

The Study of Emotion, Learning and Intelligence

Jason Williams

Department of Computer and Information Science

University of Massachusetts, Dartmouth

Phone:

g_jwilliams@umassd.edu

Xiaoqin Zhang

Department of Computer and Information Science

University of Massachusetts, Dartmouth

Phone: 508-999-8294

X2zhang@umassd.edu

ABSTRACT

Emotion has been shown to be an important factor in a human's decision making and learning processes. Affective computing is to simulate the human's emotion mechanism with computer programs. Meanwhile, reinforcement learning is an advanced machine learning technique that has attracted a lot of attention recently. However, it has difficulty learning in a dynamic world or in a large state space. The general goal of our study is to discover the relationships between emotion and learning. The first step is to find whether and how affective computing mechanism can be applied to reinforcement learning and help it overcome some of its limitations. We created two agent architectures, one based on affective computing and the other on active reinforcement learning, where we use some abstraction and modeling techniques driven from the affective computing to build the world model for the learning agent. Experiments have been performed to study the performance of both the affective computing agent and the modified reinforcement learning agent in a large, dynamic grid world environment.

Categories and Subject Descriptors

I.2.0 [Artificial Intelligence: General]

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Human Factors, Verification.

Keywords

Affective Computing, Reinforcement Learning, Artificial Intelligence

1. INTRODUCTION

A new endeavor in the field of artificial intelligence has been the study of emotions and its application. Studies in fields of study such as neurophysiology, cognition, and psychology have shown that emotion plays an important role in processes such as cognitive processing, and memory [1]. Bodenhausen et al. [2] studied the effects of different kinds of negative affect on undergraduate students. They found that anger led to more heuristic processing than sadness did. Studies conducted by

Smith et al. [3] have shown that emotions were directly related to appraisal of the situation rather than to attribution. Schwarz [4] has shown the importance of the interplay between emotion, cognition and decision making.

Research in this new field, which Dr. Pickard [5] given the name affective computing, is showing the importance of affect in decision making in computing systems. There are research projects in the area of virtual tutors [6,7], social robots [8], virtual training environments [9,10], and agents targeted at decision-making [11]. With all of the interest in computational models of affect, there is at least one researcher however, who sees the potential for inappropriately assigning the label of affect to systems that do not deserve it. Dr. Scheutz is warning researchers to proceed cautiously in this area. He has pointed out that it is difficult to make a precise notion of emotion [12]. This difficulty is made worse by the temptation to anthropomorphize the states of a computer agent. This tendency stems from our relatively young understanding of the intricacies of emotion. Even though we have been studying emotions for quite some time, our understanding is still limited.

In performing the research for this paper, we endeavored to create an affective computer agent that would avoid the pitfalls warned of by Dr. Scheutz and others. We strove to create a system that did not inappropriately assign the label of affect, but one that would also add something new and useful. Since our understanding of emotion is still relatively basic, we implemented an affect system that was likewise basic. Regardless of the theory, emotion can be described as a driving or motivational force. We applied this thinking in our study.

We took a well-known action selection problem from the set of Markov Decision Problems (MDP), specifically a gridworld type problem, and applied an affective motivation force agent to it. The typical agents used to solve gridworld type MDP's are reinforcement learning agents. A limitation to reinforcement learning agents, however, is the size of the problem that they are able to solve. As the dimensions of the world increase, and the number of states increases, utilizing past learning becomes harder and the problem becomes intractable. A similar limitation is introduced if the world is dynamic rather than static. As the location and number of reward states in the environment changes, applying past learning to the current situation is difficult at the very least. This study looks at a large, dynamic gridworld, where the number and location of reward states is in a frequently changing. We created an affectively motivated agent to solve the problem, then applied what we had learned from the creation of the agent to more traditional reinforcement type agent to see if it could be improved on.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '04, Month 1–2, 2004, City, State, Country.

Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

The remainder of the paper is organized as follows: section two discusses the design for the experiments, section three discusses the design of the two action selection agents, section four discusses the results of the experiments, and section five is the conclusion with a look at future work.

2. Dynamic Grid World Environment

The testing environment is a dynamic grid world with 100 x 100 grids, surrounded by walls and containing two types of reward objects: a positive reward, or the goal object, and a negative reward, or the penalty object. When the state space is defined in terms of physical location within the environment, there are 10,000 unique states.

When an agent occupies the same location as a reward object, either goal or penalty, the object is collected and the reward points are added to or removed from the agent's score, depending on the nature of the reward. Each trial in the investigation will be run for a specified amount of time, and each agent's fitness for the environment will be evaluated based on their achieved scores during that time. Objects are randomly created throughout the environment, as well as randomly throughout the duration of the trial. A random distribution of objects would create a high probability of having several objects available for the agent to choose between. Thereby allowing us to study how each agent performs when presented with several goals, all with the same reward value. The relationships between the objects, in this type of scenario, become more important than the object's intrinsic value. Each object a life time of random length, which determines how long an object would remain in the world if it were not collected.

3. Agent Design

Two types of problem-solving agents were constructed for this investigation: an affective agent (affectiveAgent), and an adaptive dynamic programming, active reinforcement agent (activeAgent). Each agent had 4 possible actions, one step move to an adjacent square either horizontally or vertically. Moving diagonally was not permitted. Both problem-solving agents have the same underlying structure. They perceive the updated environment states at each time pulse. It is assumed the environment is fully accessible in terms of the locations and types of each objects. The agents then make decisions on how to move in this grid world.

[Jason: do you have two agents working at the same time, what happens if both of them decide to move to the same spot?]

The differences between the agents were in the methods for selecting their next action, the area of focus for the study.

3.1 Affective Agent

In contrast to traditional agent learning systems, our affective agent is not externally motivated, meaning it is not motivated by the utility of the states in the environment. The affective agent is

created to be internally motivated. Its motivation comes from what it "feels" towards the other objects in the world.

Feelings are defined as force vectors between an agent and an object. In this experiment, we looked at the interaction of the forces generated by the agent when presented with only two types of objects: one that will have a positive influence on the agent, and one that will have a negative influence. The force of the influence for both classes of objects are proportional to distance between the agent and the object. The closer the agent is to the object, the stronger the resulting force between them. The direction of the vector depends on the agent. If the agent determines that the reward is good, or beneficial, then direction of the vector will be towards the object. And vice versa, if the reward is not beneficial then the direction of the vector will be away from the object. In this study the determination of the benefit of the object was made to be simple. If the reward increased the score of the agent, it was good, otherwise it was not.

The affectiveAgent's action is selected in the function call *NextStep*. This function evaluates the affect the agent is feeling, and moves in the direction of the greatest influence. The influence between an agent and an object has been represented as a physics style vector, meaning that it is a force that has both a magnitude and a direction. The vector can be represented as an arrow originating from the agent and pointing in the direction to indicate the influence from the reward object. A vector for a positive influence will point from the agent towards the object, while a vector for a negative affect will point from the agent away from the object – in the opposite direction.

For example, suppose there is a positive reward object 2 spaces north of the agent and a negative reward object 3 spaces south of the agent (see figure 1). If these were the only two objects in the environment, the agent would have two influences action on it. The first would be a pull affect, or force, that it feels towards the positive, goal object, and the second would be a push or repellant affect towards the negative reward. The force that the agent feels towards the positive reward is stronger than the one it feels towards the negative reward, since the positive reward is closer to it. Both forces are motivating the agent to move in the same direction, so the *NextStep* function would return an action of moving north.

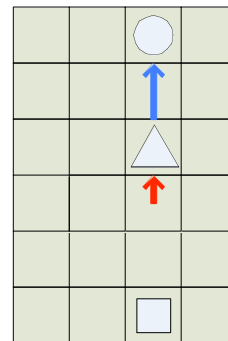


Figure 1

Things are rarely this straight forward, however. Most of the time the objects will not be in a straight line with the agent. And most of the time the influences on the agent will be pulling or pushing the agent in different directions. To deal with that, each influence vector was split into two components, corresponding to the x and y coordinate directions. In the case where there is an object 2 spaces on the diagonal away from the agent, the influence between the agent and the object would be split into an x component and a y component (see figure 2).

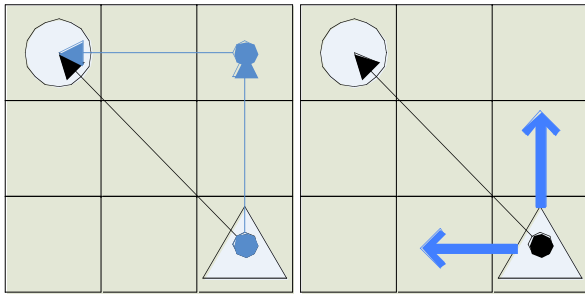


Figure 2a

Figure 2b

The force of the individual components, when added together, equals the force and direction of the original vector. The influence's magnitude in a given direction is equal to the fraction of the total magnitude that is pushing or pulling in that direction. To demonstrate, assume we have a single positive reward object. The agent is located at position (1, 1) in the world. The object is located at position (8, 2). For every space that the agent wants to move in the y direction, it wants to move 7 spaces in the x direction. When the magnitude of the vector is split, the resulting force along the x axis will be 7 times stronger than the force along the y axis.

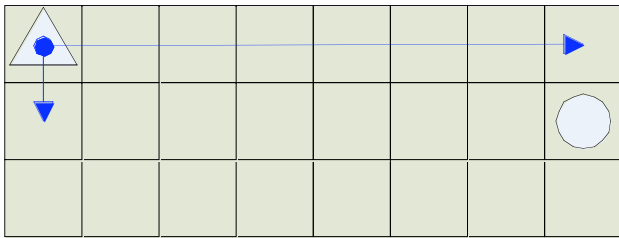


Figure 3

This provides a convenient way to represent the magnitude of the influence in terms of the possible actions of the agent. For this study we have stipulated that objects do not move as a result of the influence between in and an agent. In a future study we want to look at the situation where objects do move. We want to study the effect on the agent of different movement strategies of the object, but for this study the only way that objects move is when they are added to or removed from the environment, or when they are collected by an agent.

The heart of the affective system is in determining the strengths of the influences and being pushed or pulled in the direction that is the strongest. The strength of influence is calculated by:

$$Influence = (D_C / D_T) * (R(Obj) / 2^{(totalDistance-1)})$$

Where:

D_C = component distance to object

D_T = total distance

$R(Obj)$ = the reward value of the reward object

The reward is divided by 2, raised to the total distance between to the object. The fraction (D_C / D_T) determines how much of the total influence is working in the component direction. The strength of the influence is greatly affected by the distance between the object and the agent. Nearby objects dominant objects further away. The magnitude of the agent's affect for an object is proportional to the distance between the agent and the object. This is consistent with the paradigm of emotion behaving in a like manner as physical forces. Both gravity and electrical force grow stronger as the distance between objects decrease.

The effect of having the system with primary affect and no cognitive affect creates an agent that reacts to its environment, but doesn't learn from its reactions. The component forces of each object's influence on the agent, the push or pull force that the agent feels towards the object, are added together to determine the agent's best action. The affectiveAgent in this study only implements primary emotions. We will establish a baseline first with primary emotions, then in a future study add cognitive emotions and examine the impact they have on the system.

3.2 Active Reinforcement Agent

In order to evaluate the performance of the affective agent, we wanted to compare it an active reinforcement learning agent. As a starting point, we took an active dynamic programming (ADP) learning agent and modified it to work in the large, changing environment that the affective agent was tested in. A typical ADP agent, situated in a gridworld testbed, defines the state space in terms of x and y coordinates, or the physical location in the environment. In a dynamic environment, however, even a simple one where there is only one goal object and one pitfall object, and where the objects move only after it has been collected by an agent, the number of possible states, when defined by their coordinates, complicates an agent's ability to learn.

Each time an object is in a new location or each time the combination of the two objects forms a configuration that the agent has not previously experienced, it must either learn a new set of utilities, or try to apply what it has learned from previous configurations to this new one. If the agent had to relearn a new set of utilities for each new configuration of the environment, then the only time it can use what it has previously learned is when it sees an exact configuration that it has seen before. The probability of that occurring is small to begin with, and decreases as the state space increases. In an environment where there are several instances of each object type, where they are appearing and disappearing at uncertain time intervals, makes the problem that much more complicated. Add to that a large environment, having 10,000 states and the problem is intractable for the typical learning method.

To make the agent better suited for this environment, the state representation was redefined. States were redefined, based on what had been learned from creating the affective agent. The state that the agent occupied was determined by the distances to the reward objects. For example, assume that there is just one object in the world with the agent. The distance from the agent to the object is x . When the agent has learned the utility of a state that is x moves from the object, that can be applied to any location in the world that is also x units away from the object. With only the single object, the policy of the agent will be to simply move towards the object in the most direct method. Similar utilities can be learned for any distance away from the object. In a 100x100 grid world, with only horizontal and vertical moves permitted, the maximum distance that the agent could travel from one point to another, would be from any one corner to the corner diagonally opposite to it. The agent would have to make 198 moves, 99 horizontal moves and 99 vertical moves, regardless of the path that it followed. So in this environment, there is a maximum of 198 states, with their accompanying utilities, to learn.

For any move, there are only 3 possible outcomes: move to a state that is one step closer to the object, move to a state that is one step further away from the object, or move to a state that is the same distance from the object. Evaluating the worth of a move is therefore a simple matter of evaluating the utility of moving towards the object, moving away from it, or remaining in place.

A similar rule applies to an environment where there is just one negative reward. We make the assumption that the agent will want to move as far away from the negative object as it can. With this assumption, we can specify a utility for each of the location in the world based on the distance between that location and the location with the object in it. Evaluating the worth of moving to an adjacent location is the simple process of evaluating utility of moving towards the object, moving away from it, or staying the same distance relative to it.

[Jason: a picture is needed for the example below]

This can be extrapolated to environments containing any number of objects by treating the utility of location that the agent is in as the sum of utilities of the states of that location. For example, if there was a positive object north of the agent, and a negative object west of the agent, the best direction for the agent to select then can be determined by adding up the sub-utilities for each object. Let us assume for this example that the positive object is 5 steps north of the agent while the negative reward is 2 steps west and 1 step north of the agent. First for the positive object, moving north decreases the distance by one, increasing the utility accordingly. Moving either east, west, or south would increase the distance and therefore decrease the utility. For the negative reward, moving west would decrease the distance and also decrease the utility of the state, while moving in any of the other three directions, north, east or south, would increase the distance and utility. Added together, moving north is the only direction that increases both utilities, so it is therefore the best decision.

The same calculations can be made when the objects are not in direct line of movement. If the negative object had been to the southwest of the agent, two diagonal steps away, its distance would have been 4 (two horizontal steps and two vertical steps

away). In that situation, moving either south or west would have decreased the distance and the utility. So the agent would prefer a move to the north or to the east, which would have increased the utility.

The benefits of this way of defining the state space were being able to represent an unbounded number of reward objects in an extremely large state spaces in a usable manner. As more objects are added to the environment, their individual utilities are added as sub-utilities and are included in the calculation of the overall utility.

To determine the sub-utilities for each type of object, the activeAgent learned a policy in an off-line, two step process. Before the run started, the agent had two learning sessions, the first to learn the utility for one positive reward object - goal, and the second to learn the utility for one negative reward object - penalty. In each epoch, the agent and object were placed randomly in the world. The agent then learned the utility of states, relative to the object, by exploration. Before the agent had learned any utility values, it explores by random walking. Once it had learned the utility for at least one state, the agent developed a policy that would be used a percentage of the time to guide further exploration. The rest of the time, the agent continued to use random walking to explore. The more exploring the agent had done, the greater the weight given to the learned policy. Once a given threshold of visits to a particular state was exceeded, no further random walking was performed. The threshold for this experiment was set to be 100 visits. The training epoch concluded once the agent's policy converged.

[Jason, can you write the pseudo code of the above process, including the exploration rate, learning rate and the threshold value. Such pseudo code can be found in the AI book.]

Having the utilities defined this way lead to some situations where there was no increase in utility for any move, and therefore the agent choose to remain where it was. Once the environment changed, however, the agent would re-evaluate the situation and determine if a move was now warranted.

4. System Implementation

The implementation of our system uses the Multi Agent System Simulator (MASS) and the Java Agent Framework (JAF), both of which were developed at the Multi-Agent Systems Lab at the University of Massachusetts, Amherst.

[Jason, reference to MASS and JAF are needed]

MASS is an event-based simulator that works with JAF based, autonomous agents. The simulator is the central process of the system. Agents are connected to the system through socket connections. Since the agent are connected to the simulator and not directly to each other, the simulator also servers as a message router. Communication between agent and simulator, and as is more more often the case, between agents, happen by sending KQML based message events to the simulator, which then redirects them to the appropriate agent. Each agent runs as an independent process, and since they are connected to the simulator through sockets, they could be run from separate machines.

Each agent was developed from the JAF architecture, which provides a distinct environment from the simulator for agent specific logic. Each JAF agent has its own, subjective, representation of the environment. Their representations can vary from the true environment. In situations where the environment is not fully observable, each agent would only know that part that it could observe. For this investigation, however, the environment was fully observable, so each agent will have the same environment representation.

Control of the grid world was implemented in a JAF agent, which we named worldAgent. The worldAgent was responsible for storing and updating the state of the environment. The environmental state was comprised of the individual states of each of the current objects in it. The state of objects consisted of its location, duration, and reward. The worldAgent had control over the environment by controlling the creation, collection, and removal of the reward objects.

The JAF framework provides each agent with a pulse function that is called by MASS every time unit. Every pulse, the worldAgent removes any expired objects, attempts to create a new object, updates the data store of objects, and sends each actor agent a notification to begin their own deliberations. Creation of new object is controlled by a probability distribution. The distribution is used to make the determination if an object is created and if so which one. There is one additional constraint on object creation: the randomly selected location in the environment must be vacant. This is to prevent the environment from becoming crowded. If the location is occupied, then no object is created this turn. At the time that an object is created, its expiration time is also set. To create the expiration time, each object type has a specific duration value, given in number of pulses. This number and a randomly generated number representing fluctuation are added to the current pulse for the expiration time. The random fluctuation provides some uncertainty in the object's behavior. This provides the framework for future experiments where the agents will be able to take the object's duration into account when calculating its course of action. However, those calculations would be done at the cognitive level of affect, which is beyond

the scope of this paper.

The problem solving agents perform no actions during their own pulse functions. Rather, they respond to a message event initiated by the worldAgent during its pulse function. This was to ensure that the worldAgent had finished updating the world representation before the problem solving agents began working with it.

5. Experiments

The experiment consisted of studying the number of the randomly generated goal objects and penalty objects each agent collected in the specified time, which for these trials was set to 1000 pulses. The grid world size was the as we've been using to explain the system, namely 100x100, with 10000 possible locations. For our dynamic ADP agent, the state space was determined by the number of reward objects, and their relative distances to the agent. The non-reward states of the environment had no reward, thereby creating a neutral environment for the agents. The environment was neither negative, which would create a condition where agents would try to escape, it or positive, which would create a condition where the agents would not want to leave it. As mentioned previously, the trials were time driven rather than goal driven, so making the environment either positive or negative would not have served much purpose since the agents would not have been able to escape the environment or extend their stay in it.

[Jason, we need describe the learning process of the active agent, and how long it takes]

Twenty-five trials were run for each agent. The results of the trials are listed in Figure 1. Each agent was run on its own in the environment for the trial. Since there was no reward for any interactions between agents, there would be no force between agents and nothing to be gained by running them at the same time. Both agents would essentially be unaware of the presence of the other. Having them in the same environment, would not add to our understanding of primary emotions.

Table 1 shows the results of the active agent and the affective agent runs. The affective agent collected an average of 6.35 goals and 0 penalty objects, earning an average score of 635 points per trial. The learning agent collected an average of 6.56 goals and 0 penalty objects, earning an average score of 656 points per trial. The affective agent collected, on average, 4.11% of the goal objects generated by the environment, and the active agent collected 4.0% of the goal objects. Neither agent collected a penalty object in any of the trials.

Table 1. Comparison of Affective Agent and Learning Agent

Agent Performance

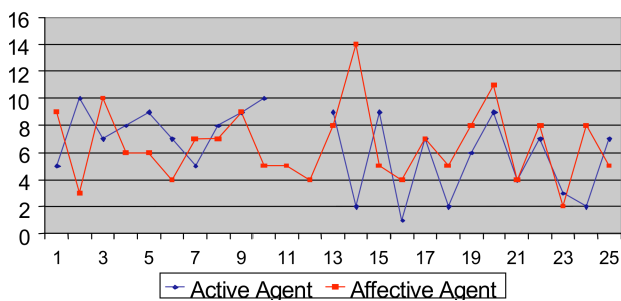


Figure 4. Results of Individual Trials

	average	std dev	min	max
Active	6.56	2.74	2	14
Learning	6.35	2.82	1	10
goals created	158.48	9.49	133	181
penalties created	147.02	12.22	114	186

Table 2. Comparison of Affective Agent and Virtual Greedy Agent

	average	std dev	min	max
Affect	3.50	0.75	2	6
Virtual	2.00	1.15	0	4
goals created	69.40	8.24	54	83
penalties created	0.00	0.00	0	0

The low percentage of goals collected is due to a combination of factors: the large area over which the goals are distributed, the time it takes to traverse the area, and the duration of the reward objects in the environment. To really understand how the agents performs, we create another virtual greedy agent as a base line of comparison.

[Jason, please describe the virtual agent, how it works, and discuss table 2 results.]

Both agents performed equally well in collecting rewards. This result was not unexpected. Considering that the dynamic reinforcement agent was built from what we learned from creating the affective agent, we suspected that they would perform equally well.

6. Conclusion and Future Work

We set out to study if the action selection problem of a dynamic grid world environment could be improved upon if examined from the perspective of the affective computing paradigm. We created one affective agent capable of succeeding in a large, dynamic environment. Then we developed an active reinforcement agent based on the same principles as the affective agent, the abstracted state is created based on the distance to the objects. This modified learning agent also succeed in the large, dynamic environment. The affective agent did perform slightly better than the active agent. More importantly, the affective agent did not have any memory problems. The memory requirements of the affective agent are light compared to the learning mechanism of the active agent.

Active agent's memory problem could, no doubt, be solved by taking care to manage the memory. However, this brings up another point where the affective agent has an advantage over the active agent – it was easier to design and implement. We can use lines of code to measure the work required to implement the agents. When one considers that the lines where all written in java, based on the same framework, and written by the same programmer, it serves as a good measure of comparison. The active agent was written in 1180 lines of code, while the affective agent was written in 830 lines. This does not count the supporting files for the JAF framework, just the class file for each agent's control process. Writing the active agent increased the lines of code by 42%.

The results of our study demonstrated a piece of the value that the new paradigm of affective computing has. There is naturally considerably more that can be offered. There are further areas in this domain that we would like to explore. The biggest area that we would like to explore is the area of cognitive emotions. We focused just on primary emotions in this study to provide a solid foundation of affect. Now we will turn our attention to how this

cognitive emotions could be added to the framework we've built. One item that we observed from the agents' behavior is that if there are penalty objects between the agent and goal objects, both agents will move away from the penalty objects even though it takes them away from the goal objects also. Neither agent is looking far enough ahead to see if there is a path that will lead them around the penalties to the goals. They do not do any planning. We will study the effects adding cognitive emotion has on this situation.

In a future study we would also like to create other objects that have different influences on the agents. Both in type and in intensity. We would like to define a fuel object that the agent will need to have the energy to move in the environment. This would introduce an affective drive to satisfy hunger motivation. This type of motivation is a homeostasis drive – a drive to maintain specified conditions. The interactions between goal driven motivation and homeostatic motivations would be worth investigating in this framework.

7. REFERENCES

- [1] Hemenover, Scott and Zhang, Shen. (2004). "Anger, personality, and optimistic stress appraisals." *Cognition and Emotion*, 18, 363-382.
- [2] Bodenhausen, G.V., Sheppard, L.A., and Kramer, G. P. (1994). "Negative affect and social judgement: the differential impact of anger and sadness." *European Journal of Social Psychology*, 24, 45-62.
- [3] Smith, Craig A., Haynes, Kelly N., Lazarus, Richard S, Pope, Lois K. (1993) "In Search of the 'Hot' Cognitions: attributions, Appraisals, and Their Relation to Emotion." *Journal of Personality and Social Psychology*, 65, 916-929.
- [4] Schwarz, Norbert. (2000) "Emotion, cognition, and decision making." *Cognition and Emotion*, 14, 433-440.
- [5] Pickard, Rosalind W. *Affective Computing*. Cambridge, MA: MIT Press, 1997.
- [6] Corbett, A.T, Anderson, J.R., and O'Brien, A.T, (1995) "Student modeling in the ACT Programming Tutor." *Cognitively Diagnostic Assessment*, 19-41.
- [7] Heylen, D., Nijholt, A., Akker op den, R., and Vissers, M., (2003). *Socially intelligent tutor agents.* IVA 2003, 341-347.
- [8] Breazeal, C., (2001). "Affective Interaction between Humans and Robots." In J. Keleman & P. Sosik (Eds) *ECAL 2001*, LNAI 2159, 582-591.
- [9] Henniger, A. E., Jones, R. M., and Chown, E., (2003). "Behaviors that emerge from emotion and cognition: implementation and evaluation of a symbolic-connectionist architecture." *AAMAS 2003*, 321-328.
- [10] Gratch, J., and Marsella, S., (2004) "Evaluating the modeling and use of emotion in virtual humans." *Autonomous Agents and Multiagent Systems*, 2004, AAMAS 2004, 320-327.
- [11] Coddington, A., and Luck, M., (2003). "Towards motivation-based plan evaluation." In I. Russell, & S. Haller, S., (Eds.). *Proceedings Of the Sixteenth International FLAIRS Conference*, 298-302.

[12] Scheutz, M. (2002). "Agents with or without Emotions?" In

Proceedings FLAIRS 02, 89-94