Abstract—In this paper we present a new controller architecture. A central design choice is that the controller can be easily modified or changed by evolution. Our aim is to initially endow the agents with as little knowledge as possible and to let them evolve their controllers autonomously. One particular aspect of the controller that we will investigate in this research is the evolution of state persistent controllers. With this is meant controllers that can carry out multiple tasks. Without state persistency, agents may suffer from so-called "unfocused attention": the case where an agent is caught in the middle between tasks and interchangeably executes these partially, but can and will never fully commit to either one and therefore never accomplish any. We will present the state-persistent controller architecture and demonstrate this property in an experiment.

I. INTRODUCTION

OOD controller design for situated autonomous agents is a crucial phase in the engineering process of multi-agent systems across a wide range of research areas - for example, collective robotics [13], [15], [16], artificial life [1] and social simulation [9], [10]. In many situations, it is not possible to design everything beforehand and adaptivity is necessary during system execution time in order to cope with the complexity of a situated environment. In this paper, we investigate if evolutionary learning can deliver such necessary adaptivity. In other words, we design 'dumb' agents that have learning capabilities which enable them to behave more intelligently over time.

In such an approach, designing on the level of primitive actions is too granular; instead we want to design agent behaviours that can be 'activated' intelligently and at will by the agent during execution - similar to the subsumption and other behaviour-based architectures [4]. This paper proposes a method that 1) allows a designer to define behaviours within a tree-structured controller, and 2) where the agents can adapt the controller at runtime by means of evolutionary learning. The method results in an hybrid agent architecture combining the good things of behaviour-based AI [2], [14], reactive planning [12] and evolutionary learning [8].

In previous work, we introduced a method for evolving decision-tree controllers for situated agents. In the current paper, we address a particular important limitation of this earlier work. The improved method can intelligently deal with state persistence. In our case, state persistence means that an agent keeps executing a plan (or: subtree, in our case) as long as that plan is appropriate (regarding current circumstances). Whether a plan is appropriate can either be decided based on specific goals of the plan or defined in the plan itself (in our case, by means of return-statements). A mechanism that intelligently accounts for state persistency allows for evolving controllers that can carry out multiple tasks instead of only one. Without state persistency, agents may suffer from so-called "unfocused attention": the case where an agent is caught in the middle between tasks and interchangeably executes these partially, but can and will never fully commit to either one and therefore never accomplish any.

Existing work related to what we present here is in the areas of agent architectures, controllers, planning and learning. Concerning agent architectures, Gat [11] describes the movement in the 80s and 90s towards the hybridisation of agent architectures encapsulating layers with behaviour-based AI (modular decomposition), reactive planning (ordering predefined plans) and deliberative planning (creation of new plans and learning behaviours). Although such an hybrid architecture is arguably more effective than either one layer on its own, programming these agents is still hard [6]: the planning layers tend to be programmed too simple because of the behaviour-based properties of the modules (emphasis on simplicity). This engineering problem is tackled by Bryson [5], [6] who proposes a development methodology for the Behaviour-Oriented Design (BOD) of agents. One characteristic of BOD controller structures is that they consist of small modules (called: competences) that encapsulate goal directed behaviour. Our controllers also have this characteristic.

The contribution of this paper is twofold. Firstly, the presented method enables the agents to be learning - the controllers of agents are completely flexible. Our aim is to initially endow the agents with as little knowledge as possible and to let them evolve their controllers autonomously. Although we are aware that this is an ambitious goal, it sets us apart from other approaches that 'program' the intelligence of agents at design-time. Secondly, the adaptive (tree) structure of our controllers enables combining hierarchical planning [5] with reactivity by

1. inclusion of global conditional rules to be checked before entering a module,
2. inclusion of one or more local exit rules for leaving a
module, and
- an adaptive mechanism able to create such global conditional rules and local exit rules.

This paper is organised as follows. In Section II we explain the agent controller and the adaptive mechanism. Section III describes a demonstration of the mechanism for performing multi-tasking in the NEW TIES\textsuperscript{2} socio-biological simulation. Section IV concludes and presents some pointers for future work.

II. THE AGENT CONTROLLER

A. Background

The NEW TIES system was developed with a specific type of application in mind: socio-biological simulations. NEW TIES agents live in a “simulated physical” world carried by space, time and energy. Space, time and energy in NEW TIES are discrete. The world is implemented as a rectangular grid, time shifts by atomic timesteps, and energy is administered in basic units. Agents can move over the grid and interact with other agents and objects such as plants or tokens. Agents have to maintain their energy level: everything, even inactively surviving a timestep, costs energy and running out of energy means that the agent dies. To gain energy, an agent must eat food (plants). Each time-step, agents are allowed to perform the following three activities:

1) processing the incoming information,
2) deciding what action to take,
3) performing the chosen action.

Step one and two will be explained below in more detail. Step three is carried out by the agent by sending the chosen action as a request to the environment. The laws of nature governing the environment determine the preconditions and the results of actions, e.g., they specify the amount of energy a plant yields when eaten, the costs of movement, the maximum lifetime for agents, or a minimum age and energy level at which agents can mate. Agents can perform a number of primitive actions, like move, turn left/right, eat, mate, talk, pick up, etc.

Before reaching the controller the raw input undergoes categorisation and conceptualisation. These reduce the dimensionality of the observation space (the raw data where attributes are the elementary properties of all possible entities in the world) by mapping it into another space, where the attributes form the so-called concepts. Raw data are aggregated in two steps. First, raw data are aggregated to form categories that are further aggregated to concepts. The incoming information is processed by categorising the raw data bundle of features. Each feature of objects in the world, like colour or shape, can be regarded as an axis in the feature space; a category is defined by a range of possible values within the whole range of a feature. For example, for the feature colour everything between 1,...,10 could be considered green. Concepts are (more dimensional) entities composed from (one dimensional) categories. To define a concept not all features have to be used. For instance, plants could be green and triangular objects, agents could be pink and circular, while pink, circular and big could correspond to the concept of “enemy”. The perceived concepts are stored in a short-term memory that we call conceptualized input memory (CIM). The memory span of the CIM can vary, but usually it is one. Agents have knowledge about these concepts, because their semantics is stored in the agent’s ontology.

B. Decision Q-Tree

The controller of NEW TIES agents is based upon a decision tree, a so-called decision Q-tree (DQT)\textsuperscript{3}, where each branch in the tree ends with an action. Action selection or decision making amounts to traversing this tree in a stochastic manner. The way this is carried out is dependent on the three structure with its different types of nodes. There are four node type: test nodes, general bias nodes, action bias nodes, and action nodes.

- **Test nodes** – are meant to provide a situation description to the agents. A test node evaluates a Boolean query, e.g., “Is there some plant ahead?” or “Is there an agent nearby?”, and depending on the answer (Yes or No) the tree is further traversed through either of the two child nodes. In these test nodes concepts are used to describe a certain Boolean query. For example the test ‘VISUAL;PLANT;REACHABLE’ would check if the current agent sees a reachable plant. Test nodes are similar to the sense predicates of BOD. During decision-making test-nodes focus on particular objects. For example, if there was first a test-node checking whether saw an agent and later it asks whether is sees something red, the result is that the focus will be on red agents. If there are two contradicting tests in a branch like see agent and see food the second test will fail.
- **Bias Nodes** – facilitate individual choices of the agents driven by their own preferences. A general bias node can be anywhere in the DQT, it may have more than two child nodes, and the choice between the child nodes is probabilistic. The probabilities belonging to each child node \(1,...,n\) are calculated by the agent’s biases. The probability \(p_i\) for choosing child \(i\) at a bias node is calculated by the following equation\textsuperscript{4}:

\[
p_i = \frac{g_i}{\sum_{j=1}^{n} g_j},
\]

where \(n\) is the number of edges and \(0 \leq g_i \leq 1\).

- **Action Bias nodes** – are similar to general bias nodes except that it is always at the last but one level (above the leaves) and its set of children is the complete set of actions. In all types of nodes the bias belong to the

\textsuperscript{2}FP6-IST-FET-003752, http://www.new-ties.org/

\textsuperscript{3}The name contains a ‘Q’ to indicate the particular reinforcement learning algorithm used in NEW TIES. In this paper, we are only concerned with evolutionary learning.

\textsuperscript{4}Note that we are only using evolution as adaptive mechanism and therefore ignore the learned bias.
node and if an offspring individual inherits the node, it also inherits the corresponding bias. Note that Bias nodes and Action Bias Nodes make the tree to be non-deterministic.

- Action nodes – are leaf nodes that contain one action. It is carried out provided the environment allows it. Possible actions that can be performed in the current setup of the system are, for example, eat and turn_left.

C. Behaviour-Oriented Decision Tree (BODT)

We extend the DQT introduced in the previous Section with three nodes: BehaviourSwitch, BehaviourReturn and BehaviourStart. We call the resulting tree a behaviour-oriented decision tree (BODT). The new nodes make it possible to create new trees. These trees are called from another tree, but they can be regarded more as a separate tree than a subtree from the calling tree, because if this it is called or entered the calling tree is substituted by it. Furthermore, for every tree is has to be decided if it will undergo evolution or not. It is possible to preserve parts of the calling tree from being replaced by another tree. The reason for this is to allow for alarm calls, conditions that always have to be checked, before entering the other tree. For example, a condition that always checks for predators.

We briefly describe each of the new BODT nodes and indicate their main functionality.

- BehaviourSwitch nodes – indicate the start of a behaviour. At the moment it is activated, it moves the tree traversal to the root of another DQT. In other words, the currently activated DQT is replaced by a new DQT. This new DQT will be used until it reaches an exit call. BehaviourReturn node (see next item), that returns the control to the previous tree. Another possibility is that this new DQT also contains BehaviourSwitch nodes that replaces it by yet another tree. The main functionality of the BehaviourSwitch node is that it gives the agent state-persistency of a certain behaviour that is present in the tree.

- BehaviourReturn nodes – return the control of decision-making from the currently active DQT to the previous one. The main functionality of a BehaviourReturn node is to exit the tree when it has performed a certain task and a goal is reached. A DQT can contain more than one BehaviourReturn node, so each DQT can have several goals of which to satisfy one is good enough to return to the previous tree.

- BehaviourStart node – protects all parts of the tree that are above the BehaviourStart node. All parts of a tree that are between a BehaviourStart node and BehaviourSwitch node are replaced by the new active DQT. In case there is more than one BehaviourSwitch node in the tree, tree replacement is carried out for the tree part between the BehaviourSwitch node and its closest BehaviourStart node. Other more upper in the tree BehaviourStart nodes will be neglected. The main functionality is to allow for alarms, request or opportunities that make pursuing a different plan more relevant similar to drive collections in Bryson’s BOD [6].

It is possible that during evolution or learning there will be behavioural modules that call each other. These agents will eventually die, because the environment allows a limited amount of decision-making time, so that agents in loops do not perform any action including eating.

III. Demonstration

By means of a simple multi-task environment in NEW TIES, we demonstrate the BODT functionality and its positive effect on the performance of agents. We do this by comparing a population of agents in which the functionality of replacement of trees is turned off (partial-BODT agents) and another population in which it is not (BODT-agents). The first population is called partial-BODT, because except for the replacement functionality the return to a previous tree and the functionality of alarm calls is still enabled. Since replacement is switched off in the partial-BODT population, the agents of this population are unable to focus correctly on multiple tasks. These agents will be stuck into one of the tasks or will switch from one task to another without finishing either one correctly. In other words, it will lack state persistency as was defined in Section I.

A. Task description

In general, as mentioned above, our method is meant for task environments in which situated agents have to perform multiple tasks in order to survive. Particularly, in NEW TIES, agents have to balance between 1) collecting food and 2) looking for partners to reproduce. For the latter, agents are only able to reproduce, if they carry at least three tokens in their bag. Tokens are distributed all over the world and they decay while being carried. Agents will thus only have limited time to collect them and find a mate which is also carrying the required number of tokens.

B. Agent setup

The initial tree is for both population of agents the same. It is depicted in Figure 1. This tree contains two other trees: the EAT tree and the MATE tree. The EAT tree (Figure 3) is depicted to the right of the main tree and the MATE tree (Figure 4) is depicted below the other trees.

This initial tree lacks state persistency. In this tree the MATE-tree of the left most branch will be used if energy is high, because the bias towards it is one. The BehaviourSwitch node of this MATE-tree is directly preceded by a BehaviourStart node. This means that everything above the BehaviourStart node is regarded as an alarm call and thus not replaced by the MATE-tree. The resulting agent behaviour is as follows. When the agent’s energy is high, it will look for tokens. Then during the search for tokens, it is very likely that its energy drops below the high threshold and it will
To get a controller that finishes the tasks of eating and reproduction correctly agents have to get rid of the alarm call by losing the BehaviourStart node before the mate-tree or use the other mate-tree by changing the bias value towards it. Agents having this capability do have sufficient time to collect tokens, because they start eating at the low threshold instead of somewhere between the low and high threshold. These agents have sufficient time to look for tokens, to look for partners and reproduce, before they are hungry and start their eating business.

Partial-BODT agents – Since these agents lack the tree replacement capacity of the BehaviourSwitch node, it is impossible to multi-task correctly between the two tasks. This is regardless how evolution changes the tree. When an agent has an energy between HIGH and LOW it has no memory for whether it started from low energy or from high energy. So it does not know whether to keep looking for food or keep looking for tokens and partners.

BODT agents – The BODT agent possess the tree replacement and are therefore able to learn the task. Notice that even when they are able to learn the take, it does not mean that they (always) will learn it.

C. Environment setup

We use a grid world of 160x160 cells. It contains 5000 plants. Each plant regenerates within 10 time steps after being picked and gives sufficient energy on average to perform 10 actions. Plants regenerate in seasons at the same spot. They always give the same amount of nutrition. The world contains 600 tokens. Each token regenerates within 20 timesteps after being picked. If picked it will decay within 100 timesteps and disappear from the bag. The settings of this environment give agents at most 20 time steps to collect three tokens.

In the experiments reported here, there is no minimal age for mating. Newborn agents can live forever. The reason for this is to keep the density of the population high enough such that agents can find a partner within a given time. A second reason is that in this way it can be easily checked how many new agents are born. The initial population of 500 agents can live for 5 to 25 years and is supplied with sufficient energy to create at least three generations of offspring in order to generate enough diversity. The number of agents is restricted to 2000, due to constrained computational resources.

Each simulation runs for 100,000 time steps, which was found a sufficient number of time steps for the newborn agents to reproduce for several generations.

D. Learning Mechanism

NEW TIES is an evolutionary system with asynchronous reproduction and selection [7]. Evolution concerns the inheritable parts of the agents. In the present study all Decision Q-Trees are made inheritable (the main tree, EAT-tree and MATE-tree), but only the main tree undergoes evolution meaning that it will undergo selection and variation. We summarise the selection (survivor and parent) and variation (recombination and mutation) operators.

- Survivor selection – NEW TIES uses a truly environmental selection method, i.e., not based on any task related notion of (centrally calculated) fitness. Agents die if they run out of energy or reach the maximum age M.
- Parent selection – an agent can decide at any time to mate (subject to some constraints). By choosing the action MATE it selects itself as a would-be parent. To become a parent, it needs to find and ‘convince’ another agent of the opposite sex. To do this, it sends a special message, a mate proposal, whose code and interpretation are hard-wired and the same for all agents. If the other agent accepts this mate proposal the two agents become real parents and produce a child. The environmental constraints on reproduction are that both agents are older than the MateAge threshold value, are of opposite sex and have to be within mating reach. In order to give a newborn child a viable start, both parents donate one third of their current energy to the child at birth.
- Recombination – DQTs are recombined by random subtree exchange as in standard Genetic Programming [3]. In both trees a random crossover point is chosen. The tree of the child is created by taking the tree of the mother and replacing the subtree residing under the crossover point by the subtree under the crossover point in the tree of the father. The genetic biases are simply copied together with the node that the bias belongs to.
- Mutation – in NEW TIES mutation complements recombination. Thus, as opposed to usual evolutionary algorithms, mutation cannot be used as a standalone operator to create offspring from a single parent, but always follows recombination. In the present system only one type of mutation operators is used: bias mutation. Bias mutation perturbs the given bias g by a random value drawn from a normal distribution \( N(0,0.5) \), enforcing lower/upper bounds by a simple boundary rule. This mutation operator is applied with a probability of five percent.

E. Experiments

We conduct two experimental series: one with agents that can learn to use BODT and one with agents that cannot. For each experimental series, we conduct 30 runs.

F. Measures

To compare the performance of agents that can learn to use BODT with agents that cannot, we are using the following measures:

1) development of the population-size over time, and
2) maximum generation reached

The generation of a child is defined as the maximum of the generations of its parents plus one
Fig. 1. Initial controller

Fig. 2.

Fig. 3.

Fig. 4.

Bias Node
Test Node
Action Bias Node
BODTModeStart Node
BODTModeReturn Node
RETURN
sub-tree

Bias Node
Test Node
Action Bias Node
BODTModeStart Node
BODTModeReturn Node
RETURN
sub-tree
The first measure indicates how well the population manages to overcome the mating restrictions, while the second measure indicates how well the solution to these restrictions is passed to the offspring.

G. Results

The results are shown in Figure 5 and Figure 6. As expected, the agents that can use the full functionalities of BODT are outperforming the ones that can only partially use it. In case of partial-BODT agents, the population size did hardly increase. They did not learn to switch correctly between the two tasks, which was expected. The BODT agents did learn to switch correctly. There is an increase in the standard deviation in the BODT setup, because the time at which a population learned the task differed between the runs. In some cases the population did not learn it at all or very late in a run. Once learned the development of a curve is very similar for every run.

In the first 10,000 timesteps, some of the offspring possessed a tree that switched efficiently between the two behaviours. These agents managed to find each other and reproduced. This offspring also received an efficient tree and started to reproduce, creating new generations of agents. Figure 6 illustrates that agents with efficient BODT trees, passed down their genes for many generations, while partial BODT agents stopped reproducing after the initial phase. Figure 7 illustrates how the new generations of agents using BODT kept reproducing over time. Evolution kept preserving the essential part until the end of the experiment.

IV. CONCLUSION AND DISCUSSION

In this paper, we introduced a new structure for adaptive controllers of situated agents. This Behaviour-Oriented Decision Tree (BODT) structure is based on the Decision-Q-Tree (DQT) that was extended with a three new node types. This structure makes it possible 1) for the developer to program behaviours, 2) for the agent to effectively combine reactive behaviour and hierarchical planning, 3) for the agent to evolve its controller, and 4) for the agent to perform successful multitasking.

We demonstrated the new structure in the NEW TIES environment; a simulated physical world, in which agents have the objective to survive. We introduced a multi-task in this world, which agents had to perform well in order to survive. We evaluated two different populations, one where agents were able to switch correctly between tasks, the BODT population, and one that could not learn the task switching. Preliminary results show that adaptive BODT has added value and that the learned information is carried over a number of generations of agents.

Despite the fact that the demonstration is rather simplistic and artificial, it shows that agent benefit from an adaptive BODT controller. We anticipate that the controller has a very generic structure and the natural combination with evolutionary learning makes it an attractive candidate for adaptive controller design in more complex environments. Further experimentation in application domains will be necessary to investigate this presupposition.

The work presented in this paper included one particular form of learning, i.e., evolutionary learning. The DQT model on which the BODT is based, has previously been used successfully for life-time (reinforcement-based Q) learning. In the short run, we will merge the evolutionary and life-time learning mechanisms in attempt to get a controller that is able to adapt faster. Additionally, on the longer term, we will include social learning (learning from other agents), in an attempt to distribute good controllers faster within a population.

REFERENCES

Fig. 5. Population size

Fig. 6. Histogram of Maximum Generation reached

Fig. 7. Maximum Generation over Time