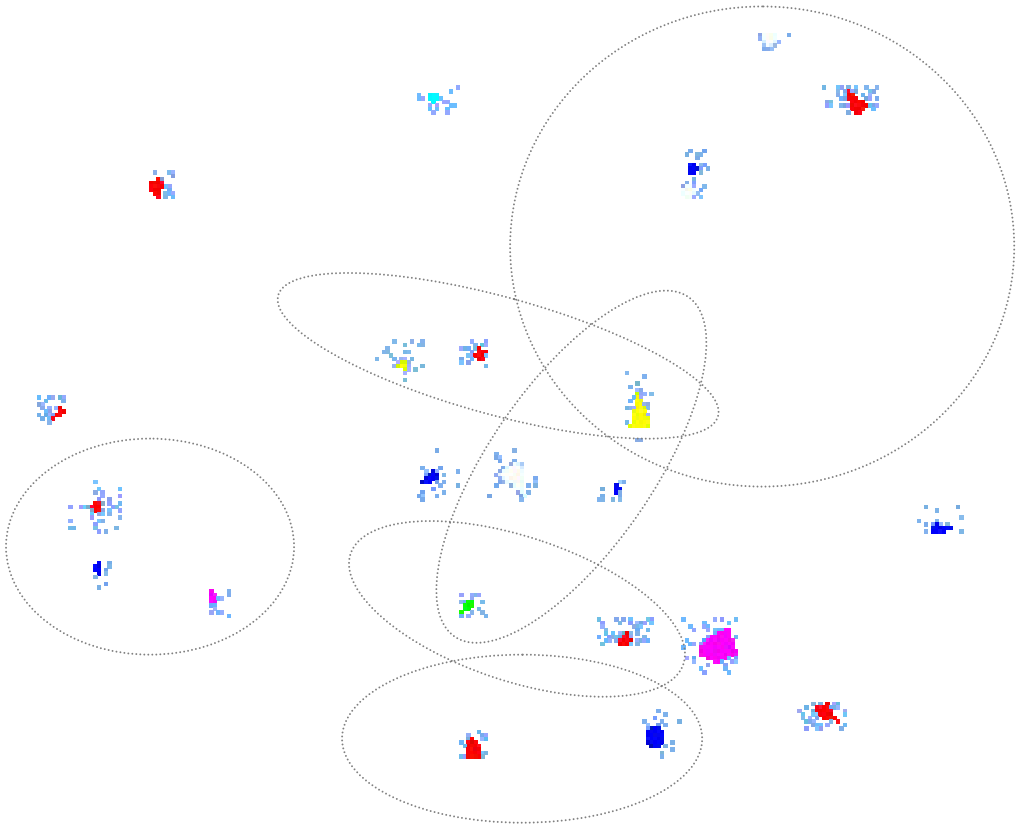
This is a novel approach in trying to reduce the search space by having the input signals being worked upon, and hence reduced, before being presented to the problem solver. Standard Reinforcement Learning systems get bogged down in the huge search spaces created by fine-resolution input states. But, unfortunately, it is not enough to just manually reduce the resolution, since it is not always clear exactly what details need to be abstracted away and what details are necessary for proper action selection. I will try the group-of-experts approach to finding the proper abstraction level.

The reduction will be achieved by having different experts work on the raw input signals, which can be of great magnitude, and their filtered input signals makes the foundation for what actions to perform to an action selector. In this way, all the input signals are being pre-digested by some experts, and the search area in which the agent must learn how to react is greatly reduced to the output signals of these experts.

This diploma builds on the project I delivered this spring with the focus on action selection combined with reinforcement learning.

Assignment given: august 2006.
Supervisor: Keith Downing

Reduction of search space using group-of-experts approach



Tore Anderson
IDI, NTNU
tore.anderson@gmail.com

Abstract

This Master thesis implements an architecture that reduces the search space used in reinforcement learning by a group of experts regime. The experts filter the original input signals, so the reinforcement learning system is left with a much smaller search space. The main goal of the thesis is to find at what abstraction level this filtering should be done at.

The experience gained through experimenting suggests that the filtered data need to be filtered intelligently, and the usage of the filtered data is processes on a high abstraction level. This thesis did not succeed in implanting an architecture that successfully deals with a huge input search space. The experiments has been done on a moderately low scale and magnitude, but is none the less a good starting point for this novel approach of combating one of the obstacles found in reinforcement learning systems: their tendency to get stuck in huge search spaces.

Other challenges connected with reinforcement learning still apply, and the potential success of a reinforcement learning system is still depended on a lot of obstacles and challenges to be overcome by the designers.

It is the hope that this thesis is a contribution in this field, and that this approach might be useful and interesting for others working with reinforcement learning systems.

Preface

This report is the result of the master thesis, which is the final project for the Master degree in Computer Science at the Norwegian University of Science and Technology (NTNU). This thesis is titled *Reduction of search space using group-of-experts approach.* The defined goal for the project goes as follows:

> *This is a novel approach in trying to reduce the search space by having the input signals being worked upon, and hence reduced, before being presented to the problem solver. Standard Reinforcement Learning systems get bogged down in the huge search spaces created by fine-resolution input states. But, unfortunately, it is not enough to just manually reduce the resolution, since it is not always clear exactly what details need to be abstracted away and what details are necessary for proper action selection. I will try the group-of-experts approach to finding the proper abstraction level.*
>
> *The reduction will be achieved by having different experts work on the raw input signals, which can be of great magnitude, and their filtered input signals makes the foundation for what actions to perform to an action selector. In this way, all the input signals are being pre-digested by some experts, and the search area in which the agent must learn how to react is greatly reduced to the output signals of these experts.*
>
> *This diploma builds on the project I delivered this spring with the focus on action selection combined with reinforcement learning.*

The simulator environment chosen to conduct the experiments was the BREVE simulator.

Finally, I would like to thank my supervisor Keith Downing for his input and for being available and understanding.

Holmestrand, January 22, 2007

Tore Rune Anderson

Contents

List of figures

1 Introduction

As technology and knowledge keep developing, new possibilities and problem domains surface the world of computer technology. Old techniques can be refined and further developed and new possibilities raise new challenges. An agent is a software program designed to solve certain problems. The complexity of the world the agents live could make it impossible for the programmer to predict all possible scenarios, and the need for the agents capability to learn in runtime by interaction with its environment arises. Agents existing in an online environment need to be able to adapt and learn continuously, and reinforcement learning is good technique for achieving this.

Nearly all theories of learning and intelligence are based on the foundational idea of learning from interaction [5]. The capability of agents learning in runtime is a need that has risen in the field of Artificial Intelligence (AI). Machine learning is a domain that has existed for decades, and offers generally three main categories of learning, namely supervised, unsupervised and reinforcement learning. Reinforcement learning has some qualities and advantages that make it the preferred choice in certain problem domains where learning is desired.

However, one challenging aspect is the size of the search space, commonly known as the scaling problem, or the curse of dimensionality. This thesis suggests a new possible strategy on how to deal with this problem.

1.1 Motivation

All though reinforcement learning is a preferred choice in many problem domains, as will be stated in chapter 2, RL has some major drawbacks, too. One of the most pronounced currently might be its problem dealing with huge search spaces. If there are many parameters to be set during learning, the RL system tends to get bogged down, [11], [13][22]. Reinforcement learning generally maps the various possible states with the various possible actions to perform. Learning from experience, the agent needs to both explore and exploit, it needs to build up knowledge through experience, and use the gained knowledge in choosing future actions. The exponential growth of the number of parameters to be learned with the size of any compact encoding of system state is often referred to as the curse of dimensionality [13]. This leads to the core of the problem of this thesis: reducing the search space by the use of group-of-experts regime.

There are many various approaches trying to deal with the challenge of huge search space in context with reinforcement learning. I would like to experiment with a novel approach, namely by having the search space reduced by experts, so that the input space in which the RL system works in, is greatly reduced. Similar approaches have been made, but generally lower level experts present possible actions to a higher level to choose between. The difference is that the lower level experts do not produce suggestions on what actions to choose, rather, they simply filter the input signals, and pass on only parts of the original input space. Instead of dealing with a huge input space, the higher level module of the agent is presented with different filtered input signals. The goal of the agent is to achieve the following: **the combination of the actual accessible information stored in the different filtered input signals, combined with the experience of which expert provides which input signals, should allow the agent to learn its task and exhibit desired behaviour**. If this is successful, the agent can then exist in much bigger and more complex worlds, and still benefit from the RL technique by simply having the search space reduced by lower level experts.

1.2 Research question

Assuming that it is possible to filter a huge search space in order to obtain a much smaller, and hence, feasible search space for a reinforcement system to deal with, the research question for this thesis can be formulated as follows:

At what abstraction level should this be done?

How much intelligence and processing of information is needed within the lower level experts, and what level of information processing is then the higher level agent left to work with? These are important aspects to consider, and deserves attention during the experimentation.

1.3 Methodology

I will use the BREVE simulator to experiment with this, as this simulator is free and open source. It makes it fairly easy to set up simulations of agents, and is a popular simulator used in areas of artificial intelligence dealing with multi-agent system and artificial life. Reinforcement learning is typically used in an online environment, and a technique often used by situated or real agents. The existence of the BREVE simulator was pointed out to be me by my supervisor Downing, and has been a good simulator to work with.

My approach will be to set up an environment in the BREVE simulator, in which an agent exists and uses the RL technique to adjusting to its environment. As the BREVE simulator easily allows you to visualize your simulations, the acting and behaviour of the agent can be observed visually. I will design an RL architecture that involves action selection and reducing the search space by the use of group-of-experts regime, and based on the experiments, I will discuss and conclude upon the results.

1.4 The aspiration level

The idea behind the reducing the search space by having it filtered by experts represents a new angle on how to overcome one obstacle in RL systems. This thesis does not revolutionary solve all problems connected with RL systems. I do hope that this work will be an entry point for an approach that might become a great resource in the RL methodology, and that this novel approach might be found interesting and valuable in the area of artificial intelligence dealing with RL systems.

1.5 Organization

The thesis is organized as follows: first, in chapter 2, comes a presentation of obstacles typically found in standard Reinforcement learning systems, and some approaches that has been tried in order to deal with the challenges. Action selection is then presented in chapter 3, describing its major categories of action selection, as well as a short presentation of some selected architectures of action selection. This is to give the reader an understanding of both the reinforcement learning and action selection disciplines before describing my architecture design in chapter 4.

I will evaluate my experiments and experiences in chapter 5, and finally make a conclusion in chapter 6. Some ideas for future work are presented in chapter 7.

2 Reinforcement learning

I will start this chapter by introducing the three main categories of learning, namely supervised, unsupervised and reinforcement learning (RL) in chapter 2.1, and argue why RL is my preferred choice in the setting of this thesis. Next I will give a description of reinforcement learning by listing its main properties and showing some examples of formulas for RL and the basic algorithm for value and policy updating, chapter 2.2 and 2.3. I will look at some challenges accompanying using RL, and specially focus on the scaling problem. The scaling problem will be illustrated by showing the process and describing why it is so computational demanding. This challenge is the obstacle this thesis offers a strategy to overcome.

2.1 Main categories of learning

I will here describe the 3 main groups of learning found in machine learning, namely supervised, unsupervised and reinforcement learning. Supervised learning is characterized by learning from labeled examples. In unsupervised learning one creates clusters from unlabeled examples. And finally, reinforcement learning is learning from interaction.

2.1.1 Supervised learning

Supervised learning is characterized by learning from labeled examples. A very typical and well used method of implementing supervised learning is back propagation. Using a neural network, with input nodes, hidden nodes and output nodes, the back propagation learning works by computing an error signal for the output neurons and spread it out over the hidden neurons. The error is calculated by comparing the actual output up against the desired, or correct, output. The weights are updated in order to increase the error given the input and desired output. Back propagation is a computer technique used in machine learning that does not have its counterpart in nature. Supervised learning, as the name states, needs an outside teacher as guidance, as illustrated in Figure 1. Whether that is the programmers knowledge put into the system at design time or some inputs the system gets at runtime, this model becomes generally unattractive dealing with for instance agents performing actions in real time. Also, in many problem domains, there exists no predefined correct answer. This could be simply because the correct answers are not known in advance. If the environment the agent exists in is changing continuously so that the state of the world, and hence any correct answers, are subject to change, they cannot be defined in advance. This is a characteristic found in the real world – there are simply too many factors to consider, making it impossible to be able to predict all future states of the world. There exist no teachers. If an agent is to learn, it needs to do so by other means that supervised learning.

Figure 1: Supervised learning

In supervised learning the agent receives some input and produces an output. This output is then measured up against a target output, the correct answer, provided by a "teacher". The

agent makes adjustment in order to increase the gap between his output and the target output.

2.1.2 Unsupervised learning

Unsupervised learning is assumed to be much more likely common in the brain than supervised learning. Here the agent receives some input, and based on that in produces some output. It has no target, or predefined correct answer, to measure the result up against, see Figure 2. There is no teacher telling you what is right and wrong.

Figure 2: Unsupervised learning

In unsupervised learning the agent receives some input, and from that it makes an output. It has no feedback from a "teacher". Its only knowledge comes from its input and its produced output. Given this knowledge, gained over time, the agent can for instance learn to group the input signals into clustered groups.

Clustering is a good example of unsupervised learning. The task of the unsupervised learning is to find and characterize structures in the input, preferably on a low dimension. To illustrate, each eye has 10^6 photoreceptors [17]. It the task is to learn to differentiate between oranges and apples unsupervised learning will hopefully be able to detect, categorize and cluster sufficiently the input patters on a much lesser dimension than 10^6. Bayesian network is a well known technique used in unsupervised learning, where one comes to the conclusions probabilistically warranted rather that logically [18]. In [19] we get examples of how Bayesian networks can be used to learn casual relationships. Unsupervised learning does not directly result in a difference in the agent's behavior, as the outputs are only internal representations. These representations can however be used other parts of the agent in a way that affects the behavior, a technique often used in perceptual systems. If the input signals are changing, this technique becomes unattractive – the clustering, or learning, achieved in the past becomes quickly of no use in a new world where the input signals are different from the ones used during the learning period. Reinforcement learning offers a method of learning while existing in a changing environment.

2.1.3 Reinforcement learning

Supervised and unsupervised learning are quite opposite approaches to learning. Reinforcement is often considered as its own category, and is closer to supervised than unsupervised learning. It receives its feedback from the environment from which it exists in, and uses changes in the environment as a tool in the learning, see Figure 3.

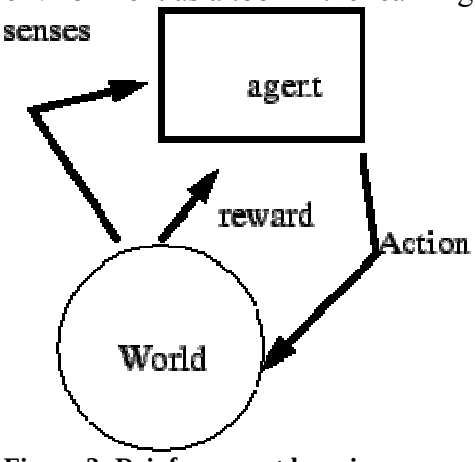


Figure 3: Reinforcement learning

In reinforcement learning the agent receives input signals and produces an output signal. It reacts in its environment, and changes in the environment give the agent feedback on its actions. The reward signal is defined to be outside of the agent's control. This reward is then used to adjust the agent's behavior and learning. The agent has no knowledge of a perfect answer, it only has knowledge of its input signals, its output signals and to some degree the effect its behavior has on the environment.

As opposed to supervised learning, where the agent can compare its output with the correct, desired output, RL receives a scalar input, a reward, from the environment. This scalar does not reveal any optimal, desired action; it only gives a value to the state the agent is in. It is up to the agent to use this scalar input to increase its expected reward signals in the future. It uses its experience to improve its performance. An interesting and distinctive aspect of RL that distinguishes RL from other typical machine learning forms, is that the programmer does not tell the agent how to solve the problem, the programmer only tells the agent what goal to achieve. The agent must discover which actions yield the most reward by trying them.

There are basically two approaches in RL. One is to learn the value function, meaning how to predict a value to a certain state. The other is to learn the policy function, meaning how to use the value function in order to choose actions.

To give an understanding of RL, I will next describe the elements of RL, and then give some examples of formulas for RL and show the basic algorithm for value and policy updating.

2.2 The elements of reinforcement learning

The example below illustrates a typical reinforcement learning iteration form an outsider, observant, perspective:

…
You are in state 45, you have a total of 4 actions to choose from.
You choose action 2 which puts you in state 44.
You receive a reward of value 1.
You are now in state 44, you have a total of 3 possible actions to choose from.
You choose action 1, which puts you in state 88.
You receive a reward of value -1.
You are now in state 88, you have a total of 7 actions to choose from.
You choose…

The state is here represented as integers. Being in a certain state, identified by an integer, gives you some various options of what action to choose next. Your action will possible put you in a new state, and you might possible receive a reward signal from the environment. Your task is to learn the optimal behavior in order to accumulate the highest possible rewards over a longer time period. This mapping of state and action is called the policy [9].

There are typically four elements to a reinforcement learning system, namely a policy, a reward function, a value function and optionally a model of the environment. S denotes the space of possible states the agent can be in, and A denotes the possible actions it can perform.

The policy (π) is the decision making function of the agent. It specifies what action A to take being in state S.

The reward function (r) defines a goal of the reinforcement learning agent. The definition of this function is greatly depended on the task and what we want the agent to learn. A simple reward function is to give a value of 1 whenever the goal is reached, and 0 for all other states, or equally opposite, to give a reward of -1 whenever an undesired state is reached, and 0 otherwise. It could have any numeric value with no boundaries to either a top score or a low score. Generally, a reward function gives a numerical reward signal to the agent based entirely on the state of the environment (which the agent is a part of), and not being under control of the agent.

The *value function* (v) specifies what is good in the long run. To exemplify the difference between the reward function and the value function, an example from the game of chess is illustrative. Checkmating your opponent is associated with a high reward, but winning his queen is associated with a high value [5]. For an agent to achieve its goal, it is of high interest to learn the value of the different states.

The last and optional element, *a model of the environment*, is an internal representation of the environment of some kind. It could be that this model predicts the resultant next state and next reward. This is a part of the reinforcement system that potentially requires most storage space. A reward function and a value function might only map states to real numbers. Given a space of states S, this would require a storage space of |S|, while an model of the world given a space of states S and a space of potential actions A, a complete model would require a storage space of size |S| x |S| x |A|, while a stochastic policy would be of a maximum size of size |S| x |A|. Some learning methods require a model of the environment, such as dynamic programming, while others do not, such as Q-learning [5]. Methods not requiring a model of the world are called model-free learning methods. Model-free methods are generally able to find optimal behavior, but model-based methods find it faster [5].

2.3 An insight in RL

Kaebling et al, [9], gives some examples of the basic algorithms found in RL, which I will briefly present here.

RL is about learning a policy, so that you can guide your actions achieving a higher accumulated sum of rewards. One way of finding such an optimal policy is to find the optimal value function. Its algorithm is illustrated below in Figure 4. It finds the optimal policy derived from the value iteration algorithm. The algorithm loops through all the states and all the possible actions that can be taken from those states, and this is done again and again, until one has settled on a value function V one is satisfied with. The detailed description of this algorithm is found in [9].

```
initialize V(s) arbitrarily
loop until policy good enough
    loop for s ∈ S
        loop for a ∈ A
            Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')
        V(s) := max_a Q(s,a)
    end loop
end loop
```

Figure 4: Value iteration algorithm

It is not obvious when to stop the value iteration algorithm, so proper stopping criterions need to be decided based upon when the policy is measured as good enough. Figure taken from [9].

An different approach is the policy iteration algorithm, where one manipulates the policy directly, without instead of finding the optimal value function, as illustrated in Figure 5. An explanation of this algorithm is found is [9]. This algorithm can illustrate the problem of scaling. It is shown that this algorithm terminates in at most an exponential number of iterations [9]. The worst case scenario is interesting, and "*it is known that the running time is pseudopolynomial and that for any fixed discount factor, there is a polynomial bound in the total size of the MDP*". (Markow Decision Process).

choose an arbitrary policy π'
loop
    $\pi := \pi'$
    compute the value function of policy π:
        solve the linear equations
            $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$
    improve the policy at each state:
        $\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s')\right)$
until $\pi = \pi'$

Figure 5: Policy iteration algorithm

One can with this algorithm ensure that any changes will strictly improve the policy. And hence, when there are no more changes, the optimal policy has been found. Figure taken from [9].

The theory of RL generally assumes the Markow property [5]. It means that a memory of recent actions and states are not necessary for the agent. The present state determines what possible choices one has. Markow Decision Process (MDP) is characterized by the agent being able to distinguish all different states. If that is not the case, one has a partially observable MDP (POMDP). All though the theory assumes this, this is very rarely the case in real-world problems, but one generally still acts as though the problems as MDPs.

2.4 Challenges

In any problem solving tasks, certain areas are more critical than others in order to succeed in solving the task. I will here take a look at some challenges the programmers and designers must deal with in standard RL systems.

2.4.1 The right RL function

It might be hard to come up with the right RL function that will induce good learning. Because of this, [6] it is argued why evolutionary computation is often better suited to solve problems than RL. Instead of being pessimistic and giving up on the RL strategy because it seems hard and difficult to implement, due to the close resemblance between RL and an understanding how learning takes in humans and in verbrates in general it is of high interest to pursuit the angle of incorporating RL into autonomous agents. This idea is also greatly supported by theories and research within neuroscience, and then specifically with regards to the basal ganglia [16]. So instead of abandoning RL because of its challenges, one should be optimistic and try to solve them instead.

2.4.2 Learning to learn

This topic is being addressed among others by the actor-critic model, and might serve as an answer to the challenge of creating the right RL-function. The actor critic model allows the agent to be able to view itself and its own cleverness, or lack of such, in learning. As stated in [7], actor critic architecture includes learning to improve evaluative feedback. An agent does not only learn things, but also increases its learning abilities.

2.4.3 Action today, reward tomorrow

In RL the agent must discover what actions yield the most rewards by trying them. Actions taken might not only affect the immediate reward, but also the next one, and through that, all subsequent rewards [20]. Having performed one action might affect all possible future rewards. So when a reward signal is received by the agent, it is not necessary clear which action or actions that lead up to this reward. This is one aspect found RL, unlike supervised learning, called the aspect of delayed reinforcement rewards. When you get the reward, the actions leading up to it is most likely in the past - but where, and which actions? Given the challenge of rewarding the correct actions with the experienced rewards, this might take some trial and error to classify what really leads to what. So learning should be an activity when one has energy, time and life enough to deal with some losses during learning. Even we, as complex and sophisticated agents as humans, might fail to learn the real lesson the first time. It might actually take some time before one, if ever, learns the true connection between the outcome and ones actions. If mapping rewards to actions is hard enough for us humans, how can we design agents to do that successfully and satisfactory? I think the answer lies in giving them time. Time to try out actions, explore, make assumptions and hypothesis about the mapping between actions and payoffs, and then test the guesswork they have done. After some testing one makes the decision and concludes that the hypothesis can be looked upon as a fact.

This problem, also called “temporal credit assignment problem”, has been dealt with a various ways. Delayed reinforcement learning is one approach, and can be divided into two categories, model-based methods and model-free methods [8]. The actor critic architecture and Q-learning are of the model-free methods.

2.4.4 When to learn, when to act

RL is different from most types of learning in that is has to deal with the dilemma of exploration versus exploitation. In supervised learning the agent is provided with the correct answer while in RL the agent uses training information that evaluates the actions [5]. The agent can evaluate how good or bad the action was, but has no idea whether it is the worst or best action possible. It needs to explore, try out different actions, and then exploit, use its gained knowledge. Kealbling et al. give a survey of reinforcement learning in [9]. They point out some general characteristics and challenges typical of RL, such as the exploration/ exploitation dilemma. Using a well-known example from math and statistics, finding an optimal strategy for playing the one-armed bandit in a given setting, they illustrate that the balance between exploration and exploitation depends on how long the agent is to play the game. The longer the agent plays the game, the worse the consequences of prematurely converging to a suboptimal strategy will be. The more time the agent has, the more it should explore. What does this imply for a real situated and autonomous agent? Hopefully the agent might stand a chance to live forever, at least relatively speaking. Or until its task has been done, be it lasting one day or 10 years to achieve it. So generally one does not know what timeframe one is working under. We don’t know at the time of implementation how many times an agent needs to perform actions with the purpose of exploration before it can begin to exploit its gained knowledge. A challenge RL have to deal with, is the adjusting of the balance between the exploration and exploitation.

2.4.5 New and changing environment

Once an agent has reached a high level of knowledge, as in having learned how to act in a given environment, how can one motivate that agent to explore new and novel situations? In of set of theories within the theoretical reinforcement learning literature, novelty acts as a surrogate reward, and some evidence show that animals to treat novelty as rewarding [10]. In a machine learning context, this would allow the agent to plan to visit novel states a number of times in order to explore the new or changed environment. This can be achieved by adding a novelty factor the input signals, as if there is a reward, and have that factor reduced as the input signals are reoccurring.

Then the agent would explore new situations, and over time learn whether or not there is any reward attached to the new situation. As the novelty factor is reduced over time, the likelihood of choosing actions leading to that situation becomes more depended on the experienced rewards.

2.4.6 Scaling up to large problems

In reinforcement learning, the environment is typically modelled as a controllable Markov process, so the agent must solve a Markov decision problem. There are typically four components to consider in a Markov decision problem, a finite set of states, a finite set of actions, a reward function that predicts the reward for performing a certain action being in a certain state, and finally the action model, which gives a probability of getting to a certain next state being in one state and performing a certain action. However, when problems get on a large scale, the set of states might not be discrete and definite. If an agent exists in the real world, one can not at design time predict all the possible states one agent might find oneself in, nor predict all possible actions, and certainly not predetermine the effect an action will have upon the environment. In a real world setting, an agent can not access all the possible information, the input signals, if all were to be covered is simply endless.

Even if the states are definite, and hence the world might be deterministic, the share volume might hinder the agent of practical use of RL systems, since going through the standard RL systems is too time consuming. In [22] we get a good illustration of how demanding it gets when the search space gets big. Imagine robots playing football. If the football field of 10x10 large, with 2 players of each team, there are roughly 6.25×10^{10} possible states, assuming a tile, a square, can only be upheld by one robot at a time. In addition, one also has to consider the possible actions the robots can take. The task of the mapping of states with actions becomes very quickly too slow.

Q-learning is a very popular and seems (dated 97) to be the most effective model-free algorithm for learning from delayed reinforcement. It does learn, no matter how the agent behaves during the learning period, as long as all state-action pairs are tried often enough. The convergence might be slow to obtain a good policy. This becomes a problem when dealing with a huge search space [9].

Some various approaches of how to deal with the curse of dimensionality comes next.

2.5 Various attempts to deal with the challenges

Standard reinforcement learning systems typically gets bogged down when dealing with a fain grained and huge search space. I will in the following chapters take a brief look at two of

approaches to overcome this obstacle that are interesting as they have similarities to the approach of this thesis.

2.5.1 Hierarchical reinforcement learning

Barto and Mahadevan find it natural to use hierarchical control architectures and learning algorithms [13]. Here one can allow the execution of temporally-extended activities which follow their own policies until termination. Learning can take place at different levels of the hierarchy. Figure 6 illustrates three different abstraction levels concerning a robot navigation task with respect to the memory state.

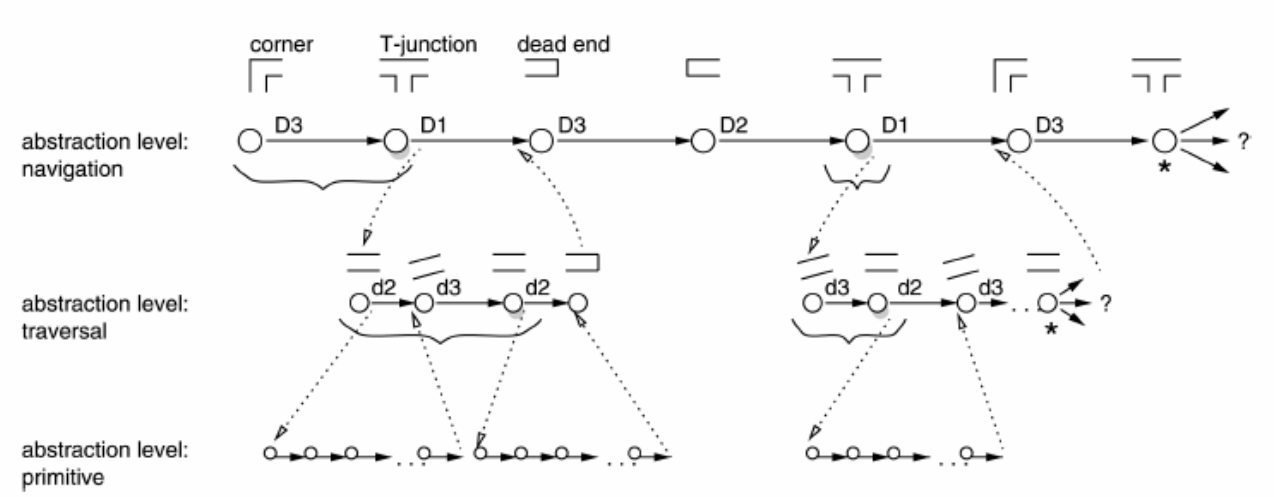


Figure 6: Hierarchical memory

At the navigation level, a decision is being made at every intersection. At the lower abstraction level, decisions are being made while the robot is in the corridor, between intersections. At the lowest level is the most primitive abstraction level, which might have to do with the output signals given to the executors (wheels or motor). Figure taken from [13].

In hierarchical learning one can benefit from working with sub-goals. The hierarchical structure is often pre-wired by the designer, as one can assume with the illustrated hierarchical memory illustrated in Figure 6. This task can be solved online, too. In [11] Bakker et al have developed methods and algorithms where high-level policies discover sub-goals, whereas low-level policies learn to specialize for different sub-goals.

This thesis also investigate at what level the abstraction should be, meaning at what level should the RL system take affect. It is similar to the hierarchical reinforcement learning in that it clearly divides levels of abstraction; only in my approach the RL system only takes place on the highest abstraction level. The implementations and architecture of systems on the lower abstraction levels is not concerned with RL systems, and the RL system does not concern itself with how the lower levels of abstraction work.

2.5.2 Value function approximation

When the number of states gets high, paring each state with a value gets intractable. The curse of the dimensionality, or the intractability of state spaces, suggests a value function approximation [12]. Value function approximation (VFA) is a well-studied problem.

Mahadevan et al tries two novel approaches in [14]. Much of the existing VFA is handcoded in an ad hoc trail-and-error process by a human designer. In contrast to this main-stream of attacking the

problem, they try to derive VFA from the geometry of the underlying state space. Their mathematical approach uses eigenfunctions of the Laplacian, Fourier analysis and diffusion wavelets.

Sutton et al [15] points out some theoretical drawbacks of value function estimation. One drawback is that the policies that would produce superior performance might be ignored since most implementations lead to deterministic policies in spite of the optimal policy being stochastic.

My strategy of overcoming the curse of dimensionality is by reducing the search space, and to design an architecture that allows the RL system to work with much fewer parameters. In stead of pairing each state or the world with a value, the RL systems has a narrow view of the world by only considering its internal generated states of the different experts agreements, and based on this, it conducts its learning. Instead of relying on raw input signals from the world, it simplifies the world by only seeing the filtered input done by the lower level experts, and has no knowledge of what is better or worse, other than to compare the different outcomes of the experts up against each other.

2.5.3 Prioritized Sweeping

Moore et al introduced a memory-based technique in [21]. Their approach is that while dynamic programming works faster in huge search spaces, classical methods are more accurate. Their algorithm tries to do the same task as the classical method Gauss-Seidel iteration, but accompanied with the usage of memory to concentrate all computation effort of the most "interesting" parts of the system. The numbers of states they operate with are more then 10000, and would initially seem to require a lot of memory allocation, but their strategy is to only allocate memory for the experiences the system actually has.

They test their algorithm on different problems, and compare the results up against other techniques, and find that their system in many cases finds the solution faster than other systems, and in some cases they are able to find an answer where the other systems they compared it up against could not. In their paper, they compare their system against others and present the results, as Figure 7 is an example of.

	Experiences to converge	Real time to converge
Q	never	
Dyna-PI+	never	
Optimistic Dyna	55,000	1500 secs
Prioritized Sweeping	14,000	330 secs

Figure 7: Prioritized Sweeping compared

This figure is taken from [21]. It illustrates the success of Prioritizes Sweeping in a given problem (rod-in-maze task) compared to other techniques.

This approach is similar to mine, in that it reduces the search space by the means of guided search. All though they don't present any mathematical proves of convergence, their tests shows that in certain settings their algorithm is able to solve a given task.

3 Architectures dealing with action selection

Since the information accessed is being used to choose what action to perform, it is natural to take a look at different architectures dealing with action selection. A short presentation of the 3 main categories of action selection architectures, namely distributed, centralized and subsumption architecture is given in 3.1, 3.2 and 3.3.

The DAMN architecture has many similarities to my approach. Because of this, I will take a closer look at this one specific architecture in 3.4.

As stated in the problem description, I will try the group-of-experts approach to finding the proper abstraction level. A presentation of the Mixture of Experts regime is presented in 3.5.

3.1 Distributed architecture

In this architecture each module or behavior receives the same input. They are all linked to each other with inhibitory links, and they all have an excitatory link to the shared output resource. The weights of these links varies, and the chosen competitor emerges from the network. Such a network, using recurrent reciprocal inhibition can support winner-take-all functionality, a model often used in action. This architecture comes at a high cost regarding both the density of connections between rivals and the cost of integrating a new competitor in an existing network.

As we see in Figure 8, the distributed architecture has no central control of which competitor is to be chosen over another. Instead, the selection is often described an emergent property of the network. This property has been found in investigations of verbrate neural circuitry, where relatively small differences in the input leads to a change of selected behavioral output [1] .

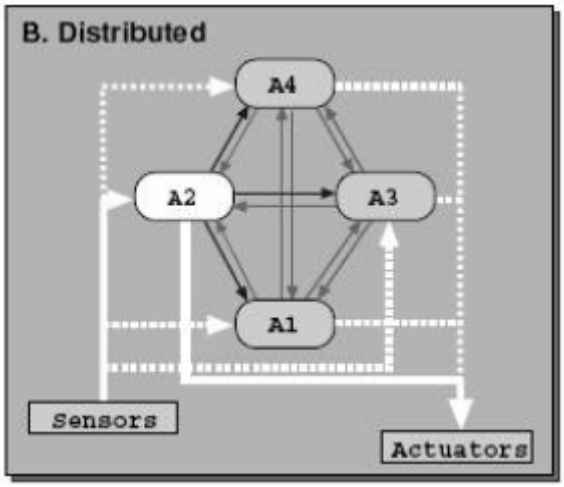


Figure 8: Action Selection Architecture – distributed

Solid arrows represents support for A2 which in turn imposes greater reciprocal inhibition on competing elements. Figure taken from [1].

This architecture is by definition distributed. All the elements are connected to each other, they all receive the same input and they all have access to the actuators. They influence each other by inhibition each other through their connecting links. This is a costly architecture with respect to the number of links. One positive effect of having the control distributed is that it handles errors efficiently. If one element goes down, the others are still functional, allowing the agent to function.

Adding more elements would require some work, though, given that all elements have links to all the others.

Whether or not to include learning is an open question for the designer. Each element could be a subject to learning, as they could learn from experience what output to give to the actuators. The weighted links between the elements could also be a target for learning, as the agent would experience the pairs of censor input and actuators output, and the effect that has on the environment. If it has a good effect, the weights could be further strengthened, increasing the likelihood of performing the same action in a similar setting again. A problem with this, is that changing the weights due to one situation, might cause the agent to perform worse in a different situation, as the changes in the weights might cause the agent to choose more poorly in other situations than the one causing the changes in the weights.

This architecture could embrace all different levels of awareness. That would depend on the environment the agent exists in, and its censor inputs. If this is used in a grid world where all the information is accessible, it would have a high degree of awareness. If this is an agent situated in the real world, it would be impossible to have access to all the information. The architecture itself puts no restriction, though, on the degree of awareness.

Since the output is being decided by the weighted links between the elements, one would assume this architecture to perform well even under time constraints. The time element would depend on each elements required time to produce an output signal.

This architecture could be totally self-contained. With all decisions being made internally, this architecture opens for a high degree of autonomousy.

3.2 Centralized architecture

Centralized architecture, as illustrated in Figure 9 is less costly that the distributed architecture [1], since a competitor only needs two connections, to and from the central selecting mechanism, while in distributed one needs two connections to every competitor. This is a desired advantage in both artificial and biological control. A second argument favoring this class of architecture is the modularity. A change directed at one aspect of the behavior or the agent could impact the switching behavior of the network with possibly negative and undesirable consequences. When using an artificial neural network, one finds that even small changes to the weights can result in great change in the output. It is desirable to adjust the network to produce the right output. By partitioning this network, one could train one part of it dealing with one sub-problem, while other parts of it deals with other problems. When an update is being performed, it should only affect the region of interest, and not interfere with the other regions causing a change in their weights. In a centralized architecture one allows for parts of the network being used for certain aspects of behavior and changes within this local cluster will have much smaller effect on other aspects of the agents behavior.

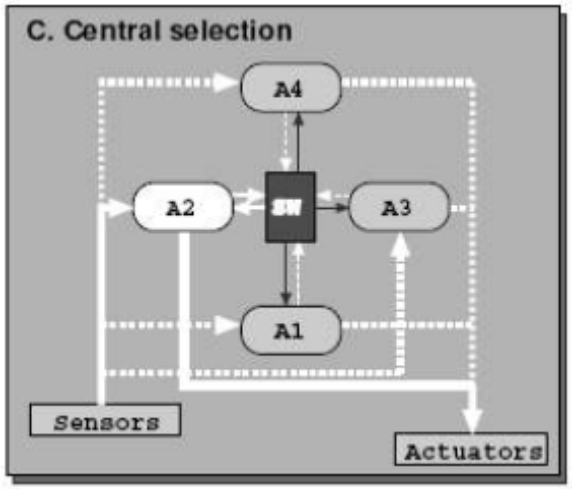


Figure 9: Action Selection Architecture – centralized

All elements receive the inputs, and they all have a linked connection to the shared output resource. The Central Switch (SW) supports the A2 element (white arrow) and inhibits the others (black arrows). Thereby the A2 element is chosen as the winner. Figure taken from [1].

Being a centralized architecture it is less costly than the distributed architecture, as all the elements does not have links to all the others, only to the central switch. The drawback is the systems vulnerability, as it would not function if the central switch failed. It could easily function even if other elements went down. And it would require less work to add new elements, as this need not affect any existing elements other than the central switch.

One could in this architecture more easily integrate learning, as this could be done onto the central switch only in order to increase the performance.

This architecture has the same restrictions with respect to awareness as does the distributed one. It opens for both low and high degrees of awareness.

There are fewer links here than in the distributed architecture, and it should therefore perform even faster. It would depend greatly on the different elements time requirements to produce an output, and especially the central switch.

It has the same potential for autonomousy as the distributed architecture.

3.3 Subsumption architecture

To subsume means to contain, to include. In this architecture the action selection is being decided hierarchically. The various behaviors to choose from are hierarchal ordered, as illustrated in **Figure 10**. The bottom ones are the most basic ones, and are implemented first. Whenever more than one behavior is competing for gaining the control of the output, it is always the highest level that wins. A fundamental principle within Brooks Subsumption Architecture [2] is the hierarchy. If the programmer decides that it is more important for the robot to avoid obstacles than to pick up garbage, the behavior PickUpGarbage will be inhibited by the behavior AvoidObstacle when needed. That is how the architecture of subsumption deals with action selection. When a more important behavior, defined by the hierarchy, competes with a less important one, it always wins.

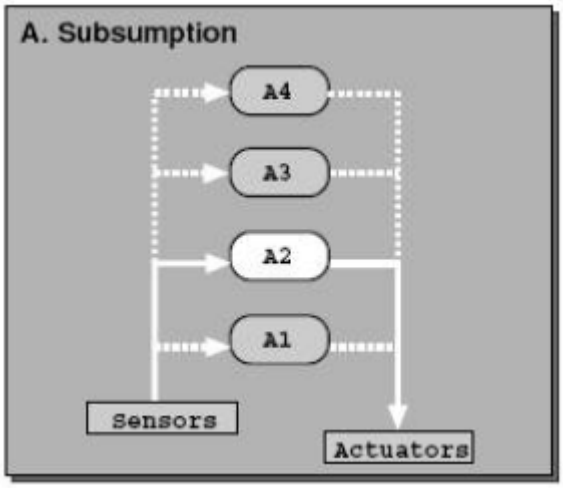


Figure 10: Action Selection Architecture - subsumption

The subsumption architecture. If any of the layers are in competition, the highest layer always wins. The agent is fully working at any level, so removing layers from the top still leaves a fully functional agent, only less complex. Figure taken from [1].

The learning aspect seems to be in the hands of the programmer when it comes to this architecture. One can observe ones agent, and add behaviors as to increase the performance of the agent. How the agent is supposed to learn itself, is not so easy to suggest as with centralized and distributed architecture. One way of introducing learning could be to add a behavior that has learning capabilities. Instead of having the programmer decide all behaviors at design time, the behavior could be guided during runtime by for instance reinforcement learning.

The degree of awareness can vary as the agent is further developed. One basic behavior could be to avoid obstacles. When adding a behavior that seeks out other agents, too, one might need to increase the input signals to also cover signals being send from other agents. To what degree the agent is aware of its surroundings is depended on what the agent is designed for. As with distributed and centralized architecture, there is generally no restriction in the level of awareness. A basic thought Brooks [2] presented, though, is that one need not explicitly represent the environment with symbols, typically found in classical AI.

3.4 The DAMN architecture

This is an architecture used to control a mobile device. The Distributed Architecture for Mobile Navigation (DAMN) is described in [3]. This architecture works by different modules, or behaviors, sending their votes to an arbiter, which then decides the output signals to send to the motors. The architecture is illustrated in **Figure 11**. **Figure 11: The DAMN architecture**

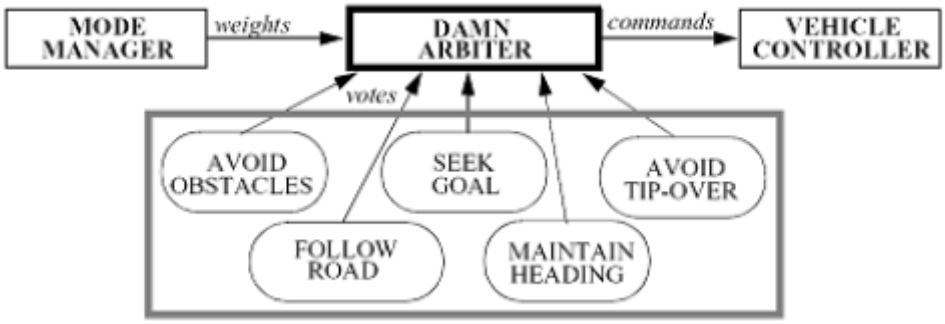


Figure 11: The DAMN architecture

This figure, taken from [3] shows the overall structure of the DAMN architecture. We see the distributed behaviors sending their votes to the arbiter, who decides the output.

The mode manager assigns weights to the votes received by the different modules, or experts. All the modules send their votes to the arbiter, and have no control over which expert gets to decide. That is determined by the weights of the mode manager.

3.4.1 Distributed and centralized

DAMN is combining a distributed with a centralized architecture, having different modules, or behaviors, being distributed, and letting a central arbiter deciding the output [3]. As we see in **Figure 11** each behavior sends their vote to the arbiter, which then decides what output to send to the vehicle control. One can add new behaviors without any need for modification of the already existing behaviors. This is similar to the subsumption architecture, only here one does not need to do it in a hierarchal manner. The arbiter does not care how the different modules reach their decisions; it only evaluates the current situation via the censor inputs and the incoming votes.

3.4.2 The learning aspect

The mode manager can influence and change the weights of the incoming votes from the behaviors, and thus change dynamically the final outcome and behavior of the agent. This is a natural place to add learning, letting the mode manager change the weights if needed according the changes in the environment. A different form for learning could be incorporated in the arbiter. The arbiter is the one who decides what signal is being send out. It could learn how to do this in the most beneficial way. Collecting all the votes, and implement a winner takes all algorithm is one way of doing it. One could also let the arbiter performs a command fusion, letting more than one behavior have say at what the output should be at any given time. Then it could be an area for learning to developed the best way of doing the fusion. Having the behavior with the most votes count 100 percent is one extreme. Having them all count according to their votes is another way. Maybe one should perform a fusion between the 3 most highest votes, or let the highest vote count for 50 percent, the next highest for 40 percent, and the rest share the remaining 10 percent. Once this is set for a specific environment, the arbiter might need to learn how to do things differently in a different environment, as with the example of icy roads versus dry asphalt with the mode manager.

This architecture opens for letting the various modules include learning, also. Nothing hinders the various modules to learn as the agent exists and performs actions, and they could improve their behavior according to their internal mechanism trying to achieve better results according to their own goals. One challenge there would then be that the arbiter might in a given situation learn that one module should not be given any control. However, as time goes on, this module might learn to act differently and should be allowed to control the agent in the same situation in the future. The arbiter would then need some sort of re-evaluation of its weights, allowing the different modules to be reconsidered as time has gone by and they might have learned to act differently.

3.4.3 The degree of awareness

This architecture is being used in situated, real mobile units, or agents. Dealing with the real world, it is impossible to have access to all relevant information considering the endless space of different states the system can be in. How much information, and what kind of information, each behavior

and the arbiter is to receive can be adjusted individually, according to each behaviors need or desired outcome. A purely reactive behavior designed to not hit obstacles have a less interest of inner values and set of beliefs as would a behavior that conducts longer term planning of actions. Being situated in the real world, there is also the aspect of noise. The censor inputs does not have all the information available, and the information it has might not be totally accurate.

3.4.4 The aspect of time

This architecture is being used in a real mobile agent, and thus faces the challenges of dealing with real time issues. If the mobile unit is about to drive of the road having a certain speed, one only has a limited time available to choose actions that will prevent the likely for future accident from happening. The frequency of how often each behavior sends its vote are adapted to each behavior. A purely reactive behavior, designed to react to a situation as the one described above, might need to be allowed to issue its vote most frequently. While a behavior that is involved in longer term planning does not need to express its vote that often, just every now and then to guide the mobile according to its long term plan.

3.4.5 The degree of autonomy

This architecture is open for any degree of autonomy (except the philosophical level that would require being omniscient). One could use the architecture in a mobile device with a very little degree of autonomy. The ABS-breaks on most new cars might serve as an example, where the human driver makes almost all the decisions, and the ABS kicks in and helps when needed. A fully automated mobile unit might perform all the choices itself regarding turning, accelerating, doing specific tasks and so on. The level of abstraction could range over the whole specter, from low level deciding whether or not to accelerate and turn in any given moment to the higher level of planning the route on how to get from the current position to a goal target.

3.5 Mixture of Experts

One could look at the agent as being not just one unit, but being made up by a number of different experts. Each expert would then be good in its specific domain, and by letting the agent being controlled by all these experts in a workable fashion, the agent as a whole would be able to deal with a variety of challenges. One approach to solving problems is the divide-and-conquer strategy. To solve a problem you can divide it into sub problems, and then sub problems into sub-subproblems, and so on and so forth until the problems are easily solvable. This general algorithm is well known and used. In [4] we find a model using this strategy. Assuming that the input space, or problems, can be clustered into similar groups, it would be smart to have certain groups of the neural network represent certain groups of the input space. Not knowing a priori the boundaries between the different groups, Jacob et al [4] suggests an architecture that tries to solve this. They talk about experts, meaning one expert is focused on one cluster or group of input values. When training the network, a gating expert chooses which expert to use for the training case at hand, and only that experts weights are updated. How this gating expert is implemented can vary, and one can also include learning to the gating expert, so that he learns to choose the most appropriate expert. This architecture is shown in **Figure 12**.**Figure 12: Group of expert's architecture**

.

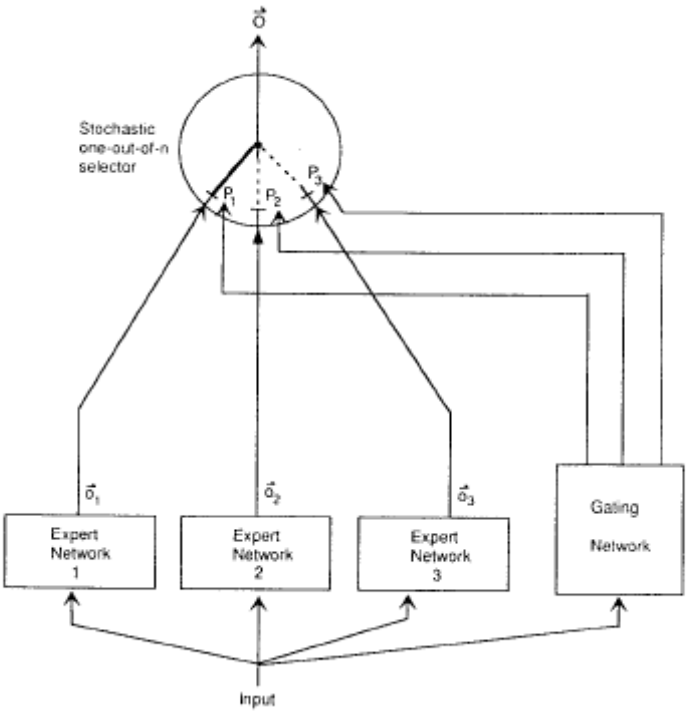


Figure 12: Group of expert's architecture

This figure is taken from [4], and illustrates the data flow in the architecture. Each expert is a feed-forward network, as well as the gating expert, and they all receive the same input signal. The weights from the gating expert, its output, is a main component for the selector when it decides which expert is given the control.

This approach of having different experts, or agents, suggesting different actions, and then through a mechanism, (e.g. a gating expert), decide who gets their action executed, is closely related to some hypothesis about how the basal ganglia works. Among other tasks and believed purposes, the basal ganglia is believed to be an area where action selection within the brain is taking place. The role of basal ganglia as an action selector is a recurrent idea in the basal ganglia literature [1].

4 The architecture

In this chapter I will describe the architecture I have designed. I will first in 4.1 describe the world the agents live in, and the restrictions I put on the world and the agents. I will then take a look at the architecture of the agent in 4.3, and specifically point out the three important features of the agent, namely the filtering of input signals by the group-of-experts, the action selection and the learning system.

4.1 The world

The scenario is as follows: the agent exits in a 3D world along with other agents. To distinguish these from the main agent, I will refer to the other agents as targets. Each target has its characteristics, such as colour and shape. Existing in a 3D world, they also have properties such as location and speed in the virtual world. Certain combinations of characteristics of the targets define the agent as inedible, being poisonous. The agent is to learn which targets are edible and which are not. The agent hunts down the targets, one at a time. Upon impact the target cease to exist, as in being eaten by the agent. If the target proves to be poisonous, a negative feedback signal is generated. If not, a positive signal is generated. This reinforcement signal is then used by the agent to learn to differentiate between the different categories of the other agents, the targets. As seen in **Figure 13** we see the agent, being a white cone, existing in a crowd of other agents. The white agent tracks down targets, and chases them until they collide. At that point, the target is eaten, and the white agent sets out for a new target. All agents move around, the targets somewhat randomly.

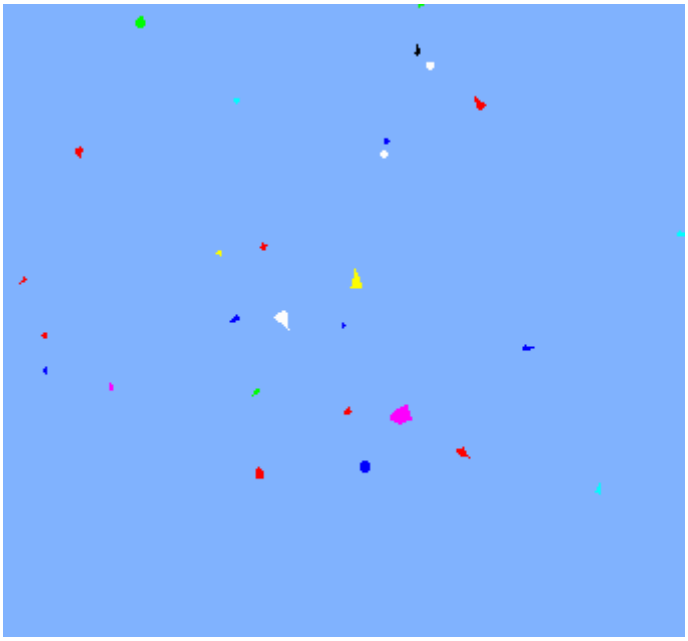


Figure 13: The scenario

In the middle of this picture we can see the agent, a white cone, existing among other agents, the targets. The targets have various shapes and colors. Certain combinations of characteristics are poisonous, and should not be eaten by the agent once it has learned whose are edible and not.

The world is of a limited size, and the number of agents existing is determined online and randomly, but within the boundaries of a maximum and a minimum number of agents. The other agents live for a period of time, either till they are eaten, or until they just expire after having

existed for a set time. This is done so that the agent, when it learns who is edible, always should have both poisonous and edible targets to choose from. The targets move somewhat randomly around. If they get to the border of the existing world, they turn around 180 degrees. If the agent comes to the end of the world, it is simply relocated to the centre of the world. This would not be possible in a real world, but the main idea is to explore how the agent can take advantage of lower level experts combined with reinforcement learning, and the resemblance to a real world setting is of less interest. BREVE opens for simulating a real world, taking into consideration gravity, mass weight and so on, but advice against using those features if it is not absolute necessary for the simulation, as it makes things much more complicated and slows down the simulation considerably.

The reinforcement signal is to be outside of the agent's control, a general principle in reinforcement learning. This can be done explicitly by having the agents inner energy go up or down as it its other targets, depending on if they are poisonous or not. For simplicity, I define both which agents are poisonous and generate the reinforcement signal within the agent. This does not collide with the definition of reinforcement learning, since it is the definition of the target that triggers either a positive or a negative feedback signal, and hence, it is outside of the agent's control. This could have been programmed differently, but the area of the agent that uses the reinforcement signal to induce learning has no control of this signal, it only receives it.

The complexity of the targets can easily be changed before the program is run, as well as the number of agents that is to exist at the same time. This makes it easy to experiment with lower and higher degrees of complexity which determine the size of the initial search space.

The agent lives in a 3D world. Its main goal is to move around and catch other "edible" mobile agents. Some agents are poisonous and not edible, and should be avoided. The agent sees the whole world, which consists of itself and the other mobiles. Each mobile has its own characteristics defined by shape and colour. They are all mobile, and move around in the virtual world somewhat randomly. Some parameters for the simulation worth paying attention to are as follows:

n: number of agents
s: number of possible shapes (cone, sphere, disk). One object can be of only one shape.
c: number of colours (red, green, blue), either being on or off (2^3=8 possibilities)
v: velocity, speed in 3D world defined by a vector of 3 dimensions (x,y,z) made up of three float values
l: location, defined by a vector of 3 dimensions (x,y,z) made up of three float values

To illustrate how infeasible it would be to represent every possible state of the world in a lookup table: each target has the permanent characteristics of shape and colour, adding up to $3x2^3=24$ variations. Imagine the agent existing on a grid world made up of 9 tiles, with itself being in the middle. That leaves 8 tiles open for possible occupations of targets. Only one object can occupy one tile at a time. If these 8 tiles are to be occupied by 8 targets, the targets can position themselves in 40320 different ways. The agent in the middle deals with a possible variations of states of the world with the size of 40320 – that is assuming that the agent represents each target uniquely, and there are only 8 different targets total. However, the agent can only observe the targets, and are left to define them by their characteristics. Since each has one out of 24 different possible variations of characteristics, the total number of states of the world can then be represented as 24^8= 110'075'314'176. The agent, however, does not exist on a grid world of size 9, it exists in a virtual continuous world. In a world defined by tiles, allowing the total number of locations of the world to be a known integer, it very quickly becomes of such a size that representing the total number of possible different states in the world becomes infeasible. If the space location of the world is

continuous, it quite simply becomes impossible to fully represent every state of the world in a table.

Defining which agents are edible are done be defining which combinations of characteristics are lethal. For instance can all red agents be poisonous, or alternatively, all blue agents with the shape of a cone are lethal. Defining which agents are poisonous takes place in the method that is called upon impact, and can be modified during runtime by changing parameters using the menus.

The agent needs to learn which objects are desirable, and which are to be avoided. This is achieved by using reinforcement learning. The experts do not give a specific advice on what action to take, but rather filter the input signals and pass on to the agent the input signals that they choose. Initially, this is done by having experts seeing only certain types of mobiles depending on their characteristic. The input signal is, thanks to breve, easy accessible by easily getting a list of all mobiles present in the environment. One expert can then pass on only red agents, while another expert only passes on agents with the shape of a cone. This can further be diverged, so that one limits the view of an expert to be of a certain distance and/or angle, while other input signals could be of importance, too, such as velocity, angle of moving direction compared to the agents moving direction and so forth.

As mentioned before, how the experts filter the input signals are of no interest to the agent. It is how the agent can learn to use the much smaller space of input signals that is of importance. From the programmers perspective, finding the proper abstraction level is the main goal, so of course, experimenting with how the various experts filter the input signals are of great interest.

Various reinforcement learning methods

I choose to use look-up tables. Normally, in a huge search space, a look-up table is not feasible. But the very effect of greatly reducing the input space allows me to use look-up table here. I do not need to have a lookup table for each possible state, but rather a lookup-table that connects chosen behaviour with which expert was given the control, or degree of control, in a given situation.

The input signals come reduced in magnitude to the agent. The agents, based on these signals, work out some various options of what action to perform, and converts the input signals into a smaller number of varieties, so action can be paired with state, and used in a lookup table for learning. This is one possible approach examined in this thesis.

4.2 The targets

The other agents are all potential targets. They all have characteristics such as shape, colour and location. They move around randomly, and exist in a time span set at runtime. Each agent is randomly given a time span set between a lower and higher boundary. This means that the agent might chase a target, and the target disappears before it has been hunted down. At all times, the number of other agents is between a lower and higher boundary limit, ensuring possible targets for the agent to find and chase down.

4.3 The agent

The agent is made up of a virtual agent set up in the BREVE simulator. It is very basic; it has the shape of a cone, a size and colour. And it moves around. The BREVE simulator gives easy access to all agents within a limited radius of the agent, which is set before runtime. By setting this limit to be wider that the size of the world, the agent has access to all the agents all the time. These other

agents, with their distinct location, velocity, shape and colour, represent the input space in which the agent is to do its search. As explained in the setting above, certain characteristics are defined to be poisonous and undesirable. This is the learning goal for the agent, to determine which are edible, and eventually avoid the poisonous ones while only chasing and eating the edible ones.

The agent uses experts (processes within the agent) to filter the input space, and based upon this filtered input signals it will do its learning and base its actions. The overall architecture is shown in **Figure 14**. How many experts, and how they process and select the input signal to be forwarded to the agent, is entirely up to the programmer before runtime. The number of experts can easily be changed, and experts can be added or subtracted from the agent very easily.

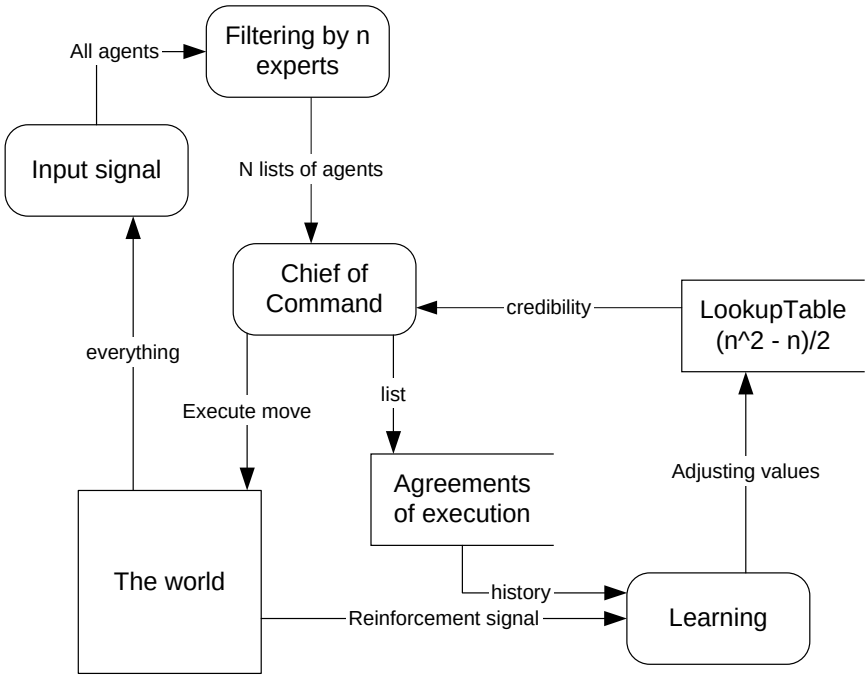


Figure 14: The architecture of the agent

The box "world" contains everything that exists in the world, including the agent itself. The agent receives input signals from the world. In my experiment, the input signals are objects, or agents, existing in the world, containing information about each agent, such as color, velocity, shape etc. These input signals are filtered by experts, and only chosen objects are passed on the Chief of Command. Here the various agreements between the different experts are evaluated by the means of the lookup table, and a target is chosen to chase down. This results in executing a move, which affects the world. The experts, chosen to decide which target to chase down based on their agreements, are stored in a list. When the agent receives a reinforcement signal from the world, it uses this to update the values stored in the lookup table.

4.3.1 The Chief of Command

The Chief of Command is the core of the agent with respect to the reinforcement learning system and action selection. This module need not have any knowledge of how the different experts produce their output, how they filter the original input signals. The Chief of Command will through experience learn which combinations of agreements between the agents are desirable and which are not. This knowledge will then be used to avoid the poisonous agents. The more detailed architecture of the Chief of Command is illustrated in Figure 15

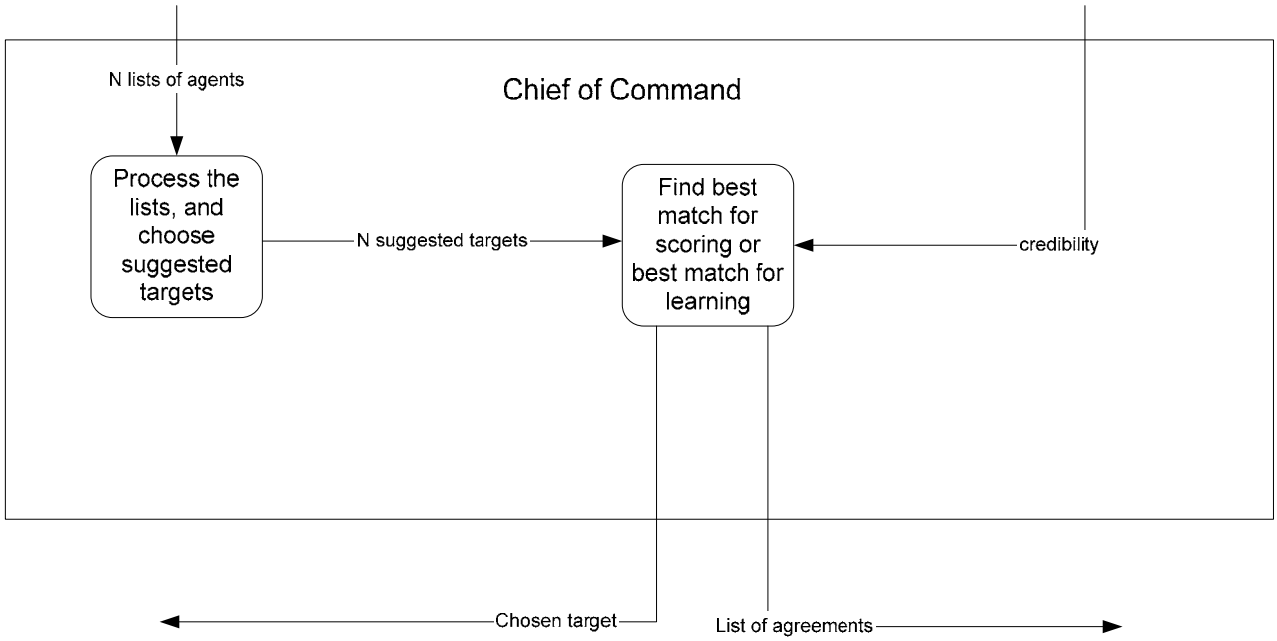


Figure 15: The Chief of Command

The chief of Command is the heart of the agent with respect to how it chooses to behave. It receives a list of agents from the experts, and processes these lists to suggest which agent to chase down, one for each expert.

In my experiment, deciding the target is done simply by finding the closest agent within each list. All agents exist in the simulated world in the BREVE simulator, and have a physical location represented as a vector made up of three float values, x, y and z. It is a trivial task in BREVE to calculate the distance between any two objects. For each list of agents filtered by the experts, the agent chooses the one agent, or target, out of those available in the lists, closest to itself, relatively speaking. The criteria on how to choose the targets can easily be changed.

The suggested targets are then compared towards each other, and any agreements between the different experts are further investigated by using the lookup table. The Chief of Command then decides what target to settle on. It can by choice either explore or exploit its knowledge. By exploring it chooses targets which predicted outcome (feedback) is not certain, and needs to be experienced. By exploiting, it chooses a target that is most likely to result in a positive feedback. When settling on a target, the location of the chosen target decides what action to perform. The list of agreeing experts resulting in choosing the target is stored. Upon receiving feedback, the stored list of agreeing experts is used to update the credibility values stored in the lookup table.

4.3.2 The experts – specially designed sunglasses

Each expert gets the whole input space as their input signals. To simplify matters in order to keep a focus on the task at hand, I have designed the experts to be very simple. One expert filters the input signals, and passes forward only those agents who contain the colour red. A different expert might only pass on all the agents with the shape of a sphere. One could have a near-sighted expert only passing on the agents that are closer than a certain set distance relative to the agent. One could have experts that only pass on agents that are moving in the same direction as itself. The possibilities are many. There are, however, at least three important aspects to consider. Firstly, their combined output should be of less magnitude than the original input. Secondly, the time spend to filter the input should benefit the agent on a whole so that the agent spends less time than it would if it were

to behave the same way without the aid of the experts filtered input signals. And last, but not least, the filtered inputs need to contain enough information for the solution, or desired behaviour and learning, to emerge.

Each expert generate their own list, being a filtered representation of the original input signals, so n agents produces n lists that is forwarded to the Chief of Command. The agent then is presented with filtered segments of the original input, on which it must base its actions, see Figure 16. The purpose of this architecture is to reduce the search space and the workload put upon the Chief of Command, while maintaining the possibility of acting and learning as desired.

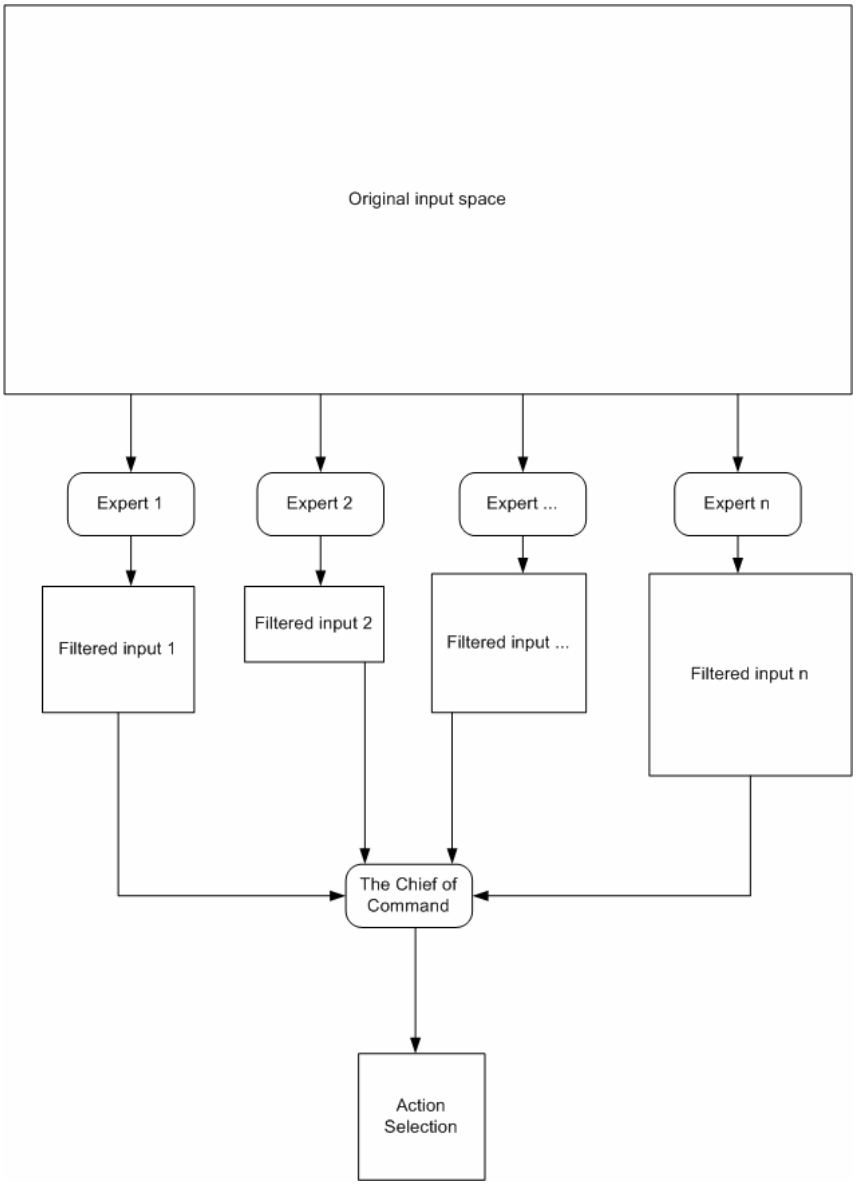


Figure 16: Experts filtering the input signals

Here we see an illustration of how the original input space is being reduced by the experts. Each expert filters the input signals, and passes on the input signals it chooses. The main agent, the Chief of Command (CC), is then presented with a fragmented input space filtered by the experts. Based on this available information, the chief of command starts the process that ends in an action selection.

The experts used in this experiment have names reflecting the lists of agents that slips through the filter of the expert. The cone expert's filtered list contains only agents with the shape of a cone, and

the same goes for the other experts, namely the sphere, the disk, the red, the green and the blue expert.

4.3.3 The action selection

The module that deals with the action selection deserves a closer look, see Figure 15. Each expert produces a list containing some, but not all, of the total input signals. In my setting, these input signals are other agents. The total input signal contains all the agents, while the filtered lists contains only certain agents, chosen by the expert making the list. Being presented with these lists of selected agents, or potential targets, the agent will choose one target out of these filtered lists with targets.

To keep things simple in my experiment, the Chief of Command module first finds the closest agent, relative to itself, in all the lists presented to it. All targets are located somewhere in the simulated world, having x, y and z coordinates, as well as the main agent. Since the agent is to chase down and eat the targets, one way of choosing what target to chase is to chase down the one closest to it. The distance between all targets in the filtered lists and the main agent itself is calculated, and the one target being the closest is suggested as a new place to move closer to. This results in n number of suggested ways to move next.

If there are two or more suggested moves that point in the very same direction, they are categorized as being in an agreement. If the suggested move based on expert 1,2 and 4 all point to the same direction, there is as agreement of the tupples (1,1), (1,2), (1,4), (2,2), (2,4) and (4,4). Each tupple has a value stored in the lookup table that reflects its predicted reward signal, see Figure 17. A positive value suggests that the effect of executing that move will result in a positive reward. A threshold value sets a limit for when the value is believed to be trustworthy. If the value is above a the set threshold, the tupple is believed to be of good credibility, meaning the agent can rely on the value to predict the next future reward signal. If the value is below a the threshold, the expected outcome is more uncertain, and should be explored further to gain experience and to adjust the value further, until it is above the threshold. The agent has no awareness of the correct answer, and can only rely on its own experts and its variety of different choices it has.

Expert_1 expert_1	0.6753
Expert_1 expert_2	-0.8700
Expert_1 expert 3	1.0000
Expert_2 expert_2	-1.0000
Expert_2 expert_3	0.0240
Expert_3 expert_3	0.7090

Figure 17: The lookup table

Each agreement between any two experts are assigned a float value between -1 and 1. Each expert is of course in agreement with itself, and is also represented in the table. A high float value means this pair of agreeing experts have received more positive than negative rewards in the past. The agent needs to explore, and try out different combinations of agreeing experts in order to increase or decrease the float value. After some experimenting, the agent can then start to exploit its gained knowledge stored in the lookup table by choosing the pair of agreeing experts with the highest float values.

In the example in Figure 17, if expert_1 and expert_3 are in an agreement, meaning the next calculated move done by the Chief of Command based on their respective filtered lists are the same, the agent can expect a positive reward signal as a result of executing that move, or set of moves, until the target has been chased down. If the threshold is set to be 0.7, then the three rows containing expert_1 expert_1, expert_2 expert_3, and expert_3 expert_3, still requires more exploration before the agent can rely on the credibility of the predicted reward value.

4.3.3.1 Exploring versus exploiting

The lookup table is the arena for learning. This is where the agent can vary between exploring and exploiting its gained knowledge and experience.

If there are a total of three experts, named expert_1, expert_2 and expert_3, the lookup table has 6 rows holding 6 float values, as illustrated in Figure 17: The lookup tableFigure 17. This lookup table is being used by the Chief of Command, see Figure 15, in order to choose which experts to listen to when deciding the next target to chase down.

If any absolute values of the lookup table are lower than the set threshold, positive or negative, the agent chooses a target from the choices that still has values below the threshold. If, however, all values are higher than the threshold, it simply chooses among those. It can choose the one with the highest value, or choose randomly between them.

To insure exploring long into the run, one could add that choosing the best candidate is of a certain probability, but that it every now and then would choose a total random move, even if the predicted outcome is negative.

Based on the example in Figure 17, the Chief of Command will over time learn that whenever expert_1 and expert_2 are in an agreement, it should not move into the direction resulting from any of their filtered input signals. If expert_1 produces lists with agents with the shape of a cone, exclusively, and expert_2 only produces lists of red agents, this implies that any agent with the shape of a cone and having the color red is not edible.

4.3.4 The learning system

The task of the learning system is to adjust the values stored in the lookup table to reflect the experiences the agents gets with the real world, so that it can avoid the bad targets, and only eat the good targets. This is done by the learning system by simply increment the credibility value of all the tupples that was in the last agreement on executing the last move, by a certain increment multiplied by the reinforcement signal. The reinforcement signal is either -1 or 1, where a negative value indicates the situation being undesirable.

Every time the agent eats a fish and receives a positive reinforcement signal, all the tupples of agreeing moves, aka experts, gets their values increased.

To illustrate, lets say that any fish with the colour red and the shape cone is poisonous. If the agent eats such a fish, all the tupples of agreeing experts that agreed to execute this past move in punished by having their lookup table value reduced. However, it might only be one tupple of agreeing experts that will end up with a learned negative value, while the others just happened to be pointing to the same direction at that time. To compensate for this, the agent needs to have enough impacts with targets that induce a positive reinforcement signal, so that all the tupples that got punished “wrongly” get their credibility adjusted up again.

The learning rate is represented by the size of the increments of which the lookup table is being adjusted by. Very small increments would imply slow learning, and maybe too slow to be able to react in a reasonable timely manner. Big increments would imply fast learning, but might then be subject to premature converging. Various techniques can be used to adjust the learning rate. One way is to just set the increment value at design time, which is what I have used most in my

experiments. Another technique could be to vary the size of the increment depending on the original value in the lookup table, with the effect that a value that is above the threshold is not as greatly affected as a value below the threshold. This would be of importance if the world were not deterministic, so that a target that is edible at certain odd times will produce a negative reward signal.

4.4 A walkthrough of a sequence in the simulation

I will here go through a possible scenario to exemplify the code. This is not an exhausting explanation, but serves as an aid in the understanding of the code and the architecture described above. I will then show examples of results, and how they can be displayed to describe the learning effect taking place.

4.4.1 The continuous loop running inside the agent

The agent goes through certain steps at each iteration, as illustrated in the list below.

1. Go through the input signals, produce filtered lists. This is done by the (internal) experts. The resulting lists are stored in the list **`filteredInput`**.
2. Calculate the next move in order to move closer to the closest target within each list, and save those suggested moves in the list **`lastVelocity`**.
3. Make a list of all unique suggested moves, store in **`indexMoves`**.
4. Make a list of all agreeing experts by comparing each unique move up against the move calculated in step 2. Save the list of agreeing experts in **`matchingMoves`**.
5. Convert the list of integers in the list **`matchingMoves`** to string names, being used in a hash table in BREVE as the actual lookup table. List of names saved in **`lookupNames`**.
6. Use the lookup table (**`namesLookupTable`**) and find the maximum and minimum value, store these values in the list **`lookupValues`**.
7. Decide which expert lists to use for deciding the next target, and hence deciding the following moves and action until the target has been chasen down or until it disappears. Either explore to gain information and knowledge, or exploit to try and avoid negative feedbacks and get positive reward signals. Save the list of experts agreeing on this move in **`lastAgreementAgents`**.
8. If target is not locked from before, then set this target to be the locked target by giving the index value of one of the experts in agreement of this move into the parameter **`lockedTarget`**. If target is already locked, continue pursuing that target until its distinction.
9. Upon impact, check the edibility of the target. If not editable, the agent receives a negative reward signal, if edible, it receives a positive reward signal. Set the value of the **`lockedTarget`** to -1, implying a new target must be decided on. Remove the target from the simulation. Increment the parameter **`good`** or **`bad`** depending on the reward signal. Do a print out of the current values of the **`good`** and **`bad`**, to a file, “output.txt” for viewing and evaluation later.
10. Use the reward signal to update the values in the lookup table. This is done by looking at the list of agreeing experts that lead to this reward signal, stored in **`lastAgreementAgents`**. The values in the lookup table **`namesLookupTable`** are then incremented by the value stored in the parameter **`RL_INCREMENT`** multiplied with the reward signal, which is either -1 or 1.

The agent undergoes these 10 points in each time step of the simulation, and will more and more have the characteristic of being in an exploitation mode as time passes by.

There are always targets available, and the targets have a relative short lifespan. This ensures a variety of the different targets present, since the agent will by learning its behaviour decrease the current number of edible targets. Adding new targets is done by the main controller, the headquarters and the staring point of the simulation.

4.4.2 Produced output saved in a text file

The file "output.txt" will after a run contain a list. By analyzing this list one can make statements about the agent's achievements and learning capabilities. The number of bad hits should decrease into the simulation, and hopefully come to a stop, while the number of good hits keeps increasing. Below is an example of a fragment of the content of the file. Each hit is recorded, stating the number of positive and negative rewards received so far. A graphical visualization illustrating the learning effect can be done by showing the ratio between positive and negative rewards, as illustrated in Figure 18.

0	good	1	bad
1	good	1	bad
2	good	1	bad
2	good	2	bad
3	good	2	bad
4	good	2	bad
5	good	2	bad
6	good	2	bad
7	good	2	bad
8	good	2	bad
9	good	2	bad
10	good	2	bad
11	good	2	bad
12	good	2	bad
13	good	2	bad
14	good	2	bad
15	good	2	bad
15	good	3	bad
16	good	3	bad
17	good	3	bad
18	good	3	bad
19	good	3	bad
20	good	3	bad
21	good	3	bad
22	good	3	bad
23	good	3	bad
24	good	3	bad
25	good	3	bad
26	good	3	bad

27	good	3	bad
28	good	3	bad
29	good	3	bad
30	good	3	bad
31	good	3	bad
31	good	4	bad
32	good	4	bad
33	good	4	bad
34	good	4	bad
35	good	4	bad
36	good	4	bad
37	good	4	bad
38	good	4	bad
38	good	5	bad
39	good	5	bad
40	good	5	bad
41	good	5	bad
42	good	5	bad
43	good	5	bad
44	good	5	bad
45	good	5	bad
46	good	5	bad
47	good	5	bad
48	good	5	bad
49	good	5	bad
50	good	5	bad
51	good	5	bad
52	good	5	bad
53	good	5	bad
54	good	5	bad
55	good	5	bad
56	good	5	bad
57	good	5	bad
58	good	5	bad
59	good	5	bad
60	good	5	bad
61	good	5	bad
62	good	5	bad
63	good	5	bad
64	good	5	bad
65	good	5	bad
66	good	5	bad

67	good	5	bad
68	good	5	bad
69	good	5	bad
70	good	5	bad
71	good	5	bad
72	good	5	bad
73	good	5	bad
74	good	5	bad
75	good	5	bad
76	good	5	bad
77	good	5	bad
78	good	5	bad
79	good	5	bad
80	good	5	bad
81	good	5	bad
82	good	5	bad
83	good	5	bad
84	good	5	bad
85	good	5	bad
86	good	5	bad
87	good	5	bad
88	good	5	bad
89	good	5	bad
90	good	5	bad
91	good	5	bad
92	good	5	bad
93	good	5	bad
94	good	5	bad
95	good	5	bad
96	good	5	bad
97	good	5	bad
98	good	5	bad
99	good	5	bad
100	good	5	bad

Figure 18: Example of content in file output.txt

Upon every reward signal received, the result is printed to the file output.txt. It shows the number of positive and negative rewards received. The desired result is for the increasing of negative reward counts to stop, meaning the agent avoids getting negative rewards.

4.4.3 Interpreting the numbers

During the run, the count of positive and negative rewards are printed on the screen, as well as the ratio between them, as illustrated in Figure 19.

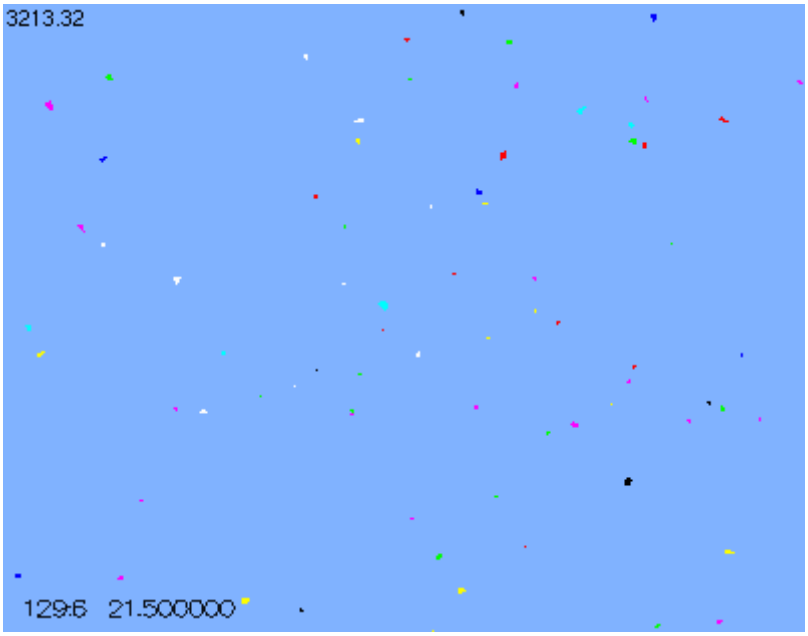


Figure 19: Screen during runtime

In the bottom left corner is the counts of positive and negative rewards displayed, with the ratio shown. In the upper left corner is the runtime shown. A desired effect is to see the ratio value increasing during the run, verifying the agents increasing capability of choosing actions leading to positive rewards and avoiding negative rewards. If no learning takes place, the ratio between positive and negative rewards will simply reveal the ratio of existing poisonous agents existing in the world.

Plotting the numbers saved in the output.txt into a excel worksheet allows me to show the development of the ratio between good and bad rewards received. It is the ratio shown taken from the example given in Figure 18 that is displayed in Figure 20.

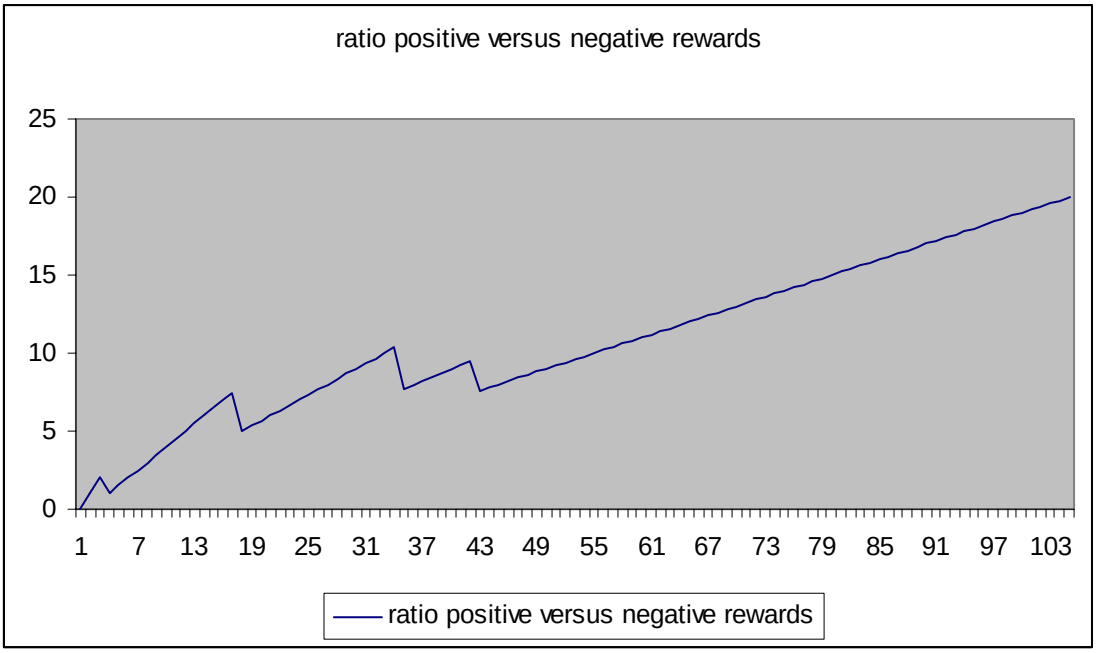


Figure 20: Ratio positive versus negative rewards

As the graph displays, the ratio of good rewards versus negative rewards keep increasing. Thus, the agent keeps getting better at avoiding targets that are not edible.

5 Results

Experimenting using the BREVE simulator gave a great opportunity to watch the behaviour of the agent during runtime. The observance of the behaviour was guiding my experimenting, and it took some work and time to reach the architecture described in this thesis. The goal was to find the proper abstraction level, assuming that the novel strategy of filtering the input signals through experts prior to the agent working on the data would actually work.

Starting out in an approach very similar to standard group-of-experts regime ala Jacobs et al approach in [4], the action selection was taken at each iteration, and the influence of the experts filtered input signals would influence every next action chosen. This, however, turned out to be problematic. A this low abstraction level in my setting meant that the agreements between the different agents would decide every single action performed by the agent, and was subject to change for each time step. In the early process of the testing, I had problems with avoiding the agent to get stuck between two different targets. It would move towards one agent, and in the next time step change direction and move towards another. The agent would often get stuck in such a state, moving forward very slowly, and not being able to actually reach any targets. Only at rare occasions, if ever, would it reach a target, spending most of its time going back and forth between possible targets. The output file "output.txt" does not reveal this as the output reveals only the ratio between good and bad rewards received, and not the time spend between each reward signal given. Using the BREVE simulator gave an opportunity to observe the agents behaviour during runtime and made this problem very easy detectable.

This undesired behaviour can be reproduced in the current code by simply commenting out the code in the method *doNext* where the agents checks if a target has been locked already, which was the main essence in solving this problem mentioned above.

This motivated me to try and raise the abstraction level. In stead of doing the traditional action selection, where every move is decided at each time step, I instead would use the input from the agents to decide on a target. Once a target has been decided, the Chief of Command, Figure 15, would decide and execute the next actual move, moving in closer towards the target. The target would remain fixed until it has been chased down. This seemed to work nice, but presented a different problem: after a short while most edible targets had been eaten, and mostly only poisonous agents were left in the world.

The solution was to maintain a certain level of both edible and poisonous agents in the world at all times. This was done by setting a maximum time span limit for the other agents to exist in. By balancing the number of total agents existing at all times along with their maximum life span a somewhat steady ratio between edible and poisonous agents was upheld. The agent could never finish off all edible food before new agents emerged. Also, the targets would disappear after a while, so the reproduction of new agents is not depended on the behaviour of the agent. This ensured a continuous supply of new agents, both edible and poisonous. By extending the lifespan of the agents, I needed to also increase the number of total agents present at all times, to insure the possibility of finding edible food.

I found that the balance of total number of targets and their individual lifespan made a difference on the result. The best result came with having many targets existing, and having their lifespan being extended. A minimum number of 100 agents with a lifespan between 100 and 200 seconds produced a good learning environment for the agent.

This reflects a typical challenge with reinforcement learning: the fine tuning of the parameters in order to achieve the desired results. Some of the parameters I changed, was determining the environment the agent lives in. I could do this, since I was in full control of that environment. If reinforcement learning is to be used in a real world setting, the opportunity to change the environment might be significantly less, to put it mildly. Then even more fine tuning of the parameters and methods inside the agent is required.

If the parameters deciding the environment were set in an unfortunate manner, the agent would fail to learn. The ratio between good and bad hits would simply reflect the ratio of edible versus poisonous food. This is shown in Figure 21. As illustrated in Figure 20, the graph should keep increasing as an effect of successful learning.

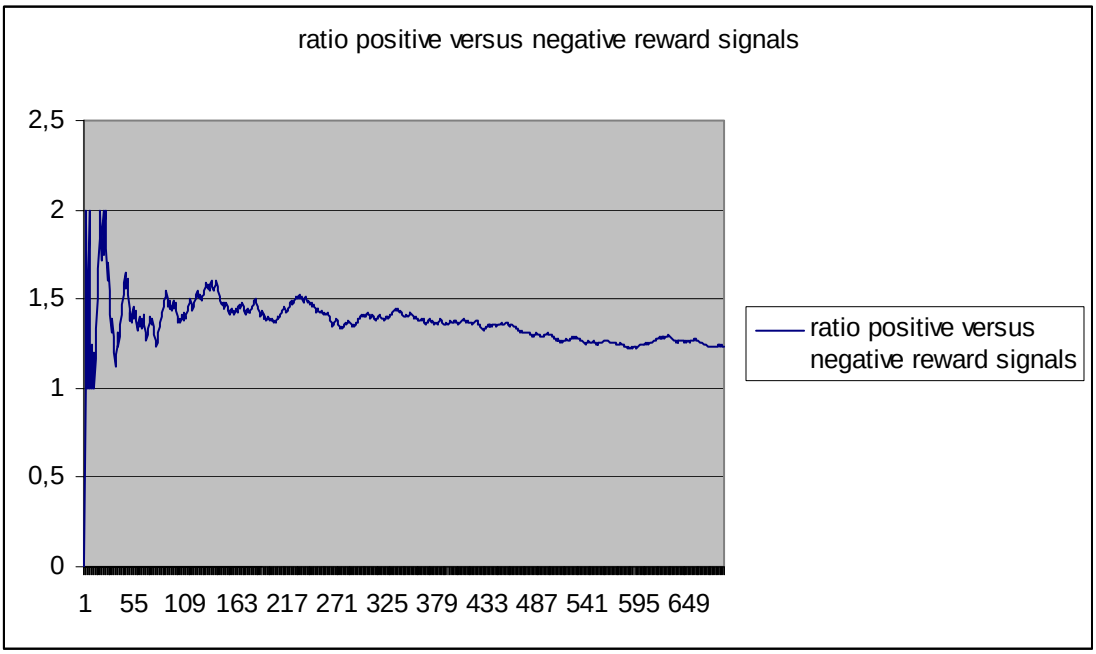


Figure 21: Not improving behaviour

As the graph displays, the ratio of good rewards versus negative stays fixed. The agent maintains the ratio of negative and positive rewards, and is not learning how to avoid the negative rewards. An example of the output file is in the attachments, "output_not_learning.txt".

As the agent finally was able to learn successfully and produce the desired behaviour, it still encountered negative rewards, all though not often. This is shown in Figure 22, where the poisonous agents were defined to be a sphere of colour red. Figure 23 shows the actual counts of positive and negative rewards generating the graph in Figure 22.

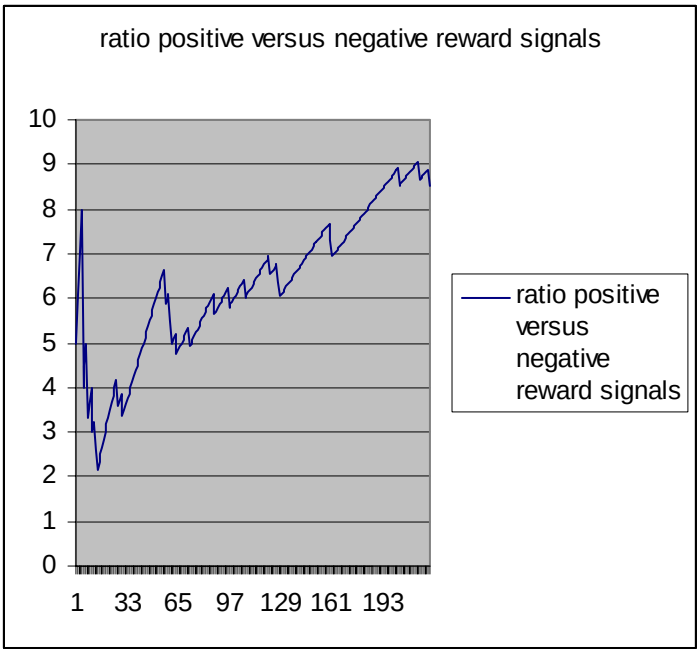


Figure 22: Learning, but still encountering negative rewards

As the graph displays, the ratio of good rewards versus negative increases. stays fixed. The agent maintains the ratio of negative and positive rewards, and is not learning how to avoid the negative rewards. An example of the output file is in the attachments, "output_learning_but_not_acting_perfectly.txt".

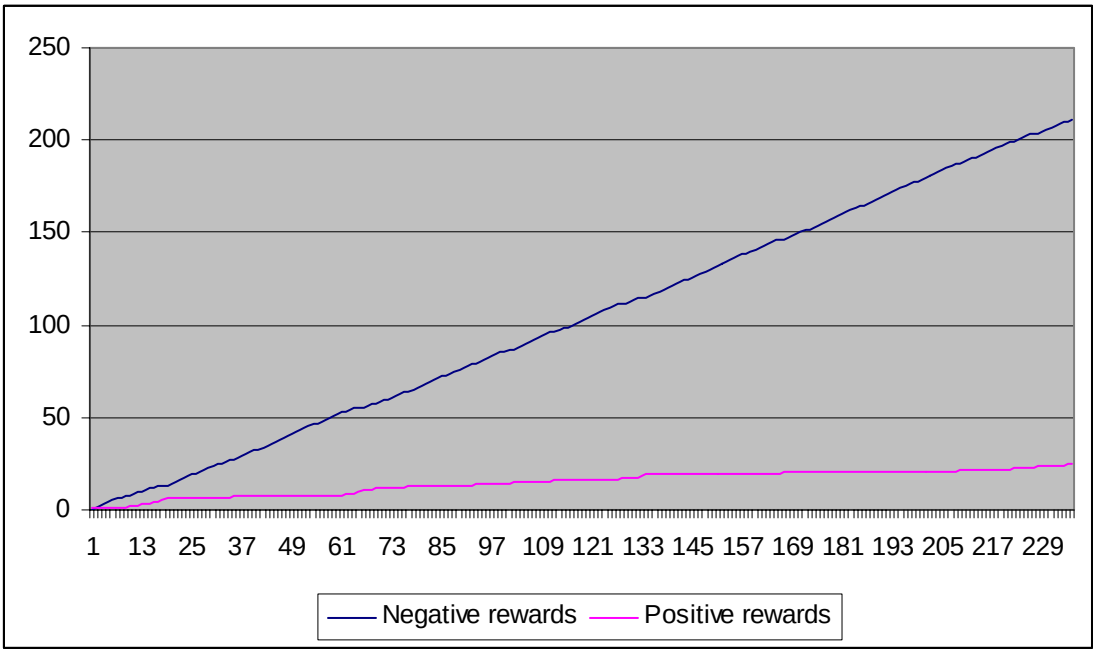


Figure 23: The numbers of positive and negative rewards

As the graph displays, the pink line keeps increasing. No matter that the agent learns, it still encounters negative rewards. This is from the same data as Figure 22. *An example of the output file is in the attachments, "output_learning_but_not_acting_perfectly.txt".*

The failure of the agent to complete avoid poisonous food led me to take a closer look at the code. The failure of the agent was of no fault of the agent. If it could only choose between poisonous targets, it would do so, in lack of any other alternative. Adding an escape from this, by adding the choice of not moving at if there are no predicted positive rewards in sight solved this issue. Instead

of having the agent standing still, waiting for edible targets to come close enough, one could add a behaviour that would cause the agent to circle randomly about while no edible targets are in sight.

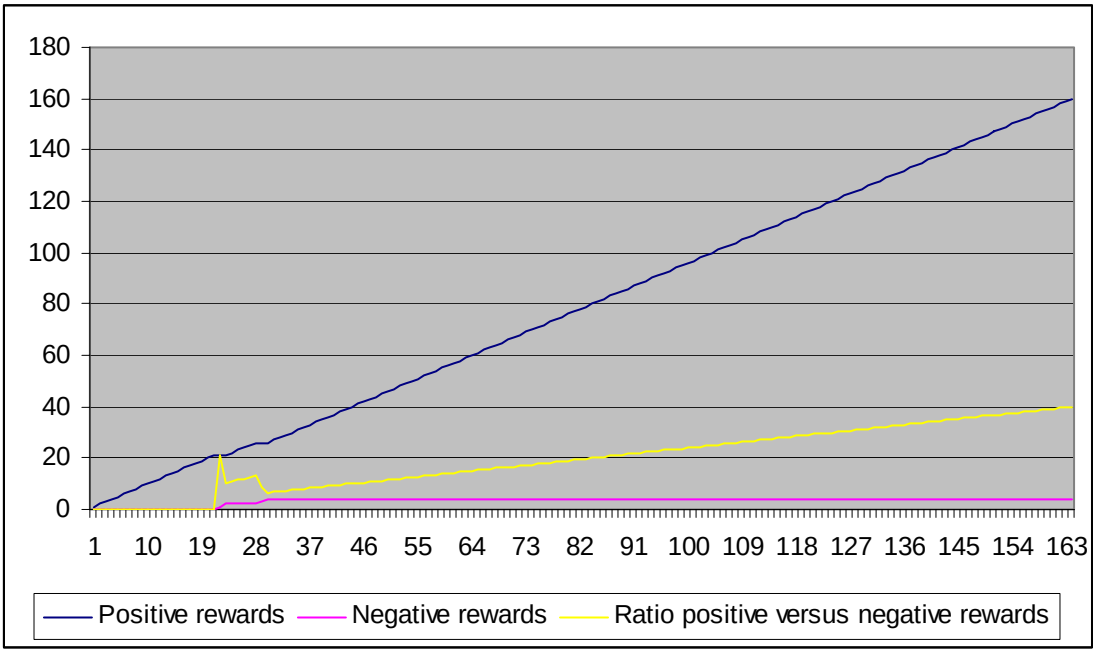


Figure 24: Successfull learning and behaviour

As the graph displays, the pink line stays fixed at 4. Having learned which targets are not edible, no more negative rewards are encountered. The ratio keeps increasing steadily. keeps increasing. An example of the output file is in the attachments, "output_successful_learning.txt".

Varying the number of combinations of edible agents would result in the same successful behaviour, but it would require a longer training period before the increasing counts of negative rewards came to a stop. This can be explained by the fact that given more combinations being non edible, the expert agreement values in the lookup table that should contain a positive value would more often get punished and increased as the likelihood of this agreement pair of expert would line up with an agreement pair of experts that pointed to non edible targets. An example of the output from such runs is given in "output_triple.txt", where the first run is with two combinations of non edible targets, and the second is with three non edible targets. As we can see in , the values in the lookup table does reflect the defined non edible targets, but the learning takes much longer. The average number of received rewards in the example below was 1000, and still the agent has not learned enough to avoid the negative rewards, but as the values show, the values are moving towards reflecting which agreeing experts points to edible food and not.

sphere sphere	-0,16
sphere cone	0,00
sphere disk	0,00
sphere red	0,24
sphere blue	-0,68
sphere green	0,40
cone cone	0,32
cone disk	0,00
cone red	-0,84

cone blue	0,96
cone green	0,80
disk disk	1,00
disk red	1,00
disk blue	1,00
disk green	1,00
red red	0,24
red blue	0,40
red green	0,12
blue blue	1,00
blue green	0,68
green green	0,96

Figure 25: Slow learning, but progressing

As the numbers reveal, in this setting, red cones and blue spheres are not edible. The learning is slower than when there are fewer combinations of non edible targets. An example of the output file is in the attachments, "ouput_triple.txt".

Again, the fine tuning of the parameters might be a solution to shorten the length of the training period, allowing the agent to reach its desired behaviour earlier. Since the possibility of encountering a negative reward, without any prior experience, is so much higher due to the increased combinations of characteristics defining the non edible targets, it results in a much more frequent decrease of the values stored in the lookup table. One possible way to counter effect this might be to give the positive rewards a greater impact on the increments being done in updating the lookup table.

Having the same combination of non edible agents as shown in Figure 24, but increasing the number of those agents relative to the total number of agents did not have any effect on the learning. One might expect the same effect as with increasing the various combinations of non edible agents, since the ratio of edible agents in both cases would increase. But this proved to not be the case, as illustrated in .

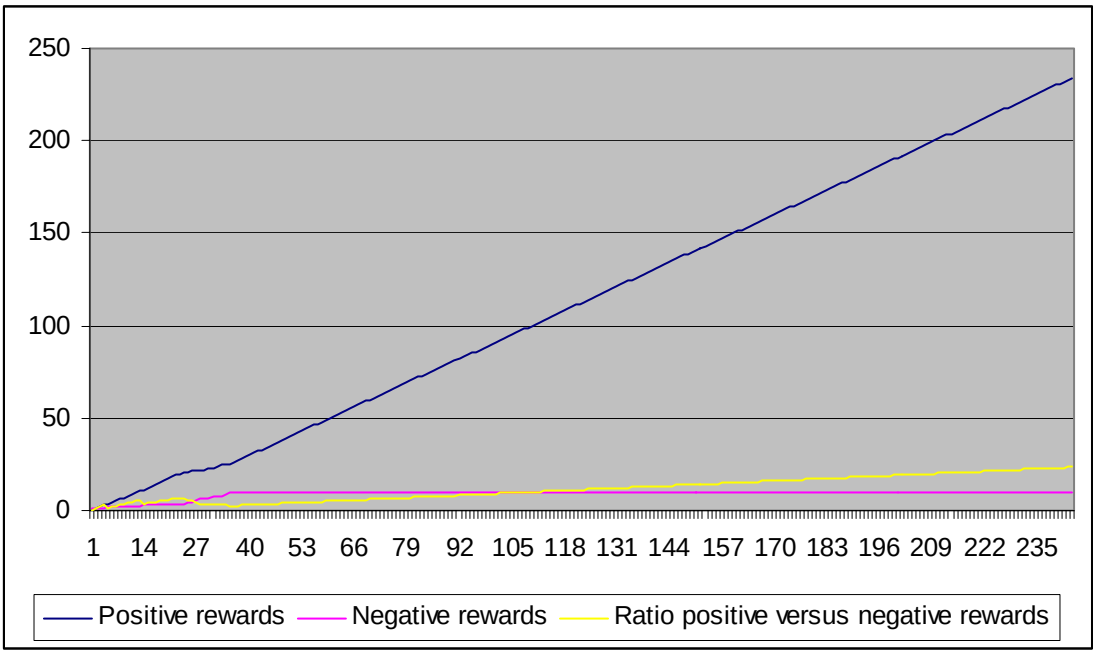


Figure 26: Displaying same learning and behaviour

As the graph displays, the pink line stays fixed at 10. Having learned which targets are not edible, no more negative rewards are encountered. The ratio keeps increasing steadily.

keeps increasing. This is the same result as in , and shows that the increased number of non edible agents does not cause an extended learning period before negative rewards are avoided. An example of the output file is in the attachments, "output_increasing_liklihood_of_number_of_targets_not_combinations.txt".

The experts should work at such a low abstraction level so their code can be executed fast, and not being subject to a huge time restraint. The agent was still able to learn the values, but encountered more hits with the non-edible targets.

Deciding more refined which targets are edible, the agent improved its learning by being able to learn the values faster. It was easier for the agent to distinguish the combination of for example green cone being poisonous versus all shapes with the colour green being non edible. If the colour green was the only characteristic needed to generate a negative reward signal, then the agent had to learn that any of the combinations green cone, green sphere or green disk was not edible.

If only a few combinations were set to be edible, the agent had to spend much more time updating the values in the lookup table. This can be explained by that the likelihood of a pair of agreeing experts, though not pointing to poisonous targets themselves, would often find themselves in an agreement with pairs of experts which did, and then be punished by having the value increased. This would then lead to a hugely increased learning period for the agent. A compensation would be to give a much higher increment on positive feedbacks, while smaller increments on negative feedbacks. This helped, but the agent still required a much longer training period before it could exploit its gained knowledge and avoid non edible fish.

The results being stored in the file "output.txt" after each simulation reveals only the ratio of good rewards versus bad rewards. It reflects the agent's ability to avoid poisonous food, but does not tell anything about the agents ability to exploit the edible food. It might, without it showing in the results stored, get stuck on one type of edible targets, and not take advantage of all the edible agents present. This might reflect upon the policy and value functions used in the architecture. The agent was not being punished for moving around, whether it moved a short or long distance between the targets. So it was not forced to learn to take advantage of the nearest, meaning the closest located relative to itself, targets. It was only punished for attacking non edible agents.

6 Conclusion

I was struggling for quite some time setting up the reinforcement system, and working out the policy and value functions. This is a typical challenge in reinforcement systems, and my approach is no different in that matter. It seemed harder for the agent to learn which are edible and not the higher ratio of not edible combinations of characteristics.

One weakness is the architecture of the value system, meaning the lookup table. In order to keep things on a small scale, I chose to use lookup table. An alternative approach would have been the use of neural networks, see Figure 28, but neural networks often tend to be slow learning systems, and it is often hard to find the good solution, or architecture, that will lead to the desired, or best, behaviour. By using a lookup table, I am in more control of what is happening and how, and can monitor the learning process more consciously.

However, using lookup tables puts a restrain on the complexity, or size. If the lookup table is too big, it will be unfeasible to learn how to set the parameters right in an online learning agent system. So the lookup table has to remain of a manageable size. To insure that the agent still has the potential of learning the optimal behaviour, one needs to be sure that all the variety necessary to induce such learning can be expressed in such a look up table.

In my experiment, I allow every expert to be measured against every other experts, creating a list of (n^2 – n)/2 tupples. These tupples are illustrated as the connecting lines in Figure 27. These measurements, represented as a float value between -1 and 1 in a lookup table, are used by the agent to decide what target to chase down. This allows for any two combinations to define a poisonous agent, but not more. A combination requiring three different characteristics to be present simultaneously can not be expressed in this lookup table, and hence can not be learned by the agent.

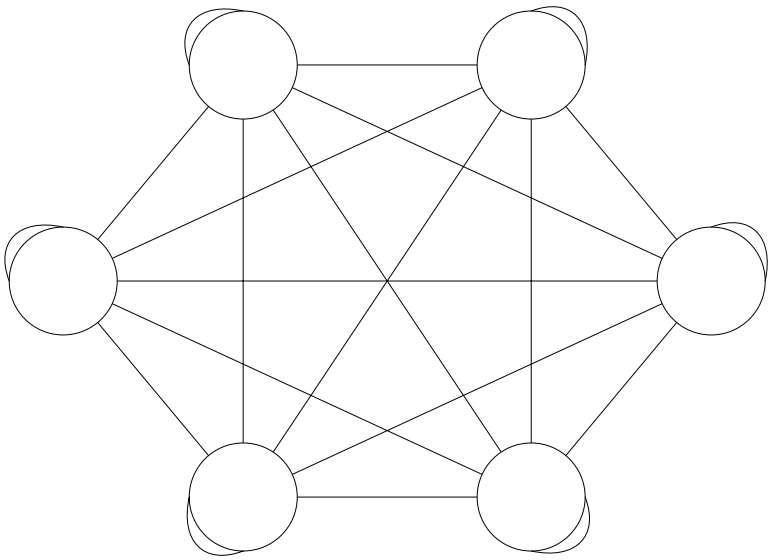


Figure 27: Agreement between experts

Each expert's filtered input is represented by a circle. A suggestion on what target to pursue is being calculated based on each experts filtered input. The connecting lines between the circles represent the agreement between the chosen targets based on the respective experts input signals. A connecting line is then either positive or not. The assigned value to each line, representing the agreement between two experts, is stored in a

lookup table, and are used to decide which target to pursue, and are subject for change as the agent learns.

Setting up the lookup table requires then a deeper insight of the problem and the filtered signals generated by the experts to ensure a possibility of detecting the parameters that defines what is desirable and not. The greater and more complex problem, the bigger lookup table is needed, and with any problems of significant size it soon becomes unfeasible. However, by reducing the input signals by having it filtered by the agents, and then have a lookup table assign credibility values to the filtered signals from the various experts, I don't have to represent every possible state in the world in the lookup table. Given that the only information available to the CC is the filtered input from the experts, the CC uses the experts to evaluate each expert's credibility in any given state represented in the lookup table. The lookup-table is hence reduces to a manageable size.

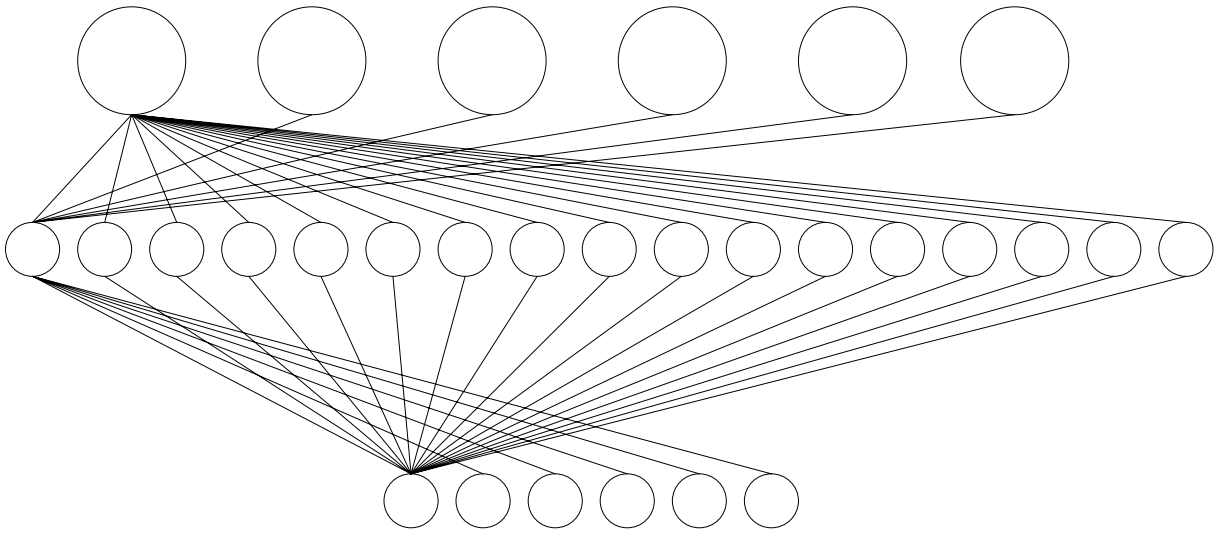


Figure 28: Neural network representing the agreements between the agents

This figure illustrates a possible shape of a neural network. The connecting lines between the nodes are not completed in this illustration, as the figure serves only to demonstrate how the lookup table can be replaced by a neural network. Each top circle is decided and connected to one separate expert. The different agreements are represented by the middle layer, and the final outcome in the output nodes. The weights in the network play the same role as the float values in the lookup table as in assigning credibility to the different agreements.

Another weakness is the simplicity of the setting. The idea was tested out in a very limited arena, and on one specific problem. To convey credibility to this approach, having tested it in other problem domains and environments would have been preferable. Many reinforcement learning tests and experiments are guilty of the critic raised in ref xx, that the experiment is conducted in a small environment with a relative small input space. This is an accurate criticism of this experiment, too. This experiment has been done on a relatively small scale, and the resulting behaviour of the agent could have been hand coded, and executed faster. However, the goal was to reduce the input signals, find at what abstraction level this is most beneficial, whilst being used in a reinforcement system. Therefore it was more important to experiment with the use of group of experts and reinforcement learning than to hand code the desired behaviour.

Having stated these weaknesses, I need to point out that the architecture opens for easily changing any module of the agent with a different built module. For instance, the lookup table can easily be replaced by a neural network, so that the agent can learn to define any number of combinations of the targets characteristics as being poisonous. Such a neural network is illustrated in Figure 22.

By doing the experiments one a small scale, the results do not guarantee a successful scaling up to large size problems of great complexity. The very goal of this thesis was to find an approach that would help in this scaling problem, the curse of dimensionality, but yet it itself suffers from being tested and experimented only on a small scale. Suggestions for future work is to refine the technique, and run analysis and compare the results with alternative ways of problem solving, to get a better understanding of how the group of experts regime might be beneficial dealing with reinforcement learning on larger scales.

It has been interesting and fun to work with this problem, and I hope that others will find this thesis of some value and interest. Having experimented with this architecture on a small scale, I believe that this approach is worth further investigation. It is my hope that this way of reducing the search space can be beneficial in many systems of great complexity. However, due to the weaknesses in my experiment, as pointed out in 5, there is no evidence that this will have the desired effect when the problem is of great magnitude and the task is of great complexity. The architecture, though, is easily adaptable to different systems, techniques and approaches. A weakness in say the learning system due to the usage of lookup tables is therefore by no means a good enough reason to discharge the fundamental idea. It can easily be changed, and the overall system might prove itself to be a fruitful approach for how to deal with problems of great magnitude and complexity when using reinforcement systems.

The problems encountered in the early stages of the testing suggests that moving the abstraction level to a higher level is preferable. However, there could be other reasons for the lack of success during the lower abstraction levels. Having achieved results with this novel approach by using the group of experts regime on a high abstraction level is backing up my hypothesis that the abstraction level should be high.

On those grounds I suggest that this architecture has not yet failed, but might deserve a closer investigation and more experimenting, even if I have not come up with hard proof of this system performing its task satisfactory when the magnitude and complexity gets high.

The answer to the main question raised in the problem description of this thesis, at what abstraction level should such a group of experts regime work at. My work suggests that it should be on a high abstract level. For the possibility of having a reinforcement learning system relying on a greatly simplified view of the world, the achieved simplification needs to be done intelligently. There has to be a certain level of quality and guaranteed information value incorporated in the filtered inputs, so that the RL system can rely only on those filtered inputs and maintains its ability to learn how to behave in the more complex world.

To insure some quality of the filtered input, and that the necessary information is to be found in the combinations of the different filtered input signals, a high abstraction level is preferred. However, the lower level experts themselves should not be able to produce the preferred behaviour themselves, as this would not require a higher level module.

I will not make any conclusions very bombastically, as my results might be depended on my special setting for the experiment, the chosen simulator and programming language, as well as the typical challenges accompanied with the technique or using reinforcement learning systems. Only further research might be able to reach a general applicable conclusion on where the abstraction level for such a system should be.

7 Future work

If this idea is to be further investigated, I would advice to test out the architecture in different domains and varying the magnitude of the input space. It would also be interesting to see how the higher level module, which I called the Chief of Command, could deal with lower level experts that themselves were learning and changing. This is not something I tested out in my experiments. Once the correct behaviour has been learned, the agent depended on the lower level experts to maintain their ways of filtering the data.

When trying out the architecture in different problem domains, when this architecture proves itself to be successful, it should be compared to other ways of attacking the same problem. In this way, the value of this architecture could be measured up against other existing ways of dealing with such problems. By doing comparisons up against other approaches one might gain an understanding of which domains this architecture is most suitable, and which domains are better addressed using other ways of solving the problem.

Bibliography

[1] P. Redgrave, T. Prescott, and K. Gurney, **The basal ganglia: a vertebrate solution to the selection problem**, 1999.

[2] Rodney A. Brooks, **Intelligence without representation**, Artificial Intelligence, 1991, number 44, pages 139-159.

[3] Rosenblatt, DAMN, **A Distributed Architecture for Mobile Navigation**, Proc. of the AAAI Spring Symp. on Lessons Learned from Implemented Software architectures for Physical Agents, Stanford, CA, 1995

[4] R. A. Jacobs, M. I Jordan, S.J. Nowlan and G.E Hinton**, Adaptive mixture of local experts,** 1991. Neural Computation, Vol. 3, pp 79-98

[5] R. S. Sutton and A. G. Barto: **Reinforcement Learning I: An Introduction**, MIT Press: Cambridge, MA. 1998.

[6] Jurgen Schmidhuber, **Evolutionary Computation versus Reinforcement Learning**,

[7] A. G. Barto, in **Models of Information Processing in the Basal Ganglia**, edited by D. B. J. C. Houk, J. Davis, The MIT Press, 1995, pp. 215–232.

[8] S. Keerthi and B. Ravindran, **A tutorial survey of reinforcement learning**, Sadhana (published by the Indian Academy of Sciences), January 1995

[9] Leslie Pack Kaelbling and Michael L. Littman and Andrew P. Moore, **Reinforcement Learning: A Survey**, Journal of Artificial Intelligence Research, volume 4, pages 237-285, 1996

[10] Sham Kakade and Peter Dayan, **Dopamine: generalization and bonuses**, Neural Networks 15, 2002 special issue, p 549-559.

[11] B. Bakker and J. Schmidhuber. **Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization: First experiments with the HASSLE algorithm**. Technical report, IDSIA, 2003

[12] Van Roy, B., **Learning and Value Function Approximation in Complex Decision Processes**, Ph.D. Thesis, Massachusetts Institute of Technology, May 1998

[13] Barto, A.G., Mahadevan, S.: **Recent advances in hierarchical reinforcement learning**. Discrete event systems (2003, to appear)

[14] Mahadevan, S., Maggioni, M.: **Value Function Approximation with Diffusion Wavelets and Laplacian Eigenfunctions**, Technical Report 2005-38, University of Massachusetts, 2005

[15] Sutton, S., McAllester, D., Singh, S., Mansour, Y.: **Policy Gradient Methods for Reinforcement Learning with Function Approximation**, Advances in Neural Information Processing Systems 12, pp. 1057–1063, MIT Press, 2000

[16] Mehdi Khamassi and Loïc Lachèze and Benoît Girard and Alain Berthoz and Agnès Guillot, Actor-Critic **Models of Reinforcement Learning in the Basal Ganglia: From Natural to Artificial Rats**, Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems, vol 13, number 2, 2005, pages 131-148, Sage Publications, Inc., Thousand Oaks, CA, USA

[17] Dayan, Peter: **Unsupervised Learning**, appeared in Wilson, RA & Keil, F. editors. The MIT Encyclopaedia of the Cognitive Sciences, 1999. http://www.gatsby.ucl.ac.uk/~dayan/papers/dun99b.pdf

[18] Eugene Charniak, **Bayesian Networks without Tears**, http://www.cs.ubc.ca/~murphyk/Bayes/Charniak_91.pdf

[19] David Heckerman, **A tutorial on learning with bayesian networks**. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington, 1995. Revised June 96

[20] Peshkin, Leonid: **Reinforcement Learning by Policy Search**, PhD thesis, Brown University, Providence, RI, 2001

[21] Moore, AW & Atkeson, CG.: **Prioritized sweeping: Reinforcement learning with less data and less real time**, journal Machine Learning, volume 13, pages 103-130, year 1993. http://citeseer.ist.psu.edu/moore93prioritized.html

[22] Morales, Eduardo F.: **Scaling Up Reinforcement Learning with a Relational Representation**, Computer Science and Engineering, University of New South Wales, Australia

Appendix A

A. Attachments

a. Source code

The source code is attached to the thesis. The BREVE simulator is continuously going through improvements and updates, so the source code attached here might not be exactly like the ones one might currently download from the BREVE website. The website does keep a good track of the changes and updates that has been done to the simulator, so using different versions should not be a problem. The latest version being used in this thesis is the version 2.5, that was released the 12th of December 2006. To run the code, it is recommended to download the BREVE simulator from its webpage www.spiderland.org/, and just copy and paste the two code files "the agent" and "abstract000.tz" from the appendix. Saving these two files in the same catalogue where the exe-file for the BREVE simulator is should be sufficient for running the code. Open the agent.tz file and run the simulator.

b. Samples of content of file "output.txt"

The graphs and figures used in chapter 5 and chapter stems from data printed to the file output.txt during the simulations. These data are attached in the file "data output file". I have chosen to add this as a separate file, as these data are not of great interest to include in a printed example of this thesis. They are attached, though, to show the data material underlying the figures shown discussing the results in chapter 5.

The Object class

```
% The root object in breve.

NULL : Object (aka Objects) [version 2.1] {
        % Summary: the top level object class.
        % <P>
    % The Object class is the root class.  All classes used in breve have
    % Object as an ancestor.  The object class implements some basic
    % services that all classes will have access to.
    % <p>
    % Subclassing Object directly is rare.  The classes OBJECT(Real) and
        % OBJECT(Abstract) are logical separations of the Object class containing
        % "real" objects (which correspond to a physical entity in the simulated
        % world) and "abstract" objects which are generally used for computation
        % or control of the real objects.  You should consider subclassing
    % one of these classes instead.

    + variables:
        objectReferences (int).
        controller (object).
               birthTime (float).

    + to init:
               objectReferences = 1.
        controller = (self get-controller).

               birthTime = (controller get-time).

    + to destroy:
        % Automatically called when this object is freed.  This method
        % should never be called manually.  If subclasses need to free
        % objects or data, they should implement their own "destroy"
        % methods.

        + section "Garbage Collection and Memory Management"

        + to enable-auto-free:
               % Experimental--enables garbage collection on a per-object basis.

               setGC(1).

        + to disable-auto-free:
               % Experimental--disables garbage collection on a per-object basis.

               setGC(0).

        - to get-retain-count:
               return getRetainCount().

    - to get-controller:
        % Returns the controller object that associated with the current
        % simulation.  It's preferable to simply reference the variable
        % "controller".

        return getController().

  + section "Getting Information About an Object"

        + to get-age:
               % Returns the number of seconds this object has existed in the
               % simulation.

               return (controller get-time) - birthTime.

    + to get-type:
        % Returns as a string the type of this object.

        return objectName(self).

    + to is a className (string):
        % This method returns true or false (1 or 0) depending on whether
        % the instance in question belongs to class className.  This
        % method checks if className is a superclass of the current
        % object as well.
        % <p>
        % The declaration of this method looks strange, but it should be
        % looked at as a statement about an instance which returns true
        % or false: object is a "thing".

        return isa(className).
```

```
    - to is-a-subclass of className (string):
              % Returns 1 if this object is a subclass of the class specified with
              % className, returns 0 otherwise.

              return (self is a className).

    + to can-respond to methodName (string):
        % Returns true or false (1 or 0) depending on whether this instance
        % can respond to a method called methodName.
        % <p>
        % But wow, what an awkward declaration!  Same reason as the
        % method METHOD(is).  Again, works like a statement that
        % replies with true or false: object can-respond to "run".
        % <p>
        % It's really not my fault that the infinitive of "can" is "be able".

        return respondsTo(self, methodName).

       + to get-description:
              % This method should provide a textual description of an object.
              % When the "print" command prints an object, it calls this method
              % to get a description of the object.  By default, print will show
              % the class and pointer of an object, but by overriding this method
              % in your own classes, you can append other sorts of data to the
              % output.

  + section "Scheduling Events and Notificiations"

    + to schedule method-call theMethod (string) at-time theTime (float):
        % Schedules a call to theMethod when the simulation time equals
        % theTime.  The margin of error of the callback time is equal
        % to iteration step (see METHOD(set-iteration-step)).
        % <br>
        % If you want to schedule an event at a time relative to the
        % current time, use the method METHOD(get-time) to get the
        % current simulation time and then add the offset you want.

        addEvent(theMethod, theTime, 0).

       - to schedule-repeating method-call theMethod (string) with-interval theInterval (double):
        addEvent(theMethod, theInterval, theInterval).

    + to observe instance theObject (object) for-notification theNotification (string) with-method
theMethod (string):
        % Causes the current object to observe theObject.  By registering as
        % and observer, the current object will receive a call to theMethod whenever
        % theObject calls the METHOD(announce) method with notificiation
        % theNotification.

        addObserver(theObject, theNotification, theMethod).

    + to unobserve instance theObject (object) for-notification theNotification (string):
        % Unregisters the current object as an observer of theObject with
        % notification theNotification.

        removeObserver(theObject, theNotification).

    + to announce message theMessage (string):
        % Sends a notification with the message theMessage to all observer
        % objects.  See METHOD(observe) for information on making an object an observer.

        notify(theMessage).

       + to call-method named methodName (string):
              % Calls the method named methodName for this object.  Returns the result of
              % the method call.

              return callMethodNamed(self, methodName, { }).

       + to call-method named methodName (string) with-arguments argList (list):
              % Calls the method named methodName for this object.  Returns the result of
              % the method call.
              % <p>
              % The arguments to the object are passed in using the list argList.  Since
              % keywords are not passed in, this method relies on the order the arguments
              % appear in the argument list and passes them to methodName in the order
              % in which they appear in methodName's definition.
              % <p>
              % Why not call a method
              % directly?  This method is used in circumstances where you might
```

```
                % want to have some sort of callback method.  As an example, let's
                % say you write a general purpose class which can sort objects based
                % on different criteria.  How would the user specify these arbitrary
                % criteria?  Using this method would allow the user to pass in the
                % name of the method they want to use, and the sorting object could
                % use this method to execute the callback.
                % <p>
                % If the concept of a callback doesn't make sense, then you can
                % probably ignore this method.

                return callMethodNamed(self, methodName, argList).

    + section "Archiving & Dearchiving"

      + to archive-as-xml file fileName (string):
          % Writes the current object to the XML file fileName.

          archiveXMLObject(self, fileName).

        + to send-over-network to hostName (string) on portNumber (int):
                % Sends this object over the network to a breve server on host hostName
                % listening on port portNumber.

                sendXMLObject(hostName, portNumber, self).

        + to add-dependency on i (object):
                % Makes this instance depend on instance i when archiving and
                % dearchiving.  This means that if this instance is archived,
                % then i will also have to be archived, and that when this
                % instance is dearchived, that i will have to be dearchived
                % first.
                % <p>
                % Dependencies can cause large numbers of instances to be archived
                % in response to a single archiving event (as dependencies of
                % dependencies, and dependencies of dependencies of dependencies, ad
                % infinitum will also be archived).  This means that you should make
                % dependencies sparingly, only when absolutely required.
                % <p>
                % Circular dependencies are forbidden.

                if i: addDependency(i).

        + to remove-dependency on theObject (object):
                % Removes theObject from this object's dependency list.  See
                % METHOD(add-dependency) for more information on dependencies.

                removeDependency(theObject).

        - to post-dearchive-set-controller:
                % Used internally to set the controller instance for this variable.
                % Used after object dearchiving.

                controller = (self get-controller).

        + to dearchive:
                return 1.

        + to archive:
                return 1.
}
```

The line class

```
Object : Line {
       + variables:
              start, end (object).
              linePointer (pointer).

       + to connect from start (Real object) to end (Real object)
              with-color theColor = (0, 0, 0) (vector) with-style theStyle = "----------------"
(string):
              % Adds a line to be drawn between Real objects start and end.
              % <P>
        % The optional argument theColor specifies the color of the line.  The default
        % color is the vector (0, 0, 0), black.
        % <P>
        % The optional argument theStyle specifies a pattern for the line.  theStyle is a
        % string of 16 spaces and/or dashes which specify the line pattern to be drawn.
        % A dotted line, for example, would use the pattern "- - - - - - - - ".  A thickly
        % dashed line would use the pattern "--------        ".  If no style is given, a
        % solid line is drawn.

              if start == end: return.

              linePointer = addObjectLine((start get-world-object-pointer), (end get-world-object-
pointer), theColor, theStyle).

       + to is-linked to worldObject (object):
              % Returns 1 if this line is associated with worldObject.

              if start == worldObject || end == worldObject: return 1.
              return 0.

       + to destroy:
              if linePointer: removeObjectLine(linePointer).
}
```

The Real class

```
@use Object.
@use Line.

Object : Real (aka Reals) [version 2.1] {
        % A class which is never instantiated--just used as a logical
        % distinction from the Abstract classes.  See the child classes
        % OBJECT(Mobile), OBJECT(Link) and OBJECT(Stationary) for
        % more information.  The methods documented here may be used
        % with any of the child classes.

        + variables:
                realWorldPointer (pointer).
                collisionHandlerList (list).
                texture (int).
                lightmap (int).
                bitmap (int).
                menus (list).
                color (vector).
                lines (list).

                textureScaleX, textureScaleY (float).
                neighborhoodSize (float).

                eT (float).
                e (float).
                mu (float).

        + to init:
                texture = -1.
                lightmap = -1.
                bitmap = -1.

                textureScaleX = 16.
                textureScaleY = 16.

                self add-menu named "Delete Instance" for-method "delete-instance".
                self add-menu named "Follow With Camera" for-method "watch".

        - to get-world-object-pointer:
                % Used internally.

                return realWorldPointer.

        + to watch:
                % Makes the camera follow this object.

                controller watch item self.

        - to delete-instance:
                % Produces a dialog box (if supported by the current breve
                % engine) asking if the user wants to delete the object.
                % This is typically used in response to a user action like
                % a click or menu callback.

                result (int).

                result = (controller show-dialog
                                        with-title "Really Delete Instance?"
                                        with-message "Deleting this object may cause a fatal error in
the simulation.  Really remove it?"
                                        with-yes-button "Okay"
                                        with-no-button "Cancel").

                if result: free self.

  + section "Configuring Physical Properties of the Object"

        + to set-e to newE (float):

                % Sets the "coefficient of restitution" a value which
                % determines the elasticity of the object in a collision.
                % Valid values range from 0.0 to 1.0, with 0.0 representing
                % a totally inelastic collision (such as a lump of clay)
                % while 1.0 represents a totally (and unrealistically)
                % elastic collision (such as a rubber ball).

                e = newE.
                setCollisionProperties(realWorldPointer, e, eT, mu).
```

```
        + to set-eT to newET (float):
                % Sets the "tangential coefficient of restitution", a
                % frictional version of the "coefficient of restitution"
                % described in the documentation for METHOD(set-e).  The
                % value ranges from -1.0 to 1.0.  Negative values mean that
                % friction pushes against the sliding object.

                eT = newET.
                setCollisionProperties(realWorldPointer, e, eT, mu).

        + to set-mu to newMu (float):
                % Sets the coefficient of friction to newMu.  mu is a
                % parameter controlling friction between two bodies and
                % may be any value between 0 and infinity.

                mu = newMu.
                setCollisionProperties(realWorldPointer, e, eT, mu).

    + section "Using Menus"

        + to add-menu named menuName (string) for-method theMethod (string):
                % Adds a menu named menuName to the application which will result
                % in a call to theMethod for the calling instance.
                % <p>
                % If the calling instance is the Controller object, then the menu will
                % become the "main" simulation menu.  Otherwise, the menu will become
                % a contextual menu associated with the specific object in the simulation.
                % <p>
                % Note that unlike the METHOD(handle-collision) which sets the collision
                % handler for the whole type (class, that is), this method affects only
                % the instance for which it is called, meaning that each instance of a
                % certain class may have a different menu.

                newMenu (object).

                newMenu = ((new MenuItem) create-menu named menuName for-object self for-method
theMethod).
                self add-dependency on newMenu.
                push newMenu onto menus.

                return newMenu.

        + to add-menu-separator:
                % Adds a separator menu item--really just an empty menu item.

                newMenu (object).

                newMenu = ((new MenuItem) create-menu named "" for-object self for-method "").
                self add-dependency on newMenu.
                push newMenu onto menus.

                return newMenu.

    + section "Using Neighbor Detection"

        + to get-neighbors:
                % Returns a list of all real objects in the simulation that are within
                % the "neighborhood" range of this object in the world.

                return getNeighbors(realWorldPointer).

        + to set-neighborhood-size to size (float):
                % Used in conjunction with METHOD(get-neighbors), this function will set
                % the neighborhood size for the current object.

                neighborhoodSize = size.
                setNeighborhoodSize(realWorldPointer, size).

        + to get-neighborhood-size:
                % gets the neighborhood size for the current object.

                return neighborhoodSize.

    + section "Changing the Appearence of Agents"

        + to show-bounding-box:
                % Shows the bounding box for the object.

                setBoundingBox(realWorldPointer, 1).

        + to hide-bounding-box:
```

```
		% Hides the bounding box for the object.  The bounding box is
		% hidden by default, so you'll only need this method if you've
		% previously enabled them using METHOD(show-axis).

		setBoundingBox(realWorldPointer, 0).

	+ to add-line to otherObject (object) with-color theColor = (0, 0, 0) (vector) with-style
theStyle = "----------------" (string):
		% Adds a line to be drawn from this object to otherObject.  The line can be removed
		% later using METHOD(remove-line).
		% <P>
		% The optional argument theColor specifies the color of the line.  The default
		% color is the vector (0, 0, 0), black.
		% <P>
		% The optional argument theStyle specifies a pattern for the line.  theStyle is a
		% string of 16 spaces and/or dashes which specify the line pattern to be drawn.
		% A dotted line, for example, would use the pattern "- - - - - - - - ".  A thickly
		% dashed line would use the pattern "--------        ".  If no style is given, a
		% solid line is drawn.
		% <P>
		% If a line to otherObject already exists, its color and/or style will be updated.

		line (object).

		self remove-line to otherObject.

		line = new Line.

		line connect from self to otherObject with-color theColor with-style theStyle.

		push line onto lines.

	+ to add-dotted-line to otherObject (object) with-color theColor = (0, 0, 0) (vector):
		% Adds a dotted line to otherObject.  See METHOD(add-line) for more information
		% on object lines.

		self add-line to otherObject with-color theColor with-style "- - - - - - - - ".

	+ to remove-line to otherObject (object):
		% Removes the line connecting this object to otherObject.

		line (object).

		foreach line in lines: {
			if (line is-linked to otherObject): {
				free line.
				return.
			}
		}

	+ to remove-all-lines:
		% Removes all lines connecting this object to other objects.

		free lines.
		lines = { }.

	+ to show-axis:
		% Shows the X and Y axes for the object.

		setDrawAxis(realWorldPointer, 1).

	+ to hide-axis:
		% Hides the X and Y axes for the object.  The axes are hidden by
		% default, so you'll only need this method if you've previously
		% enabled them using METHOD(show-axis).

		setDrawAxis(realWorldPointer, 0).

	+ to show-neighbor-lines:
		% Draws lines to this objects neighbors (when neighbor checking is
		% enabled).

		setNeighborLines(realWorldPointer, 1).

	+ to hide-neighbor-lines:
		% Hides lines to this objects neighbors.

		setNeighborLines(realWorldPointer, 0).

	+ to make-invisible:
		% Makes the object invisible.  Can be made visible again later
```

```
        % using the method METHOD(make-visible).

        setVisible(realWorldPointer, 0).

+ to make-visible:
        % Makes the object visible again (if it has previously been hidden
        % using METHOD(make-invisible).

        setVisible(realWorldPointer, 1).

+ to get-color:
        % Returns the color of the object.

        return color.

+ to set-color to newColor (vector):
        % Sets the color of this object to newColor.

        color = newColor.

        setColor(realWorldPointer, newColor).

- to set-texture to textureNumber (int):
        % Deprecated -- use METHOD(set-texture-image) instead.

        setTexture(realWorldPointer, textureNumber + 1).

+ to set-texture-image to textureImage (object):
        % Changes the texture of this object to textureImage, an instance of
        % class Image.  If textureImage is NULL texturing is turned off for
        % the object.

        if !textureImage: texture = -1.
        else texture = (textureImage get-texture-number).

        setTexture(realWorldPointer, texture).

+ to draw-as-point:
        % Draws the object as a single point.  This is by far the fastest way
        % to display an agent.  Points can be used to draw upwards of 20,000
        % agents with a reasonable frame rate, while drawing as many spheres or
        % bitmaps would slow down the simulation significantly.

        setDrawAsPoint(realWorldPointer, 1).

+ to set-texture-scale to scaleSize (float):
        % Changes the "scale" of the texture.  When a texture is applied
        % over a shape, this value is used to decide how large the texture
        % will be in terms of breve-world units.  The default value is 16,
        % meaning that a 16x16 face will have one copy of the textured image.
        % For smaller objects, this number will have to be decreased, or else
        % the texture will be too big and will not be visible.

        textureScaleX = scaleSize.
        setTextureScale(realWorldPointer, scaleSize, scaleSize).

+ to set-texture-scale-x to scaleSize (float):
        % Sets the texture scale in the X dimension.  The Y texture scale
        % value is unchanged.  See METHOD(set-texture-scale) for more information.

        textureScaleX = scaleSize.
        setTextureScale(realWorldPointer, scaleSize, textureScaleY).

+ to set-texture-scale-y to scaleSize (float):
        % Sets the texture scale in the Y dimension.  The X texture scale
        % value is unchanged.  See METHOD(set-texture-scale) for more information.
        textureScaleY = scaleSize.
        setTextureScale(realWorldPointer, textureScaleX, scaleSize).

+ to set-bitmap-image to bitmapImage (object):
        % Changes the bitmap of this object to bitmapImage, an instance of
        % class image.  If bitmapImage is NULL, bitmapping is turned off
        % for the object.

        if !bitmapImage: bitmap = -1.
        else bitmap = (bitmapImage get-texture-number).

        setBitmap(realWorldPointer, bitmap).

- to set-bitmap to textureNumber (int):
        % Deprecated.
```

```
            bitmap = textureNumber.
            setBitmap(realWorldPointer, textureNumber + 1).

      + to set-bitmap-heading to radianAngle (float):
            % If this object is in 2d bitmap mode, the rotation of the
            % bitmap will be set to radianAngle.

            setBitmapRotation(realWorldPointer, radianAngle).

      + to set-bitmap-heading-point towards-vector rotationVector (vector):
            % If this object is in 2d bitmap mode, the rotation of the
            % bitmap will be set to degreeAngle degrees.

            setBitmapRotationTowardsVector(realWorldPointer, rotationVector).

      - to set-bitmap-transparency to alphaValue (float):
            % Sets the transparency to alphaValue, a number between 0.0
            % (totally transparent) and 1.0 (fully opaque).

            setAlpha(realWorldPointer, alphaValue).

      + to set-transparency to alphaValue (float):
            % Sets the transparency of this object to alphaValue, a number
            % between 1.0 (totally opaque) and 0.0 (fully transparent).

            setAlpha(realWorldPointer, alphaValue).

      + to set-lightmap-image to lightmapImage (object):
            % Sets the object to be displayed using a "lightmap".  A
            % lightmap uses the texture specified and treats it like a light
            % source.  It's hard to explain.  Give it a try for yourself.
            % <p>
            % set-lightmap only has an effect on sphere shapes.  Other
            % shapes can be textured, but only spheres can be made into
            % lightmaps.

            if !lightmapImage: lightmap = -1.
            else lightmap = (lightmapImage get-texture-number).

            setLightmap(realWorldPointer, lightmap).

      - to set-lightmap to textureNumber (int):
            % Deprecated.

            lightmap = textureNumber.

            setLightmap(realWorldPointer, lightmap + 1).

+ section "Getting the Light Exposure of this Mobile Object"

      + to get-light-exposure:
            % When used in conjunction with light exposure detection
            % (OBJECTMETHOD(Control:enable-light-exposure-detection)), this
            % method returns the level of light exposure on this object.

            if realWorldPointer: return worldObjectGetLightExposure(realWorldPointer).
            else return 0.

      + section "Handling Collisions"

      + to handle-collisions with-type theType (string) with-method theMethod (string):
            % Adds a collision handler for this object.  When a collision
            % occurs between an instance of the this type and theType, the
            % breve engine will automatically call theMethod of the colliding
            % instance.

            push { theType, theMethod, 0 } onto collisionHandlerList.
            addCollisionHandler(self, theType, theMethod).

      + to ignore-collisions with-type theType (string):
            % Instructs the engine to ignore physical collisions with theType objects.
            % This does not affect collision callbacks specified with METHOD(handle-collisions).

            push { theType, 0, 1 } onto collisionHandlerList.
            setIgnoreCollisionsWith(self, theType, 1).

      + to dearchive:
            handler (list).

            foreach handler in collisionHandlerList: {
```

```
                        if handler{1}: addCollisionHandler(self, handler{0}, handler{1}).
                        if handler{2}: setIgnoreCollisionsWith(self, handler{0}, 1).
                }

                if texture > -1: self set-texture to texture.
                if lightmap > -1: self set-lightmap to lightmap.
                if bitmap > -1: self set-bitmap to bitmap.
                self set-texture-scale-x to textureScaleX.
                self set-texture-scale-y to textureScaleY.
                self set-neighborhood-size to neighborhoodSize.
                self set-color to color.

                return (super dearchive).

        - to delete:
                if realWorldPointer: removeObject(realWorldPointer).
                free menus.

  + section "Useful Funtions for Sensors"

        + to raytrace from-location theLocation (vector) with-direction theDirection (vector):
                % Computes the vector from theLocation towards theDirection
                % that hits the shape of this object.
                % <p>
                % If the object was not hit vector (0, 0, 0) will be returned.
                % <p>
                % The location and direction vector must be given relative to the world's
                % coordinate frame.

                return raytrace( realWorldPointer, theLocation, theDirection ).
}
```

```
The Stationary class

@include "Real.tz"
# @include "MenuItem.tz"

Real : Stationary (aka Stationaries) [version 2.4] {
    % Stationary objects are objects such as floors and walls that may
    % collide with other objects but will never move.  Stationary objects
    % require much less computation than OBJECT(Mobile) objects, so it is
    % always preferable to use a Stationary object when you know that an
       % object will not need to move during a simulation.
       % <P>
       % To setup a stationary object, you'll need to associate it with a
       % OBJECT(Shape) object using the method METHOD(register).

       # Notes for future implementation:
       # Beginning in breve 2.5, Stationary functionality has been enhanced
       # to allow repositioning at any time.  The distinction between Stationary
       # and OBJECT(Mobile) objects is that Stationary objects do not react
       # to physics and thus do not move <i>on their own</i>.  They can be
       # repositioned explicitly by simulation code at any time, but do not
       # have their own velocities or react to physics in any other way.

    + variables:
        color (vector).
               shadowCatcher (int).

        objectLocation (vector).
               shape (object).

  + section "Setting Up the Stationary Object"

    + to register with-shape theShape (object) at-location theLocation = (0, 0, 0) (vector) with-
rotation theRotation = [ ( 1, 0, 0 ), ( 0, 1, 0 ), ( 0, 0, 1 ) ] (matrix) :
        % Registers a stationary object using shape theShape at the location
        % specified by theLocation.

        if !(theShape is a "Shape"):
            die "method 'register' expects a Shape object ($theShape).".

        if !(theShape get-pointer):
            die "attempt to register Stationary object with uninitialized Shape.".

               shape = theShape.
               self add-dependency on shape.

        objectLocation = theLocation.
        realWorldPointer = addStationary( ( theShape get-pointer ), theLocation, theRotation ).
        self set-texture to 0.

               return self.

    + to get-location:
        % Returns the location of the Stationary object.

        return objectLocation.

    - to get-world-object:
        % Used internally to get the pointer to the world.  Do not use this method
        % in user simulations.

        return realWorldPointer.

  + section "Changing the Appearence of the Object"

    + to catch-shadows:
        % Informs this object that it should display shadows (and/or reflections)
        % of Mobile objects.  The shadows and reflections will always be shown on
        % the plane of the object pointing upwards on the Y axis--that is to say,
        % the plane with normal (0, 1, 0).  If the object does not have a plane
        % with normal (0, 1, 0), the shadows and reflections will not be displayed
        % correctly.  This method must be used in conjunction with the method
               % OBJECTMETHOD(Control:enable-shadows).
               % <P>
               % Before using this method, you should also refer to an improved shadowing
               % technique outlined in OBJECTMETHOD(Control.tz:enable-shadow-volumes).

        if !realWorldPointer:
            die "method 'catch-shadows' cannot be called before Stationary object is registered.".

               shadowCatcher = 1.
```

```
        setShadowCatcher(realWorldPointer, (0, 1, 0)).
        controller set-floor-defined.

    + to set-color to newColor (vector):
        % Sets the color of the Stationary object to newColor.  Textures
        % override color settings, so be sure to set the texture to -1
        % using the Real.tz method set-texture if you want a flat color
        % to be displayed--the texture is on by default for stationary
        % objects.
        % <p>
        % The vector elements of newColor are interpreted as red, green,
        % and blue values, on a scale from 0.0 to 1.0.
        % <p>
        % <b>The Stationary object must be registered in the world before
        % calling this method.  See METHOD(register).</b>

        if !realWorldPointer: die "method 'set-color' cannot be called before Stationary object is
registered.".

        color = newColor.
        setColor(realWorldPointer, color).

    + to get-color:
        % Returns the color of the Stationary object.

        return color.

    + to archive:
        return 1.

    + to dearchive:
        self register with-shape shape at-location objectLocation.
        if shadowCatcher: self catch-shadows.
        return 1.
}

Stationary : Floor {
       % A floor is a special case of the class OBJECT(Stationary).  It is a box
       % of size (1000, 5, 1000) with location (0, -2.5, 0), such that the ground
       % plane is placed at Y = 0.

       + to init:
              self register with-shape (new Cube init-with size (1000, 5, 1000)) at-location (0, -
2.5, 0).
                     # with-rotation [ ( .707, .707, 0 ), ( -.707, .707, 0 ), ( 0, 0, 1 ) ].
}
```

```
The Control class

@use Abstract.
@use MenuItem.
@use Shape.
@use Movie.
@use Image.
@use Camera.

Abstract : Control (aka Controls) [version 2.3] {
	% Summary: a parent class for the "controller" object required for all simulations.
	% <P>
	% The Control object sets up and controls simulations.  Every simulation
	% must have one Control subclass.  The user subclass of Control should
	% set up the simulation in the "init" method.
	% <p>
	% The Control class also acts as the main interaction between the user
	% and the breve environment itself.  It provides access to elements of
	% the user interface, for example, so that the user can add menus and
	% other interface features from inside simulations.
	% <p>
	% Because the breve engine is designed to run on a variety of systems,
	% there are varying levels of support for some of these features.  In
	% some cases, the features won't be supported at all on a system.  A
	% background daemon written to use the breve engine, for example, will
	% not place a dialog on the screen and wait for user input.

	+ variables:
		simTime (float).
		simStep (float).
		xRot (float).
		yRot (float).

		camTarget (vector).
		camOffset (vector).
		lightPosition (vector).

		watchObject (object).

		lightMenu (object).
		lightFlag (int).

		drawMenu (object).
		smoothFlag (int).

		backgroundColor (vector).
		fogColor (vector).
		lightAmbientColor, lightDiffuseColor (vector).

		backgroundTexture (int).

		fogIntensity (float).

		loadedImages (list).

		# we need to know when shadows and reflections are legal...

		floorDefined (int).

		shadowMenu (object).
		shadowFlag (int).

		shadowVolumeFlag (int).

		reflectMenu (object).
		reflectFlag (int).

		fogMenu (object).
		fogFlag (int).

		blurMenu (object).
		blurFlag (int).
		blurFactor (float).

		selectedObject (object).

		deltaOffset (vector).
		deltaTarget (vector).

		offsetting (int).
		frozen (int).
```

```
		drawEveryFrame (int).

		genericShape (object).
		genericLinkShape (object).

		movie (object).
		movieMenu (object).

		cameraPointer (pointer).
		camera (object).

	+ to init:
		% Initializes the Control object by setting up default values for
		% variables such as the size of the integration timestep.  Subclasses
		% of Control may override these defaults in their own init functions.

		cameraPointer = getMainCameraPointer().

		camera = new Camera.
		camera set-camera-pointer to cameraPointer.

		floorDefined = 0.

		self set-integration-step to .005.
		self set-iteration-step to .05.
		self enable-draw-every-frame.

		backgroundTexture = -1.

		self set-background-color to (.5, .7, 1.0).
		self set-fog-color to (.8, .8, .8).

		xRot = 0.0.
		yRot = 0.0.

		self move-light to (0, 0, 0).
		self point-camera at (0, 0, 0) from (0, 0, 30).

		self set-background-scroll-rate x 0.001 y 0.0001.

		lightMenu = (self add-menu named "Use Lighting" for-method "toggle-lighting").
		drawMenu = (self add-menu named "Use Smooth Shading" for-method "toggle-smooth").
		shadowMenu = (self add-menu named "Draw Shadows" for-method "toggle-shadows").
		fogMenu = (self add-menu named "Draw Fog" for-method "toggle-fog").
		reflectMenu = (self add-menu named "Draw Reflections" for-method "toggle-
reflections").
		blurMenu = (self add-menu named "Use Motion Blur" for-method "toggle-blur").

		self add-menu-separator.
		movieMenu = (self add-menu named "Record Movie to \"simulation.mpeg\"" for-method
"toggle-recording-to-movie").
		(self add-menu named "Save Snapshot to \"simulation.png\"" for-method "save-
snapshot-to-file").

		self enable-smooth-drawing.

		# lighting is off by default, but we'll call the method anyway
		# to sync up the menus (shadow & reflect menus should be disabled).

		self disable-lighting.

		self add-menu-separator.

		genericShape = (new Sphere init-with radius 1.0).
		genericLinkShape = (new Cube init-with size (.1, 1, .1)).

	- to toggle-recording-to-movie:
		if movie: {
			movieMenu uncheck.
			movie close.
			free movie.
			return.
		}

		movie = new Movie.
		movie record to "simulation.mpeg".
		movieMenu check.

	- to save-snapshot-to-file:
		self save-snapshot to "simulation.png".
```

```
      - to parse-xml-network-request from-string s (string):
            return dearchiveXMLObjectFromString(s).

      - to get-generic-shape:
            % Returns a "generic" shape for agents, a sphere of radius 1.0.

            return genericShape.

      - to get-generic-link-shape:
            % Returns a "generic" shape for links, a cube with size (.1, 1, .1).

            return genericLinkShape.

+ section "Accepting network uploads"

      + to accept-upload of uploadedObject (object) from host (string):
            % This method is automatically called when an object is uploaded
            % through a OBJECT(NetworkServer).  This implementation simply
            % prints out a message saying that the object has been received,
            % but your controller may override the method to take other actions.

            print "object $uploadedObject sent from host $host".

+ section "Saving snapshots of simulations"

      + to save-snapshot to filename (string):
            % Takes a PNG snapshot of the current simulation display and saves
            % it to a file named filename.  filename should end with ".png".

            snapshot(filename).

+ section "Updating neighbors"

      + to update-neighbors:
            % The neighborhood for each object in the simulation is the set of
            % other objects within a specified radius--calling this method will
            % update the neighborhood list for each object.  This method is
            % only useful in conjunction with methods in OBJECT(Real) which
            % set the neighborhood size and then later retrieve the neighbor
            % list.

            updateNeighbors().

+ section "Getting simulation time"

      + to get-real-time:
            % Returns the number of seconds since January 1st, 1970 with microsecond
            % precision.

            return getRealTime().

      + to get-time:
            % Returns the simulation time of the world.

            return getTime().

+ section "Pausing and stopping simulations"

      + to end-simulation:
            % Ends the simulation gracefully.

            endSimulation().

      + to pause:
            % Pauses the simulation as though the user had done so through
            % the user interface.  This method is not supported on all
            % breve client interfaces.

            pauseSimulation().

      + to unpause:
            % Pauses the simulation as though the user had done so through
            % the user interface.  This method is not supported on all
            % breve client interfaces.

            unpauseSimulation().

      + to sleep for-seconds s (float):
            % Pauses execution for s seconds.  s does not have to be a whole
            % number.  Sleeping for a fraction of a second at each iteration
```

```
            % (inside your controller's iterate method) can effectively slow
            % down a simulation which is too fast to observe.

            sleep(s).

+ section "Setting and getting the iteration and integration time steps"

    + to set-integration-step to timeStep (float):
            % Sets the integration stepsize to timeStep.  The integration stepsize
            % determines how quickly the simulation runs: large values (perhaps
            % as high as 1 second) mean that the simulation runs quickly at the
            % cost of accuracy, while low values mean more accuracy, but slower
            % simulations.
            % <p>
            % The control object and its subclasses set the integration timeStep
            % to reasonable values, so this method should only be invoked by
            % expert users with a firm grasp of how they want their simulation
            % to run.
            % <p>
            % Additionally, this value is only used as a suggestion--the
            % integrator itself may choose to adjust the integration stepsize
            % according to the accuracy of previous timesteps.

            simStep = timeStep.

    + to get-integration-step:
            % Returns the current integration step size.

            return simStep.

    + to set-iteration-step to timeStep (float):
            % Sets the iteration stepsize to timeStep.  The iteration stepsize
            % is simply the number of simulated seconds run between calling the
            % controller object's "iterate" method. <b>This value may not be
            % smaller than the integration timestep</b>.
            % <p>
            % The control object and its subclasses set the iteration stepsize
            % to reasonable values, so this method should only be invoked by
            % expert users with a firm grasp of how they want their simulation
            % to run.  Small values slow down the simulation considerably.
            % <p>
            % For physical simulations, the iteration stepsize should be
            % considerably larger than the integration stepsize.  The iteration
            % stepsize, in this case, can be interpreted as the reaction time
            % of the agents in the world to changes in their environment.

            simTime = timeStep.

            if simStep > simTime: simTime = simStep.

    + to get-iteration-step:
            % Returns the current iteration step size.

            return simTime.

+ section "Setting the Random Seed"

    + to set-random-seed to newSeed (int):
            % Sets the random seed to newSeed.  Setting the random seed determines
            % the way random numbers are chosen.  Two runs which use the same
            % random seed will yield the same exact random numbers.  Thus, by
            % setting the random number seed manually, you can make simulations
            % repedible.

            randomSeed(newSeed).

    + to set-random-seed-from-dev-random:
            % Sets the random seed to a value read from /dev/random (if available).
            % <P>
            % By default, breve sets the random seed based on the current time.
            % This is generally sufficient for most simulations.  However, if
            % you are dealing with a setup in which multiple simulations might
            % be launched simultaneously (such as a cluster setup), then you may
            % have a situation in which the same random seed would be used for
            % multiple runs, and this will make you unhappy.  Using this method
            % will restore happiness and harmony to your life.

            randomSeedFromDevRandom().

+ section "Creating a New Instance From a Classname String"
```

```
        + to make-new-instance of-class className (string):
                % Returns a new instance of the class className.

                return newInstanceForClassString(className).

    + section "Getting Command-Line Arguments"

        + to get-argument-count:
                % If this instance of breve was run from the command line, this method
                % returns the number of arguments passed to the program.  <b>The first
                % argument is always the name of the simulation file</b>.  Use this method
                % in conjunction with METHOD(get-argument).

                return getArgc().

        + to get-argument at-index theIndex (int):
                % If this instance of breve was run from the command line, this method
                % returns the argument at index theIndex.  The argument is always returned
                % as a string, though this may naturally be converted to other types
                % depending on the context.  The arguments (like arrays and lists in <i>steve</i>)
                % are zero based, meaning that the first element has index 0.  <b>The first
                % argument (the one at index 0) is always the name of the simulation
                % file</b>.  Use this method in conjunction with
                % METHOD(get-argument-count).  Make sure you check the number of arguments
                % available before calling this method--requesting an out-of-bounds argument
                % will cause a fatal error in the simulation.

                return getArgv(theIndex).

    + section "Selecting Objects"

        - to get-drag-object:
                if (selectedObject && ((selectedObject is a "Mobile") || (selectedObject is a
"MultiBody"))): return selectedObject.
                return 0.

        + to click on theObject (object):
                % Called automatically when the user clicks on an theObject from
                % the graphical display window of the simulation.  The default
                % behavior of this method is to select the object that was clicked
                % and execute its "click" method, if it exists.
                % <p>
                % If you do not wish to allow users to select objects in your
                % simulation, you should implement your own click method in your
                % controller object.
                % <p>
                % If you wish to implement your own click method, but still want to
                % maintain the default behavior of this method, make sure you
                % call "super click on theObject" from your method.
                % <p>
                % <b>theObject may be NULL</b>--an uninitialized object.  This means
                % that a click occurred, but no object was selected, i.e., a deselection
                % event.  You should test theObject before calling any of its methods.

                if selectedObject == theObject: return.

                if selectedObject: {
                        selectedObject hide-bounding-box.
                        selectedObject hide-axis.
                }

                selectedObject = theObject.

                if selectedObject && (selectedObject is a "Link") && (selectedObject get-multibody):
                        selectedObject = (selectedObject get-multibody).

                if !selectedObject ||
                        !(selectedObject can-respond to "show-bounding-box") ||
                        !(selectedObject can-respond to "show-axis") : return.

                selectedObject show-bounding-box.
                selectedObject show-axis.

        - to catch-key-0x7F-down:
                % This method is automatically called when the delete key is pressed
                % down to delete the selected object.  It deletes the selected instance.
                % Do not call this method manually--it would work, but it would be
                % a bit roundabout.

                if selectedObject: selectedObject delete-instance.
```

```
    + to get-selection:
        % Returns the "selected" object--the object which
        % has been clicked on in the simulation.

        if selectedObject && (selectedObject is a "Link") && (selectedObject get-multibody):
            return (selectedObject get-multibody).

        return selectedObject.

+ section "Background Appearance Options"

    + to set-background-scroll-rate x xValue (float) y yValue (float):
        % Sets the rate of the background image scrolling.  Purely cosmetic.

        setBackgroundScroll(xValue, yValue).

    + to set-background-color to newColor (vector):
        % Sets the background color of the rendered world to newColor.

        backgroundColor = newColor.
        setBackgroundColor(newColor).

    + to set-background-texture-image to newTextureImage (object):
        % Sets the background color of the rendered world to newTexture.
        % newTexture must be a texture returned by the method
        % METHOD(load-image) or 0 for the default texture.  Setting
        % the texture to -1 will turn off background texturing.

        backgroundTexture = (newTextureImage get-texture-number).
        setBackgroundTexture(backgroundTexture).

    - to set-background-texture to newTexture (int):
        backgroundTexture = newTexture + 1.
        setBackgroundTexture(newTexture + 1).

+ section "Camera & Lighting Options"

    + to get-main-camera:
        % Returns the OBJECT(Camera) object corresponding to the main camera.
        % This allows you to directly control camera options.

        return camera.

    + to watch item theObject (object):
        % Points the camera at the OBJECT(Mobile) object theObject.  If
        % theObject is passed in as 0, then the camera will stop watching
        % a previously watched object.

        watchObject = theObject.

    - to aim-camera at location (vector):
        % Depricated.

        self set-camera-target to location.

    + to set-camera-target to location (vector):
        % Aims the camera at location.  The offset of the camera (the offset
        % from the existing target) stays the same.

        cameraSetTarget(location).

    - to offset-camera by amount (vector):
        % Depricated.

        self set-camera-offset to amount.

    + to set-camera-offset to offset (vector):
        % Offsets the camera from the target by amount.  The target of
        % the camera remains the same.

        cameraSetOffset(offset).

    + to get-camera-offset:
        % Returns the current offset from of the camera from its target.
        % Note that the camera offset can be changed manually by the user,
        % so it may be wise to use this function in conjunction with camera
        % movements to ensure consistency.

        return cameraGetOffset().

    + to get-camera-target:
```

```
		% Returns the current target of the camera.  Note that the camera
		% target can be changed manually by the user, so it may be wise
		% to use this function in conjunction with camera movements to
		% ensure consistency.

		return cameraGetTarget().

	+ to bullet-pan-camera-offset by amount (vector) steps stepCount (int):
		% Sets the camera in motion to smoothly change the camera offset
		% over stepCount iteration steps, with the physical simulation
		% frozen in the meantime.

		frozen = 1.

		self pan-camera-offset by amount steps stepCount.

	+ to pan-camera-target to newTarget (vector) steps stepCount (int):
		% Sets the camera in motion to smoothly change the camera target
		% over stepCount iteration steps.

		if stepCount < 2: {
			self set-camera-target to newTarget.
			return.
		}

		deltaTarget = (newTarget - (self get-camera-target)) / stepCount.
		offsetting = stepCount + 1.

	+ to pan-camera-offset by amount (vector) steps stepCount (int):
		% Sets the camera in motion to smoothly change the camera offset
		% over stepCount iteration steps.

		if stepCount < 2: {
			self offset-camera by amount.
			return.
		}

		deltaOffset = (amount - (self get-camera-offset)) / stepCount.
		offsetting = stepCount + 1.

	+ to point-camera at location (vector) from offset = (0, 0, 0) (vector):
		% Points the camera at the vector location.  The optional
		% argument offset specifies the offset of the camera relative
		% to the location target.

		camTarget = location.
		self set-camera-target to location.

		if | offset | != 0.0: {
			camOffset = offset.
			self set-camera-offset to offset.
		}

	+ to pivot-camera x dx (float) y dy (float):
		% Rotates the camera (from it's current position) on the x-axis by dx and on the y-
axis by dy.

		rot (vector).

		rot = (camera get-rotation).
		rot::x += dx.
		rot::y += dy.
		camera set-rotation x rot::x y rot::y.

	+ to set-camera-rotation x rx (float) y ry (float):
		% Sets the camera rotation on the x-axis to rx and the y-axis to ry.
		% This method sets the rotation without regard for the current rotation.
		% If you want to offset the camera rotation from the current position,
		% use the method METHOD(pivot-camera) instead.

		camera set-rotation x rx y ry.

	+ to zoom-camera to theDistance (float):
		% Zooms the camera to theDistance away from the current target--
		% whether the target is a vector or object.

		cameraSetZoom(theDistance).

	+ to move-light to theLocation (vector):
		% Moves the source light to theLocation.  The default position is
		% (0, 0, 0) which is the origin of the world.
```

```
            lightPosition = theLocation.
            setLightPosition(theLocation).

      + to iterate:
            result (float).

            if watchObject: self set-camera-target to (watchObject get-location).

            # we might be doing a "bullet-time" pan

            if !frozen: result = worldStep(simTime, simStep).

            if result == -1: die "An error occurred during world simulation.".

            # if we're in the middle of a pan, continue

            if offsetting: {
                  self set-camera-offset to ((self get-camera-offset) + deltaOffset).
                  self set-camera-target to ((self get-camera-target) + deltaTarget).
                  offsetting--.

                  # if we are done the pan, let the simulation roll again.

                  if offsetting == 0: frozen = 0.
            }

+ section "Rendering Options"

      - to set-floor-defined:
            if lightFlag: {
                  shadowMenu enable.
                  reflectMenu enable.
            }

            floorDefined = 1.

      + to enable-draw-every-frame:
            % If the method METHOD(disable-draw-every-frame) has been called previously,
            % this method will resort to the default behavior, namely that the rendering
            % engine will try to render an image for each and every iteration of the breve
            % engine.

            drawEveryFrame = 1.
            setDrawEveryFrame(1).

      + to disable-draw-every-frame:
            % Allows the rendering engine to drop frames if the simulation is moving
            % faster than the display.  This can lead to faster simulations with choppier
            % displays.  Not all breve development environments support this option.  The
            % Mac OS X application does, as do all threaded command-line breve programs.
            % <p>
            % There is rarely any benefit from using this method, except in instances where
            % the drawing of a scene is complex, and the computation is simple.  The included
            % DLA.tz demo is an example of one such simulation which benefits immensely from
            % this feature.

            drawEveryFrame = 0.
            setDrawEveryFrame(0).

      + to set-z-clip to theDistance (int):
            % Sets the Z clipping plan to theDistance.  The Z clipping plan
            % determines how far the camera can see.  A short Z clipping distance
            % means that objects far away will not be drawn.
            % <p>
            % The default value is 200.0 and this works well for most simulations,
            % so there is often no need to use this method.
            % <p>
            % Using a short Z clipping distance improves drawing quality, avoids
            % unnecessary rendering and can speed up drawing during the simulation.
            % However, it may also cause objects you would like to observe in the
            % simulation to not be drawn because they are too far away.

            setZClip(theDistance).

+ section "Special Effects & Drawing Options"

      + to enable-outline:
            % Enables outline drawing.  Outline drawing is a wireframe black and white
            % draw style.  Reflections and textures are ignored when outlining is
            % enabled.  Outlining is useful for producing diagram-like images.  It
```

```
        % looks cool.

        setDrawOutline(1).

+ to disable-outline:
        % Disables outline drawing.

        setDrawOutline(0).

+ to toggle-lighting:
        % toggle lighting for the main camera

        if lightFlag == 1: self disable-lighting.
        else self enable-lighting.

+ to enable-lighting:
        % enable lighting for the main camera

        lightFlag = 1.
        setDrawLights(lightFlag).
        lightMenu check.

        if floorDefined: {
                shadowMenu enable.
                reflectMenu enable.
        }

+ to disable-lighting:
        % disable lighting for the main camera

        lightFlag = 0.
        setDrawLights(lightFlag).
        lightMenu uncheck.

        shadowMenu disable.
        reflectMenu disable.

+ to toggle-smooth:
        % Toggle smooth drawing for the main camera.  See METHOD(enable-smooth-drawing) and
        % METHOD(disable-smooth-drawing) for more information.

        if smoothFlag == 1: self disable-smooth-drawing.
        else self enable-smooth-drawing.

+ to enable-smooth-drawing:
        % Enable smooth drawing for the main camera.  Smooth drawing enables
        % a smoother blending of colors, textures and lighting.  This feature
        % is especially noticeable when dealing with spheres or large objects.
        % <p>
        % It is strongly recommended that smooth drawing be enabled whenever
        % lighting is enabled (see METHOD(enable-lighting)).  Otherwise,
        % major artifacts may be visible, especially on larger polygons.
        % <p>
        % The disadvantage of smooth drawing is a potential performance hit.
        % The degree of this performance hit depends on the number of polygons
        % in the scene.  If speed is an issue, it is often best to disable
        % both lighting and smooth drawing.

        smoothFlag = 1.
        cameraSetDrawSmooth(cameraPointer, smoothFlag).
        drawMenu check.

+ to disable-smooth-drawing:
        % Disable smooth drawing for the main camera.  See METHOD(enable-smooth-drawing)
        % for more information.

        smoothFlag = 0.
        cameraSetDrawSmooth(cameraPointer, smoothFlag).
        drawMenu uncheck.

+ to toggle-shadows:
        % Toggle shadows for the main camera.  See METHOD(enable-shadows) and
        % METHOD(disable-shadows) for more information on shadows.

        if shadowFlag == 1: self disable-shadows.
        else self enable-shadows.

+ to enable-shadows:
        % Enable shadows for the main camera.  Shadows use the current
        % position of the light in order to render shadows onto a flat
        % plane in the world.  Because of the complexity of drawing shadows,
```

```
        % they can only be drawn onto a single plane of a OBJECT(Stationary)
        % object--see the method catch-shadows of OBJECT(Stationary) for more
        % information.  The shadow-catching object must already be defined in
        % order for this method to take effect.
        % <P>
        % For an improved shadowing algorithm, see METHOD(enable-shadow-volumes).

        if !floorDefined: return.

        shadowFlag = 1.
        setDrawShadow(shadowFlag).
        shadowMenu check.

+ to disable-shadows:
        % Disable "flat" shadows for the main camera.  See
        % METHOD(enable-shadows) for more information on shadows.

        shadowFlag = 0.
        setDrawShadow(shadowFlag).
        shadowMenu uncheck.

+ to enable-shadow-volumes:
        % Enables shadows drawn using a "shadow volume" algorithm.  This
        % is an alternative to the shadows rendered using METHOD(enable-shadows).
        % <p>
        % Shadow volumes allow all objects in the simulation to shadow one-another,
        % as opposed to having objects only shadow a single plane (as is the case
        % with the METHOD(enable-shadows) algorithm).  Shadow volumes are in fact
        % superior in every way but one: shadow volumes will not generate accurate
        % shadows of bitmapped objects the way the original algorithm will.
        % If you want high-quality bitmap shadows in your simulation, use
        % METHOD(enable-shadows), otherwise, shadow volumes are likely the better
        % choice.

        shadowVolumeFlag = 1.
        setDrawShadowVolumes(shadowVolumeFlag).

+ to disable-shadow-volumes:
        % Disable shadow volumes for the main camera.  See
        % METHOD(enable-shadow-volumes) for more information on shadows.

        shadowVolumeFlag = 0.
        setDrawShadowVolumes(shadowFlag).

+ to toggle-reflections:
        % Toggle reflections for the main camera.  See METHOD(enable-reflections) for
        % more information on reflections.

        if !floorDefined: return.

        shadowFlag = 1.

        if reflectFlag == 1: self disable-reflections.
        else self enable-reflections.

+ to enable-reflections:
        % Enable reflections for the main camera.  Reflections are used to draw a mirror
        % image of objects in the world onto a single plane.  Because of
        % the complexity of drawing reflections, they can only be drawn onto a single plane
        % of a OBJECT(Stationary) object--see the method catch-shadows of OBJECT(Stationary)
        % for more information.  The reflection-catching object must already be defined in
        % order for this method to take effect.

        if !floorDefined: return.

        reflectFlag = 1.
        setDrawReflection(reflectFlag).
        reflectMenu check.

+ to disable-reflections:
        % Disable reflections for the main camera.  See METHOD(enable-reflections for
        % more information on reflections.

        reflectFlag = 0.
        setDrawReflection(reflectFlag).
        reflectMenu uncheck.

+ to disable-text:
        % Disable the timestamp and camera position texts (which appear when
        % changing the camera angle or position).  The text is on by default.
        % The text can be re-enabled using METHOD(enable-text).
```

```
        setDrawText(0).

+ to enable-text:
        % enables the timestamp and camera position texts (which appear when changing
        % the camera angle or position).  This is the default setting.  The text can
        % be disabled using METHOD(disable-text).

        setDrawText(1).

+ to toggle-blur:
        % Toggle motion blur for the main camera.  See METHOD(enable-blur) for
        % more information on reflections.

        if blurFlag == 1: self disable-blur.
        else self enable-blur.

+ to enable-blur:
        % Enables blur.  Blurring simply draws a frame without
        % totally erasing the previous frame.

        blurFlag = 1.
        cameraSetBlur(cameraPointer, 1).
        blurMenu check.

+ to disable-blur:
        % Disables blurring.  See METHOD(enable-blur) for more
        % information.

        blurFlag = 0.
        cameraSetBlur(cameraPointer, 0).
        blurMenu uncheck.

+ to set-blur-factor to factor (float):
        % Sets the blur level to factor.  Factor should be a value
        % between 1.0, which corresponds to the highest blur level,
        % and 0.0, which corresponds to the lowest blur level.
        % <p>Blur must first be enabled using the method
        % METHOD(enable-blur).  Note % that a blur level of 0.0 is
        % still a minor blur--to disable % blur completely, use the
        % method METHOD(disable-blur).

        blurFactor = factor.
        cameraSetBlurFactor(cameraPointer, factor).

+ to clear-screen:
        % Clears the camera to the current background color.  This
        % method clears blurred artifacts which are drawn after
        % enabling METHOD(enable-blur).  If blurring has not been
        % enabled, this method has no visual effect.

        cameraClear(cameraPointer).

+ to set-fog-limits with-start fogStart (float) with-end fogEnd (float):
        % The calculation which calculates fog needs to know where the fog
        % starts (the point at which the fog appears) and where the fog ends
        % (the point at which the fog has reached it's highest intensity).
        % <p>
        % This method sets the start value to fogStart and the end value to
        % fogEnd.  fogStart must be greater than or equal to zero.  fogEnd
        % must be greater than fogStart.
        % <p>
        % Fog must first be turned on with METHOD(enable-lighting) before fog
        % is displayed.

        setFogDistances(fogStart, fogEnd).

+ to set-fog-intensity to newIntensity (float):
        % Sets the fog intensity to newIntensity.  Fog must first be turned on with
        % METHOD(enable-lighting) before fog is displayed.

        fogIntensity = newIntensity.
        cameraSetFogIntensity(cameraPointer, newIntensity).

+ to set-fog-color to newColor (vector):
        % Sets the fog color to newColor.  Fog must first be turned on with
        % METHOD(enable-lighting) before fog is displayed.

        fogColor = newColor.
        cameraSetFogColor(cameraPointer, newColor).
```

```
	+ to enable-fog:
		% Enables fog for the main camera.  This adds the visual effect of fog to the
		% world.  Fog parameters can be set using methods METHOD(set-fog-color),
		% METHOD(set-fog-intensity) and METHOD(set-fog-limits).
		% <p>
		% Fog and lightmap effects don't mix.

		fogFlag = 1.
		cameraSetDrawFog(cameraPointer, 1).
		fogMenu check.

	+ to disable-fog:
		% Disables fog for the main camera.  See METHOD(enable-fog) for more information.
		fogFlag = 0.
		cameraSetDrawFog(cameraPointer, 0).
		fogMenu uncheck.

	+ to toggle-fog:
		% Toggle fog for the main camera

		if fogFlag == 1: self disable-fog.
		else self enable-fog.

	+ to set-display-text number messageNumber = 0 (int) to theString (string) at-x xLoc = -.95
(float) at-y yLoc = -.95 (float):
		% Sets a text string in the simulation display.  xLoc and yLoc
		% represent the location of the text.  The coordinate system
		% used goes from (-1, -1) to (1, 1) with (-1, -1) at the lower
		% left hand corner, (0, 0) in the center of the window and
		% (1, 1) in the top right hand corner.
		% <p>
		% The optional argument messageNumber may be used to specify
		% up to 8 different messages.

		cameraSetText(theString, messageNumber, xLoc, yLoc, (0, 0, 0)).

	+ to set-display-text-scale to scale (double):
		% Sets the scaling factor for text in the display window.
		% See METHOD(set-display-text) and METHOD(set-display-message)
		% for more information on adding text messages to the display
		% window.

		cameraSetTextScale(scale).

	- to set-display-message to theString (string) with-number messageNumber (int) at-x xLoc
(float) at-y yLoc (float) with-color textColor = (0, 0, 0) (vector):
		% Sets a text string in the simulation display.  xLoc and yLoc
		% represent the location of the text.  The coordinate system used
		% goes from (-1, -1) to (1, 1) with (-1, -1) at the lower left hand
		% corner, (0, 0) in the center of the window and (1, 1) in
		% the top right hand corner.  The color of the text is set
		% to textColor.
		% <p>
		% Up to 8 messages can be displayed in the simulation window.
		% messageNumber specifies which message "slot" to modify.
		% Subsequent calls to this method with the same slot number
		% erase previous entries.

		cameraSetText(theString, messageNumber, xLoc, yLoc, textColor).

  + section "Interacting with the User Interface"

	+ to get-mouse-x-coordinate:
		% Returns the X-coordinate of the mouse relative to the simulation window.
		% The value may be negative if the mouse is to the left of the simulation
		% view.
		% See also METHOD(get-mouse-y-coordinate).

		return getMouseX().

	+ to get-mouse-y-coordinate:
		% Returns the Y-coordinate of the mouse.
		% The value may be negative if the mouse is outside of the simulation
		% view, towards the bottom of the screen.
		% See also METHOD(get-mouse-x-coordinate).

		return getMouseY().

	+ to add-menu named menuName (string) for-method theMethod (string):
		% Adds a menu named menuName to the application which will result
		% in a call to theMethod for the calling instance.
```

```
		% <p>
		% If the calling instance is the Controller object, then the menu will
		% become the "main" simulation menu.  Otherwise, the menu will become
		% a contextual menu associated with the specific object in the simulation.
		% <p>
		% Note that unlike the METHOD(handle-collision) which sets the collision
		% handler for the whole type (class, that is), this method affects only
		% the instance for which it is called, meaning that each instance of a
		% certain class may have a different menu.

		return ((new MenuItem) create-menu named menuName for-object self for-method
theMethod).

	+ to add-menu-separator:
		% Adds a separator menu item--really just an empty menu item.

		return ((new MenuItem) create-menu named "" for-object self for-method "").

	+ to show-dialog with-title title (string) with-message message (string) with-yes-button
yesString (string) with-no-button noString (string):
		% Shows a dialog box (if supported by the current breve environment)
		% and waits for the user to click on one of the buttons.
		% <p>
		% If the "yes button" is clicked on, the method returns 1--if the
		% "no button" is clicked, or if the feature is not supported, 0
		% is returned.

		return dialogBox(title, message, yesString, noString).

	+ to beep:
		% Plays the system beep sound, if supported by the implementation.

		playSound().

	- to load-image from file (string):
		% Loads an image from a file, returning an OBJECT(Image) object.
		% <P>
		% This method is provided for backwards compatability only.
		% The p
		% <p>

		image (object).

		print "warning: the Control method \"load-image\" is now deprecated!".

		image = new Image.

		if (image load from file): {
			loadedImages{ loadedImages } = file.
		} else {
			free image.
			return -1.
		}

		return (image get-texture-number) - 1.

	- to load-image-without-alpha from file (string):
		% Deprecated.

		return (self load-image from file).

	+ to execute command systemCommand (string):
		% Executes the shell command systemCommand using /bin/sh.
		% Returns the output of command.  Supported on UNIX-based
		% implementations only (Mac OS X and Linux included), not
		% supported on Windows.

		return system(systemCommand).

	+ to set-light-color to newColor (vector):
		% Sets the color of the light to newColor.  Only has an effect on the
		% rendering when lighting has been turned on using
		% METHOD(enable-lighting).
		% <p>
		% This method sets both the ambient and diffuse colors, which can also
		% be set individually with METHOD(set-light-ambient-color) and
		% METHOD(set-light-diffuse-color).

		self set-light-ambient-color to newColor.
		self set-light-diffuse-color to newColor.
```

```
	+ to set-light-ambient-color to newColor (vector):
		% Sets the ambient, or background, color of the light to newColor.
		% Only has an effect on the rendering when lighting has been turned
		% on using METHOD(enable-lighting).

		lightAmbientColor = newColor.
		setLightAmbientColor(newColor).

	+ to set-light-diffuse-color to newColor (vector):
		% Sets the diffuse, or foreground, color of the light to newColor.
		% Only has an effect on the rendering when lighting has been turned
		% on using METHOD(enable-lighting).
		% <p>
		% The diffuse color is the color coming directly from the light,
		% as opposed to the "ambient" light that is also generated.

		lightDiffuseColor = newColor.
		setLightDiffuseColor(newColor).

	+ to set-interface-item with-id tag (int) to-string newValue (string):
		% This method will set the interface item tag to newValue.  This
		% is for simulations which have an OS X nib file associated with
		% them.

		return setInterfaceString(newValue, tag).

+ section "Manipulating and Using Colors"

	+ to unique-color for-number n (int):
		% Returns a unique color for each different value of n up
		% to 198.  These colors are allocated according to an
		% algorithm which attempts to give distinguishable colors,
		% though this is subjective and not always possible.

		return uniqueColor(n).

	+ to get-hsv-color for-rgb-color rgbColor (vector):
		% All colors in breve expect colors in the RGB format--a vector where
		% the 3 elements represent red, green and blue intensity on a scale
		% from 0.0 to 1.0.
		% <p>
		% This method returns the HSV color vector for a given vector rgbColor
		% in RGB color format.

		return RGBtoHSV(rgbColor).

	+ to get-rgb-color for-hsv-color hsvColor (vector):
		% All colors in breve expect colors in the RGB format--a vector where
		% the 3 elements represent red, green and blue intensity on a scale
		% from 0.0 to 1.0.
		% <p>
		% This method returns the RGB color vector for a given vector hsvColor
		% in HSV color format.

		return HSVtoRGB(hsvColor).

+ section "Archiving & Dearchiving"

	+ to archive:
		camTarget = cameraGetTarget().
		camOffset = cameraGetOffset().
		return (super archive).

	+ to dearchive:
		image (string).
		images (list).

		cameraPointer = getMainCameraPointer().
		camera set-camera-pointer to cameraPointer.

		foreach image in loadedImages: images{ images } = image.
		foreach image in images: (self load-image from image).

		self move-light to lightPosition.
		self set-background-texture to backgroundTexture.
		self set-background-color to backgroundColor.
		self set-fog-color to fogColor.
		self point-camera at camTarget from camOffset.
		self set-blur-factor to blurFactor.

		if lightFlag == 1: self enable-lighting.
```

```
            if shadowFlag == 1: self enable-shadows.
            if shadowVolumeFlag == 1: self enable-shadow-volumes.
            if fogFlag == 1: self enable-fog.
            if reflectFlag == 1: self enable-reflections.
            if smoothFlag == 1: self enable-smooth-drawing.
            if blurFlag == 1: self enable-blur.

            if drawEveryFrame: (self enable-draw-every-frame).
            else (self disable-draw-every-frame).

            return 1.

      + to save-as-xml file filename (string):
            % Writes the entire state of the world to an XML file, filename.
            % filename should have one of the following extensions: .xml,
            % .brevexml, .tzxml.
            % <p>
            % After saving the state of the world as an XML file, you can later
            % start a new run of the same simulation from the saved state.  You
            % will still need the original steve code which generated the file
            % in order to restart the simulation.

            writeXMLEngine(filename).

      + to dearchive-xml file filename (string):
            % Asks the controller to dearchive an object from an XML file.  The
            % XML file must have been created using
            % OBJECTMETHOD(Object:archive-as-xml).

            return dearchiveXMLObject(filename).

+ section "Detecting Light Exposure"

      + to enable-light-exposure-detection:
            % <B>Experimental</B>
            % <P>
            % Light exposure detection will attempt to tell you how much "sunlight"
            % is reaching each object in your simulation.  You can set the location
            % of the light source with METHOD(set-light-exposure-source).  Then,
            % use the method get-light-exposure (in OBJECT(Stationary), OBJECT(Mobile),
            % and OBJECT(Link)) in order to find out how much light was detected for
            % individual objects.
            % <P>
            % The direction of the sunlight is hardcoded towards the world point
            % (0, 0, 0), and only spreads out to fill an angle of 90 degrees.
            % These limitations may be removed in the future if needed.

            setDetectLightExposure(1).

      + to disable-light-exposure-detection:
            % Disables light exposure detection.  See METHOD(enable-light-exposure-detection).

            setDetectLightExposure(0).

      + to set-light-exposure-source to source (vector):
            % Changes the light source for calculating exposure.  See
            % METHOD(enable-light-exposure-detection).

            setLightExposureSource(source).

      + to enable-light-exposure-drawing:
            % Enables drawing of the light exposure buffer to the screen.
            setDrawLightExposure( 1 ).

      + to disable-light-exposure-drawing:
            % Disables drawing of the light exposure buffer to the screen.
            setDrawLightExposure( 0 ).

      + to get-light-exposure-camera:
            % Returns a camera that can be used to control the light detection
            % light-source.
            camera = new Camera.
            camera set-camera-pointer to getLightExposureCamera().

+ section "Debugging & Performance"

      + to set-output-filter to filterLevel (int):
            % Sets the output filter level.  This value determines the level of
            % detail used in printing simulation engine errors and messages.
            % The default value, 0, prints only regular output.  An output filter
            % of 50 will print out all normal output as well as some warnings and
```

```
                % other information useful mostly to breve developers.  Other values
                % may be added in the future to allow for more granularity of error
                % detail.

                setOutputFilter(filterLevel).

        + to enable-freed-instance-protection:
                % Freed instance protection means that the breve engine retains
                % instances which have been freed in order to make sure that they
                % are not being incorrectly accessed.  This has a small memory
                % cost associated with each freed object.
                % <p>
                % Freed instance protection is enabled by default, so you'll only
                % need to call this method if it has been disabled using
                % METHOD(disable-freed-instance-protection).

                setFreedInstanceProtection(1).

        + to disable-freed-instance-protection:
                % Disabling freed instance protection means that the breve engine will
                % not attempt to keep track of freed objects, and will yield better
                % memory performance when large numbers of objects are being created
                % and destroyed.
                % <p>
                % The downside is that improper access of freed instances may cause
                % crashes or unexpected behavior when freed instance protection is
                % disabled.  Simulations should thus always use freed instance
                % protection during development and testing, and the feature should
                % only be disabled when the developer is confident that no freed
                % instance bugs exist.
                % <p>
                % Freed instance protection may be reenabled with
                % METHOD(enable-freed-instance-protection), but only instances
                % freed while instance protection is enabled will be protected.

                setFreedInstanceProtection(0).

        + to stacktrace:
                % Prints out a breve stacktrace--all of the methods which have been
                % called to get to this point in the simulation.  This method is
                % useful for debugging.

                stacktrace().

        + to get-interface-version:
                % Returns a string identifying the program using the breve engine.
                % This string is in the format "name/version".

                return getInterfaceVersion().

        + to report-object-allocation:
                % Prints data about current object allocation to the log.

                objectAllocationReport().
}
```

```
The MenuItem class

@use Abstract.

Abstract : MenuItem (aka MenuItems) [version 2.0] {
    % The MenuItem class holds menu items associated with objects.  Menus
    % can be associated with Mobile objects, in which case they are shown
    % as contextual menu items, or associated with Control objects in which
    % case they are shown under the global application menu.

    + variables:
        menuPointer (pointer).
               name (string).
               method (string).
               owner (object).

               enabled, checked (int).

    + to create-menu named menuName (string) for-object theObject (object) for-method methodName
(string):
        % This method initializes a menu item with title menuName for
               % theObject which will call methodName when selected.

               name = menuName.
               method = methodName.

               owner = theObject.
               self add-dependency on owner.

        menuPointer = menuItemNew(owner, method, name).

        if !menuPointer: {
                       print "error adding menu item for method $methodName.".
                       free self.
                       enabled = 1.
                       return 0.
               }

        return self.

    + to check:
        % Places a check mark next to the menu item.

        if !menuPointer: return.
        menuItemSetCheck(menuPointer, 1).
               checked = 1.

    + to uncheck:
        % Removes the check mark next to the menu item, if it exists.

        if !menuPointer: return.
        menuItemSetCheck(menuPointer, 0).
               checked = 0.

    + to enable:
        % Enables a menu item, if it is disabled.

        if !menuPointer: return.
        menuItemSetEnabled(menuPointer, 1).
               enabled = 1.

    + to disable:
        % Disables the menu item such that it cannot be selected.

        if !menuPointer: return.
        menuItemSetEnabled(menuPointer, 0).
               enabled = 0.

       + to get-description:
               return name.

       + to dearchive:
        menuPointer = menuItemNew(owner, method, name).
               if checked: self check.
               else self uncheck.
               if enabled: self enable.
               else self disable.
               return 1.
}
```

```
The Shape class

@include "Abstract.tz"

Abstract : Shape (aka Shapes) [version 2.0] {
	% The Shape class is a work-in-progress which allows users to create
	% shapes which will be associated with OBJECT(Mobile), OBJECT(Stationary)
	% or OBJECT(Link) objects and added to the simulated world.  An instance
	% of the class Shape may be shared by several objects simultaneously.
	% <p>
	% Each Shape has it's own local coordinate frame, with the origin
	% at the middle of the shape.

	+ variables:
		shapePointer (pointer).
		density (float).
		shapeData (data).
		lastScale (vector).

	+ to init:
		density = 1.0.

  + section "Getting and Setting a Shape's Mass Properties"

	+ to get-mass:
		% If the shape is properly initialized, this method returns the
		% shape's mass.

		if shapePointer: return getMass(shapePointer).
		return 0.0.

	+ to set-mass to newMass (float):
		% Sets the mass for this Shape object.  This implicitly changes the
		% density of the object.

		if shapePointer: shapeSetMass(shapePointer, newMass).

	+ to get-density:
		% If the shape is properly initialized, this method returns the
		% shape's density.

		if shapePointer: return getDensity(shapePointer).
		return 0.0.

	+ to set-density to newDensity (float):
		% Sets the density for this Shape object.  This implicitly changes the
		% mass of the object.

		density = newDensity.
		shapeSetDensity(shapePointer, density).

  + section "Dynamically Changing the Size of a Shape"

	+ to scale by scale (vector):
		% If the shape is <i>not</i> a sphere, scales the shape by the x, y
		% and z elements of scale.  If the shape <i>is</i> a sphere, scales
		% the shape by only the x element such that the shape always remains
		% spherical.
		% <p>
		% After the size has been changed, the instances announces a
		% "size-changed" notification.

		scaleShape(shapePointer, scale).

		lastScale = scale.

		self announce message "size-changed".

	- to get-last-scale:
		% Used internally...
		return lastScale.

  + section "Initializing the Shape"

	+ to init-with-sphere radius r (float):
		% Sets this Shape object to a sphere with radius r.

		shapePointer = newSphere(r, density).

		return self.
```

```
	+ to init-with-cube size v (vector):
		% Sets this Shape object to a rectangular solid of size v.

		shapePointer = newCube(v, density).

		return self.

	+ to init-with-polygon-disk radius theRadius (float) sides sideCount (int) height theHeight
(float):
		% Sets this Shape object to an extruded n-gon of sideCount sides, in other words,
		% a disk with sideCount sides.
		% <p>
		% The distance from the center of the n-gon faces to the vertices
		% is theRadius.  sides has a maximum value of 99.  Higher values
		% will cause the shape not to be initialized.
		% <p>
		% The height, or depth of the extrusion, is theHeight.
		% <p>
		% This method is experimental, but seems to work okay.  Go figure.

		shapePointer = newNGonDisc(sideCount, theRadius, theHeight, density).

		return self.

	+ to init-with-polygon-cone radius theRadius (float) sides sideCount (int) height theHeight
(float):
		% Sets this Shape object to a cone-like shape with sideCount sides.
		% <p>
		% The distance from the center of the n-gon faces to the vertices
		% is theRadius.  sides has a maximum value of 99.  Higher values
		% will cause the shape not to be initialized.
		% <p>
		% The height, or depth of the extrusion, is theHeight.
		% <p>
		% This method is experimental, but seems to work okay.  Go figure.

		shapePointer = newNGonCone(sideCount, theRadius, theHeight, density).

		return self.

	- to get-pointer:
		% Returns the shapePointer associated with this Shape object.  This
		% method is used internally and should not typically be used in
		% user simulations.

		return shapePointer.

  + section "Getting Information About a Shape's Geometry"

	+ to get-point-on-shape on-vector theVector (vector):
		% This method is experimental.
		% <p>
		% Starting from inside the shape at the center, this function goes in
		% the direction of theVector until it hits the edge of the shape.
		% The resulting point is returned.
		% <p>
		% This allows you to compute link points for arbitrary shapes.
		% For example, if you want to compute a link point for the
		% "left-most" point on the shape, you can call this method with
		% (-1, 0, 0).
		% <p>
		% Returns (0, 0, 0) if the shape is not initialized or if an
		% error occurs.

		if shapePointer: return pointOnShape(shapePointer, theVector).
		else return (0, 0, 0).

	+ to destroy:
		if shapePointer: freeShape(shapePointer).

  + section "Serializing the Shape"

	+ to get-data-for-shape:
		% Returns serialized data for the shape (if the shape object has
		% been properly initialized).  Used for archiving/dearchiving,
		% should generally not be called manually, unless you <i>really</i>
		% know what you're doing.

		if shapePointer: return dataForShape(shapePointer).

	+ to archive:
```

```
                shapeData = (self get-data-for-shape).
                return 1.

        + to dearchive:
                shapePointer = shapeForData(shapeData).
                return (super dearchive).

}

Shape : CustomShape (aka CustomShapes) {
        % A CustomShape is a subclass of (Shape) which allows the user to
        % construct an arbitrary convex shape by specifying the faces of
        % the shape.
        % <P>
        % The shapes must conform to the following rules:
        % <li>The point (0, 0, 0) must be on <b>inside</b> (not outside or
        % on the surface of) the shape.
        % <li>The shape must be convex.
        % <li>The shape must be solid and sealed by the faces.
        % </ul>
        % <p>
        % If any of these conditions are not met, you will get errors
        % and/or unexpected results.

        + to init:
                shapePointer = newShape().

        + to add-face with-vertex-list vertexList (list):
                % Adds a face defined by the list of vectors in vertextList.

                addShapeFace(shapePointer, vertexList).

        + to finish-shape with-density theDensity (float):
                % This method must be called after all of the faces are added
                % to complete initialization of the shape.  The density given
                % here will effect the physical properties of the shape if
                % physical simulation is used.  A value of 1.0 is reasonable.
                % <P>
                % If the shape specified is invalid (according to the constraints
                % listed above), this method will trigger an error.

                return finishShape(shapePointer, theDensity).
}

Shape : Sphere (aka Spheres) {
        % This class is used to create a sphere shape.

        + to init-with radius r (double):
                % Initializes the sphere with the given radius.

                shapePointer = newSphere(r, density).

                return self.
}

Shape : Cube (aka Cubes) {
        % This class is used to create an extruded rectangle.  Even though the class
        % is named "Cube", the shapes do not need to be perfect cubes--they can be
        % rectangular solids of all sizes.

        + to init-with size s (vector):
                % Initializes the cube to a rectangular solid with size v.

                shapePointer = newCube(s, density).

                return self.
}

Shape : PolygonDisk (aka PolygonDisks) {
        % This class is used to create a polygon-disk.  This is a shape which can be
        % described as an extruded polygon.

        + to init-with radius theRadius = 1 (float) sides sideCount (int) height theHeight (float):
                % Initializes the polygon-disk.
                % <p>
                % The distance from the center of the n-gon faces to the vertices
                % is theRadius.  sides has a maximum value of 99.  Higher values
                % will cause the shape not to be initialized.
                % <p>
                % The height, or depth of the extrusion, is theHeight.
```

```
                shapePointer = newNGonDisc(sideCount, theRadius, theHeight, density).

                return self.
}

Shape : PolygonCone (aka PolygonCones) {
        % This class is used to create a polygon-cone shape.  This is a shape with a polygon
        % base which tapers off to a point.  A pyramid is an example of a polygon-cone with
        % 4 sides.  As the number of sides increases, the base becomes more circular and the
        % resulting shape will more closely resemble a true cone.

        + to init-with radius theRadius (float) sides sideCount (int) height theHeight (float):
                % Initializes the polygon-cone.
                % <p>
                % The distance from the center of the n-gon faces to the vertices
                % is theRadius.  sides has a maximum value of 99.  Higher values
                % will cause the polygon-cone not to be initialized.
                % <p>
                % The height, or depth of the extrusion, is theHeight.

                shapePointer = newNGonCone(sideCount, theRadius, theHeight, density).

                return self.
}

Shape : MeshShape {
        % An experimental class to load arbitrary 3d mesh shapes.
        % <p>
    % <b>Full collision detection is not currently supported for MeshShapes</b>.
    % MeshShapes are currently collision detected using spheres, with the radius
    % defined by the maximum reach of the mesh.

        + to load-from-3ds file filename (string) with-node nodename = "" (string):
                % Attempts to load a mesh from a 3D Studio scene file named filename.
                % The optional argument nodename specifies which mesh in the scene
                % should be loaded.  If nodename is not provided, the first mesh found
                % in the scene is loaded.

         shapePointer = meshShapeNew(filename, nodename).

                return self.
}
```

```
The Movie class

@use Abstract.

Abstract : Movie {
        % Records MPEG movies of breve runs.
        % <P>
        % The dimensions of the movie are determined by the size of the simulation
        % viewing area when the movie export begins.  Resizing the viewing area
        % while the movie is exporting will produce undesirable results.
        % <P>
        % The Movie class does not work when using the non-graphical
        % ("breve_cli") breve.

        + variables:
                moviePointer (pointer).

        + to record to filename (string):
                % Create a new MPEG movie file with the name filename.
                % New frames will be automatically added to the movie as the
                % simulation runs until the object is released or METHOD(close)
                % is called.  filename should end with ".mpg" or ".mpeg".

                moviePointer = movieCreate(filename).

        + to iterate:
                self add-frame-from-display.

        + to close:
                % Closes the MPEG file and stops recording.

                if moviePointer: movieClose(moviePointer).
                moviePointer = 0.

        - to add-frame-from-display:
                % Add a frame from the current simulation display.

                if moviePointer: movieAddWorldFrame(moviePointer).

        + to destroy:
                self close.
}
```

```
The Image class

@use Abstract.

Abstract : Image (aka Images) [version 2.0] {
    % The Image class provides an interface to work with images and
    % textures.  The individual pixels of the image can be read
    % or changed by the simulation as desired.
    % <P>
    % The image class can read rendered images from the screen using the
    % method METHOD(read-pixels), so that agents in the 3D world
    % can have access to real rendered data.  In addition, the method
    % METHOD(get-pixel-pointer) can be used to provide a pointer to the
    % RGBA pixel data so that plugins can access and analyze image data.
    % This could be used, among other things, to implement agent vision.

    + variables:
        imageData (pointer).
        textureNumber (int).
        modified (int).

    + to init:
        textureNumber = -1.

    + to iterate:
        if modified: {
            imageUpdateTexture(imageData).
            modified = 0.
        }

  + section "Loading and creating Images"

    + to load from imageFile (string):
        % Loads an image from the file imageFile.

        if imageData: imageDataFree(imageData).

        imageData = imageLoadFromFile(imageFile).
        if !imageData: {
                        print "Error loading image $imageFile!".
                        return 0.
               }

        if textureNumber != -1: modified = 1.

        return self.

    + to init-with width imageWidth (int) height imageHeight (int):
        % Creates an empty image buffer with width imageWidth
        % and length imageLength.

        if imageData: imageDataFree(imageData).

        imageData = imageDataInit(imageWidth, imageHeight).

        return (self).

    - to destroy:
        if imageData: imageDataFree(imageData).

    - to get-image-data:
        return imageData.

  + section "Getting information about the size and format of an image"

    + to get-width:
        % Returns the width of the image.

        if !imageData: return 0.
        return imageGetWidth(imageData).

    + to get-height:
        % Returns the width of the image.

               if !imageData: return 0.
        return imageGetHeight(imageData).

  + section "Getting the value of pixels"

    + to get-red-pixel at-x x (int) at-y y (int):
        % Returns the red pixel at the image coordinates (x, y).
```

```
        % The pixel value is given on a scale from 0.0 to 1.0.

               if !imageData: return 0.
        return imageGetValueAtCoordinates(imageData, x * 4, y).

    + to get-green-pixel at-x x (int) at-y y (int):
        % Returns the green pixel at the image coordinates (x, y).
        % The pixel value is given on a scale from 0.0 to 1.0.

               if !imageData: return 0.
        return imageGetValueAtCoordinates(imageData, (x * 4) + 1, y).

    + to get-blue-pixel at-x x (int) at-y y (int):
        % Returns the blue pixel at the image coordinates (x, y).
        % The pixel value is given on a scale from 0.0 to 1.0.

               if !imageData: return 0.
        return imageGetValueAtCoordinates(imageData, (x * 4) + 2, y).

    + to get-alpha-pixel at-x x (int) at-y y (int):
        % Returns the alpha channel pixel at the image coordinates (x, y).
        % The pixel value is given on a scale from 0.0 to 1.0.

               if !imageData: return 0.
        return imageGetValueAtCoordinates(imageData, (x * 4) + 3, y).

    + to get-rgb-pixel at-x x (int) at-y y (int):
        % Returns the red, green and blue components of the pixel
        % at image coordinates (x, y) as a vector.

        r, g, b (double).

               if !imageData: return (0, 0, 0).

        r = imageGetValueAtCoordinates(imageData, x * 4, y).
        g = imageGetValueAtCoordinates(imageData, (x * 4) + 1, y).
        b = imageGetValueAtCoordinates(imageData, (x * 4) + 2, y).

        return (r, g, b).

+ section "Setting the value of pixels"

    + to set-red-pixel to redPixel (float) at-x x (int) at-y y (int):
        % Sets the red pixel value at coordinates (x, y) to redPixel.
        % redPixel should be a value between 0.0 and 1.0.

        modified = 1.
        imageSetValueAtCoordinates(imageData, (x * 4), y, redPixel).

    + to set-green-pixel to greenPixel (float) at-x x (int) at-y y (int):
        % Sets the green pixel value at coordinates (x, y) to greenPixel.
        % greenPixel should be a value between 0.0 and 1.0.

        modified = 1.
        imageSetValueAtCoordinates(imageData, (x * 4) + 1, y, greenPixel).

    + to set-blue-pixel to bluePixel (float) at-x x (int) at-y y (int):
        % Sets the blue pixel value at coordinates (x, y) to bluePixel.
        % bluePixel should be a value between 0.0 and 1.0.

        modified = 1.
        imageSetValueAtCoordinates(imageData, (x * 4) + 2, y, bluePixel).

    + to set-alpha-pixel to alphaPixel (float) at-x x (int) at-y y (int):
        % Sets the alpha pixel value at coordinates (x, y) to alphaPixel.
        % alphaPixel should be a value between 0.0 and 1.0.

        modified = 1.
        imageSetValueAtCoordinates(imageData, (x * 4) + 3, y, alphaPixel).

    - to set-pixel to pixelVector (vector) at-x x (int) at-y y (int):
        % Deprecated -- for compatibility only.

        self set-red-pixel to pixelVector::x at-x x at-y y.
        self set-green-pixel to pixelVector::y at-x x at-y y.
        self set-blue-pixel to pixelVector::z at-x x at-y y.

    + to set-rgb-pixel to pixelVector (vector) at-x x (int) at-y y (int):
        % Sets the red, green and blue pixel values at image coordinates
        % (x, y) from the values in pixelVector.
```

```
        self set-red-pixel to pixelVector::x at-x x at-y y.
        self set-green-pixel to pixelVector::y at-x x at-y y.
        self set-blue-pixel to pixelVector::z at-x x at-y y.

    + section "Reading Pixels from the Screen"

      + to read-pixels at-x x (int) at-y y (int):
          % Reads pixels into this Image from the rendered image on the
          % screen.  The resulting image can be written to a file or
          % analyzed if desired.  This is only supported in graphical
          % versions of breve.

          imageReadPixels(imageData, x, y).

    + section "Getting a Pointer to Pixel Data"

      + to get-pixel-pointer:
          % Returns a pointer to the pixels this image is holding in RGBA
          % format.  The size of the buffer is 4 * height * width.  This
          % data is provided for plugin developers who wish to read or
          % write pixel data directly.

          return imageGetPixelPointer(imageData).

      - to get-texture-number:
          % Internal use only.

          if textureNumber == -1: textureNumber = imageUpdateTexture(imageData).

          return textureNumber.

    + section "Writing an image to a file"

      + to write to imageFile (string):
          % Write the image to imageFile.  The image is written as a
          % PNG file, so imageFile should end with .PNG.

          imageWriteToFile(imageData, imageFile).

}
```

```
The Camera class

@use Abstract.

Abstract : Camera (aka Cameras) [version 2.4] {
	% Summary: creates a new rendering perspective in the simulated world.
	% <P>
	% The Camera class is used to set up a viewing perspective in a simulation.
	% Creating a new camera object places a viewing area with the new camera
	% perspective in the main viewing window.
	% <P>
	% See the OBJECT(Image) class to read data from a Camera (or from the
	% main simulation window) into a pixel buffer.  This can be useful for
	% implementing vision algorithms.

	+ variables:
		cameraPointer (pointer).
		shared (int).

	- to set-camera-pointer to p (pointer):
		% Used internally.

		if !shared: cameraFree(cameraPointer).

		cameraPointer = p.
		shared = 1.

	+ to init:
		cameraPointer = cameraNew().
		self set-size with-height 100 with-width 100.
		self set-position with-x 0 with-y 0.

  + section "Configuring the Camera"

	+ to enable-smooth-drawing:
		% Enable smooth drawing for the camera.  Smooth drawing enables
		% a smoother blending of colors, textures and lighting.  This feature
		% is especially noticeable when dealing with spheres or large objects.
		% <p>
		% The disadvantage of smooth drawing is a potential performance hit.
		% The degree of this performance hit depends on the number of polygons
		% in the scene.  If speed is an issue, it is often best to disable
		% both lighting and smooth drawing.

	 cameraSetDrawSmooth(cameraPointer, 1).

	+ to disable-smooth-drawing:
		% Disable smooth drawing for the main camera.
		% See METHOD(enable-smooth-drawing) for more information.

		cameraSetDrawSmooth(cameraPointer, 0).

  + section "Enabling and Disabling the Camera"

	+ to disable:
		% Disables this camera.  The view from this camera will not be
		% updated or drawn to the viewing window.

		cameraSetEnabled(cameraPointer, 0).

	+ to enable:
		% Enables the camera.  The view from this camera will be updated
		% and drawn to the viewing window after each iteration.

		cameraSetEnabled(cameraPointer, 1).

  + section "Enabling and Disabling Text in the Camera's Display"

	+ to enable-text:
		% Enables text for this camera.
		cameraTextSetEnabled(cameraPointer, 1).

	+ to disable-text:
		% Disables text for this camera.
		cameraTextSetEnabled(cameraPointer, 0).

  + section "Changing the Size, Position and Perspective of the Viewing Window"

	+ to get-width:
		% Returns the current camera width.
```

```
        return cameraGetWidth( cameraPointer ).

    + to get-height:
        % Returns the current camera width.

        return cameraGetHeight( cameraPointer ).

    + to set-size with-height newHeight (double) with-width newWidth (double):
        % Sets the size of the camera viewing area.

        cameraResizeDisplay(cameraPointer, newWidth, newHeight).

    + to set-position with-x newX (double) with-y newY (double):
        % Sets the position of the camera viewing area inside the main window.

        cameraPositionDisplay(cameraPointer, newX, newY).

    + to look at target (vector) from position (vector):
        % Moves the camera to position and aims it at target.  target is
        % is the target's location <b>relative to the camera</b>, not the
        % target's "real-world" location.

        cameraPosition(cameraPointer, position, target).

    + to set-z-clip to distance (double):
        % Sets the Z clipping plan to theDistance.  The Z clipping plan
        % determines how far the camera can see.  A short Z clipping distance
        % means that objects far away will not be drawn.
        % <p>
        % The default value is 500.0 and this works well for most simulations,
        % so there is often no need to use this method.
        % <p>
        % Using a short Z clipping distance improves drawing quality, avoids
        % unnecessary rendering and can speed up drawing during the simulation.
        % However, it may also cause objects you would like to observe in the
        % simulation to not be drawn because they are too far away.

        cameraSetZClip(cameraPointer, distance).

    + to get-rotation:
        % Returns a vector containing the rotation of the camera about the X-
        % and Y-axes return cameraGetRotation(cameraPointer).
        return cameraGetRotation(cameraPointer).

    + to set-rotation x rx (float) y ry (float):
        % Sets the rotation of the camera about the X- and Y-axes.
        cameraSetRotation(cameraPointer, rx, ry).

    - to delete:
        if !shared: cameraFree(cameraPointer).
}
```

```
The Mobile class

@use Real.
@use Shape.
@use MenuItem.

Real : Mobile (aka Mobiles) [version 2.2] {
	% Mobile objects are objects in the simulated world which move around
	% and interact with other objects.  This is in contrast to
	% OBJECT(Stationary) objects which can collide and interact with
	% other objects but which never move.
	% <P>
	% When a Mobile object is created, it will be by default a simple
	% sphere.  You can change the appearence of this sphere by using
	% methods in this class, or its parent class OBJECT(Real).  Or
	% you can change the shape altogether with the method METHOD(set-shape).

	+ variables:
		worldObjectShape (object).
		linkPointer (pointer).

		archiveLocation (vector).
		archiveRotation (matrix).
		archiveVelocity (vector).
		archiveRvelocity (vector).
		archiveAcceleration (vector).

		linkForce, linkTorque (vector).

		physicsEnabled (int).

	+ to init:
		e = .2.
		eT = .5.
		mu = .2.

		color = (1, 1, 1).

		linkPointer = linkNew().
		realWorldPointer = linkAddToWorld(linkPointer).

		self set-shape to (controller get-generic-shape).

	- to get-link-pointer:
		% For internal use only.

		return linkPointer.

  + section "Setting the Shape of a Mobile Object"

	- to register with-shape theShape (object):
		% Deprecated.  Don't use.
		print "warning: the method \"register\" of Mobile is deprecated, use the method
\"set-shape\" instead".

		self set-shape to theShape.

	- to set shape theShape (object):
		% Deprecated.  Don't use.
		print "warning: the method \"set\" of Mobile is deprecated, use the method \"set-
shape\" instead".

		self set-shape to theShape.

	+ to set-shape to theShape (object):
		% Associates a OBJECT(Shape) object with this Mobile object.
		% Returns this object.

		if !(theShape get-pointer):
			die "attempt to register Mobile object with uninitialized shape ($theShape)".

		if worldObjectShape:
			self remove-dependency on worldObjectShape.

		worldObjectShape = theShape.
		self add-dependency on worldObjectShape.

		linkSetShape(linkPointer, (theShape get-pointer)).

		return self.
```

```
	+ to get-shape:
		% Returns the OBJECT(Shape) associated with this Mobile object.

		return worldObjectShape.

+ section "Configuring Physics Parameters"

	+ to enable-physics:
		% Enables physical simulation for a OBJECT(Mobile) object.
		% This must be used in conjunction with a
		% OBJECT(PhysicalControl) object which sets up physical
		% simulation for the entire world.
		% <p>
		% When physics is enabled for an object, the acceleration
		% can no longer be assigned manually--it will be computed
		% from the forces applied to the object.

		physicsEnabled = 1.
		linkSetPhysics(linkPointer, 1).

	+ to disable-physics:
		% Disables the physical simulation for a OBJECT(Mobile) object.

		physicsEnabled = 0.
		linkSetPhysics(linkPointer, 0).

	- to suspend-physics:
		linkSetPhysics(linkPointer, 0).

	- to resume-physics:
		linkSetPhysics(linkPointer, physicsEnabled).

	+ to get-mass:
		% Returns the mass of the object.

		return (worldObjectShape get-mass).

+ section "Controlling the Agent's Motion and Position"

	+ to move to newLocation (vector):
		% Moves this object to location newLocation.

		if !realWorldPointer:
			die "attempt to move uninitialized Mobile object.".

		linkSetLocation(linkPointer, newLocation).

	+ to set-rotation to theRotation (matrix):
		% Sets the rotation of this object to the rotation matrix theRotation.
		% Working with matrices can be complicated, so a more simple
		% approach is to use METHOD(rotate).

		linkSetRotationMatrix(linkPointer, theRotation).

	+ to set-rotation-euler-angles to angles (vector):
		% Sets the rotation of this object to the Euler angles specified
		% by angles (in radians).

		m (matrix).
		r00, r01, r02, r10, r11, r12, r20, r21, r22 (double).

		r00 = cos(angles::z)*cos(angles::x) - cos(angles::y)*sin(angles::x)*sin(angles::z).
		r01 = cos(angles::z)*sin(angles::x) + cos(angles::y)*cos(angles::x)*sin(angles::z).
		r02 = sin(angles::z)*cos(angles::y).

		r10 = -sin(angles::z)*cos(angles::x) - cos(angles::y)*sin(angles::x)*cos(angles::z).
		r11 = -sin(angles::z)*sin(angles::x) + cos(angles::y)*cos(angles::x)*cos(angles::z).
		r12 = cos(angles::z)*sin(angles::y).

		r20 = sin(angles::y)*sin(angles::x).
		r21 = -sin(angles::y)*cos(angles::x).
		r22 = cos(angles::y).

		m = [ ( r00, r01, r02 ), ( r10, r11, r12 ), ( r20, r21, r22 ) ].

		self set-rotation to m.

	- to rotate around-axis thisAxis (vector) by amount (float):
		% Deprecated.  Renamed to METHOD(set-rotation).

		self set-rotation around-axis thisAxis by amount.
```

```
+ to set-rotation around-axis thisAxis (vector) by amount (float):
       % Sets the rotation of this object around vector axis thisAxis
       % by scalar amount (in radians).  This is an "absolute" rotation--the
       % current rotation of the object does not affect how the
       % object will be rotated.  For a rotation relative to the
       % current orientation, set METHOD(relative-rotate).

       length (float).

       # normalize the axis

       length = |thisAxis|.

       if length == 0.0: return.
       thisAxis /= length.

       linkSetRotation(linkPointer, thisAxis, amount).

+ to relative-rotate around-axis thisAxis (vector) by amount (float):
       % Sets the rotation of this object around vector axis thisAxis
       % by scalar amount (in radians).  This is a rotation relative to the
       % current position.

       length (float).

       # normalize the axis

       length = |thisAxis|.

       linkRotateRelative(linkPointer, thisAxis, amount).

+ to point vertex theVertex (vector) at theLocation (vector):
       % An easier way to rotate an object--this function rotates
       % an object such that the local point theVertex, points towards
       % the world direction theLocation.  In other words, theLocation
       % is where you want the object to face, and theVertex indicates
       % which side of the object is to be considered the "front".

       v (vector).
       a (float).

       v = cross(theVertex, theLocation).
       a = angle(theVertex, theLocation).

       if |v| == 0.0: {
              self rotate around-axis theVertex by 0.01.
              return.
       }

       self rotate around-axis v by a.

+ to get-location:
       % Returns the vector location of this object.

       return linkGetLocation(linkPointer).

+ to get-rotation:
       % Returns the matrix rotation of this object.

       return linkGetRotation(linkPointer).

+ to offset by amount (vector):
       % Moves this object by amount, relative to its current position.

       linkSetLocation(linkPointer, (self get-location) + amount).

+ to set-acceleration to newAcceleration (vector):
       % Sets the acceleration of this object to newAcceleration.
       % This method has no effect if physical simulation is turned
       % on for the object, in which case the physical simulation
       % engine computes acceleration.

       linkSetAcceleration(linkPointer, newAcceleration).

+ to set-rotational-acceleration to newAcceleration (vector):
       % Sets the rotational acceleration of this object to
       % newAcceleration.  This method has no effect if physical
       % simulation is turned on for the object, in which case the
       % physical simulation engine computes acceleration.
```

```
        linkSetRotationalAcceleration(linkPointer, newAcceleration).

+ to get-acceleration:
        % Returns the vector acceleration of this object.

        return linkGetAcceleration(linkPointer).

+ to set-velocity to newVelocity (vector):
        % Sets the velocity of this object to newVelocity.

        if !realWorldPointer: {
                print "set-velocity called with uninitialized Mobile object".
                return.
        }

        linkSetVelocity(linkPointer, newVelocity).

+ to set-force to newForce (vector):
        % Sets the velocity acting on the object to newForce.  This
        % force will remain in effect until it is disabled with a
        % new call to METHOD(set-force).

        if !linkPointer: {
                print "set-force called with uninitialized Mobile object".
                return.
        }

        linkForce = newForce.

        linkSetForce(linkPointer, linkForce).

+ to get-force:
        % Returns the force acting on the object, which was previously
        % set using METHOD(set-force).

        return linkForce.

+ to set-torque to newTorque (vector):
        % Sets the torque acting on the object to newTorque.  This
        % torque will remain in effect until it is disabled with a
        % new call to METHOD(set-torque).

        if !linkPointer: {
                print "set-torque called with uninitialized Mobile object".
                return.
        }

        linkTorque = newTorque.

        linkSetTorque(linkPointer, linkTorque).

+ to get-torque:
        % Returns the torque acting on the object, which was previously
        % set using METHOD(set-torque).

        return linkTorque.

+ to set-rotational-velocity to angularVelocity (vector):
        % Sets the rotational velocity of this object to
        % angularVelocity.

        linkSetRotationalVelocity(linkPointer, angularVelocity).

+ to get-velocity:
        % Returns the vector velocity of this object.

        return linkGetVelocity(linkPointer).

+ to get-rotational-velocity:
        % Returns the vector angular velocity of this object.

        return linkGetRotationalVelocity(linkPointer).

+ to get-distance from otherObject (object):
        % Returns the scalar distance from this object's center to
        % otherObject.

        return | (self get-location) - (otherObject get-location) |.

+ to transform world-vector theVector (vector):
        % Transforms theVector in the world coordinate frame to a
```

```
                % vector in the frame of this object.

                return vectorFromLinkPerspective(linkPointer, theVector).

        + to get-bound-maximum:
                % Returns the vector representing the maximum X, Y and Z locations of
                % points on this link.

                return linkGetMax(linkPointer).

        + to get-bound-minimum:
                % Returns the vector representing the minimum X, Y and Z locations of
                % points on this link.

                return linkGetMin(linkPointer).


    + section "Changing an Object's Appearence"

        + to set-label to theLabel (string):
                % Sets the label to be drawn along side the object.

                linkSetLabel(linkPointer, theLabel).

        + to remove-label:
                % Removes the label that would be drawn next to an object.
                linkRemoveLabel(linkPointer).

        - to archive:
                archiveLocation = (self get-location).
                archiveRotation = (self get-rotation).
                archiveVelocity = (self get-velocity).
                archiveRvelocity = (self get-rotational-velocity).
                archiveAcceleration = (self get-acceleration).
                return (super archive).

        - to dearchive:
                linkPointer = linkNew().
                realWorldPointer = linkAddToWorld(linkPointer).
                self set-shape to worldObjectShape.
                self move to archiveLocation.
                self set-rotation to archiveRotation.
                self set-velocity to archiveVelocity.
                self set-rotational-velocity to archiveRvelocity.
                self set-acceleration to archiveAcceleration.
                return (super dearchive).

    + section "Determining Whether an Object is Colliding"

        - to check-for-penetrations:
                % Depricated.
                return (self get-colliding-objects).

        + to get-colliding-objects:
                % Returns a list of objects currently colliding with this object.
                % This is not meant as a general purpose collision
                % detection tool -- it is meant to detect potentially troublesome
                % configurations of links when they are created.

                return linkGetPenetratingObjects(linkPointer).

        + to check-for-self-penetrations:
                % Determines whether this link is currently penetrating with other links
                % in the same multibody.  This is not meant as a general purpose collision
                % detection tool -- it is meant to detect potentially troublesome
                % configurations of links when they are created.

                return linkCheckSelfPenetration(linkPointer).
}
```

```
The 000abstract class

@use Object.


Object: MyAbstract {



+ to listExperts:
    experts (list).

    experts{0}="sphere".
    experts{1}="cone".
    experts{2}="disk".
    experts{3}="red".
    experts{4}="blue".
    experts{5}="green".

    return experts.


+ to lookupNames with indexes (list):
    lookupNames,temp,experts (list).
    m,n,a,b (int).
    namea, nameb (string).

    experts=(self listExperts).

    for m=0,m<|indexes|,m++:{
        temp={}.
        for n=0,n<|indexes{m}|,n++:{
            namea=experts{indexes{m}{0}}.
            nameb=experts{indexes{m}{n}}.
            push "$namea $nameb" onto temp.
        }
        push temp onto lookupNames.
    }

    return lookupNames.


    #return the location (vector) of the closest agent
    + to askExpert_a with agentsInSight (list) from myLocation (vector):
        agent (object).
        closestLocation (vector).
        closestDistance, difference (float).

        closestLocation = (myLocation). # returning itself if no objects insight, meaning standing
still
        closestDistance = 1000.0.
        difference = 1000.0.

        foreach agent in agentsInSight:{
            difference = |(myLocation)-(agent get-location)|.
            if(difference<closestDistance):{
                closestLocation = agent get-location.
                closestDistance = difference.
            }
        }

        return closestLocation.







   +  to printText with text (string):
        print text.

    + to findSpheres with agentsInSight (list):
        agent (object).
        spheres (list).
        foreach agent in agentsInSight:{
            if((agent getShape)=="sphere"): push agent onto spheres.
        }
        return spheres.

    + to findCones with agentsInSight (list):
```

```
    agent (object).
    cones (list).
    foreach agent in agentsInSight:{
        if((agent getShape)=="cone"): push agent onto cones.
    }
    return cones.

+ to findDisks with agentsInSight (list):
    agent (object).
    disks (list).
    foreach agent in agentsInSight:{
        if((agent getShape)=="disk"): push agent onto disks.
    }
    return disks.

+ to findReds with agentsInSight (list):
    agent (object).
    reds (list).
    foreach agent in agentsInSight:{
        if((agent get-color)::x==1): push agent onto reds.
    }
    return reds.


+ to findGreens with agentsInSight (list):
    agent (object).
    greens (list).
    foreach agent in agentsInSight:{
        if((agent get-color)::y==1): push agent onto greens.
    }
    return greens.


+ to findBlues with agentsInSight (list):
    agent (object).
    blues (list).
    foreach agent in agentsInSight:{
        if((agent get-color)::z==1): push agent onto blues.
    }
    return blues.


 # returning lists of fish in somewhat ordered form according to distance (close, medium, far)
# an extended version of expert_a
+ to askExpert_d with agentsInSight (list):
    agent (object).
    difference (float).
    targetList,closeList,mediumList,farList (list).
    closeView,mediumView,farView (float).


    farView = (controller vision_expert_d).
    mediumView = (controller vision_expert_d)/2.
    closeView = (controller vision_expert_d)/4.


    foreach agent in agentsInSight:{
        difference = |(self get-location)-(agent get-location)|.
        if(difference<closeView): push agent onto closeList.
        else: if(difference<mediumView): push agent onto mediumList.
        else: if(difference<farView): push agent onto farList.
    }

    push closeList onto targetList.
    push mediumList onto targetList.
    push farList onto targetList.

    return targetList.

 + to checkPoison with fish (object):
    #variables that makes up the fish: velocity, shape, color, location
    red,blue,green,cone,sphere,disk (int).
    {
    red=(fish get-color)::x.
    blue=(fish get-color)::y.
    green=(fish get-color)::z.
    cone=(fish getShape)=="cone".
    sphere=(fish getShape)=="sphere".
    disk=(fish getShape)=="disk".

    #if(red+sphere == 2): return -1.
```

```
        if(green == 1): return -1.

        return 1.
        }


    + to get-angle to otherMobile (object) from mySelf (object):
        tempVector (vector).

        tempVector = (otherMobile get-location) - (mySelf get-location).
        return angle((mySelf get-velocity), tempVector).

}
```

The Agent class

```
@define PI              3.14159.
@define SIZE_WORLD      150.
@define MINFISH         80.
@define MAXFISH         160.
@define MINAGE          100.
@define MAXAGE          200.
@define PROB_RED        0.5.
@define PROB_GREEN      0.5.
@define PROB_BLUE       0.5.
@define PROB_DISK       1.0.
@define PROB_CONE       0.66.
@define PROB_SPHERE     0.33.
@define SPEED_FISH      0.1.
@define SPEED_AGENT     3.5.
@define TABULARASA      1.
@define READINPUT       0.
@define THRESHOLD       0.3.
@define RL_INCREMENT   0.1.
@define RUNLENGTH      800.
@define RUNTIMES       10.
@define PRINTING       (0,0,0).
@define STRATEGY       1.


@include "Stationary.tz"
@include "Control.tz"
@include "Mobile.tz"
@include "000abstract.tz"
@use Object.
@use File.



Controller myControl.



Control : myControl {
    + variables:
        myAgent (object).
        runNumber (int).

   + to init:
        (random[20]+1) new Fish.
        runNumber=1.
        myAgent = new JamesBond.
        self point-camera at (0, 0, 0) from (SIZE_WORLD/2,SIZE_WORLD/2,SIZE_WORLD).
        self displayText with runNumber.

    + to displayText with text (string):
        self set-display-text to text at-x -.95 at-y -.95.

    + to iterate:
        newInt (int).
        self update-neighbors.
        super iterate.
        myAgent doNext. # here it all happens
        #self point-camera at (myAgent get-location) from (SIZE_WORLD/6,SIZE_WORLD/6,SIZE_WORLD/3).
        #self point-camera at (myAgent get-location) from (0,0,0).
        #self watch item myAgent.
        if(|all Mobile|<MINFISH):{
            newInt = MAXFISH - |all Mobile|.
            newInt new Fish.
        }


        if((myAgent get-age)>RUNLENGTH):{
            myAgent recordValues.
            # pause simulation
            self sleep for-seconds 5.
            runNumber++.
            free all Mobile.
            free myAgent.
            if(runNumber>RUNTIMES): self end-simulation.
            (random[20]+1) new Fish.
            myAgent = new JamesBond.
            self displayText with runNumber.
        }
```

```
    + to pause:
        self sleep for-seconds 4.

    + to runNumber:
        return runNumber.
}

Mobile : JamesBond {
+ variables:
    energi (float).
    output,input (object).
    good,bad,lost (int).
    lastVelocity,experts,lookupTable,lookupNames,lastAgreementAgents (list).
    namesLookupTable (hash).
    myAbstract (object).
    lockedTarget,lastIndex (int).
    lockedObject (object).

+ to recordValues:
    count,minicount (int).
    namea,nameb,joint,tekst (string).
    { #START: saves the learned values stored in namesLookupTable to the file "output.txt".
    input open-for-appending with-file "output.txt".
    for count=0,count<|experts|,count++:{
        for minicount=count,minicount<|experts|,minicount++:{
         namea=experts{count}.
         nameb=experts{minicount}.
         joint = "$namea $nameb".
         #input write-line text "$joint".
         tekst = namesLookupTable{joint}.
         input write-line text "$joint=$tekst".
        }
    }
    count=(controller runNumber).
    joint="Finished run $count".
    input write-line text joint.
    input close.
    } #END ------------------------------------------------------------

+ to updateLookupTable with feedback (int):
    count (int).
    namea (string).
    valueb (float).
    changes (list).
    tekst (string).
    change (float).

    { #START: Updates the values stored in namesLookupTable
        if(PRINTING): print "feedback: $feedback".
        for count=0,count<|lastAgreementAgents|,count++:{
            namea=lastAgreementAgents{count}.
            tekst= namesLookupTable{namea}.
            #change=max((1-|namesLookupTable{namea}|),RL_INCREMENT).
            change=RL_INCREMENT.
            #print "change $change".
            if(PRINTING):printf "enige om målet: $namea",namesLookupTable{namea},"->".
            #if(feedback>0):
namesLookupTable{namea}=(min((RL_INCREMENT*feedback)+namesLookupTable{namea},1)).
            #if(feedback<0):
namesLookupTable{namea}=(max((RL_INCREMENT*feedback)+namesLookupTable{namea},-1)).
            if(feedback>0):
namesLookupTable{namea}=(min((change*feedback*3)+namesLookupTable{namea},1)).
            if(feedback<0):
namesLookupTable{namea}=(max((change*feedback)+namesLookupTable{namea},-1)).
            if(PRINTING):print namesLookupTable{namea}.
            #print change.
        }
    #print "- - - -".
    # keep a backup-file
    # self copyBackup with (controller get-real-time).

    #update the file to the newlearned values
    #self setValuesToFile.

    } #END ------------------------------------------------------------



+ to getFishData with fish (object):
    shape,color (string).
    { #START: Returns the characteristics of the fish. Used for printing during simulation.
```

```
    shape=(fish getShape).
    if ((fish get-color)::x)==1: color="red,$color".
    if ((fish get-color)::y)==1: color="green,$color".
    if ((fish get-color)::z)==1: color="blue,$color".

    return "$shape($color)".
    } #END ------------------------------------------------------------

+ to kill with fish (object):
    rewardSignal (int).
    history,fishInfo,namea,nameb,name (string).
    myage,count,m (int).
    procent (float).
    { #START: Calls the method (self updateLookupTable with rewardSignal), sets the parameter
lockedTarget to -1., frees the fish
    if(PRINTING):print "- - -".
    #rewardSignal=(myAbstract checkPoison with fish). # negative rewardSignal means poison,
punishment
    rewardSignal=1.

    # colors: x::red y::green z::blue


   #if((fish get-color)::x==1):{ #red sphere
    #    if((fish getShape)=="sphere"): rewardSignal=-1.
           #rewardSignal=-1.
    #}

   if((fish get-color)::x==1):{ #red sphere
        if((fish getShape)=="cone"): rewardSignal=-1.
        if((fish getShape)=="disk"): rewardSignal=-1.
           #rewardSignal=-1.
   }

   if((fish get-color)::z==1):{ #
        if((fish getShape)=="cone"): rewardSignal=-1.
   }




    if(rewardSignal>0): good++.
    else:{
        bad++.
        #print "BAD BAD BAD".
    }
    energi+=rewardSignal.

    self updateLookupTable with rewardSignal.

    #myAbstract printText with "fish has been killed.".
    #myage=(self get-age).
    fishInfo=(self getFishData with fish).
    #if(rewardSignal==-1): print "FASIT: cone(red) HIT: $fishInfo".
    history = "$good;good;$bad;bad".
    output open-for-appending with-file "output.txt".
    output write-line text "$history".
    output close.
    lockedTarget=-1.
    free fish.
    #self move to (0,0,0).
    if(bad>0): procent=(good+0.0)/(bad+0.0).
    controller displayText with "$good:$bad $procent".

    if((bad+good)%50==0):{
        for count=0,count<|experts|,count++:{
            for m=count,m<|experts|,m++:{
                namea=experts{count}.
                nameb=experts{m}.
                namea="$namea $nameb".
                print "$namea",namesLookupTable{"$namea"}.
            }
        }
    }

    } #END ------------------------------------------------------------

+ to doNext:
    fish (object).
    neighbors,spheres,cones,disks,reds,greens,blues,filteredInput,indexMoves,matchingMoves,temp,
lookupValues (list).
```

```
    velocity_new,expertA,tempMove (vector).
    n,m,count (int).
    tempIndex (int).
    hashMove (hash).
    chooseMe (int). # the index pointing to the best choice according to the lookupTable
    chooseLocal,chooseBest,tempMin,tempMax,float_max,float_min (float).
    namea,nameb (string).
    accelerate (int).
    chooseAmongThese (list).


    accelerate = 1.

    { #START 1 filteredInput(e):       the input goes through the experts, storing filtered input
signals
    # getting all the agents that are in the neighborhood, aka the whole world.
    foreach fish in (self get-neighbors): if !(fish == self): push fish onto neighbors.


    #Transforms raw input into filteredInput (list) containing objects indexed by n (number of
experts) aka expert
    spheres=(myAbstract findSpheres with neighbors).
    cones=(myAbstract findCones with neighbors).
    disks=(myAbstract findDisks with neighbors).
    reds=(myAbstract findReds with neighbors).
    blues=(myAbstract findBlues with neighbors).
    greens=(myAbstract findGreens with neighbors).

    push spheres onto filteredInput.
    push cones onto filteredInput.
    push disks onto filteredInput.
    push reds onto filteredInput.
    push blues onto filteredInput.
    push greens onto filteredInput.
    } #END 1- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 2 lastVelocity(e):       Caculates next suggested velocity
    #  containing vectors
    #  indexed by n (number of experts) aka expert

    # doing it simple, finding the closest target relative to itself within each expert-given data
    for n=0,n<|experts|,n++:{
        tempMove=((self askExpert_a with filteredInput{n})-(self get-location)).
        if(tempMove):tempMove=(tempMove/|tempMove|)*SPEED_AGENT.
        lastVelocity{n}=tempMove.       #used in lookuptable when receiving feedback
    }
    } #END 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 3 indexMoves(eu):        eu indicates a sample of a unique move. contains min 1, max 6
elements
    #Find one unique example of each move indexMoves (list)
    #  containing ints matching index in lastVelocity
    #  indexed by m

     for n=0,n<|lastVelocity|,n++:{        # index n saved in list indexMoves
        if(!hashMove{lastVelocity{n}}):{
            hashMove{lastVelocity{n}}=1.
            push n onto indexMoves.
        }
    }
    } #END 3- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 4 matchingMoves(eu):     each unique move is mapped against the matching expert e
      # for instance: eu(1): e1,e3
      #               eu(2): e2
      #               eu(3): e4,e5,e6
      # saving a list temp (with all matching indexes) in a list matchingMoves{index} where index
is in same order as indexMoves
      for m=0,m<|indexMoves|,m++:{
            temp={}.
            for count=0,count<|lastVelocity|,count++:{
                if(lastVelocity{count}==lastVelocity{indexMoves{m}}): push count onto temp.
            }
            push temp onto matchingMoves.
      }
    } #END 4- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 5 lookupNames(eu):        the same as matchingMoves, only replacing ints with actual
names of experts
    #Make a list of lookupNames (list) with strings identifying the hash indexes of matching moves
    #  containing a list of strings
```

```
    #  indexed by m
    lookupNames = (myAbstract lookupNames with matchingMoves).


    if(1==-1):{                     # printing out values - strings - in lookupNames
        print "_____________________".
        for n=0,n<|lookupNames|,n++:{
            printf n.
            for m=0,m<|lookupNames{n}|,m++:{
                printf " ",lookupNames{n}{m}.
            }
            print "".
        }
    }

    if(1==-1):{                     # printing integrity of the values so far
        print "Verifying integrity of the values so far".
        for count=0,count<|experts|,count++: print experts{count},count,lastVelocity{count}.
        for count=0,count<|indexMoves|,count++:{
            printf experts{indexMoves{count}},count,indexMoves{count},"ints: ".
            for n=0,n<|matchingMoves{count}|,n++:{
                printf matchingMoves{count}{n},"names: ".
                printf lookupNames{count}{n}.
            }
            print "".
        }
    }
    } #END 5- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 6 lookupVales(eu):       min and max value in lookuptable for each unique move
    #Find max and min value in lookupTable for each matchingMoves matchingMovesCred (list) (aka
experts agreement matrix)
    #  containing a list with pairs of ints, max and min value found in lookuptable
    #  indexed by m

    lookupValues={}.
    for n=0,n<|lookupNames|,n++:{
        tempMin=2.0.
        tempMax=-2.0.
        for m=0,m<|lookupNames{n}|,m++:{
            namea=lookupNames{n}{m}.
            if(1==-1):print "Unique move number $n, agreement number $m named $namea has value
",namesLookupTable{namea}.
            tempMin=min(tempMin,namesLookupTable{namea}).
            tempMax=max(tempMax,namesLookupTable{namea}).
        }
        push {tempMin,tempMax} onto lookupValues.
    }
    if(1==-1):{                     # printing values
        for n=0,n<|lookupValues|,n++:{
            print                                   "AGAIN,                                     unique
move",n,"min",lookupValues{n}{0},"max",lookupValues{n}{1},lastVelocity{indexMoves{n}},experts{index
Moves{n}}.
        }
    }

    } #END 6 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    { #START 7 chooseMe:              decide on target. If no target is locked from before, decide
on new target
    if(STRATEGY==0):{ # choose the one with the highest max score
        float_max=lookupValues{0}{1}.
        chooseMe=0.
        for count=0,count<|lookupValues|,count++:{
            if(lookupValues{count}{1}>float_max):{
                float_max=lookupValues{count}{1}.
                chooseMe=count.
            }
        }
    }

    if(STRATEGY==1):{ # choose the one with the lowest max score
        float_min=lookupValues{0}{1}.
        chooseMe=0.
        for count=0,count<|lookupValues|,count++:{
            if(lookupValues{count}{1}<float_min):{
                float_min=lookupValues{count}{1}.
                chooseMe=count.
            }
        }
    }
```

```
if(STRATEGY==2):{ # choose the one with the highest min score
    float_max=lookupValues{0}{0}.
    chooseMe=0.
    for count=0,count<|lookupValues|,count++:{
        if(lookupValues{count}{0}<float_max):{
            float_max=lookupValues{count}{0}.
            chooseMe=count.
        }
    }
}



} #END 7 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



chooseAmongThese={}.
# if m gets a non-negative value, then some available values needs to be learned
m=-1.
for n=0,n<|lookupValues|,n++:{
    if(|lookupValues{n}{0}|<THRESHOLD):{
        m=n.
        #print "need to learn $n".
        push n onto chooseAmongThese.
    }
    if(|lookupValues{n}{1}|<THRESHOLD):{
        m=n.
        #print "need to learn $n".
        push n onto chooseAmongThese.
    }
}



# if m<0, means all values available has been learned, then exploit information
if(m==-1):{
    chooseAmongThese={}.
    m=0.
    tempMax=-2.0.
    for n=0,n<|lookupValues|,n++:{
        #if(lookupValues{n}{1}>tempMax && lookupValues{n}{0}>THRESHOLD):{
        if(lookupValues{n}{0}>THRESHOLD && lookupValues{n}{1}>tempMax):{
            tempMax=lookupValues{n}{1}.
            m=n.
            #push n onto chooseAmongThese.
        }
    }
    push m onto chooseAmongThese.
    #print "don't need learning, expected min",lookupValues{m}{0},"max",lookupValues{m}{1}.

}

# stay focused if target has been locked, and there still exists fish of this kind
if(lockedTarget>-1 && |filteredInput{lockedTarget}|>0):{
    tempMove=lastVelocity{lockedTarget}.
    accelerate=10.
}else:{
    if |chooseAmongThese|>0:{
        count=random[|chooseAmongThese|].

        if count==|chooseAmongThese| && count>0:count--.

        m=chooseAmongThese{count}.
    }
    #print m.
    #print count,"<",|chooseAmongThese|.
    #count=(random[count]).
    #count=random(|chooseAmongThese|).

    #print "random",count.
    #for count=0,count<|chooseAmongThese|-1,count++:{
    #    print "$count",chooseAmongThese{count}.
    #}
    #print "value stored",chooseAmongThese{count}.
    #if(count>|chooseAmongThese|):count--.
    #m=chooseAmongThese{count}.
        tempMove = lastVelocity{indexMoves{m}}.
        lockedTarget=m.
```

```
            lastAgreementAgents=lookupNames{m}.
            lastIndex=m.


    }

    # testing if tempMove scores bad on the lookupTable

    #controller displayText with experts{indexMoves{m}}.
    expertA=tempMove.

    #print "choosing expert $maxNumber".
    velocity_new=expertA.
    #(controller displayText with experts{maxNumber}).


    if(|velocity_new|>0):{
        self set-velocity to ((velocity_new/|velocity_new|)*SPEED_AGENT*accelerate).
        self point vertex (0, 1, 0) at (self get-velocity)/|(self get-velocity)|.
    } else: {
        self set-velocity to tempMove.
    }


    # sending the agent back to the centrum if too far off.
    if (|(self get-location)|>SIZE_WORLD*2):{
        (self move to (0,0,0)).
        lockedTarget=-1.
        #lastIndex=-1.
    }



+ to get-angle to otherMobile (object):
    tempVector (vector).
    { #START
        tempVector = (otherMobile get-location) - (self get-location).
        return angle((self get-velocity), tempVector).
    } #END

+ to findSmallestIndex with targets (list):
   location (vector).
    count,smallest (int).
    distance (float).
    { #START
    distance=SIZE_WORLD*4.
    foreach location in targets:{
        if(|location - (self get-location)|<distance):{
            distance=|location - (self get-location)|.
            smallest=count.
        }
        count++.
    }
    return smallest.
    } #END

#return the location (vector) of the closest agent
+ to askExpert_a with agentsInSight (list):
    agent (object).
    closestLocation (vector).
    closestDistance, difference (float).
    { #START
    closestLocation = (self get-location). # returning itself if no objects insight, meaning
standing still
    closestDistance = 1000.0.
    difference = 1000.0.

    foreach agent in agentsInSight:{
        difference = |(self get-location)-(agent get-location)|.
        if(difference<closestDistance):{
            closestLocation = agent get-location.
            closestDistance = difference.
        }
    }

    return closestLocation.
    } #END

+ to init:
    runNumber (int).
```

```
    count,minicount (int).
    RLrows (int).
    namea,nameb,name (string).
    poisonList (list).
    initialValue (float).
    { #START

    myAbstract = new MyAbstract.

    #myExpert = new ExpertDef.
    output = (new File).
    input = (new File).

    runNumber=(controller runNumber).
    print "Now doing runNumber $runNumber".

    output open-for-appending with-file "output.txt".
    output write-line text "************************************".
    output write-line text " run $runNumber - high level abstraction".
    output write-line text "".
    output close.

    #output open-for-appending with-file "output.txt".
    #input open-for-reading with-file "input.txt".
    self set-shape to (new PolygonCone init-with radius 0.5 sides 6 height 1.0).
    self set-neighborhood-size to SIZE_WORLD*3.
    self set-color to (1, 1, 1). # white  (red ,green ,blue)
    self move to (20,20,20).
    energi = 1000.0.
    print "... and action!".
    self handle-collisions with-type "Fish" with-method "kill".

    experts{0}="sphere".
    experts{1}="cone".
    experts{2}="disk".
    experts{3}="red".
    experts{4}="blue".
    experts{5}="green".

    for count=|experts|, count>0, count--: {
        RLrows+=count.
        push (0.0,0.0,0.0) onto lastVelocity.
    }


   if(READINPUT==1): (self getValuesFromFile).
   if(READINPUT==0):{
        for count=0,count<|experts|,count++:{
            for minicount=count,minicount<|experts|,minicount++:{
                namea=experts{count}.
                nameb=experts{minicount}.

                #print "$namea $nameb".
                if(TABULARASA==1): namesLookupTable{"$namea $nameb"}=0.0.
                else :{
                    namesLookupTable{"$namea $nameb"}=(random[0.75] - 0.25).

                }
            }
        }
    }

    #(self setValuesToFile).

    if(1==1):{
        for count=0,count<|experts|,count++:{
            for minicount=count,minicount<|experts|,minicount++:{
                namea=experts{count}.
                nameb=experts{minicount}.
                name="$namea $nameb".
                print "$name",namesLookupTable{"$name"}.
            }
        }
    }
    } #END


# (controller get-real-time): add to name when saving a copy of previous files.


+ to copyBackup with name (string):
```

```
    count,minicount (int).
    newName,namea,nameb,joint (string).
    newName="$name backup.txt".

    input open-for-writing with-file "$newName".
    for count=0,count<|experts|,count++:{
        for minicount=count,minicount<|experts|,minicount++:{
             namea=experts{count}.
             nameb=experts{minicount}.
             joint = "$namea $nameb".
             input write-line text "$joint".
             joint = namesLookupTable{joint}.
             input write-line text "$joint".
        }
    }
    input close.
#        output open-for-appending with-file "output.txt".
#        output write-line text "$history".
#        output close.


+ to setValuesToFile:
    count,minicount (int).
    namea,nameb,joint (string).
    input open-for-writing with-file "input.txt".
    for count=0,count<|experts|,count++:{
        for minicount=count,minicount<|experts|,minicount++:{
             namea=experts{count}.
             nameb=experts{minicount}.
             joint = "$namea $nameb".
             input write-line text "$joint".
             joint = namesLookupTable{joint}.
             input write-line text "$joint".
        }
    }
    input close.
#        output open-for-appending with-file "output.txt".
#        output write-line text "$history".
#        output close.

+ to getValuesFromFile:
    name (string).
    value (float).
    count (int).

    input open-for-reading with-file "input.txt".

    while((input is-end-of-file)==0):{
        count++.
        name=(input read-line).
        value=(input read-line).
        if(name && value):{
            namesLookupTable{"name"}=value.
            if(1==-1):{
                print name,namesLookupTable{"name"}.
            }
        }
    }
    input close.

    #(controller pause).

}
Mobile : Fish {
    + variables:
        myColor (vector).
        maxAge (int).
        myHeight (float).
        myRadius (float).
        myShape (string).
        number (float).

    + to init:
        startSpeed (vector).
        rand (float).

        myRadius = 0.5.
        myHeight = 1.0.
        myColor=(0.0,0.0,0.0).
        maxAge=random[MAXAGE - MINAGE] + MINAGE.
```

```
        #rand=random[1.0].
        #myColor=(0,0,1).
        #if(rand<0.33): myColor=(1,0,0).
        #if(rand>0.66): myColor=(0,1,0).
        if(random[1.0]<PROB_RED): myColor = myColor+(1,0,0). #  red | green | blue
        if(random[1.0]<PROB_GREEN): myColor = myColor+(0,1,0). #  red | green | blue
        if(random[1.0]<PROB_BLUE): myColor = myColor+(0,0,1). #  red | green | blue

        number=random[1.0].

        myShape="disk".
        self set-shape to (new PolygonDisk init-with radius myRadius sides 3 height myHeight).

        if(number<PROB_CONE):{
            myShape="cone".
            self set-shape to (new PolygonCone init-with radius myRadius sides 3 height myHeight).
        }
        if(number<PROB_SPHERE): {
            myShape="sphere".
            self set-shape to (new Sphere init-with radius myRadius).
        }




        self set-color to myColor.
        self            move            to            random[(SIZE_WORLD,SIZE_WORLD,SIZE_WORLD)]-
((SIZE_WORLD/2),(SIZE_WORLD/2),(SIZE_WORLD/2)).
        startSpeed =(random[(1,1,1)]-(0.5, 0.5, 0.5)).
        self set-velocity to (startSpeed/|startSpeed|)*SPEED_FISH.

        #print "New fish has entered the world, lifespan $maxAge".

     + to iterate:
        oldS,newS (vector).


        oldS = self get-velocity.
        newS = (oldS +random[(2,2,2)]-(1,1,1)).
        if(random[5]<2 && newS): self set-velocity to ((newS/|newS|)*SPEED_FISH).
        if(|(self get-location)|>(SIZE_WORLD)) : {
            self set-velocity to (self get-velocity)*-1.
        }

        if(self  get-velocity):self  point  vertex  (0,  1,  0)  at  (self  get-velocity)/|(self  get-
velocity)|.

        if((self get-age)>(maxAge)):{
            #print "tired of waiting, disappear!!!!".
            free self.
        }

    + to getShape: return myShape.
}
```