The College of Wooster Libraries
Open Works

Senior Independent Study Theses

2015

Learning Emotions: A Software Engine for Simulating Realistic Emotion in Artificial Agents

Douglas Code
The College of Wooster, dcode15@wooster.edu

Follow this and additional works at: https://openworks.wooster.edu/independentstudy

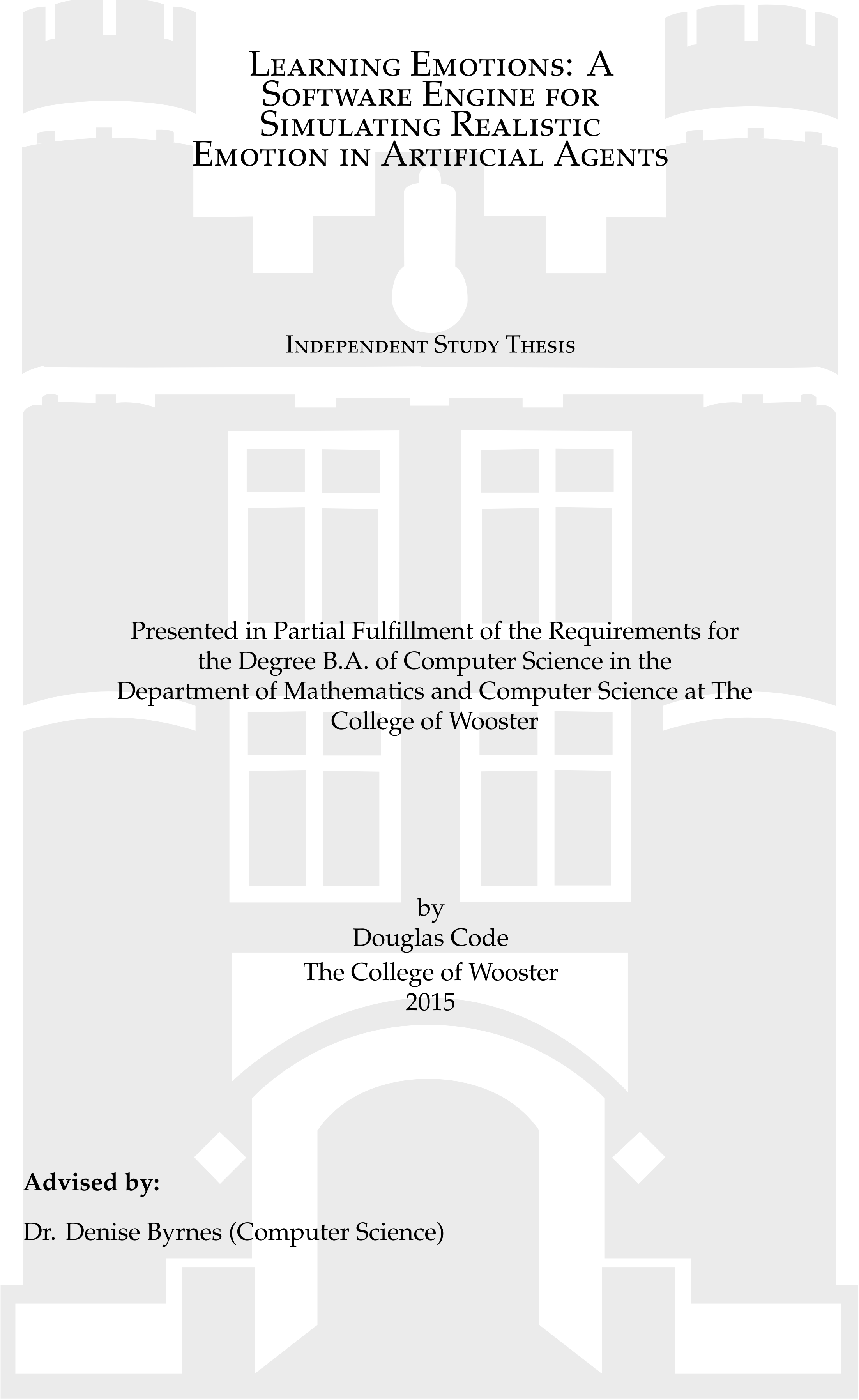
Part of the Artificial Intelligence and Robotics Commons, Computational Neuroscience Commons, and the Theory and Algorithms Commons

Recommended Citation

Code, Douglas, "Learning Emotions: A Software Engine for Simulating Realistic Emotion in Artificial Agents" (2015). *Senior Independent Study Theses.* Paper 6543.
https://openworks.wooster.edu/independentstudy/6543

This Senior Independent Study Thesis Exemplar is brought to you by Open Works, a service of The College of Wooster Libraries. It has been accepted for inclusion in Senior Independent Study Theses by an authorized administrator of Open Works. For more information, please contact openworks@wooster.edu.

© Copyright 2015 Douglas Code



Learning Emotions: A Software Engine for Simulating Realistic Emotion in Artificial Agents

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree B.A. of Computer Science in the
Department of Mathematics and Computer Science at The
College of Wooster

by
Douglas Code
The College of Wooster
2015

Advised by:

Dr. Denise Byrnes (Computer Science)

THE COLLEGE OF
WOOSTER

© 2015 by Douglas Code

ABSTRACT

This paper outlines a software framework for the simulation of dynamic emotions in simulated agents. This framework acts as a domain-independent, black-box solution for giving actors in games or simulations realistic emotional reactions to events. The emotion management engine provided by the framework uses a modified Fuzzy Logic Adaptive Model of Emotions (FLAME) model, which lets it manage both appraisal of events in relation to an individual's emotional state, and learning mechanisms through which an individual's emotional responses to a particular event or object can change over time. In addition to the FLAME model, the engine draws on the design of the GAMYGDALA emotional engine for games. Evaluations of the model's behavior over a set of test cases are performed, with a discussion of the model's efficacy in different situations.

Acknowledgments

I would like to thank Dr. Byrnes for her help and support through not only the Independent Study process, but also through my entire time as a student at the College of Wooster. I would also like to thank Timothy Duhon for acting as a sounding board for many of my ideas, and Rachel Schwartz for her encouragement and unwavering support.

VITA

Fields of Study Major field: Computer Science

Minor field: Mathematics

Minor field: English

Specialization: Artificial Intelligence

CONTENTS

Abstract iii

Acknowledgments iv

Vita v

Contents vi

List of Figures viii

List of Tables x

List of Listings xi

CHAPTER PAGE

1 Introduction 1
- 1.1 Simulating Emotion 2
- 1.2 Research Goals 3

2 Theory 5
- 2.1 Fuzzy Logic 5
- 2.2 Q-Learning 11

3 Related Models 16
- 3.1 OCC Model 16
- 3.2 EMA Model 19
- 3.3 WASABI Architecture 21
- 3.4 TABASCO Architecture 23
- 3.5 GAMYGDALA 25

4 FLAME Model 32
- 4.1 Model Overview 32
- 4.2 Event Appraisal Models 34
- 4.3 FLAME Design 35
- 4.4 Desirability 38
- 4.5 Appraising Events 39
- 4.6 Filtering Emotions 41
- 4.7 Determining Behavior 42
- 4.8 Learning 43
  - 4.8.1 Classical Conditioning 44

4.8.2 Event Impacts . . . 45
4.8.3 Expectations . . . 48
4.8.4 Social Values . . . 49

5 Model Design and Implementation 51
5.1 Purpose . . . 51
5.2 Software Overview . . . 52
5.2.1 Fuzzy Logic . . . 52
5.2.2 Emotional Engine . . . 55
5.3 Visualization . . . 58

6 Testing and Results 60
6.1 Future Work . . . 64
6.2 Conclusion . . . 65

APPENDIX PAGE

A Software Manual 66

References 76

List of Figures

Figure Page

2.1 Structure of a fuzzy logic system (Zeynep 2006). 7
2.2 Fuzzy sets representing cold, cool, warm, and hot temperatures . . . 7
2.3 Warm AND Cool ($A \cap B$). 8
2.4 Warm OR Cool ($A \cup B$). 9
2.5 Fuzzy output sets clipped according to their membership values. . . 10
2.6 Three rooms that an agent can move between along connecting lines. 11
2.7 Set of rooms with reward in room 5. 12

3.1 The OCC Model [9]. 18
3.2 The EMA Model [8]. 21
3.3 The TABASCO Architecture [14]. 24

4.1 The emotional component of the FLAME model [5]. 36

5.1 High-level overview of FLAME model. [5] 51
5.2 Fuzzy system architecture. 52
5.3 Centroid estimation using a representative triangle [6]. 54
5.4 Full engine architecture. 55
5.5 Model visualization and user interface. 59

A.1 Visualization control menu. 66
A.2 Agent addition controls. 67
A.3 Agent goal controls. 67
A.4 Event addition controls. 68
A.5 Event triggering controls. 69
A.6 Motivational state controls. 69
A.7 Agent standards controls. 70
A.8 Object introduction controls. 71
A.9 Decay controls. 72
A.10 Information display controls. 72
A.11 "Emotions" information display. 73
A.12 "Goals" information display 73
A.13 "Events" information display 74
A.14 "Motivations" information display 74

A.15 "Associations" information display 75

List of Tables

Table Page

2.1 Initial Q-Values. 13
2.2 Updated Q-Values. 14

3.1 Primary emotions and their PAD values [2]. 22
3.2 Goals in the GAMYGDALA Engine [11]. 26
3.3 Goals in the GAMYGDALA Engine [11]. 27

4.1 Emotional states produced through appraisal of desirability and expectation. 35
4.2 Fundamental emotions and the situations from which they arise (El-Nasr et al., 2000). 39
4.3 Equations for intensity of expectation and desirability-dependent emotions. (El-Nasr et al., 2000). 40

LIST OF LISTINGS

Listing Page

3.1 Code to incorporate GAMYGDALA engine into modeling a blacksmith in a role-playing game [11] 30

6.1 Test of appraisal system and social standards. 60
6.2 Test of motivational states and classical conditioning. 61
6.3 Test of learning mechanism. 62

CHAPTER 1

INTRODUCTION

In "Affective Computing," [10] Rosalind Picard writes,

> Most adults know that too much emotion can wreak havoc on reasoning, but less known is the recent evidence that too little emotion can also wreak havoc on reasoning. Years of studies on patients with frontal-lobe disorders indicate that impaired ability to feel yields impaired ability to make decisions; in other words, there is no "pure reason." Emotions are vital for us to function as rational decision-making human beings.

Picard's insights into the importance of emotion in computing spawned Affective Computing, a field of Computer Science that has grown quickly with improvements in hardware and research into intelligent algorithms. Machines that can interpret and express emotions have applications from robotics, where behaviors can be chosen based on the perceived emotions of nearby humans, to tutoring software, where algorithms to detect emotions in a student's voice or face can adjust lessons to respond to frustration or confusion.

1.1 Simulating Emotion

While a large portion of the research performed on affective computing focuses on reading and interpreting human emotions through various sensors, another component of the field is devoted to realistically giving computers and simulated agents the capacity to act emotionally. With voice recognition software systems like Google's Google Now, Apple's Siri, and Microsoft's Cortana, many people interact with software in an increasingly human, conversational way. As these technologies grow more complex and powerful, the depth of these interactions only increases. A key part of creating computational systems that are easy to engage and interact with on a social level is the simulation of emotion. As Picard points out, not only are emotions a meaningful component of the human condition, they are essential for decision making in situations where all information relevant to the decision cannot possibly be known.

Two areas where human interactions are frequently modeled are the social sciences and gaming. Economic and sociological models frequently rely on attempts to predict the behavior of humans, a difficult or impossible task without accounting for emotional responses to events. Many software suites exist for creating agent-based models (Repast, MASS), and the incorporation of frameworks that enable the simulation of emotion in artificial agents would provide powerful tools for creating models and simulations with greater depth and realism.

In a different domain, video games frequently incorporate large numbers of non-player characters (NPCs). Games increasingly focus on open-world gameplay, where players are free to move around large worlds as they please. These games typically seek to create immersive environments that feel real and living to the player. A key aspect of this realism is programming the game's NPCs to behave realistically. However, these behaviors are often scripted ahead of the time, relying on simple event triggers to initiate a corresponding behavior. By simulating emotion in NPCs rather than relying on scripted events, game creators can create more dynamic human ecosystems where emotion acts as an intermediary between the event and the behavior. In this way, an event triggers emotion, and behavior arises from that emotion. This allows for more immersive and engaging in-game interactions, as NPCs can respond dynamically to the events around them and the actions and emotions of surrounding NPCs.

1.2 Research Goals

The primary goal of this research is to design and implement a software framework for modeling emotion in artificial agents. A survey of computational models of emotion is provided, looking at the Ortony, Clore and Collins (OCC) model [9] that was the basis for many of the compuational models of emotion that followed, as well as the TABASCO architecture [14], the WASABI model [2], the EMA model for appraisal [8], and the FLAME model [5]. Particular attention is given to the FLAME model and its use of fuzzy sets to represent emotion, and fuzzy rules to map emotions to behaviors. The FLAME model acts as the basis for the developed software framework, as it extends existing systems with mechanisms for learning and classical conditioning.

An existing system, GAMYGDALA [11], is also examined. GAMYGDALA is an emotional engine for games, providing a software framework that is genre-independent for incorporating realistic emotions into NPCs. The engine was designed based on research in psychology and neuroscience, with ease of use, generalizability, and resource-efficiency in mind as key heuristics for success.

This research and the accompanying software use GAMYGDALA as a key starting point for emotion engine design, and attempts to extend the work of Popescu et al to incorporate elements of the computational models above. In particular, this research proposes a modification of the GAMYGDALA engine that makes use of the fuzzy logic, classical conditioning, and learning mechanisms proposed by Nasr et al. in their FLAME model.

CHAPTER 2

Theoretical Components

2.1 Fuzzy Logic and Set Theory

In classical logic and set theory, any given proposition falls into one of two truth values: true or false. Systems like this, where there are exactly two truth values, are called bivalent [3]. In a bivalent system, an object can only either belong to a set or not belong to a set. For example, the number 3 belongs to the set of real numbers, and as such, the statement "3 is a real number" evaluates to true. While classical logic has broad applications in mathematics, it fails to appropriately represent systems where propositions can be partially true. Such situations frequently occur in evaluations of objects and their states, as well as in the semantics of human language [7]. The statement, "The shirt is red," does not necessarily evaluate directly to a result of true or false, as shirts can be "reddish" or "very red."

To accommodate this partial truth, fuzzy logic can be used. Where classical logic could only evaluate to true or not true in regards to membership in a set, fuzzy logic allows evaluation to values representing differing intermediate states of truth. These values typically fall somewhere on the interval [0,1], with fuzzy sets occupying different spaces within this interval. In this case, the shirt may have a truth value of 0.9, indicating that the shirt is very red, while a shirt whose truth value is 0.65 would also be considered red, but less-so than the shirt whose truth value was 0.9. This "truth-value" can be interpreted as the degree to which the object in the proposition belongs to the proposed set [3].

Formally, the distinction between classical and fuzzy sets can be shown through their functional mappings. A classical set is the function:

$$A : U \rightarrow 0, 1 \tag{2.1}$$

for A(x), where x is some element of the universal set, U, and evaluates to either 0 or 1. In contrast, a fuzzy set is the function:

$$B : U \rightarrow [0, 1] \tag{2.2}$$

for B(x), where x, some element of the universal set, U, evaluates to some value in the range 0 to 1, inclusive. Here B(x) acts as the truth value, representing the degree of x's membership within set B. The fundamental difference between the two functions is that A(x) can only take the values 0 and 1, while B(x) can take any continuous value within that range [3].

Figure 2.1 shows the structure of a fuzzy logic system. Inputs are crisp values that are then fuzzified according to defined fuzzy sets. These fuzzified inputs are then passed through an inference engine that uses AND and OR rules to determine fuzzy values for each rule. These rules are then combined to create a fuzzy output set that maps a fuzzy to each defined output of the system. Finally, these fuzzy outputs are defuzzified to get a single crisp value for each output.

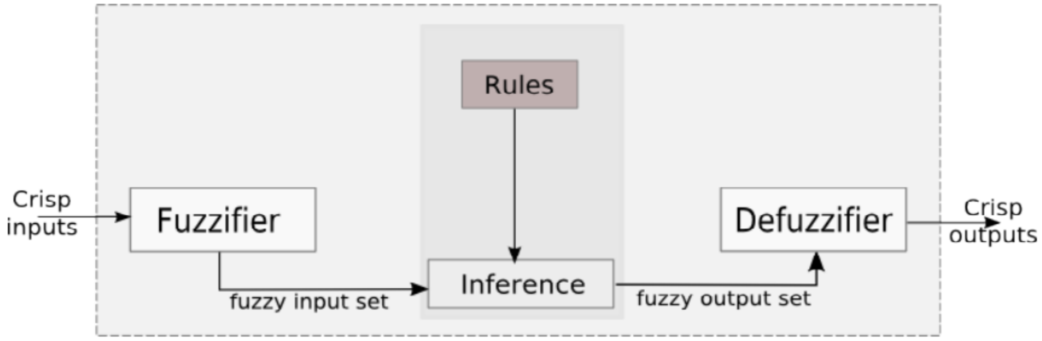

Rules
Crisp inputs
Fuzzifier
Inference
Defuzzifier
Crisp outputs
fuzzy input set
fuzzy output set

Figure 2.1: Structure of a fuzzy logic system (Zeynep 2006).

Figure 2.2 gives a visual representation of the membership functions for fuzzy temperature classification sets defined for temperature inputs on the range $[0, 100]$. As shown by the graph, input values can belong to multiple fuzzy sets at once. For example, the temperature input of 50 degrees belongs to both the "Cool" and "Warm" fuzzy sets with a membership value of 0.5 for each.

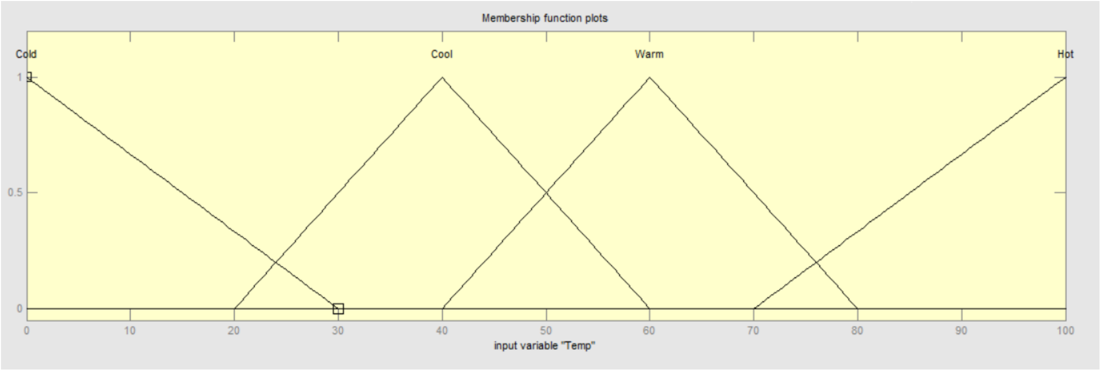


Figure 2.2: Fuzzy sets representing cold, cool, warm, and hot temperatures

After membership functions over the input are used to convert a crisp input value to one or more fuzzy values, the fuzzy input set is run through a set of rules to map the inputs to an output. Fuzzy rules use AND and OR operators on fuzzy sets in the following form (assuming there are three output sets: low, medium, and high):

```
If Temp is Cold OR Hot, then Action is High
If Temp is Cool AND Occupied is Low then Action is Low
If Temp is Cool AND Occupied is High then Action is Medium
If Temp is Warm AND Occupied is Low then Action is Low
If Temp is Warm AND Occupied is High then Action is Medium
```

Here a second input variable, "Occupied", is introduced. The inference engine lets any number of input variables be mapped to any number of output variables. In this case, the two input variables, "Temp" and "Occupied", are mapped to a single output variable, "Action". Lofti Zadeh, the creator of fuzzy logic [16], defines AND (intersection) and OR (union) as follows:

$$C = A \cap B \tag{2.3}$$

$$f_c(x) = min[f_a(x), f_b(x)] \tag{2.4}$$

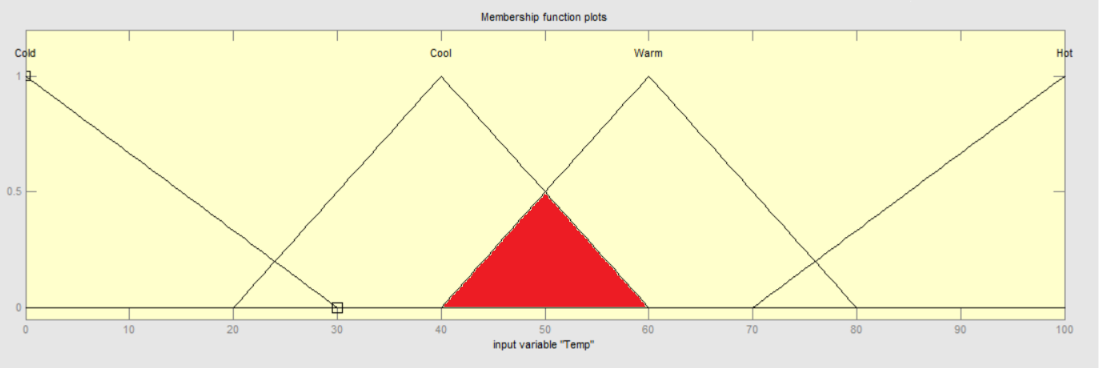


Figure 2.3: Warm AND Cool ($A \cap B$).

$$D = A \cup B \tag{2.5}$$

$$f_d(x) = max[f_a(x), f_b(x)] \tag{2.6}$$

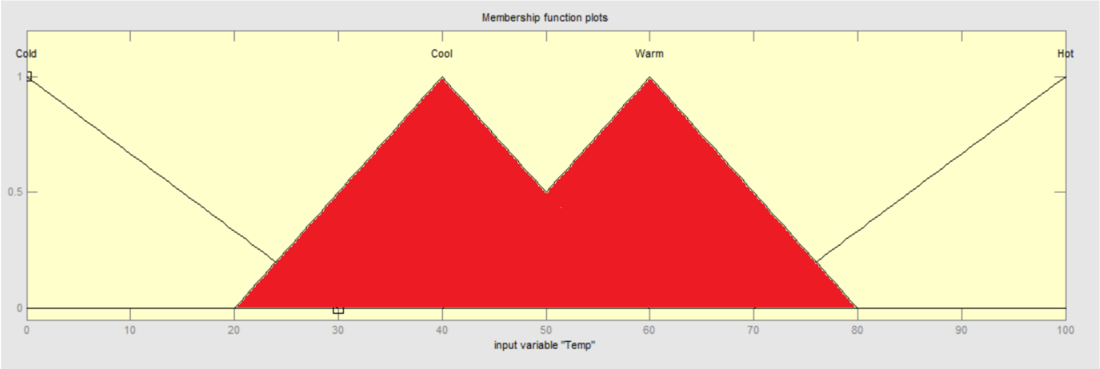


Figure 2.4: Warm OR Cool ($A \cup B$).

With these definitions, a fuzzy value for each rule can be calculated. If we assume that the crisp temperature input had a membership in the "Warm" set of 0.5 and the crisp "Occupied" input had a membership in the "Low" set of 0.3, the fuzzy value for the rule

`If Temp is Warm AND Occupied is Low then Action is Low`

can be calculated by taking the maximum (AND) of these two values (0.5). This maps a "truth" value of 0.5 to the "Low" output set.

Returning to the rules listed above, there are multiple rules mapping to a single output example. The resulting values for these rules must be combined into a single fuzzy value for each output so that there are not multiple membership values for a single ouput set. There are a number of methods for doing this involving summations and normalization. For this work, the combined value is the maximum value of all rules mapping to the output in question. For example, if the calculated value for the rule:

`If Temp is Warm AND Occupied is Low then Action is Low`

is 0.5, and the calculated value for the rule:

`If Temp is Cool AND Occupied is Low then Action is Low` is 0.3, the final fuzzy output membership mapped to the "Low" output set is 0.5.

Once the fuzzy input sets are mapped to fuzzy output sets by the inference engines, the membership values for each fuzzy output set must be combined into a single, crisp output value for the output variable. This is achieved through defuzzification. In this paper, centroid defuzzificcation is used. Under centroid defuzzification, membership sets are "clipped" at the height corresponding to the fuzzy output value for that set. Figure 2.5 shows the result of this process in red for a mapping that resulted in a value of 0.7 for the "Medium" output set and 0.3 for the "Low" output set.

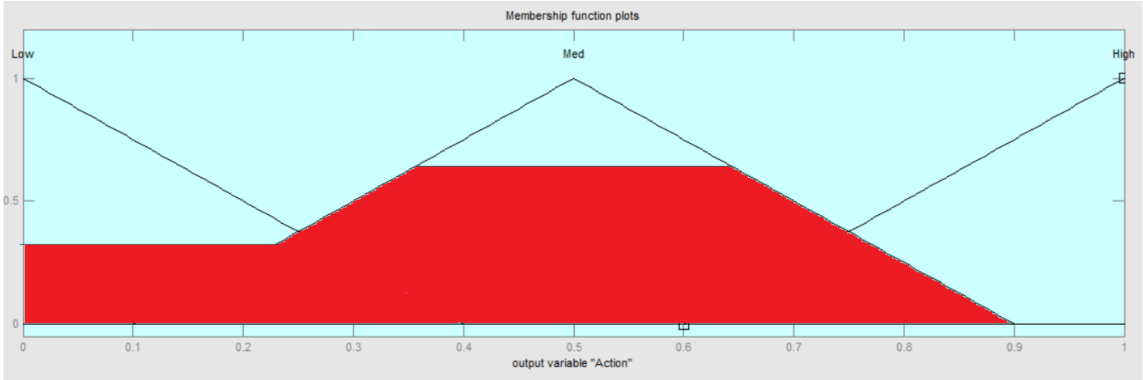


Figure 2.5: Fuzzy output sets clipped according to their membership values.

The center of mass for the determined area is then calculated using the function [5]):

$$y_{final} = \frac{\int \mu_{combined}(y) y dy}{\int \mu_{combined}(y) dy} \tag{2.7}$$

where $\mu_{combined}(y)$ is the new, combined membership function (outlining the red region in Figure 2.5). This calculation defuzzifies the fuzzy output sets, resulting in a final, crisp value for the output variable.

2.2 Q-LEARNING

Q-learning is a machine learning technique for learning about the rewards of actions or sequences of actions. In particular, it lets an agent learn about systems where the reward for an action may be delayed from the execution of the action. This handles situations where actions can improve the agent's likelihood of receiving a reward after subsequent actions. For example, Figure 2.6 shows three connected rooms, numbered 1-3.

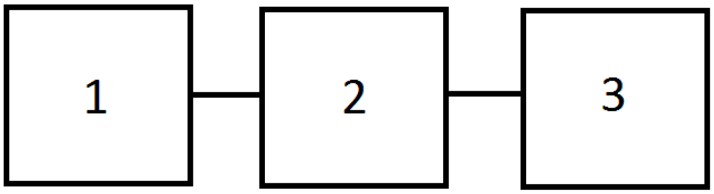


Figure 2.6: Three rooms that an agent can move between along connecting lines.

If an agent is in room 1, and will be rewarded for entering room 3, it is advantageous for the agent to move to room 2, as this will allow it access to room 3. However, there is no immediate reward for entering room 2, presenting the problem of creating a positive association between the agent and the movement from room one to two. Q-learning uses a state-action matrix to associate rewards with the steps that led up to them, with the association decaying as the steps become further removed from the reward.

Q-learning relies on a recursive maximization function oriented towards maximizing the Q-value of the next action taken, where the Q-value is a measure of the expected reward for the action given the individual's current state. The maximization function is as follows [15]:

$$Q(s,a) = r(s,a) + \gamma max_{a'}(Q(s',a')) \tag{2.8}$$

Where $Q(s, a)$ is the Q-value for a state-action pair, $r(s, a)$ is the immediate reward for a state-action pair, and $Q(s', a')$ is the Q-value of a state-action pair following the current action. γ is a decay constant, $0 \leq \gamma \leq 1$, that determines how much the rewards of subsequent actions determine the Q-value of a current action. For example, if $\gamma = 0.1$, the Q-value of moving into room 2 from room 1 will be less influenced by the reward in room 3 than if $\gamma = 0.8$.

The fundamental algorithm behind Q-learning is:

- Choose an action
- Receive a reward (or no reward) for the action
- Determine the Q-value of new state
- Update previous state using equation 2.3

To illustrate how Q-learning works, a short example is provided. The example operates on the set of rooms shown in Figure 2.7.

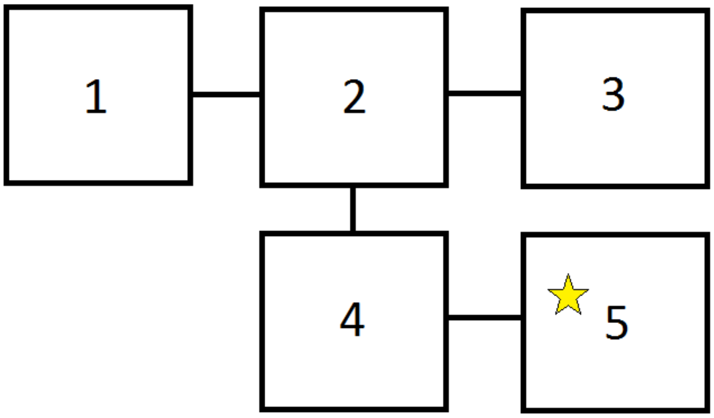


Figure 2.7: Set of rooms with reward in room 5.

Each room can be considered to be a state, and the movement between rooms can be considered an action. For example the state-action pair for being in room 1 and moving to room two would be (1, move2). For the example, the reward value of entering room 5 is 100 and the γ value is 0.5.

The algorithm starts with the individual having an empty matrix of Q values, as they have no previous examples from which to learn. Table 2.1 shows the initial matrix.

	move1	**move2**	**move3**	**move4**	**move5**
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Table 2.1: Initial Q-Values.

The individual's starting position is randomly assigned as room 2. From here the individual has three possible actions: move1, move3, and move4. Say the individual chooses to move to room 4. The Q-value for the state-action pair (2, move4) can now be updated using equation 2.3:

$$Q(s,a) = r(s,a) + \gamma max_{a'}(Q(s',a')) \tag{2.9}$$

$$Q(2, move4) = r(2, move4) + 0.5 * max(Q(4, move2), Q(4, move5)) \tag{2.10}$$

$$Q(2, move4) = 0 + 0.5 * max(0,0) = 0 \tag{2.11}$$

Since the Q-value of (2, move4) is already 0, nothing needs to be updated. The updated state of the individual is now 4, giving the actions move2 and move5 as options. If we choose move5, the Q-value for (4, move5) is calculated as follows:

$$Q(4, move5) = r(4, move5) + 0.5 * max(Q(5, move4)) \tag{2.12}$$

$$Q(4, move5) = 100 + 0.5 * max(0) = 100 \tag{2.13}$$

The matrix can now be updated with the new value for Q(4, move5), as seen in Table 2.2.

	move1	**move2**	**move3**	**move4**	**move5**
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	100
5	0	0	0	0	0

Table 2.2: Updated Q-Values.

Now that the goal has been reached, the algorithm can be started again with the updated $Q(s, a)$ matrix. Let us once again place the individual in room 2 as the starting room. Our individual, again, chooses to move to room 4. Using equation 2.3, $Q(2, move4)$ is calculated.

$$Q(2, move4) = r(2, move4) + 0.5 * max(Q(4, move2), Q(4, move5)) \tag{2.14}$$

$$Q(2, move4) = 0 + 0.5 * max(0, 100) = 50 \tag{2.15}$$

Thus, Q(2, move4) can now be updated to 50 in the matrix. Here, the individual has learned that there is some value in moving from room 2 to 4, as this sets up a subsequent action that is rewarded, even though the move from 2 to 4 itself has no reward.

Q-learning's domain-independence ability to learn about sequences of events make it useful for modeling emotion, where emotional behaviors are often influenced by an understanding of the results of chains of events following an action. Given enough iterations, the individual's Q-values converge to those optimal for determining actions to take to achieve its goals in a deterministic setting. One issue presented by the algorithm in the context of emotional modeling is that models of environments and interacting actors and users are non-deterministic, so the guarantee of convergence no longer holds [5]. Ways of addressing this are discussed later in section 4.8.2.

CHAPTER 3

RELATED MODELS

This section covers three models of emotion: the Ortony, Clore, Collins (OCC) Model, the WASABI Architecture, and the TABASCO Architecture. The discussion of these models is intended to give insight into the state and breadth of models of emotion, as well as illustrating the fundamental ideas behind the FLAME Model that was used in this project. Following the discussion of these three models, the GAMYGDALA Emotional Engine is described. GAMYGDALA is an emotional modeling engine built for use in game development, and serves as an important inspiration for design decisions when working on implementing a computational model of emotion.

3.1 THE OCC MODEL

The Ortony, Clore, Collins (OCC) cognitive model of emotions [9] forms the basis for many of the dominant cognitive and computational models of emotion. In particular, the GAMYGDALA engine (section 3.4) and the FLAME Model (Chapter 4) are based on the ideas put forth by Ortony et al.

Many predecessors to the OCC Model argued for a basic set of emotions that form all other emotions through combination. OCC differed from these models by suggesting that there is no clear basic set of emotions. Instead, the OCC Model describes emotions as responses to the surrounding world that can be differentiated using variables [1]. Ortony et al. suggest that positive/negative is the most basic spectrum by which emotions can be differentiated, with other variables like expectedness and agent responsibility allowing more complex differentiation. From this approach arises the set of appraisal-based models of emotion. Appraisal-based models describe emotions as the results of "appraisals" of one's environment using a set of variables. For example, a simple emotion differentiation may use only two variables: desirability and expectation. In this model, the emotion *disappointment* would be classified as the result of an expected, desirable event failing.

The OCC Model identifies three fundamental triggers for emotional changes: events, agents, and objects. Responses to these triggers are correspondingly determined by goals (events), agents (standards), and attitudes (objects)(Arbib, 1990).

Goals are divided into three subsets:

- Active Pursuit Goals (What an agent wants to achieve)
- Interest Goals (What an agent wants to happen, but does not personally feel capable of strongly influencing)
- Replenishment goals (Goals that do not go away when they are achieved, ex: Eating may achieve a goal of not starving, but it does not make the goal disappear)

Goals are directly tied to events, as an event's desirability is calculated by how an event affects an agent's goals, and to what extent.

Standards describe the social context within which the agent lives. These include concepts like social norms and conventions, along with social expectations regarding the role served by the agent. For example, a policeman may have different social expectations relating to their job than a teacher. Standards are directly tied to reactions regarding other agents, as these interactions are capable of producing emotions that rely on norms (shame, guilt, pride)[1].

Attitudes govern how an agent feels about objects independent of the agent's goals. This reflects the agent's natural disposition towards objects in the surrounding world. For example, an agent may dislike a particular style of furniture, even though it has no justification for its dislike of the style, and regardless of whether the style has any influence on one of the agent's goals. Attitudes also allow for emotional mechanisms like classical conditioning, a component of the FLAME Model discussed in section 4.8.1.

Figure 3.1 shows the overall structure of the OCC model.

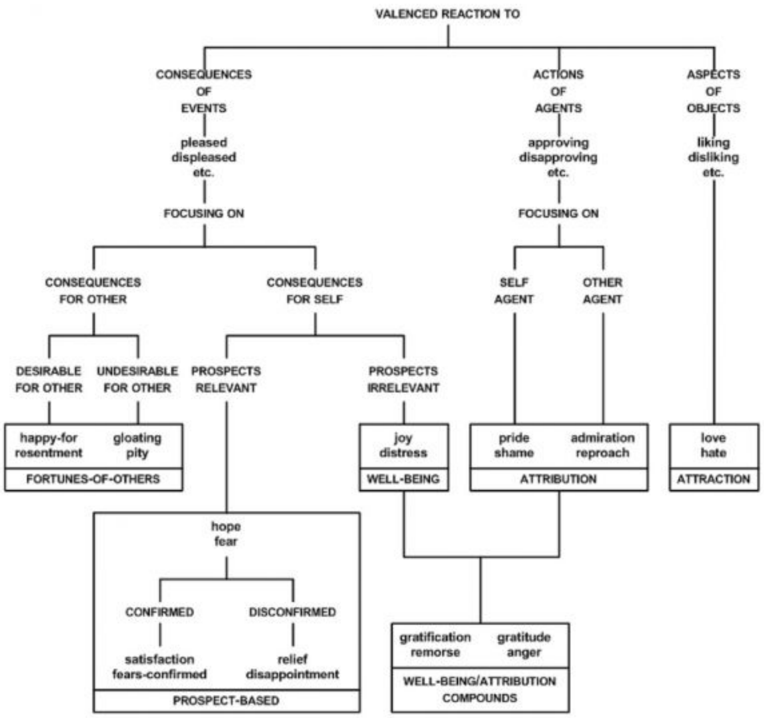

VALENCED REACTION TO
CONSEQUENCES OF EVENTS
ACTIONS OF AGENTS
ASPECTS OF OBJECTS
pleased displeased etc.
approving disapproving etc.
liking disliking etc.
FOCUSING ON
FOCUSING ON
CONSEQUENCES FOR OTHER
CONSEQUENCES FOR SELF
SELF AGENT
OTHER AGENT
DESIRABLE FOR OTHER
UNDESIRABLE FOR OTHER
PROSPECTS RELEVANT
PROSPECTS IRRELEVANT
happy-for resentment
gloating pity
FORTUNES-OF-OTHERS
joy distress
WELL-BEING
pride shame
admiration reproach
ATTRIBUTION
love hate
ATTRACTION
hope fear
CONFIRMED
DISCONFIRMED
satisfaction fears-confirmed
relief disappointment
PROSPECT-BASED
gratification remorse
gratitude anger
WELL-BEING/ATTRIBUTION COMPOUNDS

Figure 3.1: The OCC Model [9].

This shows how the emotions resulting from the OCC Model are highly context-dependent. The fundamental distinction of events, agents, and objects first determines the emotional options available, with further distinctions regarding context allowing for greater variety and complexity of the appraised emotions. For example, a reaction to an event with undesirable consequences for another agent can result in gloating or pity, whereas a desirable event in the same situation can result in either happiness for the other agent or resentment, based on the values of variables used in the appraisal process.

3.2 EMA Model

Like the OCC Model, The EMA (EMotion and Adaptation) Model, proposed by Marsella and Gratch [8] is rooted in appraisal theory. A number of prominent emotional appraisal models from the early 2000s suggest that appraisal occurs in two processes: fast, memory-based associations and inferences, and longer-term, conscious reasoning. EMA proposes a less complicated single-process model for appraisal, with appraisal itself occuring quickly and automatically, and the representation of the world itself evolving over time [8].

Marsella and Gratch identify a set of core components they deem essential to any appraisal model:

- **Relevance, valence, and intensity**: In order to assess events and their emotional responses, an appraisal model must be able to account for the valence (positive or negative effect) and intensity of results of actions or events.
- **Future implications**: To manage emotions that are dependent on future events, appraisal models must provide mechanisms for reasoning about how likely events are to occur and what the possible consequences of those events may be.

- **Blame and responsibility**: The cause of an event plays a crucial role in appraisal, as events caused by one actor may warrant a drastically different emotional response than events caused by another actor. As a result, appraisal models must let the modeled individual assign blame or credit for external events and actions.

- **Power and coping potential**: An individual's feeling of power over a situation significantly affects appraised emotions. For example, similar events may elicit fear in an individual who feels powerless and anger in an individual who feels like they have power over the results of those events. This means that appraisal models must account for an individual's feeling of power over both the external world, and their own internal state.

- **Coping strategies**: Motivational states, like hunger and pain, can suppress or enhance emotions and influence an individual's internal representation of the surrounding world. Methods for beliefs, desires, intentions, and plans to change according to coping strategies let appraisal models account for this.

Figure 3.2 shows the overall structure of EMA. The *Agent-Environment Relationship* represents the memory of the agent, constituted by the agent's desires, plans, beliefs, and perceived probabilities. Probabilities correspond to beliefs, representing the certainty with which the individual holds the belief. Inferences about the world coming from both events and changes to the Agent-Environment Relationship can be mapped back into the Agent-Environment with appropriate modifications to the agent's memory.

Emotions that have been run through coping mechanisms can be mapped to actions, with actions having two primary components: preconditions and effects. Preconditions are the requirements that must be fulfilled before an action can be carried out. For example, prior to entering a building, an agent will need to approach the doors of the building. Effects are the expected state changes occurring after the execution of an action.

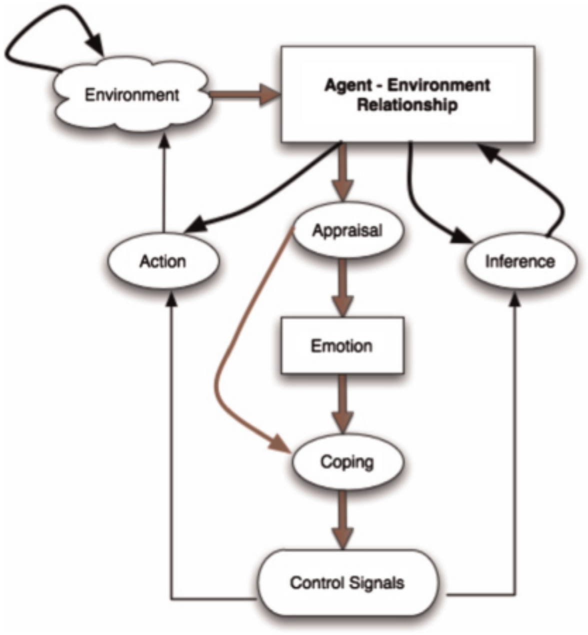

Environment
Agent - Environment Relationship
Action
Appraisal
Inference
Emotion
Coping
Control Signals

Figure 3.2: The EMA Model [8].

3.3 WASABI Architecture

The WASABI Architecture [2] is designed for accurately modeling emotional changes and their resulting facial expressions, incorporating relevant research from motion psychology, neurobiology, and developmental psychology. The architecture is appraisal-based and relies on the PAD (Pleasure, Arousal, Dominance) Emotional State Model to represent internal emotional states. The PAD model describes emotional states as points in three dimensional space with the axes being pleasure (how pleasant or unpleasant an emotion is), arousal (the emotion's intensity), and dominance (whether the emotion is dominant or submissive). For example, the emotion *annoyed* has a PAD value of (-50, 0, 100) in the WASABI Architecture, as it is unpleasant (pleasure value of -50), of neutral intensity (arousal value of 0), and dominant (dominance value of 100).

WASABI separates emotions into *primary* and *secondary categories*. Primary emotions are defined as base emotional states that are intrinsic to being human. They are triggered reflexively and quickly, with little conscious consideration necessary. In contrast, secondary emotions are triggered consciously, arising from evaluating events against goals and past experiences. The separation of primary and secondary emotions makes WASABI capable of dealing with simulating emotion in different ages, as children tend to express primary emotions with little inhibition [2]. Table 3.1 shows the primary emotions in the WASABI Architecture and their PAD values. The secondary emotions for the model are taken from the "prospect-based" emotions outlined in the OCC model. Out of the six emotions outlined by Ortony et al., hope, fears-confirmed, and relief are modeled as secondary emotions in WASABI.

Emotion	**PAD Value**
Angry	$(80, 80, 100)$
Annoyed	$(-50, 0, 100)$
Bored	$(0, -80, 100)$
Concentrated	$(0, 0, \pm 100)$
Depressed	$(0, -80, -100)$
Fearful	$(-80, 80, 100)$
Happy	$(80, 80, \pm 100)$
Sad	$(-50, 0, -100)$
Surprised	$(10, 80, \pm 100)$

Table 3.1: Primary emotions and their PAD values [2].

WASABI determines which emotions to express through an awareness likelihood function. The likelihood that an individual will become aware of an emotion and express it is a function of the distance between the emotion's PAD value (see Table 3.1) and the current emotional state of the individual. For example, given that the

PAD value for *angry* is (80, 80, 100), if an individual's PAD value is (70, 80, 100), their likelihood of becoming aware of and expressing anger will be greater than if their PAD value was (30, 80, 100).

Secondary emotions are triggered by the cognitive processes of the agent and decay over time, always returning to a default intensity of zero unless further events occur that retrigger a rise in the intensity of the secondary emotion [2]. In contrast, primary emotions do not decay in intensity over time, as their likelihood of expression is determined by the current PAD value's distance from the defined PAD value for the emotion.

Through the differentiation of primary and secondary emotional states, WASABI creates "high" and "low" paths of emotion elicitation. The "low" path triggers non-conscious appraisal and elicitation of primary emotions for situations where emotions are intrinsically felt. For example, entering a peaceful place may trigger pleasant emotions without conscious appraisal. The "high" path deals with conscious appraisal, evaluating an event against the individual's goals to determine an appropriate emotional response [2].

3.4 TABASCO ARCHITECTURE

The TABASCO Architecture [14] is an appraisal-based architecture rooted in reactive planning. Reactive, or dynamic, planning is an action-planning paradigm that focuses on real-time responses to events in the environment, allowing dynamic planning in regards to external changes. Reactive planning systems perform an evaluation for a particular moment in time, and decide on exactly one plan to respond to the current external state.

TABASCO works with two systems: appraisal-based emotion generation, and action planning meant to cope with the results of the generated emotions. Figure 3.3 gives a broad overview of the architecture's structure.

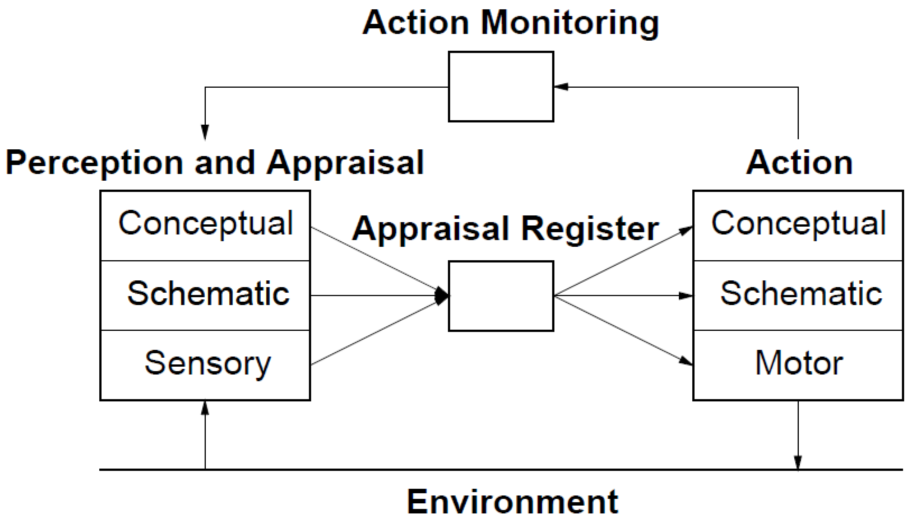

Action Monitoring
Perception and Appraisal
Action
Conceptual
Schematic
Sensory
Appraisal Register
Conceptual
Schematic
Motor
Environment

Figure 3.3: The TABASCO Architecture [14].

The Perception and Appraisal component of the model is divided into three subcomponents: sensory, schematic, and conceptual. The sensory component deals with immediate sensations and their pleasantness or unpleasantness. The schematic component compares inputs against social and self-imposed schema for behavior. The conceptual component works on the level of learned knowledge and beliefs about the world, and how these beliefs affect the individual's desires and intentions.

The sensory, schematic, and conceptual levels of the Perception and Appraisal mechanism are mirrored in the motor, schematic, and conceptual levels of the Action mechanism. In the Action mechanism, the motor level triggers physical expressions of emotion (e.g. smiling), while the schematic and conceptual levels determine long-term planning and coping methods for the individual based on the output of the Appraisal Register [14].

The Appraisal Register acts as an intermediary between the Perception and Appraisal component and the Action component, taking information from all three levels of the Perception and Appraisal component and using that information to trigger different levels of the Action component. For example, the Appraisal Register may first trigger a motor response to the emotional information it received from the Perception and Appraisal component, then trigger the conceptual level to generate a long-term planned response based on the conceptual information the Register received [14].

The Action Monitoring component feeds information from the Action component back into Perception and Appraisal. This lets the appraisal mechanism know what actions have been attempted and respond accordingly, allowing for more comprehensive appraisals. With the incorporation of Action Monitoring, the model accomodates emotions that are triggered based on previous efforts to solve a problem (like frustration and hopelessness).

3.5 GAMYGDALA Emotional Engine

GAMYGDALA [11] differs from OCC, TABASCO, and WASABI in that it is an implementation of a software engine for emotion rather than a model. The stated goal of GAMYGDALA is to make video games more interesting, entertaining, and immersive by modeling emotion in non-player controlled (NPC) characters. GAMYGDALA is a black-box engine rooted in the methods of appraisal described by the OCC Model. It is intended to be similar in use to physics engines, in that developers can incorporate the engine into their software and realistically simulate emotion without needing a deep understanding of the mechanics of emotion. The engine's emotional output can then be used by the developer to change the look of the character's model or animation in response to their emotional state, or to determine the behavior of the character based on their emotions. In their pursuit of an easy-to-use black-box emotional engine, Popescu et al. identify key heuristics for the creation of an effective engine [11]:

- Psychological Grounding - The engine should be rooted in valid and tested psychological research
- Modularity - The engine should be modular and easily incorporated into larger pieces of software
- Efficiency - The engine should be able to operate quickly on a large enough number of agents to be usable in its intended domain
- Ease of Use - Developers should be able to use the engine effectively in their software without extensive knowledge of theories of emotion or appraisal theory

The GAMYGDALA engine operates primarily on *goals* and *beliefs*. Goals are defined as what an individual character wants to achieve or maintain. All goals are annotated with a name, owner, and utility. Utility can take the value of -1 to 1 inclusive, with -1 indicating that the character does not want the goal to happen to the greatest degree possible, and 1 indicating that the character wants the goal to happen to the greatest degree possible. Table 3.2 shows example goals provided by Popescu et al. and how they would be represented within the engine.

Owner	**Name**	**Utility**
Knight	Kill Monster	1
Knight	Self Die	-1
Knight	Princess Die	-1
Knight	Find Gold	0.5

Table 3.2: Goals in the GAMYGDALA Engine [11].

In addition to goals, beliefs are an essential part of GAMYGDALA's functions. Beliefs are events that are represented with the essential information that GAMYGDALA needs to process them. The required pieces of information regarding an event are: event name, likelihood, causal agent (can be left empty in a case where no agent caused the event), and affected goals. Along with a listing of affected goals, a congruence value is provided for each goal. Congruence values can take the value -1 to 1 inclusive, and represent the effect the event has on the likelihood of a goal being achieved. Table 3.3 shows two beliefs, the princess being attacked by a monster, and a magic sword being found, and how these beliefs relate to the goals in Table 3.2.

	Princess Attacked	**Magic Sword Found**
Likelihood	0.5	1
Causal Agent	Monster	Knight
Affected Goal (Congruence)	Princess Die (0.8)	Kill Monster (0.5)

Table 3.3: Goals in the GAMYGDALA Engine [11].

Goals and beliefs are used in the engine's appraisal process to calculate desirability and likelihood, the two primary appraisal variables used by the engine to determine an emotional output. For a character, c, a goal, g, and a belief, b, desirability is defined as

$$desirability(b, g, c) = congruence(b, g) * utility(g) \tag{3.1}$$

Similarly, the likelihood of a goal is defined as

$$likelihood(g) = \frac{congruence(b, g) * likelihood(b) + 1}{2} \tag{3.2}$$

From the desirability of an event, likelihood of a goal, and change in likelihood of a goal, an emotional response can be determined. The internal emotion definitions described by Popescu et al. are as follows [11]:

- **Hope**: a desirable uncertain goal increases in likelihood of success or an undesirable uncertain goal decreases in likelihood of success
- **Fear**: an undesirable uncertain goal increases in likelihood of success or a desirable uncertain goal decreases in likelihood of success
- **Joy**: a desirable goal succeeds or an undesirable goal fails
- **Distress**: an undesirable goal succeeds or a desirable goal fails
- **Satisfaction**: a desirable goal was expected to succeed and it succeeds
- **Fears Confirmed**: an undesirable goal was expected to succeed and it succeeds
- **Disappointment**: a desirable goal was expected to succeed and it fails
- **Relief**: an undesirable goal was expected to succeed and it fails

In addition to internal emotions, a number of social emotions are modeled by GAMYGDALA:

- **Anger**: an undesirable event is caused by another NPC
- **Guilt**: this NPC causes an undesirable event for a liked NPC
- **Gratitude**: a desirable event is caused by another NPC
- **Gratification**: this NPC causes a desirable event for a liked NPC
- **Happy For**: a desirable event happens to a liked NPC
- **Pity**: an undesirable event happens to a liked NPC

- **Gloating**: an undesirable event happens to a disliked NPC
- **Resentment**: a desirable event happens to a disliked NPC

For social emotions, the extent to which one NPC "likes" another NPC is a function of the net effect of all events caused by that NPC. For example, if NPC1 causes an event that is desirable to NPC2, the amount that NPC2 likes NPC1 will increase.

Once an emotion is determined by appraisal, intensity can be calculated with the following equation (for internal emotions):

$$intensity(e) = |desirability(b, g, self) * \Delta Likelihood(g)| \tag{3.3}$$

or, if it is a social emotion:

$$intensity(e) = |desirability(b, g, q \neq self) * \Delta Likelihood(g) * like(self, q)| \tag{3.4}$$

where q is the other character, and $like(self, q)$ is the numerical representation of how much the character likes q.

Each NPC is given a set of default emotional states and social stances. These default states can be used to incorporate a sense of personal history and personality into the modeled characters. For example, a character who is supposed to have a cheerful personality may have a default emotional state of joyful. Over time, characters' emotions and social stances decay towards their defaults [11]. As such, a character with a default emotional state of joyful would return to that emotional state over time in the absence of any new events that move their state back away from joyful.

Popescu et al. found that GAMYGDALA was able to effectively model emotions in a wide range of game genres, with the researchers writing implementations of

Pac-Man, a generic role-playing game, a generic real-time strategy game, and a generic first-person shooter game that used the GAMYGDALA engine to model character emotions. To illustrate the API to the engine, Popescu et al. provide a code snippet to illustrate the use of the engine, shown below.

```
EmoBrain brain = new EmoBrain()
brain.Goal("to live", 0.7f)
brain.Belief("provides house", 1f, "village")
brain.AffectsGoal("provides house", "live", 1f)
brain.Update()
brain.Goal("village destroyed", -1f, "village")
brain.Belief("village is unarmed", 0.7f)
brain.AffectsGoal("village is unarmed", "village destroyed", 1f)
brain.Update()
brain.Belief("provide weapons", 1f, "self")
brain.AffectsGoal("provide weapons", "village destroyed", -1)
brain.Update()
```

Listing 3.1: Code to incorporate GAMYGDALA engine into modeling a blacksmith in a role-playing game [11]

Here a new brain is created for the blacksmith, and the blacksmith is given his first goal, "to live," with a utility of 0.7. A belief then occurs, with the village providing the blacksmith with a house. The congruence between the village providing a place for the blacksmith to live and the blacksmith's goal of living is set to 1. This creates a positive feeling towards the village by the blacksmith, and triggers the emotion *gratitude* on the brain's update. A new goal is added to the blacksmith, giving him a strong negative utility to the village being destroyed. A new belief occurs, where the village is unarmed, with a likelihood of 1 (confirmed). The congruence between the village being unarmed and being destroyed is set to 1. Since this is a negative event for the village, and the blacksmith likes the village, pity is triggered on the brain's update. Finally, a belief of providing weapons occurs, with the blacksmith being passed in as the causal agent. The belief is defined to have a strong negative effect on the goal of the village being destroyed. Since this has a positive effect on the village, and the blacksmith likes the village, the *happy for* emotion is triggered. Additionally, since the blacksmith was the causal agent in a desirable event for the liked NPC (the village), *gratification* is also triggered [11].

Popescu et al. find that GAMYGDALA is able to model emotions with a high level of efficiency. Using a single-core, 3.0 GHz processor, GAMYGDALA was able to model 5000 NPCs with 5 goals each and 5 new beliefs per iteration in under one second (0.816 s) per iteration [11].

CHAPTER 4

FLAME MODEL

The Fuzzy Logic Adaptive Model of Emotion, or FLAME, is a computational model for emotion proposed by Magy Seif El-Nasr, John Yen, and Thomas R. Ioerger in "FLAME - Fuzzy Logic Adaptive Model of Emotions". This model uses two event-appraisal models: the OCC model [9] and the model proposed by Roseman, Spindel, and Jose [13] as a foundation. Likewise it is rooted in the inhibition model proposed by Bolles and Fanslow [4]. However, FLAME focuses on adaptation and learning, an aspect the creators feel is lacking in many other emotional models and computational models of emotion. While other models typically work in predetermined ways, where event E has effect A on some simulated agent, FLAME lets simulated agents learn from remembered experiences, adaptively changing the effect of some event. This approach fits with psychological studies that have found learning and memory to be highly important to the mechanisms that determine emotion in humans [5].

4.1 MODEL OVERVIEW

Many emotional models incorporate an agent's expectation of an event into the calculation of that agent's emotional state. While that expectation can be hard-coded into a system by the system's developer, letting expectations change dynamically with events allows for more adaptive and believable emotional agents, particularly in systems where a human user is interacting with said agents. One mechanism by which expectations can dynamically change is classical conditioning, where some initially neutral stimulus comes to be associated with a particular emotion. For example, if a dog sees its owner open the closet that contains the dog's leash each time the owner takes it for a walk, the dog may come to expect being taken for a walk whenever the owner opens the closet door. In this case, the door being opened, originally a neutral stimulus, alters the expectation of the dog, triggering excitement.

In addition to modeling classical conditioning to create associations between objects and events, the FLAME model uses machine learning techniques to let agents develop associations and expectations relating to sequences of events, along with expectations regarding the behavior of a human user. For sequences of events, a mechanism is proposed for enabling agents to connect sequences of events to their final outcome. In terms of expectations of human users, FLAME lets agents learn from user feedback to create expectations of user response to a given event or behavior [5]. A more in-depth discussion of the model's learning mechanisms is provided later in this section.

The FLAME model is also unique in regard to other computational models of emotion in its heavy use of fuzzy logic. Fuzzy logic provides an intuitive way to include intensity value in an internal representation of emotion, along with allowing smooth transitions between emotional states and mixed intensities of different states, enabling complex emotions. The natural expressiveness of fuzzy logic also lends itself well to rulesets that determine behavior based on events and emotions.

4.2 Event Appraisal Models

As previously mentioned, FLAME is heavily influenced by event appraisal models, particularly the model proposed by Roseman, Spindel, and Jose [13] and the OCC model [9]. Event appraisal models focus on calculating emotion through a number of assessments an agent makes in regards to an event. These assessments are made using a number of variables, with the predominant factors being:

- Whether the event is consistent or inconsistent with the agent's desires and goals
- Whether the agent in question or another actor, or no one in particular, is responsible for the event
- The degree to which the agent expects/expected the event to occur
- Whether the event would be followed by a punishment or reward for the agent in question
- The degree to which the agent feels capable of influencing the outcome of the event

Emotions are then defined in terms of these factors (and many others, depending on the appraisal model being used), allowing the determination of emotional state in response to an event. Table 4.1 shows a number of emotions and their appraisal variables under the OCC model [9].

In Table 4.1, we see that if an agent has the expectation that an undesirable event will occur, but that event ends up not occurring, the agent will feel relief. Likewise, if an agent expects a desirable event to occur, and that event does not occur, the agent will feel disappointment.

While FLAME is strongly influenced by these event appraisal models, it is designed to address some common shortcomings of using event appraisal for

Emotion	**Desirability**	**Expectation of Occurrence**	**Event Occurred**
Joy	Desirable	n/a	Yes
Relief	Undesirable	High	No
Fear	Undesirable	High	n/a
Disappointment	Desirable	High	No

Table 4.1: Emotional states produced through appraisal of desirability and expectation.

modeling emotion. Event appraisal models do not provide clear mechanisms by which events are determined to be desirable or undesirable by agents, nor do they typically have defined methods by which an agent can determine the likelihood of an event occurring, both of which are crucial to realistically simulating emotion. Additionally, appraisal-based emotional modeling is focused outward from the agent, with external factors and events taking the dominant role in the determination of an agent's emotional state. This ignores the importance of internal factors such as memory, experience, personality, and intelligence in emotion. The FLAME model seeks to address these issues through the incorporation of machine learning, giving agents the capacity to reason about events, their desirability, and their probability, based on the past experiences of the agent. The FLAME model also incorporates components of inhibition models to provide a mechanism for filtering the conflicting emotions that can sometimes arise from appraisal of events that may have both desirable and undesirable effects [5].

4.3 FLAME Model Design

There are three main components that make up the FLAME model: the emotional component, the learning component, and the decision-making component [5]. These components form a pipeline that results in a decision being made by the agent. The learning component provides expectations and associations regarding events to the emotional component, which it uses in conjunction with its current knowledge

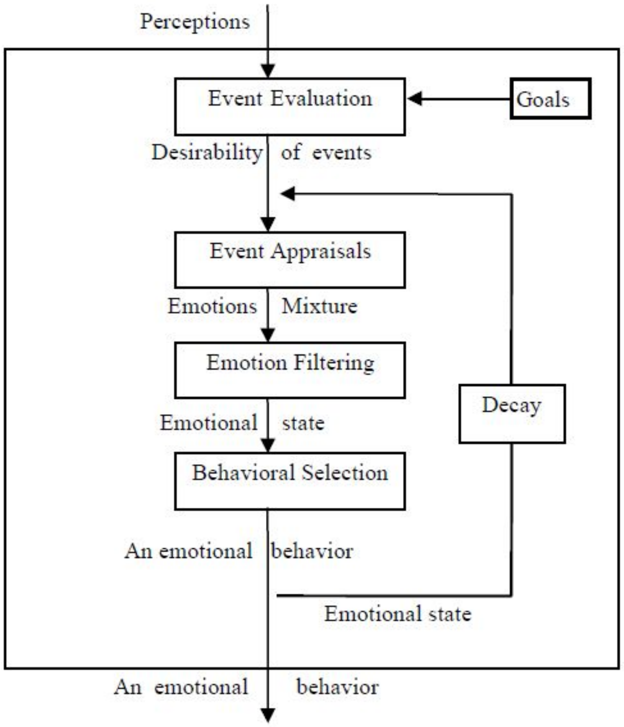

Perceptions
Event Evaluation
Goals
Desirability of events
Event Appraisals
Emotions Mixture
Emotion Filtering
Decay
Emotional state
Behavioral Selection
An emotional behavior
Emotional state
An emotional behavior

Figure 4.1: The emotional component of the FLAME model [5].

of the surrounding world to determine an emotional state for the agent. The decision-making component can then combine information about an event with the agent's current emotional state to decide on an action to perform. For example, to go back to the dog used in a previous example, the learned (learning component) response to its owner opening the closet door would give the dog a high expectation of a desirable event, which would be translated into hope (emotional component). A hopeful emotion and the door-opening event could then trigger the dog to take the action of running to the front door (decision-making component).

The emotional component is composed of 5 core processes: event evaluation, event appraisals, emotion filtering, behavioral selection, and decay. Figure 4.1 shows how the emotional component translates a perception into an emotional behavior through these processes. The event evaluation process calls on the learning component for learned information from past experiences. This information is then used to determine the importance of the agent's goals, whether an event affects these goals, and to what extent. Fuzzy logic rules are then used to map this information to a desirability value for the given event [5]. With the desirability of an event calculated, the agent can then perform an event appraisal to calculate the intensity of each emotion available to the agent. The result is a mixture of emotions with varying intensity, which are then passed through an emotional filtering process, which inhibits certain emotions in the case of conflicting emotional states to produce a final emotional state. With a current emotional state determined, behavior selection can then occur using rules based on the emotional state of the agent and the particular event that occurs.

Finally, a decay process occurs at the end of each iteration of the emotional component. The decay process models the natural dissipation of emotions over time, with an emotional response to an event decaying towards a neutral state as time since that event increases. This decay process lets the emotional effects of one event continue into later iterations through the emotional component, with the lasting effect having decreased influence with each iteration. Decay is achieved by determining a decay constant for positive emotions and a decay constant for negative emotions, with the authors suggesting a positive decay constant between 0.1 and 0.3, and a negative decay constant between 0.4 and 0.5. When the decay process is carried out, the intensity of a negative emotion is multiplied by the negative decay constant, and the intensity of a positive emotion is multiplied by the positive decay constant. As such, if the intensity of the fear emotion is 0.8 through one iteration through the emotional component, the intensity will be 0.4 at the initialization of the next iteration if the negative decay constant is 0.5 [5].

4.4 CALCULATING DESIRABILITY

As discussed previously in relation to event appraisal, it is essential to be able to calculate the desirability of a particular event to an agent in order to make an appraisal of the event for that agent and generate a corresponding emotion. In FLAME, these desirability calculations are performed using fuzzy sets and fuzzy rules. By representing emotions with fuzzy sets, FLAME allows the mapping of complex emotional states and interacting goals to behaviors. Due to the ease of expressing fuzzy rules linguistically, these mappings are easy to understand and create [5]. Rules to determine the desirability of an event take an if... then form, as shown below:

If Impact($Goal_1$, $Event$) is A_1
And Impact($Goal_2$, $Event$) is A_2
. . .
And Impact($Goal_k$, $Event$) is A_k
And Importance($Goal_1$) is B_1
And Importance($Goal_2$) is B_2
. . .
And Importance($Goal_k$) is B_k
Then Desirability($Event$) is C

In this way, desirability is calculated using the net impact of the event on one or more affected goals and the importance of those goals. For example, if an agent has a goal to win a running race, the event of that agent spraining its ankle has a highly negative impact on that goal. If winning the race is a moderately important goal for the agent, then these two factors together will give the event a desirability rating of moderately undesirable. Using the notation above, this rule takes the form:

If Impact(Winning Race, Spraining Ankle) is Highly Negative
And Importance(Winning Race) is Moderately Important
Then Desirability(Spraining Ankle) is Moderately Undesirable

4.5 Appraising Events

Using the desirability level calculated above, an event appraisal can be made to determine the emotional state resulting from the event. Three key components go into an appraisal under the FLAME model: the calculated desirability of an event that occurs, the agent's expectation that the event would occur, and the learned effects of previous events. The process by which an agent determines its expectation for an event to occur is discussed in a later section. A history of previous emotional states is essential to realistically modeling emotion, as many emotions are dependent on prior states. For example, if an agent desires an event, but is unsure whether that event will occur, it will feel hopeful. If the agent then finds out that the event did not occur, it will then feel disappointment. In this case, the feeling of disappointment is dependent on the feeling of hope that precedes it. Table 4.2 shows the emotions modeled by FLAME, which are based on those outlined in the OCC model [9].

Emotion	**Rule**
Joy	Occurrence of a desirable event
Sadness	Occurrence of an undesirable event
Disappointment	Occurrence of a disconfirmed desirable event
Relief	Occurrence of a disconfirmed undesirable event
Hope	Occurrence of an unconfirmed desirable event
Fear	Occurrence of an unconfirmed undesirable event
Pride	Action done by the agent and is approved by external standards
Shame	Action done by the agent and is disapproved by external standards
Reproach	Action is done by other and not approved of by the agent's standards
Admiration	Action is done by other and approved of by the agent's standards
Anger	Complex emotion formed from sadness and reproach
Gratitude	Complex emotion formed from joy and admiration
Gratification	Complex emotion formed from joy and pride
Remorse	Complex emotion formed from sadness and shame

Table 4.2: Fundamental emotions and the situations from which they arise (El-Nasr et al., 2000).

From this table, we see that disappointment must be preceded by hope, as disappointment is the disconfirmation of a desirable event, and prior to that disconfirmation, there must have been the existence of an uncomfirmed desirable event (hope). By establishing these relative relationships, a history of emotions can be used to trigger new emotions. In the example above, if hope was felt in one iteration, and in the next step that desirable event was disconfirmed, then disappointment can be triggered as the succeeding emotion, with the prior relevant emotion affecting the intensity of the new emotion (in this case, a higher intensity of hope would result in a higher intensity of disappointment when the desirable event is disconfirmed) [5].

To calculate the intensity of an emotion that is a function of expectation and desirability, FLAME uses equations determined in Price et. al [12]. The formula for each relevant emotion is given in Table 4.3.

Emotion	**Formula for Intensity**
Joy	$Joy = (1.7 * expectation^{0.5}) + (-0.7 * desirability)$
Sadness	$Sadness = (2 * expectation^{2}) - desirability$
Disappointment	$Disappointment = Hope * desirability$
Relief	$Relief = Fear * desirability$
Hope	$Hope = (1.7 * expectation^{0.5}) + (-0.7 * desirability)$
Fear	$Fear = (2 * expectation^{2}) - desirability$

Table 4.3: Equations for intensity of expectation and desirability-dependent emotions. (El-Nasr et al., 2000).

4.6 Filtering Emotions

At the conclusion of each iteration of the event appraisal, the agent has some mixture of emotions, with an intensity associated with each emotion. At this point, some filtering process must occur to determine how certain emotions express themselves more or less when this combination of intensities occurs. For example, in a case where an agent feels intense fear and moderate joy, the fear will typically inhibit the feeling of joy until a time when the fear becomes less intense.

One intuitive method for filtering emotions is to make the most intense emotion inhibit the others. However, this prevents the occurrence of complex emotional states that combine multiple emotions at different intensities. FLAME addresses this issue by using motivational states.

Motivational states deal with the most fundamental needs of the agent. These states are often domain-dependent. For example, Nasr et al. use a simulated dog to test their model, so their motivational states are hunger, thirst, pain, and fatigue. However, these motivational states may be much different in a economic model simulating agents trading stocks. As such, to be generalizable, the model must provide some mechanism for defining motivational states and how to determine their intensities.

Each motivational state is given an intensity, represented by a fuzzy set [5]. During the emotional filtering stage of the emotional component, the model assesses the intensity of the agent's motivational states and inhibits emotions accordingly. For example, in a case where the hunger motivational state has a high intensity, any joy that was calculated during the event appraisal process will be suppressed. In other cases, emotions may inhibit motivational states. Intense fear tends to inhibit hunger, thirst, pain, and fatigue in many cases.

In addition to the capability of motivational states and emotions to inhibit each other, emotions need to be able to inhibit each other too. FLAME takes the approach of having a higher-intensity emotion inhibit a lower-intensity emotion when those emotions are oppositional to each other. If an agent feels anger at a high intensity and gratitude at a low intensity, the anger will dominate and suppress the gratitude.

The final mechanism that FLAME uses to filter emotion is mood. In contrast to some other models, FLAME uses mood as a tool to filter emotions, rather than as an emotional state itself. Mood is determined by calculating whether the net positive intensity of these 5 states was greater than or less than the net negative intensity. This assigns either a positive mood to the agent or a negative mood. The mood is then used as a filtering mechanism in cases where opposite emotions are felt in similar intensities. If the agent's mood is negative, and a negative emotion is felt at a similar intensity to an opposite positive emotion, the mood of the agent will influence the negative emotion to inhibit the positive emotion [5].

4.7 Rule-Creation for Determining Behavior

Behavior selection in the FLAME model is performed using fuzzy logic. The ease with which fuzzy rules are mapped to and from natural language makes for intuitive rule creation, an important aspect to consider when deciding on a model that will be used in an emotional engine that is intended to be incorporated into other software developers' work. These rules incorporate the agent's emotional state, a triggering event, and the agent that triggered the behavior to determine what behavior to perform. The general form for these rules is provided below [5].

If $Emotion_1$ is $Intensity_1$

And $Emotion_2$ is $Intensity_2$

. . .

And $Emotion_k$ is $Intensity_k$

And Event is E

And Cause of E is Agent

Then Behavior is F

The ability to consider the intensity of multiple emotions enables the creation of behaviors based on complex emotional states. By incorporating events and their causes, the model has the expressive capacity to let agents respond dynamically to events based on their current emotional state. For example, a person may respond very differently to an event if they feel guilty as opposed to hopeful. Finally, the use of event causes in behavior selection rules ensures that behaviors will be properly directed. An angry behavior in a case where another agent caused the triggering event will be distinctly different than an angry behavior in a case where the agent itself caused the event.

4.8 Learning in the FLAME Model

A key component of FLAME is its incorporation of learning mechanisms. In their research, Nasr et al. found that learning was an essential component of intelligent behavior. In the testing of their model, they asked test users to evaluate a simulated dog based on the FLAME model. In situations where the dog's behavior was chosen randomly, the participants, on average, rated the overall intelligence of the dog as 1.143 out of 10. Using only the appraisal component of the model, the mean overall intelligence score rose to 3.05. Once the learning component was incorporated into the model, the mean overall intelligence score rose to 7.05 out of 10. Likewise, participants were asked to score the dog's overall ability to learn from the environment around it and interactions with the user. Here, the random case had a mean score of 0.24, while the appraisal without learning case had a mean score of 1.14. The incorporation of learning improved scores drastically, raising the mean score to 7.95 out of 10 [5]. Both of these cases illustrate significant benefits to the perceived realism and intelligence of the simulated agent once learning is introduced into the emotional model. In FLAME, there are four key learning mechanisms:

- Associating objects in the environment with events and emotions (classical conditioning)
- Reasoning about the impact of an event on the individual's goals
- Reasoning about the likelihood of an event's occurrence
- Learning about social values from the feedback of others

Each of these components is discussed in greater depth in the following sections.

4.8.1 Classical Conditioning

In the context of emotional modeling, classical conditioning is the pairing of a neutral stimulus with an emotional response. In the most famous example of classical conditioning, Russian physiologist Ivan Pavlov began ringing a bell prior to feeding his dog. With repeated feedings preceded by the bell, Pavlov found that eventually the dog began to salivate from the bell ringing (neutral stimulus) alone, without any food needed as a stimulus.

In the FLAME model, classical conditioning is a learning technique that allows simulated individuals to associate objects with an emotional response. For example, a neutral object that is present for repeated fearful experiences will eventually elicit fear through its presence alone. The model takes into account the frequency and severity of the conditioning. As such, objects that are frequently presented with a particular emotion elicit a strong response, as do objects that are presented with an emotion of high intensity. FLAME determines the intensity of the elicited emotion with the following equation [5],

$$(Intensity(Emotion)|Object) = \frac{\sum_{events(i)} I_i(Emotion)}{NumEvents_{Object}}. \tag{4.1}$$

Here we see that the intensity of the elicited emotion, given that the object has been presented $(Intensity(Emotion)|Object)$, is determined by taking the intensity of the emotional response $(I_i(Emotion))$ to all past events involving the object $(\sum_{events(i)})$ and dividing by the total number of events involving the object $(NumEvents_{Object})$. As such, the intensity of the response is the average of all previous emotional responses involving the emotion and object in question.

Through the classical conditioning mechanism, the model can bypass the emotional process and elicit emotional states without appraisals. For example, happiness can be triggered directly by the classical conditioning mechanism described above rather than the emotional component discussed in section 4.3.

4.8.2 EVENT IMPACTS

A key component of an appraisal model's ability to determine an appropriate emotional response is the ability to reason about how an event will impact the agent's goals. While some events have an immediate impact on an agent's goals, cases frequently occur where an event or action triggers a set of subsequent events or actions that eventually affect the agent's goals. In cases like this, the agent is able to recognize that the initial event or action played a role in the final result, as opposed to assigning all responsibility to the final event in the chain [5]. Q-learning, discussed in Section 2.2, is used by the FLAME model to let the agent learn about the long-term effects of events. As discussed in this section, the fundamental equation that drives Q-learning is,

$$Q(s,a) = r(s,a) + \gamma max_{a'}(Q(s',a')). \tag{4.2}$$

Where $Q(s,a)$ is the Q-value for a state-action pair, $r(s,a)$ is the immediate reward for a state-action pair, and $Q(s',a')$ is the Q-value of a state-action pair following the current action, and γ is a decay constant between 0 and 1 that determines how much the rewards of subsequent actions determine the Q-value of a current action.

While Q-learning is proven to converge to the optimal decision-making values in a deterministic system, FLAME models a non-deterministic system, where the results of an action taken from a particular state are not necessarily the same each time that action is executed in that state. To address this, El-Nasr et al. incorporate conditional probability into the Q-learning algorithm, with more likely results affecting the final value more than less likely results. The modified equation is shown below [5],

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) max_{a'}(Q(s',a')). \tag{4.3}$$

Two changes are made in the new equation. First, the reward for the state-action pair (s,a) is changed to the expected value of the reward for the state-action pair (s,a), which is the average of all previous rewards for this pair. Second, the Q-value is now modified by the probability that the state s' will result from the state-action pair (s,a). This results in a summation across all possible resulting states from this pair, based on the agent's past experiences.

For example, if the agent experiences two different states as a result of taking action a in state s, the new Q-value is determined as follows. We call the first resulting state s_1 and the second resulting state s_2, with the maximum Q-value for s_1 being 2, and the maximum Q-value for s_2 being -1. If s_1 is the result of taking action a from state s for twenty percent of past experiences, and s_2 was the result for eighty percent of past experiences, the expected Q-value can be calculated:

$$\sum_{s'} P(s'|s,a) max_{a'}(Q(s',a')) = \tag{4.4}$$

$$P(s_1|s,a) max_{a'}(Q(s_1,a')) + P(s_2|s,a) max_{a'}(Q(s_2,a')) = \tag{4.5}$$

$$(0.2)(2) + (0.8)(-1) = -0.4. \tag{4.6}$$

In addition to incorporating conditional probability to account for the non-determinism inherent in the modeled situations, FLAME uses the agent's current mood to influence how the agent perceives the value of certain actions. When the agent is in a positive mood, it expects more positive results from its actions, and likewise, when the agent is in a negative mood, it expects more negative results from its actions. This is achieved by using a β value to increase the Q-value for actions that result in a state whose perception matches the agent's mood. For example, with a β value of 0.25 and an agent in a positive mood, actions resulting in a positive state are favored as more likely by a factor of 1.25 compared to their original probabilities. Likewise, states that do not match the current mood are modified to be seen as less likely by an α value computed with the following equation [5],

$$\alpha = \frac{1}{\sum_{nonMatch}} (1 - (1 + \beta) \sum_{Match} P(s'|s,a)). \tag{4.7}$$

These modifications result in a final Q-value update equation for the modified Q-learning algorithm implemented by FLAME,

$$Q(s,a) = E[r(s,a)] + \gamma \sum_{Match} (1+\beta)P(s'|s,a)max_{a'}(Q(s',a'))+ \\ \gamma \sum_{nonMatch} \alpha P(s'|s,a)max_{a'}(Q(s',a')). \quad (4.8)$$

4.8.3 Determining Expectations

In the emotional models discussed in Chapter 3, the individual's expectation for an event to occur is typically input into the system as a hardcoded value. For example, a user of GAMYGDALA may input a belief and specify that there is a likelihood of 0.6 that the belief will occur. FLAME attempts to create more dynamic expectations by letting the modeled agent determine the likelihood of an event on its own based on past events using conditional probability.

To calculate the conditional probability of an event occurring, a record of past event sequences must be kept. To do this, an event sequence length must be chosen. A short memory of events may not let the agent establish important relationships between events in a sequence, but increasing the lengths of sequences in memory drastically increases the total number of sequences the individual must manage. For their simulation, the authors chose to do sequences of maximum length 3. If three events, X, Y, and Z, are defined, then a three-dimensional table can be kept to keep track of all past events. Each time the sequence of events X, Y, Z occurs, the corresponding entry in the table is incremented. This lets the table represent the total number of times any sequence of three events has occurred. From this, the conditional probability that event Z will occur given that events X and Y have occurred can be calculated as follows:

$$P(Z|X,Y) = \frac{C[X,Y,Z]}{\sum_i C[X,Y,i]} \quad (4.9)$$

Here the probability of Z occurring given that X and Y have occurred is the count of occurrences of the sequence X, Y, Z, divided by the total number of occurrences of sequences that began with X, Y.

In cases where the agent doesn't have enough prior experience to effectively assess longer chains of sequences, a single prior event can be used to determine expectation:

$$P(Z|Y) = \frac{\sum_i C[i, Y, Z]}{\sum_j \sum_i C[i, Y, j]} \tag{4.10}$$

In cases where the expected event has never occurred after the preceding sequence, the expectation for the event is the average probability of all events in the table:

$$P(Z) = \frac{\sum_{i,j} C[i, j, Z]}{\sum_{i,j,k} C[i, j, k]} \tag{4.11}$$

4.8.4 Social Values

The learning methods described above rely on reinforcement learning, where an agent learns from previous rewards and uses those experiences to try to maximize future rewards. All of these learning systems focus only on how the agent itself is rewarded. To increase the complexity and flexibility of the model, emotions based on the desires of others must also be modeled. This lets individuals assess how events and the individual's actions fit into value systems imposed by other agents or a user.

Social values are tracked as an average of all external feedback for previous actions. This relies on the incorporation of a feedback system, where one agent, or the user, can perform different feedback actions to another agent. Each of these feedback actions has an associated *value*, with values greater than one representing positive feedback and values less than one representing negative feedback. For an action, a, the expected social value of the action can be calculated with the following equation:

$$value(a) = \frac{1}{|A|} \sum_{e \in A} feedback(e+1) \tag{4.12}$$

where $e + 1$ is the external response to the event following the action. Using this expected social value, social emotions like pride, shame, reproach, and admiration can be modeled in a learned setting.

CHAPTER 5

Model Design and Implementation

5.1 Purpose

This chapter describes the implementation of the software portion of this research. The implementation provides a domain-general emotional engine based on the FLAME model. Figure 5.1 shows the general structure of the model.

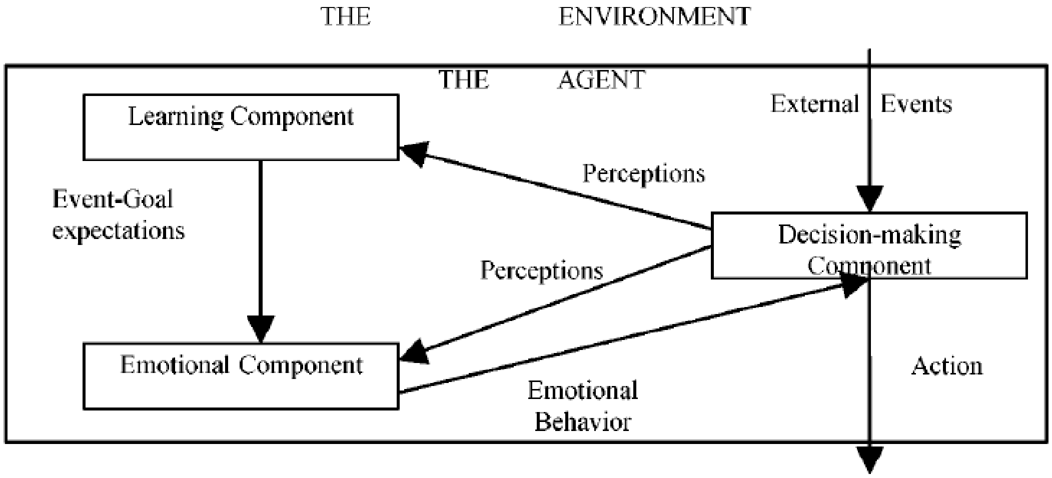

THE ENVIRONMENT
THE AGENT
Learning Component
External Events
Perceptions
Event-Goal expectations
Decision-making Component
Perceptions
Emotional Component
Action
Emotional Behavior

Figure 5.1: High-level overview of FLAME model. [5]

The externally visible components of the software are limited to the *Decision-making Component* shown in figure 5.1. As such, the engine maps external events in the environment to actions based on an agent's internal emotional state. For example, an agent in a simulated farming community may serve a feast (action) due to their relief (emotional state) over a large harvest (event). Changing the event to a poor harvest changes the emotional state (disappointment, fear), and the consequent action is changed as a result. In this way, the software allows dynamic emotional states and behaviors to emerge from events occurring in a simulated world.

The engine itself is written in JavaScript, with the intention of being a an easy-to-use library for emotional modeling in web apps, particularly in-browser games and social science simulations. Interactions with the engine are performed through an API that requires minimal user knowledge of the workings of the engine or emotional modeling in general.

5.2 SOFTWARE OVERVIEW

5.2.1 FUZZY LOGIC

The engine relies heavily on fuzzy logic for rule definitions. A basic implementation of a JavaScript fuzzy logic library facilitates this. The class diagram for this system is shown in Figure 5.2.

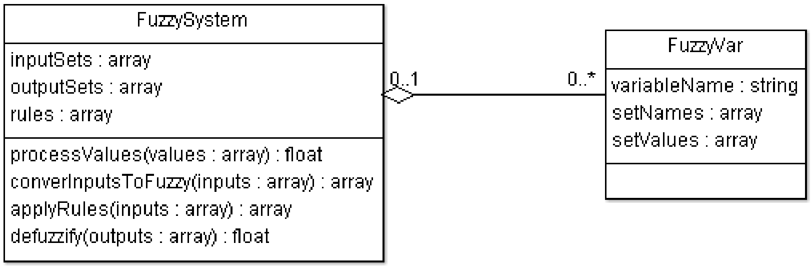

FuzzySystem
inputSets : array
outputSets : array
rules : array
processValues(values : array) : float
converInputsToFuzzy(inputs : array) : array
applyRules(inputs : array) : array
defuzzify(outputs : array) : float
0..1
0..*
FuzzyVar
variableName : string
setNames : array
setValues : array

Figure 5.2: Fuzzy system architecture.

New fuzzy variables are created by instantiating a FuzzyVar object. All fuzzy sets in the system have triangular membership functions. At creation, all needed components for the variable are provided to the constructor in the form,

```
var temperature = new FuzzyVar(variableName, setNames,
    setValues);
```

where *variableName* corresponds to the name of the fuzzy variable, *setNames* is an array of the variable's fuzzy set names, and *setValues* is an array of arrays containing the three x values of the membership function's leftmost point, apex, and rightmost point. To create a fuzzy variable relating to temperature, where inputs will fall between 0 and 100 degrees fahrenheit, the user enters,

```
var name = "Temperature"
var sets = ["Cold", "Cool", "Warm", "Hot"]
var values = [[-1,0,40], [30,45,60], [50,60,70],
    [60,100,101]];
var temp = new FuzzyVar(name, sets, values);
```

To create a full fuzzy system that maps one or more crisp inputs to a crisp output, a FuzzySystem object must be created. This is achieved with the code,

```
var fuzzy = new FuzzySystem(inputSets, outputSets, rules)
```

where *inputSets* is a list of fuzzy input variables, *outputSets* is a list of fuzzy output variables, and *rules* is an array of mappings from input variables to output variables. Rules are entered as strings of the form,

```
"InputVar SetName Operator ... THEN OutputVar SetName"
```

The system supports both OR and AND operators. To enter the fuzzy rules discussed in the example in chapter 2, which mapped temperature and room occupancy to the action of a heating/cooling system, the following are the entered rules.

```
["Temp Cold OR Temp Hot THEN Action High",
"Temp Cool AND Occupied Low THEN Action Low",
"Temp Cool AND Occupied High THEN Action Medium",
"Temp Warm AND Occupied Low THEN Action Low",
"Temp Warm AND Occupied High THEN Action Medium"]
```

The fuzzy system uses a defuzzification process proposed by Ginart and Sanchez [6]. The process estimates the centroid of a region by creating a single triangle with approximately the same mass distribution as the region, then taking the centroid of the triangle. Figure 5.3 shows the calculated triangle (dotted line) overlaid on the region it estimates.

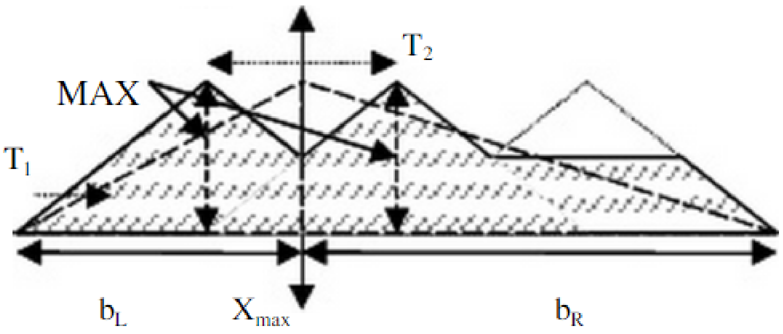


Figure 5.3: Centroid estimation using a representative triangle [6].

The centroid calculation is performed using the equation,

$$\overline{x_t} = x_{max} + \frac{1}{3}(b_R - b_L) \tag{5.1}$$

where x_{max} is the x value of the point in the region with the maximum membership, and b_R and b_L are the distances along the x axis from x_{max} to the right and leftmost nonzero values, respectively. This calculation is computationally efficient, requiring a single loop over the sets, and shows better error results than competing Mean of Maximum (MoM) and Bisector methods for centroid estimation [6].

5.2.2 Emotional Engine

The Engine class drives all of the emotional engine's functions, and acts as the entry point for all interactions between the user and the engine. From this class, each module in learning and emotional components is triggered. Figure 5.4 shows the overall class structure for the engine.

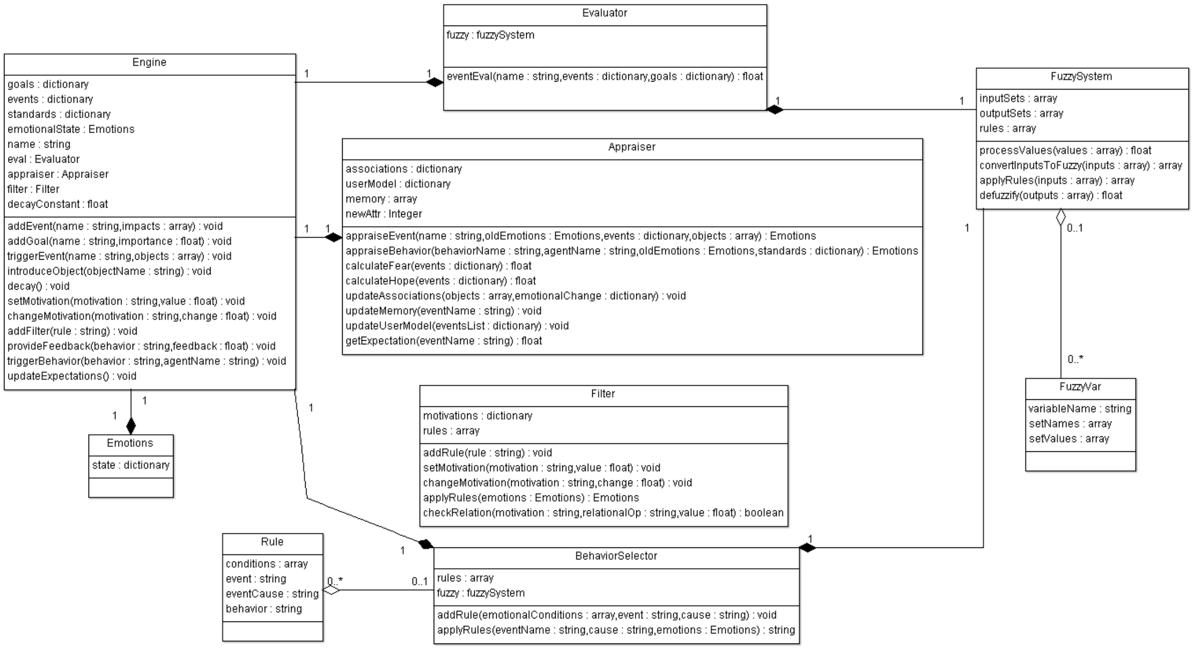

Evaluator
fuzzy : fuzzySystem
eventEval(name : string,events : dictionary,goals : dictionary) : float
Engine
goals : dictionary
events : dictionary
standards : dictionary
emotionalState : Emotions
name : string
eval : Evaluator
appraiser : Appraiser
filter : Filter
decayConstant : float
addEvent(name : string,impacts : array) : void
addGoal(name : string,importance : float) : void
triggerEvent(name : string,objects : array) : void
introduceObject(objectName : string) : void
decay() : void
setMotivation(motivation : string,value : float) : void
changeMotivation(motivation : string,change : float) : void
addFilter(rule : string) : void
provideFeedback(behavior : string,feedback : float) : void
triggerBehavior(behavior : string,agentName : string) : void
updateExpectations() : void
FuzzySystem
inputSets : array
outputSets : array
rules : array
processValues(values : array) : float
convertInputsToFuzzy(inputs : array) : array
applyRules(inputs : array) : array
defuzzify(outputs : array) : float
Appraiser
associations : dictionary
userModel : dictionary
memory : array
newAttr : Integer
appraiseEvent(name : string,oldEmotions : Emotions,events : dictionary,objects : array) : Emotions
appraiseBehavior(behaviorName : string,agentName : string,oldEmotions : Emotions,standards : dictionary) : Emotions
calculateFear(events : dictionary) : float
calculateHope(events : dictionary) : float
updateAssociations(objects : array,emotionalChange : dictionary) : void
updateMemory(eventName : string) : void
updateUserModel(eventsList : dictionary) : void
getExpectation(eventName : string) : float
0..1
0..*
FuzzyVar
variableName : string
setNames : array
setValues : array
Emotions
state : dictionary
Filter
motivations : dictionary
rules : array
addRule(rule : string) : void
setMotivation(motivation : string,value : float) : void
changeMotivation(motivation : string,change : float) : void
applyRules(emotions : Emotions) : Emotions
checkRelation(motivation : string,relationalOp : string,value : float) : boolean
Rule
conditions : array
event : string
eventCause : string
behavior : string
0..*
0..1
BehaviorSelector
rules : array
fuzzy : fuzzySystem
addRule(emotionalConditions : array,event : string,cause : string) : void
applyRules(eventName : string,cause : string,emotions : Emotions) : string
1

Figure 5.4: Full engine architecture.

To add emotional processing to an agent, the user constructs an Engine object for that agent with the command,

```
var brain = new Engine(id, decayConstant);
```

where *id* is a string identifier for the agent, and decayConstant is a decimal value greater than zero and less than one. The decay constant is used in the decay() function as a multiplier for emotional intensities, simulating their decay over time. The following calls the decay() function on the agent's emotional state:

```
brain.decay();
```

To add a goal definition to the agent, the user invokes the command,

```
brain.addGoal(name, importance);
```

where *name* is a string identifier for the goal, and importance is a floating point value between 0 and 1 indicating the importance of achieving the goal for the agent.

To add an event definition to the agent, the user invokes the command,

```
brain.addEvent(name, impacts);
```

where *name* is the name of the event, and *impacts* is an array of goal-impact pairings. This array alternates between goals and impacts, taking the form,

```
impacts = [goalName, impact, goalName, impact, ...];
```

Impacts are floating-point variables between -1 and 1, inclusive. Negative impacts indicate that the event makes a goal less likely to be achieved, and positive impacts indicate that the event makes a goal more likely to be achieved.

Once an event is added, it can be triggered with the command,

```
brain.triggerEvent(name, objects);
```

where *name* is the name used for the event in the addEvent(name, impacts) call that defined the event. *Objects* is an array of strings representing the names of objects that the agent should associate with this occurrence of the event. These associated objects are used in the classical conditioning mechanism to produce associations between objects and emotions. These associations can be activated by introducing the object to the agent, triggering any emotions the agent associates with the introduced object. This is achieved with the command,

```
brain.introduceObject(objectName);
```

where *objectName* is a string representing the name of the introduced object.

Motivational states are a key part of the model's determination of emotional state. New motivational states are created using the command,

```
brain.setMotivation(motivation, value);
```

where *motivation* is a string representing the name of the motivational state, and *value* is the initial intensity of the state. The setMotivation command can also be used to set an existing motivational state to a particular value. Relative changes to motivational states are performed with the command,

```
brain.changeMotivation(motivation, change);
```

where *change* is a number. Positive values increase the intensity of the motivational state and negative values decrease the intensity of the motivational state.

Filtering rules use motivational states to influence the emotions of the agent. New filtering rules are added using the command,

```
brain.addFilter(rule);
```

where *rule* is a string representing the rule. Rule strings must be of the form,

```
rule = "motivation relationalOperator value AND/OR motivation
    relationalOperator value AND/OR ... THEN emotion";
```

where *motivation* is a motivational state, *relationalOperator* is either ==, >, <, >=, <=, or ! =, *value* is a number, and *emotion* is the emotion that should be inhibited if the conditions evaluate to true. For example, the command,

```
var rule = "Hunger > 30 THEN Joy";
brain.addFilter(rule);
```

inhibits the emotion Joy when the intensity of the motivational state Hunger is greater than 30.

Social standards are a key part of emotional response in the model. Social standards are rooted in behavior feedback, which is achieved with the command,

```
brain.provideFeedback(behavior, feedback);
```

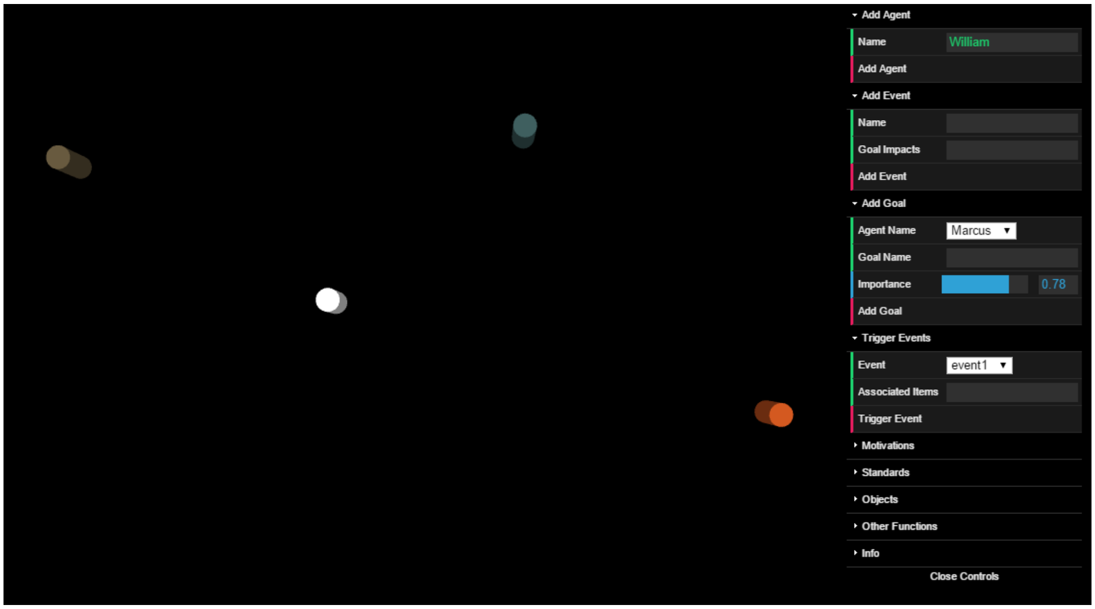
where *behavior* is a string representing the name of the behavior for which feedback is being provided, and *feedback* is a number representing the type and strength of the feedback. For example a *feedback* value of -2 provides strongly negative feedback for a behavior, while a *feedback* value of 0.2 provides weakly positive feedback for a behavior. Emotional responses to behaviors carried out by the agent themself or other agents are carried out with the command,

```
brain.triggerBehavior(behavior, agentName);
```

where *behavior* is the name of the behavior and *agentName* is the name of the agent who carried out the behavior. If the agent carried out the behavior themself, their own name should be used as the *agentName*.

5.3 Visualization

A web app was created with WebGL and the Three.js JavaScript library to visualize the workings of the engine. The app allows the creation of new agents by clicking inside a window. This creates a colored cylinder on-screen representing the created agent, with different colors representing different emotional states. Additional information on the agent's state is accessible through menus in the GUI. Figure 5.5 shows the web app's interface to the engine, with the screen populated with agents.



Name: William

Agent Name: Marcus

Importance: 0.78

Event: event1

Add Agent
Name
Add Agent
Add Event
Name
Goal Impacts
Add Event
Add Goal
Agent Name
Goal Name
Importance
Add Goal
Trigger Events
Event
Associated Items
Trigger Event
Motivations
Standards
Objects
Other Functions
Info
Close Controls

Figure 5.5: Model visualization and user interface.

The commands discussed in section 5.2.2 can be passed to the engine's API through a GUI in the web app, with the on-screen agents updating in response to the issued commands.

CHAPTER 6

Testing and Results

This chapter examines the operation of the engine through a series of tests. Each test is focused on illustrating a different mechanism of the engine. The code below demonstrates the core appraisal process of the engine.

```
brain = new Engine("William", 0.4);
brain.addGoal("findShelter", 0.75);
brain.addEvent("fog", ["findShelter", -0.3]);
brain.triggerEvent("fog", [""]);
brain.decay();
brain.decay();
brain.decay();
brain.addEvent("findMap", ["findShelter", 0.7]);
brain.triggerEvent("findMap");
brain.provideFeedback("exploring", 1.2);
brain.decay();
brain.addEvent("findRoad", ["findShelter", 0.7]);
brain.triggerBehavior("exploring", "William");
brain.triggerEvent("findRoad", [""]);
```

Listing 6.1: Test of appraisal system and social standards.

After line 4, William's emotional state is a mixture of sadness and fear (intensity 1.01) due to the expectation and occurrence of the Fog event, which has a negative impact on his goal of finding shelter. Over time, these feelings decay due to the decay() calls, leaving the intensity of both emotions at 0.06 after line 7. The introduction of the positive event, finding a map, and its subsequent occurrence trigger Joy and Hope (intensity 1.78) as expected after line 9. Fear also increases to 0.68 due to the fact that William still has some expectation that the negative Fog event will occur again. While sometimes appropriate, one unintuitive aspect of the current implementation is that events do not have a cool-down period, so the next occurence is still hoped for or feared immediately after the initial occurrence of an event.

It is also of note that with so few experiences to learn from, the learning mechanisms are not very active yet, making Joy/Hope and Sadness/Fear have equal intensities. This is due to the fact that there are not enough experiences in the system to meaningfully modify the event expectations from their initial values. The implication for the engine is that, like most learning-dependent algorithms, its full efficacy is not revealed until a large number of training experiences have occurred.

After line 14, William feels pride and gratification (intensity 1.20) due to his performance of a behavior that was positively reinforced in the past (line 10), and the fact that his behavior was followed by a positive result (finding the road). These responses are intuitive and of a reasonable intensity.

Overall, the test shows realistic emotional responses to both positive and negative events, as well as social standards.

```
brain = new Engine("Timothy", 0.4);
brain.addGoal("findShelter", 0.75);
brain.addEvent("findMap", ["findShelter", 0.7]);
brain.setMotivation("Hunger", 50);
```

```
5 brain.addFilter("Hunger > 30 THEN Joy");
6 brain.triggerEvent("findMap", ["compass"]);
7 brain.changeMotivation("Hunger", −30);
8 brain.introduceObject("compass");
```

Listing 6.2: Test of motivational states and classical conditioning.

This test focuses on motivational states and classical conditioning. Timothy is given some of the same goals and events as William was in the first test. However, the motivational state Hunger is introduced in line 4, with a filter added in line 5 that inhibits Joy when Hunger's intensity is greater than 30. The effects of this are seen after line 6, where Timothy feels hope from the "findMap" event, but the Joy he should also feel is inhibited, leaving him with a Joy intensity of zero due to the fact that his Hunger is greater than the limit set in the inhibition rule. While this works as expected, one issue with this implementation is that emotions are inhibited completely, whereas a system that only partially inhibits (multiplies by some constant between zero and one) emotional intensities would be more expressive.

The effects of classical conditioning are seen in line 8, where a decrease in Hunger has ended the inhibition of Joy. When the compass is introduced, Timothy's Joy intensity rises from 0 to 1.51 as a result of his association between the compass and finding the map. Like the inhibition, this behaves as expected based on the model, but could be made more expressive by making associations build in intensity over time, so that objects don't have such strong associations without repeated occurrences of associated positive or negative events.

```
brain = new Engine("Rachel", 0.4);
brain.addGoal("findShelter", 0.75);
brain.addEvent("event1", ["findShelter", 0.3]);
brain.addEvent("event2", ["findShelter", 0.3]);
```

```
5  brain.addEvent("event3", ["findShelter", 0.3]);
6  for(var runs = 0; runs < 5; runs++) {
7      brain.triggerEvent("event1", [""]);
8      brain.triggerEvent("event2", [""]);
9      brain.triggerEvent("event3", [""]);
10 }
11 brain.triggerEvent("event1", [""]);
12 brain.triggerEvent("event2", [""]);
13 brain.triggerEvent("event2", [""]);
14 brain.triggerEvent("event1", [""]);
15 brain.triggerEvent("event2", [""]);
```

Listing 6.3: Test of learning mechanism.

This test focuses on the agent's ability to learn from past experiences to predict what events are likely to occur in the future. Since this test focuses on triggering enough events to give the learning mechanism enough information to use, generic events event1, event2, and event3 are used. The sequence event1 then event2 then event3 is triggered repeatedly, with one occurrence of an altered sequence, event1 then event2 then event2, occurring in lines 11-13. Event1 and event2 are then triggered. After line 15, Rachel expects event3 with a measure of 0.8 and expects event2 with a measure of 0.2. This is intuitive, as the event1, event2, event3 sequence occurred much more frequently than the event1, event2, event2 sequence. As such, it is expected and reasonable for Rachel to expect event3 with a much higher probability than event2.

6.1 Future Work

El Nasr et. al demonstrated with FLAME that the addition of learning algorithms to emotional models drastically increases the believability of the emotions generated by the model. Future computational models of emotion that incorporate more comprehensive learning mechanisms can address FLAME's deficits, particularly in its focus on probabilistic models that don't necessarily correspond with human decision-making.

One major shortcoming of the current engine implementation is the isolation of a single instance of the model within each agent, with no overarching control structure between agents. This approach doesn't allow large-scale operations that work with multiple agents at a time, unless the developer of the surrounding software implements their own methods for tracking and operating on batches of individuals. GAMYGDALA is one example of such an implementation, with an individual "brain" for each individual encapsulated within a listing of all individuals managed by the engine.

While the FLAME model and this engine implementation take social standards into account, neither contain fine-grained management of social relationships. In the current implementation and model, an individual's emotional response to an action or event is the same regardless of the individual's relationship to the agent that performed the action or caused the event. Some management of relationships is achieved through the behavior selection module, which enables different behaviors to be chosen based on the individual that caused the event. However, this occurs after event appraisal and emotional filtering, meaning that the emotions of the individual are not changed based on the event's cause, only the resulting behavior. The introduction of mechanisms that alter appraisal based on social relationships would allow more dynamic and realistic social interactions.

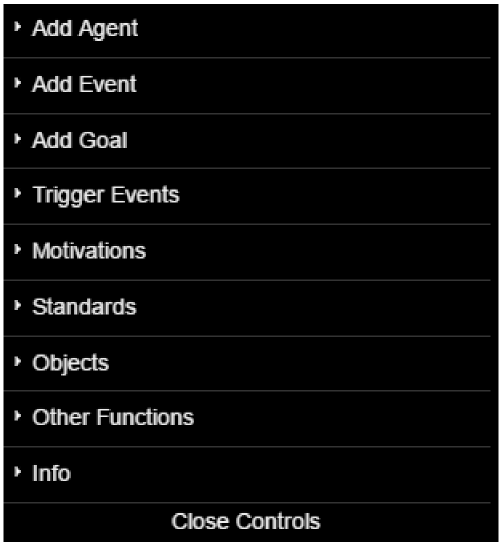
6.2 CONCLUSION

The implementation of the FLAME model discussed in this work demonstrates that it is possible to create emotional agents that are realistic and adaptive to the world around them. Agents modeled using the engine can be given intuitive and surprisingly complex emotions with ease due to the abstraction provided by the engine. As cognitive models of emotion continue develop, more advanced computational models of emotion will follow. While the discussed engine relies almost exclusively on the FLAME model, future adaptations can incorporate aspects of various models to increase emotional realism. In particular, more advanced learning mechanisms and models that incorporate social systems with more depth offer promising results to future engines.

APPENDIX A

SOFTWARE MANUAL

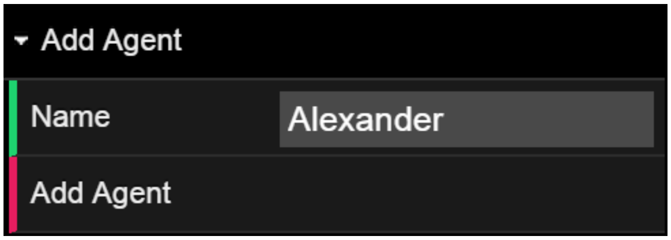
This section provides instructions for the use of the visualization created for the engine. To view the visualization, open the provided *index.html* file. All controls are in the menu at the top-right corner of the browser window, while the agents are visualized in the black windows that covers the rest of the screen. Figure A.1 shows the control menu.



Add Agent
Add Event
Add Goal
Trigger Events
Motivations
Standards
Objects
Other Functions
Info
Close Controls

Figure A.1: Visualization control menu.

Clicking on any portion of the menu opens that section of controls. Clicking the heading again collapses the controls. The *Add Agent* section is shown in Figure A.2.

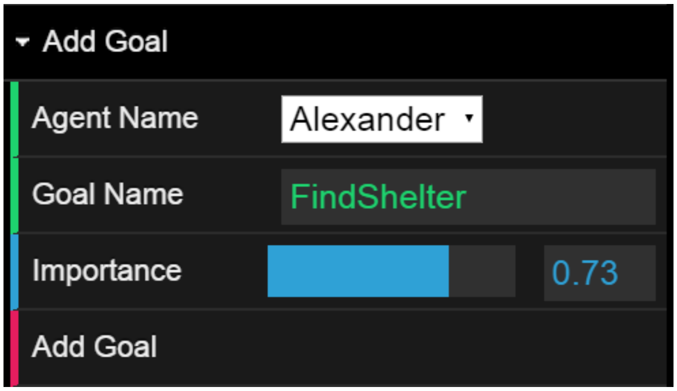


Name: Alexander

Figure A.2: Agent addition controls.

In this section, the user can enter a name for a new agent that will be used as an identifier in other control sets. Clicking the *Add Agent* button puts the software into agent entry mode. In this mode, the user can click anywhere on the screen to place the new agent, represented by a cylinder. Once the agent is placed, the software automatically exits agent entry mode, letting the user continue clicking on the screen without placing more agents until the *Add Agent* button is clicked again.

The *Add Goal* section is shown in Figure A.3.



Agent Name: Alexander ▾

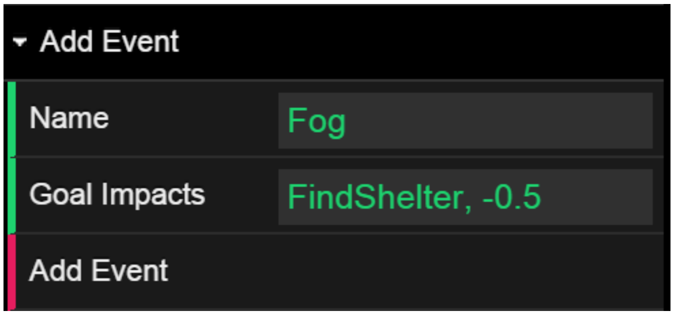
Goal Name: FindShelter

Importance: 0.73

Figure A.3: Agent goal controls.

In this section, the user is able to select an agent from a dropdown menu that dynamically updates as new agents are added. This is the agent who the goal will be added to. The *Goal Name* field lets the user enter a name to be used as the identifier of the goal, and the *Importance* slider informs the engine how important the goal is (on a scale from 0 to 1) to the agent. Once the desired parameters are chosen, clicking the *Add Goal* button gives the selected agent the described goal.

The *Add Event* section is shown in figure A.4.



Name: Fog

Goal Impacts: FindShelter, -0.5

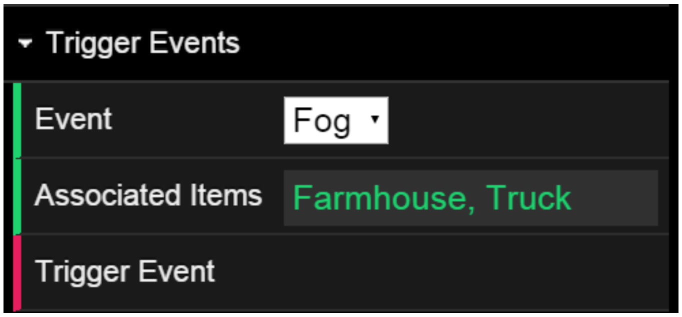
Figure A.4: Event addition controls.

This section contains a *Name* field, where the identifier for the new event is provided. Additionally, the *Goal Impacts* field lets the user describe how the event influences various goals. Entries in this field must take the form:

```
goalName, impact, goalName, impact, ...
```

where *goalName* is the (case sensitive) name of a defined goal, and impact is a value between -1 and 1 describing how the event affects the preceding goal. High, negative numbers reflect a strong negative impact on the achievement of the goal, whereas high, positive numbers reflect a strong positive impact on the achievement of the goal. Selecting the *Add Event* button adds the event to the system, but does not trigger it.

The *Trigger Events* section is shown in figure A.5.



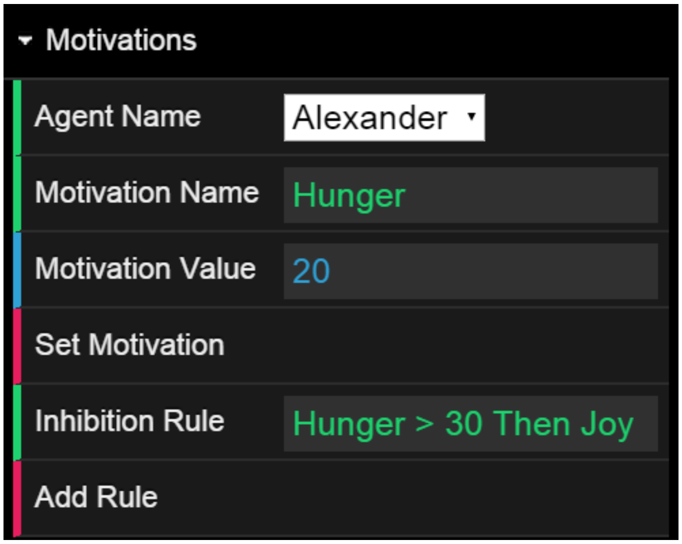
Event: Fog

Associated Items: Farmhouse, Truck

Figure A.5: Event triggering controls.

This section lets the user select an event defined in the *Add Event* section to be triggered. The *Associated Items* field takes a comma-separated list of objects that the agents should associate with that instance of the event for classical conditioning.

The *Motivations* section is shown in figure A.6.



Agent Name: Alexander

Motivation Name: Hunger

Motivation Value: 20

Inhibition Rule: Hunger > 30 Then Joy

Figure A.6: Motivational state controls.

The *Motivations* section is divided into two parts. The top section allows the creation and management of motivational states. The *Motivation Name* field provides the name for the motivational state to be modified, while the *Motivation Value* field provides the intensity of the motivational state. Entering a new *Motivation Name* creates a new motivational state when the *Set Motivation* value is clicked. To modify an existing state, enter the existing state's name and the new intensity for the state, then click the *Set Motivation* button.

In the *Inhibition Rule* field, the user can enter inhibition rules for the chosen agent. These rules take the form,

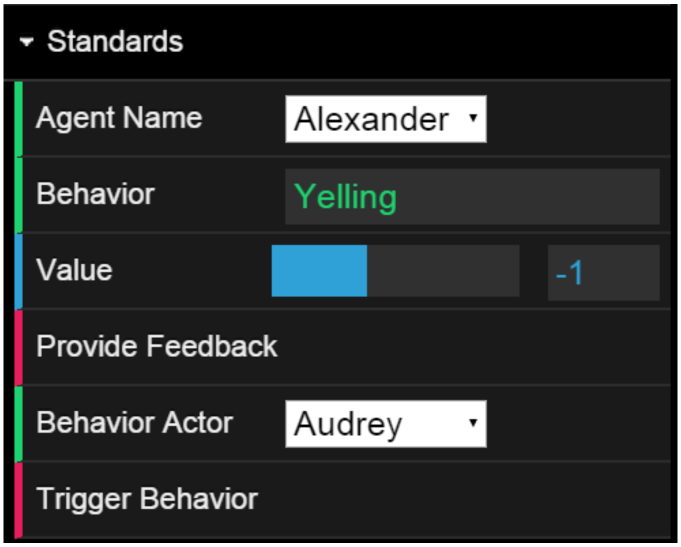
```
MotivationalState RelationalOperator Value AND/OR ... THEN InhibitedEmotion
```

For example, the rule,

```
Hunger > 30 AND Thirst > 20 THEN Joy
```

inhibits the emotion Joy when the Hunger motivational state rises above 30 and the Thirst motivational state rises above 20. Clicking the *Add Rule* button adds the new inhibition rule to the system.

The *Standards* section is shown in Figure A.7.



Standards

Agent Name: Alexander

Behavior: Yelling

Value: -1

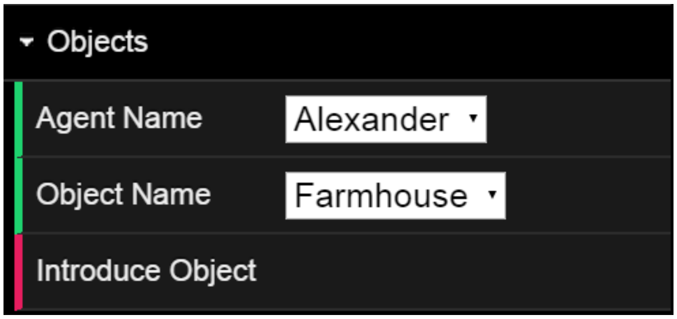
Behavior Actor: Audrey

Figure A.7: Agent standards controls.

The *Standards* section is divided into two parts, with the top section devoted to providing social feedback, and the bottom section devoted to letting the agent respond to the actions of themself or others. The *Behavior* field lets the user specify what behavior the feedback applies to, while the *Value* field specifies the nature of the feedback. For example, entering "HoldingDoor" as the *Behavior* and 0.4 as the *Value* provides slightly positive feedback to the agent, making them more likely to perceive holding the door positively when they do it themself or when others hold the door. Clicking the *Provide Feedback* button sends this feedback to the agent selected in the *Agent Name* menu.

The *Behavior Actor* menu lets the agent respond to their own and others' behaviors. The agent chosen in the *Agent Name* menu is the agent who responds, and the agent chosen in the *Behavior Actor* menu is the agent who performs the action. For example, if the agent has learned to perceive holding the door positively through feedback, they will feel pride when they are chosen as the *Behavior Actor* for the HoldDoor *Behavior* and the *Trigger Behavior* button is pressed.

The *Objects* section is shown in Figure A.8.



Agent Name: Alexander
Object Name: Farmhouse

Figure A.8: Object introduction controls.

This section uses classical conditioning to trigger responses to objects by an agent. The agent is chosen with the *Agent Name* menu, and the *Object Name* menu is automatically populated with any objects entered during event triggering. Selecting the *Introduce Object* button triggers any classically conditioned response the chosen agent has to the chosen object.

The trigger button for *Decay* is shown in figure A.9.

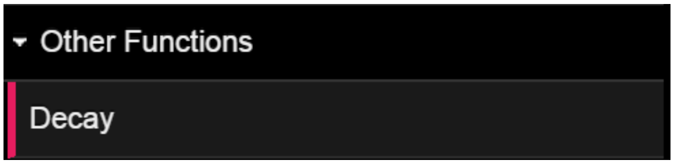
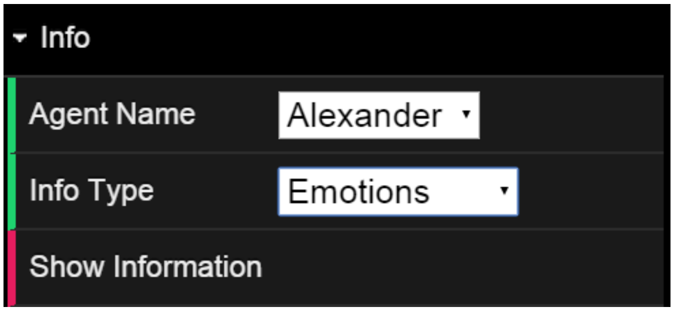


Figure A.9: Decay controls.

The *Decay* button decays all emotions by a constant proportion each time it is pressed. This lets the user simulate emotions decaying in intensity over time.

The *Info* section is shown in Figure A.10.

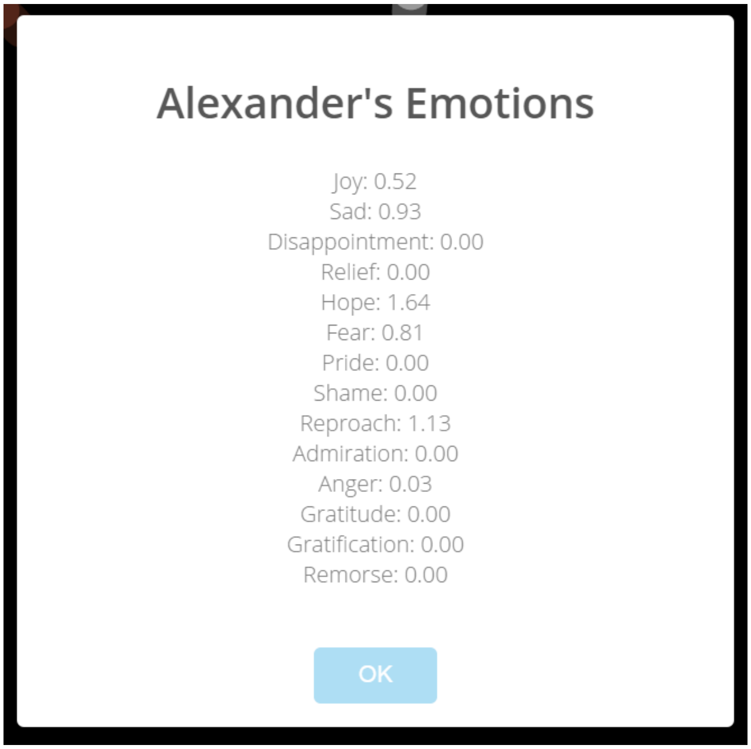


Agent Name Alexander

Info Type Emotions

Figure A.10: Information display controls.

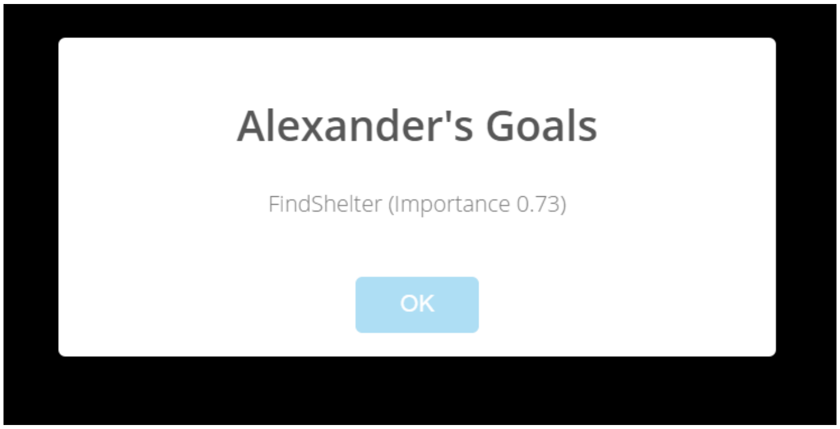
This section lets the user view different pieces of information about the internal states of agents. The *Agent Name* menu determines which agent's information is displayed, and the *Info Type* menu determines what type of information is shown. *Info Type* can take five types: Emotions, Goals, Events, Motivations and Associations. Figures A.11 through A.15 show examples of the information displayed for each type.



Alexander's Emotions

Joy: 0.52
Sad: 0.93
Disappointment: 0.00
Relief: 0.00
Hope: 1.64
Fear: 0.81
Pride: 0.00
Shame: 0.00
Reproach: 1.13
Admiration: 0.00
Anger: 0.03
Gratitude: 0.00
Gratification: 0.00
Remorse: 0.00

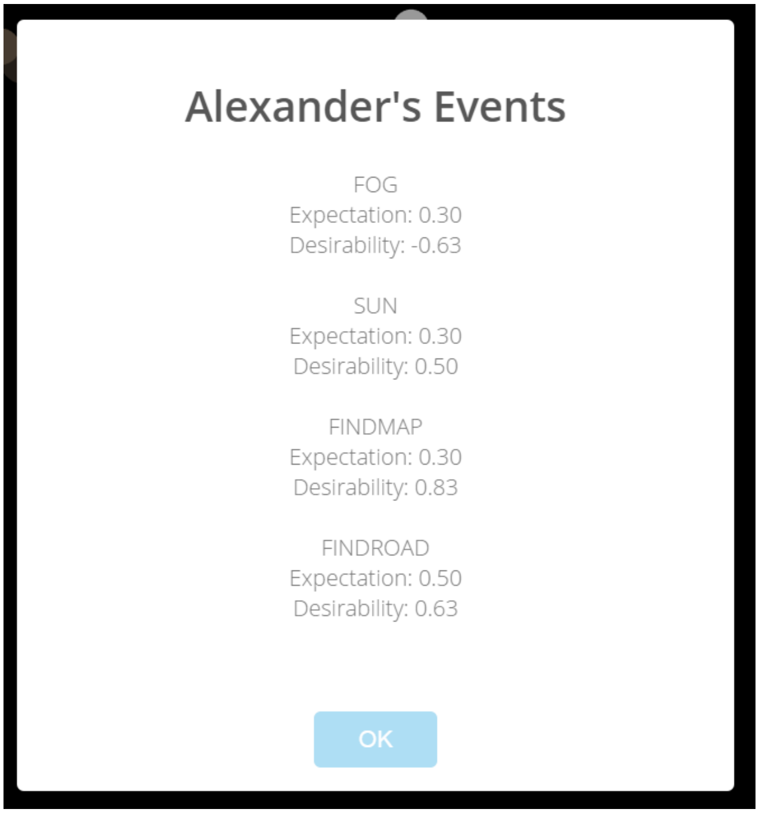
Figure A.11: "Emotions" information display.



Alexander's Goals

FindShelter (Importance 0.73)

Figure A.12: "Goals" information display



Alexander's Events

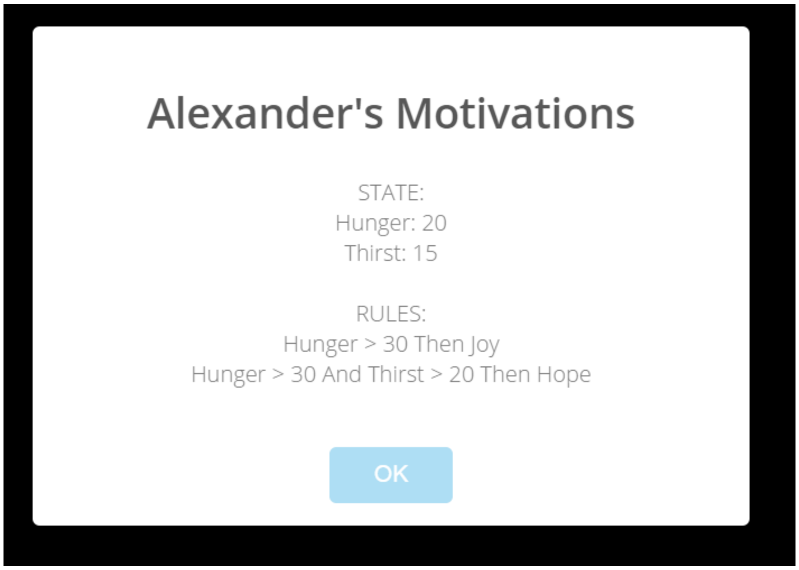
FOG
Expectation: 0.30
Desirability: -0.63

SUN
Expectation: 0.30
Desirability: 0.50

FINDMAP
Expectation: 0.30
Desirability: 0.83

FINDROAD
Expectation: 0.50
Desirability: 0.63

Figure A.13: "Events" information display

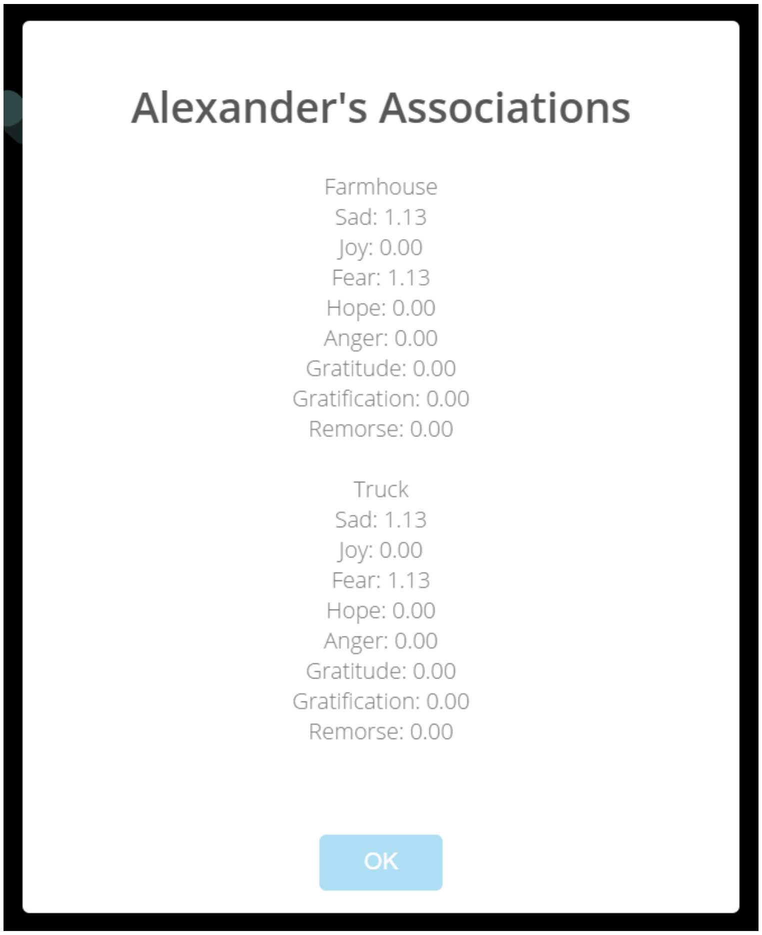


Alexander's Motivations

STATE:
Hunger: 20
Thirst: 15

RULES:
Hunger > 30 Then Joy
Hunger > 30 And Thirst > 20 Then Hope

Figure A.14: "Motivations" information display



Alexander's Associations

Farmhouse
Sad: 1.13
Joy: 0.00
Fear: 1.13
Hope: 0.00
Anger: 0.00
Gratitude: 0.00
Gratification: 0.00
Remorse: 0.00

Truck
Sad: 1.13
Joy: 0.00
Fear: 1.13
Hope: 0.00
Anger: 0.00
Gratitude: 0.00
Gratification: 0.00
Remorse: 0.00

Figure A.15: "Associations" information display

REFERENCES

1. Arbib, M.A. "The cognitive structure of emotions." *Artificial Intelligence* 54, 1-2: (1992) 229–240. 17, 18

2. Becker-Asano, Christian. *WASABI: Affect Simulation for Agents with Believable Interactivity*. IOS Press, 2008. x, 3, 21, 22, 23

3. Belohlavek, Radim, and George J. Klir. *Concepts and Fuzzy Logic*. MIT Press, 2011. 5, 6

4. Bolles, Robert C., and Michael S. Fanselow. "A perceptual-defensive-recuperative model of fear and pain." *Behavioral and Brain Sciences* 3, 02: (1980) 291–301. 32

5. El-Nasr, Magy Seif, John Yen, and Thomas R. Ioerger. "FLAME-Fuzzy Logic Adaptive Model of Emotions." *Autonomous Agents and Multi-Agent Systems* 3, 3: (2000) 219–257. viii, 3, 10, 15, 32, 33, 35, 36, 37, 38, 40, 41, 42, 44, 45, 46, 47, 51

6. Ginart, Antonio, Gustavo Sanchez, I. Links, and G. Back. "Fast defuzzification method based on centroid estimation." *Applied Modelling and Simulation* . viii, 54

7. Jamshidi, Mohammad, Nader Vadiee, and Timothy Ross. *Fuzzy Logic and Control: Software and Hardware Applications*. Pearson Education, 1993. 5

8. Marsella, Stacy C., and Jonathan Gratch. "EMA: A process model of appraisal dynamics." *Cognitive Systems Research* 10, 1: (2009) 70–90. viii, 3, 19, 21

9. Ortony, Andrew, Gerald L. Clore, and Allan Collins. *The Cognitive Structure of Emotions*. Cambridge England; New York: Cambridge University Press, 1990. viii, 3, 16, 18, 32, 34, 39

10. Picard, R. W. *Affective Computing*. 1995. 1

11. Popescu, A., J. Broekens, and M. van Someren. "GAMYGDALA: An Emotion Engine for Games." *IEEE Transactions on Affective Computing* 5, 1: (2014) 32–44. x, xi, 3, 25, 26, 27, 28, 29, 30, 31

12. Price, Donald D., James E. Barrell, and James J. Barrell. "A quantitative-experiential analysis of human emotions." *Motivation and Emotion* 9, 1: (1985) 19–38. 40

13. Roseman, Ira J., Martin S. Spindel, and Paul E. Jose. "Appraisals of emotion-eliciting events: Testing a theory of discrete emotions." *Journal of Personality and Social Psychology* 59, 5: (1990) 899–915. 32, 34

14. Staller, Alexander, and Paolo Petta. "Introducing emotions into the computational study of social norms: A first evaluation." *Journal of artificial societies and social simulation* 4, 1: (2001) U27–U60. viii, 3, 23, 24, 25

15. Watkins, Christopher J. C. H., and Peter Dayan. "Q-learning." *Machine Learning* 8, 3-4: (1992) 279–292. 11

16. Zadeh, L. A. "Fuzzy algorithms." *Information and Control* 12, 2: (1968) 94–102. 8