TEACHING PROBLEM SOLVING AND AI WITH PAC-MAN

Jim Smith
Department of Computer Science
University of the West of England
Bristol, BS16 1QY, UK
james.smith@uwe.ac.uk
http://www.bit.uwe.ac.uk/~jsmith

Steve Cayzer
Department of Computer Science
University of the West of England
Bristol, BS16 1QY, UK
steve.cayzer@uwe.ac.uk
http://www.bit.uwe.ac.uk/~jsmith

ABSTRACT

We describe a series of resources and practical exercises supporting the teaching of introductory topics in Artificial Intelligence using the metaphor of the well-known arcade game "Pac Man". They are aimed at level one students from a range of disciplines and motivated by a view of Artificial Intelligence as a means of automating the problem solving process. The first set of exercises start with search strategies and gradually build up via rule-based and expert-system approaches to create a Pac-Man player based on "traditional AI". Subsequent apply computational approaches such as artificial neural networks and evolutionary computation provide a radically different approach to generating and improving controllers

Keywords

Artificial Intelligence, Problem Solving, Groupwork, Coding.

1. INTRODUCTION

Systems containing some kind of Artificial Intelligence (AI) form the state of the art in many applications of computing, across the spectrum of social and economic activity, and the future is certain to contain more, rather than less such systems. Moreover, since AI fundamentally concerns itself with problem solving, it should provide an ideal medium to explore issues and provide generic transferable skills. Despite this, many undergraduates view AI as a rather dry subject matter, a situation that is not helped by many of the learning exercises to be found in textbooks and on-line coursework materials. Several authors have attempted to address these using robotics-based tutorials (Kumar and Meeeden 1998, Karapetsas and Stamatis 2008, Parsons and Sklar 2004) but these tend to be specific to certain aspects, and often require programming skills beyond a typical level one student. More importantly, a series of informal conversations with current and potential students revealed a view of anything using robots as somewhat "techie" and off-putting. Certainly there is a concern that they can reinforce a view of AI as primarily relevant to robotics rather than to all disciplines within computing. A second problem facing the would-be teacher of a general course on AI is that books are often specialised, and/or often use different examples, and different software to illustrate and teach different types of AI. This raises issues where either students face the overheads of learning many different software packages, or are required to code their own algorithms, which risks turning AI classes into coding tutorials. This analysis lead to the identification of the following "wish-list" to which any new set of activities should adhere:

- Use of a "metaphor" which is well-known and liked across a range of backgrounds;
- Emphasis on problem-solving skills that can inform and assist other topics.
- Use of a common programming environment, supporting tutor-provided scripts / functions to create a "naturalistic" style of programming with as low a learning overhead as possible;
- Coverage of the syllabus elements of search strategies, rule-based systems, expert-systems, artificial neural networks, evolutionary computation and swarm intelligence (multi-agent systems);
- Use of a free, lightweight, and simple to install software environment, preferably one that also contains or supports a range of other examples to stimulate the more able students.

In this paper, we describe how this analysis lead us to design a series of practical exercises using the metaphor of the well-known "Pac Man" game, and implemented in the Netlogo environment (Wilensky 1999).

2. BACKGROUND AND CONTEXT

These materials were developed for the UWE level one module "Introduction to Artificial Intelligence". This

module is compulsory on several degrees such as the BSc's in Computer Science, Robotics, and Games Technology, and optional on several others. The syllabus of the module is fairly conventional. In the first semester the students briefly cover "philosophical issues" such as Turing's test, and Searle's Chinese Room problem, but primarily as a means of focussing on what "AI in practice" can or might involve, in opposition to what Hollywood might portray. From there a number of scenarios (driving a car unaided, acting as a tour guide, playing table tennis) are used to highlight how "AI" behaviour can be broken down into groups of related tasks, and the "input-model-output" paradigm of computing is used to group these and introduce the idea of all learning as a search through a space of possible inputs (or sequences), models, or outputs. The formal lectures then move on to cover blind and informed search strategies as means of automating search (depth-/breadth-/best- first, hill-climbing, A*) then knowledge representation: moving through first order logic and rule-base systems, via expert systems to the Semantic Web.

Semester two starts with by recapping the idea of learning as search, the three principle classes of search space and some examples of relevant Machine Learning technologies, and then covers three areas in more depth. The first is Artificial Neural Networks: perceptrons and simple MLPs with back-propagation as a hill-climbing search method. The second is Evolutionary Computation as a general-purpose search method, illustrated by Genetic Algorithms for optimisation and Genetic Programming for model building. Finally the cellular Genetic Algorithm is recast as an emergent process and used to introduce the topic of "swarm intelligence" illustrated by optimisation (Ant Colony Algorithms), model building (ant-based clustering) and simulation ("Boids" (Reynolds 1987).

The majority of the activities designed to support this learning had the students working in groups of three or four. Typically, after the task was introduced, everyone would be asked to think about it individually for five minutes before the group activities commenced, and at the end of the session a "wrap-up" discussion would see the groups working together either in competition or collectively to come up with ideas how this work could be used to solve some larger task.

The Netlogo environment is an open-source multi-platform package for developing and running simulations of multi-agent systems. Based on a variant of the logo programming language, it combines a simple interpreted scripting tool with an interactive interface for running, pausing and manipulating scripts. Scripts can also be saved as applets embedded in a web page containing the instructions. Netlogo has been used for both research and teaching in a range of disciplines covering the physical sciences, sociology, economics etc. A screenshot is shown in Figure 1. In addition to the very well written, user-friendly environment, Netlogo has the advantage of a small language and syntax, and great extensibility. Thus the students only need to learn how to use Boolean constructs (which end in "?"), and the syntax of if, ifelse, while and foreach. By writing suitably named procedures the tutors can hide all the rest of the detailed syntax and coding niceties such as list operations, observer functions, and exception handling.

Pac-Man is a classic arcade game in which the player steers a character (the pacman) around a maze eating pellets of food and trying to avoid a number of mobile "ghosts" which attempt to eat the pacman – reducing the number of "lives" left. As extra complications "power pellets" temporarily change the appearance of the ghosts, rendering them edible to the pacman. While conceptually simple, the game involves search (finding the valid paths to uneaten pellets), classification (recognizing the edibility of ghosts), tasks hierarchies (avoiding being eaten) and planning (e.g. not eating the "power pellet" if no ghosts are within range).

3. TOPIC-SPECIFIC EXERCISES

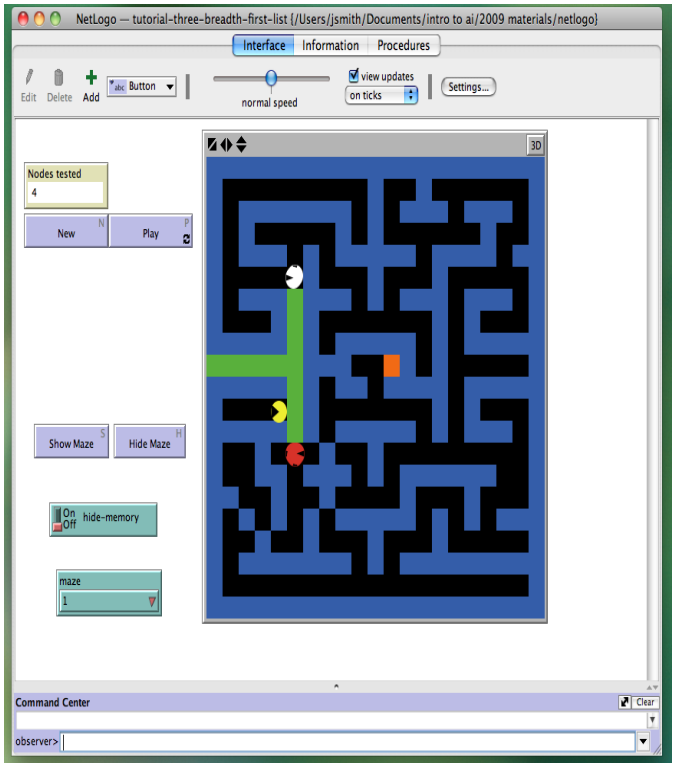
3.1 Search Methods

In order to accustom the students to the idea of blind search, how people use memory and clues, and how making assumptions can help search, the first tutorial takes the form of an interactive applet which presents the students with an initially blank grid and asks them to manoeuvre the pacman to find a gold square. The students are encouraged to write down the strategies they are using, and an on-screen display of the number of moves taken forms a natural basis for competition and comparing strategies. One powerful teaching tool is the ability to toggle the maze's *visibility*, either by displaying a limited 'window' around the pacman, or by simply displaying the whole maze at all times; the latter allowing strategies which employ 'look ahead'. Another equally important parameter is *memory*. This is achieved by colouring visited cells green, and hence facilitating movements that prioritise unexplored directions. The students were asked to formalise how these factors affected their planning strategy, and to document their "solution" online sufficiently unambiguously that another group could follow it.

The role of the tutors in these sessions is critical to it becoming more than just a play session. We used two or three tutors per session, who moved between the groups asking questions, prompting and commenting. The other factor critical to success is the use of a twenty-minute "wrap-up" session with plenty of student interaction. During this stage a number of common points should be drawn out and then related back to the

more formal language of the lectures. As mentioned above, this module is core to several different degrees, so in addition to the obvious points about memory and the value of heuristics (e.g. most students do "turn towards the middle") the exercise could facilitate discussion tailored to different degrees – e.g. global vs. local models for Robotics students, storage and run-time analysis for Computer Science students etc. At the end the students are asked

In the second session the students are asked to modify the scripts from the first tutorial to implement either depth-first of bread-first search. The scripts are heavily commented to show where the new automated code should go. They are provided with examples of "if", "if-else" and nested versions of these, to aid them with syntax. Most importantly, they are given a set of primitive constructs from which to build their code. These include Boolean tests (e.g. "*can-turn-left?", "wall-ahead?"*) and functions to turn the pacman ("*turn-left", "go-ahead", "turn-around", "turn-north*",…). The surrounding code deals with the movement of the pacman, so their code only needs to decide which way the pacman should face at each timestep.




```
...
if need-to-pick-another-pacman?
[
set current-pacman-id get-oldest-pacman
;;set current-pacman-id get-youngest-pacman
;;set current-pacman-id get-closest-of-all-pacmen
;;set current-pacman-id get-furthest-of-all-pacmen
;;set back-tracking-allowed? FALSE
;;set current-pacman-id get-closest-child-of-last-parent
;;set current-pacman-id get-oldest-child-of-last-parent
;;set current-pacman-id get-youngest-child-of-last-parent
update-parent-id
]
```

Figure 1: Left: Screenshot of Breadth-first search. Current pacman is coloured yellow. Two other pacmen at same depth are white, three at the next depth is red. Some pacmen are slightly overlaid. Right: corresponding code fragment allowing students to create different search strategies.

Most students correctly identified their previous wall-following strategy as a version of depth-first chose to implement that, and recognised that they should use relative rather than global co-ordinate systems. They were given various mazes so that the more able students could think about how to modify their code to cope with recursive or looping mazes. Most groups successfully built the controller to navigate the loop-free maze.

In session three they are then shown a new Netlogo script in which, rather than a single pacman moving around the web, the current pacman moves along the maze to the next junction whereupon it spawns one child facing along each unexplored path (if there are any) and then dies, whereupon a decision has to be made which of the extant pacmen to move next. Each pacman contains variables recording their "depth" and the initial "distance_to_goal". This latter records the Euclidean distance to the goal, which is of course not necessarily the "path" distance but it is at least an optimistic heuristic, which is a prerequisite for the optimality of A* search. Manhattan distance is another alternative that can be used. All these details can be hidden from the students but are available for discussion if appropriate. Figure 1 shows a screen shot of breadth-first search with this model, and the relevant code fragment. The learning task is to apply the conceptual ideas of search to this particular scenario and thus create implementations of depth/breadth and best-first search, Hill-Climbing and A*. A number of mazes are provided, and an informal competition between the groups encourages experimentation. At the end of these sessions all groups had working implementations of procedures that could be built into a final pacman controller to provide different search mechanisms, and some understanding of the characteristics of each search method.

3.2 Knowledge Representation

The decision to ask the students to build a controller that decided which way to face at each timestep, while hiding the looping and movement phases, allowed a deliberately focus on conditional logic. Once the lectures move on to consider knowledge representation, rule-based and expert systems this is further extended in a series of exercises which build on this rather stylised form of propositional logic.

In the first exercise the students are asked to create and save new ghost shapes in netlogo's "turtle shape editor", then to create some classification rules that will assign this ghost shape to the class "edible" or "inedible" based on visible characteristics. They are asked to add the ghost details into a "world map" - a spreadsheet that contains maze and ghosts characteristics. They need to instantiate new ghosts, label them as edible or inedible and finally create new attributes describing their ghosts. The students are also provided with a script, which loads the map and calls a stub function that classifies each ghost in turn. The primary learning task is thus to write the code that correctly classifies all the ghosts. Depending on how the ghosts are labelled, and the number of attributes used in the classification rules, this activity provides scope for discussion of rule-hierarchies, default rules, disjoint and conjoint logic. Of course there are many equivalent ways of doing this, providing scope for a "wrap-up" session where the different versions produced by students are shown to be formally equivalent.

The penultimate set of tutorials focussed on the specification, creation and maintenance of expert systems. Using the maze from the previous tutorial, the students were asked to devise a set of rules to control the Pac-man using a range of (supplied) primitives. Choice functions include returning a set of directions to test and condition clauses include simple tests (clear, safe), ones that involve lookahead (ghost ahead), classification (edible ghost ahead), heuristics (nearer to edible ghost) and memory (unexplored). Higher level planning for the more advanced students was provided through the use of route planning (choose next edible ghost). The rule production was done as a "pyramiding" activity – each member of the group was asked to come up with single rule describing a particular situation, and then the group collectively refined the combined rule set and implemented it – iterating as necessary to achieve the desired behaviour. The groups were free to use the procedures provided, or any other ones they had developed or used previously.

In the final practical of this section, the students were invited to put their classification, search and expert system work together in a series of provided mazes, with challenges such as dead ends and moving ghosts. Supporting material has been made available for motivated students to extend this work into the full pacman game with pellets and power pills. As hoped, this activity provided a valuable experience in the "naturalness" of expert systems, but also of the difficulties of ensuring correctness and maintaining the rule bases. It also showed how increasing levels of knowledge could be built into rules that deal with environments at a range of complexities. The classification, expert system and route planning activities also showed how AI techniques could be employed at different levels of abstraction.

3.3 Artificial Neural Networks

The lectures for this topic cover the use of perceptrons as a stylized representation of the action of biological neurons, Hebbian learning, and the strengths and weaknesses of single perceptrons – e.g. the XOR problem. They move on to cover simple multi-layer perceptrons (MLPs) and back-propagation. Among the set of models that come provided with Netlogo are a simple perceptron and a three-input, two-hidden node MLP. These come with train/test routines for learning and testing simple functions such as or/and/xor. A particularly nice feature is that the values of the weights are shown graphically by varying the width of the link between nodes. Next, using training and test sets built from variations on the set of four ghosts from Figure 4, students learn how these formal functions relate to practical problems by showing that neural networks are capable of learning rules that correctly classify ghosts. A range of exercises, and careful placement of (static) ghosts on mazes is used to illustrate issues such as over-fitting, problems of feature selection/sparse datasets, etc.

3.4 Evolutionary Computation

Like many agent-based systems, netlogo comes complete with a model of a simple genetic algorithm using one-point crossover, bitwise mutation and fitness-proportionate selection to evolve a solution to the binary "OneMax" problem, where the fitness is simply the number of bits in a string. The students begin with a structured series of exercises, using sliders in the interaction tab to explore the effect of changing population size, selection pressure, mutation and crossover rates so as to illuminate the effect and value of these operators and settings. The second exercise uses a fitness function that periodically switches between "OneMax" and "ZeroMax". Students are asked to first design on paper, and then evaluate, strategies which maximize the response time, and time-averaged vales of the mean and maximum population fitness. This aids discussions of issues such as the need for multiple runs with stochastic algorithms

The final set of exercises concerns the task of designing an EA that will evolve a controller for a Pac-Man to find the "gold" in a maze. Solutions are interpreted as a set of rules governing motion, so at each time step the "state" of the neighbourhood is queried and the evolving solution is queried to provide the relevant action. The students are given a choice of two representations – in the first the "state" is the timestep – so a solutions is represented as a sequence of moves. In the second, the state is inferred from the Pac-Man's current location and orientation, so the binary representation for this problem is effectively a Pittsburgh-style classifier system. The occupancy (wall/empty) of the three squares (left, ahead, right) provides for 8 states, which with four directions requires a sixteen-bit representation. The extension to consider ghosts etc is trivial. They are required to select appropriate parameters (mutation rate, population size etc.) for their chosen representation.

The final design decision requires the students to create a fitness function based on weighting factors such as final distance from goal, number of cells explored, the reward for reaching the goal, and the number of steps taken to reach it. In practice most explored both representations, and the particular choice of maze nicely illustrated issues such as over-fitting – the rule based version first "evolved" depth first, then "refined" it to exploit characteristics of the specific maze used.

4. CONCLUSIONS

This paper reports an attempt to enliven the teaching of artificial intelligence by a coherent set of providing practical activities illustrating the main elements of an introductory course to AI via the metaphor of PacMan.

These activities have been running since September 2009 and have been well received. Attendance, already high in previous years has been extremely high and activity of discussion board sand between classes has been higher than usual. Of 50 students filling in an anonymous module evaluation form, more than half used the "extra comments" section to comment positively on the use of Pac-Man.

5. ACKNOWLEDGEMENTS

The authors would like to thank Doctor Gordon Downie for his considerable input to this project. The work was supported by grants from the Faculty of Environment and Technology at UWE ("Development of interactive small group tasks to be embedded within larger group tutorials") and the Higher Education Academy Subject Centre in Information and Computer Sciences ("Teaching Artificial Intelligence with PacMan").

6. REFERENCES

Kumar, D, and Meeden, L. "A Robot Laboratory for Teaching Artificial Intelligence", Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE-98), Daniel Joyce, Editor, ACM Press, 1998.

Karapetsas , E. and Stamatis, D., "Teaching AI Concepts Using a Robot as an Assistant", in Fasli, M. (ed.) Proceedings of the 4th Artificial Intelligence in Education Workshop, Cambridge, UK. HEA-ICS, 2009.

Parsons, S. and Sklar, E. Teaching AI using LEGO mindstorms, Proceedings of the AAAI Spring Symposium on Accessible Hands-on Artificial Intelligence and Robotics Education, Stanford, 2004.

Wilensky, U. NetLogo. Available from http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL. 1999

Smith, J.E. "Re-usable Online Assessment Materials for Teaching Artificial Intelligence" in Fasli, M. (ed.) Proceedings of the 4th Artificial Intelligence in Education Workshop, Cambridge, UK. HEA-ICS, 2009.

Reynolds, Craig (1987), "Flocks, herds and schools: A distributed behavioral model.", SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques (Association for Computing Machinery): 25--34, doi:10.1145/37401.37406, ISBN 0-89791-227-6