Agents for Education: When too much intelligence is a bad thing.

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

ABSTRACT

Intelligent agents are frequently designed to be personal assistants, helping a single user accomplish a specific task. The work discussed here explores the idea of building intelligent agents for use in on-line educational environments, where helping a user too much can get in the way of the user's learning. We offer three categories of agents, designed to meet the varied and changing needs of a population of human learners.

Keywords

Education, learning, evolutionary computation

1. INTRODUCTION.

The overarching goal in an educational setting is for the student to learn. An intelligent agent that is successful in this type of environment is not the same as in traditional agent-based settings where agents act as browsing assistants [14], matchmakers [5, 11], recommenders [2] and filterers of email and news group messages [7, 13, 12]. The purpose of an educational agent is not to perform a task *for* a user or to simplify a task for a user, but rather to help the user learn how to accomplish the task.

Malone [15] makes an important distinction between *toys* and *tools* when discussing computer games. He defines toys to be systems that exist for their own sake, with no external goals; in contrast, tools are systems that exist because of their external goals. Good games are difficult to play, in order to increase the challenge provided to the player. Good tools should be easy to use, in order to expedite the user's external goal.

Good agents are like good tools. Good educational agents should be both easy to use and should provide challenges for the human learner. The external goal is for the user to learn how to perform a given task, so the agent should make the process of learning how to accomplish that task easy — the *process*, not the task. The work discussed here explores the idea of building intelligent agents for use in on-line educational games, where helping a user too much can get in the way of the user's learning.

We posit 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 participating in group projects where they interact with their peers and from doing their homework on their own. To support these needs, we define three categories 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 have the opportunity of interacting with only one or a combination of two or all three categories of agent, just as at different stages in a learner's development, she will need to receive instruction from a teacher, collaborate with her classmates and practice on her own.

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 agents. Perhaps the largest cost associated with any educational software product is the amount of effort required to design 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 she progresses.

In order to meet both of these goals, we are using *evolutionary computation* (EC) to evolve the agents. This methodology meets our first goal of minimizing knowledge engineering because the behaviors of the agents are controlled by neural networks and the neural networks are trained using human interactions with our system. 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 one human could be used to teach other 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 at a variety of levels.

This paper describes methodologies for constructing the three categories of agent. As a prototype and to demonstrate the viability of the techniques, we draw from our prior work on two Internet games and we take human data collected in these games as the basis for training the agents. We detail our methodology and describe examples for constructing instances of the first two agent categories. Then we outline our current work which is involved in developing a control system for deciding which agents to deploy under what conditions and bringing these agents to life within an on-line educational environment.

2. THEORETICAL FRAMEWORK.

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 [1, 16], checkers [21], chess [22] and backgammon [3, 26, 18, 17].

The following is an *evolutionary* approach to machine learning [4, 8, 10]: 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 [9]. However, as described earlier, in some situations 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 [23, 25].

Our longterm goal is to characterize the types of human behaviors that occur in various settings and to build agents that embody these behaviors, automatically deploying them as needed. Our system will recognize which types of agents are required at a given time, depending on the behaviors of the humans who are connected and what activities the humans are engaged in.

We begin by identifying several characteristics of on-line game environments:

- *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)

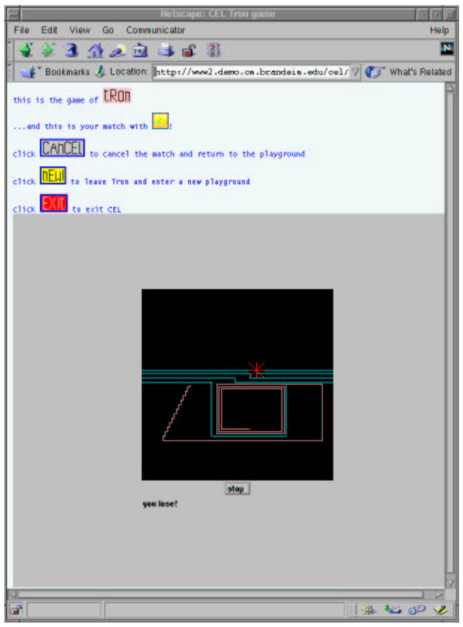
Over the last few years, we have been building and experimenting with different on-line games, each exhibiting some of these characteristics. Several of the games have been implemented on the Internet, and we have collected a significant amount of human interaction data with these games. This data becomes the basis for training the agents using evolutionary techniques.

We begin by describing two of the games in the ensuing sections, as background for understanding the training techniques presented in the remainder of the paper.

2.1 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 [6], 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 [10]. 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.



this is the game of tROn

...and this is your match with !

click CAnCEL to cancel the match and return to the playground

click nEW! to leave Tron and enter a new playground

click EXIt to exit CEL

you lose!

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 × 256 pixels.

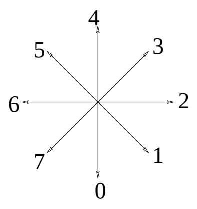


Figure 2: Agent sensors.

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*.

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.

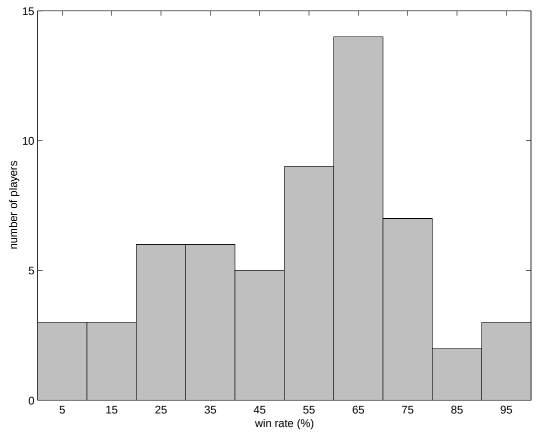


Figure 3: Distribution of win rates of human players who participated in the Tron Internet experiment.

2.2 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) [23]. We characterize Keyit as: multi-player, asynchronous, episodic, environmentally static, deterministic, simple, accessible, discrete and time-critical.

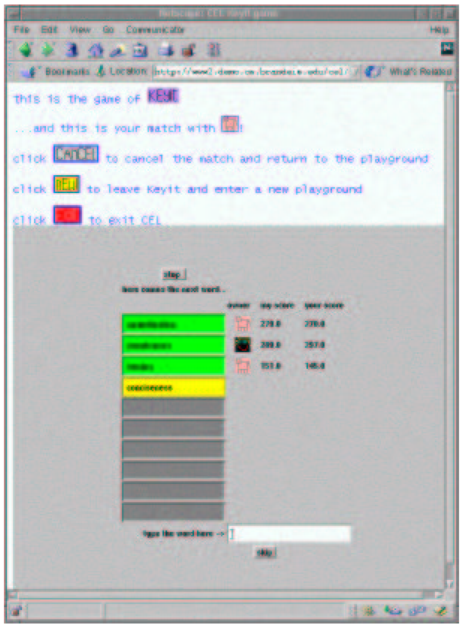
Both players are presented with the same set of words, selected automatically from a dictionary, displayed one at a time and in the same order. For each player, a timer begins when she types the first letter of a word and stops when she presses the *Enter* key to terminate the word — at which time, the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

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.

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

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



this is the game of KEYIT

...and this is your match with

click CANCEL to cancel the match and return to the playground

click NEW to leave Keyit and enter a new playground

click to exit CEL

Figure 4: The game of Keyit.

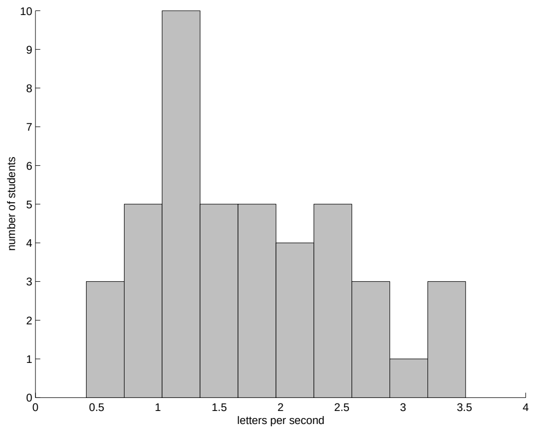


Figure 5: Distribution of typing speeds of students who participated in the Keyit classroom experiment.

3. METHODOLOGY.

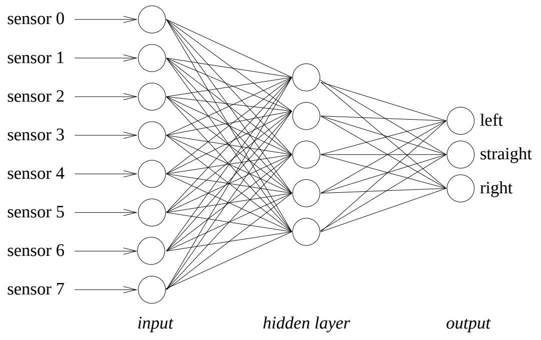
This section describes the methodology used to evolve agents that can play each of the games discussed in the previous section. All the agents are controlled by neural networks, and here we outline the architecture of each network as well as the training methods employed. Note that these are similarly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

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.

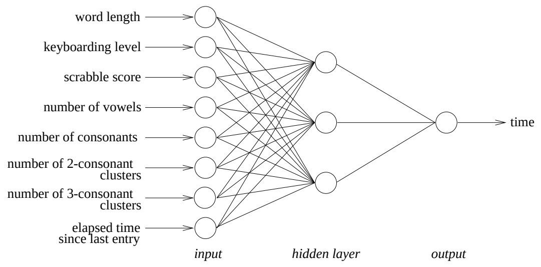
For training Tron agents, we used game data collected on the Internet site. This includes the content of each game, i.e., every turn made by either player, the global direction of the turn and the time in the game at which the turn was made. There were 58 humans who played more than 500 games on our Internet site during the first 30 months of data collection. In earlier work [24], 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 6a. 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.

We trained agents using *supervised learning* [19, 27], 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 [20].



a. Tron controller.



b. Keyit controller.

Figure 6: Agent control architectures.

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 [23]. 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 6b. 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.

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.

4. EXAMPLES.

This section presents examples of using our methodology to create each of the three types of agents: clones, peers and instructors.

4.1 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. Playing a game with one's clones is a form of self-play or solitaire, but here enabled in a multi-player game. This mode has the added advantage of allowing players to spot weaknesses in their own games. For the human learning task, we consider this to be a means of practicing skills alone, like doing homework on one's own.

We have trained clones for both Tron and Keyit. From the Tron data set, fifty-eight clones were produced and figure 7a shows the results. The win rate of each trainee is compared with its trainer. From the Keyit data set, forty-four clones were produced, as illustrated in figure 7b. The typing speed for the trainees (horizontal axis) versus their trainers (vertical axis) is shown, for both the test and training data sets. For both graphs, if the results were perfect (i.e., noiseless), then each mark on the plots would fall on a line of slope 1.

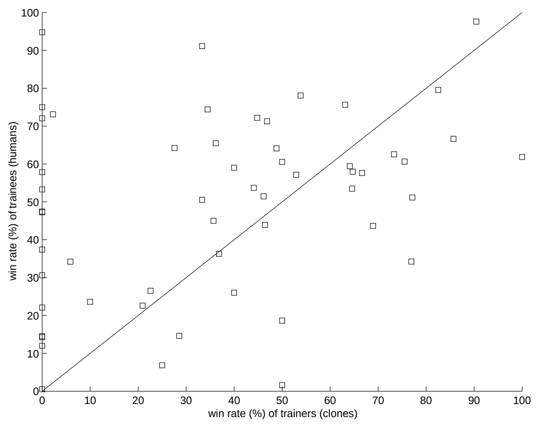
4.2 Peers.

Peers are agents that represent the behavior of a group of human users. A human interacting with a set of peers is akin, in the human learning realm, to a student doing a group project within her classroom. Here, students are grouped by age (because they are in the same classroom) and then, within the classroom, groups may be defined in a variety of ways. For example, groups may be formed by the teacher according to ability — putting all the "smart" kids in the same group or putting one smart kid in each group.

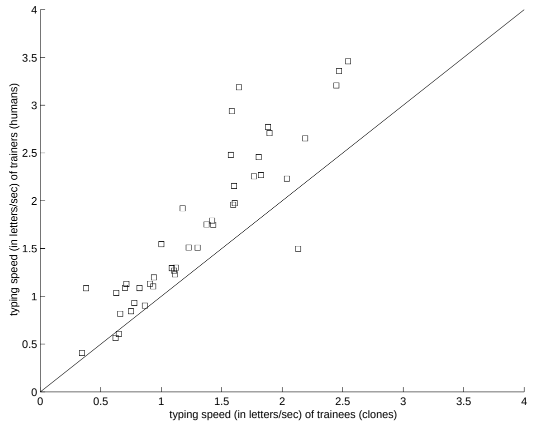
For the purpose of training agents as peers, we select a performance metric (e.g., win rate of a game), and then group together the behavioral data for all humans exhibiting the same (or similar) metric. We then train the agents using this composite data set. The result is a population of peers, where individual peers are intended to be representative of all the humans within a single grouping. The whole population of peers is meant to be representative of all the groupings.

We have trained peers in both of our test domains. 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.). Figure 8a shows the correlation between trainers (humans) and trainees (peers). The axes are the same as in figure 7. While the correspondance is fairly good overall, it is interesting to note that the human with the highest win rate produced a clone with one of the lowest win rates. We speculate that the human player's behavior patterns were noisy and thus it was difficult to train a reasonable agent. We are investigating this further by comparing various behavioral statistics from the human population with the correlation coefficients in the training exercise.

From the Keyit data set, eight 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.

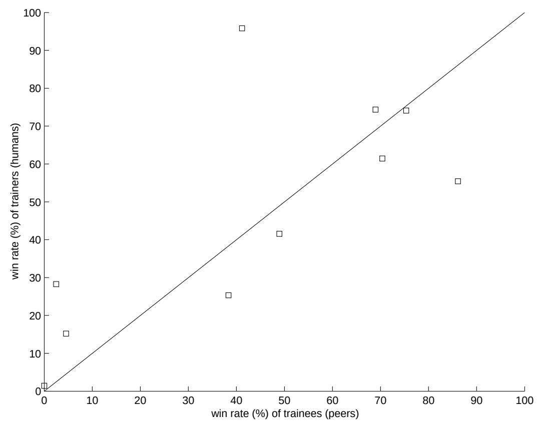


a. Tron clones.

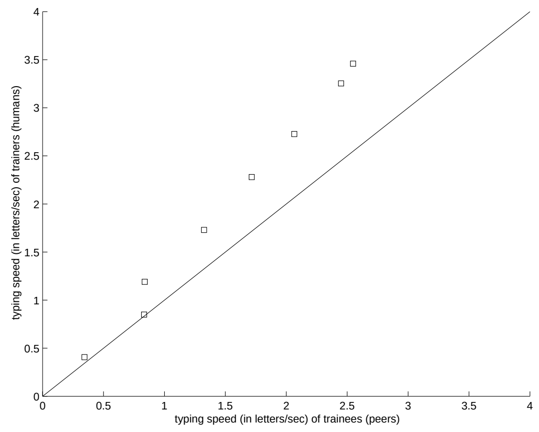


b. Keyit clones.

Figure 7: Training clones.



a. Tron peers.



b. Keyit peers.

Figure 8: Training peers.

Figure 8b compares the average speeds of the trainers and trainees, again using the same axes as in the previous figures.

4.3 Instructors.

Instructors are agents that emulate the behavior of a human expert. This translates into a teacher being available to provide a student with the right answer to a problem or the "right" way to accomplish a task.

In some domains, there is always a right answer or a correct response, such as the correct spelling of a word (in Keyit) or the solution to a arithmetical expression. In other domains, such as Tron, the "right" move at a given time is not deterministic. We make the assumption that a good Tron player — one with a high win-rate — exhibits the right way to play the game.

Thus, 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 define instructors by using the technique described in section 4.2 and only use the humans with the highest win rates to train the agents.

5. DISCUSSION.

The clone and peer agents correlate better with their human trainees within the Keyit domain than in the Tron domain. We believe that there are several factors contributing to this situation. First, the Keyit game uses a simple, static environment, whereas the Tron environment is dynamic. Second, the amount of training data we had for Tron was much greater than that for Keyit, and the human population participating in the Tron experiment had much wider demographics; these factors account for the naturally occuring variations found in such a large data set. One aspect of our current efforts is to improve on the correlation between trainers and trainees in dynamic environments.

While we have developed agents for the domains described, we are currently working on methods of deploying the agents and, in particular, knowing which type of agent to deploy under which conditions. Ideally, we would like our system to choose the proper agent for a user to interact with, given the user's behavior with the system. This is not a simple task. It depends not only on a user's performance with the system, but also on the user's motivation. The correct choice of agent at the right time will mean the user is continually challenged and, through this challenge, motivated.

We have designed a simple control mechanism that probabilistically chooses which type of agent to deploy (figure 9). The next step is to learn the correct probabilities for each student and adjust them as the student progresses. This will be done using a simple network and an evolutionary algorithm.

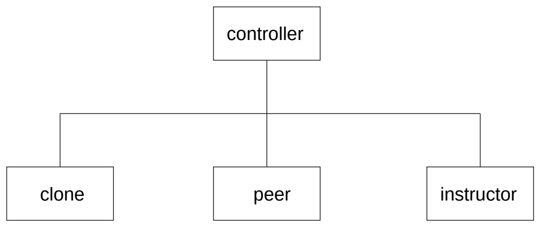


Figure 9: Simple control architecture.

After we complete the control system, we will embed it in our existing on-line educational system[3]. We are continuing to develop educational games — more complex than those described here — exhibiting the variety of features in our framework and reinforcing particular curricular topics, as advised by classroom teachers. Pilot studies will be conducted with students, in cooperation with teachers, to test the effectiveness of the system, both from pedagogical and motivational standpoints.

6. SUMMARY.

We have provided a theoretical framework for building 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. Thus we have defined three categories of agents and presented examples for constructing these agents using techniques from evolutionary computation. We include in our framework a scheme for characterizing the features of on-line interactive environments in order to help measure the progress of our work and the robustness of the agents produced. The next step is to deploy

[3]http://satchmo.cs.columbia.edu/tip

these agents in a live environment and pilot test them with students in classrooms.

7. ACKNOWLEDGMENTS.

We extend our gratitude to Pablo Funes, Alan Blair and Jordan Pollack for their collaborations on the Tron project. Special thanks to Simon Parsons for constructive comments, as always. This work was supported in part by NSF #REC-0113317.

8. REFERENCES

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

[2] M. Balabanović. *Learning to Surf: Multiagent Systems for Adaptive Web Page Recomendation.* PhD thesis, Stanford University, 1998.

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

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

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

[6] P. Funes, E. Sklar, H. Juillé, and J. B. Pollack. 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*, 1998.

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

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

[9] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, 1997.

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

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

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

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

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

[15] T. Malone. Toward a theory of intrinsically motivating instruction. *Cognitive Science*, 4:333–369, 1981.

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

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

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

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

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

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

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

[23] E. Sklar. *CEL: A Framework for Enabling an Internet Learning Community.* PhD thesis, Brandeis University, 2000.

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

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

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

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