Game-Related Examples of Artificial Intelligence

Ken T. N. Hartness

Sam Houston State University
SHSU Box 2090, Huntsville, TX 77341
hartness@shsu.edu

Abstract

The field of artificial intelligence needs to attract new researchers to the field to continue current explorations and look for novel approaches to tomorrow's problems. One approach involves providing students with learning tools that excite their imagination and help them obtain an appreciation for what artificial intelligence can do. The tools described here are used in an undergraduate course at Sam Houston State University. They include heuristic-driven search in a potential game's terrain map, reinforcement learning in a tank battle game, and game tree search techniques in tic-tac-toe.

Introduction

An artificial intelligence course often lacks the time to explore anything other than toy problems. Even when these toy problems fail to excite the imagination of students, the discussion of a particular area of artificial intelligence can be greatly enhanced by an implementation of the solution, something that a student can adjust and examine the impact of certain choices on how well the system solves its problem. If the goal is to simply demonstrate what a technique can accomplish, implementations of complex practical problems could be used to interest students in the power of the technique even if the implementation is not easily comprehended.

The author hypothesizes that students will learn the implications and purpose of different areas of artificial intelligence more thoroughly with hands-on activities. In order to limit the time spent by students on one particular topic in a course that touches on numerous aspects of artificial intelligence, the author seeks prototypes, partially-implemented applications, and full, interesting applications that touch on such material as path planning, genetic algorithms, expert systems, and reinforcement learning. This paper describes tools currently used by the author for this purpose.

Philosophy

Computer science enrollment has dropped for many programs. Students interested in artificial intelligence represent a subset of the decreasing computer science student population. Russell et al (2007) suggest that one reason for this decline is that students do not truly comprehend what computer science is or the opportunities to which it may lead. Computer science educators need to make a special effort to teach courses that remind students of the interesting possibilities that may await them in industry or academic research. Artificial intelligence, in particular, has a potential for helping students consider opportunities other than coding jobs, and instructors should take advantage of this opportunity.

Caspersen and Bennedsen (2007), in trying to describe the important qualities of a programming course, concluded that most students prefer learning from examples. Evidence exists to support the idea that learning is improved if a mixture of examples and student work is used, and partial examples that must be completed by the student seemed to be a powerful means of teaching concepts. A number of attempts have been made to provide a learning environment for artificial intelligence. One, Flexible Learning with an Artificial Intelligence Repository (FLAIR), provided instructors with insight into a failure on the part of students to comprehend or apply some of the material from the lecture (Ingargiola et al, 1994). The students were unable to predict the behavior of a path-planning algorithm that found a path between cities in spite of knowing the different heuristic-based cost-estimation functions being used with the A* algorithm. The use of examples, partial or complete, can be very valuable to the students' learning process. By including examples that have significant practical use or are entertaining, interest in the subject is enhanced as well.

Examples Currently Used

I teach my introduction to artificial intelligence course in two parts. The first part covers classical search techniques that utilize heuristics to create estimates of cost or quality with regard to an intermediate step in obtaining a solution; this section covers the A* algorithm, hill-climbing search, and minimax search. The remainder of the course provides an overview of interesting artificial intelligence topics ranging from the more static expert systems to machine learning techniques like genetic algorithms and reinforcement learning. Students are required to explore a topic in more detail as a semester project, but, otherwise, there is little time to explore any one topic in depth.

One tool used to explore A* search is the terrain map explorer. The A* algorithm can simulate skill and intelligence on the part of a computer-controlled character in a game by locating the most efficient path to a destination. The terrain map explorer represents a computer-controlled character attempting to reach a castle in a strange land. The land is represented as a two-dimensional array of tiles where each tile represents a type of terrain; some types of terrain can be crossed quickly (inexpensively), and other types are more difficult to cross (more expensive). The terrain object only allows access to terrain information as a 3x3 grid around a specified position (the tile at that position and its neighboring tiles). The character maintains its own map of explored terrain and its current tile within that terrain. Because undergraduate students often mistakenly consider the cost of backtracking, one version of this tool uses the A* algorithm on the character's incomplete map; this search process is used to generate a path to where the character assumes the goal may be located. As the terrain map is explored, new paths are calculated based on the new information. The A* algorithm is seen, here, as a supporting tool for considering possible paths rather than directly driving the character's exploration of the terrain. The tool is designed so that a student only has to write a function, h, for estimating the cost of reaching a goal.

A tic-tac-toe program provides a simple example of a two-player, turn-oriented game. The instructor can remove the minimax algorithm and assign its implementation to the student, change two lines and require that the student implement alpha-beta pruning and compare the efficiency, or simply ask the student to provide the evaluation function to implement appropriate heuristics for playing the game. Students are encouraged to consider the difficulty of creating a good evaluation function that performs well at lower look-ahead values or increases pruning.

Students learn about reinforcement learning with the aid of an open-source game developed to encourage users to learn Java. Robocode (Nelson, 2001) is a program written in Java that simulates robot tanks shooting at one another as they move across the playing field. Each robot's control software is implemented as a Java class. Students are free to create their own robots independently of those based on reinforcement learning algorithms and allow them to compete on the battlefield. A reinforcement learning algorithm called Q learning (Dean et al, 1993) has been written as a Java class, and a robot class called QRobot (Hartness, 2004) has been defined for the purpose of applying positive and negative reinforcement to the Q learning model as it successfully attacks and destroys other robots or is attacked, itself. A file of learned data is maintained under the robot's name, allowing it to learn across sessions. Students must determine how to associate positive and negative reinforcement with events during the battle; fortunately, the robots are notified of relevant events.

A version of the Robocode robot has also been designed to work with a genetic algorithm. The robots load control parameters from a server, then send an evaluation of their performance back to the server. Because of limited population size and the fact that an entire battle must be fought before a new generation can be explored, I have preferred to have students work with a simple genetic programming implementation, although the robots are available for independent student projects. One could conceivably have students pit their own creations against the evolving robots throughout the semester.

Conclusion

Examples are a powerful tool for enhancing student learning. With limited time to create our own examples for every area of artificial intelligence, A.I. educators should be willing to share interesting examples that seem to be successful at clarifying a concept for students. Mine are at www.shsu.edu/~csc_kth/cs582/.

References

Casperson, M., and Bennedsen, J. 2007. Instructional design of a programming course – A learning theoretic approach. *ICER '07: 3rd International Workshop on Computing Education Research*, September 2007.

Dean, T., Basye, K., and Shewchuk, J. 1993. Reinforcement learning for planning and control. 67–92. Minton, S., ed. *Machine Learning Methods for Planning*. San Mateo, CA: Morgan Kaufmann.

Hartness, K. 2004. Robocode: Using games to teach artificial intelligence. *Journal of Computing Sciences in Colleges* 19(4): 287–291.

Ingargiola, G., Hoskin, N., Aiken, R., Dirbey, R., Wilson, J., Papalaskari, M., Christensen, M., Webster, R. 1994. A repository that supports teaching and cooperation in the introductory artificial intelligence course. In *SIGCSE '94: 25th SIGCSE Symposium on Computer Science Education*, 36–40. New York, NY: ACM.

Nelson, M. 2001. Robocode. http://robocode.alphaworks.ibm.com/home/home.html.

Russell, J., Russell, B., and Pollacia, L. F. 2007. Reversing the Decline of CIS Enrollment in Colleges and Universities by Creating Viable and Attractive Minors in CIS: A Statistical Study of CIS Minors at US Colleges and Universities. In *The Proceedings of ISECON 2007*, v 24 (Pittsburgh).