Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Student-Friendly Java-Based Multiagent Event Handling

Dan Tappan
Department of Computer Science
Eastern Washington University
dtappan@ewu.edu

Abstract

This work describes a student-friendly, pedagogy-oriented event-handling system for managing multiagent AI simulations in a classroom environment.

Problem

The primary intent of senior-level and graduate AI programming assignments at Eastern Washington University is to survey a breadth and depth of relevant concepts and practices. Students do not have a working knowledge of AI programming languages already, and it is not within the scope of the course to investigate them in enough detail for practical use. The only reasonable solution is to use the primary programming language of the curriculum: Java.

Despite substantial background in Java programming, very little of this experience translates into the type of programs that are the foundation of modeling and simulating mobile intelligent agents. As a result, students tend to solve assignments in a very haphazard manner, with more time spent hacking the programmatic details than addressing the underlying AI aspects of interest.

In particular, real-time event handling is a major hurdle. Something as conceptually simple as moving an agent realistically from one point to another at a certain speed or over a certain amount of time is actually non-trivial. Students either have no concept of how to approach such a problem and rapidly become frustrated and distracted, or they attempt to misapply the closest familiar approach of threading. Both traditionally result in unmanageable solutions that undermine the strategy of controlled experiments for formal analysis.

Solution

This work in progress introduces an expressive Java API with a reasonable learning curve for handling the types of concurrent events among multiple agents that are common in AI assignments (Bourg 2002; Bourg and Seemann 2004).

Pedagogical Framework

The API is based on a pedagogical framework for designing, implementing, testing, and evaluating agents. It emphasizes forethought in the development process, which can be critiqued before students actually start coding. In particular, it partitions a solution into model and simulation.

The model defines the data and control elements of the agents. Data is what the agent is, based on the properties that appropriately describe it; e.g., size and speed. Control is what it can do, based on the actions that are expected of it; e.g., move and shoot. This framework directly corresponds to the familiar principles in object-oriented programming.

The simulation is what is actually done with the model. It establishes an operational context for executing and evaluating the model under controlled conditions; e.g., agents of different sizes and speeds moving and shooting at each other over multiple runs, with presumably the most effective agent configuration statistically being the one that wins most often.

An agent's process of deciding what to do, as well as when, where, why, and how, is based on the familiar concept of finite-state diagrams, which translate at the architecture level into the interagent communication and event-handling protocols provided by the API.

Architecture

The architecture is a standard model-view-controller structure. The model belongs to the students. Almost any view can be accommodated in a plug-and-play manner

(e.g., gnuplot), but a three-dimensional visualizer from related work is provided by default (Tappan 2008). The controller is the event-handling framework discussed here.

The implementation makes heavy use of established software design patterns, especially the Strategy, Command, and Observer patterns, which are familiar to the students (Gamma et al. 1995). In particular, it allows them to register their agents for context-sensitive callbacks, thereby delegating the simulation-level event coordination to the architecture side and the model-level event processing to their side. An added benefit is that the architecture can log events for later detailed analysis of how the model executed.

The currently supported event categories are designed to control typical physical actions in three-dimensional space (Russell and Norvig 2009). Most have both a simple form, such as moving between two points at a constant speed, as well as advanced forms for aspects like acceleration and deceleration. There are usually multiple ways to request the same behavior, which allows students to choose the most intuitive one based on the context of their solution. This approach facilitates translating their design into an implementation in a disciplined manner.

Example

Landing an autonomous aircraft in Figure 1 demonstrates the handling of the most common event categories.

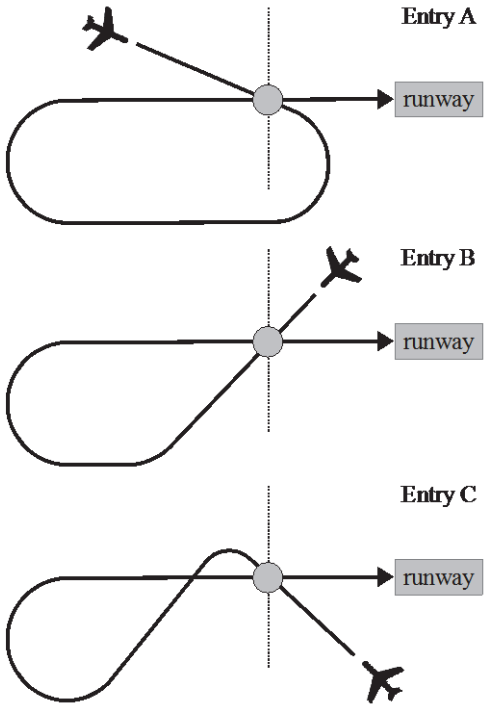

Figure 1: Approach Patterns

The aircraft initially flies to the approach fix, designated as the circle. Depending on the approach angle, one of three entry procedures, *A*, *B*, or *C*, is conditionally executed to align the aircraft with the runway. In the process, its speed and altitude also decrease.

The events to schedule for Entry C, for example, are as follows:

1. Fly to the fix at an initial altitude.
2. Turn left 135 degrees relative to the landing course while slowing to approach speed.
3. Fly for 45 seconds.
4. Turn right to the landing course.
5. Fly to the fix at the approach altitude.
6. Fly to the runway at the landing altitude while slowing to landing speed.
7. Slow to taxi speed.

Evaluation and Future Work

Piecemeal proof-of-concept versions of this work were fielded over two semesters of an undergraduate AI course with very favorable anecdotal feedback. The current work is intended to unify many of the loose ends and simplify the process. The solutions to earlier assignments are being revamped to take advantage of the current version, which will be fielded in two upcoming classes. Example concepts like flocking, following, chasing, and evading were investigated in assignments for bats hunting insects through echolocation, bees communicating to locate flowers, dogs chasing a ball and a laser-pointer dot, aircraft executing maneuvers, and many others. Once stable, the API and assignments will be freely available to the AI-education community at shelby.ewu.edu.

References

Bourg, D. 2002. *Physics for Game Developers*. O'Reilly, Sebastopol: CA.

Bourg, D. and Seemann, G. 2004. *AI for Game Developers*. O'Reilly, Sebastopol: CA.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Indianapolis: IN.

Russell, S. and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River: NJ.

Tappan, D. 2008. A Pedagogical Framework for Modeling and Simulating Intelligent Agents and Control Systems, Technical Report, WS-08-02, AAAI Press.