Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS 2010)

Reactive Teaming for Intelligent Game Characters

Frederick W. P. Heckel, G. Michael Youngblood and Nikhil S. Ketkar
University of North Carolina at Charlotte
9201 University City Blvd
Charlotte, NC 28223
{fheckel, youngbld, nketkar}@uncc.edu

Abstract

Reactive methods for controlling intelligent agents have proven to be a very strong alternative to planning methods, especially in areas such as robotics and game agents, where real-time performance is critical. Despite this success, coordinating teams of reactive agents can be a non-trivial problem. In this work, we present our initial development of a novel method for coordinating reactive agents using behavior-based control by transferring layers between different agents.

Introduction

Reactive control methods provide efficient and robust alternatives to planning for agent control. Behavior generated by reactive methods is generally less optimal than what can be achieved through planning, but in many cases, speed is more important than optimality. Unfortunately, coordinating teams of reactive agents can still be difficult.

Several qualities of game agents makes the use of lightweight reactive methods appealing. First, the action space of the capabilities of a particular agent at a particular moment in time may differ based on the objects available to it (the agent's inventory). Second, agents have a relatively short life-span, making mistakes in task allocation less expensive. Finally, while game agents must follow the dynamics of their particular game environment, they are not physically embodied, and generally are executed on the same computer and must share system resources with each other.

Our method of transferring individual components of a behavior specification from one agent to another has been successful in initial tests in our game environment. In this paper we describe the basics of our reactive teaming approach and briefly discuss the complexity of our teams.

Background

Much of the research in multi-agent planning focuses on providing near-optimal task assignments for teams. Many different approaches have been taken to finding near-optimal solutions for team coordination, including planning, auctions, and free-market methods (Brumitt and Stentz 1996; Gerkey and Matarić 2002; Kalra et al. 2005).

Several different reactive architectures are commonly used in games, including finite state machines, behavior trees, and subsumption architecture (Isla 2005; Brooks 1986). We focus on behavior-based control due to the simplicity of the representation and its inherent parallelism. Behavior-based control is a generalization of the subsumption architecture with a less restrictive model (Matarić 1992). Subsumption-based agents are composed of multiple behavior layers organized in a prioritized list, with rules governing how layers interact. We use BEHAVEngine, a behavior-based control AI engine for game agent control (Heckel, Youngblood, and Hale 2009). BEHAVEngine provides a hierarchical behavior-based control framework, including a library of pre-defined behaviors that can be composed into agent controllers at run-time through agent specification files.

Method

Our method allows the AI designer to specify teaming agents as normal standalone agents with minimal additional information required. Layers in the behavior-based control specification must be tagged as transferable or non-transferable. Additional information can be added, such as a normalized priority to be used when deciding where the layer should be placed. Behaviors must also be tagged with some basic metadata specifying the requirements for the behavior to run successfully, such as items that the receiving agent must possess and actions it must be able to perform. In addition, only one high-level layer may be transferred at a time, though this may be a complex hierarchical layer internally composed of multiple layers.

Agents must request a layer transfer from another agent. If one agent, Alice, is seeking an additional layer from Bob, it requests that Bob send it a layer. Bob then may either reject the request because it has no transferable layers available, or send a behavior layer. Alice then must confirm the transfer, either accepting or rejecting the layer.

Receiving a layer involves three major steps: locating the appropriate place in the behavior hierarchy to place the layer, moving the other layers and changing priorities as necessary to accommodate the new layer, and finally inserting the new layer and determining the new subsumption policy for the layer.

Our current implementation uses a normalized priority value to help determine where new layers should be placed. The normalized priority defines the percent of layers that should be below the new layer; for example, a priority of 0 means it should be the base layer, while 1.0 should be the top layer.

If the new layer is to be placed somewhere other than the base or top, the priorities of other layers must be adjusted to avoid conflicts. We assume that adjacent layers will not have identical priorities; this is not a substantial limitation as such a situation is still allowable with a hierarchical layer. The priorities of each layer then are evenly spaced, and the new layer is placed in the appropriate position.

Finally, the new layer is inserted in the correct position with a valid priority, and the subsumption policy is adjusted automatically. For this work, we assume that the layer will subsume any lower layers that have the same output effector type, and allow others to run.

Reactive teaming requires the addition of a special meta-layer to each agent; this layer monitors the activity of lower layers. The control layer is placed as the highest priority layer of the agent, but runs before other layers rather than simultaneously. The output of the layer is limited to requests for transfers, behavior transfer offers, and transfer result confirmation.

Evaluation

The system described above was implemented as an extension for BEHAVEngine, and evaluated with simple scenarios as case studies. The scenario included four blank agents and a fully specified agent with behaviors describing a number of patrol routes in the game environment. The blank agents started up, requested behaviors from different agents until the fully specified patrol agent transferred patrol routes. They then activated the given behavior. Communication was treated as independent of distance, so the agents were able to immediately receive behaviors without finding the fully specified agent initially.

An additional test scenario created a single garbage-cleaning robot with behaviors to wander through the world and pick up objects. Several blank agents (from 4 to 50) were added that wandered the world, and in this case, would request behavior transfers only when within range of one another. Once a blank agent received the garbage-cleaning behavior, it would continue to wander, and transfer this behavior to other agents if requested.

For a set of n agents that can perform m behaviors, the overall complexity of the team specification is $n * m$ for fully specified agents, instead of $n + m$ for reactive teaming. This reduces the memory requirements per-agent. Runtime complexity is also reduced, as agents have a fewer behaviors to choose from.

Centralized methods for task assignment require that agents estimate their ability to perform a task. This generates $n{*}m$ scores to be considered. In contrast, reactive teaming requires only constant time interactions between pairs of agents. The task assignment procedure only delays the agent that is requesting a behavior, rather than all agents. In the worst case, it will take $O(n)$ game ticks to assign viable behaviors to all agents, as some agents may only find a viable behavior on their final transfer request.

Agent design complexity will also be minimal. Reactive teaming allows the creation of a small number of agents with all behaviors, which are combined in a scenario with blank agents. The few fully specified agents need no additional information added apart from a single bit per layer specifying whether or not it may be transferred. Additional options are not required, but can be used to adjust how often layers can be transferred.

Conclusions and Future Work

For many applications, including games, reactive methods provide robust, efficient techniques for controlling agents. Team coordination can benefit from reactive methods as well, and we have presented a viable approach for reactive teaming using behavior-based control. Our technique has been implemented as part of the BEHAVEngine AI engine, and initial tests have proven successful.

Though the initial work is promising, the presented technique is limited in its scope. Several choices were deliberately kept simple, using naïve approaches. Having created a proof of concept for layer transfers, further work will address these shortcomings. We plan to improve the process that chooses when to request a new layer, where it should be placed, and what layer should be transferred when a request is received. All of these will take into account more about the current status of the agent, as well as user-definable parameters to adjust how the agent should behave in a team.

References

Brooks, R. 1986. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of* 2(1):14–23.

Brumitt, B. L., and Stentz, A. 1996. Dynamic mission planning for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2396–2401.

Gerkey, B. P., and Matarić, M. J. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation* 18(5):758–768.

Heckel, F. W. P.; Youngblood, G. M.; and Hale, D. H. 2009. Making Interactive Characters BEHAVE. In *Proceedings, Florida Artificial Intelligence Research Symposium.*

Isla, D. 2005. Handling Complexity in the Halo 2 AI. In *Proceedings of the 2005 Game Developers Conference.*

Kalra, N.; Dias, M. B.; Zlot, R. M.; and Stentz, A. T. 2005. Market-based multirobot coordination: A comprehensive survey and analysis. Technical Report CMU-RI-TR-05-16, Robotics Institute, Pittsburgh, PA.

Matarić, M. J. 1992. Behavior-based control: Main properties and implications. In *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, 46–54.