AC 2012-5090: ENHANCING EXPERTISE, SOCIABILITY, AND LITERACY THROUGH TEACHING ARTIFICIAL INTELLIGENCE AS A LAB SCIENCE

Prof. Stephanie Elizabeth August, Loyola Marymount University

Stephanie August is an Associate Professor and Special Assistant to the Chief Academic Officer for Graduate Education at Loyola Marymount University, Los Angeles. She teaches courses in artificial intelligence, database management systems, and software engineering. Her research interests include applications of artificial intelligence including interdisciplinary new media applications, natural language understanding, argumentation, and analogical reasoning. She has several publications in these areas. August is actively involved in the scholarship of teaching and learning community and is a 2006 CASTL Institute Scholar (Carnegie Academy for the Scholarship of Teaching and Learning). She is currently directing graduate and undergraduate students on two NSF-funded projects, to develop materials for teaching artificial intelligence through an experimental approach modeled after the lab sciences, and to develop a Virtual Engineering Sciences Learning Lab in Second Life to provide an immersive learning environment for introductory engineering and computer science courses. Her industry experience includes software and system engineering for several defense C3I programs, and applied artificial intelligence research for military and medical applications.

Enhancing Expertise, Sociability and Literacy through Teaching Artificial Intelligence as a Lab Science

Abstract

Artificial intelligence and software engineering course material can be interwoven and presented in a lab experiment paradigm to provide experiential learning opportunities in which students collaboratively solve problems. This approach has the potential to increase retention of women and non-traditional computer science students in computer science courses, while reinforcing best practices in software engineering.

Overview

The Teaching Artificial Intelligence as a Laboratory Science[†][1] (TAILS) project is designed to develop a new paradigm for teaching introductory artificial intelligence (AI) concepts by implementing an experiment-based approach modeled after the lab sciences. It explores whether structured labs with exercises that are completed in teams before students leave the classroom can build a sense of accomplishment, confidence, community, and collaboration among students, characteristics which have been shown to be critical to retain women and non-traditional computer science students in the field.

TAILS presents to students an array of fundamental AI algorithms as a set of hands-on activities made available through a database of lab activities, including software exercises and experiments that provide experience with concepts from multiple perspectives and multiple modes of representation. Best practices in software engineering will be reinforced in students through careful design and documentation of the modules.

The proposed activities are designed to engage the kinetic learner and provide the "big picture" that model-driven learners need to assimilate course material. Existing research has shown that structured labs with exercises that can be completed before students leave the classroom build a sense of accomplishment and confidence[2,3]. Progressively sophisticated experiments teach inexperienced students and challenge more advanced students.

TAILS contributes two components to STEM education: a set of lab experiments to promote student retention of concepts and retention of majors, and insight into student learning through the labs. TAILS contributes to exemplary STEM education by creating learning materials and strategies, implementing new instructional strategies, and assessing and evaluating student achievement.

This paper describes the components, presents an example from adversarial search, and identifies a mapping of outcomes and objectives to assessments.

[†] This material is based upon work supported by the National Science Foundation under Course, Curriculum, and Laboratory Improvement (CCLI) Grant No. 0942454. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Components of TAILS Lab Experiments

TAILS will deliver the tale of each AI algorithm or concept through a story with nine parts, including a description of the concept, relevant applications, sample test data, design description, exercises that guide the student in implementation, a test driver, suggested experiments, source code that implements the algorithm, and complexity analysis. This choice of components is patterned after the organization found in the files of software support that accompany Winston's approach[4] and standard software engineering practice. Previous work[5] identified components that model for the student an array of abstractions which they can use for presenting an algorithm or concept, each geared toward a different audience.

The Idea provides an overview of the concept to be presented, a functional description that avoids implementation details. This component contains references to additional sources of information on the topic. When an algorithm is presented in the context of an application, such as a game that uses heuristic search, the rules of the game will be included to orient the user. This description of the concept is especially well suited to the non-computer scientist or when a general introduction to the concept is needed. Students will learn that an extended version is appropriate when the focus is on the big picture, while a minimal version (one-sentence) would preface a presentation of additional information, such as a UML model of the algorithm.

The ***Applications*** section sets the algorithm in context and provides descriptions of real world applications that use the concept. Graphics, such as a soccer-playing robot or a game board and related references support the visual learner, and provide additional avenues of exploration. Knowledge of applications enables students to explain the significance of the algorithm to a general audience.

Sample Input/Process/Output contains an annotated trace of the program in execution, including system input, a description of the processing taking place, and display of the resulting output. Interactive demonstrations of the concept will be included where feasible to allow the student to run live demonstrations, as well as experiment with various forms of input. Sample Input/Process/Output (IPO) corresponds to the concept of operations and user manuals, and supports black box testing of the concept using the related source code. A *readme* file describing the procedure for running the demonstration will be included. An IPO example is useful for establishing the scope of program, and as the preface to a detailed design description.

Implementation-independent Design Description provides an abstract, high-level view of the system implementing the algorithm. Both textual descriptions and diagrams allow the user to explore the design in a top-down manner. This description is language-independent for clarity and longevity. Programming languages used to implement algorithms and programming environments change far more rapidly than the ideas implemented and TAILS code is not restricted to a particular software platform. Having the design in addition to the code enables the student to evolve a system at the design level, rather than at the code level. A developer would present this model when reviewing the overall design of the algorithm before a technical manager or as a preface to a review of the actual code.

Implementation-specific "HINT" File(s) contain part of the code needed to implement an algorithm or concept in a particular programming language, and guidelines or HINTs the user can use to implement the remainder of the code. For example, a program for visiting the nodes of a tree would be provided with the enqueing technique stubbed out. The student would be instructed to write the code that would transform the program into one that does a depth-first, breadth-first, or non-deterministic search. A hint file for at least one language will be included for each concept and will include file header and comment blocks, and follow an established coding style, reinforcing coding standards. A *readme* file describing the procedure for compiling and running the student's code will be included for each HINT file.

Test Suite and Driver(s) are provided for each implementation-specific HINT file that will run the student's program and test it using the data in the test suite. The student can then compare the expected results with the results generated. At least one of the tests will correspond to running the example in the IPO discussed above. A *readme* file describing the procedure for running the test will be included for each driver along with directions for downloading the specific programming environment needed, since TAILS source code will be written for a variety of platforms.

Experiments give students a starting point for interacting with both the concept and the code. This section lays out the preparation needed to complete the experiments and provides a set of increasingly complex tasks for students to complete collaboratively in pairs. Some tasks can be completed by students with any level of sophistication during a single class period, while others are suitable as course projects for advanced students. The experiment assignments include an implementation-*independent* set of test data and expected results, as well as ideas for enhancements and extensions, and go beyond the basic testing outlined for the test driver above. They also include an implementation language-*dependent* test driver with trial input and expected output for the simpler experiments, which is useful to verify that the code has been correctly implemented. The HINT files, test drivers and experiment drivers will be repeated for each implementation. For example, there might be a Java version, a C++ version, and a Common Lisp version of an implemented concept.

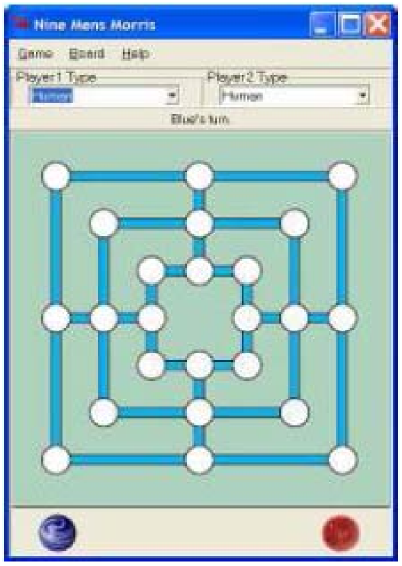
Source Code will be provided because learning by example is a powerful paradigm for mastering a new subject. We offer the ending of the tale -- solutions to the exercise in the HINT files, as well as more extensive implementations readily available from other sources. Many students, especially non-majors, benefit from having solutions provided in order to fully understand the material and gain a sense of accomplishment as they experiment with the code. In the case of an algorithm implemented within a game, having access to a fully functioning version of the game will facilitate learning for those students not familiar with the game. Majors have ample opportunity to write original code in software engineering, database, and capstone courses. Providing both the implementation-independent design description with the corresponding source code models best practices in software engineering. Providing an executable version of a program, such as a game that uses heuristic search, allows students to understand its functionality first hand before trying to implement it.

Complexity Analysis complements the work done in a data structures or algorithms class, and reveals the various ways that complexity can be measured for this particular problem. Students can analyze the changes they have made in the experiments measuring, for example, time to execute vs. memory required. A developer presents the complexity analysis to an audience considering whether the design meets performance requirements.

Sample Course Module: Adversarial Search - Implementation of the Minimax Algorithm for Nine Men's Morris [6]

The *Tale* of the minimax algorithm is told in the context of the zero-sum game Nine Men's Morris (NMM), a strategy board game using adversarial search. Highlights of module components are summarized here.

The Idea provides an overview of the minimax algorithm, alpha-beta pruning and heuristics. These are illustrated with examples from tic-tac-toe. The history of Nine Men's Morris (NMM), which traces its roots back to 2000 BCE, gives the student a context for the game, and outlines its rules and strategies for play. Details of the human machine interface (HMI) for NMM appear in figure 1.



Player1 Type: Human
Player2 Type: Human
Blue's turn

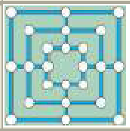
Game Board Details	
	The board consists of three concentric squares with the faces of each square connected by four intersecting lines. Each intersection is a space on the board that can be occupied by one of either player's pieces.
○	A space on the board that can be occupied by one of either player's pieces.
━	An intersecting line that shows the paths that a players piece can take from one space to another
🔵	A blue piece. Blue is always player 1. When it is the blue players turn, they can drag-and-drop this piece to any valid space on the board according to the rules of the game. For details describing the rules and how to play the game see Section 2 (Background) of the idea statement. When the red player creates a mill they can remove any blue piece that is not in a mill by clicking on that blue piece.
🔴	A red piece. Red is always player 2. When it is the red players turn, they can drag-and-drop this piece to any valid space on the board according to the rules of the game. For details describing the rules and how to play the game see Section 2 (Background) of the idea statement. When the blue player creates a mill they can remove any red piece that is not in a mill by clicking on that red piece.

Figure 1. Nine Men's Morris HMI and game board details.

Several ***applications*** of minimax with alpha beta pruning are described, conveying a sense of its broad range of usefulness. These include the automated chess player Deep Blue, Envelope Constrained Filters used in radar pulse compression and real-time pursuit evasion algorithms.

Sample Input/Process/Output illustrates the user interface for NMM game play and modes of play (human vs. human, human vs. computer, computer vs. computer) and briefly describes the reasoning used to make automated plays.

Implementation-independent Design Description includes system overview and architecture diagrams, as well as class and sequence diagrams using the Unified Modeling Language, along with narrative to augment the diagrams. Figures 2-4 provide examples of the code and data design documentation.

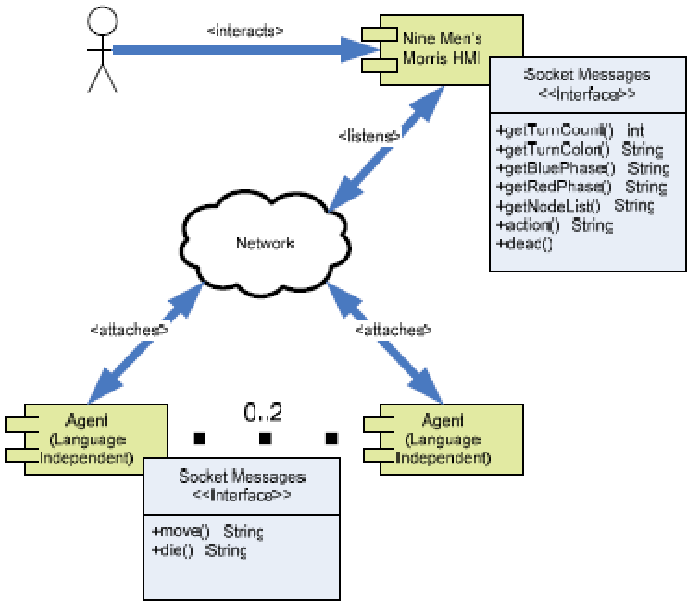


Figure 2. System overview diagram focused on HMI.

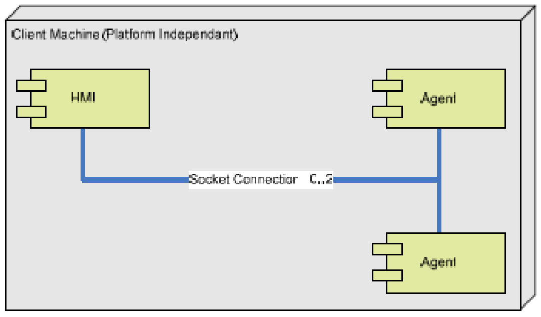


Figure 3. Software architecture diagram

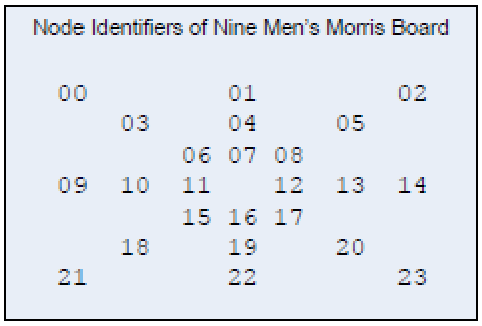


Figure 4. Map of nodes on NMM game board.

Implementation-specific "HINTs" corresponding to each of the experiments outline the details of code changes required by the exercises and offer illustrative snippets of code to guide the student.

Test Suite and Driver(s) describes and identifies test preparations, descriptions, and the process used to test the NMM minimax agent and its graphical user interface (GUI). The test environment is defined, including details about workstation requirements, software requirements, and the test environment setup. Test descriptions which define prerequisite conditions, test inputs, expected test results, criteria for evaluating results, the test procedure, and assumptions and constraints for testing are provided. Lastly, the features to be tested and those which are not tested are identified.

The ***Experiments*** section as shown in figure 5 identifies the prerequisite knowledge and preparation and provides an overview of experiments that range from learning how to play NMM to enhancing its performance. Figure 6 provides additional detail for experiment #2. Students complete a subset of the exercises according to the course in which the module is used, their maturity as a programmer, and course requirements.

Well-designed and commented ***Source Code*** includes an implementation for the GUI, game board, and computer player agent.

Complexity Analysis reviews the time and space requirements for the minimax algorithm and minimax with alpha-beta pruning.

Pre-lab preparation Read relevant textbook section on adversarial search; attend classroom presentation on and discussion of the topic; study TAILS module overview, which includes a brief discussion of the minimax algorithm and an overview of the Nine Men's Morris (NMM) board game; read TAILS description of applications of minimax to chess, Envelope Constrained Filters used in radar pulse compression, and pursuit evasion algorithms; study the TAILS sample input/process/output and the implementation-independent design description which provides use cases, requirements for the related software components, and UML diagrams of the code. Meet with lab partner to preview the TAILS exercises and implementation-specific hints for implementing segments of the code.

Lab experiment #1: Becoming Familiar with Nine Men's Morris (30 min.) Become familiar with the NMM game by playing a version online with classmates.

Lab experiment #2: Baseline the NMM Agent (30 min.) Play against the NMM computer player agent and collect statistics on performance agent to use as baseline in subsequent experiments.

Lab experiment #3: Alpha-beta Pruning and Performance (60 min.) Enhance agent's source code by adding alpha-beta pruning to the minimax algorithm provided. Compare performance to baseline.

Lab experiment #4: Importance of Evaluation Function (60 min.) Improve the evaluation function implemented in experiment #3. Compare performance to baseline.

Lab experiment #5: Taking Advantage of Symmetry (16 hours - class project) Add a state generator to omit symmetrical states from the generated successor states. Improve the evaluation function implemented in experiment #3. Compare performance to baseline.

Lab experiment #6: Caching States (16 hours - class project) Add a state generator to cache successor states until branches they belong to are no longer needed. Compare to baseline.

Post-lab wrap-up Students compare code and results, prepare a lab report that includes complexity analysis of their work and the entire game. Results are compiled into a written lab report and discussed in class. Students complete relevant assessments.

Figure 5. Nine Men's Morris experiments for studying adversarial search.

Lab experiment #2: Baseline the NMM Agent (30 min.) Play against the NMM computer player agent and collect statistics on performance agent to use as baseline in subsequent experiments.

Description

In this experiment you will be playing several games against the provided adversarial agent. This agent implements the basic MINIMAX algorithm with a depth cut-off at level six. For details about the MINIMAX algorithm see the Background Section located in the Laboratory Idea. The purpose of this experiment is to baseline the performance of the provided adversarial agent. This baseline will then be used in the follow-up experiments to show either performance improvement or degradation after modifications are made to the agent's source code.

Hypothesis

Write your hypothesis stating the outcome you expect to see when you play against this agent ten times. Your hypothesis should have the following format: "I think that if I … then …".

Materials Needed

- this laboratory guide;
- a clean copy of the source, and;
- a computer with the Tcl/Tk Interpreter installed.

See the Test Environment section of this laboratory guide for details about the environment required to run this experiment.

Setup

There is only one step needed to setup this experiment: Copy the provided Nine Men's Morris source code to a scratch area.

Procedure

Play against the agent at least ten times. Record the results of the games you play against your modified adversarial agent and record the result using the table provided.

Observations

Record the results of the games you play against your modified adversarial agent and record the results again.

Conclusions

- What conclusions can you make about this experiment?
- Did you prove the hypothesis correct?
- How would you improve this experiment?
- How can the hypothesis be applied?

	Win	Tie	Loss
Game 1			
Game 2			
Game 3			
Game 4			
Game 5			
Game 6			
Game 7			
Game 8			
Game 9			
Game 10			
Totals			

Figure 6. Details of Experiment #2.

TAILS Outcomes, Objectives, and Assessments

The TAILS project addresses learning outcomes in five categories: skills, concepts, communication, application, and research. The learning outcomes, specific objectives for each outcome and their planned assessments are shown in table 1. We will rely in part of the Dewar-Bennett Knowledge Expertise Grid[7] to analyze our data. The grid defines criteria for summative evaluation that can be adapted for evaluating knowledge of engineering-related content and rates a student's affective and cognitive knowledge in terms of the student's level of expertise. We are in the process of defining the rubrics that can be used to grade student work and assess the outcomes in a reliable manner. Scoring rubrics will be developed for products produced by students, student writing, and open-ended responses on exams. Information learned about student accomplishment of the outcomes will be used to improve the course, both as it is in progress, and for future offerings of the course.

Table 1. TAILS learning outcomes.

Category	Outcome	Objective	Assessment
Skills	Students will demonstrate the ability to solve problems collaboratively	Student will demonstrate collaboration and teamwork skills	Students will work in pairs to complete the lab activities, then: • Complete a teamwork attitude questionnaire[8] • Write a team process log to record perceptions about collaboration[8]
Concepts	Students will demonstrate knowledge of artificial intelligence concepts	Students will demonstrate recall and general understanding of AI concepts	• Answer exam questions • Complete pre- and post-tests • Explain and write software code • Draw a concept map[9 p.197-202]
		Students will demonstrate a deep understanding of course concepts	• Contrast multiple concepts[9 p.168] • Define and give one example of a course concept[9 p.38]
	Students will demonstrate knowledge of software engineering practices	Students will demonstrate proficiency in software engineering practices at background-appropriate (grade- and major-appropriate) level	• Specify requirements for a software program • Complete a domain-level design for a software program • Design an algorithm at an implementation-specific level • Reverse engineer software for an algorithm
Communication	Students will be able to describe course concepts at multiple levels of abstraction	Students will be able to describe course concepts clearly and without technical jargon	• Write an elevator statement[9 p.183-187] geared toward the student's grandmother to describe the concept
		Students will be able to describe course concepts for a classmate or technical manager	• Write an algorithm in pseudocode to describe the concept for a technical manager
Application	Students will be able to identify applications of AI concepts	Students will be able to identify real world applications for AI concepts beyond those provided in course materials	• Complete application cards[9 p.236-239]
Research	Students will demonstrate curiosity about course material	Students will demonstrate the ability to extend course concepts	• Describe one new experiment that can be used in conjunction with each algorithm studied; explain the objective of the experiment and why this is a worthwhile objective • Describe one enhancement to the algorithm studied and explain why the enhancement is worthwhile

Conclusion and Future Work

TAILS contributes to the STEM education knowledge base by promoting individual efforts to solve a programming assignment while building an education community through laboratory work that encourages cooperation and teamwork among students. The paradigm can be adapted to computer science courses at all academic levels and is expected to increase participation in the field by shortening the time required to prepare undergraduates to engage in research.

Computing and software are ubiquitous. There is a compelling need for software engineering education in computer science[10,11] and engineering[12,13,14,15], as well as animation, biology and other disciplines in which computing plays an ever increasing role. The TAILS model demonstrates a technique for integrating software engineering concepts that can be used in computing-intensive courses beyond traditional computer science programs.

Alpha testing is underway on the initial version of the adversarial search/Nine Men's Morris module. Work has begun on developing course materials for unification, basic and informed search and conceptual clustering algorithms and we continue to define rubrics used to grade student work and assess outcomes in a consistent manner. Future considerations include the possibility of building upon laboratory projects developed as part of the Machine Learning Experiences in AI framework[16] or the Model AI Assignments presented at the Symposium on Educational Advances in Artificial Intelligence[17].

References

[1] August, Stephanie E. CCLI: Enhancing Expertise, Sociability and Literacy through Teaching Artificial Intelligence as a Lab Science. NSF Grant no.0942454, 2010.

[2] Beyer, S., Rynes, K., Perrault, J., Hay, K., Haller, S. Gender differences in computer science students. *SIGCSE '03*, 2003, pp.49-53.

[3] Strok, D. Women in AI. *IEEE Expert*, *7*:4, August 1992, pp.7-22.

[4] Winston, Patrick Henry. *Artificial Intelligence*. 3rd edition. Addison-Wesley, Reading MA, 1992.

[5] August, S.E. Integrating Artificial Intelligence and Software Engineering: An Effective Interactive AI Resource... does more than teach AI. In Mehdi Khosrowpour (Ed.), *Proceedings of the 2003 Information Resource Management Association International Conference*). Hershey PA: Information Resource Management Association, 2003, pp. 17-19.

[6] Shields, Matthew. *Adversarial search: An implementation of the minimax algorithm for Nine Men's Morris.* CMSI 677 class project, LMU, spring 2009.

[7] Dewar, Jackie and Bennett, Curtis. 8-dimensional Mathematical Knowledge-expertise Grid. http://myweb.lmu.edu/carnegie/webport/knowgrid.htm, Loyola Marymount University, 2004. (last accessed 10 January 2012)

[8] *OERL: Online Evaluation Resource Library*. http://oerl.sri.com/home.html (last accessed 10 January 2012)

[9] Angelo, Thomas A. and Cross, K. Patricia. *Classroom Assessment Techniques; A Handbook for College Teachers*. 2nd edition. San Francisco: Jossey-Bass, 1993.

[10] Pour, Gilda; Griss, Martin L.; and Lutz, Michael. The push to make software engineering respectable. *Computer*, May 2000, pp.35-43.

[11] Lethbridge, Timothy C. What knowledge is important to a software professional? *Computer*, May 2000, 44-50.

[12] Long, L.N. The Critical Need for Software Engineering Education. *CROSSTALK, The Journal of Defense Software Engineering*, 10(1), January 2008, pp.6-10.

[13] IEEE Computer Society and the ACM. “Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.” http:// sites.computer.org/ccse/SE2004Volume.pdf. 2004. (last accessed 10 January 2012)

[14] Sanders, P. Improving Software Engineering Practice. *CROSSTALK, The Journal of Defense Software Engineering*, January 1999, pp. 4-7.

[15] Vaughn, R. Software Engineering Degree Programs. *CROSSTALK, The Journal of Defense Software Engineering*, *13*(3), March 2000, pp. 7-9.

[16] MLeXAI: Machine Learning Experiences in AI: A Multi-Institutional Project. NSF DUE 0716338. http://uhaweb.hartford.edu/compsci/ccli/index.htm (last accessed 11 January 2012)

[17] Model AI Assignments, Symposium on Educational Advances in Artificial Intelligence. http://eaai.stanford.edu/ (last accessed 11 January 2011)