# It Takes a Virtual Village: Towards an Automated Interactive Agency

**Elizabeth Sklar**
Department of Computer Science
Columbia University
New York, NY 10027 USA
*sklar@cs.columbia.edu*

**Abstract**

Internet agents are frequently designed to be personal assistants, helping a single user accomplish a specific task. The work presented here explores the idea of agents as independent entities existing in a *virtual village*, geared towards education. Three classes of agents are presented, designed to meet the varied and changing needs of a population of human learners.

## Introduction.

With the advent of the Internet, humans have found new and exciting ways to interact, with each other and with a seemingly infinite network of information and shopping sources. The explosive popularity of electronic mail, instant messaging, on-line chat, virtual shopping outlets and auction houses is proof that, like the telephone and the department store, the World Wide Web is here to stay.

Humans are not alone in populating the Internet; software agents also inhabit cyberspace. Internet agents are generally built to be personal assistants, helping a single user accomplish a specific task. Over the last decade, Internet agents have gained prominence as browsing assistants (Lieberman 1995), matchmakers (Foner 1997; Kuokka & Harada 1997), recommenders (Balabanović 1998) and filterers of email and news group messages (Goldberg *et al.* 1992; Lashkari, Metral, & Maes 1994; Lang 1995). Some agents use various machine learning techniques to adapt their behavior to the needs of individual users (Qureshi 1996; Balabanović 1998).

The existence of interactive Internet agents allows us to establish the notion of a *virtual village*, wherein humans in diverse places and time zones can meet, shop and trade just as in a traditional village, but the need for spatially or even temporally co-locating participants and goods is now eliminated. This notion is not only useful for shopping and trading, but also — and maybe more importantly — it is useful for education. If students populate the village, they can learn from each other, anytime and anywhere.



The work discussed here explores the idea of agents existing in a virtual village geared toward education, designed to meet the varied and changing demands of a population of human learners. We surmise that a learner needs to interact with a variety of others exhibiting different talents and abilities in order to maintain interest and to progress. In school, students learn from teachers, from doing their homework and from participating in group projects where they interact with their peers. To support this type of environment in a virtual setting, we define three classes of agents that a student may interact with:

- **instructors**: agents that emulate the behavior of a human expert
- **peers**: agents that capture the mode of a group of humans sharing similar behavioral characteristics
- **clones**: agents that copy the behavior of an individual human

Students can have the option of interacting with only one or a combination of two or all three classes of agent, just as at different stages in a learner's development, s/he will need to receive instruction from a teacher, practice on her own and collaborate with her peers. Within the village, there will be a variety of agents of each class, embodying behaviors suitable for interacting with humans at different stages of learning.

We have two overriding implementation goals: one technical and one pedagogical. Our technical goal is to minimize the amount of knowledge engineering that goes into building and maintaining the village. Perhaps the largest cost associated with any educational software product is the amount of effort required to architect and enable domain-specific learning sequences for users. Our second, pedagogical goal is to create varied learning experiences for each participant, to accommodate different types of learners at different stages of development. We want a participant's experience to remain challenging and exciting as a s/he progresses.

In order to meet both of these goals, we are using *evolutionary computation* (EC) to evolve the agents that populate our virtual village. This methodology meets our first goal of minimizing knowledge engineering because the behaviors of the agents are controlled by neu-

ral networks and the neural networks are trained using human interactions in the village. This is based on the premise that the behavior or responses of one human can be used to teach another human how to behave or respond; thus an agent emulating the first human could also be used to teach humans how to behave or respond. The methodology also meets our second goal because of the way in which we have implemented the evolutionary techniques. While EC has typically been used to train one agent to emulate a single user (or type of user), here we use EC to train a population of agents that can interact with human participants in a variety of ways.

This paper describes methodologies for constructing the three classes of agent. As a prototype and to demonstrate the viability of the techniques, we draw from our prior work on two Internet games and use as the basis for training the agents human data collected in these games. We detail our methodology and describe examples for constructing instances of the first two agent classes. Then we discuss our current work which is involved in bringing an entire village to fruition within an educational resource web site.

## Theoretical framework.

The examples presented in the ensuing sections are based on data collected at two web sites, both of which contain games, one is educational and the other is purely for fun. Computer games are great for educational purposes, because of the motivational aspects they provide (Malone 1981; Soloway 1991; Brody 1993).

A game can be played by using a certain strategy, or set of strategies — a method and order for applying the rules of the game, with the intent of achieving the fixed goal. If we sat down and enumerated all the possible ways of playing a game, the result would typically be a huge list. So the question becomes a matter of *search*. Given a very large list of possible strategies, how can we find the ones that will result in achieving the game's goal? Many games are dynamic, so players must adjust to changes in environment, opponents and teammates; how can we adapt a player's strategies in accordance with these changes?

*Machine learning* has often been applied in attempts to answer these questions. Here, computer programs advance "automatically", developing better and more efficient ways to accomplish given tasks without needing humans to retrain them manually or update behavioral databases by hand. Since at least the 1950's, researchers have experimented with games including tic-tac-toe (Michie 1961; Angeline & Pollack 1993), checkers (Samuel 1959), chess (Shannon 1950) and backgammon (Berliner 1980; Tesauro 1992; Pollack, Blair, & Land 1996; Pollack & Blair 1998).

The following is an *evolutionary* approach to machine learning (Fogel 1962; Holland 1975; Koza 1992): rather than try to engineer a winning strategy, enumerate a manageable number of strategies, use these to play games and see how well they perform. Then keep the strategies that do well and use *selection* and *reproduction* techniques to replace the ones that do poorly with other strategies that have not yet been tried. Using this method, a population of successful strategies is built up gradually. At any time, the population will represent some ways of playing the game; eventually, hopefully, the population will contain the optimal way(s).

The definition of *optimal* varies depending on researchers' goals. The goal of the Deep Blue project was to create a chess player that could beat the human world champion. The goal of RoboCup is to develop a team of soccer-playing robots that are capable of defeating the human world champions (H. Kitano 1997). However, in some applications the goal is not for agents to embody experts but rather human peers. In an educational game, it is not always beneficial for a human to play with an expert; it is sometimes more desirable for human learners to interact with players whose abilities are similar to their own, providing motivation through appropriate challenges (Sklar 2000; Sklar & Pollack 1998).

Our goal is to characterize the types of human behaviors that work in various settings and to build agents that embody these behaviors, automatically deploying them as inhabitants in our virtual village. The agents can maintain a constant presence in the village, sustaining it when not enough humans are logged into the system. The system will be able to recognize which types of agents are needed in the village at any given time, depending on the behavior of the humans who are connected and what activity they are engaged in.

All of the activities in our village are modeled after games, so we have identified several characteristics of on-line games to help us distinguish the types of environments in which our agents will interact:

- *single player* vs *multi-player*
- *synchronous* (i.e., turn-taking) vs *asynchronous* (i.e., players do not wait between turns but may act continuously)
- *episodic* vs *non-episodic* (in an episodic game, all players make a move simultaneously, without knowledge of their opponents' moves, then the system processes all the moves and returns an outcome; examples include iterated prisoner's dilemma or silent auctions)
- *dynamic* vs *static* environment (in a dynamic game, changes that occur are not only due to moves of the other player(s), but the environment itself might be changing; e.g., in soccer, the ball keeps rolling even after a player contacts it, whereas in chess, once a player has made her move, the board remains unchanged until another move is made)
- *deterministic* vs *non-deterministic* (i.e., at any given time, a player has one or many choices of legal move(s) to make)
- *simple* vs *complex* strategy space (the branching factor in the game tree is a good measure of complexity)

- *accessible* vs *inaccessible* (i.e., player has access to all necessary information required to make an informed decision about what move to make next)
- *discrete* vs *continuous* strategy space (in some games, moves may be defined discretely, while with others, the difference between two moves may simply be a matter of degree)
- *time-critical* vs *non-time-critical* (i.e., value of a player's move depends on how fast she makes it)

Over the last few years, we have been building and experimenting with different games that exhibit some of these characteristics. Several of these have been implemented as interactive games on the Internet. We describe two of the games in the ensuing sections.

## Tron.

Tron is a video game which became popular in the 1980's, after the release of the Disney film with the same name. We characterize Tron as: multi-player, asynchronous, non-episodic, environmentally static, non-deterministic, simple, accessible, discrete and time-critical.

In Tron, two futuristic motorcycles run at constant speed, making right angle turns and leaving solid wall trails behind them — until one crashes into a wall and dies. In earlier work (Funes *et al.* 1998), we built a Java version of the Tron game and released it on the Internet[1] (illustrated in figure 1). Human visitors play against an evolving population of intelligent agents, controlled by genetic programs (Koza 1992). During the first 30 months on-line (September 1997 through April 1999), the Tron system collected data on over 200,000 games played by over 4000 humans and 3000 agents.

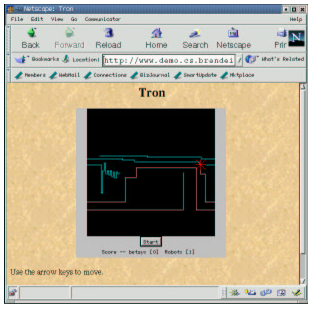


Figure 1: The game of Tron.

In our version of Tron, the motorcycles are abstracted and are represented only by their trails. Two players — one human and one software agent — each control a motorcycle, starting near the middle of the screen and heading in the same direction. The players may move past the edges of the screen and re-appear on the opposite side in a wrap-around, or *toroidal*, game arena. The size of the arena is $256 \times 256$ pixels.

The agents are provided with 8 simple sensors with which to perceive their environment (see figure 2). Each sensor evaluates the distance in pixels from the current position to the nearest obstacle in one direction, and returns a maximum value of 1.0 for an immediate obstacle (i.e., a wall in an adjacent pixel), a lower number for an obstacle further away, and 0.0 when there are no walls in sight. The game runs in simulated real-time (i.e., play is regulated by synchronized time steps), where each player selects moves: $left$, $right$ or $straight$.

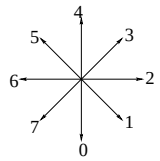


Figure 2: Agent sensors.

Our general performance measure is the **win rate**, calculated as the number of games won divided by the number of games played. Figure 3 illustrates the distribution of performances within the human population, grouped by (human) win rate for the fifty-eight humans who played the most games on the site during the first 30 months of the experiment.

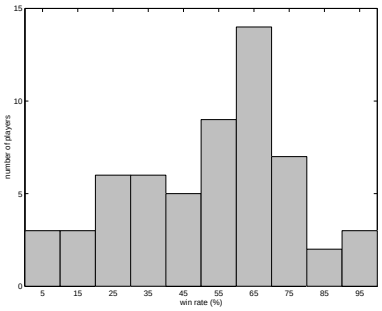


Figure 3: Distribution of win rates of human players who participated in the Tron Internet experiment.

## Keyit.

Keyit is a simple two-player typing game in which participants are each given ten words to type as fast as they can (see figure 4) (Sklar 2000). We characterize Keyit as: multi-player, asynchronous, episodic, environmentally static, deterministic, simple, accessible, discrete and time-critical.

Both players are presented with the same set of words, selected automatically from a dictionary, displayed one at a time and in the same order. For each player, a timer begins when she types the first letter of a word and stops when she presses the *Enter* key to terminate the word — at which time, the system

[1] http://www.demo.cs.brandeis.edu/tron

presents her with the next word to type. Players are scored based on speed and accuracy.

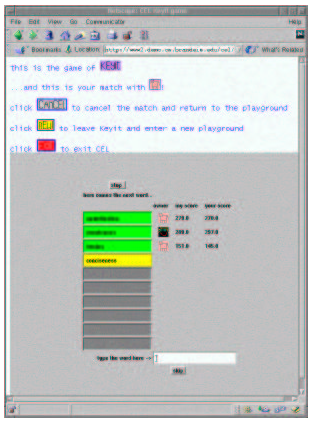

Figure 4: The game of Keyit.

Each word in the dictionary is characterized by a vector of seven feature values: word length, keyboarding level[2], Scrabble score, number of vowels, number of consonants and number of 2 and 3-consonant clusters. These feature values are used in attempt to capture the relative difficulty of each word.

Our general performance measure is the **typing speed**, calculated in letters per second. During the first half of 1999, we conducted a 6-month classroom study involving forty-four 10-12 year old students. Figure 5 illustrates the distribution of performances within the student population, grouped by typing speed.

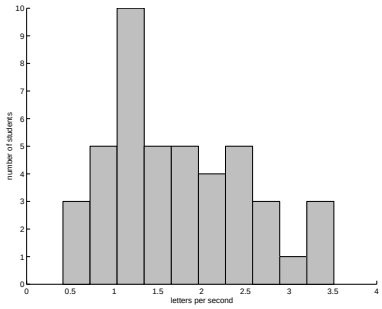


Figure 5: Distribution of typing speeds of students who participated in the Keyit classroom experiment.

## Methodology.

This section describes the methodology used to evolve agents that can play each of the games discussed in the previous section. All the agents are controlled by neural networks, and here we outline the architecture of each network as well as the training methods employed. Note that these are similarly structured, despite the variants between the domains.

[2]Based on a standard order for introducing keys to students learning typing.

The task for a Tron agent is as follows: given the state of the arena, as determined by evaluating the eight sensors, decide whether it is best to turn left or right or to keep going straight. Play is controlled through simulated time steps, and this decision is made at each time step.

For training Tron agents, we used game data collected on the Internet site. This includes the content of each game, i.e., every turn made by either player, the global direction of the turn and the time in the game at which the turn was made. There were 58 humans who played more than 500 games on our Internet site during the first 30 months of data collection. In earlier work (Sklar *et al.* 1999), we trained agents to play Tron using games played by these humans as the training set. Note that we split this data set in half and reserved one half for post-training evaluation.

The Tron agents are controlled by a full-connected, two-layer, feed-forward neural network, as illustrated in figure 6. Each network has 8 input nodes (one for each of the sensors in figure 2), 5 hidden nodes and 3 output values. Each output represents a value of merit for choosing each of the three possible actions (*left*, *right*, *straight*); the one with the largest value is selected as the action for the agent.

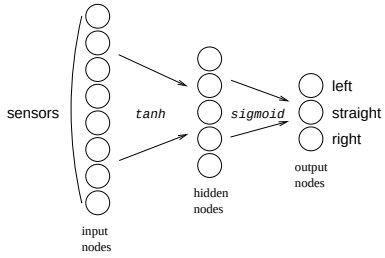


Figure 6: Agent control architecture.

We trained agents using *supervised learning* (Pomerleau 1993; Wyeth 1998), designating a player to be the *trainer* and replaying a sequence of games that were played by that player against a series of *opponents*. We suspended play after each simulated time step and evaluated the sensors of the trainer. These values were fed to a third player, the *trainee* (the agent being trained), who would make a prediction of which move the trainer would make next. The move predicted by the trainee was then compared to the move made by the trainer, and the trainee's control mechanism was adjusted accordingly, using the backpropagation algorithm (Rumelhart, Hinton, & Williams 1986).

The task for a Keyit agent is as follows: given a word, characterized by its corresponding set of seven feature values, output the length of time to type the word. In addition to using the feature values for input, we also

consider the amount of time that has elapsed since the previous word was typed.

For training Keyit agents, we used game data collected on the Internet from the 44 students who participated in the classroom study described in the previous section (Sklar 2000). For each student, we gathered all the moves from all games of Keyit. A "move" includes a timestamp, the word being typed, the amount of time that the player took to type the word and the time that had elapsed between moves. We split this data set in half and reserved one half for post-training evaluation.

The Keyit agents are controlled by fully-connected, two-layer feed-forward neural networks. The network architecture is shown in figure 7. There are 8 input nodes, corresponding to each of the seven feature values (normalized) plus the elapsed time. The elapsed time is partially normalized to a value between 0 and (close to) 1. There are 3 hidden nodes and one output node, which contains the time to type the input word, in hundredths of a second.

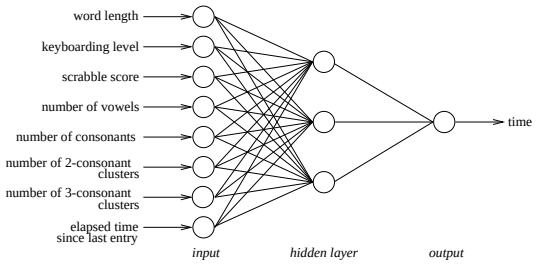


Figure 7: Neural network architecture.

Again, we used supervised learning to train the agents, designating a player to be the trainer and replaying a sequence of games. For each move in a game, the network predicted the trainer's speed for that move based on the feature vector of the word to type and the length of time that elapsed since the last move. Based on the accuracy of the trainees' predictions, the network weights were adjusted using backpropagation.

## Clones.

**Clones** are agents that capture the behavior of an individual human. The goal in training a clone is for it to emulate the human as closely as possible. We have trained clones for both Tron and Keyit.

From the Tron data set, fifty-eight clones were produced and figure 8 shows the results. The win rate of each trainee is compared with its trainer. If the results were perfect, then each mark on the plot would fall on a line of slope 1.

From the Keyit data set, forty-four clones were produced and figure 9 shows the results. The typing speed for the trainees (horizontal axis) versus their trainers (vertical axis) is shown, for both the test and training

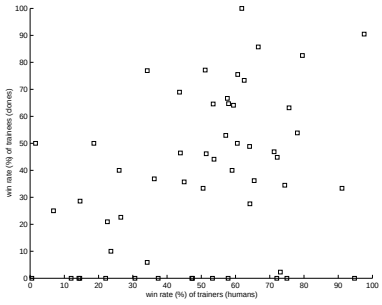


Figure 8: Tron clones.

data sets. Again, if the results were noiseless, then each mark would fall on a line of slope 1.

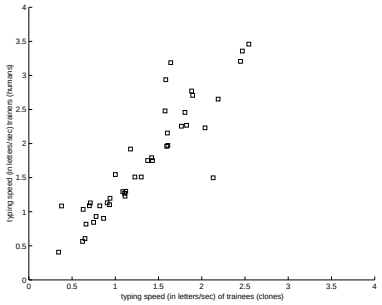


Figure 9: Keyit clones.

## Peers.

**Peers** are agents that represent the behavior of a group of human users. Users can be clustered based on a feature like age or gender or win rate (of a game), and the behavioral data for all humans exhibiting the same feature value can be grouped and analyzed, in an attempt to recognize characteristics of different user groups. Individual peers are intended to be representative of all the humans within a single cluster. Populations of peers are meant to be representative of all the clusters. We have trained peers in two domains: Tron and Keyit.

From the Tron data set, ten peers were produced, by dividing the 58 individual humans into 10 groups based on their win rates (e.g., group 1 had 0-10% win rate, group 2 had 10-20% win rate, etc.). Since the objective with peers is to produce a small group of agents representative of a larger population, the direct correlation between trainer and trainee is less important to evaluate. Instead, we look at the distribution of peers across the range of characteristics they are intended to represent — in this case, win rate.

Figure 10 shows these results. The peer trainers (i.e., grouped human data) and trainees are sorted within each population according to their win rate, so the comparison is not a direct one between individual trainees and their trainers, but rather a population-based com-

parison looking at the distributions of the trainer and trainee populations. The horizontal lines denote boundaries for grouping players (according to win rate). The plot demonstrates that the controllers have learned to play Tron at a variety of different levels and that, as a whole, the trainee population is representative of the respective trainer population.

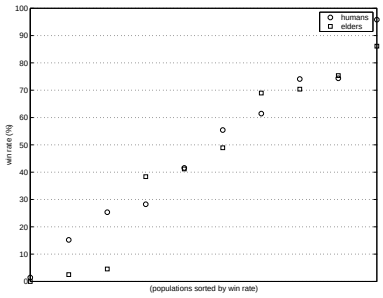


Figure 10: Tron peers

From the Keyit data set, ten peers were produced, by dividing the 44 students into eight groups based on typing speed. Group 1 – the slowest group – had a typing speed of less than 0.5 letters per second. Group 8 – the fastest group – had a typing speed of over 3.5 letters per second. Figure 11 compares the average speeds of the trainers and trainees, using the same population-based analysis described above for the Tron peer data. The comparison is made by first sorting both populations according to speed and then plotting the corresponding values. Again with Keyit, as with Tron, the resulting trained population is representative of the distribution of the original population.

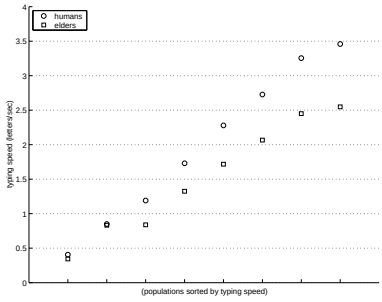


Figure 11: Keyit peers.

## Instructors.

**Instructors** are agents that emulate the behavior of a human expert. In some domains, there is always a right answer or a correct response, such as the correct spelling of a word or the solution to a arithmetical expression. In other domains, such as Tron, the "right" move at a given time is not deterministic.

For a simple domain like Keyit, we can define the behavior of an instructor merely by setting the typing speed and producing the correctly spelled word after a fixed amount of time has passed. For a complex domain like Tron, we can define instructors by using the technique described above for peers and only use the humans with the highest win rates to train the agents.

## Current and future work.

We have provided a theoretical framework for the development of a virtual village, populated by a variety of software agents geared towards education. The basis for our work is the pedagogical belief that students need to experience a variety of learning opportunities, by themselves, with peers and with teachers. So we have defined three classes of agents and we presented examples for constructing these agents using techniques from evolutionary computation.

Our village contains educational games, for reasons of human motivation. Thus we include in our framework a scheme for characterizing the features of on-line games so that as our work progresses, we can devise a myriad of agents that can behave in settings exhibiting various combinations of these features.

Our current work involves deploying all three classes of agents in our on-line educational system. We are continuing to develop educational games — more complex than those described here — exhibiting the variety of features listed in section  and reinforcing particular curricular topics, as advised by classroom teachers. As well, we are building agents to help humans navigate our educational web site, which contains not only games, but also resources for teachers including a database of lesson plans and information on classroom technologies.

In our view, learning doesn't stop just because a student leaves school, either at the end of a school-day or upon graduation. Rather, life itself is a learning experience and the everyday interactions one has with people all around can provide additional learning experiences as well. Thus, the long-term goal of this work is to develop techniques for automatically building interactive agencies that can help humans navigate through any virtual setting.

## Acknowledgments.

Special thanks to Pablo Funes, Alan Blair and Jordan Pollack for their collaborations on the Tron project.

## References

Angeline, P. J., and Pollack, J. B. 1993. Competitive environments evolve better solutions for complex tasks. In Forrest, S., ed., *Genetic Algorithms: Proceedings of the Fifth International Conference (GA93).*

Balabanović, M. 1998. *Learning to Surf: Multiagent Systems for Adaptive Web Page Recomendation.* Ph.D. Dissertation, Stanford University.

Berliner, H. J. 1980. Backgammon computer program beats world champion. *Artificial Intelligence* 14.

Brody, H. 1993. Video games that teach? *Technology Review* November/December.

Fogel, L. J. 1962. Autonomous automata. *Industrial Research* 4:14–19.

Foner, L. 1997. Yenta: A multi-agent referral based matchmaking system. In *Proceedings of the First International Conference on Autonomous Agents (Agents97).*

Funes, P.; Sklar, E.; Juillé, H.; and Pollack, J. B. 1998. Animal-animat coevolution: Using the animal population as fitness function. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior.*

Goldberg, D.; Nichols, D.; Oki, B. M.; and Terry, D. 1992. Using collaborative filtering to weave and information tapestry. *Communications of the ACM* 35(12).

H. Kitano, e. a. 1997. Robocup: The robot world cup initiative. *Proceedings of the First International Conference on Autonomous Agents (Agents-97).*

Holland, J. H. 1975. *Adaption in Natural and Artificial Systems.* University of Michigan Press.

Koza, J. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Kuokka, D., and Harada, L. 1997. Matchmaking for information agents. In Huhns, M., and Singh, M., eds., *Readings in Agents.* Morgan Kaufman.

Lang, K. 1995. Newsweeder: Learning to filter news. In *Proceedings of the Twelfth International Conference on Machine Learning.*

Lashkari, Y.; Metral, M.; and Maes, P. 1994. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.* AAAI Press.

Lieberman, H. 1995. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Malone, T. 1981. What makes computer games fun? *Byte* December.

Michie, D. 1961. Trial and error. *Science Survey* part 2:129–145.

Pollack, J. B., and Blair, A. D. 1998. Co-evolution in the successful learning of backgammon strategy. *Machine Learning* 32:225–240.

Pollack, J. B.; Blair, A. D.; and Land, M. 1996. Coevolution of a backgammon player. In Langton, C., ed., *Proceedings of ALIFE-5.* MIT Press.

Pomerleau, D. 1993. *Neural Network Perception for Mobile Robot Guidance.* Kluwer Academic.

Qureshi, A. 1996. Evolving agents. In *Proceedings of the First International Conference of Genetic Programming (GP-96).*

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning representations by back-propagating errors. *Nature* 323.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:210–229.

Shannon, C. E. 1950. Programming a computer for playing chess. *Philosophical Magazine [Series 7]* 41.

Sklar, E., and Pollack, J. B. 1998. Toward a community of evolving learners. In *Proceedings of the Third International Conference on the Learning Sciences (ICLS-98).*

Sklar, E.; Blair, A. D.; Funes, P.; and Pollack, J. B. 1999. Training intelligent agents using human internet data. In *Proceedings of Intelligent Agent Technology (IAT-99).*

Sklar, E. 2000. *CEL: A Framework for Enabling an Internet Learning Community.* Ph.D. Dissertation, Brandeis University.

Soloway, E. 1991. How the nintendo generation learns. *Communications of the ACM* 34(9).

Tesauro, G. 1992. Practical issues in temporal difference learning. *Machine Learning* 8.

Wyeth, G. 1998. Training a vision guided robot. *Machine Learning* 31.