Student-Friendly Java-Based Multiagent Event Handling

Dan Tappan

Department of Computer Science, Eastern Washington University

Introduction

Java is the AI programming language at EWU because of its familiarity. Students are not, however, prepared from previous experience to apply it effectively to the modeling, simulation, visualization, and analysis tasks that are the backbone of the AI course.

This work provides a student-friendly API for designing, implementing, testing, and evaluating diverse physical agents, with emphasis on disciplined real-time, concurrent event handling.

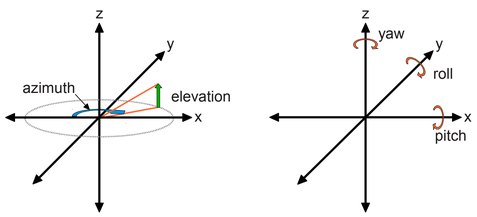
The API supports a traditional model-view-controller architecture that takes advantage of sound software-engineering practices implemented through common design patterns. The student emphasis is on the model.

Model

The model defines an agent as a class structure combining the common API elements and the student's own contributions.

Data

Data is *what an agent is*. The API maintains its identifier, position (arbitrary [x, y, z] or world-based [*longitude*, *latitude*, *altitude*]), attitude (two degrees of freedom [*azimuth*, *elevation*] or three [*yaw*, *pitch*, *roll*]), and velocity.

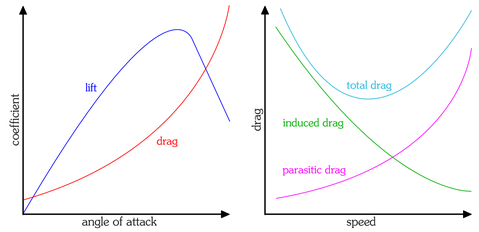


Attitude and velocity are also defined in terms of how they change. Each has a minimum and maximum value, as well as acceleration and deceleration as a scalar value (e.g., meters per second squared), a mathematical function (m/s^2 as a function of altitude), an arbitrary lookup table, or arbitrary code.

Students provide whatever data is appropriate to define their contributions, such as how far an agent can see and how precise its vision is.

Model

Modeling real-world processes often involves looking up complex performance data [2]. The API supports multidimensional lookup tables with linear interpolation, such as curves for coefficient of lift and drag in a flight model as a function of angle of attack and speed [1].



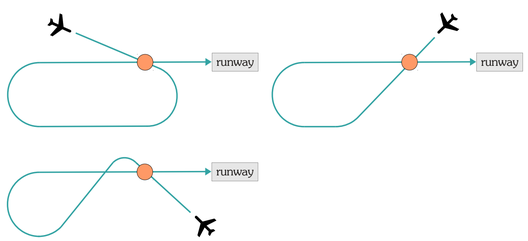
Control

Control is *what an agent can do*. The API provides corresponding methods for all its data in multiple helpful forms (e.g., changing speed, course, altitude, and attitude) and updates agent states accordingly. Students provide the code for their own purposes.

Controller / Simulation

Behavior is *what an agent actually does* in an operational context being studied. The controller manages this aspect through a shared clock in a discrete-event simulation that updates all agents periodically to execute the API and student code in the models.

An example is landing an unmanned aircraft, which flies to the circle and then, depending on the approach angle, executes one of three series of maneuvers to intercept the runway.

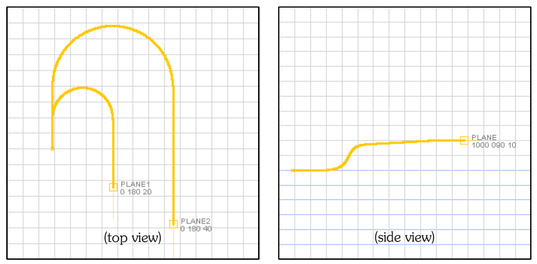


The simulation also manages logging agent states for controlled scientific experiments as a stochastic Monte Carlo methodology that promotes formal evaluation of model performance.

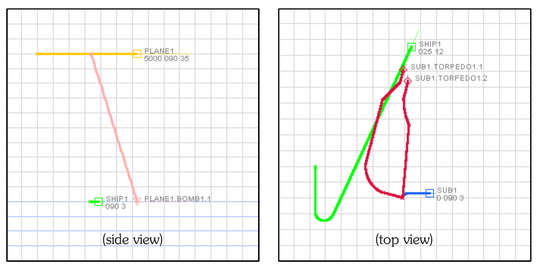
View

Visual inspection is often the most intuitive way to assess the performance of a model in terms of realism and consistency with real-world expectations. The API provides multiple forms.

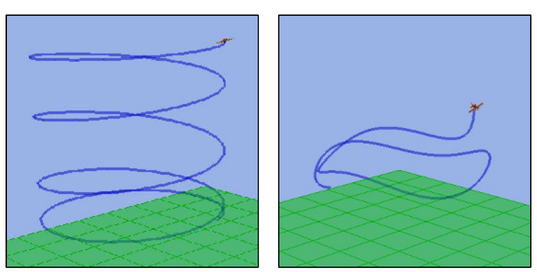
The native two-dimensional viewer presents real-time top, front, and side views. For example, the turning radius of an aircraft varies according to its speed, and altitude changes show a constant rate of climb with smooth transitions.



Although physics is not explicitly modeled, many realistic expectations are naturally captured by the event system. For example, a bomb dropped from an aircraft has both downward and forward movement. Similarly, the reaction of torpedoes chasing a ship lags due to their defined processing delay and turning capabilities as the ship makes evasive maneuvers. They also momentarily confuse each other as targets.

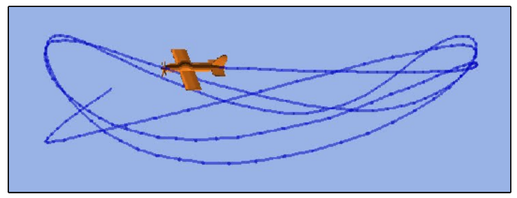


An external Java 3D viewer provides richer post-execution visualization [3]. For example, differences in student solutions of an aircraft attempting a climbing turn are apparent.

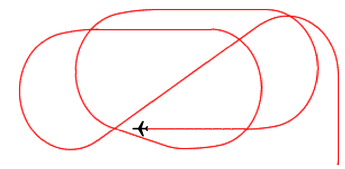


View

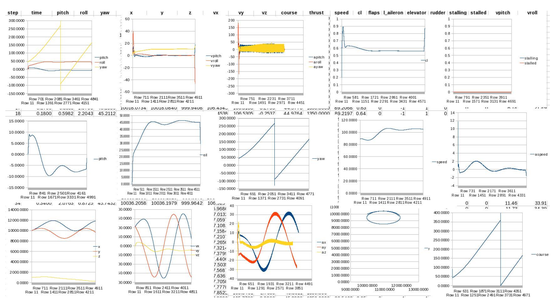
The 3D viewer also accommodates a variety of metainformation about agent execution. The breadcrumb track here shows not only the aircraft path, but the dots indicate the discrete-event positions it was actually calculated to be at.



The same position information can be exported to Gnuplot for post-execution viewing and manipulation with its rich processing features. Here is an aircraft holding pattern.



Finally, all API event data, as well as selected student data, can be exported to Excel in comma-delimited format for presentation in chart form.



References

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

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

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

dtappan@ewu.edu