Emotion Oriented Programming: Computational Abstractions for AI Problem Solving

Kevin Darty, Nicolas Sabouret

Emotion Oriented Programming: Computational Abstractions for AI Problem Solving

Kevin Darty and **Nicolas Sabouret**
Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie, Paris, France

Abstract

In this paper, we present a programming paradigm for *AI* problem solving based on computational concepts drawn from Affective Computing. It is believed that emotions participate in human adaptability and reactivity, in behaviour selection and in complex and dynamic environments. We propose to define a mechanism inspired from this observation for general *AI* problem solving. To this purpose, we synthesize emotions as programming abstractions that represent the perception of the environment's state *w.r.t.* predefined heuristics such as goal distance, action capability, *etc*. We first describe the general architecture of this "emotion-oriented" programming model. We define the vocabulary that allows programmers to describe the problem to be solved (*i.e.* the environment), and the action selection function based on emotion abstractions (*i.e.* the agent's behaviours). We then present the runtime algorithm that builds emotions out of the environment, stores them in the agent's memory, and selects behaviours accordingly. We present the implementation of a classical labyrinth problem solver in this model. We show that the solutions obtained by this easy-to-implement emotion-oriented program are of good quality while having a reduced computational cost.

Introduction

Decision making in complex and dynamic environments gives rise to space-state combinatorial explosion that makes classical *AI* methods difficult to use, whereas human beings perform rather well in such environments (Minsky 2006). (Gigerenzer and Brighton 2009) suggests that one reason why this phenomenon is the use of heuristics by human beings and several authors showed that emotions play a crucial role in action selection for human and animals (Darwin, Ekman, and Prodger 2002; Baumeister et al. 2007) and act as a heuristic in decision making (Antos and Pfeffer 2011). In this context, recent work in affective computing, such as (Lisetti and Gmytrasiewicz 2002), (Scheutz 2002) and (Antos and Pfeffer 2011), demonstrates that emotions and classical *AI* decision making should not be opposed to each other, but can be used in a complementary manner to overcome limitations of classical *AI* approaches.

Emotions play several roles in human behaviour: some of them allow us to focus our attention on specific elements, some alter learning capabilities and some support the communication of information about our internal states (Minsky 2006; De Melo et al. 2011). In this paper, we are interested in one particular function of emotions that has not been much considered in the literature: emotions also allow people to interpret observations in a way that is consistent with their internal state (Lazarus 1991).

Whereas in classical planning (Hoffmann 2001) and in machine learning (Sutton and Barto 1998), precise goal information is required for the problem solver to find a solution, human beings decide based on abstract and incomplete representations of their current state and goal. In this context, emotions can be seen as abstractions synthesizing a representation of the world at a definite moment and how it has been perceived by the agent in relation to its goals. Our claim is that we could design simple solvers whose behaviour is driven by such "emotions". Not only would these solver ease the programmer's task (since it works with abstractions instead of complete state descriptions) but it could improve the agents performance by reducing the space state.

In this article, we propose a programming paradigm for *AI* problem solving based on abstract concepts called emotions that ease both the problem definition and solving algorithm. In the next section, we present related work in affective computing and decision making. Section presents our emotion-oriented programming model. Section presents our evaluation on a dynamic labyrinth problem, which proved that this method led to results similar to humans in quality, while limiting the programmer's task.

Related Work

Emotions play an active role in people's behaviour. (Lazarus 1991) showed that people try to keep the situation under control by regulating their emotions. In this view, emotions occur as a result of environment stimuli and several appraisal models are proposed to capture this "cognitive evaluations" of the situation (Scherer, Wallbott, and Summerfield 1986; Roseman 1996; Ortony, Clore, and Collins 1990). Although the psychological validity of these models has been criticized (Barrett 2006), the idea that emotions capture the situation in a comprehensive way seems interesting when it comes to problem solving in non-trivial environments, *i.e.* dynamic and partially observable environments with a "reasonably large" space-state. This is why we propose to de-

velop *AI* problem solvers that use emotions as first-order entities.

There are two main approaches to emotion representation in the literature: emotion categories (Ortony, Clore, and Collins 1990; Batra and Holbrook 1990), which consists in enumerating all kind of emotions; and dimensions (Mehrabian 1996), which considers that all emotions can be represented as positions in an N-dimension space. We will not discuss here the validity of each model nor the number of necessary categories or dimensions: (Gebhard 1996) showed that dimensions are an efficient approach to parameter an agent's activity, whereas categories made narrative and scenario construction easy, and that it was possible to relate one to the other. When it comes to the question of programming *AI* problem solving, this suggests that we should use emotion categories to make life easier for programmers who want to design decision rules, whereas the agent's internal should rely on a dimensional model of emotions to enhance decision making.

(Loewenstein and Lerner 2003) wrote a comprehensive study on the role of emotions in decision making. Some researchers tried to reproduce the exact functioning of the human decision making (Anderson et al. 2004). More recently, (de Melo, Zheng, and Gratch 2009) and (Antos and Pfeffer 2011) proved that emotion could help in decision tasks. However, in most approaches, emotions are used to alter the decision making process by giving additional information to the agent. (Antos and Pfeffer 2011) suggests a different path: emotions could be used directly as a heuristic for goal selection. Our approach directly follows their tracks but in a task-oriented programming scheme (whereas Antos and Pfeffer use goal-oriented agents). Emotions in our model are abstractions that control the behaviour selection.

One key element in this approach is that emotions in agents cannot be assimilated to human emotions. According to (Scheutz 2002), agents emotional models in *AI* should stay away from anthropomorphizing labels: *while a robot can be described as being in a state of "disappointment", which leads to a state of "sadness", these states will bear no resemblance to human states labeled the same if they are, for example, merely implemented as states of a finite automaton*. In other words, our model has to remain easily implementable for programmer and does not need to perfectly fit with facts. As a consequence, the programmer in our paradigm should not have to consider affective dimensions, but rather representations of the agent's internal state associated with stimuli, that will allow to specify the action's rules. The agent internal state must be managed by the solver itself. To this purpose, we follow (Loewenstein and Lerner 2003)'s proposition that mood plays a key role in regulating the perception-decision balance (*i.e.* the repartition of cognitive activity between perception and decision-making processes), and we implement an automated mechanism for balancing between decision epochs and perceptions. This mechanism will thus serve as the basis for the emotion-based action selection.

The last element that requires specific attention in our programming paradigm is the management of memory. (Loewenstein and Lerner 2003) showed that emotions play a role in the memorization of events. According to Miller, the human being cannot focus on more than a few information (Miller 1956). As a consequence, (James, Burkhardt, and Skrupskelis 1981) propose to distinguish between the primary memory (or short-term memory) to focus on important information and a secondary memory containing the whole knowledge. We adopted this model in our approach: mood (and thus perception-decision balance) will be computed based on a limited number of recent elements, whereas all collected information about the problem will be stored in a long-term memory, thus allowing the programmer to design more complex behaviour.

Emotion Oriented Programming

Our proposal is to draw the line to a new programming paradigm for *AI* problem solving that considers emotions as the core elements. Emotions in this work must not be understood as human-like emotions, but as abstract concepts playing two roles: 1) they synthesize information about the solving process (distance to the goal, current progress, *etc*) and 2) they allow programmers to write simpler rules, based on the emotional state of the agent and partial elements of its environment. We call this paradigm Emotion Oriented Programming (*EOP*) and we claim that it leads to interesting results in non-trivial environments.

In this section, we first present the general architecture of the model, and we separate what is part of the problem definition, what is processed automatically by the *EOP* interpreter and what is devoted to the solution's programmer. We give a complete definition of the vocabulary for both the environment and solver agent definition.

General Architecture

The *EOP* model architecture has three parts:

- the problem definition part which allows the **problem** programmer to specify the problem as an environment, characterized by observable variables (*e.g.* the agent is located on (x, y)) and operations on these variables (*e.g.* move one step forward).
- the behaviour definition part which allows the **solution** programmer to design behaviours used by the emotion-oriented solver. Behaviours can be understood as series of operations guarded by emotions.
- the *EOP* interpreter which automatically computes emotions from the environment, balances the decision and observation processes, and selects a behaviour based on the programmed *solution*.

On the figure, these three parts are represented with red dashed blocks for tasks devoted to the problem programmer, green dotted blocks for those devoted to the solution programmer and blue circled blocks for the *EOP* interpreter.

At each step of the execution, the environment builds an evaluation of the problem state which is used by the solver to generate an *emotional state*. It also provides information about its variables as perceptions. Perceptive filters can be used to block some of them, whereas all others automatically generate what we call *emotions*. Emotions describe

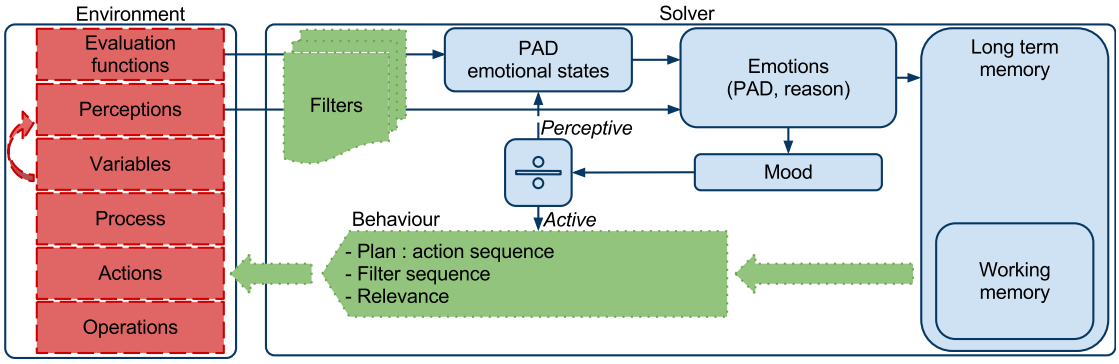

Figure 1: General architecture

why a given emotional state was generated at a given time period. They are stored in the memory and the most recent ones (*i.e.* those stored in the working memory) are used to actualize a *PAD*-based *mood* values (*i.e.* the agent's internal state). This mood will be used to balance between action and perception cycles. Moreover, during a perception cycle, the reaction behaviour is selected depending only on the working memory.

Emotions

The *EOP* model uses a dimensional representation of emotional state for both stored *emotions* and internal *mood*. An emotional state $\overrightarrow{PAD} = (P, A, D) \in [-1, 1]^3$ contains the three Mehrabian dimensions (Mehrabian 1996): *Pleasure*, *Arousal* and *Dominance* which are interpreted as follows:

- the pleasure expresses the proximity of a local goal. An important pleasure means that the solver is getting closer to the solution. An aversion denotes that the solver is moving away from it goal.
- the arousal represents the novelty and is measured with differences between the current and the previous problem state. Thus, a neutral value correspond to a traversal in which states are similar. When sudden changes take place, the arousal increase.[1]
- the dominance indicates if the situation is under control (*i.e.* the ability for the solver to act on the environment). A strong dominance means that the solver has a wider range or possible actions to perform.

This model is used at two levels. First, the agent's internal mood, whose role is to balance between perception and decision, is directly a $\overrightarrow{PAD}$ vector. Second, each perception from the environment is turned into an *emotion*. In our model, an emotion is a triple $e = (\overrightarrow{PAD_e}, R_e, t_e)$ where R_e is the *reason* of the emotion, *i.e.* the environment's elements that triggered the affective dimensions (for instance, I was feared by a monster) and t_e is the date of the emotion.

[1]One could argue that arousal only occurs when state change is connected to the agent's goal. This notion of "connection to the goal" is however difficult to capture in a generic context. Filters and relevance functions, presented later, can be used to control this notion.

The computation of $\overrightarrow{PAD_e}$ is presented in section "Problem Solving".

Problem description

The problem definition part is used by the problem programmer to describe the environment. An environment E is defined as a 7-tuple $E = \langle V, P, O, A, N, F, Proc \rangle$ with:

- V is a set of private variables v with domains D_v;
- $P : v \rightarrow D_v$ is perception function that associate values to variables (only a subset of V is visible);
- O is a set of operations $o = \langle Prec, Eff \rangle$ with preconditions and effects on V;
- A is a set of possible actions for the solver: $\forall a \in A, a = \langle o, c \rangle$ with c the action cost associated to operation o;
- N is the total number of action points that the solver can use at each execution step;
- F is a set of evaluation functions for the solver;
- $Proc$ is the environment's internal process, *i.e.* a series of operations applied at each execution step.

Four evaluation functions are mandatory in our model for the computation of the system's emotional states (*i.e.* the *PAD* value) in our *EOP*-solver:

- $Solved : V \rightarrow \mathbb{B}$ which returns *true* only if the problem is in a final state;
- $Solvable : V \rightarrow \mathbb{B}$ which returns *true* if the problem still has a solution;
- $Goal(s) : V \rightarrow \mathbb{R}^+$ which returns an evaluation of the distance to the goal;
- $Distance : d(s, s') \rightarrow \mathbb{R}^+$, $d(s, s') = 0 \Leftrightarrow s = s'$ which returns an evaluation of the distance between two different states.

Emotion-oriented behaviour description

Once the problem has been defined (previous section), an *EOP* solution consists in a set of behaviours. The *EOP* solver (presented in the next section) selects behaviours based on its mood and received emotions.

A behaviour $b = \langle p_b, f_b, r_b \rangle$ is characterized by:

- Its plan, *i.e.* a linear sequence of actions $p_b = (a_1, a_2, ..., a_n)$ to perform while this behaviour is active. Keep in mind that actions are defined with precondition (for example, moves need to check for wall) and effect (*e.g.* changing position). If a precondition fails, the behaviour is stopped and re-examined at the next decision epoch (see next section).
- A set of filters applicable to each plan step: $\forall a_i, f_i \in V$ is the subset of variables that the agent wants to ignore: no emotion will be computed based on the perceptions $P(v), v \in f_i$. As a consequence, filtered perceptions no longer affect the mood nor future decisions.
- A relevance function $r_b : m \times E_{WM} \rightarrow \mathbb{R}^+$ where $m = \overrightarrow{PAD}$ is the current mood and E_{WM} is the set of emotions in the working memory.

As will be explained in the next section, behaviours are selected based on the relevance function r_b, *i.e.* on the emotions in the working memory and the current *mood* of the solver $\overrightarrow{PAD}$. This allows the programmer to better control the behaviour selection. For instance, a flee behaviour can be triggered as soon as the user is in a fear mood, or only if a real danger is perceived in the environment: all this is encompassed in r_{flee}.

Problem solving

The *EOP* problem solver works in cycles of 3 steps. First, the environment performs the next actions in the currently selected behaviour (in the limit of the maximum allowed action points). If no behaviour has been selected or if the solver is in a "perception" epoch (depending on its mood), a new behaviour is selected. Second, the environment runs one step (see $Proc$ in "Problem description"). Third, new perceptions are received (based on the current state and filters), corresponding emotions are generated and the mood is updated accordingly. This subsection presents the behaviour selection step and the mood update.

Behaviour selection Whenever new behaviour must be selected, the solver compares the r_b values of all behaviours b such that $r_b > 0$. These values are normalized and we use a random selection based on the order of r_b.

Mood update The new mood $\overrightarrow{m'}$ is computed from the previous mood $\overrightarrow{m}$, the initial mood $\overrightarrow{m_0}$ and the working memory WM:

$$\overrightarrow{m'} = \gamma.\left(\epsilon\left(\overrightarrow{\sigma} \times \frac{\sum_{e \in WM} \overrightarrow{PAD_e}}{|WM|} - \overrightarrow{m}\right) + \overrightarrow{m}\right) + \overrightarrow{m_0}$$

where $\gamma \in [0, 1]$ is the convergence factor toward the initial mood, $\epsilon \in [0, 1]$ is influence factor of new perceptions on the mood and $\overrightarrow{\sigma} \in [0, 1]^3$ is the sensibility on each dimension.

Once the mood value has been updated, it is used to balance between perception and action epochs:

$$bal = \frac{m_D + m_P + 2}{4} \times \frac{m_A + 1}{2}$$

Whenever $bal \geq 0$, the solver is set in an action epoch: it will perform the next actions of the currently selected behaviour. On the contrary, when $bal < 0$, it enters a perception epoch and will select a new behaviour.

Perception and emotions When new perception occur (*i.e.* at the end of each step), the system computes emotion values as follows:

- Pleasure is based on the distance to the goal: $P = f(goal(v') - goal(v))$, where $f(x) = 1 - exp^{-x}$ if $x \geq 0$ and $exp^x - 1$ otherwise.
- Arousal is based on the distance with previous state: $A = 1 - 2exp^{-d(v,v'))}$.
- Dominance is based on the number of possible actions: $D = 2\left(\frac{|a \in \mathcal{A}.a_{Prec} = \top)|}{|\mathcal{A}|}\right) - 1$.

These emotions will, in turn, lead to mood update and eventually to new behaviour selection.

Experimentation

Implementation

Our model was implemented as a *Java 1.5 API*. This *API* allows programmers to implement the problem part as an *Environment* subclass and the behaviour definition part as a *EOPSolution* one. The problem definition requires to implement the evaluation functions (*goal*, *distance*, *solved*, *etc*) and the *process* method that applies the action sequence, updates the problem state and sends agent perceptions (all these elements being problem-dependant).

The *EOP* model itself, with perception, mood and action selection, is integrated and the programmer needs only to fill its *EOPSolution* with instances of *Behaviour* and *Filter*. Behaviours define the action plans and its precondition in terms of emotions. Filters are sets of perceptions, emotions and relevance functions which define in which emotional state perceptions will be filtered.

Example

To test our *EOP* model, we implemented a *Wumpus* problem, using the Java *API*. The labyrinth contains treasures, traps (that must be avoided) and monsters (that chase the agent). The goal is to reach the exit with a maximum amount of treasure, while avoiding traps and monsters. The agent perceives only the 4 surrounding positions when they contain walls, traps or treasures. It also receives information when a monster is less than 4-cells away (monster smell). The agent does not know the position of the exit, but it knows the distance to this exit (through the *distance* function). It can "see" the exit at a distance of 3.

The agent has five possible actions: moving one step up, down, left or right, and picking up a treasure when the current position contains one. This problem makes an interesting testbed for the *EOP* model because it is both dynamic (monsters move) and partially observable (positions of monsters, traps, treasures and exit is unknown). Moreover, using action points, we can also add a time constraint for decision and action. In our implementation, out of 50 action points per turn, each move costs 10 and taking a treasure costs 40.

Figure 2: The implemented Wumpus problem. Exit is represented by a flag, start by a coin, walls by bricks, treasures by rubies, traps by policeman's helmets, monsters by yellow squares, agent location by a blue square.

We developed several solvers for this labyrinth, including a simple random-based action selection model and a naive depth-first search algorithm, but we focus the discussion on two solvers:

- a modified *A** that has access to the full labyrinth configuration (which removes the partially-observable aspect but gives an idea of what could be an "optimal" solution on the labyrinth),
- a simple *EOP*-based solution which relies on 4 affective behaviours:
  - *Discover* which selects the less visited patch among the 4 surrounding ones. It does not apply any filter to the agent's perceptions. Its relevance is set to 10 (*i.e.* low *w.r.t.* other behaviours) when it is in a neutral emotional state. When there is a monster or a trap nearby, its relevance does not fall down completely to 0. As a consequence, it has a (low) probability to be selected even in other situations (*i.e.* mood and working memory).
  - *FollowEnd* draws a direct path to the exit with moving actions and filters all perceptions so as to focus on the exit. Its relevance is very high (100) when the arousal increases and the exit patch is visible. It cannot be selected when the exit position is unknown.
  - *GoToTreasure* builds a plan to move toward a treasure and pick it up. It filters other possible treasure perceptions so as to focus on this one. Its relevance is 20 if pleasure and arousal increase due to a surrounding patch containing a treasure. It cannot be selected when no treasure has been detected.
  - *RunAway* which moves 4 patches back to avoid a monster. It does not apply any filter. Its relevance is 50 when pleasure and dominance are negative (due to a monster perception). It cannot be selected in the absence of monster nearby.

Note that since our model clearly separate the problem definition to its resolution, the behaviour definition part on the *EOP* model was an effortless task. Using emotions as programming abstractions makes the solver definition quite easy. The real difficulty is in the problem definition itself (defining action rules and evaluation functions).

Experimental setting

The studied class problem is not solvable with random or classic naive algorithm (our depth-first search implementation hardly obtains 5% success when monsters are in the labyrinth). Problem specific strategies give better results. In particular, good results are obtained by the *A** algorithm with full information (however, this implementation does not fulfill the partially-observable hypothesis and lacks reactivity when monsters are in the path to the exit).

In a slightly different configuration, Antos and Pfeffer showed that *PDM* learning can lead to good results (Antos and Pfeffer 2011), especially when the stochastic impact is reduced (which is the case when the number of monsters is limited). However, for this kind of problems, human beings tend to be much more efficient. For this reason, we propose to compare our *EOP* model's results with those obtained by human beings.

We implemented a graphical user interface (*GUI*) that allows users to select the agent sequence of actions and to observe perceptions in a visual representation of the labyrinth. Unknown patches are replaced by a question mark and the labyrinth boundaries are not visible to the user. Six people participated to this evaluation. Each subject was first introduced the system's functioning through a test labyrinth, after which it had to solve 6 different labyrinths with increased order of difficulty.

Results

We compared the overall score (computed after the number of collected treasures defined here as subgoals, the total number of actions, the tics count and whether the exit (*i.e.* the goal) was reached or not) between the human users and our *EOP* algorithms (run six times) on the same labyrinth configuration. We also compared the human and solver action sequences (respectively seq_h and seq_s) using the Levenshtein distance (LD) normalized by the length of the longer solution:

$$1 - \frac{LD(seq_h, seq_s)}{max(|seq_h|, |seq_s|)}$$

Human and solver perception times are not comparable and are not considered during our tests, this is why we remove perception tics from the sequence.

When the exit is reached, scores obtained by human subjects ($avg = 989$) are slightly better than those obtained by the *EOP* solver($avg = 867$). *A** performs better ($avg = 1012$), obviously because it has access to the entire labyrinth structure. These values tend to confirm that human users adopt behaviours that lead almost to the maximum score (which was one of our working hypothesis) and that the *EOP* solver is not so bad on these non-trivial problems.

More interestingly, two out of six configurations were complete failures for both human subjects and our *EOP* solver: whenever the human user cannot solve the problem, neither does our *EOP* model. In these situations, the "survival time" of the solver is three times higher than those of the human subjects or the *A** solver: our solution selects the *RunAway* behaviour quite often.

As far as action sequences are concerned, we first observe that there is a low similarity between different human subjects on the same labyrinth (42% similarity in average, with a peak to 58% for one labyrinth and a standard deviation of 0.079). This reflects that they adopted different solutions on the same labyrinth. Compared to those human behaviours, our solver has a 42% similarity (with a standard deviation of 0.063), which tends to confirm that it uses some sort of similar actions, although this would require a closer look to determine which aspects are similar (*e.g.* maximum and average similar sequence length).

Conclusion and future work

This paper presents a new approach to *AI* problem solving in non-trivial environments, based on programming with abstract concepts, called emotions. These emotions capture information on the solving process, based on information given by the solver, and serve as the basic element for the action rules definition.

We implemented a prototype version of this model in Java and tested it against different environments in a dynamic labyrinth problem with partial information. We compared our results with classical *AI* approaches and with human behaviours. We showed that our model obtains slightly better results than human testers and that the path to the solution shows similarities with the behaviour adopted by humans. Moreover, the implementation of the *EOP* solution, compared to other possible approaches (*e.g.* Q-Learning), is quite straightforward: the programmer gives action rules based on emotions and perceptions. It could be interesting to compare the EOP performance to *POMDP* approaches.

The work is only a first step toward a real programming paradigm and many perspectives can be foreseen. With the growing interest in affective computing and its connection to decision making, new operators for emotional concepts can be proposed, *i.e.* different abstractions of the situation and the progress to the goal. Those operators should be integrated into an Emotion Oriented Platform, so that programmers could compare and combine them together.

Acknowledgments

Thanks to Cindy Mason for giving us the permission to exploit the original idea of emotion oriented programming.

References

Anderson, J.; Bothell, D.; Byrne, M.; Douglass, S.; Lebiere, C.; and Qin, Y. 2004. An integrated theory of the mind. *Psychological review* 111(4):1036.

Antos, D., and Pfeffer, A. 2011. Using emotions to enhance decision-making. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Barrett, L. F. 2006. Solving the emotion paradox: Categorization and the experience of emotion. *Personality and social psychology review* 10(1):20.

Batra, R., and Holbrook, M. 1990. Developing a typology of affective responses to advertising. *Psychology and Marketing* 7(1):11–25.

Baumeister, R. F.; Vohs, K. D.; DeWall, C. N.; and Zhang, L. 2007. How emotion shapes behavior: Feedback, anticipation, and reflection, rather than direct causation. *Personality and Social Psychology review* 11:167–203.

Darwin, C.; Ekman, P.; and Prodger, P. 2002. *The expression of the emotions in man and animals*. Oxford University Press, USA.

De Melo, C.; Gratch, J.; Antos, D.; and Carnevale, P. 2011. A Computer Model of the Interpersonal effects of Emotion Displayed in Social Dilemmas. In *Proc. Affective Computing and Intelligent Interaction*.

de Melo, C.; Zheng, L.; and Gratch, J. 2009. Expression of moral emotions in cooperating agents. In *Intelligent Virtual Agents*, 301–307. Springer.

Gebhard, P. 1996. Alma: A layered model of affect. *International Conference On Autonomous Agent* 29–39.

Gigerenzer, G., and Brighton, H. 2009. Homo heuristicus: Why biased minds make better inferences. *Topics in Cognitive Science* 1(1):107–143.

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI magazine* 22(3):57.

James, W.; Burkhardt, F.; and Skrupskelis, I. 1981. *The principles of psychology*, volume 1. Harvard Univ Pr.

Lazarus, R. 1991. Emotion and adaptation. In *Handbook of personality: Theory and Research*. Oxford University Press. 609–637.

Lisetti, C., and Gmytrasiewicz, P. 2002. Can a rational agent afford to be affectless? a formal approach. *Applied Artificial Intelligence* 16(7-8):577–609.

Loewenstein, G., and Lerner, J. 2003. The role of affect in decision making. *Handbook of affective science* 619:642.

Mehrabian, A. 1996. Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology* 14(4):261–292.

Miller, G. 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review* 63(2):81.

Minsky, M. 2006. *The emotion machine: commensense thinking, artificial intelligence, and the future of the human mind*. Simon and Schuster.

Ortony, A.; Clore, G.; and Collins, A. 1990. *The cognitive structure of emotions*. Cambridge Univ Pr.

Roseman, I. 1996. Appraisal determinants of emotions: Constructing a more accurate and comprehensive theory. *Cognition & Emotion* 10(3):241–278.

Scherer, K.; Wallbott, H.; and Summerfield, A. 1986. *Experiencing emotion: A cross-cultural study*. Cambridge University Press; Paris: Maison des Sciences de l'Homme.

Scheutz, M. 2002. Agents with or without emotions. In *Proceedings FLAIRS*, volume 2, 89–94.

Sutton, R., and Barto, A. 1998. *Reinforcement learning: An introduction*. The MIT press.