# Learning Through Programming Games: Teaching AI With Pac-Man and Netlogo

Jim Smith and Steve Cayzer

**Abstract** In this paper, we describe a series of practical exercises designed to aid 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. Therefore each piece of practical coding is preceded by an exercise to illuminate the human cognitive activities involved. 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. The second semesters activities concentrate on how computational approaches such as artificial neural networks and evolutionary computation provide a radically different approach to generating and improving controllers.

## 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, from data-mining to films and games via most non-trivial web-based applications. Moreover the future is certain to contain more, rather than less such systems.

Despite this, many undergraduates view AI as a rather dry subject matter, a situation which 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 [1,2, 3] but these tend to be specific to certain

Jim Smith
Department of Computer Science, University of the West of England, Bristol BS16 1QY, UK, e-mail: james.smith@uwe.ac.uk

Steve Cayzer
Department of Computer Science, University of the West of England, Bristol BS16 1QY, UK, e-mail: steve2.cayzer@uwe.ac.uk

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 the robotics community 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.

An alternative approach, based on a series of group-work paper exercises has been tried with some success at UWE in recent years In seeking to develop the courses to which AI contributes, it was recognised that the inclusion of more practical activities and problem-solving exercises would provide benefits beyond the remits of this module alone. 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.
- 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 creation and maintenance, artificial neural networks, evolutionary computation and swarm intelligence (multi-agent systems).
- Use of a free, lightweight, and simple to install software environment. Preferably one which also contains or supports a range of other examples to stimulate the more able students

In the rest of this paper, we describe how this analysis lead us to design a series of practical exercises using the metaphor of the well-known arcade game Pac Man, and implemented in the Netlogo environment [4]. The paper proceeds as follows:

- Section 2 provides background context on the size and nature of the module, and the use of group exercises.
- Section 3 describes the netlogo environment and how it has been used for this work.
- Section 4 describes the first semesters activities, based on traditional AI.
- Section 5 describes the activities in the second semester supporting the learning of Computational AI.
- Finally in Section 6 we discuss our findings and experience to date and draw some conclusions.

## 2 Background and Context

These materials were developed to be used on the UWE level one module Introduction to Artificial Intelligence. This module is compulsory on several degrees such as the BScs in Computer Science, Robotics, and Games Technology, and optional on several others. Typically there are between 125 and 150 students taught via one hour lecture and a 90 minute tutorial each week. The tutorial is timetabled to run two or three times with 50-75 per session, and uses a trolley of tablet pcs for interactive group work. The module makes extensive use of the Blackboard ł VLE to deliver course notes, reading materials and weekly self-assessment tests [5], and to mediate facilitate groupwork through discussion fora, dropboxes etc.

The syllabus of the module is fairly conventional. In the first semester the students briefly cover philosophical issues such as Turings test, and Searles 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. During these first two-three weeks the tutorials activities primarily use relatively large group exercises (10-15) where the students work on a problem (identifying tasks in the above scenarios, designing questions for their online Turing test) before presenting their results to the class. These tutorials have been developed and refined to serve additional purposes as ice-breakers and development of skills in group working and collaboration.

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 a recap of the ideas of learning as search, the three principle classes of search space and some examples of relevant technologies before moving on to cover 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 (see e.g. [6]). 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 [7], model-building with ant metaphors [ibid] and simulation with Reynolds Boids [8].

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.

## 3 The Netlogo Environment

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. A netlogo file consists of three sections, one describing the interface, a second the instructions, and a third containing the code for the runtime procedures. These are parsed into different tabs in the netlogo environment. 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 wide range of models of systems are available, including the Pac-Man implementation, which inspired this work. A screenshot is shown in Figure 1.

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 games 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). .

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, ”foreach” and while. All the rest of the detailed syntax and coding niceties (list operations, observer functions, exception handling) can be hidden by the tutors writing suitably named procedures.

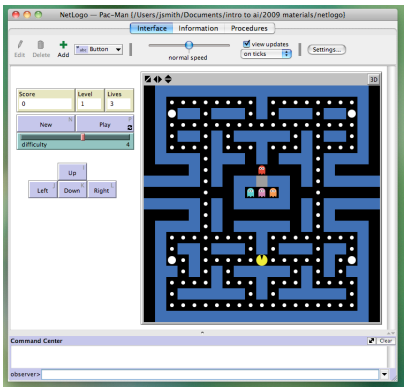


**Fig. 1** Screenshot of Netlogo PacMan model. Note tabs for switching between interface, Information and scripts (Procedures).

## 4 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. They are told that they have to manoeuvre the pacman around using up/down/left/right keys to find a gold square. As the students manipulate the pacman they are encouraged to write down the strategies they are using. An on-screen counter displays how many moves they have made which form a natural basis for competition and comparing strategies.

One powerful teaching tool is the ability to toggle the mazes 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.

To start off with, the task is set with a maze that contains no loops and the gold is towards the middle. In the 2009 run, about half of the groups discovered or recalled a wall-following strategy. There seemed to be correlation between success, and experience of physical mazes.

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 to document their solution online sufficiently unambiguously that another group could follow it.

In the second session the students work in the Netlogo environment. They 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 give an 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.

Most students correctly identified their previous wall-following strategy as a version of depth-first and chose to implement that, and recognised that they should use relative (left, forward, right) rather than global (North, East, South, West) co-

ordinate systems. Given the range of experience and ability of a large level one class, this task is challenging for some, and achievable for others. They were given two mazes  one with loops  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.

To reinforce this, at the start of the second session the students are asked to self-organise themselves so that no two members of the previous weeks groups are together, at least one person in each group did get the first maze solved last week, and at least one did not. They are then given a short period to explain their previous code to each other, and decide on the final version of their code, before a random member of the team is asked to demonstrate and explain it.

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 which can be used. Care needs to be taken to take account of wrapping (that is, if the world is a torus or a 2D grid). All these details can be hidden from the students but are available for discussion if appropriate.

Figure 2 shows a screen shot of breadth-first search with this model, and Figure 3 the code. As shown in Figure 3, the script contains a ranger of primitives and switches, and the task is use these to 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.

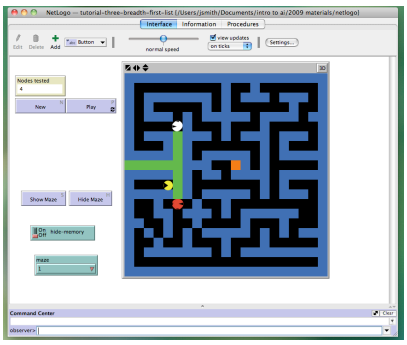


**Fig. 2** Breadth-first search. Current pacman is coloured yellow. Two other pacmen at same depth are coloured white (these are slightly overlaid, waiting to explore their paths) and a pacman at the next depth is coloured red.

## 5 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 shown netlogos turtle shape editor and are first asked to create and save new ghost shapes. They are asked to create some classification rules that will assign this ghost shape to the class edible or inedible based on visible characteristics. Figure 4 shows an example of such a shape.

They are asked to add the ghost details into a world map which is a spreadsheet containing 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 a map containing four ghosts (two colours, with or without spectacles). The script calls a stub function that classifies each ghost in turn. The primary learning task is thus to write the code that

```
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
]
```

**Fig. 3** Code Fragment for search currently implementing hill-climbing.

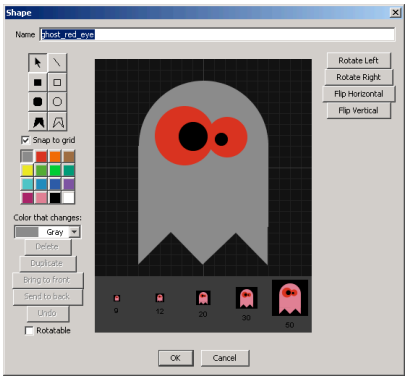


**Fig. 4** Screenshot of Turtle Shape Editor showing a new ghost shape created by the students with the attribute red eye indicating that this ghost is inedible.

correctly classifies all the ghosts. Depending on how the ghosts are labelled, and the number of attributes assigned to the ghost breed, this activity provided scope for discussion of rule-hierarchies, default rules, disjoint and conjoint logic.

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. The script shown in Figure 5 shows one of the possible ways of correctly classifying the four ghosts. 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.

In the current years run (2009-10) the task of putting together the classification code with the search strategy was left as an exercise for the students to work on in their own time. A review session at the end of the term provided for more intensive help and guidance with this. Instead the final 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 determine the step-by step actions of the pacman.

Figure 6 shows 2 example rules, built using a range of (supplied) primitives including choice, condition and action functions. Choice functions include returning a set of directions to test in the absolute (north, south, east, west) or relative (forward, left, right, behind) or random direction. 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). Available actions include both absolute (move north) and relative (move left, turn around) options. 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.

```
to-report classify-ghost

 ;; test for conjunction
 if (color = 15) and (glasses? = true)
    [report "edible"]

  ;; 2nd test: disjunction
 if  (color = 85) and (glasses? = false)
    [report "edible"]

 ;; otherwise assume inedible
 report "inedible"
```

**Fig. 5** One version of code to classify ghosts.

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 for motivated students to extend this work into the full pacman game with pellets and power pills.

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 can be built into rules which deal with environments at a range of complexities. The classification, expert system and route planning activities also showed how AI techniques can be employed at different levels of detail.

## *5.1 End of Semester Competition*

In the final tutorials it was pointed out that the pellets in a traditional pacman maze can be considered as simply a different type of static edible ghost. The students were provided with a version of pacman with pellets and ghosts that moved at the same speed (and just after) the pacman and a competition was held to see which traditional AI based strategy could achieve the highest score. This will be repeated at the end of semester two.

```
set pacman-still-to-move? true

;; rule 1
foreach all-directions
[
   if pacman-still-to-move?
      and is-direction-clear (?)
      and is-direction-safe (?)
      and is-direction-unexplored (?)
   [move (?) ]
]

;; rule 1
foreach all-directions-random
[
   if pacman-still-to-move?
      and is-direction-clear (?)
      and is-direction-safe (?)
   [move (?) ]
]
```

**Fig. 6** expert system style rules for a pacman controller.

## 6 Artificial Neural Networks

It should be pointed out at this stage that these activities are being run for the first time in 2009-10, so the activities described hereafter have been planned and tested by the tutors but not yet run in class.

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 value of the weights are show graphically by varying the width of the link between nodes. Having used the pre-built models to demonstrate the behaviour on simple logical functions, the first set of activities then uses a modified versions of the perceptron model. Using training and test sets built from variations on the set of four ghosts from Figure 4, the learning goal is to show how these formal functions relate to practical problems by showing that the perceptron is capable of learning rules that correctly classify ghosts - effectively using and and or, but not xor. In a subsequent set of activities the multilayer perceptron is used with the same tasks. Issues such as over-fitting and scalability are illustrated by varying the number, type and size (number of attributes) of ghosts in the training set. In the final set of activity the students will take the trained MLPs and insert the code into their previous controllers in place of the rule-based classifiers.

## 7 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. Sliders in the interaction tab allow the effect of changing selection pressure, mutation and crossover rates, and population sizes to be explored. Modifying the fitness function to explore individual tasks relevant to the pacman game is relatively straightforward. In this case an interactive client-server model has been devised, where the tutor runs a genetic algorithm that interactively requests the fitness value for each member of the population. The tutor distributes the individual binary strings to the different groups via a Blackboard discussion fora. Each group pastes their solution into a file, then loads that file into the pacman simulator where it is interpreted as a set of rules governing motion (technically each rule is a turtle owning an action attribute which it reads from the file). At each time step the state of the neighbourhood is queried and used as an index. The corresponding rule is then queried to provide the relevant action. The fitness given to a rule set is the score obtained (number of pellets eaten in a fixed number

of lives and time steps). The binary representation for this problem is effectively a Pittsburgh-style classifier system, and works as follows:

- Only the three squares left, right and ahead of the pacmen are considered.
- Initially each cell is considered occupied (by a wall) or empty.
- In the more complex version, the occupancy of each of these takes one of four values: empty, wall, edible, inedible

There are 8 (64) of these situations, for each of which there are four possible outputs (turn left/right/around, go ahead) coded by two bits, so the representation is a 16 (128) bit binary string. Working with 15 groups per session - and so a population size of that size - should provide well able to solve the simpler problem, and able to make a good attempt at the more complex one. In practice for the latter it may be necessary to tweak the rule set or to seed the initial population with some rule sets that actually achieve something. Should time permit a script that merges the pacman and genetic algorithm code and runs the pacman simulator without displaying it will be produced. This will show a simple plot of fitness vs. time and possible the behaviour of the best in the final generation. Another option under current consideration is the creation of a Genetic-Programming-like system to evolve rule sets for classifying ghosts or to control behaviour based on the Sondahls GP framework in netlogo and Maze Marchers model [9].

## 8 Swarm Intelligence

The agent-base nature of netlogo makes it eminently suitable for illustrating swarm intelligence, and models exist in the distribution to show ant-based clustering and route-finding. As an exercise the students shown how to modify the pacman model code to start with multiple pacmen, and then asked to investigate how they interact and to find a set of rules that facilitates productive interaction to clear the maze of pellets more rapidly. This is deliberately left as an open-ended task to encourage exploration of different ideas.

## 9 Conclusions

This paper reports an attempt to enliven the teaching of artificial intelligence by a coherent set of providing practical activities which illustrate al 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 course at this stage it is too early to say whether this is because of a change in the nature of the activities, or the use of PacMan. Developing these activities has been an extensive and iterative

task. It is expected that the activities for semester two may change from week to week as they are rolled out. Should time allow, there are a huge number of options for future development for example the Hub-net client-server model provided in Netlogo should prove ideal to co-ordinate some of the activities in semester two.

## 10 Acknowledgements

The author would like to thank Doctor Gordon Downie for his 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).

## 11 References

[1] 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.
[2] 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.
[3] 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.
[4] Wilensky, U. NetLogo. Available from http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL. 1999
[5] 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.
[6] Eiben, A.E. and Smith, J.E. Introduction to Evolutionary Computation Springer, 2003.
[7] Dorigo, M. Swarm Intelligence. Oxford University Press, 1999.
[8] 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
[9] Sondahl, F., Solving Mazes with Genetic Programming Final Year project, ?CS 460: Multi-Agent Modeling, Northwestern University (2005).