Experiences with Artificial Intelligence as an Undergraduate Creative Capstone Course

V. Scott Gordon
CSU, Sacramento
gordonvs@ecs.csus.edu

Abstract

This paper introduces a model called the "creative capstone" for teaching Artificial Intelligence at the undergraduate level. The model leverages AI's image as a creative endeavor, while mitigating its image as a less practical option. The course was implemented and taught five times, and the resulting experiences are described. Specific techniques for helping ensure the success of the course are also detailed.

Introduction

One of the challenges facing Artificial Intelligence (AI) education is that it is subject to the ups and downs of the field. Another is that it is a vast area, and people disagree on what constitutes its core topics. An approach for teaching AI is needed that can accommodate differing opinions while remaining appealing.

This paper describes a model for teaching Artificial Intelligence at the undergraduate level in a different way than is generally done. The proposed model, dubbed *creative capstone*, leverages AI's image as a creative endeavor. It was taught this way at two universities with continuing success and numerous benefits.

Sections 2 and 3 outline the motivating factors that led to the development of the course in its current form. The course itself is defined in section 4. A number of lessons were learned over the past five years when the course has been offered - these and other observations are given in section 5.

AI in Computer Science Curriculum

Artificial Intelligence as a field of study enjoys, or rather suffers from, a sort of cyclical popularity in which it is either all the rage, or the kiss of death. The term "AI winter" has come to epitomize popular attitudes towards AI during the downswings, when funding for AI projects becomes difficult and students choose other options. Whether we are currently in an AI winter or an AI *spring* is a matter of debate [Mar06, Hav05].

Throughout the ups and downs, the field of Artificial Intelligence has expanded greatly, to the extent that it has become impossible to survey within one semester. The preface to Russell and Norvig's textbook *Artificial Intelligence, a Modern Approach*, considered by many to be the standard academic AI text, unabashedly states: "...working through the whole book requires a two-semester sequence" [RN03]. Other texts admit to focusing on a subset of the field – Jones on weak AI, Coppin on practical and methods derived from "natural" systems, and Negnevitsky on systems suitable for a large professional audience [Jon05, Cop04, Neg02].

Thus computer science departments find themselves in a quandry – how, and at what level, to teach AI? Admittedly, departments with strong AI research programs are free to offer multi-semester comprehensive AI course sequences at the graduate level. But institutions with M.S. as terminal degree may be hard-pressed to populate so many units from a pool of students faced with options considered more practical.

And what of the undergraduate level? Given the virtual impossibility of offering a comprehensive one-semester survey course, what role can AI play in undergraduate CS curricula? The IEEE/ACM guidelines for Computing Curricula are of little help, listing only 9 hours of core material comprising search, constraint satisfaction, knowledge representation and reasoning, and an hour of "fundamental issues" [IA01]. The implication is that these topics would need to be covered in, say, an algorithms course, or included in a larger required AI course – the latter being difficult to justify.

AI at the Undergraduate Level

AI has been described as having "no clear core" [KW95]. Given the continuing perception of AI winter, and the increasing difficulty in defining what exactly would constitute a standard undergraduate AI course, many departments have little recourse but to consider AI as a sort of fun elective. It naturally follows that decisions of course content are then left largely to the individual professor. In perusing undergraduate AI courses at various universities, it becomes clear that instructors simply

choose the subset of Artificial Intelligence topics that interest them, or that they are comfortable with, or that they themselves have engaged in research.

While there is nothing in and of itself wrong with course content being determined on an instructor-by-instructor basis – graduate courses commonly work this way – at the undergraduate level it can lead to a perception that the course itself is not serious..

Still, educators commonly utilize (and even cite) AI's inherent possibilities for providing capstone material [Kum99]. Students at Universities which require undergraduate capstone projects often choose AI-related projects [PU07]. However, departments would be understandably hesitant to choose AI as their *required* capstone experience, considering the overwhelming demand for more direct industry-related experiences in networking, web services, software engineering, etc. While a structured AI capstone may therefore not be feasible, leveraging those desirable aspects of the AI capstone experience into an AI course can help improve its focus, value, and attractiveness. It also provides additional avenues for promoting the course and its context within the department's curricular structure.

AI as a Creative Capstone Course

Clearly, AI's image suffers with respect to its perceived practicality. However, even in the darkest of times it retains a positive image with respect to creativity. With the undergraduate student facing an increasingly dizzying array of technologies to learn over the course of a typical B.S. degree, opportunities for creativity are scant, and herein lies an opportunity for AI to find its niche, defined here as *creative capstone*.

Creative capstones exist in other disciplines, typically in the liberal arts [Swa03]. Computer Science and Engineering are often loathe to build courses with highly subjective assessment criteria – AI offers a vehicle for a more open-ended and subjective experience. Whether or not a department chooses to officially endorse a creative capstone as part of its curricular structure, a suitably constructed AI course can offer students the option of a curricular capstone experience – even unofficially. If it succeeds, it can become a popular and valued destination course. As students approach graduation, they become acutely aware of their portfolio, and seek opportunities to set themselves apart not only for their knowledge, but also for their unique achievements. They understand that employers seek practical skills, but also want to know what projects they have done, what original work they have done, and what has excited them.

Another advantage of structuring the undergraduate AI course as a creative capstone is that it separates the focus of the course, to some degree, from the actual topic list. That is, rather than the course being defined as a list of AI subjects, it is instead defined by the nature and number of project(s) included, and student expectations with respect to originality, technical depth, etc. Individual instructors are then free to emphasize those topics they feel most qualified in providing subjective assessment and creative guidance.

Structure of the Course

The course described in this paper was offered at two schools: Sonoma State University (SSU) and Sacramento State University (CSUS). Both are typical California State Universities (CSU) which terminate at the Master's degree. Since they do not offer PhDs, they are less research-oriented (than Universities that do), and have students with a wide spectrum of backgrounds, goals, and preparedness.

At SSU there already existed an undergraduate AI course, which for the Spring 2002 semester was restructured as a creative capstone. The instructor later moved to CSUS, where there was no undergraduate AI course, and proposed an identical course be added. Thus was formed CSc-180, titled Intelligent Systems.

Four topics were chosen to expose students to a variety of AI modalities and project experiences:

1. Knowledge Engineering
2. Adversarial competition
3. Machine Learning
4. Problem Solving from Nature

Each area affords a variety of possible particular methods and projects. Specific topics chosen for CSc-180 that correspond to the above areas are shown in Figure 1.

AREA	PROJECT
1. Knowledge Engineering	Develop rule-based expert system, using human expert, in an area student knows nothing about.
2. Adversarial competition	Develop minimax-based program for a provided game. Tournament among student programs is held.
3. Machine Learning	Use supervised neural-network learning on a real-world data set developed by the student.
4. Problem solving from nature	Develop genetic encoding for a student-chosen problem, and use genetic algorithm to solve it.

Figure 1 – Undergraduate course projects

Students spend 3-4 weeks on each project, and are required to write project reports on each one except #2. Projects 1 and 3 can be done in teams of two. Semester grades are based primarily on the project reports, and to a lesser extent on a single midterm and final exam.

Additional Project Details

Project 1 is done using an off-the-shelf expert system shell such as Fuzzy-CLIPS or JESS. Students are required to identify a domain expert in an area they themselves are not familiar with, and proceed through the knowledge engineering process. Each student works on a different project.

In Project 2, students are given a board game to implement using the typical minimax/alpha-beta algorithm. Everyone implements the same game, and are encouraged to add typical optimizations as time permits, such as quiescence, bit-mapping, transposition tables, etc., as well as their own static evaluation function. A public tournament event is held at the end of the project with students playing their programs against each other. Winners from each year are displayed on a web page.

Project 3 is done using genetic algorithm code provided to the students, although students are allowed to implement the code in other ways. They are also required to implement their own genetic operator, and describe the analogous natural process on which it is based.

Project 4 is done using neural network code provided to the students, although students are allowed to implement the code in other ways. A big part of the project is building the training and test data for a problem identified by the student. Each student works on a different project.

Alternatives

As structured, the four areas lend themselves to a number of alternative choices for topics. For example:

- *Knowledge engineering*
  a. Theorem proving
  b. Expert system shell development
  c. Control system development
- *Adversarial competition*
  a. Battlebot or other agent-controlled robots
  b. Expectiminimax or other statistical gaming
  c. Steering or path traversal
- *Problem solving from nature*
  a. Particle Swarm Optimization
  b. Artificial life
  c. Ant colony optimization
- *Machine learning*
  a. Genetic programming
  b. Decision tree induction
  c. Reinforcement learning

Experiences and Lessons Learned

The course described in section 4 has now been taught five times, in five successive fall semesters. Enrollment was 25, 32, 22, 18, and 29 respectively. The course turned out to be a success from the start, so it has undergone very little adjustment from its original structure.

The ordering of topics proved key, for two reasons. First, it was important to conduct the game tournament relatively early in the semester, because if it were held near the end, harried student schedules would render the event less competitive and of less interest. Second, by placing the genetic algorithm assignment last, we were able to offer students the opportunity to overlap the assignment with another course CSc-165 – Computer Game Architecture. That is, if students incorporate genetic learning into their video game (from the other course), they can count that as the fourth project in this course. A small number of students (2 or 3) avail themselves of this option each semester.

Effectiveness of a Capstone-Style Course

The course is officially simply an upper-division elective. However, it has taken on a rather larger position from both student and department perspectives. To wit:

Student Evaluations The campus utilizes a standard student evaluation instrument for all courses. The ratings for CSc-180 have been extremely high. On two occasions students gave it the highest rating in the department, out of nearly 100 courses (sections) offered.

Written comments and other anecdotal feedback is also hugely positive. Students often report it as being their favorite course experience at CSUS, and one which feels like a culmination where they are able to bring their knowledge and skills to bear in creative ways.

After having taken CSc-180, a number of the students elect to also take the graduate AI course, while still undergraduates. The popularity of CSc-180 has turned the image of AI around in the school from being an obscure, largely theoretical sub-area, to an exciting state-of-the-art field limited only by one's imagination.

Annual Tournament Project #2 (the game tournament) has become well-known throughout the department and beyond. It attracts spectators: student, faculty, and even alumni. Previous winners often attend. The school dean attended the last competition, and it was documented by local reporters. Students are aware of the webpage that lists previous winners, and many who enter the course do so with the intent of earning a spot on the list.

For this reason, the class has taken on a bit of a competitive air, and as such many of the best students in the department enroll in it. It also makes the event seem in a way almost separate from the class itself, and take on a larger identity. So, although this is a senior-level class,

student hype for the course starts well before. In fact, many spectators are younger students planning to enroll in the course the next year.

STEM showcase The campus STEM steering committee (STEM stands for Science, Technology, Engineering, and Mathematics) asked each department to identify a lab or other departmental location as their STEM "showcase", to help in better focusing campus promotional efforts. The Computer Science department chose to identify the lab associated with CSc-180 as the STEM showcase. The lab also houses CSc-165 (Computer Game Architecture), and CSc-155 (Advanced Computer Graphics), so to be fair the choice was not only because of AI. However, the department has recently started touting the strength of its offerings in the area of "AI, Graphics, and Games."

Publications Although there was never an intent to structure the course so as to produce publishable results, conference publications can be (and have been) a byproduct.

Suggestions for Effective Delivery

Having delivered CSc-180 in this manner for five years, a number of lessons have been learned which help in ensuring effective delivery and overall success:

Written Reports Writing skills of college students appear to be at an all-time low, and by and large this has been true in this course as well. However, since students are generally doing different projects, clear and complete reports are essential for assessing the quality of the work. Although suggested report outlines and other guidelines are provided, the quality of reports submitted for the first project is usually poor.

Early on, students must learn that in this class they will be graded subjectively, and largely on the basis of the thoughtfulness of their work and to the degree that they are able to convey their motivations, successes, and failures in their reports. Before project #1, they are shown examples of good reports and bad reports from previous semesters. Still, the quality of reports submitted for the first project is usually poor. For some reason, students tend to consider good writing as abstract rather than as something actually expected of them.

However, there are invariably a few students who are excellent writers, whose reports can serve as catalysts for the rest of the class. For each project, some percentage of the grade (10%) is on the look of the report, and for project #1 the full 10% is awarded only to those two or three best writers. Then, when projects are returned, examples of poor reports are displayed (names hidden), followed by examples of the excellent reports (names divulged). Projects with otherwise excellent ideas or findings are also described (by the students themselves, time permitting). The idea is to include the report-writing as part of the competitive aspect of the course, so that students can see that their research indeed *is* judged largely on its presentation. Also, by seeing that excellent, polished reports *are being produced by students sitting next to them*, the bar is raised.

Insisting on Innovation and Excellence By the time this course comes around, most Computer Science students have acclimated to a learning environment in which they are asked to do very specific tasks, and where efficiency is rewarded. It is important to break this model whenever it rears its head. It is impossible for students to think innovatively if they are also trying to find shortcuts at every turn. Examples:

- *Using the "web" as expert*. Most students will ask if they can use web pages as experts in project #1. The answer is always: *no, you must find a human*. The identity of the human must be included in the report.
- *Modeling a trivial function*. Most students will find examples of neural networks learning simple mathematical functions, which render the entire process of forming training data easy. Again, they must find something more substantial to model.

One way of encouraging students to select more challenging problems is by encouraging them to select problems from among their hobbies or outside interests. Although this isn't possible for projects #1 and #2, it can be a useful trick in projects #3 and #4. When they already have knowledge and interest in the details of a particular topic, they are more likely to explore it thoroughly and be reflective of their own degree of satisfaction in the results. Furthermore, applications from seemingly mundane hobbies can reveal themselves as surprisingly difficult and interesting problems.

Running a Successful Tournament Utilizing a "minimax tournament" is not a new idea by any means. Game tournaments are frequently incorporated into AI courses, and have been for at least 30 years. However, running one that is consistently fair, challenging, and successful as a spectator event requires some attention to presentation, a well-coordinated and clear set of rules, and a well-designed game - preferably a new one.

Designing an Appropriate Game This can be a subtle and challenging step. The game must have the following characteristics:

1. *Does not already exist*. Otherwise, students will look for solutions on the web or from prior semesters.
2. *Simple enough* to code in a week or two.
3. *Complex enough* that perfect play won't be achieved.

4. *Lends itself to simple input/output* display.
5. *Will definitely end*. Games in which players can move aimlessly back and forth may never end, necessitating additional rules.
6. *No closed-form solutions*. Many games appear interesting, but upon inspection reveal simple ways of ensuring optimal play or avoiding loss.
7. *Deep search beats shallow search*. Surprisingly, some games – even complex ones – have the strange property that a 10-ply search doesn't produce better moves than a 3-ply search.

The points above virtually necessitate that the instructor design a brand new game every semester. Point #7, in particular, makes this a non-trivial task, because of the testing involved. Fortunately, a few relatively simple changes to existing games can often produce games of sufficient uniqueness and complexity to adequately serve for the relatively short time they are needed.

Even so, there is no avoiding the instructor implementing and extensively testing the game prior to assigning it. Point #7, above, is particularly subtle and means that the instructor needs to insure for example that a 10-ply search will beat a 6-ply search, and that a 6-ply search will beat a 3-ply search. *If the game doesn't have this property, the tournament will be discouraging and unsatisfying for the students that work the hardest on it*. (The same is true if the game has a closed form solution). Once coming up with an idea, the instructor should implement it and fine-tune it, making modifications to the game until it meets the properties described above.

One example of an ideal game was devised for the Fall 2006 section and dubbed "Rabbit Race". The rules were a sort of hybrid of chess and checkers and played on a 7x8 board, as shown in Figure 2. Pieces (X's and O's) move as do chess pawns (but also diagonally like checkers), capture by jumping (as in checkers), and the winner is the first player to have a piece reach square "*" on the opposite side of the board (dubbed the "rabbit hole"). The game is guaranteed to end because pieces can never move backwards.

			*			
X	X	X	X	X	X	X
X	X	X	X	X	X	X
O	O	O	O	O	O	O
O	O	O	O	O	O	O
			*			

Figure 2 – Rabbit Race board game, starting position

It took about a week of trial-and-error to develop Rabbit Race. Each semester a new game is devised.

Tournament Rules At SSU and CSUS, two class sessions are set aside for the tournament, and it is configured using a round-robin followed by double elimination so that each program plays several matches, and everyone stays active. For it to be an exciting spectator event, there needs to be constant activity, and the participants must not be bored. The atmosphere can also be spruced up with informational wall charts that people can peruse.

Here too, even the most innocuous mistakes in forming rules can turn an exciting event into a disaster. For example, the SSU tournament had a rule that no program could "think" longer than 30 seconds on any one move. As it turned out, some of the students implemented timers ensuring that their program *always* used the maximum 30 seconds on *every* move – even on obvious ones. Although they were within the rules to do so, their games lasted agonizingly long, often grinding the tournament to a halt waiting for them to finish their games. After that experience, time limits per move were reduced to 5 seconds.

There is also the question of computer platform. Students quickly learn that their program plays stronger on some machines than others, simply because more processing speed can mean a few more plies of search. At CSUS students may use any campus machine – such as one in the lab where the tournament is held, or a shared department computer they can telnet into. Thus, students are encouraged to explore the advantages of various machines, but no student has an advantage in this regard over another.

The rules that have developed at CSUS over the years are detailed in Figure 3. Rules #12 and #13 may seem merely cute, but actually are important to the presentation and atmosphere of the tournament. It allows wall charts to be made that emphasize – and even to some extent anthropomorphize – the programs. Students and spectators find themselves studying and comparing the descriptions of the programs, forming favorites, and referring to the programs by their given names rather than by the names of their authors. Additionally, although not required, several students choose to implement fancy graphics to give their games a more personal look.

There are a few additional academic requirements – for instance it is explicitly required that students use minimax and alpha-beta pruning and avoid trying to find closed form solutions. There are also typically some additional appropriate technical requirements included for the purposes of grading – such as requiring that programs search a minimum of 5 plies, using minimax with alpha-beta pruning, etc.

1. *Each move must be made in under 5 seconds.*
2. *Programs must be run on campus hardware.*
3. *Any programming language may be used.*
4. *Programs must comply with the provided, standard notation to facilitate move entry by the opponent.*
5. *The board must be displayed after every move.*
6. *Programs must be able to move first or second.*
7. *All matches are two games, with each program moving first one game, then second in the next.*
8. *The programs must be able to detect when the game has ended, and who has won.*
9. *An illegal move is a loss, no exceptions.*
10. *A program must be able to detect when an illegal move has been attempted, and ask that the move be re-entered (this is essential, otherwise an innocuous typo can force a game to be restarted, delaying the tournament).*
11. *A runtime crash is a loss, no exceptions.*
12. *Before the tournament, each student submits a summary of the techniques used, search depth, estimated strength, etc. of their program.*
13. *Students must invent a "name" for their program.*

Figure 3 – Board Game Tournament Rules

Conclusions

This paper introduced an approach for teaching Artificial Intelligence at the undergraduate level as a creative capstone. Such a course consists of a set of four projects covering a spectrum of AI areas, while retaining flexibility regarding the particular topics chosen. Although the term capstone is used to describe the approach, it is not necessary for the course to be an official capstone in the major. The motivation for modeling the course in this way is to leverage AI's image as a creative endeavor, while mitigating its image as a less practical option.

The course as defined was implemented and taught five times, once at one university and four times at another. The course was observed to have a number of positive benefits to the department and the field. Interest in the study of AI was raised, and students appreciated the opportunity to apply their skills creatively as well as the chance to add unique achievements to their resumes. The course proved valuable as a visible promotional tool. It also coordinated naturally with other project-based courses, such as game architecture and graphics.

In order to ensure that the course is successful and meets its objectives, instructors should take steps to maintain high standards for the written student reports, direct students towards challenging projects, and design competitive activities with great care. A number of specific guidelines were presented.

The dreaded AI winter need not diminish the exciting opportunities the field has to offer an aspiring computer scientist.

References

[Cop04] B. Coppin, *Artificial Intelligence Illuminated*, Jones and Bartlett 2004.

[Hav05] H. Havenstein, *Spring Comes to AI Winter*, Computerworld, Feb 14, 2005.

[IA01] *Computing Curricula 2001 Computer Science*, The Joint Task Force on Computing Curricula, IEEE Computer Society and the Association for Computing Machinery, 2001.

[Jon05] T. Jones, *AI Application Programming*, 2nd ed. Charles River Media 2005.

[KW95] D. Kumar and R. Wyatt, *Undergraduate AI and its Non-Imperitave Prerequisite*, ACM SIGART Bulletin Special Issue on AI Education, Vol 6 No. 2, April 1995.

[Kum99] D. Kumar, *Beyond Introductory AI*, ACM Intelligence Magazine Vol 10, No. 3 Fall 1999.

[Mar06] J. Markoff, *Brainy Robots Start Stepping into Daily Life*, New York Times, July 18, 2006.

[Mil06] I. Millington, *Artificial Intelligence for Games*, Morgan-Kaufmann, 2006.

[Neg02] M. Negnevitsky, *Artificial Intelligence, a Guide to Intelligent Systems*, 2nd ed., Addison Wesley 2002.

[RN03] R. Norvig and P. Norvig, *Artificial Intelligence, a Modern Approach*, 2nd ed., Prentice Hall 2003.

[PU07] Senior Capstone in Computer Science at Pacific University (accessed July 2007): http://www.pacificu.edu/as/compsci/capstone

[Swa03] Swart, M., *Creative Capstone Computer Projects for Post-Graduate Students of English*. Journal for Language Teaching, 37(1), 2003.