Proceedings of the 2007 IEEE Symposium on
Artificial Life (CI-ALife 2007)

The Gridbrain: an Heterogeneous Network for Open Evolution in 3D Environments

Telmo Menezes and Ernesto Costa
Centre for Informatics and Systems
of the University of Coimbra (CISUC)
{telmo, ernesto}@dei.uc.pt

***Abstract*— A reasoning system for autonomous agents under open evolution is presented. The system's design goal is to bridge the gap between Artificial Life concepts and popular computer simulation systems, namely 3D graphics engines and Newtonian physics simulation engines, with possible applications to complex systems research and digital entertainment. The theoretical model of a dual layered heterogeneous network of components is proposed as a solution for reasoning in environments where agents' perceptions are provided as arbitrarily sized vectors of symbols. A set of network components aimed at spatial reasoning is presented. Algorithms designed to maintain open evolution under real-time computational constraints are proposed. Experimental results are presented and analyzed.**

I. Introduction

The gridbrain is a reasoning system conceived for autonomous agents that operate in simulated three dimensional Newtonian physics worlds. An important goal for our work is the possibility of using Artificial Life concepts, like open evolution, in the development of generative 3D worlds. We present not only a reasoning system, but also a set of algorithms designed to maintain open evolution in a society of agents. We aim at both scientific and engineering applications. Scientific applications exist in the realms of Artificial Life and Complexity Science, where simulation is an important research tool. Possible engineering areas of application are digital entertainment, educational software and others where real time interaction with human users is important.

We identified a set of challenges that must be addressed to achieve this goal: real time constraints; abstraction of the sensor/actuator model and facilitation of spatial reasoning.

Traditional Artificial Life systems like Tierra [1] or Avida [2] focus on the study of biological processes by establishing an analogy between living systems and computer programs. These systems attempt to, in a reasonable amount of time, simulate phenomena that occur in nature in very large time scales. One problem is that the results of these simulations tend to be meaningful only to scientists. We are interested in showing these phenomena to non-scientists, presenting them in the context of recognizable environments that can be interacted with. Technological progress in the digital entertainment industry provides us with very realistic visualization and physical simulation tools. We attempt to establish a bridge between Artificial Life concepts and these environments.

We identify two types of real time constraints that must be taken into account: the computational complexity of the reasoning algorithm that agents execute and the time frame under which open evolution produces visible improvements and changes. The first type of constraint was considered when designing the gridbrain model, while the second was addressed by the evolutionary algorithms we propose.

Other Artificial Life simulations exist that are able to display open evolution in 3D environments, notably Polyworld [3], one of the inspirations for our work. These simulations tend to operate under predefined world models and use sensors and actuators that aim at mimicking nature as much as possible. In Polyworld, one of the main sensors for agents is a simulation of direct vision, in the form of a pixel grid representing the point of view of the agent. A reason for this drive to simulate realistic sensors, actuators and world models is the application to robotics. We do not contest the merit of this approach, but we want to propose a different one aimed at a different set of environments.

In computer games, the common practice is to provide agents with information about their surroundings in a preprocessed form, usually analogous to vectors of perceivable entities and their characteristics [4]. This is an approach aimed at the engineering of pragmatic artificial intelligence solutions. We believe this approach is also interesting for scientific purposes, as it more easily allows for the definition of sensor/actuator models at different abstraction levels. This is an important feature for research in Complexity Science due to the interdisciplinary nature of the field, ranging from physical and chemical to social phenomena [5] [6].

The model we will describe is an evolution of a previous one with similar goals [7], which was inspired by John Holland's work with the ECHO [8] simulation and showed promising results. The experimental results were obtained using Bitbang, an agent framework developed by our research lab [9] and two open source software engines, one for 3D visualization [10] and one for Newtonian physical simulation [11].

In the next section we will describe the gridbrain theoretical model. Then mutation and recombination operators will be addressed, followed by the proposal of a set of components for spatial reasoning. The evolutionary process will be discussed and experimental results presented.

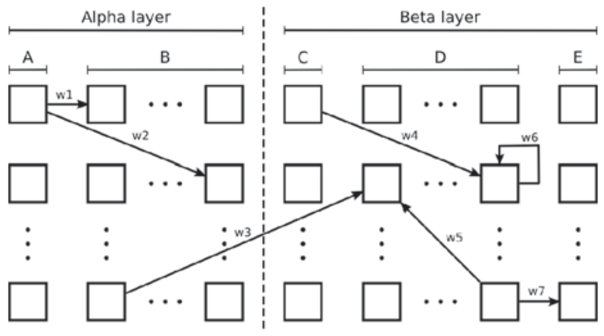

Fig. 1. The gridbrain abstract model. A set of valid connections is displayed.

II. The grid brain abstract model

As shown in figure 1, the gridbrain consists of a network of components which are placed in a grid. This grid is divided in two layers, the first α columns belonging to the alpha layer and the last β columns belonging to the beta layer. To these two layers correspond different topological rules and computational methods. The alpha layer is responsible for analyzing perceptions relative to other entities in the world and feeding the data resulting from this process to the beta layer. The beta layer is responsible for general decision-making.

There are five types of components: external perceptions, internal perceptions, operators, aggregators and actions. External perceptions are only allowed on the first column of the alpha layer (A), the external perception column. Internal perceptions are only allowed on the first column of the beta layer (C), the internal perception column. Actions are placed on the last column of the beta layer (E), forming the action column. We will call the remaining columns in the alpha layer the alpha middle zone (B) and the remaining columns in the beta layer the beta middle zone (D). Operators are allowed in both middle zones, while aggregators are allowed only in the alpha middle zone.

External perceptions are relative to entities in the perception field of the agent. An external perception has an implicit parameter, which is a numerical identifier of the entity it refers to at that moment. As the alpha layer is evaluated for every entity perceivable by the agent, each external perception is instantiated for each one of these entities during a computation cycle of the gridbrain. The usefulness of external perceptions is maximized by placement in the first column of the alpha layer. This is due to the fact that a perception component accepts no input, while this location makes its output available to all other components that accept inputs. Internal perceptions are relative to some internal state of the agent, and thus are placed in the beta layer. Unlike the external type, internal perceptions are not parametric, so no instantiation is needed. The same reasoning is thus applied for its placement in the first column of the beta layer. Action components cause the triggering of actions when activated. Multiple actions may trigger in the same computation cycle. Action components produce no outputs, so they are placed in the last column of the grid. Operators are computational components that produce an output that is a function of its inputs. Operators are stateless, even across multiple alpha passes of the same computation cycle. Aggregators are also computational components, but they preserve their state during alpha passes. They are used for maintaining global views of the entity vector during the alpha layer computation. The internal state of aggregators is reset in the beginning of each computation cycle.

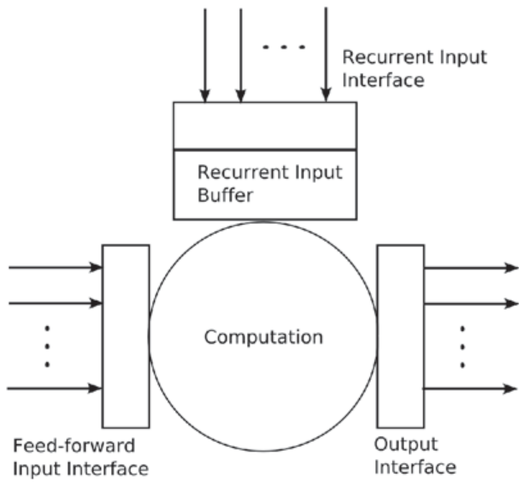

Fig. 2. The generic component model.

Connections are weighted and directed. The weight of a connection is a real value in the $[-1, 1]$ interval. In the alpha layer only feed-forward connections are allowed, while in the beta layer recurrent connections are also allowed. Both the origin and the target components of these recurrent connections must be within the beta layer. Connections may skip columns. A component in the alpha layer may thus connect to any other component with a greater column number, while a component in the beta layer may also connect to any component with a lesser or equal column number if this column belongs to the beta layer.

This topology was devised to deal with the problem of creating a simple representation for a network that must perform computations based on the properties of a vector of entities of arbitrary size. While the rules of component placement are somewhat complicated, the representation of the network for evolutionary or learning purposes is simply a directed graph with weighted connections. Typically the component placement will be predefined and equal for every agent, all evolution or learning taking place by changing the connections.

The component model is shown in figure 2. A component has two input interfaces (one for feed-forward and another for recurrent connections), one output interface, one computation unit and one recurrent input buffer. Input interfaces define the way inputs are stored. The computation unit defines how the inputs are transformed into an output value and the output interface simply transfers this output to the next components. A specific component is defined by its input interfaces and computation unit. The output interface and recurrent input buffer are generic.

Recurrent connections are a way to preserve memory between computation cycles. The values from recurrent connections are stored in a buffer, and only processed in the next cycle. This establishes a way to preserve memory in the beta layer, at the same time not having to deal with stabilization issues associated with recurrence in networks. Depending on the components available, the beta layer may be Turing complete, considering computations spread across multiple simulation cycles. This is the case when a perceptron component is available, as a recurrent neural network can be shown to be Turing complete [12]. Although Turing completeness is not a strict requirement for Artificial Life, it may be useful in some cases, for example as a simple way to provide agents with memory.

Each computation cycle of the grid brain is a sequence of three steps: the first alpha pass sets the internal state of the aggregators, the second alpha pass feeds data from external perceptions to the beta layer and the beta pass fires actions. Both alpha passes are executed one time for each entity in the perception range of the agent, while the beta pass is executed only once. An execution consists of looping through the columns and inside each column through the rows. Each component computes its output and then feeds it to the input interfaces of all the components it is connected to, multiplied by the weight of the connection.

Pseudo-code for a grid brain computation:

```
update all perceptions
clear action list
reset beta layer feed-forward inputs
reset all aggregators' internal state

// first and second alpha passes
loop 2 times:
   for each entity e in the agent's perception range:
      reset alpha layer feed-forward inputs
         for each column c in alpha layer:
            for each row r:
               if component is external perception:
                  set external perception parameter to e
                  compute component at c, r
                  for every connection from this component:
                     output = output * connection weight
                     propagate output to target component \
                     feed-forward interface

// beta pass
for each column c in beta layer:
   for each row r:
      for every input i in component at c, r recurrent \
      input buffer:
         propagate input to component \
         recurrent input interface
         clear recurrent input buffer for component at c, r
         compute component at c, r
         if computation returned action a:
            add a to action list
            for every connection from this component
               output = output * connection weight
               if (target column > c):
                  propagate output to target component \
                  feed-forward interface
               else
                  add output to target component recurrent \
                  input buffer
for every action a in action list:
   execute a
```

III. Mutation and Recombination

The gridbrain was designed to be used in evolutionary environments, so we will detail two conventional genetic operations: mutation and recombination.

For each mutation, one of three atomic operations is performed: creating a new connection, deleting an existing connection or changing the weight of an existing connection. The atomic operation to perform is chosen at random for each mutation, each one having an equal probability of occurrence. After experimenting with alternative approaches, we found that giving deletion and creation of connection equal probability of occurrence effectively prevents bloat. Bloat, in the case of the gridbrain, is a useless increase in the number of connections. When creating new connections, the new connection to create is randomly selected from the space of possible connections. This is done in a way that guarantees equal probability to each possible connection. Due to topological differences in the two layers (feed-forward restrictions on the alpha layer and limitation of recurrent connection to inside the beta layer), each column has a specific number of connections that may originate from it. This number is calculated through equations (1) and (2). Being A_c the number of connections originating from alpha layer column c, B the number of connections originating from a beta layer column, α the number of alpha layer columns, β the number of beta layer columns and h the height of the grid we have:

$$A_c = (\alpha + \beta - c) \cdot h^2 \tag{1}$$

$$B = \beta \cdot h^2 \tag{2}$$

Thus the total number of possible connections, T is:

$$T = \sum_{c=1}^{\alpha} A_c + \sum_{c=\alpha+1}^{\alpha+\beta} B \tag{3}$$

So an alpha layer column has a probability of being selected as origin of the new connection of:

$$p_{c\alpha} = A_c/T \tag{4}$$

And a beta layer column:

$$p_{\beta} = B/T \tag{5}$$

For a connection with an origin in the alpha layer, the target column is selected with equal probability from all columns with greater column numbers. For a connection with an origin in the beta layer, the target column is selected with equal probability from all columns in the beta layer. The origin and target rows are selected from the set of possible rows with equal probability. This same strategy is applied when generating connections for the initialization of a random population of agents in the beginning of the experiment.

Weight change is done by randomly selecting a delta value in the $[-1, 1]$ real interval and adding it to the current weight. The new weight is capped by $[-1, 1]$.

Recombination is done by the definition of connection groups. Every time a new connection is created, it is tagged with an integer value G such that $0 <= G < M$, where M is the number of connection groups predefined for the experiment. When recombination between two agents occurs, one of the progenitors is selected at random to provide its connection set for each group. This strategy attempts to promote the self-organization of functional groups in the connection set.

IV. A SET OF COMPONENTS FOR SPATIAL REASONING

Now that we have described the abstract model of the gridbrain, we will detail a set of components conceived for the development of agents' brains that operate in a three dimensional Newtonian physics simulated world. In the kind of experiment we will describe latter, the agent is provided with perceptions relative to other entities in the world. These perceptions are of an heterogeneous nature. One kind of perception is of a spatial relative nature. One example would be the position of an entity according to a cartesian axis of the agent's frame of reference, normalized to a $[-1, 1]$ real interval. If perception A was the position along the X axis and perception B was the position along the Y axis, a value of -1 for A would mean that the entity was on the far left of the agent's vision field while a 1 value for B would mean that the entity was on the bottom of the agent's vision field. The same logic may be applied to other vectorial values, for example relative velocity. Another kind of perception is of a scalar nature, for example the energy level of the agent normalized to a $[0, 1]$ real interval. Yet another kind of perception is of a boolean nature, for example the existence of a certain characteristic in an exterior entity, where 0 would mean the inexistence of this characteristic and 1 its existence.

The underlying concept for the set of components we will now propose is the merging of analogical signal processing capabilities with boolean logic. We consider three operator components: the threshold, the multiplier and the negator and one aggregator component: the maximizer.

The threshold is equivalent to a common artificial neuron used in neural networks. If the sum of all inputs reaches a threshold value, the output is triggered. Our threshold component uses the absolute value of this sum and has a fixed threshold value of 0.1. This way, if the sum of all inputs is in $[-0.1, 0.1]$ the output will be 0, otherwise it will be -1 if the sum of inputs is negative or 1 if it is positive. The threshold value is fixed because we want all the specific information of a brain to be contained in the connection weights, so that mutation and crossover operators as well as metrics are simpler to define. The threshold value determines the minimum cut-off interval of the threshold. For a signal propagated through a connection with weight w, the cut-off interval for a threshold value of t is $[-t/|w|, t/|w|]$. As $w \in [-1, 1]$, the smallest interval possible is $[-t, t]$. Aiming at general applicability we chose a low 0.1 threshold value.

The behavior of the component was chosen to be symmetrical for an easier mapping to the type of perceptions we described.

The multiplier outputs the product of all the inputs capped by -1 and 1. From a boolean logic perspective this operator works as an AND gate, and from an analogical perspective as an amplifier.

The negator outputs 1 if the sum of its inputs is 0 and outputs 0 in any other case. From a boolean logic perspective this operator works as a NOT/NAND gate, and from an analogical perspective as a comparator.

The maximizer returns 1 if its state equals the maximum value it has received so far and 0 in any other case. Notice that though to negative connection weights, the maximizer may as easily be used as a minimizer.

The multiplier and negator components, equivalent to AND and NOT/NAND gates, define a complete boolean logic system. A network of thresholds may express general purpose analogical signal processing structures. The multiplier and negator perform more specific but commonly useful analogical processing functions. We are aware that common neural network artificial neurons, like the threshold, may be used to express complete boolean logic [13]. However, the purpose of this set of components is to provide expressive building blocks for the class of problems we are studying. This choice is expected to facilitate the evolutionary task of finding viable agents and provide for smaller, less computationally intensive solutions.

V. THE EVOLUTIONARY PROCESS

The evolutionary process is implicitly defined by providing the agents with reproductive actions. Unlike more conventional evolutionary algorithms, there is no explicit fitness function. The agents that are more capable and willing to reproduce have greater chances of passing their genetic code to future agents. This simple process creates evolutionary pressure. In other Artificial Life systems, although there is no explicit fitness function, there still exists a centralized algorithm that takes care of creating new generations. We chose to let the agents decide when to reproduce because we expect this to create interesting trade-offs that contribute to the goal of evolving complex societies. Current states of the agent and of the world may determine different optimal strategies in terms of energy management. An agent may choose to delay reproduction to a point where it has more stored energy, in order to maximize its chances of survival after the reproductive action. Survival considerations may dictate that it is better to use energy at a certain moment to escape a predator or to take actions to increase the availability of food in the future.

We still have a form of centralized control in the reproductive system, that was created to deal with two problems: the initialization of the world to a point where the agents achieve self-sustained evolution and computational power limits. The first problem results from the fact that reproduction is a behavior the agents must develop, and that to develop this behavior the agents must evolve, which is assured by reproduction itself. It is thus necessary to provide for an initial form of evolution that takes the agents to a point where they develop reproductive technology. The second problem is that very successful agents may reproduce and generate a bigger population of agents than the simulation is able to compute in reasonable time. This problem is especially relevant to certain applications with real-time requirements such as digital entertainment. The solution we found to these problems is the reproduction buffer and the population limits. The reproduction buffer is a FIFO (First In, First Out) list of inactive agents of a predefined size. The population limits are lower and upper boundaries for the size of the population of agents. The first problem is addressed by executing the following process at each world simulation cycle:

```
while population < population lower limit:
   if reproduction buffer size <= 0:
      if population > 0:
         select random agent a from population
         copy agent a to agent b
         mutate agent b
         place agent b in the world
      else:
         create randomly initialized agent
   else:
      select random agent a from reproduction buffer
      copy agent a to agent b
      mutate agent b
      place agent b in the world
```

This process guarantees that a new agent is generated if the population size goes bellow the lower limit. If both the world and the reproduction buffer contain no agents, random agents are generated. This constitutes a first phase of evolution that works by brute force. This phase typically lasts for a very short period because the slightest advantage in the ability to conserve energy will give some agents an advantage over others. If agents are present in the world but the reproduction buffer is empty, agents are selected from the world population for cost-free reproduction until the lower population limit is achieved. The longer an agent lives, the higher the probability of it being chosen for reproduction. This defines a second phase where evolutionary pressure exists towards the gathering of energy.

When an agent chooses to reproduce, a process takes place that is described by the following pseudo-code:

```
copy agent a to agent b
add agent b to the beginning if the reproduction buffer

if reproduction buffer size > maximum reproduction \
buffer size:
   remove agent from the end of the reproduction buffer

if population size < population higher limit:
   select random agent a from reproduction buffer
   copy agent a to agent b
   mutate agent b
   place agent b in the world
```

When the first agent reproduces, the system achieves self-sustained evolution and the third phase starts. Notice that the successful execution of a reproductive action does not guarantee the propagation of the agent's genetic code to future generations but only to the reproduction buffer. Evolutionary pressure is maintained because the more times an agent reproduces the more likely it is that its genetic code is transferred to the future generations. Mutation of genetic code is done in the reproduction buffer to world transference step. This is done to increase diversity.

The process described directly applies to reproductions that only involve one progenitor and thus result from pure mutation. If two progenitors are involved, the agent copied to the reproduction buffer is a recombination of the genetic code of the two progenitors. In all cases, mutation is only performed when an agent is transferred from the reproduction buffer to the world.

To summarize, during the bootstrap - phases 1 and 2 of evolution - the only way for an agent to increase its chances of passing its genetic code to future generation is by surviving longer. This is so because during these phases, agents are selected at random for reproduction, and an agent that survives longer is more likely to be chosen. Surviving longer depends on the agent's ability to gather and preserve energy, creating evolutionary pressure to improve these skills. When phase 3 starts, the ability and willingness to reproduce becomes dominant in determining the chances of genetic code propagation because all new agents are created from the reproductions buffer. The only way for an agent to place its genetic code in the reproduction buffer is to successfully execute a reproduction action. This has the added advantage of removing older genetic code - possibly belonging to other agents - from the buffer, thus increasing the change of genetic propagation. Energy gathering and preservation skills are still indirectly important, since reproduction has an energetic cost.

VI. Experimental results - the floaters world

To test and experiment with the gridbrain model we defined a simulated world that we will call the ”floaters world”. This world consists of an unmovable closed cubic box of internal side 500, within which agents and food exist. There is no gravity, so the agents have a total degree of freedom in the three dimensions. Both the minimum and the maximum number of agents allowed is 20, to reduce randomness in the results caused by population fluctuations. Agents have a mass of 1 and are physically modeled as perfect spheres of radius 30. Food items have infinite mass and are modeled as perfect spheres of radius 5. We chose to run these experiments with unmovable food items, again to reduce randomness in the results, in this case caused by agitation in the world. Especially in the beginning of simulation runs, agents tend to bump into food, making it variably harder for future generations to eat. Experiments with variable populations and movable food produced similar results, but with less regularity. An air drag effect is modeled, causing a force to be applied to objects in the opposite direction of its linear and/or angular velocity. The value of this force is a constant multiplied by speed squared. The value used for this constant was 10. The purpose of the drag is to make agents maneuverable.

An agent has the following actions available: *go*, *yaw*, *pitch*, *eat* and *reproduce*. *Go* applies a linear impulse that makes the agent go forward according to its current referential and has an energetic cost of 0.0001 per impulse. *Yaw* and *pitch* apply rotational impulses relative to two axis of this same referential, according to aviation conventions, with a cost of 0.0001 per impulse. *Eat* allows the agent to eat an item of food if the item is in both physical contact with it and in its view range. *Eat* has an energy gain determined by the energetic content of the food item. *Reproduce* involves two progenitors and is initiated by one of them. Reproduction is allowed if the initiating agent is in physical contact with the other progenitor and has the other progenitor in its view range. *Reproduce* has an energy cost of 1.5, paid by the initiating agent. There are 5 connection groups for recombination purposes and 10 mutations are performed by reproduction.

An agent has the following perceptions available: *horizontal position*, *vertical position*, *proximity*, *is food* and *energy level*. *Energy level* is an internal perception, all others are external. *Horizontal and vertical position* give the relative position of an object according to the own agent's referential, normalized to a $[-1, 1]$ real interval. *Proximity* is $1 - relativedistance$, where the relative distance is the distance of another object according to the agent's referential, divided by the agent's vision range. *Proximity* thus returns a value in the $[0, 1]$ interval. *Is food* returns 0 if an object is not a food item, 1 if it is. *Energy level* gives the current energy of the agent divided by its maximum energy, thus returning a value in the $[0, 1]$ interval.

Food is constantly supplied to the world, so that 250 food items are always available. It is distributed uniformly in the space inside the box. Each food item provides 1 unit of energy. The agents have a cone of vision in front of them that is defined by an angle of 150^o and a range of 300. Any entity that is contained in this cone will be seen by the agent and thus processed in the alpha passes of the gridbrain. The agents have a metabolic rate that makes them loose 0.02 energy units per simulation cycle. The maximum energy level per agent is 10. If an age limit of 10 simulated seconds is reached or the energy level drops below 0, the agent dies and is removed from the world.

The physical simulation step is 0.01 seconds long.

Operator and aggregator components are distributed by the middle layers of the agents' gridbrains in and alternating and regular fashion. If components A, B and C are available, they will be distributed along the columns of the middle layers in an ABCABCABC sequence. The agents have 10 alpha middle layers and 10 beta middle layers. A conservative high value of layers was chosen to reduce interference with the results when comparing different setups of components.

Results where collected by storing information about each agent when it dies: the time of death, the total amount of energy gathered (including initial energy) and the number of reproductions performed. This log is then analyzed by calculating the average energy gathered and number of reproductions per agent in a time period. The results presented are the average of 30 simulation runs of 5000 simulation seconds,

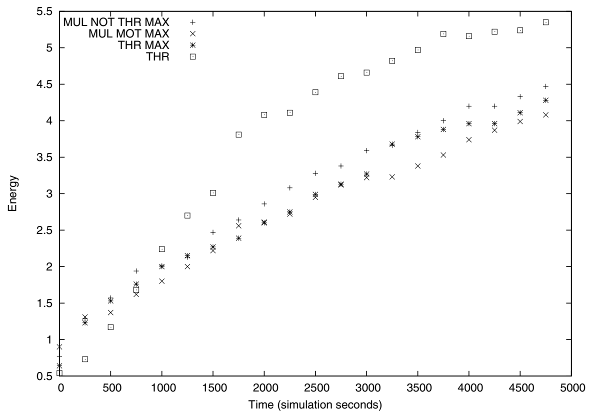

Fig. 3. Evolution of the average energy gathered by agent (average of 30 simulation runs).

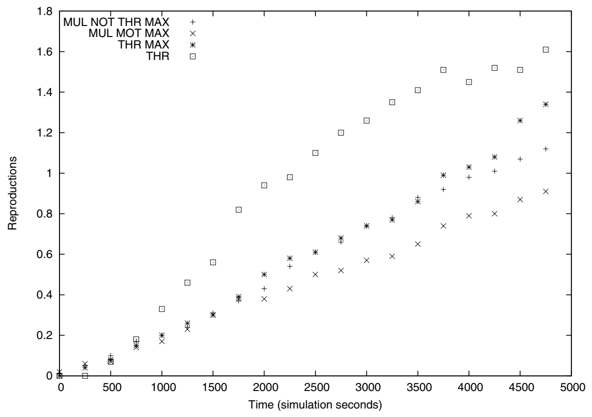

Fig. 4. Evolution of the average number of reproductions by agent (average of 30 simulation runs).

discretized in time periods of 250 simulation seconds. We consider total energy gathered and number of reproductions to be a good indicator of agent performance in this scenario.

Figure 3 present the results for energy gathered using different component setups. The perception and action components formerly enumerated are never changed, but we are interested in determining the impact of the several proposed middle layer components in agent evolution. We found setups using multipliers, negators and maximizers to perform similarly to setups using thresholds and maximizers. Combining negators and thresholds with the maximizer also yields similar results. In this experiment, the generalist capacities of the threshold seem to be sufficient. We then experimented with removing the maximizer and found the pure threshold setup to be the best performer. Initial testing showed setups with

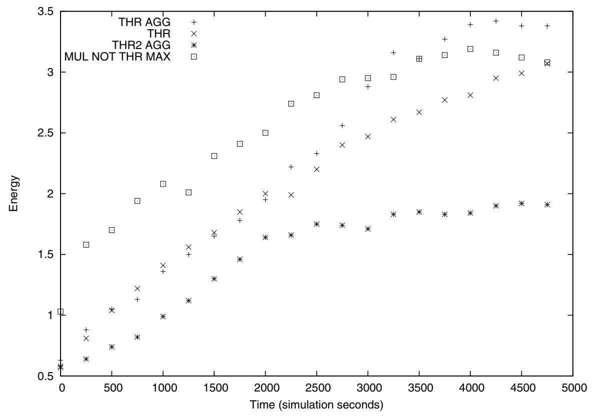

Fig. 5. Average of 30 simulation runs of the evolution of the average energy gathered by agent. (25 food items)

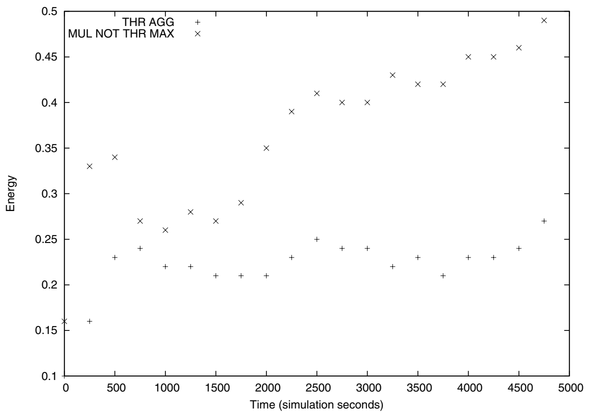

Fig. 7. Evolution of the average energy gathered by agent in the predators experiment.

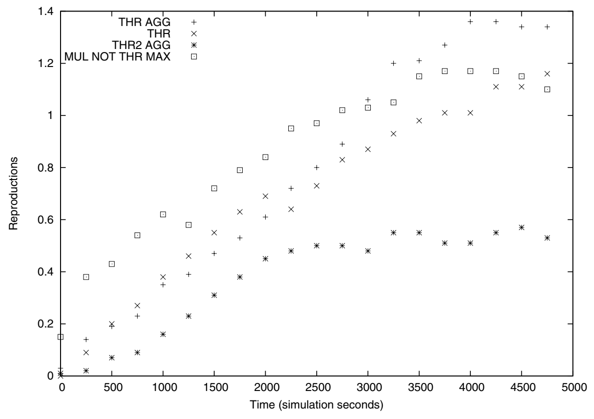

Fig. 6. Average of 30 simulation runs of the evolution of the average number of reproductions by agent. (25 food items)

only multipliers or negators to not promote any evolution at all, so they were discarded from further experimentation. Figure 4 shows similar results for the average number of reproduction per agent.

We repeated the experiment changing the number of food items available to agents by an order of magnitude. There are now 25 food items available in the world at any moment, with an energy content of 5. This scenario presents a much harsher challenge to agent evolution. As can be seen in figure 5, a threshold-only setup now underperforms the multiplier, negator, threshold and maximizer (MUL NOT THR MAX) setup during most of the time, converging in the end. At this point we wanted to test the impact of aggregation using a more general approach than the maximizer, so we introduced a new component, the aggregator threshold. This new component is equivalent to the old threshold except that it is considered an aggregator, so it is not reset between alpha passes. The aggregator threshold is labeled as ”AGG” in the graph. As can be observed, the threshold and aggregator threshold configuration obtains the best performance of all setups considered. We also wanted to test the importance of the threshold symmetrical behavior (as discussed in section IV), so we introduced a new component, the THR2, which behaves more like a conventional artificial neuron. If the sum of its inputs is equal or greater then 0.01 it fires, outputting 1, otherwise the output is 0. As can be seen, the THR2 AGG setup is the worst performer in the experiment.

Figure 6 shows similar results for reproductions per agent.

In the final experiment we tested the two most promising setups from previous experiments - the THR AGG and the MUL NOT THR MAX under a new scenario. There are now two competing species. We call them species in the sense that no reproduction is allowed between them. In this scenario, each species is the predator of the other. The eat action is replaced with a shoot action, with an energy cost of 0.02. The agent may shoot at any moment. The nearest object in a cone with 50% the angle of the vision cone will be the target. If no target exists, the energy is still expended and nothing happens. Any agent shot is destroyed. Shooting an agent of the other species gives 1 energy unit to the shooter, while shooting an agent of the same species gives a 10 energy unit penalty. Agents have now different colors according to species: one is red and the other is blue. Agents are provided with 3 new external perceptions: *is red*, *is blue* and *is target*. *Is red* and *is blue* return 1 if the object is of that color, 0 otherwise. *Is target* returns 1 if the object is the current target, 0 otherwise.

As can be seen in figure 7, the THR AGG setup now underperforms the MUL NOT THR MAX setup. Besides spatial navigation, this scenario involves crucial decisions of a boolean nature. The use of the boolean-oriented MUL and NOT components seems to pay off.

VII. Final Remarks

In this first publication about the gridbrain model, we tried to present the motivations for its design and a set of experimental results that justified its relevance. We are aware that much more experimentation has to be done, as there are many aspects of the system to study: impact of mutation rates and number of recombination component groups, other mutation operators, other components, the utility of recurrent connections in the beta layer and, not less importantly, more interesting scenarios leading to more complex agent behaviors. We believe that the experimental results showed that the dual layer design is effective in providing a reasoning system for agents that perceive the world as vectors of objects. We also believe that the importance of alpha layer aggregation and the study of alternative components became established.

A new class of components will be studied for the beta layer, attempting to facilitate memory and synchronization.

Another bio-inspired approach that was considered when designing the gridbrain, but not discussed in this article, is speciation. By introducing new alpha and/or beta middle columns in evolved brains, we will attempt to simulate evolutionary forks, where new species develop new functionalities, layering upon previous, already established capacities.

One important reason we chose to work in physically simulated worlds is that we believe complexity arises from co-evolution with the environment. Future work will contemplate parallel evolution of physical traits in the agents. One simple example would be to evolve the air drag constant of the agents in the scenarios formerly presented, thus establishing an interesting trade-off: maneuverability versus energy saving.

One subjective but nevertheless relevant aspect of the results we obtained is of a visual nature, and can not be conveyed in this article. The combination of realistic physics simulation and open evolution made the agents, at many points, appear to human observers to have an organic behavior, as opposed to a more mechanic behavior normally associated with computer simulations. Although it can not easily be measured, it is a goal of our work to provide this kind of experience to human observers.

Acknowledgments

The first author would like to acknowledge grant SFRH/BD/19863/2004 from *Fundação para a Ciência e Tecnologia (FCT)*, Portugal.

References

[1] T. S. Ray, "Evolution and optimization of digital organisms," *Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers, Athens, GA, 30602: The Baldwin Press, The University of Georgia*, pp. 489–531, December 1991.

[2] C. Adami and C. Brown, "Evolutionary learning in the 2d artificial life systems avida," in *Proceedings of Artificial Life IV, R. Brooks, P. Maes, Eds.* MIT press, 1994, pp. 377–381.

[3] L. Yaeger, "Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or polyworld: Life in a new context," in *Proceedings of the Artificial Life III Conference*, 1994.

[4] S. Rabin, "Ai game programming wisdom." Charles River Media, 2002.

[5] M. Waldrop, "Complexity: The emerging science at the edge of chaos." Simon & Schuster, 1992.

[6] B. Pullman, "The emergence of complexity in mathematics, physics, chemistry, and biology." Princeton University Press, 1997.

[7] T. Menezes and E. Costa, "A first order language to coevolve agents in complex social simulations," in *Proc. of the European Conference on Complex Systems*, 2006.

[8] J. H. Holland, "Hidden order - how adaptation builds complexity." Addison-Wesley, 1995.

[9] T. Baptista, T. Menezes, and E. Costa, "Bitbang: A model and framework for complexity research," in *Proc. of the European Conference on Complex Systems*, 2006.

[10] N. Gebhardt, "Irrlicht engine." [Online]. Available: http://irrlicht.sourceforge.net

[11] R. Smith, *Open Dynamics Engine - v0.5 User Guide*, 2006. [Online]. Available: http://www.ode.org

[12] H. Hytyniemi, "Turing machines are recurrent neural networks," in *Proceedings of STeP'96*. Finnish Artificial Intelligence Society, 1996, pp. 13–24.

[13] M. Hassoun, "Network realization of boolean functions," in *Fundamentals of Artificial Neural Networks*. MIT Press, 1995, ch. 2.1.1.