

An Extended Behavior Network for a Game Agent: An Investigation of Action Selection Quality and Agent Performance in Unreal Tournament.

Hugo da Silva Corrêa Pinto¹, Luis Otávio Alvares¹

¹Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{hsspinto, alvares}@inf.ufrgs.br

***Abstract.** This work describes an application of extended behavior networks to the control of an agent in the game Unreal Tournament. Extended Behavior Networks (EBNs) are a class of action selection architectures capable of selecting a good set of actions for complex agents situated in continuous and dynamic environments. They have been successfully applied to the Robocup, but never before used in computer games. We verify the quality of the action selection mechanism and its correctness in a series of experiments. Then we assess the performance of an agent using an EBN against a plain reactive agent with identical sensory-motor apparatus and against a totally different agent built around finite-state machines. We discuss the results of our experiments, point our future work and conclude that extended behavior networks are a good control mechanism for game agents.*

1 Introduction

The problem of selecting a good set of actions for a complex agent situated in a dynamic and complex environment remains a challenge. Robotics was the primary field in which this problem was deeply studied, but recently computer games have become a major motivation for revisiting it

Modern computer action games usually have the agent situated in a 3D virtual environment, interacting in varied ways with several entities in real-time. The scenarios an agent may face are varied and complex. The agent has many weapons available, each with certain properties and several items to use. It moves over different landscapes and interacts with several other agents, both opponents and teammates. The action repertory is large (run, walk, turn, crawl, shoot, change weapons, jump, strafe, pickup item and use item among others) and an agent may carry out more than one action simultaneously, such as shooting while jumping. Also, the agent has many possibly conflicting goals, such as fighting and keeping its safety.

An interesting architecture to tackle this problem is the Extended Behavior Network [1]. Behavior Networks are a class of action selection architectures for selecting good enough actions in dynamic and complex environments. They combine properties of traditional planners (chaining of actions based on preconditions and

effects) and connectionist systems (activation spreading). Action selection is based in the mutual excitation and inhibition among the network nodes, via activation spreading.

Behavior Networks have been constantly evolving since their first appearance [2] as shown by [3], [4], [5] and [6]. They have been applied to animal simulation [7], interactive storytelling [4] and the Robocup [8].

Despite the good results in robotic soccer as reported in [5] and [8], extended behavior networks apparently have never been applied to computer games. Our research investigates how this technique behaves in the control of game robots, providing a novel realm of application in a needed area and contributing to the validation of Extended Behavior Networks as an action selection mechanism for autonomous agents. This last aspect is important from a theoretical AI perspective, because it was hard to tell if the good performance in the Robocup was due to the action selection mechanism employed or due to the sensory-motor system of the robots. Thus further testing of the mechanism is important for its validation.

This work is presented in the following way. We start by presenting an overview of the extended behavior network architecture followed by a presentation of the environment the agent is situated in and the agent architecture. In section 4 we investigate the quality of the action selection in a series of experiments. In sections 5 and 6 we investigate the performance of our agent compared to a totally different agent that used finite-state machines and to an agent that had identical sensors and behaviors but used a purely reactive action selection mechanism. We conclude with considerations on the suitability of behavior networks for controlling game agents and point our next research steps.

2 Extended Behavior Networks

An extended behavior network can be viewed as a set of linked modules and goals that mutually inhibit and excite each other via activation spreading, starting at the goals and flowing to the modules. The modules with higher activation and executability that do not use the same resources are selected for execution at each step. In the next subsections we examine in detail the structure of the network and the action selection algorithm.

2.1 Structure of an Extended Behavior Network

An extended behavior network (EBN) is defined by a set of behavior modules (M), a set of goals (G), a set of sensors (S), a set of resources (R) and a set of control parameters (C). Figure 1 shows the specification of a subset of the behavior network used in this work and Figure 2 the linked version of this subset.

A goal i is defined by a proposition that must be met (Gi), a strength value (Sti) and a disjunction of propositions that provide the context for that goal, called the relevance condition (Li). The strength provides the static, context-independent importance of the goal and the relevance condition the dynamic, context-dependent

one. The total importance of the goal at a certain instant is obtained by multiplying the dynamic and static importances.

<pre> Module precondition EnemyInSight action ShootEnemy effects EnemyHurt 0.6 LowAmmo 0.1 using Hands 2 Head 1 endModule Resource name Legs amount 2 endResource Resource name Head amount 1 endResource Resource name Hands amount 2 endResource </pre>	<pre> Goal condition EnemyHurt strength 0.8 Context endGoal Goal condition Not LowAmmo strength 0.6 context LowAmmo endGoal Parameters name ActivationInfluence value 1.0 name InhibitionInfluence value 0.9 name Inertia value 0.5 name GlobalThreshold value 0.6 name ThresholdDecay value 0.1 endParameters </pre>
---	---

Figure 1. Specification of a simple behavior network

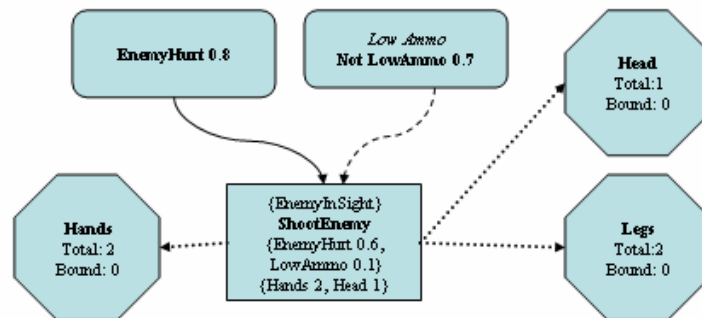


Figure 2. Linked version of the behavior network of Figure 1. The goals are represented by round cornered rectangles, the behaviors by sharp cornered rectangles and the resource nodes by octagons. Straight lines represent predecessor links, dashed lines conflict links and pointed lines resource links

The use of two kinds of conditions in the goals enables us to express goals that become more or less important depending on the situation the agent is in.

A context-independent goal is modeled leaving it without relevance conditions. Goal “EnemyHurt” in figure 1 is an example of such a goal. Note that a goal without relevance conditions amounts to a goal that is always relevant, i.e., its relevance is always maximal.

Each behavior module is specified by a conditions list, an effects list, an action and a resources list. The first list is a conjunction of real valued propositions that represent the needed conditions for the module to execute. The effects list is a conjunction of propositions (each possibly negated) whose values are the values that we expect them to have after the module’s action execution. The resources list is made of pairs (resource, amount), each indicating the expected amount of a resource an agent uses

to perform the action. Let us examine behavior “ShootEnemy” in figure 1. We see that the precondition is that there is an enemy in sight {EnemyInSight}. The expected effects are that the enemy will become hurt with 60% chance (or, conversely, that EnemyHurt verity will be 0.6) and that the agent will be with low ammo with 10% chance {EnemyHurt 0.6, LowAmmo 0.1}. It needs both hands and its head to perform the behavior {Hands 2, Head 1}.

A module receives activation energy from the goals and from other modules along two kinds of links. Predecessor links go from a module or goal B to a module A, for each proposition in the condition list of B that is in the effects list of A such that the proposition has the same sign in both ends of the link. The link from goal *EnemyHurt* to module *ShootEnemy* in figure 2 is an example. Conflict links go from a module or goal B to a module A, for each proposition in the condition list of B that is in the effects list of A such that the proposition has opposite signs at either end of the link. In figure 2, the link from *Not LowAmmo* to *ShootEnemy* is a conflict link. Conflict links take energy away from their targets and predecessor links input energy to their targets. This way a module tries to inhibit modules whose execution would undo some of its conditions and attempts to bring into execution modules whose actions would satisfy any of its conditions.

Each resource is represented by a resource node. These nodes have a function $f(s)$ that specifies the expected amount of the resource available in situation s , a variable *bound* that keeps track of the amount of bound resources and a resource activation threshold $\theta_{Res} \in (0.. \theta]$, where θ is the global activation threshold.

The modules are linked to the resource nodes through resource links. For each resource type in the resources list of a module there is a link from the module to the corresponding resource node.

The control parameters are used to fine tune the network and have values in the range [0,1]. The activation influence parameter γ controls the activation from predecessor links. Inhibition influence, δ , the negative activation from conflict links. The inertia β , the global threshold θ and the threshold decay $\Delta\theta$ have their straightforward meanings. These parameters enable us to influence the degree of persistence of the agent (the higher the inertia the greater the persistence) and how reactive it is (the greater the global threshold the longer the sequence of actions considered when selecting a module for execution), amongst other properties. Default parameters that work well under various circumstances for the Robocup domain are shown in [5].

2.2 Action Selection Algorithm

The modules to be executed at each cycle are selected in the following way: 1) The activation a of the modules is calculated, first spreading from the goals and then from the modules. 2) The executability e of the module is calculated using some triangular norm operation over its condition list. 3) The execution-value $h(a,e)$ is calculated by multiplying a and e . 4) For each resource used by a module, starting by the last non-available resource, the module checks if it has exceeded the resource threshold and if there are enough resources for its execution. If so, it binds the resource. 5) If a module

has bound all of its needed resources it executes and resets the resources thresholds to the value of the global threshold. 6) The module unbinds the resources used.

The thresholds of the resources linearly decay over time, ensuring that eventually a behavior will be able to bind its needed resources and that the most active behavior gets priority.

3 Agent Architecture and Environment

The game in which the agent is situated is Unreal Tournament [10], a first person shooter game. In the game mode we used, DeathMatch, the agent is an armed warrior who must kill all other warriors in an arena. The agent has many weapons available, each with certain properties and several items to use. It moves over different landscapes and interacts with several other agents, both opponents and teammates. The action repertory is large (run, walk, turn, crawl, shoot, change weapons, jump, strafe, pickup item and use item among others) and an agent may carry out more than one action simultaneously, such as shooting while jumping. The scenarios are three-dimensional and the action happens in real-time, so the agent has to decide quickly what to do at each time step.

The primitive sensory information and actions of the agent are defined by an add-on to the game called Gamebots [11]. This package provides a socket interface to the agent and a protocol to interact with the game. Unreal Tournament server send native game messages to Gamebots. In turn Gamebots send its own messages to our agents. Our agent's sensors process these messages and update the values of the conditions of the behavior network and the internal state of the agent. Action selection takes place and the appropriate actions are carried out by sending commands to Gamebots that in turn send low-level Unreal Tournament messages to the server.

Figure 3 gives a detailed overview of the extended behavior network of the agent.

Each proposition of the Behavior Network had a sensor associated to it. Each sensor had a proposition function P and an internal state function I . Function P took the current state of the environment and the internal state of the agent and returned a proposition with value between $[0..1]$. Function I updated the internal state of the agent according to its perception of the environment and the internal state

The actions of the behavior modules were implemented as augmented finite-state machines. When selected for execution each behavior action has a certain amount of time to execute. Each behavior action keeps track of its state independently and has access to the internal state of the agent. A behavior action may issue one or more primitive commands to the game when executing.

4 Assessing Action Selection Quality

Our first series of experiments were designed to assess the quality of action selection. We give a description of how the perceived state of the agent has changed, the actions it executed and the values of the control parameters of the network during the experiment. The default configuration for the network was: γ (ActivationInfluence) =

the two previous ones one after the other, was the most common overall behavior of the agent. We see that the agent makes reasonably long consistent chains of actions even though the agent makes no formal planning.

3. **Reactivity and Persistence:** We can see the sequence of actions {*Explore*, *GoToEnemy*, *FinalizeWithHammer*} as a plan to fulfill the goal *EnemyHurt*. If while going to the enemy the agent was shot, it stopped behavior *GoToEnemy*, executed *BehaviorDodge* and, having evaded the shot, resumed *GoToEnemy*. We see that the agent reacted to an event in the environment and then got back to our perceived “plan”. It exhibited a good compromise between reactivity and persistence. For large values of β , the inertia, the agent took some time to dodge after perceiving a shot and only dodged when a sequence of shots happened.
4. **Resolution of conflicts among goals:** Let us take a look at figure 3 again. We see that goal *EnemyHurt* tries to make behavior *ShootEnemy* execute and goal *Not LowAmmo* tries to prevent it from executing. The goal *Not LowAmmo* has little influence until the agent starts to be with very low ammunition. When this happens, the conflicting influence of *Not LowAmmo* makes the agent switch to the hammer weapon (*FinalizeWithHammer*), because it does not need ammunition. It is an unusual, though sensible approach to ammunition saving, that emerged on interaction of the goals (note that the “designed” way to save ammo is using behavior *StopShooting*). Another case of conflict resolution happens at behavior *GoToEnemy*. To better fulfill *EnemyHurt* the agent has to get near, but to satisfy *Not IAmBeingShot* it better not. We saw in 1 and 3, the network deals with this conflict well, dodging when appropriate and resuming going to the enemy.
5. **Preference for actions that contribute to several goals:** The network always preferred *StandLookout* and *Explore* to *Stand* when they had equal executability, even when we used larger expected values for the *EnemyInSight* effect of *Stand*. The reason is simple: both *StandLookout* and *Explore* contribute to *EnemyHurt* and *HaveHighHealth*, while *Stand* just contributes to *EnemyHurt* (In fact it contributes to *ShootEnemy* that in turn contributes to *EnemyHurt*).
6. **Proper combination of concurrent actions:** We see that the agent makes good use of its resources and combine the actions properly. It shoots while dodging a bullet or running towards the enemy, it stops shooting while exploring or getting a medical kit and even continues to shoot while getting a medical kit. All combinations of actions are reasonable and the good action combinations we could conceive were observed, as the previous analysis attest.

5 The Behavior Network Agent Compared to a Completely Different Agent based on Finite-State-Machines.

To first validate the mechanism, we used an agent implemented by other researchers, Carnegie Mellon’s CMU_JBot, a Java agent based on finite-state machines that comes with the Javabots package [12]. This would prevent bias in our design of its first opponent. We made a series of 10 games of 1 minute. For each game we recorded the number of times the agent hit the opponent, the number of times the agent was hit, the number of times the agent was killed and the number of times the

agent killed the opponent. The total score was got by giving 0.1 to each time the agent hit the opponent and 1 to each time the agent killed the opponent. Table 1 summarizes our results. The rightmost column presents the difference between the score of the agent using the extended behavior network (EBN_Bot) and the score of CMU_JBot.

We see that our agent scored much higher than the agent that used finite state machines. The low number of killings, even when many hits happened, is due to the absence of a chasing mechanism in both agents. The agents wandered through the environment, shot each other and then separated, several times.

This experiment adds evidence to the validity of behavior networks as a suitable action selection mechanism for Unreal Tournament agents, but a doubt remains: Is the better performance due to our sensors and behaviors or due to the action selection mechanism used? The next experiment sheds light on this issue.

Table 1. DeathMatch Results of EBN_Bot and CMU_JBot

Experiment #	EBNBot Hit	EBNBot Kill	CMU_JBot Hit	CMU_JBot Kill	Difference
1	0.7	0	0.2	0	0.5
2	0.1	0	0.0	0	0.1
3	0.3	1	0	0	1.3
4	0.7	0	0.1	0	0.7
5	0.9	0	0	0	0.9
6	0.4	1	0.1	0	1.3
7	0.6	0	0	0	0.6
8	0.7	1	0.0	0	1.7
9	0.9	0	0.1	0	0.8
10	0.2	1	0.2	0	1.0
Mean	0.55	0.4	0.06	0	0.8

6 The Behavior Network Agent Compared to a Plain Reactive Agent that Uses the Same Sensory-Motor Apparatus

In this experiment we compared the EBN agent to an agent that had exactly the same sensors and behaviors, but used a different action selection strategy: At each time step we disregard activation spreading for action selection and take into account only the executability of each module. This amounts to a pure reactive agent using fuzzy sensors in which the most highly likely to execute modules have priority for execution.

Now that we do not have activation we are faced frequently with situations in which two modules have the same execution-value. Let us consider for instance *FinalizeWithHammer* and *ShootEnemy*. When we have *EnemyNear*=1.0 we necessarily will have *EnemyInSight*=1.0. So, both will have identical execution-values. As these behaviors use the same resources and there are resources for just one of them to execute, we need to hard code some priority rules or insert additional

conditions to decide which one to launch when appropriate. We have opted for the first approach in most cases, to differ as little as possible from the original behavior network.

Behavior *Dodge* has priority over *GoToEnemy*, behavior *GoToReachableMedkit* has priority over both *GoToMedkitInSight* and *GoToKnownMedkit*, and behavior *GoToMedkitInSight* has priority over behavior *GoToKnownMedkit*. We incorporate one subtle but important rule: *FinalizeWithHammer* gets priority over *ShootEnemy*. This is needed because whenever *EnemyNear* is true *EnemyInSight* is also necessarily true, and we want the robot to hammer if the enemy is near. We changed module *Explore* to have the condition *Not IAmBeingShot*, both in the behavior network agent and in the plain reactive agent.

To overcome the low number of killings we implemented also a chasing behavior (identical) in both agents. Table 2 summarizes our results for 10 games of 30 seconds each.

Table 2. DeathMatch Results of EBN_Bot against a Reactive Agent.

#	EBNBot Hit	EBNBot Kill	ReactiveBot Hit	ReactiveBot Kill	Difference (EBN-Reactive)
1	0.7	0	0.9	1	-1.2
2	0.1	1	0.1	0	1.0
3	0.3	1	0.2	0	1.1
4	0.2	0	0.3	1	-1.2
5	0.9	1	0.3	0	1.6
6	0.4	1	0.1	0	1.4
7	0.0	0	0.1	0	-0.1
8	0.7	1	0.9	0	0.8
9	0.6	0	0.6	1	-1.3
10	0.2	1	0.3	0	0.9
Mean	0.44	0.6	0.34	0.3	0.3

7 Discussion and Conclusion

We have seen that the extended behavior network was a good action selection mechanism for a complex game agent with complex goals and actions situated in a dynamic continuous environment.

We verified the properties of a good enough action selection mechanism in the 3D action game domain, namely, persistence, exploitation of opportunities, preference for actions that satisfy multiple goals, proper resolution of conflicts, performing of actions in sequences to achieve goals and sensible selection of concurrent actions.

When compared to another robot built around finite-state machines that used different sensors and behaviors but the same low-level commands, our robot performed very well.

To further validate the mechanism we measured the performance of our agent to an agent that was identical to it except for the action selection mechanism employed. This agent was totally reactive, with some priority rules to enhance its performance added. Again, our robot had significantly bigger overall scores. One interesting point is that our robot had 100% more killings but just a little over 30% more hits. This is due to the quality of its action chains. It stopped to heal itself when very hurt and dodged bullets when taking many consecutive shots. Another point that catches attention is that the mean difference in total score was much smaller in the experiment against the agent with identical sensors and behaviors. It confirms that the quality of sensors and behaviors were in great part responsible for the superior performance of our agent against CMU_JBot.

The superior performance of the robot using extended behavior networks in both experiments makes the case for extended behavior networks as a good and competitive solution to action selection for game agents. The assessment of the quality of the actions selected in section five contributes to validate extended behavior networks as an action selection mechanism for complex agents with many goals situated in complex, dynamic and continuous environments.

Our next steps are investigating ways to make the network automatically adapt its global parameters to maximize the fulfillment of its goals and make the agent adjust the expectations of the effects of the behaviors according to its experience.

References

1. Dorer, K. Extended Behavior Networks for Behavior Selection in Dynamic and Continuous Domains. In: Proceedings of the ECAI workshop Agents in dynamic domains, U. Visser, et al. (Hrsg.) Valenzia, Spanien, (2004).
2. Maes, P. How to do The Right Thing Connection Science Journal, Vol. 1, No. 3., (1989).
3. Tyrrell, Toby. An Evaluation of Maes Bottom-up mechanism for behavior selection. In Journal of Adaptive Behavior 2 (4).(1994). 307- 348.
4. Rhodes, Bradley. PHISH-Nets: Planning Heuristically in Situated Hybrid Networks . MSc Thesis. MIT. (1996).
5. Dorer, K. Extended Behavior Networks for the Magma Freiburg Team. In RoboCup-99 Team Descriptions for the Simulation League. Linkoping University Press, (1999a). 79-83.
6. Nebel, B. and Babovich, Y. Goal-Converging Behavior Networks and Self-Solving Planning Domains, or: How to Become a Successful Soccer Player. s.l. IJCAI03. (2003).
7. Tyrrell, Toby. Computational Mechanisms for Action Selection. PhD Thesis. University of Edinburgh. 1993.
8. Müller, K. RoboterFussball: Multiagentensystem CS Freiburg, Diplomarbeit. Univ. Freiburg. Germany. Feb. (2001).
9. Brooks, R. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation. Volume RA-2, Number 1. (1986).
10. Unreal Tournament <http://www.unrealtournament.com> 28/03/2005
11. Kaminka, G. et al. GameBots: A Flexible Test Bed for Multiagent Team Research. Communications of the ACM Vol. 45, Issue 1. (2002). 43-45.
12. Javabots <http://utbot.sourceforge.net/> 28/03/2005.