

Implementierung und empirische Evaluierung eines Emotionsmodelles in einer Spielumgebung

Diplomarbeit an der Universität Ulm
Fakultät für Informatik



vorgelegt von:
Alexander Festini-Mira
Matrikel-Nr.: 0403683

August 2007

1. Gutachter: Prof. Dr. Michael Weber
2. Gutachter: Dr. Henrik Kessler

Table of Contents

1 Introduction.....	1
2 The Why and What For.....	2
3 History of artificial emotion.....	5
3.1 Simon's interrupt system.....	5
3.2 Toda's Fungus Eater.....	6
3.3 Yesterday's models today.....	7
4 Used Tools.....	8
4.1 The OCC model of emotion.....	8
4.2 FFM, the Big Five.....	11
4.3 PAD-Space (Pleasure-Arousal-Dominance).....	14
4.4 Combining PAD and OCC.....	15
5 Existing models and implementations.....	17
5.1 Em, Oz.....	17
5.2 Artificial Emotion Engine.....	18
5.3 FLAME.....	20
5.4 ALMA.....	24
6 The emotion module.....	26
6.1 Characters.....	26
6.2 Interface.....	30
6.2.1 Goals.....	30
6.2.2 Events.....	31
6.2.3 Prospects.....	34
6.2.4 Retrieving Information.....	35
6.3 Update.....	36
6.4 Appraisal.....	38
6.5 Open Issues.....	42
6.5.1 Praiseworthiness.....	42

6.5.2 Long-term prospects.....	44
6.5.3 Memories.....	45
6.5.4 Interaction and inhibition between emotions.....	46
7 Scenarios.....	48
7.1 EmoSettlers.....	48
7.1.1 Resources.....	49
7.1.2 Buildings.....	50
7.1.3 Actions.....	51
7.1.4 Adding AI and AE.....	60
7.1.5 Handling events.....	67
7.1.6 Evaluation.....	69
7.1.7 Problems with EmoSettlers.....	69
7.2 SIMPLEX (Simulation of Players Emotional Experience).....	70
7.2.1 AI and AE strategies.....	71
7.2.2 Event evaluation.....	72
7.2.3 Evaluation and results.....	73
8 Conclusion.....	82
9 Downloads.....	83
References.....	84

Illustration Index

Illustration 1: OCC (from Bartneck [1]).....	9
Illustration 2: The emotion module.....	26
Illustration 3: UML: Character.....	27
Illustration 4: UML: Emotions.....	28
Illustration 5: UML: Events.....	32
Illustration 6: Emotional Settlers Dos-Client.....	48
Illustration 7: Emotional Settlers Gui-Client.....	49
Illustration 8: Shifting the resource window.....	50
Illustration 9: Build dialog.....	51
Illustration 10: Other's auction dialog.....	51
Illustration 11: Auction dialog.....	52
Illustration 12: Steal dialog.....	52
Illustration 13: Peek dialog.....	53
Illustration 14: Trade dialog.....	53
Illustration 15: Bail dialog.....	53
Illustration 16: Conspiracy dialog.....	54
Illustration 17: Buy dialog.....	54
Illustration 18: Internal state of a bot.....	55
Illustration 19: SIMPLEX.....	65
Illustration 20: Game duration.....	68
Illustration 21: Emotions triggered per move and player.....	69
Illustration 22: Single and joint attacks.....	69
Illustration 23: Deviation in emotions.....	70
Illustration 24: Emotions by ranking.....	71
Illustration 25: Emotions by personality.....	72

1 Introduction

Emotional agents are becoming a more and more relevant topic. Be it for games, interactive storytelling like *Façade* (<http://www.interactivestory.net>) or simply more life-like wizards and assistants to make the interaction feel more natural. Extending classic AI and logic by adding simulated emotions can be useful to improve the users experience in many ways. And while Hollywood has always had some trouble getting emotions out of its robots, actually getting emotions into them in real life has not been of too much interest until relatively recently. Yet, the need for emotional agents has been noticed over 45 years ago.

The goal will be to closer examine and analyze different existing solutions and models used for artificial emotions. Some comparisons will be drawn between early and recent models, but the main focus will be the suitability of different systems for generic use, especially in large scale scenarios like those found in many Massively Multiplayer Online games (MMO).

Eventually this will lead to the design and implementation of an emotion module that is supposed to be context independent, so it can be used for all kinds of applications and scenarios. For that reason, keeping the interface as small and simple as possible will be necessary, while at the same time, the module needs to be flexible enough. A compromise between simplicity, flexibility and required effort to use the module has to be found. Effort most of all meaning the work that is required to create an emotional agent. Complex scenarios can require large numbers of agents, so if each of them requires a lot of individual set up, it would severely limit the modules usefulness.

Also, issues like goals, emotions, mood, personality, memory and relationships between agents will have to be considered. These are the main factors next to pure logical reasoning that determine our behavior and they can often be even more important. All these factors need to be modeled and represented in such a way, that they can work together and interact at least similarly to the way they do in a real person. Though the benefits of in-depth and superficial simulation will have to be considered as well.

Finally, the resulting module should be able to create the illusion of emotional agents acting accordingly to their emotional state as well as analytical reasoning. To test this module, a simple game will be created in parallel that will later be extended by adding the module. Virtual players will react to game events by experiencing emotions and mood shifts, while also letting these factors influence or control their decision making.

Finally, the effects of this module will be evaluated to see if the resulting emotions and behavior are comprehensible and whether the module shows any promise or turns out to be too much work for too little gain.

2 The Why and What For

The question may be asked, why we would even want to add emotions to computers. It's not like emotions would make them more efficient in what they are doing now, or we wouldn't need them to do complex calculations for us. Now, of course the notion of actually having a whole emotional computer is not what this is about. Nobody wants a computer that might get distracted by a loud noise and stops encoding a video until it has found and classified the cause of the sound, deciding it is nothing to worry about. We will, however, find out why exactly this kind of inefficiency can be as vital to autonomous agents as it is to us. Nevertheless there are more applications that can profit from adding emotions. If autonomous robots need emotions to function in a wild environment then characters in a game would obviously benefit as well, seeing how many games are becoming more and more of a simulation rather than the abstract representations they have started out as. These virtual characters could then act scared or angry, because they actually (virtually) experience these emotions instead of simply being scripted to show these emotions in predefined situations. At first it might appear pointless to add this kind of complexity, only to achieve something that could be done a lot easier. In fact, for many simple scenarios it will probably remain less troublesome to create a simplified solution that's tailored to the task at hand.

But first off, let's consider an example for the natural evolution of a game resulting in the addition of simple emotions. A look at a typical situation in an usually rather simple genre of games, the ego shooter. The AI might be programmed for believable behavior, for example to look for cover and hide behind it when being fired upon. Now the result would be all opponents acting the same, but obviously this way is a lot easier than a complicated mechanic to create fear and letting the character act on this fear. But what if different degrees of an emotion should result in different behavior, ranging from simply going into a crouching position to diving behind the next available cover and not being able to do anything but trembling in fear? Now either different events would have to cause different reactions, or a variable has to be used to keep track of the characters fear, maybe even a finite state machine with states like *worried* or *panicking*. We just introduced the first emotion, though so far it is still all very easy. Different events would cause different degrees of fear and it would automatically decrease over time. Hearing a gunshot might make him worried, while being wounded could cause him to panic.

Yet even now, all the characters would still be acting exactly the same. Maybe we want the game to exhibit various reactions to the player firing a shot. Some should jump for cover while others don't even flinch and look around to figure out where it came from. So one might start to give each of them different degrees of fearfulness. Planned or not, now there are personality traits as well. Also, the question arises how exactly these things should interact. Should this trait scale or limit an emotion? And by how much? So far, there are no universally accepted and widely used rules and formulas, so it would be up to the developer to constantly test and tweak the numbers until the behavior is just right.

But even this simple game offers many situations for multiple emotions. Surprise over

the player suddenly rounding a corner, happiness after forcing him to flee, anger or sadness if he kills another bot, even gloating if his last grenade went wide. All these still in the context of a simple ego shooter. In fact, most of these emotions could even occur in a game like Space Invaders, one of the first arcade games where the player would try to shoot descending aliens before they reach the bottom. However, the most interesting use might be in role-playing games, a genre often filled with very generic characters that serve no actual purpose other than making the environment look less empty. Imagine how much these games would improve by actually making it interesting to just have a conversation and interact with these filler characters.

And even beyond gaming, there are a great number of useful applications of artificial emotion. Assistants, especially if they are able to recognize the users emotions, could react in a more appropriate fashion to the user. By displaying emotions, such an assistant could appear less like an interactive help system with pointless animations and might even cause less frustration for the user by being less of a stoic character. Think of Microsoft Office and the annoying paper clip assistant and how nice it would be to make it cry, weep and beg for forgiveness before eventually disabling it. Yet, it is an example where the range of emotions would have to be limited. Anger or impatience are not what a user wants to be confronted with when looking for help and asking the same question in different ways, for example because all the previous answers have been useless.

One usage that is probably a little more in the future could be automatically generated entertainment. With a look at *Façade*, an attempt to go beyond scripted dialog by using complex artificial intelligence to create an interactive drama, what if the user is replaced with another artificial intelligence and his interaction with a bit of random selection? How about watching randomly determined personalities reacting to different situations and one another? While such artificial characters are unlikely to perform Shakespearean stage plays, they could still be an useful source of inspiration and might still beat your average soap opera, not to mention invite you to just play with different settings and personalities to see how it affects the resulting story.

Finally, with a sufficiently refined model and implementation that is relatively accurate in simulating the behavior of a real person with a certain personality, artificial emotion could even become an useful tool in sociology or psychology. Take the Stanford Prison Experiment, where some test subjects played the role of prison guards while others took on the role of prisoners. The experiment had to be aborted after six days instead of lasting the intended two weeks, because of the psychological effects the situation had on the test persons. The controversy about whether or not this is acceptable or if the insights justify the risk of someone getting injured or even traumatized could be avoided, once purely virtual simulations are precise and life-like enough.

Patients with anti-social behavior could use virtual environments or games to be demonstrated the reactions to his behavior and especially how the reactions would differ if he behaves differently. Being able to experiment with virtual characters, such a patient might be much less inhibited compared to dealing with actual people. One reason is that real people might be encountered again, a thought that can be

uncomfortable in many situations, especially compared to a virtual person that will simply be reset afterwards. To some degree, this could be achieved by anonymously interacting with real people in the same virtual environment, though knowing that there is a real person behind the graphical avatar would still be expected to influence ones behavior. Of course, he could just be told that the other person isn't real, but this deception greatly relies on it either being actually possible or the patient believing it to be possible.

However, so far many existing solutions are tailored to the specific task and context, because it allows to consider and implement only whatever is actually necessary for the intended use. But even this can require considerable time and effort that could be saved by using an existing universal solution. Even if it might be more complex than is actually necessary, this would not become a problem, as long as the interface is kept small and simple. Just like using an existing graphics or physics engine, an existing emotion engine would allow concentrating on the important aspects of the project itself, rather than reinventing the wheel all over again. There have been quite a few recent projects to create such engines or at least develop a model that an engine could be based on. Some of them will be examined a little closer, as well as their potential use as a generic plug-in solution.

3 History of artificial emotion

A very extensive source for the subject of emotional computers and different models and approaches has been given by Gerd Ruebenstrunk [11] and is available online (<http://www.ruebenstrunk.de/emeocomp/content.HTM>). Only two of the first models will be described here, not just for historical reasons, but to allow a comparison with more recent attempts at modeling emotions.

3.1 Simon's interrupt system

The early 60s saw what might have been the first important debate about emotions and artificial intelligence. In 1963, Ulric Neisser criticized existing and planned systems like GPS (general problem solver), that were supposed to simulate human problem solving processes. In his eyes, they lacked several aspects that strongly influence human thought processes and are therefore required for a realistic simulation. The most important to us is that thinking and emotions are strongly connected, but he also criticized the single minded approach to work on only one problem, while humans are usually having more than one thing on their mind at any time. Another point was the lack of what could be put simply as *learning*.

In response, one of the developers of GPS, Herbert Simon, wrote an article in which he made emotions part of a model of cognitive processes for the first time. His intention was providing a theoretical foundation for a system incorporating emotions and multiple goals. It also allowed for important processes to interrupt and gain more attention to satisfy important needs. His idea was that of two parallel systems, one working on achieving goals and one observing the environment for events that require immediate attention. The second process in his design was supposed to be emotions. In his eyes, this design was necessary for all kinds of autonomous systems and not just living beings. In fact, this possibility of interrupting current cognitive processes is vital for survival, as it allows reacting to threats, but also paying more attention to the surrounding when a threat is expected.

This system would also have been able to learn, in that the reaction to these interrupts could change, either by becoming more familiar with certain stimuli causing them or by becoming more efficient in handling them.

Images might come to mind of this system failing. People crossing a street and being hit by a car they never noticed, animals falling prey to other animals while they are preoccupied with feeding. But that only shows how much this concept applies, as in both cases, had the threat been noticed, the resulting shock or fear would have caused an immediate reaction. Though at times the reaction is to just freeze and not being able to react, the common reaction to this fear would be a fight or flight response. Even how this is adjusted by learning is easy to see. For example, the animal might learn to better and quicker react, when noticing a predator sneaking up on it. And while hopefully

nobody will learn how to quickly dodge cars that are about to run him over, while mindlessly crossing a street, his learning might include paying more attention to cars in the first place.

3.2 Toda's Fungus Eater

Another theory was developed between 1961 and 1989 by Masanao Toda, a physics teacher turned professor for psychology. Toda's motivation wasn't frustration with or criticism of computational models, however. His frustration was with the methods of experimental psychology. And still, it resulted in the design of an autonomous robot and some important impulses for emotional computers. Even though it was never actually fully implemented, partial implementations do exist.

His point was that experimental psychology would provide questionable results that don't give any real answers about human behavior in a natural environment, but human behavior in laboratories and artificial surroundings. Interestingly, just like Neisser criticized this in programs, Toda also pointed out that these experiments usually require to single-mindedly solve a task at hand, while in natural environments they have to be dealing with multiple issues at once.

At first Toda only wanted to create a scenario that requires concentrating on multiple issues at the same time to create a more realistic situation for experiments. In this scenario, the task was to remote control a mining robot and collect as much ore as possible. At the same time, every operation of this robot required energy that could only be refreshed by collecting a special fungus, hence this robot was called a Fungus Eater. Operations included adjustments to the sensory equipment used to find ore and fungi, as well as movement. In this game, only the collected ore would be of interest, but obviously the fungi were just as important, as running out of energy would immediately end the game by making the robot immobile.

To make things more interesting, obstacles, lighting and other factors have to be considered, but most of all, the presence of other Fungus Eaters is radically changing the way this game needs to be played and significantly complicates the way a player would plan his moves. After all, suddenly the vital deposit of fungi might already be gone by the time the Fungus Eater gets there or passages might be temporarily blocked.

Expanding on this scenario, Toda came to the conclusion that, in order to survive on their own, these Fungus Eaters would require to have emotions, in fact that they need to be controlled by emotions even more than by pure intellect. This has been considered as very much consistent with the way human emotions were originally designed to help surviving in a wild environment.

However, Toda named them “urges” instead of emotions and on closer examination and comparison with newer models, it is apparent that these urges are playing the roles of needs and goals as well. He called them “motivational subroutines”, and while some of them are actual emotions like joy or anger, others are in fact much better described as

motivations. Different cognitions can trigger different urges, where all cognitions are rated in terms of how important they are for the survival of the Fungus Eater. This relevance to survival determines the intensity of the resulting urge. Once triggered, the subroutine representing this urge is called immediately, unless another urge with higher intensity was activated as well.

There are different sets of urges, biological urges, such as hunger, or cognitive urges, of which Toda only mentions the curiosity urge. Emergency urges are *startle*, *fear* and *anxiety* urges, which are arranged in a simple hierarchy. The *startle* urge stops all other activities and causes the Fungus Eater to identify the potential threat that caused the urge. If none is detected, the *anxiety* urge is triggered, else the *fear* urge is becoming active. Finally, there are social urges, which are organized in subcategories, helping urges, social system urges and status-related urges. The helping urges include *rescue*, *gratitude* and *love* urges. Urges like *protection*, *demonstration*, *joy*, *frustration*, *anger*, *grief*, *hiding* and *guilt* are making up the social system urges and the *confirmation* urge is part of the status-related urges.

3.3 Yesterday's models today

It is easy to see the parallels to Simon's interrupts in the way how urges can halt all other processing. But we will also see, that it has some similarity to the OCC model [10], where an event is appraised, and the relevance to a goal determines the intensity of a resulting emotion. In Toda's model, the goal was survival and the relevance to this goal determined the urges intensity. Many such parallels can be found in different and independently created models. But seeing how they all attempt to model emotions, this should not be surprising, but could be seen as consensus on many different aspects, even if different names are used or things like emotions, moods and goals are grouped in different ways.

These older models can easily be found to some degree in today's games, especially the constantly expanding Sims franchise. Here, simulated people have much the same urges as specified by Toda. Biological needs like eating and sleeping, social needs like interaction with others or simply entertainment. Such a Sim might be watching TV, but if he is getting hungry, he will interrupt this activity to get something to eat. Just like proposed about 40 years ago. Modern models might often be a lot more complex, but obviously even the first models of emotion are working well enough to sell countless add-ons and variations. Though newer versions are still extending the feature list and add for example user-defined personalities.

4 Used Tools

4.1 The OCC model of emotion

The 1988 OCC model by Orthony, Clore and Collins [10] is an appraisal-based approach to the creation of emotions and was explicitly developed with the use in computational models in mind. They wanted to offer a foundation that could be used for artificial emotion systems and succeeded in that it actually is at the core of many modern models and approaches to artificial emotions.

The idea is that every single emotion experienced by someone is triggered as a reaction to something. This can be an attribute of some object, like the shape or color of a car or the subjective beauty of a painting, a certain event or somebody's action. Each is evaluated based on certain kinds of knowledge and criteria, usually resulting in multiple emotions at different intensities.

For example, the appraisal of **attributes** requires the agent to have attitudes like taste or preferences to decide on the attractiveness of the object. Depending on the contexts complexity this can already require a large knowledge base. Consider a single simple object like a tennis ball and all its attributes that could have an effect on how appealing it is to someone. Maybe an agent simply likes the color, finds round shapes particularly aesthetic or likes the pattern? He might also be a great fan of the sport and like the tennis ball less because of the balls innate qualities and more because of the association. Now imagine a virtual world that is full of different objects and not just all the information that needs to be attached to each object, but all the defined preferences for each agent. Obviously, filling a typical modern game with all this data is not feasible.

The appraisal of attributes determines how much an object is liked or disliked by the agent and results in either the emotion of love or hate. While usually emotions are considered as temporary reactions that disappear relatively soon, it is questionable if love and hate should really be considered emotions when building an artificial emotion system on OCC. On the other hand one could argue that these aren't constantly experienced emotions and in the context of OCC one has to only consider the times where they are consciously experienced by the agent. To help understand the difference, don't think of love as a description of how much sympathy you have for someone, but the emotion when you feel the urge to actually express it.

Events, or rather **consequences of events**, are appraised by analyzing their impact on the agents goals. This determines the events desirability. The degree of desirability depends on how the goal is affected and how much closer to or further away from achieving the goal the agent will be afterwards. Another factor is the goals relevance. A moderately negative impact on an unimportant goal will hardly be judged overly undesirable, while even a small impact on a highly important goal might cause much stronger reactions. The emotions joy and distress are direct results of desirable and undesirable events, considering the consequences they have for the agent himself..

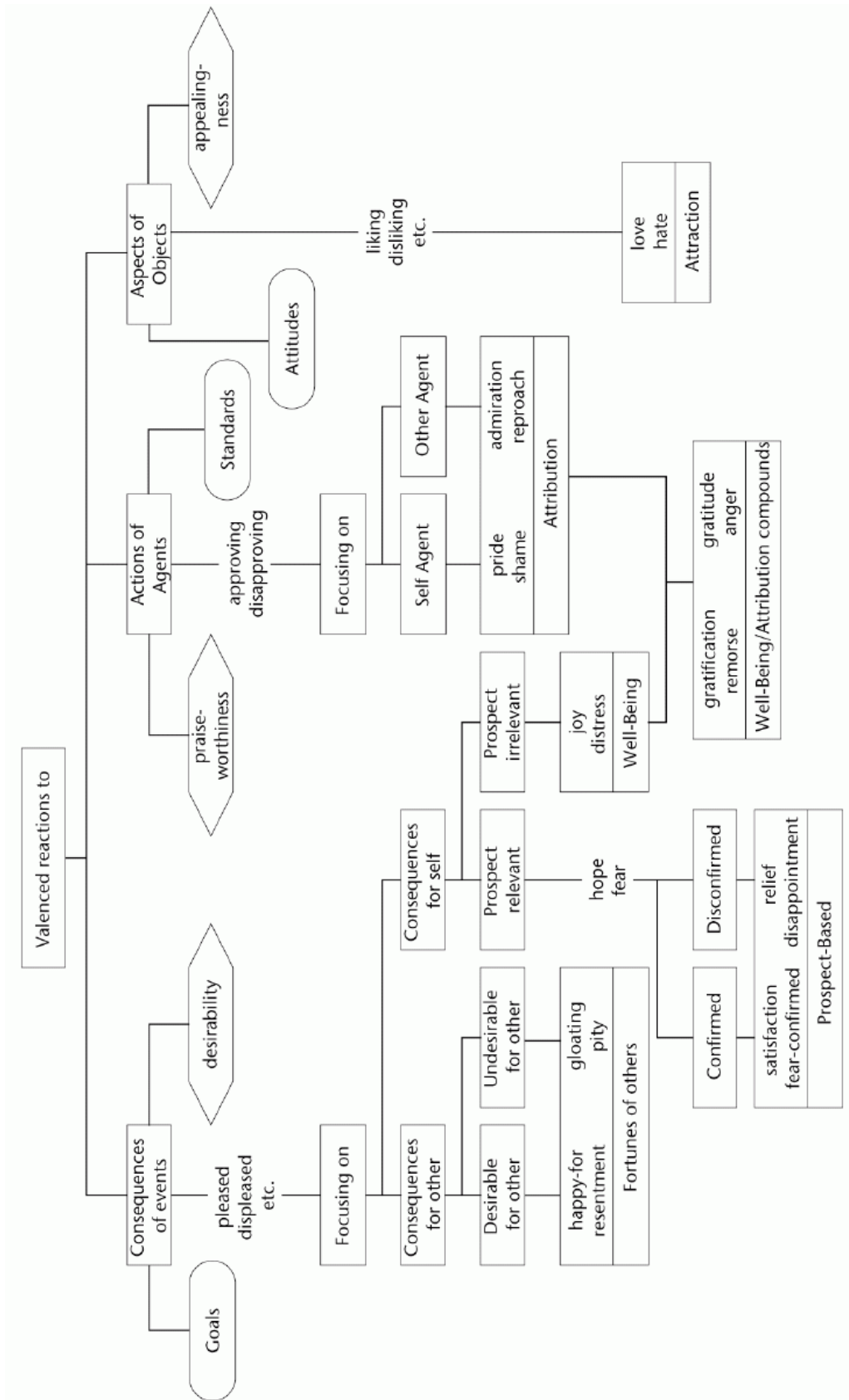


Illustration 1: OCC (from Bartneck [1])

But this evaluation is not limited to consequences that only concern the agents own goals, but also the **fortunes of others**. Emotions like pity require processing events, even if it has nothing to do with the agent and his goals. Unfortunately this is posing the first problem, because it's not always obvious if the agents own goal priorities should be used. While it would seem perfectly natural to judge the consequences based on your own views, it neglects the possibility for an agent to be aware of those of the other agent. Consider a friend collecting stamps and losing a very rare one. You yourself might not even have a goal related to stamps, but would still experience pity, as you know how important it is to him. At the same time you wouldn't rate the loss as drastic as he himself would, particularly if you have a hard time understanding his obsession and hence empathizing with him. This can be somewhat alleviated by thinking in abstract goals like *not losing precious property*. For the same reason as with appraising objects attributes, it would even appear very important to keep the goals general and abstract, to avoid ending up with a too many and too specific goals, especially as for every event it has to be determined which goals it actually affects.

The resulting emotions to other agents fortunes also depend on how well-liked they are. A bad thing happening to another agent can result in pity or gloating, while a good consequence results in either being happy for him or feeling resentment, depending on the relationship between them. Sympathy is another thing that is considered to be a given, so implementations using OCC need to develop their own way to measure this. By looking at other agents as objects, the emotions love and hate could be used.

Another issue when appraising events are **prospects**, essentially hope or fear that something does or does not occur. These are necessary for the prospect-based emotions like disappointment or relief. The intensity of these emotions is usually based on the intensity of the preceding hope or fear, so the stronger the hope for a certain event, the bigger the disappointment when it doesn't happen.

Note that the prospects themselves also have an intensity and can technically be considered emotions, even if they don't appear as such in the diagram for the OCC model. The prospects intensities depend on the events perceived probability and its expected consequences. As the model considers all emotions as reactions, the non-occurrence of an event also needs to be treated as an event, for example as the expected event but with different consequences. One way to create hope and fear is by appraising potential or hypothetical events. Instead of joy or distress, hope or fear would be caused, though in addition to desirability the events probability is used.

The criteria to appraise **actions** is **praiseworthiness**. Generally, praiseworthy actions cause pride and blameworthy actions cause shame, if the agent himself is the one acting. For other agents, the emotions are admiration or reproach. Unfortunately, the standards it is based on, can be as complex as the attitudes used for attributes of objects and are almost as subjective and individual. So an agent would need to have an opinion about every possible action or the description of actions would need to be sufficiently abstract. To make matters worse, OCC mentions many factors that can influence the final praiseworthiness and the resulting emotions intensity.

Responsibility is one of the most important factors, even the first reaction might be that everybody is always responsible for his own actions. However, in a situation where one is being forced or doesn't have any choice, neither is a bad deed as bad or a good deed as good as if it would have been committed purely out of free will. You wouldn't blame someone as much for kicking a puppy, if he was forced at gun point, compared to someone doing it for fun. In the same way, the rules of a game might force you to remove one of two players from play, so while you might blame him for his choice, you couldn't blame him for making his move.

Another factor is conformity with **role expectations**. For example, fire fighters are expected to rescue people from burning buildings, but small children aren't. Of course that doesn't make it completely non-praiseworthy for the former, but it will usually be perceived less so, as he is "just doing his job", while the child will receive a lot more praise than an average rescuer would have.

Cost is one more influence on praiseworthiness. Doing a good thing and doing so despite having a high price to pay for it would rate very differently. In the same way a positive action for selfish reasons would be considered less praiseworthy. Hence it isn't uncommon that a celebrity donating money is accused of doing it for publicity, especially if the amount was relatively small compared to their assumed wealth. Yet, it should still be handled in a way that allows for pride or admiration, even if nobody but the agent profits from it. A self-made millionaire would still experience pride, though he might be the only one benefiting from his money.

Finally, the intensity of pride or shame can depend on **public awareness**. Typically, these emotions will be experienced stronger if a large number of people is aware of what the agent has done. At the same time, pride often leads to the desire of more people learning about it, while shame can cause fear of others finding out.

An interesting phenomenon is the ability of feeling proud or ashamed of someone else's actions. Simply put, the closer an agent feels related to the acting agent(s), the more he will identify with him for the purpose of action appraisal. This is called the **strength of the cognitive unit** and can range from parents being proud of their child to a soccer fan being ashamed for his teams performance.

4.2 FFM, the Big Five

What OCC is not covering, is the influence of personality. As essentially personality can be considered where different individuals differ from each other, it is obvious that some model of personality is needed. Without it, all agents would react exactly the same. OCC already offers a few possibilities to add diversity. Different goals, standards and attitudes will automatically result in differences during the appraisal process and either emotions of varying intensities or even opposite emotions. However, in psychology personality is not just a long list of preferences, so on top of standards and attitudes being hardly practical to implement, they are also not an useful description. Also, personality can have more effects on our behavior, not all of them sufficiently

covered by the above appraisal. Moodiness for example couldn't be represented, as OCC doesn't deal with moods.

What needs to be found is a model that allows deriving rules to adjust the appraisal process and if possible to help with the determination of standards. It has to be able to shift the agents perception and even make it possible for two agents to react in almost opposite ways to the same event. Of two agents, one might gladly accept help while the other reacts offended. Of course, this model should also be as simple as possible and still cover the whole range of possible personalities.

The model of choice was the Five Factor Model [6]. This model using five different factors to describe personalities has emerged in the 1980ies, though one could say it has been in the making for much longer. For a long time, hundreds of possible traits have been tossed around and were attempted to be grouped and ordered in a meaningful fashion. The fact that often different names would be used for what was essentially the same thing didn't help to make it any less confusing. Eventually, two approaches led to the five dimensions that are known and often used today, also known as the Five Factor Model or FFM.

One of them is based on the so-called Lexical Hypothesis, the idea that all the relevant and important personality traits are so omnipresent that they would eventually find their way into the different languages. As a consequence, one should be able to identify all these traits by examining any language, for example by looking at a dictionary, as Allport and Odbert have done in 1936.

However, as there are still debates about whether some are redundant or overlapping, if the factors are the correct ones or if the model is missing important factors, the origin of this theory deserves a quick recapitulation. Based on the English language, they were able to find a not so manageable number of 18000 traits, which were eventually reduced to 4500. Taking a personality test to be rated in that many traits still doesn't seem very appealing. In 1957, Cattell further reduced this number to 171 by removing all synonyms and was able to group them in 35 different clusters. Through a number of personality tests and factor analysis, he could finally identify 16 major traits. In 1961, Tupes and Christal performed more tests and found that there were 5 prominent recurring factors. In 1963, Norman confirmed that these 5 factors were sufficient to describe a large set of collected personality data. He called them Surgency, Agreeableness, Conscientiousness, Emotional Stability, and Culture.

It is worth pointing out that this approach completely based on language, not any actual psychological theory. Hence, arguments for the correctness of this selection often include that these traits exist in many different languages like English and Chinese and are rated and grouped in more or less the same way.

The other approach is based on questionnaires. These have been extremely different, using different scales depending on what they were used for and what they were supposed to measure. Well, personality, of course, but remembering the 18000 traits originally found in a dictionary, it is no surprise that there would be significant

differences. Except for two recurring measures, regularly experiencing negative emotions (neuroticism) and interaction with others. It was obvious, that there had to be more than these two, if all types of personalities were to be covered. The approach was similar, looking at the traits in different questionnaires and trying to group them together. Independently, Tellegen and Atkinson as well as Costa and McCrae suggested different, yet similar third dimensions. *Openness to Absorbing and Self-Altering Experience and Openness to Experience*. Later they again brought up similar dimensions of *constraint* and *self-control*, though by this time the two different approaches had already merged and resulted in the FFM as it is used today.

Looking at the FFM or *Big Five* today, the way they are commonly encountered is very much the same as the first approach proposed. Agreeableness and Conscientiousness have been taken over from the first approach, Surgency and Emotional Stability have been replaced by the Extroversion and Neuroticism as introduced by the second approach, where Neuroticism is essentially just the inverse of Emotional Stability. Culture has disappeared and Openness to Experience is taking its place, but even here a vague resemblance can be seen, as open personalities are often interested in arts.

Note that though sometimes these factor might be called traits, they are actually groups of different traits. This can make it difficult to give a clear-cut definition for each factor, especially since rating high in one of these factors doesn't necessarily mean that you will also display all of the traits associated with this factor. As a result, they are often described in terms of *tend to be like this* or *often display this or that trait*. Additionally, there have been many debates about how appropriately a factor has been named, if a trait should really be connected with a certain factor and even if five factors aren't too few to sufficiently describe all possible personalities or too many. For example, during the 80ies it has been proposed that three factors would already be enough and Agreeableness and Conscientious could be replaced by Psychoticism. Some even argued for only two factors. Yet, in 1989 and 1990, several independent analyses showed that only a number of five factors would work out. This seems to have ended the majority of ongoing debates and challenges of the FFM.

For these reasons the following description of the factors should not be considered as even remotely complete or all encompassing. These are just the most important traits that were also kept in mind during the design of our module.

Agreeableness refers to a tendency to cooperate and compromise in order to get along with others. High agreeableness often means a positive outlook on human nature, assuming people to be good rather than bad. Low agreeableness essentially translates to selfishness, putting your own needs above everyone else and not caring about the consequences your actions might have for others.

Conscientiousness is usually high for people that plan a lot, think everything through, are very tidy or achievers. Extreme cases can appear compulsive or pedantic. The opposite is sloppiness or ignoring your duties.

Extroversion can be a measure to how much people experience positive emotions. High values suggest an enthusiastic and active person that enjoys company and

attention, while a low score means a quiet, introverted individual with a higher need to spend time alone, though this does not mean he has to be shy.

Neuroticism is partially an opposite of Extroversion in being a tendency to experience negative emotions. However, it can also mean a higher sensitivity in general and emotionally reacting to small events that usually wouldn't trigger a response. They can be prone to mood swings and tend to be more negative in their interpretation of a situation. Low numbers mean high emotional stability and describe calm people that aren't easily upset.

Finally, those scoring high on Openness to Experience are often creative individualists full of curiosity, interested in art and more in touch with their own emotions. The opposite are conservative persons with a small number of interests, preferring straight and simple over fancy and not caring about art or science. It's suspected that Openness can be influenced by education.

4.3 PAD-Space (Pleasure-Arousal-Dominance)

One commonly encountered model are the six basic emotions anger, disgust, fear, joy, sadness, surprise. All possible emotions are considered to be a mixture of these basic emotions. They are often found in combination with facial animations and many software packages for character modeling support them out of the box. However, not everybody agrees with this concept.

Wilhelm Wundt for example rejected the notion of elemental emotions and in 1909 introduced the idea of a *Gesamtgefühl* [2], which is also the result of different emotions. While this would at first appear to be exactly the same with a fancy name tacked on, he didn't limit the emotions that contribute to the *Gesamtgefühl* to any number or even any time. So even emotions that were experienced over a certain time could still affect the current total emotion. To describe these emotions, he introduced three dimensions to describe them. Pleasure, Arousal and Tension. In this space, emotions, or rather the whole experience of an emotion, is described as a curve, typically starting and ending in the origin. It should be noted that he arrived at these results through introspective psychology, meaning that they are based on very subjective self observations.

Almost 70 years later, Albert Mehrabian presented his own space to describe emotions, as well as moods and even personality traits. Surprisingly, this space is using the dimensions Pleasure, Arousal and Dominance (PAD [9]). Even though he arrived at these three factors from a different angle and never mentioned Wundt as a reason for his selection, two of them coincide with the dimensions of Wundt. Yet, there are important differences, most of all, he describes emotions as points in this space, not as a curve progression that includes how the emotion is coming and going. To him, an emotion is just a state.

The advantage of treating emotion as one continuous space and not just a set of a fixed number of possible emotions, is that it is not just easier to use, but also allows to

represent emotions that might not even have a name. Although the same questions that have been discussed for a long time regarding the FFM need to be considered again. Are three dimensions too few to cover all emotional states or already more than necessary? Are these three a good choice? Another model is using only arousal and valence. While this would be even simpler to handle, there are reasons why PAD has been used.

What makes PAD highly useful is that it can combine emotions, mood and personality in one common space, when otherwise they would exist in complete isolation, despite being closely related. That means single emotions can be described in terms of Pleasure, Arousal and Dominance values as well as whole personalities. This also means that for example Arousal cannot just be the degree of arousal associated with a specific emotion, but also the arousability of a person.

While Pleasure and Arousal are rather self explanatory in both models, Dominance in regard to an emotion is a measure of experienced control of or influence on the situation. This can for example make the difference between fear and anger. Both are states of negative pleasure and high arousal, but not feeling in control is what separates fear from anger, where the agent would at least believe to have potential influence.

Mehrabian also gives specific meanings to the different octants in PAD-space and describes the diagonally opposite octants as Exuberant/Bored, Dependent/Disdainful, Relaxed/Anxious, Docile/Hostile. So, mood is not so much a position in PAD-space, but a whole octant.

4.4 Combining PAD and OCC

The difficulty here is that these models have nothing in common, other than being used to represent emotions, one in a numerical fashion, the other in a purely semantic way. As there is no mathematically correct way to convert between them or decide where a certain emotion should be located in PAD-space, this decision is rather arbitrary and the position of for example *pity* is the averaged gut feeling of several test persons, who have been asked where they would place certain emotions. Fortunately, Mehrabian has already provided a conversion from FFM to PAD [7] and a mapping from OCC to PAD has also been specified [4], so all the necessary tools already exist.

There is also another issue. PAD is continuous and OCC emotions are discreet with an additional intensity value. Taking a look at the conversation of anger from OCC to PAD, it is placed at -0.51 pleasure, 0.59 arousal and 0.25 dominance. It would appear natural that multiplying these numbers with 2 results in the agent being twice as angry and looking at PAD on its own, nothing would speak against that at first. In fact, the question is where the test subjects would have placed *a little bit angry* or *irritated* as a weaker form of anger. They might just use a lower value for arousal or pleasure, but not for dominance, so PAD might not be as linear and easy to use as it seems.

As a result, the intensity variable used in OCC cannot simply disappear behind scaled PAD values and specific emotions remain positions in space, rather than vectors with

Combining PAD and OCC

their length equaling their intensities. This feels very unintuitive and often confusing in combination with how mood is used in the module.

5 Existing models and implementations

Several existing models to create artificial emotions were examined to research different approaches and methods to look for a starting point and get an overview of aspects to consider for our own model and implementation.

5.1 Em, Oz

Bates and Reilly created a model for the Oz project, a system to create interactive drama by using virtual agents as actors. These agents were supposed to have a wide range of capabilities, while the extent and detail of a single capability wasn't too important. A decision based on their view that a believable agent has to react to many different situations and it is therefore more important to be able to react at all instead of being limited to only a small set of highly complex reactions. However, only their emotion module called Em is of interest. Note that the goal in this model was not to be psychologically correct, but to create believable agents.

Their model of emotion is based on OCC, yet the rules to generate different emotions have been changed in a way that simplifies the process without limiting the possibilities. Some already mentioned issues are nicely solved. Remembering the question, whether consequences for others should be appraised based on the agents own goals or the others goals, here the solutions is quite simple. The emotions in the fortunes-of-others group are based on the other agents emotions instead. Pity is not the result of appraising the consequences of an event for some other liked agent, but the result of this agent being sad. That means, if the other agent suffers a bad consequence that isn't sufficient to cause distress, no pity will be experienced, while if an agent displays distress, it doesn't matter what the cause of this emotion might be. Interestingly, this would also appear to be coming closer to reality. Why feel pity for a friend that was left by his partner, if he himself doesn't care? And does it really matter why a friend is feeling bad? Would one not feel pity for him, just because the cause is unknown? The actual reason for this was that it might be difficult for one agent to know another agent's goals, since these can change at any time, but this new approach feels much easier and more feasible. In fact, the same appraisal process that causes distress in the other agent would cause pity for our agent, given that there is positive sympathy. So instead basing pity on his distress is a simplification that, depending on the exact implementation of course, might not even affect the final result.

Goals are also more important than events in this approach and so the trigger for emotions changed mostly to goals being achieved or becoming likely to be achieved. Technically, this doesn't appear to make much difference, as an event's desirability is most likely based on the same criteria, however, it is also a more direct approach and doesn't require an explicit event or action to take place. Other than that, two emotions were added. Frustration is experienced when a plan of the agent fails and he can be startled as a reaction to a loud noise.

5.2 Artificial Emotion Engine

One of the first implementations to look at was the Artificial Emotion Engine, which is further described in IWilson00 [13] by Ian Wilson. It is no surprise, that it works on the expected three layers, emotions, mood and personality. There is a clear definition for which of these is determining the characters action. If there is an emotion, the actions will be based on it. Without an emotion, the engine falls back to using the mood and lacking even that, the personality is used.

The Emotion Engine (EE) uses a different and more straightforward model of personality. Unlike the very common Big 5, EFA is a three-dimensional space, describing personality traits in terms of Extroversion, Fear and Aggression. In this, EFA is not unlike PAD when used to describe personalities. Here, an area around the point representing the personality is used and all traits located inside this area are considered the traits that are available to the specific character. It is interesting to notice that the EE paper lists anxiety as an example for a personality trait, while Mehrabian uses “anxious” as one of the mood octants.

Wilson explains the use of EFA in its correspondence with the Approach system, the Behavior Inhibition system and the Fight/Flight system, which “may” be the three central systems in the human brain that determine our behavior. While there seems to be a lot of speculation and one might question the use of EFA for psychological use, it would appear simple and useful enough for usage in a computational model of emotions. Maybe even more so, as its three dimensions are very basic attributes that are easier to grasp by intuition, making it easier for a programmer to work with it.

The paper also describes positive and negative moods and refers to mood types and levels in a diagram. However no further explanation is given, leading to the assumption that mood is either reduced to a matter of degrees between “good” and “bad” or that there several distinct kinds of mood like maybe “sad”. What is explained is the influence of personality, where Extroversion determines the maximum for good moods, Fear for negative moods and Aggression affects the speed of mood changes. As the rest of the paper suggests mood to be in deed just a single value, personality would essentially simply determine the mood limits and moodiness of a character. At first it seems strange than introverted people obviously aren't supposed to be in a really good mood, however, regarding this as the mood that is actually displayed on the outside, it does make sense, as an introverted person would not openly show it as much as an extroverted person. As the goal of this engine is to control behavior and it is usually not of that much interest how someone feels inside compared to what can be seen, this seemingly simplistic approach might work very well.

As input, EE tries to keep it in terms of reward and punishment wherever possible. This can easily be compared to the desirability of an event in OCC. This input is adjusted based on personality, but also by how often it occurred before. The agent can get used to a certain input, lowering the impact it will eventually have, which Wilson calls habituation. Novelty is basically the opposite of this, meaning a rare or unprecedented input.

The hierarchy of needs is of more interest, as they can be considered to be nothing else but basic goals. Only physiological needs are explained in depth and consist of hunger, thirst, and the need for warmth and energy. In which way energy differs from the need to eat is somewhat unclear. The priority for each of these needs depends on how far they are from being fulfilled, meaning that a very hungry agent will automatically consider this as more important goal. Each of these goals can be overachieved, simply imagine eating too much for example. Safety, affiliation and esteem needs are the remaining layers. While physiological needs are almost the most important, the order of the other layers can vary, depending on what is more important to someone. Think of the achiever that puts his job first and neglects his family in favor of a career, while someone else values friends and family above all else.

As it was hinted at before, there are several personality traits known to the engine and rated by social desirability. It would appear that mood determines which of these traits will be used, essentially indicating that an agent in a bad mood will tend to give into traits that are less socially accepted.

Memory is described as very constrained at the time the paper was written, limiting itself to storing how much others are liked. Again, this can be found in OCC as well, where sympathy is used to cause different emotions for liked and disliked entities.

Emotions are limited to the six basic emotions fear, anger, joy, sadness, disgust and surprise. This might appear like a limited selection compared to the 24 emotions of OCC, but as all emotions are considered mixtures of these basic emotions and most facial animation packages are based on these, this is quite a sensible choice. Personality is again used to adjust these emotions, so a character with personality that is low in Fear will obviously not experience as much fear as others.

The Emotion Engine is very much geared towards displaying emotions through gestures and facial expressions, outputting data that can be used for skeletal animations and supporting MPEG4 Facial Action Parameters. It can also create simple plans, however this should be confused with actual planning systems. Instead, the plans appear to be limited to the likes of *work on need X*, the needs priority and a personality trait picked according to the current mood. One could argue if this deserves being called an *action plan*, seeing how it consists of only a goal with no actual plan on how to achieve this goal other than *very anxiously*. Access to the raw emotion data would seem to be the most useful way to use the engine when it is not used for animation purposes.

According to the engine's website, it was supposed to be available as SDK and the company offering support and services. However, no more news have appeared on the site since January 2005, so the current state of the engine is unknown. In recapitulation, the engine looks like a simple but robust approach with more emphasis on the visual aspects of emotions than a complex and perfectly accurate simulation.

5.3 FLAME

Fuzzy Logic Adaptive Model of Emotion (FLAME [12]) is partially based on OCC, but what separates FLAME from other models is the use of fuzzy logic. This results in a much simpler appraisal process than working with actual values. For example, in OCC an event's desirability is based on how it affects a goal and the goal's importance. Instead of dealing with numbers and requiring formulas that will produce the wanted output for all possible numbers, FLAME uses only three categories for goal priority and five for an event's impact. The resulting desirability is then also one of five possible values. This smaller number of combinations allows to set down rules about which combination should result in which desirability, making the whole process a lot easier and more stable.

It also allows completely arbitrary mapping. Consider prospects and the way an event's probability affects hope. The idea is that the more likely the event is, the more hope is generated, but at the same time if the event is so likely that it is almost certain, there isn't really much room for hope. Of course this particular case could be handled by defining the emotion of hope as not just hope, but also anticipation. But there are other cases, where this can't be ignored and a numerical approach would have to use different formulas for different values and combinations. The approach of FLAME avoids this pitfall by reducing the number of possible values and combinations and having separate rules in the first place.

While this kind of categorization could of course be done without fuzzy logic as well, by simply declaring the range [0.6; 1.0] for an event's consequence as "highly positive" and a goal's priority within [0.666; 1.0] as *very important*, these sudden changes in category are usually not wanted. Imagine the impact as 1.0, but the priority as 0.66 (just shy of the boundary to be *very important*). The approach without fuzzy logic wouldn't care about this and treat it the same as if the priority was 0.34 and maybe result in the event being *slightly desirable*. However, the fact that the consequence is so far on the upper end of the scale might make it a lot more appropriate to declare the event as *highly desirable* instead. This is where fuzzy logic comes into play and proves to be a viable and preferable solution.

One aspect that turned out to be quite important during the testing of our emotion module is the interaction between emotions. FLAME is calling this emotional filtering and doesn't limit it to emotions, but considers motivational states as well. This concept is easiest explained with examples. Say an agent is usually so afraid of being caught, that he would never think about stealing, but at some point he might be unable to obtain food and get so hungry that the need to eat overcomes his fear. That means the right motivation can allow an agent to ignore or suppress an emotion in favor of reaching a goal. As an example how emotions can inhibit each other, imagine you are applying for one of two job openings. You know that about a hundred others are applying for the same job, yet you succeed in getting one of them. Obviously you would feel joy and maybe even pride. But on your way out, you learn that the other job was given to the department managers' unqualified brother. Suddenly you might feel anger about having had to work so hard to get the job, while someone else is unfairly preferred by virtue of

being related to the boss. At this point, your anger could prevent you from feeling any joy about getting the job. In a way, one might ask if the previous emotion of pride would not just be inhibited, but could also actually increase the intensity of your anger. FLAME solves the occurrence of opposing emotions by having the stronger emotion inhibit the weaker emotion(s) with a slight preference of negative emotions. However, particularly the last part might be solved better by considering the agents personality.

Another obvious way to handle conflicting emotions is mood. This is also done by FLAME. Mood seems to be a simple matter of being either positive or negative, which is determined by comparing the intensities of positive and negative emotions over the last few time steps. If the summed up intensities of positive emotions are higher then the mood is positive as well. Now, if a positive and negative emotion occur with roughly the same intensity, the mood determines which of these emotions will inhibit the other emotion.

As there appears to be little research about decay of emotion, FLAME uses a simple constant decay, though positive emotions are decayed faster than negative emotions. Again, this is something that might better be based on the agents personality instead, but FLAME is not implementing a personality as such, instead differences in behavior are created through learning. Ignoring genetic influences on personality, learning would seem to be a relatively sufficient replacement on top of being an important factor in many other ways. An agent could learn that reacting angry will give him what he wants, teaching him to act more choleric. However, to function as a real personality it would still have to be included in the appraisal process and at least influence the display of emotions. Still, certain personality traits will more often cause negative than positive results, so it is unlikely an agent would ever take on a disadvantageous personality.

FLAME implements multiple kinds of learning. Classical conditioning is one of them and already covers many situations, where fear or hope needs to be created. It should be noted, that conditioning is associating expectations with objects, not events. The latter is handled by another form of learning. This is also a simple form of memory, where the agent remembers previously experienced emotions. A dog will quickly learn that having his feeding dish put in front of him means getting food, which might result in joy over achieving the goal of being well fed. After a while, the dog will associate a certain intensity of joy with the dish. As a result, seeing the dish will cause hope or even joy, depending on the associated emotion. The expected emotion intensity is simply the average over all events involving the object. So placing the dish again and again without actually putting food into it, will cause the dog to associate less joy with the dish.

A much more interesting issue is learning about consequences of actions or events. This is simple whenever an action directly causes a result. For example, learning that eating will result in feeling less hungry is rather trivial. But consider an experiment where a test subject presses a button and a little later somebody will bring a plate of food. It is quite possible that these events aren't even connected and the timing is pure coincidence. But it also possible that pressing the button starts a chain of events that eventually did cause the food to be delivered. The task of detecting complex causalities can be rather complicated.

FLAME is using Q-learning, a form of reinforcement learning, to solve this. The agent is keeping track of states, actions taken when being in a state and the results of the resulting state transition. This way, the agent collects information about which action can be taken in a given state and what other state this might lead to, as well as any reward resulting from it. Q-values are calculated for each pair of state and action, where the reward and the Q-values of the following states determine the Q-value of the current state. As the information is constantly updated, after enough repetition all coincidence should be filtered out and the agent can make predictions about long term results for a possible action. This is getting a bit more complicated for FLAME, as usually the agent and the user take turns in their actions, making it harder to make long term predictions. So in addition, the agent has to keep track of the users action to determine the likelihood of a specific action by the user in a given state. Mood is also used to modify the expectations, so in a negative mood, the agent will be more pessimistic and treat the actions that lead to negative results as more likely, while a good mood does the opposite.

Another form of learning is recognizing patterns in the users behavior by observing sequences of actions or events. FLAME is only considering sequences of a length of three. For this kind of learning, it simply counts the occurrences of sequences. So if the events A, B and C occur in this order, an entry is created with a count of one. To make predictions, after the events A and B have occurred, the agent simply compares the number of times that the sequence A,B,C was observed with the number of times that A,B,x was observed, where x is any event including C. So, after the first time someone leaves the house, gets into his car and drives the away, if the person does the first two steps, the agent will predict that he will drive away with a likelihood of 100%. If the person changes his routine and first turns on the radio after getting in the car, this prediction will be adjusted, however he will now expect that the sequence of getting into the car and turning on the radio will be continued by driving away. This could easily be applied to other agents as well, however FLAME seems to only consider a scenario with one agent and one user, typically acting alternately.

The last, but also one of the most important kinds of learning in FLAME is learning about the value of actions. Remember that OCC relies on an action's praiseworthiness, which is based on the agents standards. Instead of expecting these standards as predefined knowledge, they are learned from interaction with the user or other agents. The main reason for needing this, is that so far the agent would act completely in his own interest and only pick actions by the desirability of their consequences for himself. But social behavior requires considering the consequences for others as well.

FLAME uses a simple method relying on user feedback. Every time the agent acts, the users next action is considered to be a reaction to this. However, this is based on a value that was already assigned to each user action, not the desirability of the users action for the agent. One might wonder, why not simply and directly assign a value to the agents actions instead. The answer is that an agent might have a wide range of possible actions and each actions might have different standards and hence require different values. Yet the user can be limited to a handful of feedback actions, requiring only little effort. What the agent eventually does, is averaging all values of the users reactions and using

this number as the value of this action. So if an action is often followed by a positive feedback by the user, the action is considered good. These values could also be used to rate other agents actions.

Using learning instead of predefined knowledge bases feels like a very sensible way to avoid most of the issues of using OCC. At the same time, it might not really be any more feasible for complex scenarios, as all the agents would first need to be trained and given sufficient opportunity to develop their standards and associations. Still, unless one is willing to take shortcuts or remain very abstract, this would seem to be best approach, as the tables used to store the learned knowledge could still be filled in manually. Most of all, learning allows agents to adjust, which is an important part of appearing believable, so this should be implemented anyway. The evaluation of FLAME is unfortunately very focused on whether the agent appeared intelligent or if he is perceived to be able to learn, so obviously the learning version would score exceptionally high compared to other version. What can be said, is that the agents general believability was perceived significantly higher when using learning, which is also no surprise. While this says more about the agents decision making than the emotional believability in terms of how emotions were caused in reaction to different events, it still demonstrates that learning is an important aspect for virtual characters and that there are good reasons for making it an integral part of artificial emotions instead of seeing it as purely in the realm of artificial intelligence.

As emotions, learning and decision making are very intertwined in FLAME, it is probably not the best choice for a separate module that's supposed to be easily added to an application. Too much is handled by FLAME that might better be left to the applications own logic and the definition of states could also pose a problem. What should a state consist of? Just the agents internal state? That would not really suffice to accomplish learning. Take fruit on a tree. The agent might shake the tree and fruit falls down, but if it's not part of the state that the fruit was up there in the first place, it will shake it again, not achieve anything and consider the first time as coincidence. At least all the relevant aspects of the world state would need to be considered, leading to very complex state descriptions. The agent not being able to know about a certain aspect of the state is another issue. Take a simple experiment, where pressing a button causes someone to bring food from another room. The agent couldn't know that pressing the button causes a light to go on and food is only delivered if somebody is actually around to see it. Instead, he needs to derive other hints, like the time of day. To function correctly a state would initially have to be very complex, considering pretty much everything the agent can tell about it. Then performing it quite often would remove the indifferent parts of the state.

So it would seem that FLAME can be very handy for simple scenarios, like a virtual pet, an interactive assistant or other uses with a clearly defined but narrow context. One could even imagine it working very well in a game like The Sims, particularly more recent versions that simulate the entire life of a Sim and the learning experience could be an integral part of the game. Another example where FLAME could be put to use are the Black and White games, where the player raises and trains his own creature. But adult characters would require a large knowledge base to not appear completely clueless

about the world. That would explain why FLAME was tested with the agent being either a dog or a baby.

5.4 ALMA

This model, called ALMA (A Layered Model of Affect [4]), has a different background and was created with the idea of controlling dialogs of multiple agents in scenarios where two or more agents are used to deliver information as part of a conversation, rather than having a single agent directly tell the user. As such, it is focusing heavily on the necessary information to control facial expressions, different ways of phrasing of something and body gestures.

ALMA, like the Emotion Engine, is differentiating between short-term, mid-term and long-term affects and also tries to connect them to different behavioral aspects. So emotions are for example affecting facial expressions or gestures, while mood might control an agent's posture. It is not surprising that these different affects turn out to be emotions (short-term), mood (mid-term) and personality (long-term).

Emotions are covered by implementing the OCC model, while personality is using the Big Five. The model itself is based on the Emotion Engine and extends it by adding mood, which uses Mehrabian's PAD-space. To have a way to describe different strengths in mood without using numbers, the length of the mood vector is used and divided into three ranges, which are mapped to three different intensities. A default mood is calculated based on personality and the active emotions, generated by the Emotion Engine, are used to adjust the agents mood using a push and pull function. This means ALMA calculates the average position of all emotions in PAD space as well as the average intensity of all emotions. Depending on whether the mood is between this position and the origin or already at or beyond this position, the mood is either pulled towards this point or pushed away deeper into the current mood octant.

ALMA uses a XML-based language called AffectML for configuration and definition of input and output data. Here, different tags and their attributes are defined, for example the desirability of *GoodEvent* or different actions and their desirability and praiseworthiness. This allows for someone using ALMA to comfortably define all possible actions and events, though ALMA seems to require these definitions for every single character. An interesting part of this are the definitions about emotion and mood displays. The display of an emotion or mood, either by the agent or others are also considered as events with different desirabilities.

What ALMA receives at runtime is a list of appraisal tags, act tags and emotion/mood tags, so another agent stating an insult might be translated into an appraisal tag like *BadEvent* and/or an act tag like *Insult* or *RudeGesture* together with an emotion tag like *AngerDisplay*. As for each of them different desirability, and where necessary praiseworthiness values have been defined, these tags can be used for the OCC appraisal process. The resulting mood and emotions are then used to select for example the phrasing of a sentence, gestures or facial expressions.

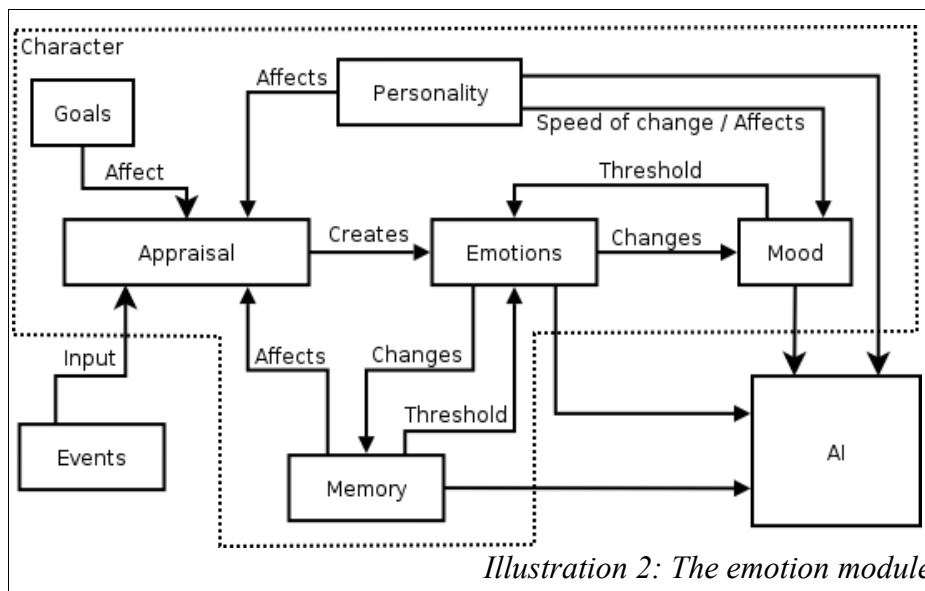
As the model of ALMA was used as basis for our approach, the description was kept short and mostly focused on the parts where it differs from our implementation. It is aimed at controlling a character's verbal and non-verbal expressions, heavily interacts with a dialog generation system and is implemented as part of the VirtualHuman system. Unfortunately the paper didn't give any information about how personality is used other than specifying a default mood, however the *personality profiles* as specified in AffectML can be counted as part of the personality and in that case they clearly define how the different events and actions are rated by the agent. This does not appear to be too useful in scenarios with a great number of agents, where one would possibly prefer to be able to determine random values for the personality traits and have this kind of information automatically be derived.

ALMA is cleverly avoiding issues like goals or standards, by directly defining desirability and praiseworthiness as part of events and actions for each agent. While predefined events and actions are sufficient for ALMA's intended purpose, in many other applications there might not be much point in defining this kind of information in advance. A simple scenario that allows the agent to find random amounts of money can show the problem. Here, the amount of money that's found and the agents financial situation are important factors in determining the desirability at runtime. A predefined event wouldn't suffice to handle this and even defining a dozen different events for different amounts would hardly be practical. Essentially, this is where goals would come into play and one would have to decide what kind of impact the event has on them. Though it is of course a viable option to leave the determination of desirability to the application. Analyzing an event's impact on various goals will typically have to be handled by the application anyway.

6 The emotion module

So the Emotion Engine was offering a simple and straightforward approach with emphasis on the visual display of emotions, FLAME offered a very complete learning based solution with the drawback that complex scenarios can quickly make the state descriptions unwieldy, and ALMA offers a less complex model, where complex scenarios increase the effort in two ways, namely more extensive personality profiles for every not supposed to act like a previously defined agent. Only the Emotion Engine seems to be useful for a less limited scenario without requiring a serious amount of extra work, yet we attempted to create a solution based on ALMA that doesn't require the specification of these profiles.

The diagram gives an overview over the different aspects of the module and how they interact. Most parts of the module affect several others and only the personality is stable and never changes. The AI is part of the application using the module and can access and use the different informations in any way the programmer decides to.



6.1 Characters

Almost all interaction with the module is happening through character objects that represent a single emotional agent and can be **saved** and **loaded** to retain their current mood and most of all the relationship information with other entities, which is so far the only kind of long-term memory. XML files are used for readability and ease of use, as many existing libraries allow to comfortably work with XML.

All agents are identified by their **name** string, though there is yet no check for duplicate names and the application programmer will have to look out for that. For a final version a more efficient and unique identifier should be used, especially to avoid conflicts with

user defined names in combination with emotion targets, emotion causes and **relationships**. These are stored as simple PAD value for each entity name that has caused emotions in the agent so far. Essentially, the relations are just an emotion average and can, for example, be considered as an associated mood towards the entity.

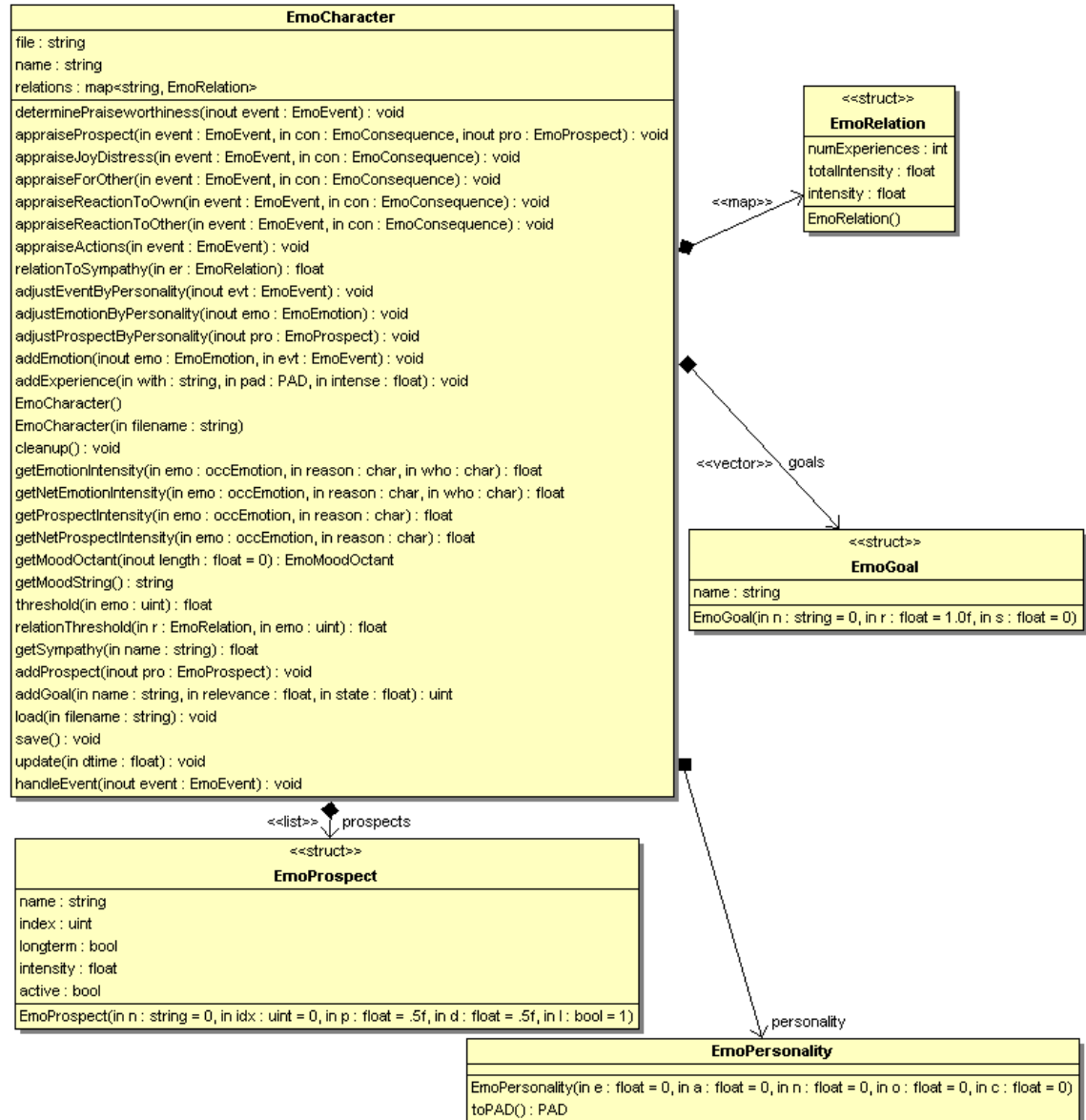


Illustration 3: UML: Character

First designs stored this in a more complex fashion, using values like sympathy, respect, trust and an estimated personality. Each of them would be important in matters of decision making and trying to guess the other entities personality would be useful for manipulating or simply predicting how another agent might react to a certain action. However, where the single PAD coordinate is easy to handle, the matter of how to actually determine all the above values in the first place would bring up many difficult questions. One possibility could be linking the different groups of emotions to different factors, for example sympathy for all positive/negative emotions, trust for gratitude,

respect for praiseworthy actions and so on. A more complicated mechanic could use an event, the displayed emotional reaction of another agent and knowledge of the appraisal process to draw conclusions about his goals and/or personality. However, at this point many other issues would be more important, as their results can be observed a lot more directly. Especially since the above would only provide values for a programmer to base decisions on, but wouldn't cause a different behavior in the module itself.

Mood is also stored as PAD and only changes over time by moving towards the average of all currently experienced emotions. Note that this includes inactive emotions as well. The reason is easily explained by a simple example. Imagine an event that should cause joy, but isn't positive enough for joy to exceed the threshold. Now more positive events keep happening and one would expect that after a while the sum of positive events will eventually cause active joy. This is achieved by letting the inactive joy affect the mood and hence lower the threshold. Otherwise, hundreds of those events could happen without having any impact on the agents mood, which would feel rather unrealistic.

All **emotions** have a cause or reason and a target in addition to an intensity. There are reasons for and against this. The above mechanism to let inactive emotions lower the threshold wouldn't be necessary, if all instances of the same emotion could be combined in a single instance. This would simplify the module, even if it might not be realistic in all cases. While the cause for an emotion is only relevant for debugging reasons, the

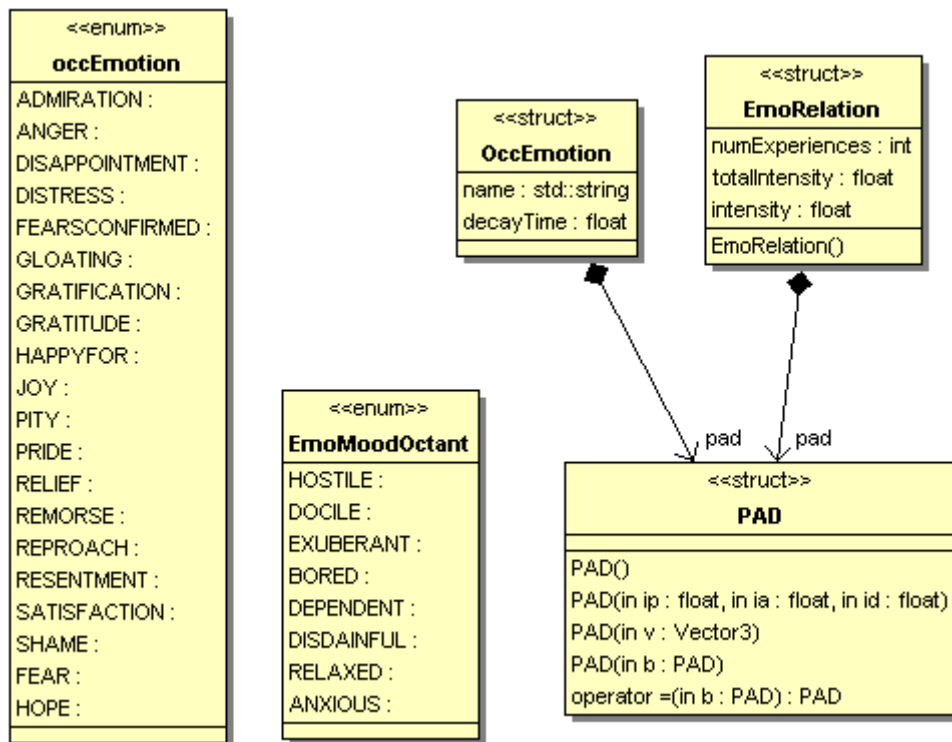


Illustration 4: UML: Emotions

target is too important to be removed. Being angry at a specific person usually shouldn't cause aggressive behavior towards another person, so it will be necessary for an application to have this information. At the same time, experiencing anger at one person

often causes people to lash out at others too or being easily irritated. Again, this should be reasonably modeled by anger affecting mood which lowers the threshold for anger that might be directed at others.

In its current version, the cause is ignored and a new emotion with the same target is increasing the intensity of an existing emotion. For this, the intensity of the new emotion is scaled by one minus the intensity of the old emotion to avoid an unrealistic linear increase and most of all to keep the resulting value in a valid range.

The **personality** of a character is stored as Big Five, but can be converted to PAD. While typically the former is used, there is one reason for the conversion to PAD, as personality works as a default mood, so when no other emotions are active, the current mood is slowly changing back to one corresponding with the agents personality traits. This is using the same formulas as ALMA.

$$P = 0.21 \cdot \textit{Extroversion} + 0.59 \cdot \textit{Agreeableness} + 0.19 \cdot \textit{Neuroticism}$$

$$A = 0.15 \cdot \textit{Openness} + 0.30 \cdot \textit{Agreeableness} - 0.57 \cdot \textit{Neuroticism}$$

$$D = 0.25 \cdot \textit{Openness} + 0.17 \cdot \textit{Conscientiousness} + 0.60 \cdot \textit{Extroversion} - 0.32 \cdot \textit{Agreeableness}$$

Each of the five traits has a range of [-1;1], though in practice it became apparent that a range of [0;1] would often be better. The current range is based on 0 typically being used as the average and values above or below are in relation to an average persons rating in this factor. The conversion to PAD relies on the currently used range.

As mentioned before, individuals showing some traits associated with one of these factors don't necessarily display all of them. This means using only the FFM would be too crude for a completely realistic simulation, yet separately specifying hundreds of traits wouldn't be practical either. But as these traits would eventually be specific to the context, an application could either use the Five Factors as default value and override it for other traits where necessary or treat them as goals instead. For example, punctuality would belong to conscientiousness, but doesn't internally matter to the module. So to model a generally not so conscientious but very punctual agent, punctuality could simply be defined independently while using the rating in conscientiousness for other traits like tidiness. Alternatively he could just have a high priority to a goal like *being on time*, depending on how the application is making use of it.

While short term prospects have to be created manually, long-term **prospects** are handled automatically. Little thought has been put into this part so far, so the way it is working at this time will only be explained briefly. Every event with positive consequences automatically creates a small amount of hope that the event will occur again. In the same way fear is created by negative consequences. There are many obvious problems, for example, there might be events where the agent should know that they won't happen again. In other words, many factors like probability are ignored. Additionally, automatically creating hope or fear of an event repeating itself can quickly cause erratic results, such as resulting in extreme satisfaction over irrelevant

achievements, because small but frequent achievements cause hope to constantly increase in intensity. Even with small increases, the values will eventually become unreasonably large, so this part of the module is best left unused until a better solution has been found.

6.2 Interface

The goal was to design and implement a library that allows to create and manage emotional agents independently of context. So it had to be very generic without becoming useless to any specific task, but at the same time as easy to integrate as possible by offering a simple interface. What this module is not supposed to do is decision making, planning or controlling the applications AI, as this is impossible without any context information. The programmer still needs to implement the layer between his application and the emotion module. For this, the first question to be answered is the agents role, most of all, if it should consider itself as an entity inside the application or as just another user. This mostly affects goal relevance. Praiseworthiness would strongly depend on that, but as it is automatically derived from all desirabilities, which in turn depend on the affected goals relevance, this is already taken care of indirectly.

6.2.1 Goals

Each agent needs to be given goals, so that during event appraisal the desirability of the event can be determined. A goal is simply defined as a string as identifier, a relevance value ranging from 0 to 1 and the current state of realization, also in the range 0 to 1. The last value simply means to which percentage the goal is already achieved.

```
unsigned addGoal(const string& name, float relevance, float state)
```

Currently, the implementation might be confusing and uncomfortable to use, as the function to define a goal returns the index of this goal in the characters vector of goals. As a result, the goal IDs must be stored separately for each character. This was originally done to allow for multiple ways of referencing a goal. Either by string, which would be easy but inefficient or by ID, which is faster, but requires extra work by the application programmer. At this point, only IDs can be used, but the final implementation should have overloaded functions to allow using an ID or name or specifying a relevance value instead. Whether this last option would be useful, depends on how exactly the goals will eventually be used during the appraisal process. Meaning, whether the change in realization, the changed realization or a combination are used. As goals have been added as a bit of an afterthought, they are not yet saved or loaded from files.

Originally, desirability and relevance were directly passed as parameters for a consequence, the introduction of goals was supposed to relieve the programmer of

having to determine these and automate the process some more. Unfortunately the loss in flexibility turned out to be difficult to compensate as in different situations, different ways to interpret the goal realization can be more appropriate than others. There is no single correct way to treat all goals. For example, there are goals that need to be achieved once and can then be removed. Other goals might be permanent, like staying well-fed or not being thirsty. As Orthony, Clore and Collins also suggested three different kinds of goals, it would appear like a worthwhile idea to extend the module in this direction. In short, they differentiated between active pursuit goals (A-goals), interest goals (I-goals) and replenishment goals (R-goals). For the purpose of this module, at least two different kinds should be implemented, where A-goals and I-goals could be considered achievement goals (goals, that need to be achieved only once) and R-goals are goals with a recurring character, like for example many biological needs. Especially the latter would be likely to decrease in relevance the more they are achieved.

So a future implementation should use a global vector of goals to provide consistent IDs, allow for loading and saving them in a separate file and possibly differentiate between different kinds of goals as described by the OCC model.

6.2.2 Events

The most important part of the layer between application and module is the translation of application events into emotional events. These can become complex structures, though often overly complex event objects are a sign that too many separate events are treated as one. All events need a **name**, as this is used in combination with prospects to check if a particular event was feared or hoped for. It will also be used as the reason for an emotion, because sometimes it is useful to know why an emotion was actually caused.

```
EmoEvent(const string& n)
```

Other than that, events are nothing but a list of **consequences** and **actions** that led to the event. Events without any actions can be considered to have happened without any direct cause, such as lightning striking a tree compared to a car that has been stolen by someone. Generally using one event per action is best, however more complicated chains of causality can be tricky to model.

Consequences need to specify the agent or entity that is affected, meaning **for whom** the consequences apply. To be flexible, this is specified as a string. It should however be noted, that some emotions might be created using this string as a target. More specifically all fortune-of-others emotions (pity, gloating, happy-for, resentment). Also, the affected **goal** needs to be specified. If a consequence influences more than one goal, multiple consequences need to be given.

```
EmoConsequence& EmoEvent::addConsequence(const string& forWho, unsigned goal, float desirabilit)
```

It might be worth thinking about implementing a hierarchy of goals, as this might cover for many cases, were this could be necessary. It would also allow for the module to derive consequences for higher level goals. As an example, imagine the high level goal is building a house. This could be divided into subgoals, like buying the property, having an architect creating a design and hiring a construction company. Finding an affordable architect would not only get you closer to achieve the goal of having plans for the building, but obviously would also get you closer to the higher level goal of building it. This could be done manually, but the goal for this module's design was to make the interface as simple and easy to use as possible.

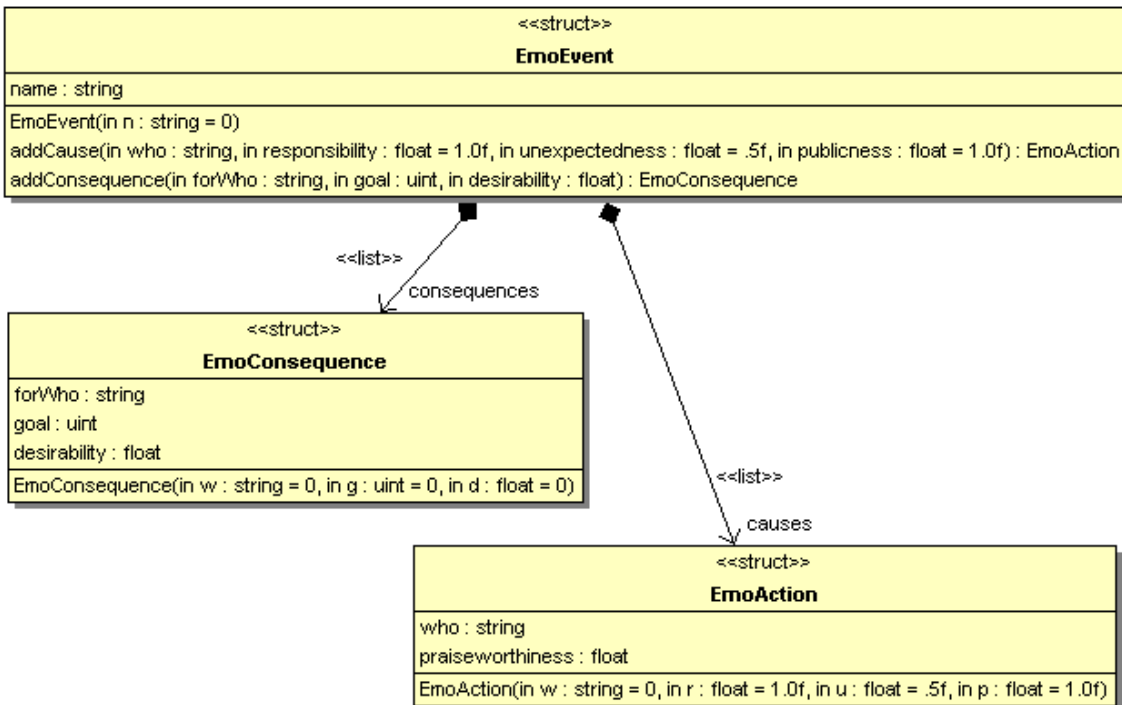


Illustration 5: UML: Events

The last property of consequences is the actual **effect** on the goal. So far, this is a rather abstract and isolated value, basically describing how useful it was for the goal on a scale of 0 to 1. A better approach might be specifying the difference in or the new state of the goals realization after the event. Obviously, these are only minor differences, however, it might still reduce the modules flexibility by removing the possibility of a nonlinear scale. For example, a goal changing from 10% to 15% realization might be considered to have more impact than a change from 90% to 95%. This might better be done by adjusting the goals relevance, though.

Defining **actions** is the most extensive task, as it requires the most parameters that might not be easy to determine. The **actor** is straightforward, simply the name of the acting entity as string. It is important to remember, that all caused emotions will be attributed to this entity and as a result it will be used as a target for some emotions. Most of all, relationships will be created with this entity, if it doesn't exist. Required values to adjust the actions praiseworthiness are the degree of **responsibility**, the **publicness** of the action and the **unexpectedness**. These values also range from 0 to 1

and conform to the factors described in the OCC model. All of these depend on context information and hence can't be derived by the module itself. **Cost** is missing as of now, because of an attempt to instead use the consequences for the acting agent, meaning that negative consequences for his goals are considered as costs.

```
EmoAction& EmoEvent::addCause(const string& who, float responsibility=1.0f, float
unexpectedness=.5f, float publicness=1.0f)
```

Finally, the event needs to be sent to all characters able to observe it.

```
void EmoCharacter::handleEvent(EmoEvent& event)
```

The biggest problem with the current setup is that event definition matters and can make a big difference to the caused emotions. Generally, it should be avoided to do too many things in one event and it should be carefully considered which actions and consequences belong together. Imagine someone running into a burning building to save someone. Another person is watching this. However, the attempt fails and both die. One would not expect to be blamed for the attempted rescue, but if running inside and the failure is treated as one attempt, the module would treat the action or going inside as the reason for the two persons death and the watcher might experience reproach instead of admiration. Only if the action is handled separately from the outcome will the watcher experience first admiration and later distress and pity.

As another example, the SIMPLEX scenario made it necessary to separate the selection of a target from the actual attack or rather its outcome. This is just a symptom of a more general problem. Causalities can often be very complex in real life and an action might have indirect consequences that the module simply can't recognize. As an example, agent A might drink the last bottle of water, leading to a thirsty agent B not finding any water and deciding to go and buy some. If B has a fatal accident on his way, it is quite possible that some agent C was very close to B and now blames A. Even though it might not be too rational to see A as responsible for the accident, just because of drinking a bottle of water, real people aren't always rational, especially in extreme situations. Writing a module that can already discover this kind of vague causality the first time it occurs and handle it automatically would be a very difficult task.

6.2.3 Prospects

Prospects are partially like emotions in that they have intensities and can be active or inactive. Short term prospects must be created by the application and require the name of the related event. Two more parameters need to be specified, the **expected consequence** and the events **probability** to come to pass. Both are used to determine the intensity of the prospect. The next time an event of this name is processed, active prospects are looked at and used to cause the appropriate emotion and then removed. Strictly following the OCC model, it would require the appraisal of a potential event to

create prospects. The way a prospect can be created more or less directly is to be considered a shortcut to defining a whole event, especially since the addition of an event probability would cause a completely separate appraisal path to be taken. So it is instead removed from the regular appraisal of actual events for simplicities sake.

The intensity is a function of probability and expected desirability that ignores the issue of a high probability causing less hope. Looking only at hope alone, it is quite right that an event that is almost guaranteed to happen will cause less hope, as hope requires some degree of uncertainty. However, hope and fear have no other function than being a prerequisite for the prospect based emotions. As their intensity is only based on the prospects intensity, one should not just consider the isolated prospect, but the consequences for the resulting emotions. If such an almost certain event doesn't happen, there is no reason for less intense disappointment. While on a semantic level hope would turn into anticipation, this distinction doesn't matter for the appraisal process. The same applies to fear. Being absolutely sure about a bad thing going to happen will not let you feel any less relief when it doesn't. So in this case fear will simply be looked at as dreadful expectation.

```
EmoProspect(const string& n, unsigned idx=0, float p=.5f, float d=.5f, bool l=1)
void EmoCharacter::addProspect(EmoProspect& pro)
void EmoCharacter::addProspect(const string& n, unsigned idx=0, float p=.5f, float d=.5f)
```

The last function prevents the programmer from having to construct a temporary EmoProspect object and instead creates it internally. The overloaded version taking an EmoProspect reference is left in for internal use.

6.2.4 Retrieving Information

Various functions are supposed to help a programmer using the characters emotional state for decision making or other tasks. Two functions are returning the current **mood octant**, either as defined constant and optionally strength or as a string where the strength is expressed in categories ranging from *slightly* to *extremely*, for example *moderately anxious*. How strong the mood is, depends on how far the mood is into the respective octant. To calculate this, the mood coordinate is projected on the vector to the octant's corner and divided by $\sqrt{3}$.

```
EmoMoodOctant EmoCharacter::getMoodOctant(float* strength=0)
string EmoCharacter::getMoodString()
```

This solution is temporary. Remember that the mood is attracted by the averaged position of all current emotions, hence it can never move deeper into any octant than the furthest emotion. So a mood will usually never come close to the maximum. To address this, it might be necessary to reintroduce the push-phase and change the way mood is used as a threshold. Instead of using the distance between mood and emotion, the

threshold could depend on how far into the emotions mood octant the mood currently is. However, it needs to be discussed if this a sensible approach for emotions that are close to the border between two or more octants.

Another important value for internal and application use is the **sympathy** towards other entities. Different attempts were made to come up with a good function to derive this from the stored PAD coordinate. One was using the position of the OCC emotions *love* and *hate* and either using the distance to each of them or a projection on the different position vectors.

```
float EmoCharacter::getSympathy(const string& name)
```

For simplicity and easier evaluation the module is for now simply using the pleasure coordinate. This is based on the idea that essentially associating pleasant emotions with an agent will cause sympathy in most cases. However, emotions like pity prove this to be difficult in some cases, as it requires positive sympathy to occur, but has a negative pleasure value and hence decreases the experienced sympathy. The question would be if constantly having to feel bad about a person would really cause someone to like this person any less. Keep in mind, that pity is considered a negative emotion, while gloating is positive, because positive and negative don't mean socially acceptable or morally good, but simply pleasant to the agent.

Finally, different functions are meant to return current **emotion** or **prospect intensities** to allow agents to act on impulse. For example, experiencing brief but intense anger can lead to actions that neither mood or negative sympathy would cause. These functions are kept somewhat generic with optional filter parameters like reason and target. This allows getting a total anger intensity, but also a total intensity over a certain event or directed at a certain agent. Two versions return either the full intensity or a net intensity with subtracted threshold.

```
float EmoCharacter::getEmotionIntensity(occEmotion emo, const char* reason, const char* target)
float EmoCharacter::getNetEmotionIntensity(occEmotion emo, const char* reason, const char* target)

float EmoCharacter::getProspectIntensity(occEmotion emo, const char* reason)
float EmoCharacter::getNetProspectIntensity(occEmotion emo, const char* reason)
```

Note that this might return intensities larger than one. So it might be a good idea to use the same mechanic that is used to combine multiple instances of the same emotion and multiply each intensity with 1 minus the current total intensity to keep the value at or below one instead.

6.3 Update

The last function of this application-module layer is calling the **update** function with the time that has passed. This will cause the character to update its internal state, which results in emotions being reduced in intensity depending on their decay values and the current mood. Active emotions dropping below their threshold will become inactive,

emotions dropping to zero or less are removed. It is also possible that changes to the mood can cause lingering inactive emotions to become active. All remaining emotions are used to calculate the emotion center, which then attracts the current mood. So does the default mood that's based on the agents personality, but a lot less than the current emotions. ALMA and the first implementation of the module were using a push-pull function. Here, if the mood is located between the origin and the emotion center, it is simply moved closer to the emotion center. If it is located beyond the emotion center, it is pushed outwards even more. But as emotions are positions and not directions in PAD-space, the push-phase has been removed.

To decay an emotion, the distance of the emotion to the mood is used. As the theoretical maximum distance is $2\sqrt{3}$ and the factor is supposed to be in the range $[0.5;2]$, it is scaled accordingly. The decay time is not given in decrease per second, but as the time needed for the emotion to fully decay from 100% to 0. It's worth pointing out that neurotic persons are also said to spend more time on negative emotions, meaning they should decay slower than positive ones. However, at this time all emotions are treated equally and the module does not differentiate positive and negative emotions. This would require explicitly specifying which emotion is supposed to be affected or at least into which category it belongs.

$$moodFactor = 0.5 + 1.5 \cdot \frac{|mood - emotion|}{2 \cdot \sqrt{3}} \quad intensity = \frac{moodFactor \cdot dtime}{decayTime}$$

For the emotion center, the emotions should not all have the same influence, so instead, the intensities of all emotions are added up and the ratio is used as a weight.

$$emotionCenter = \sum emotion \cdot \frac{emotionIntensity}{totalIntensity}$$

The average intensity of all emotions is required to determine how much the mood is attracted to the mood center. Obviously, that way the number of emotions won't matter and one strong emotion could essentially be canceled out by many weak emotions, even if they are happening to lie close to the strong emotion. Using the total intensity would seem the better approach, however, consider a case, where all emotions are evenly distributed and the center is close to the origin. The total intensity might be very high, attracting the mood at high speed, but the origin means that any experienced mood will be rather weak. It might be better, to instead keep the mood in its current position, as it is technically pulled in all directions. Not calculating the emotion center and instead having the mood attracted by all emotions in turn might be easier and avoid this problem.

Another factor is based on the agents neuroticism. Higher neuroticism means a tendency to be moody and experience mood swings, which is simulated by simply shifting the mood faster. Two variables determine the minimal and maximal speed.

$$speed = moodChangeMin + (moodChangeMax - moodChangeMin) \cdot \frac{neuroticism + 1.0}{2}$$

$$newMood = mood + \frac{emotionCenter - mood}{|emotionCenter - mood|} \cdot avgIntensity \cdot dtime \cdot speed$$

Finally, the mood is also shifted towards the default mood, but with a factor of 0.02 this is happening only slowly. Neuroticism should actually be used here as an additional factor as well.

$$newMood = newMood + \frac{personality - mood}{|personality - mood|} \cdot dtime \cdot 0.02$$

The update function is implemented as a loop with fixed time steps to guarantee consistent behavior.

6.4 Appraisal

Each event handled by a character is first **adjusted** according to the agents **personality**. At this stage, only the consequences are adjusted based on the agents neuroticism. As neurotic people tend to see things more negatively, it is used to rate the consequence worse than it actually is. The factor by which neuroticism can reduce the desirability is configurable. A value of 0.3 is currently used.

$$desirability = desirability - neuroticism \cdot neuroticismInfluence$$

Note that all personality traits are in the range [-1;1], so negative neuroticism would actually make the consequence more positive. It's questionable if something slightly bad should actually end up being perceived as good in any way and whether a range of [0;1] might be better. For now, the justification is to imagine positive people thinking “it could have been worse” to explain this effect.

Afterwards, the **praiseworthiness** for all actions is determined. Because this step is supposed to not require a large knowledge base, only the consequences and passed parameters can be used as a measure. Basically, the more positive consequences an action has, the more praiseworthy it is considered to be. However, sympathy should play a role in this. For example, an action with negative effect won't be rated as bad for someone the agent doesn't like, as it would be for someone he cares about. Using sympathy as a factor would come to mind, but while it would appear quite natural for small values, it would result in absurd evaluations for large values. For example, a good thing for a slightly disliked person would result in a very slightly negative praiseworthiness, but one wouldn't expect a horrible deed to become extremely praiseworthy. The proverb “not wishing it on your worst enemy” comes to mind. For now, sympathy is added to positive values and subtracted from negative ones.

$$perceivedGoodness = desirabilityForOther + sympathy \cdot sgn(desirabilityForOther)$$

This is still far from perfect or even realistic and requires more work, but it results in usefully shifted values for non-extreme sympathy values. It should also be considered

that conscientiousness might determine how much impact sympathy should have at this point, as it stands to reason that very conscientious people are more strictly applying their standards and ignore personal relations. The same argument can be made when looking at the consequences for self. Typically, one would expect that they are factoring more heavily than consequences for others, though high conscientiousness might result in a more neutral point of view. At this point, they are simply scaled by 50%.

$$\textit{perceivedGoodness} = \textit{desirabilityForSelf} \cdot 1.5$$

After the adjusted values for all consequences have been summed up, **conscientiousness** is so far used to adjust the final result, by being scaled and subtracted, so the more conscientious an agent is, the harder it will be to commit an action so positive that it is deemed praiseworthy. This applies to both, actions of other agents and actions of the agent himself. However, this doesn't allow for situations where a conscientious agent is deeming an action more praiseworthy than others, for example because he is impressed with someone sticking to standards and principles. A simple example would be refusing to commit a small misdemeanor to prevent large problems for a friend. Here, the result should be exactly the opposite and the conscientious agent might approve of it, while others condemn letting down a friend. Unfortunately, without having defined standards and a way to measure how much an action is in accordance with them, the module again lacks the necessary context information to correctly handle these cases.

$$\textit{praiseworthiness} = \textit{conscientiousness} \cdot \textit{conscientiousInfluence} + \sum \textit{perceivedGoodness}$$

Agreeableness works the opposite way, but only for actions of others, based on agreeable people tending to be more forgiving in order to get along with others. Apart from a different weight, this results in the same as negative conscientiousness, wrongly implying that these two traits are opposites. Again, this issue can be avoided by not allowing negative values for personality traits.

$$\textit{praiseworthiness} + \textit{agreeableness} \cdot \textit{agreeablenessInfluence}$$

The remaining factors that had to be passed as parameters for the action (**responsibility**, **unexpectedness**, **publicness**) are as of now simply averaged and used to scale the result of the above calculations. So at this point, the positive or negative 'praiseworthiness' absolute value can only become smaller. Finally, as cost is attempted to be derived from consequence for self, it is subtracted, before the calculated praiseworthiness is averaged over the number of consequences or rather the number of affected agents.

$$\textit{praiseworthiness} \cdot \frac{\textit{responsibility} + \textit{unexpectedness} + \textit{publicness}}{3}$$

$$\frac{\textit{praiseworthiness} - \textit{desirabilityForSelf}}{\textit{numberOfConsequences}}$$

The resulting value is also used as the intensity for admiration or reproach, depending on whether it is positive or negative. If the agent is appraising his own actions, the

emotions are pride or shame instead.

Unfortunately there are endless possibilities to use all these factors, none of which can really be considered correct in any objective way and there are many special cases and exceptions that depend on context information the module doesn't have. Determination of praiseworthiness is still an open issue that is important to be answered, as it is required for 8 out of 20 OCC emotions that can be created by the module.

Once the praiseworthiness has been calculated, the list of prospects is searched for all those that are active and match the name of the event. For each, the **prospect appraisal** function is called, which will determine the net desirability by multiplying it with the affected goals relevance.

$$netDesirability = goalRelevance \cdot desirability$$

This value will be compared to the expected desirability for this event. Different approaches were tried to create believable results. The simplest situation is when a positive consequence was expected but a negative one occurs. This would obviously cause disappointment. However, this is also the case if a very high desirability was hoped for and the actual consequences are less positive, but still not negative. The first version only compared the signs to determine the emotion and the difference to calculate the intensity. Currently, a different formula is used.

$$quality = 2 \cdot \left(\frac{netDesirability}{expectedDesirability} \right) - 1$$

Having a hope fulfilled results in satisfaction. So if an event has exactly the expected consequences, there should be the full intensity for the emotion. But at which point below the expected result should this satisfaction turn into disappointment? This value is very arbitrary again and though there might be personality traits that should affect this, the implementation is using 50% as turning point. So, assuming that 0.8 was hoped for, the results for different actual values would be 1 for .8, 0.5 if the desirability is 0.6 and 0 if the value is 0.4, i.e. half the expected value. Anything below that will result in negative values and create disappointment that reaches full intensity at a desirability of -0.8. The resulting values are clamped to [-1;1], as consequences that are even better than expected can quickly cause very large numbers. It would appear normal that the satisfaction is even greater in that case, however, this is better represented as joy, which is appraised separately.

The emotions intensity is the product of the determined *quality* and the prospects intensity. If there was very little hope, there can't be strong satisfaction. This is why the clamping happens in the step above and not after the multiplication.

$$emotionIntensity = prospectIntensity \cdot quality$$

Which emotion is created depends on the kind of prospect and the sign of the quality value.

	Quality > 0	Quality < 0
Hope	Satisfaction	Disappointment
Fear	Fears confirmed	Relief

After the prospect appraisal is done, short term or one-shot prospects are removed.

Appraisal in regard to **joy and distress** is done for each consequence affecting the agent himself, while appraisal for pity/gloating and happy-for/resentment is done for the remaining consequences. The former is straightforward, weighs the desirability with the goals relevance and directly uses the absolute value as intensity. To determine the intensity for emotions that are reactions to consequences for others, this value is additionally multiplied with the sympathy to this entity. For entities that the agent is rather indifferent towards the intensities will be very low. Like the adjustments during the determination of praiseworthiness this causes one problem in that an agent will always react with gloating to a negative event for a disliked character, regardless of the severity. Even a slightly disliked character should usually be pitied if the event is bad enough. At the same time, it works well for positive events, as resentment towards a slightly disliked character appears quite logical. So negative and positive consequences would be better treated separately to handle these cases more correctly. On the other hand, it might make sense to cause pity and gloating at the same time and see which one is the dominating emotion.

	Sympathy > 0	Sympathy < 0
Desirability > 0	Happy for	Resentment
Desirability < 0	Pity	Gloating

The remaining emotions are the result of combining the consequences for self and praiseworthiness, hence they are referred to as **compound emotions**. If the agent himself was the cause, the emotion is determined based on the desirability of the event. Positive consequences cause gratification, negative ones result in remorse. For the intensity, four different cases are considered.

$$desirability > 0 \wedge praiseworthiness > 0 : intensity = desirability \cdot praiseworthiness$$

This means that gratification can only occur, if the results were positive and the action is deemed praiseworthy as well. However, it seems that 25% gratification as the result of 50% desirability and 50% praiseworthiness is too low. Using the average value instead could prevent the typically created intensities to be too low to actually get above the threshold. The same function with negated result is used for remorse.

$$desirability < 0 \wedge praiseworthiness < 0 : intensity = -desirability \cdot praiseworthiness$$

Opposing combinations are also used and it should be comprehensible that less remorse over bad consequences is experienced if the agent can fall back to knowing that he did the right thing, so to speak. At the same time, if an event is positive for him, he might

ignore the fact that his actions were blameworthy and still feel gratification.

$$desirability > 0 \wedge praiseworthiness < 0 : intensity = desirability \cdot (1 + praiseworthiness)$$

$$desirability < 0 \wedge praiseworthiness > 0 : intensity = desirability \cdot (1 - praiseworthiness)$$

It quickly becomes apparent that these functions don't work well, because of the jump at the transition from positive to negative praiseworthiness. Where positive praiseworthiness reduces the intensity to 0 when it is nearing 0, the negative praiseworthiness uses the full desirability at 0 and only decreases towards -1. One could argue to completely drop these functions instead, however that wouldn't allow for the above cases where result and praiseworthiness partially compensate for each other. Using the average value would easily solve this as well and should be used in the future.

Appraisal for compound emotions regarding other entities' actions are handled analogous and cause either gratitude or anger. Table 1 is giving an overview of emotions, their positions in PAD-space and their criteria.

	Emotion	PAD	Cause
Positive Emotions	Admiration	0.5 , 0.3 , -0.2	Praiseworthy deed by other
	Gloating	0.3 , -0.3 , -0.1	Bad consequence for disliked other
	Gratification	0.6 , 0.5 , 0.4	Good consequence through own deed
	Gratitude	0.4 , 0.2 , -0.3	Good consequence through others deed
	Happy for	0.4 , 0.2 , 0.2	Good consequence for liked other
	Hope	0.2 , 0.2 , -0.1	Potential good consequence expected
	Joy	0.4 , 0.2 , 0.1	Good consequence
	Love	0.3 , 0.1 , 0.2	Attractive entity
	Pride	0.4 , 0.3 , 0.3	Praiseworthy deed
	Relief	0.2 , -0.3 , 0.4	Expected bad consequence not confirmed
	Satisfaction	0.3 , -0.2 , 0.4	Expected good consequence confirmed
Negative Emotions	Anger	-0.51 , 0.59 , .0.25	Bad consequence through others deed
	Disappointment	-0.3 , 0.1 , 0.4	Expected good consequence not confirmed
	Distress	-0.4 , -0.2 , -0.5	Bad consequence
	Fear	-0.64 , 0.6 , -0.43	Potential bad consequence expected
	Fears confirmed	-0.5 , -0.3 , -0.7	Expected bad consequence confirmed
	Hate	-0.6 , 0.6 , 0.3	Repelling entity
	Pity	-0.4 , -0.2 , -0.5	Bad consequence for liked other
	Remorse	-0.3 , 0.1 , -0.6	Bad consequence through own deed
	Reproach	-0.3 , -0.1 , 0.4	Blameworthy deed by other
	Resentment	-0.2 , -0.3 , -0.2	Good consequence for disliked other
	Shame	-0.3 , 0.1 , -0.6	Blameworthy deed

Table 1: Emotions, PAD and appraisal

6.5 Open Issues

6.5.1 Praiseworthiness

Determining praiseworthiness based on nothing but consequences has various drawbacks. For one, it is less flexible and makes it impossible to model a person with irrational views on what's a good or bad deed. What looks like a rare exception at first, has many examples when considering cultural differences. Take an extreme example like honor killings. To us it is inconceivable, how brutally murdering a relative could ever be a good and praiseworthy thing, especially when the reason is nothing but a minor misdemeanor to us. However, an argument for using consequences instead of standards can be derived from this, as cultural concepts of right and wrong didn't randomly come into existence. There are reasons and these reasons can often be modeled as a goal. In this case, upholding the families honor must have a very high priority. Unfortunately, this also requires more context information. For example that a misdeed of one family member is considered as shameful for the whole family. This would again leave the application programmer with the problem of specifying the goals accordingly and considering all relevant consequences. Eventually, the loss of a family member must be rated as less undesirable than the perceived shame. As it would be questionable to claim that the first goal has little value in these cultures, this value would have to be adjusted as a result and more so, an exception would have to be made for this one member, as the others aren't affected. More work for the programmer and this is just one example of very many.

Also, minor and subtle issues are even harder to model, but at least they would mean a lot of work in either case, no matter if many standards are defined or many goals and consequences need to be considered. How to place the cutlery after eating, leaving something on the table, shaking hands, colors to wear on weddings and funerals are all examples of daily pitfalls that are perceived completely differently in different cultures and it is virtually impossible to handle them all. But again, for the end result, it doesn't matter if they are appraised by different standards or based on different goals. In fact, the goals would even be the same, just the perceived desirability would differ. Here, handling all this in a personality profile as ALMA does would even make sense and keep the actual code cleaner. However, an option to include more generic profiles would appear to be an important change to lessen the workload. All the above examples could for example be treated as *cultural profile*, yet, this implies a mechanic to adjust them by personality or allow for overriding single entries to handle special cases. For example, an agent might have moved and picked up some cultural habits, but not all.

Finally, another consideration missing in the current implementation is the agents sympathy between each other, not just the sympathy of the appraising agent towards the affected one. The SIMPLEX scenario has a situation where a decision has to be made about supporting another player. Both options will have a positive consequence for one and negative one for the other, both of equal strength. In this case, if the appraising agent has the same sympathy towards both, he would eventually be indifferent with a net praiseworthiness of zero. In a more realistic scenario loyalty towards friends would

become an issue, making the choice that benefits a friend more praiseworthy than one hurting him. This would require the agent to consider their sympathies as well, given that he knows about them. Yet, how much so would again depend on how he values friendship. Another case where trying to relieve the programmer removes a lot of flexibility and it seems like a better approach to let him define the praiseworthiness of an action himself.

There doesn't seem to be a clean and simple way to implement this, that achieves the goal of keeping the module context free and causes as little work as possible for a programmer using it. As the above considerations show, the problem can be shifted from standards to goals/consequences, but the fact remains that too many variables, cultural and situational factors influence how an action is perceived. Personality alone could never cover for all of them and even the factors specified in the OCC model only handle a part of them. Maybe more research into the origin of different cultural standards and goal priorities could help to derive this more easily in the future.

6.5.2 Long-term prospects

The implementation of prospects suffers from the fact that they weren't explicitly considered at first and are only a provisional implementation to allow for prospect based emotions. As a result, only manually defined short term prospects have been of interest so far.

While the current model considers long-term prospects, they are simply expectations about the occurrence of an event based on previous occurrences. However, there is no interaction with defined goals. The module needs a mechanic that creates hope to achieve a goal, when the agent is getting closer to doing so. From there arises the question of how decay of prospects should work and how its intensity should be determined in the first place. For example, if due to an event the realization of a goal reaches 98%, should the resulting intensity be independent of whether it previously was 60% or 97%? How long should it remain at this intensity? One would expect hope to be quite intense at first, but not to remain at this intensity for many years if no further progress is achieved. While the current model could easily use an event that reduces goal realization as a trigger to reduce hope, the only mechanic to reduce hope on its own (without any event with negative consequences for this goal) is decay. But the time for decay can be radically different. The relevance of a goal might be enough information to determine a believable amount of time, but should be more important to the intensity instead. Though obviously starting at a higher intensity also means a longer time before decay reduces it to zero, this is insufficient to create the vast difference between giving up hope after a few minutes and many months.

A related issue is where a prospects intensity should set in if it simply decayed for a while and is now coming back to mind. This won't be a problem if the intensity is only based on the degree of a goal's realization, but might be of interest if it also depends on the change in realization. It also prevents disappointment if after a long time without change an event has negative consequences, as hope wouldn't be active or might have

decayed completely. So it might be better not to have decay at all and use a more differentiated concept than active/inactive. There could be dormant prospects, that, while still above the threshold, wouldn't be active, but come back to mind when a related event occurs.

Finally, it should be considered to completely drop the concept of long-term prospects, as nothing comparable is found in the other models. Instead, an approach like FLAME, where learning about causalities can be used to automatically create regular prospects in the appropriate situations might be preferable. It also far better models the original idea behind these prospects, which is simulating expectations based on previous occurrences of an event. Learning and considering sequences of events is a lot less crude than deriving it from single events alone. To compare the similarities, imagine playing the lottery. If the agent happens to win straight away, both approaches would lead him to expect to win again, as both would ignore the chances to win the lottery. After not winning a number of times, this expectation would be reduced, in the current module that means the hope for event *win lottery* would lose in intensity and maybe be replaced by fear of losing. However, while FLAME learns about causalities, it would not miss the obvious requirement to actually play the lottery, while our agent would simply hope for the event to happen again, even without playing.

So, while there are some ideas about how to couple prospects and goal achievement, this might be an attempt to create a direct connection where it shouldn't exist. At the same time, it roughly resembles learning about the probability of an event and does so in a way that is self-contained within the module and doesn't require additional information or the introduction of learning.

6.5.3 Memories

The relationship information has been called memory before, but one should still consider the effects real memories can have on people. For example, they might cause emotions with no apparent cause. Though this is more unlikely when the mind is kept busy with other tasks, once it is getting idle, it might wander off, following a train of thought or a chain of obscure associations where it is impossible to see a connection to the current situation. It would require more research into how the human mind works or picks the memories it brings up. However, simpler situations could be adequately handled with more obvious associations. In fact, FLAME is doing just that by using simple conditioning and tying emotions to objects. While this is still a very simple approach, it already allows for realistic reactions without having to store too much information. Somebody that had a traumatic experience would eventually have a number of related objects that could trigger the same emotion at a high intensity. Of course this skips the actual memory of the event coming back to mind and being appraised again, as one might expect, but the result is the same and so it would be sufficient to convince a user of the agents realistic behavior.

This also corresponds with the mentioned *broad and shallow* approach that was used for the Oz project. Instead of investing too much time and effort in making one aspect of

the agents behavior as detailed and realistic as possible, it is considered more important to his believability to react at all. The user probably won't care if the agents reacts to an event with joy at an intensity of 34% instead of 52% because of an unrefined formula, but he will immediately notice it, if a dog doesn't react to a stick he has been repeatedly beaten with. In short, one shouldn't always attempt to recreate the exact processes going on in a real person or even recreating the exact same results. For most scenarios and uses this wouldn't be noticed by the user anyway.

Now, considering that relationships are stored as the average PAD-value caused by the other agent, this is already very much the same as the conditioning in FLAME, except that it only applies to other agents. As these are only represented as a string, it could easily be extended to store the same information for all kinds of objects as well. It would still be necessary to add a mechanic that actually reacts to this agent or object according to this value. Another issue is the way the old value is reduced when a new experience is made. This was supposed to have new experiences overshadow older ones, however, it is more likely that a real person would react differently and, for example, after the other agent performs a bad deed, would remember previous negative experiences more prominently than positive ones. One way to model this might be a temporary modifier to the stored value or separately storing the intensities of positive and negative experiences. But as the benefits of broad and shallow were just explained, one should first use the simple method and only make things more complicated if it proves to be necessary for a more convincing emotional agent.

Then of course, if the goal is a realistic simulation, the agent shouldn't have a perfect memory, be unable to memorize certain things, permanently or temporarily forget others. While that reduces the amount of data that needs to be stored for an agent, modeling the mechanics of human forgetfulness in a way that covers highly distracted and confused persons as well as photographic memories will be yet another difficult task.

6.5.4 Interaction and inhibition between emotions

Another very important aspect is missing in the current model, though it can be considered to be at least partially and indirectly implemented. As mentioned before, some emotions can prevent other emotions. So far, only using the mood as threshold is somewhat modeling this, as negative emotions would automatically make it harder for most positive emotions to be triggered, depending on their position in PAD-space. However, this only works over a longer period of time, after the mood had time to actually shift. But sometimes conflicting emotions can be caused almost simultaneously or by the same event.

A simple example is pain and fear. In a dangerous situation where one was hurt and the threat is still imminent, fear would let you ignore the pain, so you could flee and get to safety. Once the threat is gone, pain takes over and now helps suppressing the fear in order to treat your injuries. Now, pain is actually more a sensation and not an emotion, but the concept is still the same. FLAME is handling this by letting the more intensely

experienced emotion win over the opposite emotion.

Transferring this to the above example, fear would be more intense than pain with the threat still around, while later the fear is reduced below the intensity of the pain. So letting the stronger emotion win would seem to be a sensible solution. However, it would still have to be determined which emotions are considered as opposites and if some emotions weigh more heavily than others. A simple approach to be tested would be treating all emotions commonly considered as negative as opposites for all positive emotions. Yet, it was decided early on that the module should allow for ambivalence, so a more refined selection has to be found, if this goal is to be achieved. This would also help reducing the number of active emotions that can be the result of a single or very few events.

Ambivalence and filtering would also seem to be the key to achieve opposite reactions in different agents. To pick up the example of offered help being answered with gratitude or anger, all that is needed is one agent assigning positive desirability, while the other perceives it as undesirable. A simple shift in desirability is unlikely to wield the desired results, but different goals could achieve this. If both agents have the same goal and the offered support would help achieving it, both agents would create gratitude. However, if one of them also has the goal of appearing self-reliant and independent, this offer would also be considered undesirable and as the helping agent is considered the reason, anger at him would be caused. Which reaction will eventually win is depending on which of these two goals has the higher priority and the sympathy towards the helping agent affects the net intensity of the anger.

One could decide that just adding up all consequences and their desirabilities will have the same result, however there is one important difference. Take a desirability of 0.5 and one of -0.6. Using the sum of both, the appraisal would use only -0.1, which might not even allow the resulting emotion to cross the threshold. The alternative would be two emotions of intensity 0.5 and 0.6, here, the winning emotion would still be at full intensity.

7 Scenarios

In order to test the module, a game scenario was created that had to fulfill multiple **requirements**. The most important was possible cooperation, but besides cooperation between players, it should allow for different styles of play suiting different personalities. So, while cooperation with others was intended to be a crucial part of the game to cause more interaction, it had to be possible to instead play it on your own, or to act friendly or aggressively. At first it was decided to look for suitable open source games and simply modifying the existing AI. However, it quickly became apparent that finding such a game was very difficult, not to mention that actually understanding the existing code to make meaningful changes would take a lot of time.

7.1 EmoSettlers

A new game was written instead. As the board game Settlers of Catan was considered to be going in the right direction, the new game was mostly based on it, but simplified in many ways. The goal of the game was to be the first to build a specific building (a palace) to win. The core element was the accumulation of different resources, which were then used for buildings and could be traded with between players. Resources were distributed randomly, but each had its own probability, hoping that rare resources would enforce increased trading among players. Additionally, a predetermined number of other buildings had to be built, before the palace became available. Deciding on which buildings would be used to achieve the minimum number of buildings was left to the player. Each building had a different effect on game play or would make new options available. The number of actions per turn was limited for different reasons that will be explained later.

The game was written as **client/server** application, where the client would be mostly limited to display the current state and take the users input. This not only reduces the possibilities of manipulating the client to gain an advantage over other players, but most of all made it easier to write different clients.

During early development a simple console client was used to allow for easier prototyping and minimal effort for changes and additions.

```

C:\temp\Rar$EX00.953\Client.exe
  Player      W  S  C  I  Glds Hrbr  Manu  Twrs  Swts  Wall
T->Player 1  0  1  1  3  1   0   0   0   0   0
              ?  ?  ?  ?  1   0   0   0   0   0

2 actions left: <B>uild
                  <A>uction <D>one

                <G>uild <H>arbor <M>anuf. <T>ower <S>weats. <W>alls <P>alace
Wood           22         1         1         1         0         1         2
Stone          14         0         0         1         0         1         2
Clay            1         2         1         1         2         2         2
Iron            2         1         1         3         1         4         2
  
```

Illustration 6: Emotional Settlers Dos-Client

Obviously this wouldn't be suitable for the planned evaluation phase, where other people would have to use it and the console client was offering no overview over

relevant information. Most are more comfortable using a GUI. So later on a new client with a graphical interface was written using wxWidgets to maintain some degree of platform independence. Multiple test games with less computer savvy players had already shown some issues that had to be taken care of in the new client. Particularly, an ever present overview over resource values and building prices were needed, as during most auctions a player needs to know what an offer is worth and which resources he could actually need. While this could be learned within a few games, it would have started off the evaluation by irritating the test person.

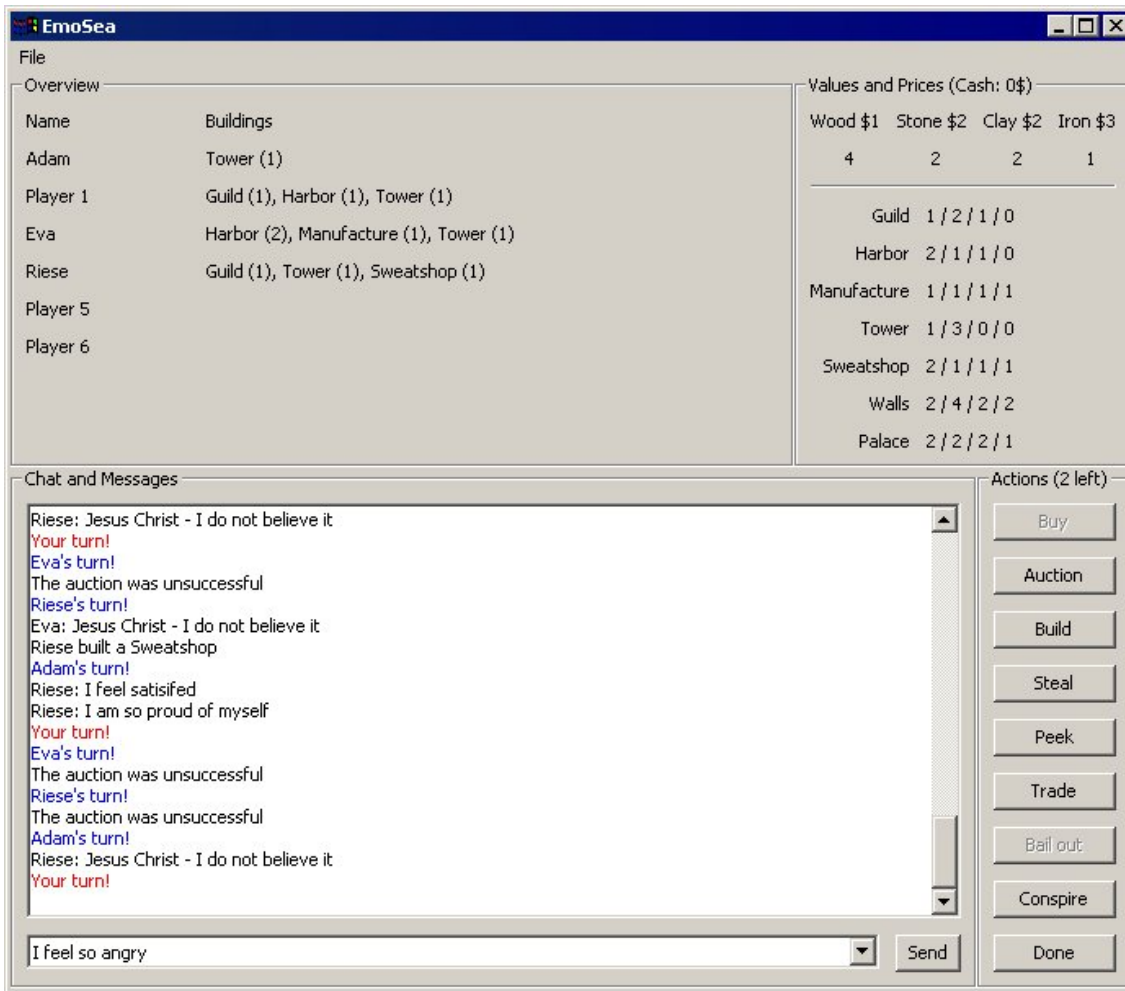


Illustration 7: Emotional Settlers Gui-Client

7.1.1 Resources

The **resources** were wood, stone, clay and iron, where wood had a value of 1 and would be dealt with a probability of 42%. Stone and Clay had a value of 2 and were received at a 22% chance. The most valuable resource with iron with 3 and a likelihood of 14%. Prices and probabilities were kept mostly proportional. Each player would be dealt 4 resources of random choice at the beginning of the game and 2 on the beginning of each of his turns. This is done by creating a random integer between 0 and 99 and picking the

respective resource by stacking the probabilities. So 0-41 would return wood, 42-63 was stone, 64-85 clay and 86-99 iron. However, despite using an existing implementation of the Mersenne Twister to create pseudo random numbers, in most test games a scarcity of Stone was perceived. This might be a subjective impression, as Stone was required the most of all resources.

7.1.2 Buildings

The buildings required a specific amount of each resource. While regular buildings didn't have any other prerequisites, two exceptions were made. Walls, which required four other buildings first and the palace, which could only be built if there were walls. Each building was limited to a maximum of three, to force at least some variation. An unplanned effect of the implementation was that buildings could not be used until the next turn. This was because the server didn't recheck the available actions for the current player after entering his turn. But as this felt like a sensible feature, it was kept this way to give other players a chance to react.

A thieves **guild** allowed for a negative kind of interaction, namely the attempt of stealing other players resources. Multiple guilds would increase the chance of succeeding and reduce the risk of being caught and losing your next turn.

The **manufacture** increased the chance of getting more valuable or multiple resources. This was achieved by adding an offset depending on the number of manufactures to the random number used to pick a resource.

Type of resource : result mod 100

Amount of resource : 1 + result ÷ 100

The effect is that by shifting the window, the probabilities for each resource remain the same, but it becomes more likely to receive the more expensive resources or multiple cheap resources. Each manufacture would cause a shift of 33%.

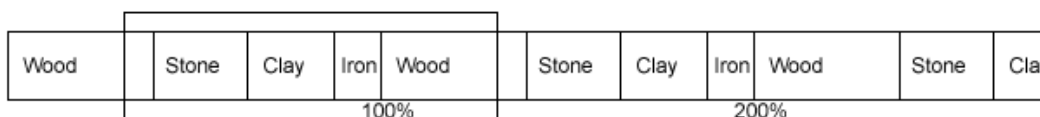


Illustration 8: Shifting the resource window

To allow for a more independent style of play, a **harbor** was introduced. This made it possible for a player to trade his resources at a fixed price and without requiring other players to consider his auctions. To balance things out and prevent every player from relying completely on this option, it was made more expensive. This additional cost could be reduced by building multiple harbors.

Certain options would benefit from being able to use them multiple times. A **sweatshop** could be built to increase the number of available actions by 1 for each of these

Buildings

buildings, allowing for example more auction or attempts to steal resources.

Satisfying curiosity and allowing for a defensive style of playing is what the **tower** was used for. Each tower not only reduced other players chances to steal from you, but also increased the likelihood of them getting caught. Additionally, for each tower you could take a quick look at the other players resources, which were otherwise hidden from you. **Walls** didn't have any use of their own other than being a prerequisite for the **palace** and a warning to other players that you were close to winning the game.

7.1.3 Actions

The most essential option was the construction of **buildings**. The player was presented with a list of options to choose from and the required resources. Only possible options were selectable, yet the server would still double-check the validity of the selected building. Disabling invalid options is not just more user-friendly, but also removes the need for various error messages about why the choice was wrong. Possible reasons would be not having the needed resources, having reached the limit for this building or building a wall or palace without meeting the requirements.

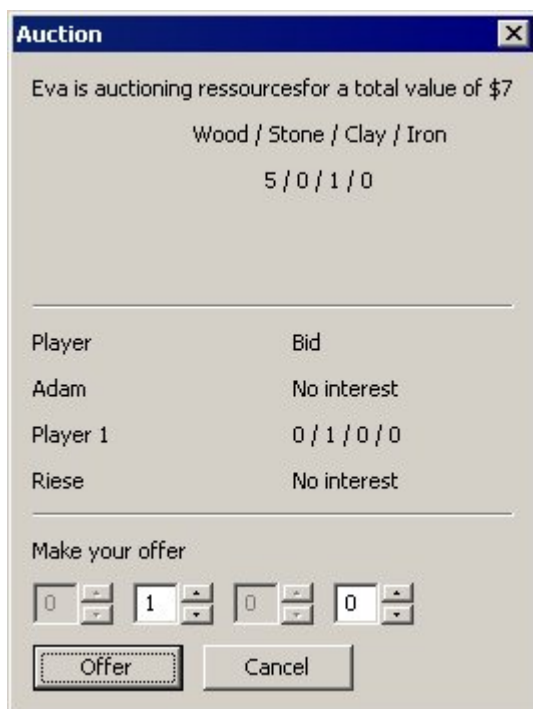


Illustration 10: Other's auction dialog

limits to adjust your offer or changing your mind completely until the seller made his decision about which offer to accept (if any).



Illustration 9: Build dialog

A player could start an **auction** and offer some of his resources. Unlike a real auction, he could decide whether to accept any of the offers and which one. This was supposed to allow for sympathy between players to cause more cooperation and even making a less than optimal choice in favor of a well liked agent. To simplify the process, the player could select which resources he would accept in turn, so others making an offer would be limited to those desired by the seller. Each player could then dismiss the auction if he wasn't interested or he could send his offer. There were no

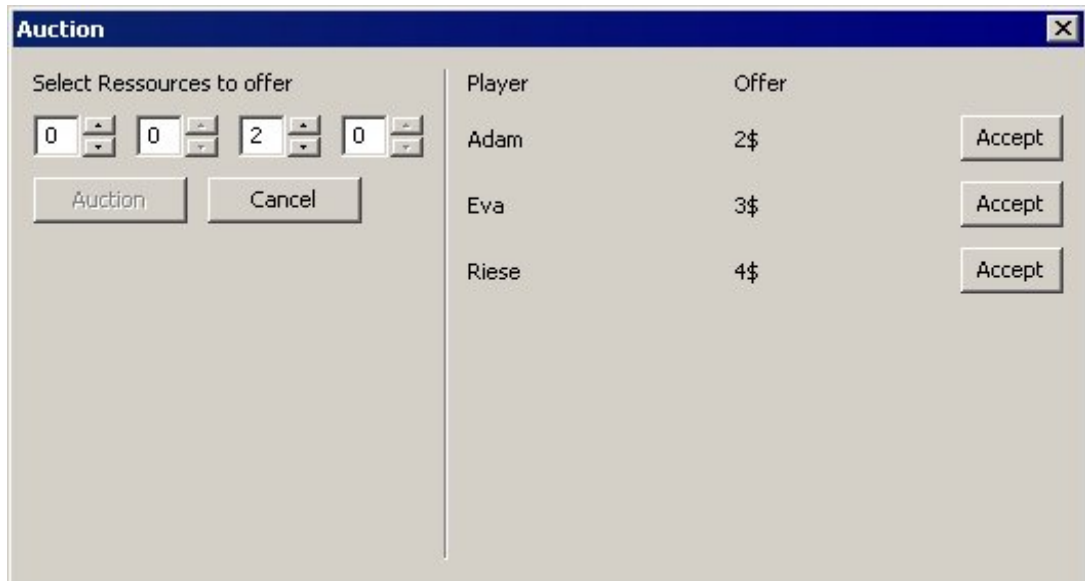


Illustration 11: Auction dialog

This method turned out to be too inflexible, as the seller could not change his offer after the auction started. As each auction would use up one action, this could be somewhat frustrating if it turned out that a player would have been interested in a slightly different selection of resources. However, using the same mechanic as for bail offers would have required extensive changes to the AI and creation of events for the module.

An attempt to **steal** could be made, where the player would decide who to steal from and which resource. He could also steal a random resource, which had a higher chance of succeeding, but might not result in the required resource. As he couldn't see what kind of resources the target had, it also avoided picking an unavailable resource. The chance of success for randomly stealing was 50%, while that of stealing a certain resource was only 25%. In both cases, the chance to be caught was 20%. This would of course always mean failure to get anything. Each additional guild would increase the chance of success by 25%/50%, while each tower of the selected target would consecutively half this number. Additional guilds would decrease the chance of being caught by 20%, each tower would increase it by the same amount.



Illustration 12: Steal dialog

$$success : guilds \cdot chance \cdot \left(\frac{1}{2}\right)^{towers} \quad caught : 0.2 - 0.2 \cdot guilds + 0.2 \cdot tower$$

Getting caught in the process would mean the immediate end of this turn and landing in jail until the end of your next turn. However, you were still able to take part in auctions or partake in other actions initiated by other players.



Illustration 13: Peek dialog

As other players resources were invisible, towers allowed to **peek** at them. This was mostly useful as a preparation for stealing. For each tower, you could select one player and get a list of his current resources. Originally, a tower allowed to always see the targeted players resources and the action would only be required to change the observed players. This in combination with the protective function of the tower made it too strong an option. So it was changed to just allow a quick glimpse and as a side effect make the sweatshop more important in combination with the guild.

If auctions would fail too often or a player did not want to risk others getting required resources, he could trade if he had a harbor. Shipping cost and trade tax must be paid, so an auction was usually a cheaper way to attain resources. The shipping cost is a fixed value of 3 that applied to the whole transaction, but could be reduced by building multiple harbors. Each additional harbor would reduce this by 1. The trade tax started at 75% of the value of the bought resources and each additional harbor would reduce it by 25%. So a player with a sufficient number of harbors would be able to trade his resources with reasonable extra cost. It was possible to sell resources of more value than one actually bought. This was mostly for convenience if the exact value couldn't be reached. Unfortunately this option turned out to be a little too important as succeeding auctions were quite rare

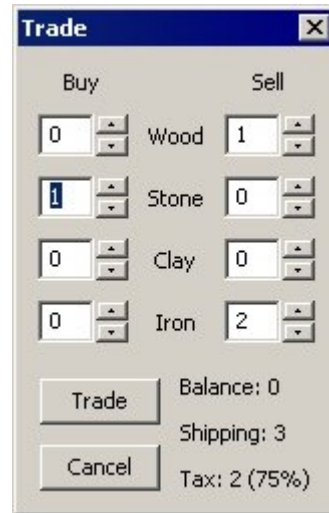


Illustration 14: Trade dialog



Illustration 15: Bail dialog

Players in jail could be **bailed** out by other players. For this, the player making the offer of freeing him would select resources he wants in return. The arrested player could make a counter offer until one of them would accept the others offer. If they agree, the transaction is made, but there is a 10% chance of the rescuer being caught as well. So a player needs to consider if the gain in resources is actually worth the risk of losing a full turn as well.

Finally, a player could **conspire** with other players to destroy someone's building. He would select a target player and building and who he wants to ask for help. Each of these players will be informed and can either deny to help or pay any number of resources to assist. Depending on the option used, it would either require double of each resource that the building cost or twice the total value. Note that walls were

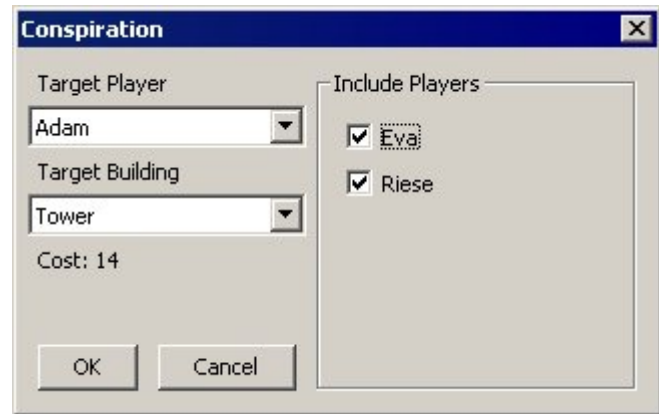


Illustration 16: Conspiration dialog

particularly expensive and that once they were built, destroying other buildings would not keep a player from building his palace and winning the game. This made it important to keep track of the other players buildings and sabotage them before it was too late. The target player would afterwards be informed about who participated in the conspiracy, to allow for emotional reactions towards these players.

After noticing that most auctions would not be successful, it became apparent that often the offered resources weren't needed or the requested resources could not be spared. As a solution, cash was introduced and optional rules allowed to convert all resources to cash when selling or offering them in auctions. So, if a player would sell two units of wood and one of stone, a player successfully bidding for this and offering resources of the same total value would receive four units of cash and so would the player that started the auction. In addition, a new action would be available to **buy** resources without any extra costs. The result were more confusing auctions, as the on-the-fly conversion would create auctions where essentially a certain amount of cash would be traded for the same amount of cash. Also, it opened a simple back door to avoid being stolen from by trying to have cash instead of resources. Fixing this would have required changes to the whole stealing process.



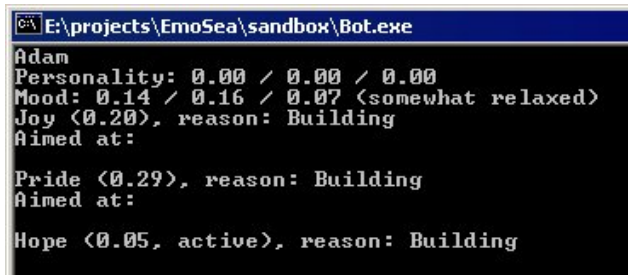
Illustration 17: Buy dialog

If no other action would be of use, selecting **done** gave up any remaining actions and ended the turn.

7.1.4 Adding AI and AE

The plan was to first write a pure AI player for the game and later expand it by using the emotion module. For this, the AI would rate all possible actions based on the current situation and pick the highest rated one. Adding the emotion module required two steps. One was to translate everything that happened into events that could be processed by the

emotion module, the other was to adjust the AI generated ratings depending on emotional influences. This appeared to be a sensible solution, as most decisions are still based on a logical examination of the situation and emotions will only play a role when multiple possibilities are equally good choices. Only very strong emotions would be able to cause really bad decisions, though in real life it would be more realistic to have them affect the analysis as well. How this is handled is entirely up to the programmers AI implementation.



```
E:\projects\EmoSea\sandbox\Bot.exe
Adam
Personality: 0.00 / 0.00 / 0.00
Mood: 0.14 / 0.16 / 0.07 (somewhat relaxed)
Joy (0.20), reason: Building
Aimed at:
Pride (0.29), reason: Building
Aimed at:
Hope (0.05, active), reason: Building
```

Illustration 18: Internal state of a bot

Some information needs to be collected to make decisions. The missing resources to construct the currently selected building and estimates about the other players resources are the most important. Estimates about other players resources are updated whenever the bot uses a peek action or a

transaction including a known number of resources is taking place. These estimates are conservative and always the minimum the bot can be sure of. So if the number was unknown before and a player receives two units of a resource, the new guess would be two as well. Values smaller than 0 are treated as unknown, so whenever a player spends more than he was suspected to have the state automatically changes to unknown.

The **build action** is judged by rating the separate buildings, based on different factors with different weights. If using emotions, a second step is calculating an adjustment value that will be clamped to the range [-.5,.5] and added to the rating. With little guidelines, these weights are selected rather arbitrary and would need to be tweaked later on after extensive testing.

Guilds are more useful if the player already has some, as it improves the chances, while at the same time, the average number of towers of other players is making guilds less attractive. If the player has harbors and can trade in the required resources and/or doesn't have to rely on other players, this also makes guilds more interesting, while they become less helpful if the player already has accumulated many resources and hence little need to steal more.

$$0.1 \cdot playerGuilds - 0.1 \cdot averageTowers + 0.1 \cdot playerHarbors + 0.5 - \frac{resourceValue}{40}$$

The adjustment for emotions uses the fear of failing to steal. With working long-term prospects that would mean an increased likelihood of building more guilds if stealing fails relatively often. Another factor is the current moods dominance. A character feeling in control will act more aggressively, which includes buildings that support this style of play. The same goes for experienced anger, while the agents agreeableness and conscientiousness are subtracted, as stealing is considered wrong and no help in getting along with other players.

$$fearStealFailure + moodDominance + anger - agreeableness - conscientiousness$$

Importance of **harbors** only depends on the number of resources and the number of manufactures, as these produce more random resources that might have to be traded.

$$\frac{resourceValue}{60} + 0.2 \cdot playerManufactures$$

Harbors are adjusted depending on the fear of unsatisfying auction bids or auction failures. The current dominance is subtracted, meaning that positive dominance supports interaction, whereas negative dominance promotes playing in a more independent fashion instead of relying on other players or bothering them with offers and requests.

$$fearOfAuctionFailure + 0.2 \cdot fearOfBid - moodDominance$$

Like guilds, **manufactures** are of little worth, if the player already has many resources at his disposal. However, having many harbors means receiving more resources to trade guarantees getting the needed resources for a building. Finally, the average number of towers means manufactures are preferable to guilds to avoid the risk of getting caught.

$$0.5 - \frac{resourceValue}{40} + 0.1 \cdot averageTowers + 0.2 \cdot playerHarbors$$

There are no adjustments for manufactures, though using dominance in the same fashion as for harbors might make sense. However, the scenario was dropped before all factors and possible adjustments were considered.

As **sweatshops** are allowing more actions, they become more useful, the more guilds the player has, while other players towers make them less appealing. Own towers only support this decision in case of at least one existing guild, even though technically more actions would allow peeking at other players resources more often. However, looking at their resources is typically not useful without the possibility of actually acting on this information. Also, sweatshops don't grow more effective the more you build, so they get less important the more you already have. However, this point would depend on the number of guilds or rather the chances of being caught. If they are for example 50%, having 5 actions and spending them on stealing would be very useless, as most of the time you would get caught and have your turn end before getting to use all of them.

$$0.3 \cdot playerGuilds + 0.1 \cdot playersTowers - 0.15 \cdot averageTowers - 0.2 \cdot playerSweatshops$$

There are also no emotion adjustments for sweatshops.

Towers are mostly important for defense against theft, so the important factor is how many guilds the other players have. At the same time, they give information that is useful for stealing, so the own guilds make them more worthwhile as well. In addition to that, the more players are in the game, the more towers you need to look at their resources.

$$0.2 \cdot allGuilds - 0.1 \cdot playerTowers + \frac{numPlayers}{16}$$

Only the fear of other players stealing from the agent is used to adjust the value, though for example openness could be equated with curiosity.

1.5· *fearOfTheft*

Finally, **walls** and **palace** are automatically the most important buildings, as soon as it is possible to construct them, so they are simply set to sufficiently high values if available. When rating the build action itself, all buildings that can be built with the currently available resources receive a bonus of .25.

The priority of **bailing** out another player depends on which resources the agent needs and which he expects the jailed player to have. Each resource the player has is multiplied with its value and added up. Resources also needed by the agent are receiving a bonus of 50%. Finally the result is scaled down. This is done for each player currently in jail.

$$\frac{\sum (resource \cdot value) \cdot [1.5]}{35}$$

While it would seem obvious to use the sympathy towards the other player, there is actually the issue that a well liked player would be made an offer out of sympathy, while a disliked player would be made an offer to exploit his situation. So eventually both cases would result in roughly the same decision. Personality traits or mood could have been used in regard to the small risk of getting caught, too.

To determine a rating for **buying** resources, the value of all missing resources is added up. The larger the difference between available cash and needed cash, the higher the action is rated. However, no resources will be bought, unless the cash suffices to buy all required resources at once.

$$0.5 + 0.1 \cdot (playerCash - \sum resource \cdot value)$$

Also no adjustments for buying. Strictly speaking it would always be the best option if a player has enough cash and actions to immediately spend the bought resources.

Stealing is more complicated to rate. First, it is determined how many different resources are needed. If this number is larger than half the different resources in the game or zero, stealing at random is the better choice. Else the chance to succeed and the danger to be caught are considered. The former receives a bonus, if the other player already has enough building to construct walls, as this means it is important to sabotage his progress. If the agent doesn't need any resources, the rating is further reduced by 25%. Only resources where the agent does not already know that the target has none are considered. This is done for each other player.

$$stealChance - 0.25 \cdot bustedChance \cdot [1.5] \cdot [0.75]$$

Multiple emotional factors affect this rating. Sympathy is lowering it, while anger at the

target increases it. Like with guilds, conscientiousness and agreeableness reduce an agent's willingness to steal in general and so does the fear of the attempted theft to fail.

$$-sympathy + angerAtTarget - conscientiousness - agreeableness - fearOfFailure$$

To determine the rating for **auctions**, it is necessary to look at the excess resources as well, as they are an obvious requirement to start one. The resource that is needed the most as well as the cheapest resource is determined. Only if the value of excess resources is larger than the cheapest required resource does it make sense to have an auction. Each required resources probability is scaled down and added together unless a player is known to be in possession of it. In this case, the number of resources in this player's hands is used.

$$\sum [0.15 \cdot estimatedResources \vee 0.025 \cdot resourceProbability]$$

The rating is affected by fear of the auction failing, hope to see acceptable bid and the characters current dominance in his mood. Again, a dominant character is supposed to interact more with other players.

$$hopeBids - fearAuctionFail + moodDominance$$

Trading is similar to buying, but not automatically the best option because of extra costs. The balance is calculated by determining the difference between the value of excess resources and required resources, including shipping cost and trade tax. The action is discarded, if not all needed resources can be bought in one transaction. Else the rating depends on how many resources would be left.

$$0.2 + \frac{0.75 \cdot (excessValue - neededValue - tax - shipping)}{10}$$

Fear of an auction failing is encouraging trading, while hope for good bids and a feeling of dominance are reducing its rating.

$$fearAuctionFail - hopeBids - moodDominance$$

The **peek** rating for each player simply depends on how little is known about his resources. The more unknowns, the more important it is to change that. More towers mean more information can be gathered in one action, so this also increases the rating. Each unknown resource increases the rating by a fixed amount, but is divided by the number of different resources in the game, so even if all resources are unknown, the value can't be larger than one.

$$\frac{unknownResources \cdot 0.2 \cdot playerTowers}{4}$$

No adjustments are made for this action. As many of the highest rated players as the number of towers allows are peeked at, if this action turns out to be the highest rated.

Finally, how much an agent values conspiring depends on the number of buildings of other players and himself. The more buildings someone is ahead of him, the more important it becomes to destroy his buildings. If this opponent is already able to build walls, there will again be a bonus of 50%. Having unneeded resources also improves the rating, while already lacking resources to get your own buildings decreases it. Ratings are calculated for each other player.

$$0.10 \cdot (\text{otherPlayersBuildings} - \text{playerBuildings}) \cdot [1.5] + \frac{\text{excessValue}}{50}$$

Anger at the conspiracy's target and fear of this player being able to build again soon will improve the rating, hope to build something yourself decreases the need to hinder someone else's progress instead of concentrated on your own.

$$\text{anger} + \text{fearOtherPlayerBuilding} - \text{hopePlayerBuilding}$$

Some situations required additional decisions by the AI, namely auctions, conspirations, bail offers and trading. When trading, the AI would sell excess resources and buy required resources. The rating process already guarantees that in this step the needed resources can be afforded, so the AI simply removes as many resources from the sale as possible without getting below the price of the transaction, beginning with the most valuable.

When starting an auction, the AI would compare the value of the cheapest required resource with the total value of all surplus resources. This was supposed to avoid pointless auctions where nobody would have reason to make any bids, as the AI would usually already make the same offer over and over again in every turn. The reason this was not blocked completely is that each player might have received resources in the meantime that allow to finally make a bid, so not trying again would not have been entirely logical. Long-term prospects, namely the fear of an auction failing, were supposed to prevent that.

Also, the first implementations would always offer all unneeded resources, yet expect offers of equal worth. As a result, with each turn it would offer more and more resources, making it less and less likely that another player would have enough of the wanted resource to make a sensible bid. This was changed in a way to let the AI reduce the offer to the same value as the needed resources in the same way that is used for trading.

Whenever a bid is made, the AI compares the total value against the value of his own offer. If the bid is at least as high, the offer is taken into consideration. When using emotions, the bid is adjusted by the sympathy for this player. This can make a difference in the range of [-3;3], so high sympathy means accepting a bid that would otherwise be below an acceptable value. After waiting for a set time or if all players dismissed the auction, the auction is either aborted by the agent or the best offer including the sympathy modifier is accepted.

On the other hand, bots deciding if they should make a bid are first checking if they need the offered resources and if they can spare the requested resources. After the introduction of cash, only the latter consideration mattered. If they have no resource to offer, they will dismiss the auction, otherwise they determine a maximum bid they are willing to make. As before, this is adjusted by sympathy, meaning they are willing to pay more if they like the auction holder. The same method as for trading is then used to reduce their offered resources to a selection with a total worth equal or below the set limit.

An obvious problem is that offers and accepting offers are based on nothing but equal value and sympathy, where a convincing AI would need to have some kind of urgency for an auction to work or a foreign auction to fail. This leads to very predictable behavior.

In conspirations, most work was already done when deciding on which action to take. As there can never be too many players in a conspiracy and adding resources to the pool, everyone but the target is asked for help. The decision if an agent is willing to take part or not is based on the same factors as that of the initiating agent. The difference in buildings is adjusted by two times the sympathy, so a liked player can have up to two buildings more and the agent would still refuse to help. If the target is sufficiently ahead to be considered a threat, the bot will as usual start with all spare resources and consecutively reduce them until the value is below the required value. This value is the cost to destroy the building minus the resources already offered by other players. Note that there is no mechanic to keep a single bot from paying the whole bill, which means that who is last to make his offer will usually get to pay less or even nothing. Implementing realistic behavior, especially based on traits that could be considered as greed or avarice would have made this even more complex.

The same mechanics are used for bail offers. A maximum to the requested resources value is determined by calculating the average value of resources the jailed player would receive in his turn and adjusted by three times the sympathy. Starting with all currently required resources, this selection is again reduced until it is just below the set maximum. After the initial offer, the agent will wait for a reaction, either accepting or refusing or a counteroffer.

A counteroffer will be evaluated by determining its total value and comparing it to a minimum offer, calculated the same way as the maximum for the initial offer. There is one difference in that offered resources that aren't required will increase this minimum by one. It's however easier to imagine the value of useless resources being reduced instead. If the value is still above the determined minimum, the counteroffer is accepted.

The agent in jail will do almost the same. He determines which resources he can spare, calculates a maximum he is willing to pay as above and either accepts or makes a counteroffer. Only if he has no resources to spare or his limit is set so low that every possible offer would be beyond it will he refuse completely.

So far, we have only looked at one direction concerning the module, which was also the more complex task of the layer between application and module. Little can be changed

about that, without adding a solution to let the module have abstract context information and take over the decision making process. It's questionable if this would be a good idea, as decision making is always a combination of reason and emotion. Also, an application's AI usually already has all the necessary information and uses whatever format is appropriate for the task at hand. Redundantly having to feed the module with the same knowledge would most likely impose limitations, add overhead and be more difficult than using emotion information from the module. At one point a solution was considered, where for each action a set of influences and weights could be defined and the module could be queried for the current rating. However, this would be exactly what was done above, just with more overhead to store the sets and weights and less flexibility than writing a function for each action.

7.1.5 Handling events

The other direction is a more straightforward task, as events in the application just need to be analyzed in terms of OCC criteria to construct events for the module.

The first approach turned out to be too high level. There were no goals at this point, so every event consequence would instead be passed the relevance and desirability. But this was done for each event as a whole, resulting in lots of often redundant code. Later it was broken down into more elementary parts, in this case getting or losing resources and getting or losing buildings. However, the hierarchy of goals turned out to be very linear, as to win the game you have to build and to build you need to gather resources. So each of these goals can essentially be reduced to a single goal. Yet, two goals were kept, building and winning.

The desirability of getting resources was based on their value, but also rated 50% higher if the increase resulted in enough of the resource for the planned building. The sum was scaled down to a tenth. Obviously, in some situations the desirability could end up higher than 1, whenever the total value was above 10. This is one of many examples, where no sensible way to limit the values to the correct range could be found. Clamping the value would result in getting more resources not rating higher than getting fewer resources. Not very realistic, though it might make sense to use a logarithmic function, where the increase of desirability is growing smaller for higher numbers. In this case, it was simply not likely to get that many resources in one step and in addition to that, numbers above 1 don't break anything are scaled down by relevance, anyway.

Losing resources was handled in the same way, except the rating would be 50% lower if after the change in resources a previously possible building becomes unavailable. The reason there is no constant modifier is because different resources are not equally easy to get, so it made more sense this way, as now lacking a rare resource would throw you back more than a common resource.

Getting a new building is rated according to its price and a comparison of the agents number of buildings with the highest number of buildings of other players.

$$totalValue \cdot 0.035 \cdot (1.1 + (maxBuildings - ownBuildings) \cdot 0.1)$$

If a building was destroyed, the evaluation gets more interesting. Again, the total value of the building is used, however, if the agent already built walls and a different building was destroyed, the rating is reduced to 25%, as it will not stop him from building a palace to win. Also, if he does not have walls yet, but still has enough buildings to construct them, the rating is reduced to 50%. Otherwise, the same formula as above is used.

All event evaluations are broken down into changes in resource and buildings if possible. Using **theft** as a detailed example, an object of type `EmoEvent` is created with the name *Theft*. If the evaluating agent himself is the thief, it is named *Stealing*. This is necessary, as the module only has name strings to differentiate the events and prospects can only work if there is a difference between fear of being stolen from and fear of failing to steal. Technically, fear of failing to steal actually means fear of the event *Stealing* with a negative consequence for self.

Next, consequences are added for the thief and the victim, affecting their goal to build. In case of success, desirability is determined as described above based on the change in resource. Else, if the thief was caught, desirability is set to -0.7 or if it simply failed to -0.1, as he still wasted an action. The thief is added as cause and the event is sent to the character to be processed. Note that in this scenario an older version of the module was used and no factors like responsibility or publicness came into play.

The previously described way to determine desirability is used when a player constructs a **building**. However, if another player than the agent built it, the consequence for the agent is calculated by using the above function for losing a building and reducing the result to 20%. This is because in relative terms it doesn't matter if he got or you lost a building. The player who constructed the building is used as cause.

Auction **bids** can't completely rely on the same functions, as they are only potential changes in resources. In this case, 5% of the difference in values between bid and offer are used as desirability and the player making the bid is considered the cause. After an auction ends, the winning agent will use 7.5% of the difference in value and add a constant value of 0.15, with the auction holder as the cause. The constant value is used, as winning the auction at all is considered to be desirable in itself or the agent wouldn't have made his offer.

Similarly, **bail offers** compare the average value of collected resources per turn with the requested resources and use 10% of the difference as consequence for both players. This may not seem to make much sense at first, as the player offering to free someone doesn't have to make that comparison. However, to him it is not desirable for other players to get resources, so this difference can actually be applied for him as well if it is simply negated. Counteroffers are handled the same way, except with the jailed player as cause.

No events were caused for **conspiration** invitations or refusals to take part and only the consequence for the victim were considered with each player adding resources for the

destruction as the cause. Here, the spent resources of each player should have been considered as well and it becomes apparent that the lack of factors like responsibility make it impossible to blame the players to different degrees.

It can be seen that even a rather mechanical task like creating events for the module require many arbitrary factors and sometimes not everything happening in a game can be broken down into elementary events. Using the module in a complex context can cause considerable extra work, not just in terms of more code, but also in terms of finding the right factors or determining a working scale to get a convincing result.

7.1.6 Evaluation

Originally it was intended to use the EmoSettlers scenario and test persons to evaluate the module. As the bots playing style wouldn't be distinct enough to draw conclusions about their emotional state, there were two options taken into account. A graphical display using a 3D avatar or chat messages. An avatar was already implemented using simple vertex interpolation to morph and blend between expressions, but the interface of the client was already cluttered enough, plus the conversion from OCC or PAD to the six basic emotions would have posed another problem. Some of them exist in OCC as well, others are split up into multiple emotions, and for example surprise, if considered a reaction to an unexpected result, would mostly apply to disappointment or relief. But the actual expression one typically finds in animation packages hardly matches either. On top of that, the interface would have had to allow a human player to set the expression for his own avatar efficiently enough to not give away who was a human or a bot.

So instead the bots were able to send chat messages when an emotion was triggered. To do this, each emotion had a few predefined messages for different intensities of which the bot would pick the best match. For tests where the probands would have to tell the difference between a bot and a human player, it was necessary to limit all players to the same selection of chat messages, so the repetitiveness wouldn't have given it away. Other difficulties would have been the timing, so the bots would have needed delays for all actions with random variations.

7.1.7 Problems with EmoSettlers

It was discovered too late that the scenario had constantly grown in complexity the more actions, buildings and game options were introduced. As a result, AI decisions were often hard to understand as they were based on too many arbitrarily selected and weighed factors, not to mention often required different paths to deal with different options. Also, all decisions were based on nothing but the current game state without any kind of long-term strategy, which could quickly result in inconsistent behavior. But to be fair, I often displayed the same inconsistencies as too many options blurred the lines between different strategies.

During the development of the scenario and pure AI, balancing was an issue left for later, without realizing the growing heap of factors and values becoming more and more unmanageable in time. The eventual effort to tweak all the numbers and factors in the game was gravely underestimated and so was the required time to extensively test the game. So the scenario was dropped rather late and replaced by a last minute scenario, where the previous mistakes were avoided.

What could be learned from this is that actually merging a game's AI and the module can be a tricky task. One quickly tends to overengineer the problem and includes all kinds of factors for each decision in an attempt to make it as realistic as possible. But even without making things too complicating it shows a fundamental problem of the module. Constructing events for the module is relatively easy, but it doesn't offer any guidance when it comes to acting on the created emotional state. As a result, the agents behavior can quickly depend more on the specified weights for different factors than anything else.

Without knowing in advance which intensities of different emotions will actually occur, it is hard to decide on the right numbers. If intensities are relatively large, emotions might drown out reason, which is comparatively harmless compared to the opposite. Overly emotional reactions might appear strange, but are usually experienced as believable. If emotions don't make much difference it is typically feeling much more wrong. A player that has everybody stealing his resources is expected to either build towers for protection or return the aggression. But not reacting at all will look very unrealistic. Unfortunately, the intensities and hence the correct weights depend on the event definitions, resulting in a complex system. For a complex scenario this might require too much work to be justified. Again, *broad and shallow* seems to be the way to go.

7.2 SIMPLEX (Simulation of Players Emotional Experience)

The new approach was very different. Instead of trying to create a game that would actually be interesting to human players and adding more and more options and features, the simplest possible scenario that could still trigger all possible emotions was conceived. Compared to the old scenario, one could say that it was reduced to one abstract resource and one possible action. The new resource was squares on the board and the single action was stealing a square adjacent to one of your own. An addition had to be made to allow for explicit cooperation, so before an attempt to take over a square is made, the other players can be asked for help to improve chances of success. However, if he disagrees, the chances are reduced, so it should be considered who to ask and who not.

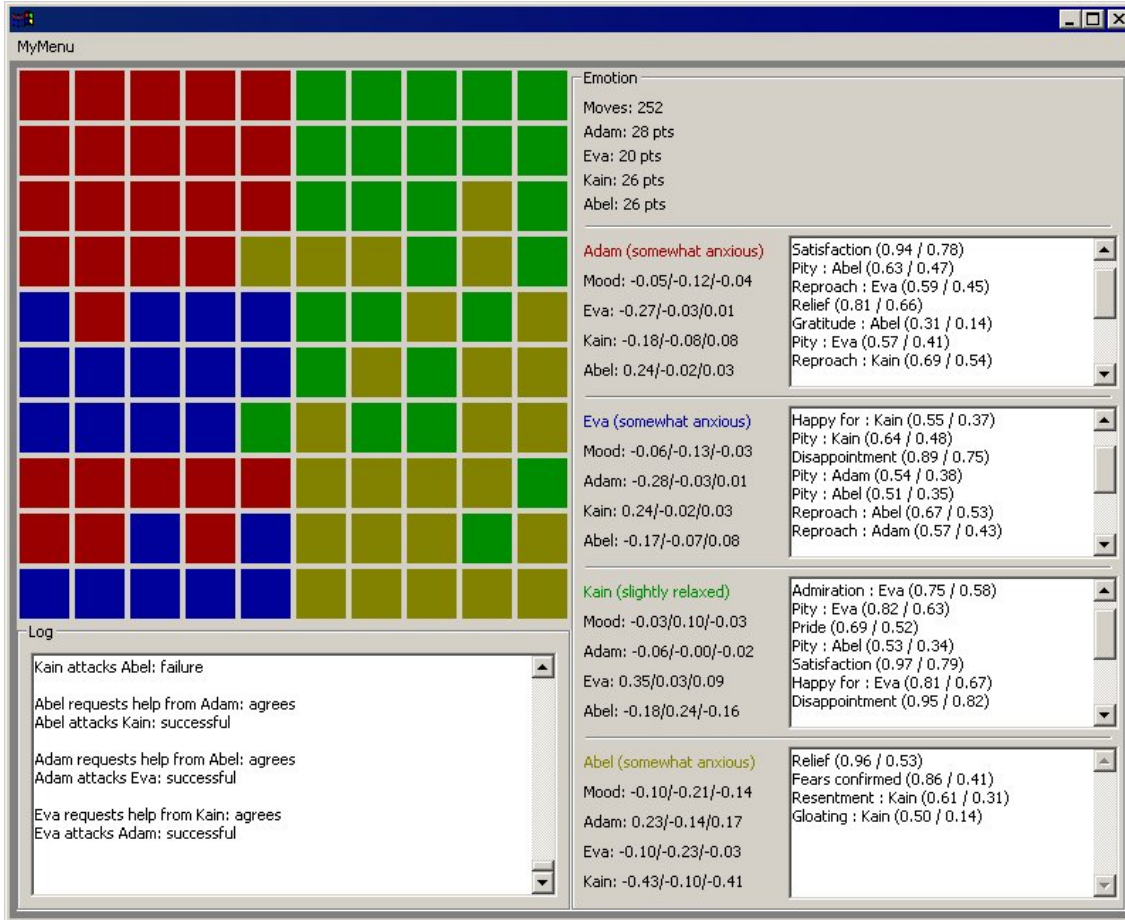


Illustration 19: SIMPLEX

7.2.1 AI and AE strategies

Even here, the AI would have been possible in many ways. The first version picked a random square in reach, but that's not how a real player would act. However, one player might attempt to be fair and attack each player in turn, while another would decide on one target and stick with it until the end. Eventually, the AI was set to always pick the player with the least number of squares to eliminate him from the game as quickly as possible in an attempt to reduce the potential attackers. The only exception was a player with more than 80% of all squares. Such a player would be considered close to winning and then be attacked instead.

The decision to ask other players for help is purely based on their previous responses. If they were agreeing in 50% of all cases, they are asked again. Before they have been asked at least five times, this statistic is ignored and they are always asked. The response in turn depends on the number of squares of the affected players. Typically, a bot would always agree, while the attacker had less squares than he himself. Else, only if the victim had more squares than the attacker would the player get involved to level the playing field. The attacker would technically be in a position to already know if another player was willing to help, so it would be simple to only let them ask when the

answer was known to be positive. However, allowing the AI this kind of clairvoyance would have skewed the results in an unrealistic fashion and the lack of time meant keeping it as simple and straightforward as possible. This also included the fact that no human player could join the game and the evaluation by probands was dropped in favor of collecting statistical data. Eventually, this game would be extremely boring and tedious to a real player, anyway, especially considering how long a game could last.

Unlike the first scenario, the emotion module is not used to affect AI decisions. Because of the simple nature of this new game, the emotion module alone could play the game and allow a direct comparison between AI and AE. Events were still evaluated by the module, but all decisions were based on sympathy. The least liked player would be attacked and only players with positive sympathy would be asked for help or supported.

Obviously this is a very simplistic AI and there are no long-term strategies like cutting off other players, neither does it keep track of who attacked whom how many times. An infinite number of modifications could be imaged, like highly conscientious players distributing their attacks evenly in an attempt to play particularly fair. However, as the first scenario demonstrated, trying to consider too many factors can quickly grow out of hand and cannot just fail to make the agents behavior seem more realistic, but instead achieve the opposite if the factors aren't weighed correctly.

7.2.2 Event evaluation

To prevent odd emotions, the events had to split up in a sensible fashion. Target selection and outcome could not be the same event, as the chances don't depend on who is attacked and would cause a false causality that could create regret over attacking a specific player, even one player had to be picked and the choice wouldn't have made any difference. Again, desirabilities were mostly determined arbitrarily and tweaked later, though one guideline was that with 100 squares in the game, winning or losing one would be 1/100 desirable or undesirable. However, this turned out to result in values that were too small to regularly cause emotions.

When **selecting a target**, the cause is obviously the acting player. By now the new factors had been introduced, so while the responsibility for making this choice and the publicness was 100%, the action is only rated 20% unexpected, as the rules of the game require this choice to be made. This prevents an unreasonable amount of blame. The consequence of this choice is 30% negative for the targeted player, as he is now in risk of losing a square. At the same time, it is 10% positive for the other players who have not been chosen. Handling it like this is necessary to allow gratefulness for not being attacked. This event is sent to all players in the game.

For the determination of praiseworthiness a real person would most likely consider many more aspects, like how many times in a row he was attacked. Being picked as a target twice before would typically let you perceive it as even worse when the third player also decides to pick you. This is just one more example of there always being a large number of factors a real human would be able to keep in mind that can't be

handled in software.

Asking for help is the next event. When a player is asked for help, hope and fear are first created. Hope of a positive response, with a probability based on the percentage of previous agreements and expected to be 20% positive. Fear of the other player rejecting has the remaining probability and would be 20% negative. Again, these one-shot prospects need to be created manually, as the module lacks the kind of information to do it automatically. The event of asking and the response are much the same. In fact, the only reason to not combine them into one event is being able to differentiate based on the event name. Both have consequences that are 20% positive for the attacker and 20% negative for the victim or the other way around if the request was denied. Only the causes vary, as once it is the player asking and once the player accepting or rejecting the offer.

Right before the **attack**, more prospects are created for the attacker and victim. Probabilities can be easily calculated as 50% +/- 10% per ally. Winning or losing a square is rated at 50%. The attack event itself does not have any player as a cause, as the outcome is independent of that and the influences have already been processed separately. Only the consequences are used, which are 50% positive or negative if the attack succeeds. If it fails, it is considered 25% negative for the attacker, who could not use his turn to his advantage, but also 15% positive for the attacked player, who did not lose anything. In hindsight of his goals, this raises more questions. Technically he didn't get closer to his goal, so this can't be considered desirable. But logically it often makes sense to treat the absence of a negative consequence as positive in itself.

No blame for a failed attack goes to those who rejected to help, neither is there gratefulness to those that agreed. That's the downside of splitting up the events and a general problem of having the correct causalities.

Certain changes were made to reduce the number of emotions existing at the same time. As the decay time is usually longer than the time it takes for one turn, there would often be older emotions that made it hard to keep track of which emotion was the result of the last turn. So the time passed to the update function was large enough for all previous emotions to completely decay in the meantime. Unfortunately that also means the mood will usually have more than enough time to return to its default value determined by the personality.

7.2.3 Evaluation and results

After the first scenario was dropped, the evaluation process was reconsidered as well. As the new scenario was no networkable client/server application, it would not have been possible to set up a test where it was unknown whether a player was a bot or human. Instead, data was collected from thousands of test runs. The first batch consisted of one thousand games with four bots using the emotion module. Another thousand games would mix two emotional bots with two pure AI bots and the final batch was thousand games of four AI players.

To collect the data, every time an emotion was changing from inactive to active, the callback function would increase the counter for this emotion. However, as the length of the game would greatly vary, especially between the set using emotion and the set using AI, the results were divided by the number of turns. The game would be aborted, when only two players were left, as at this point the game could last forever. Other data collected was the number of attacks that were executed by a player alone and those where he requested help. Also, the percentage of denied requests and the number of turns until the first and second player were out of the game were gathered.

As the GUI was completely unnecessary for the data collection and causing too much overhead, a stripped down version without any interface was written to dump the results in several log files.

Multiple test runs were made and compared to make sure the results within the different groups were consistent and stable. While the different test groups showed significant differences between each other, the groups themselves did not.

The results of the AI games weren't very interesting, as their behavior could easily be predicted from the code. The first player to lose a square would always be attacked by all other players, so games would be over very quickly. The emotional bots also always fell into the same way of playing, though in their case this happened naturally and was exactly the kind of emergent behavior that the module was implemented for.

Each player would end up with one liked ally and two disliked players, one of them usually disliked about twice as much. These relationships were completely symmetrical. So, as decisions were purely based on sympathy, that means two players would always attack each other with one of the other two supporting him. As a result, games would last a very long time as squares would constantly go back and forth between these two players. Once the first player was out, the remaining player would lose very quickly. Considering the very simple decision making rules of the application itself, the bots automatically forming teams after a few moves is a good example for the complex behavior that could emerge in more interesting scenarios. Contrasted with the previous scenario and the rather inconclusive behavior resulting from complex rules, this demonstrates old wisdom. Good systems create complex behavior through simple rules, bad systems create trivial behavior through complicated rules.

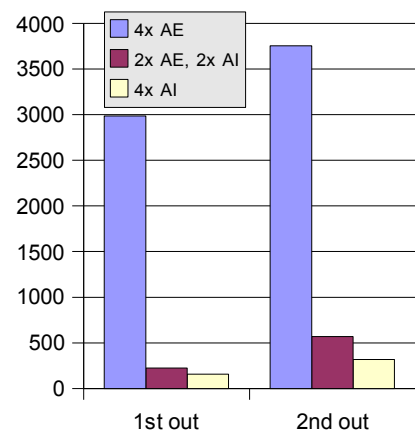


Illustration 20: Game duration

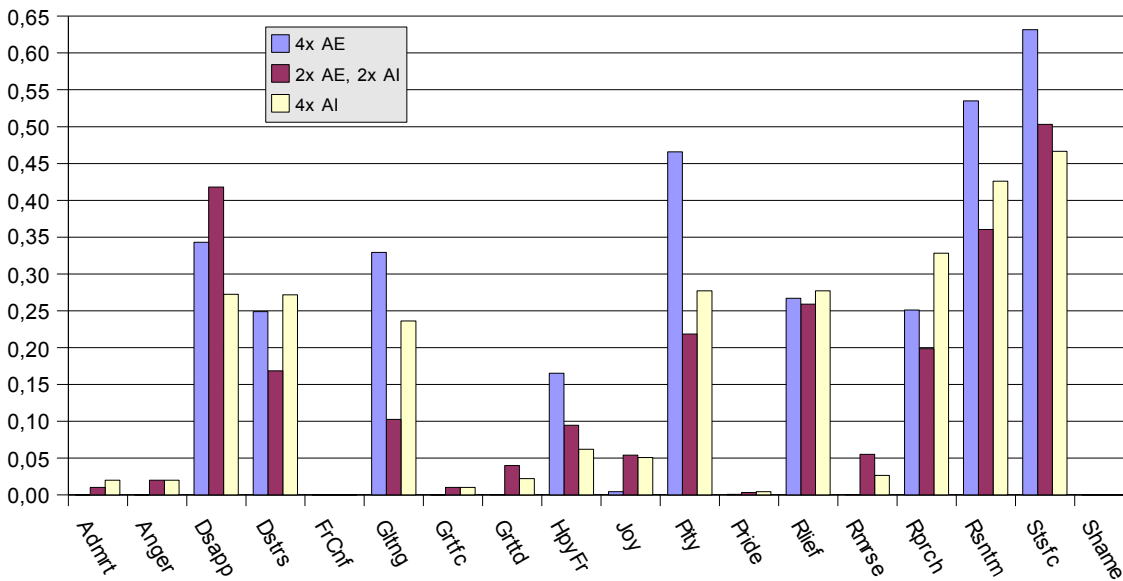


Illustration 21: Emotions triggered per move and player

It turns out that the pure AI was displaying the least cooperation, with only about 40% of the attacks being joint attacks and only 75% of the requests being answered positively. This isn't surprising. The leading player would always be denied and soon stop asking. This is also the reason why only 25% of requests were refused. Only the weakest players would regularly receive help and keep asking.

Looking at the emotional players, over 90% were joint attacks and almost 100% of them would be granted. These 90% must be taken with a grain of salt, as an attack is counted as

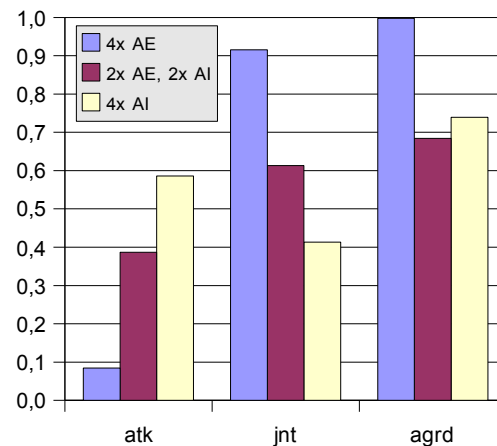


Illustration 22: Single and joint attacks

joint attack, if at least one other player is invited to join. As mentioned above, the emotional players would always have two enemies and one ally, so only one request is made each time. The remaining 10% are the result of the initial few rounds, where teams are still taking shape and most of all the final rounds, where only three players are left, one of which won't have his ally anymore and hence attacks alone. However, it is worth pointing out, that the close to 100% success rate means that sympathy was always mutual. An interesting result, as no extra code had to be written that predicts the other players answer to guarantee a positive response. Something that would have been a very sensible thing for the pure AI to optimize the chances of success.

Comparing the standard deviation for each emotion between the thousand games and test groups is showing that the emotional players are much more consistent and stable, the deviation in the number of times that a certain emotion was triggered usually being at least two times smaller than the pure AI and mixed test group. This is interesting in so far, that both, pure AI and pure AE, have a very fixed style of playing, so one would

expect the same emotions being triggered more or less the same number of times, especially since these numbers are normalized by the number of turns.

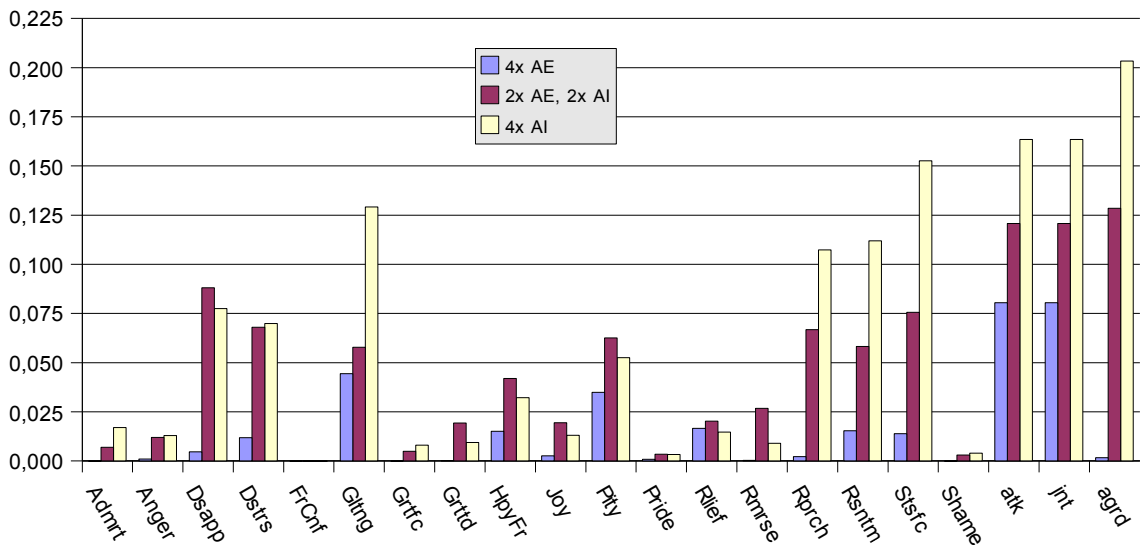


Illustration 23: Deviation in emotions

The number of times the different emotions were experienced by the different players in regard to who eventually lost or won are the most interesting, though at first irritating, results. As the emotional players fell into a very symmetrical way of playing, where each player would equally play the role of attacker, victim and supporter, it would not yield much information to differentiate between winners and losers. Instead, the way the AI players immediately focus on the weakest player is giving some insight into the different emotions between a player that always wins without ever being attacked and a player that is constantly attacked by everybody else. One would expect the latter to experience mostly negative emotions, but the results are somewhat surprising and have to be explained so as not to be discarded as signs for a faulty implementation.

With few exceptions, all emotions are experienced a lot more often by the losing player, including, for example, joy. For this, the different opportunities for each emotion should be considered. Joy is the result of something positive happening to the agent, this can be successfully taking another player's square, getting support from another player, another player being selected as target, an attack on the agent failing or another player refusing to help in the attack on him. Two of these can only happen to the attacked player and two can only happen to the player who is acting this turn. Getting support can happen up to two times and an attacker not getting support can happen twice as well. They both have about the same number of actions/events that can cause joy. At the same time, a player gets to act only once every four turns, but the victim is attacked three times out of four, meaning he has about three times more potential for joy.

Evaluation and results

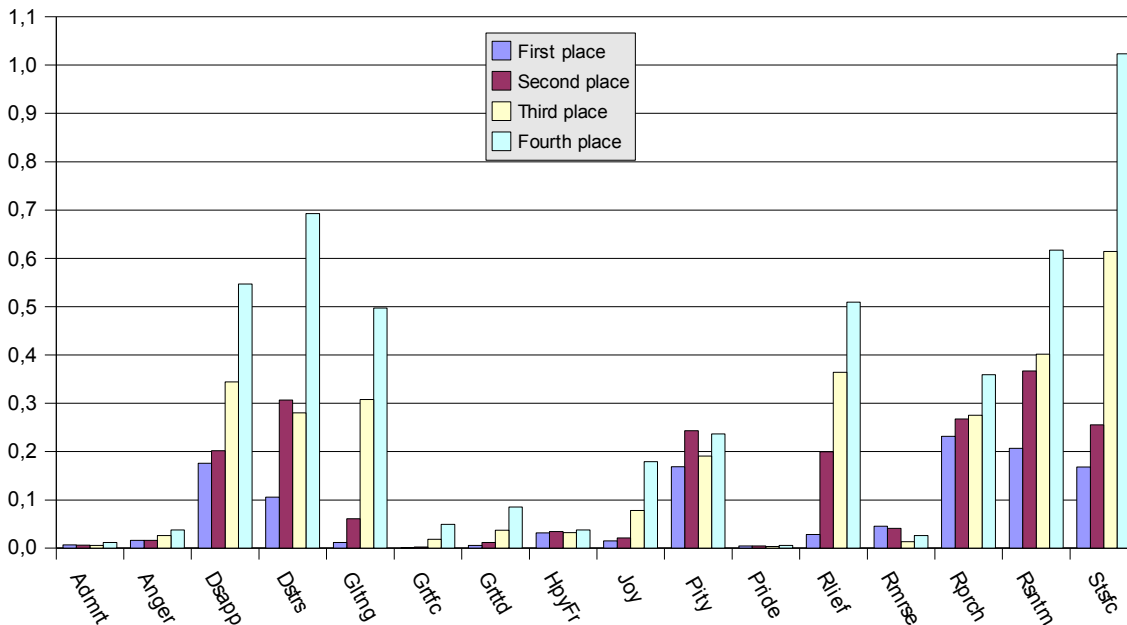


Illustration 24: Emotions by ranking

Because of the AI's consistent behavior, it's also easy to see the probabilities and numbers of refused or granted support. The leading player will not get support from anyone, the second placed player is typically supported by one of two and third place is helped by both. So, in the other players turns, there are three denials and a 70%, 50% and 30% chance of the attack failing. In his own turn however, he will also receive support by both players, leading to a 70% chance of taking over a square. There is also an issue coming from all players appraising each event, meaning the agent might also experience joy over selecting another player as the target. At first it might feel nonsensical, but could be treated as a positive reaction to finally getting to act as well.

Comparing this to the leading player, one can see that he has only two reasons for joy in his turn. The fact that he gets to select another player and a 30% of winning the square. The other players will usually all pick the weakest other player as target.

As all players had a default personality with each trait being average, meaning 0, the default mood was at the origin. With the large duration of a turn, the mood would usually not deviate from this point by any meaningful degree. This is one reason for these results. With the exception of relief and satisfaction, the negative emotions were experienced a lot more, which would normally shift the mood away from the positive emotions, increasing their thresholds and keeping them from being triggered. But as the emotions were almost all experienced in the same relative numbers to each other, the resulting moods would be similar and the results the same with negative.

For a final test, different and extreme personalities were used to see if and how much the influence the result. Only the purely emotional style of playing was used, as this usually results in a symmetry that gives each player the same experiences and events. Some numbers can grow beyond one, as the turn time and emotion decay has been adjusted to allow for mood changes as well. As a result, an emotion can be created once,

but triggered multiple times, if the mood first moves far enough to away to make the threshold too high and later moves close again, while the emotion still hasn't completely decayed.

The first personality was completely average as before. A completely conscientious and disagreeable personality was second, this one should not forgive any blameworthy actions by anyone. Third was an absolutely open, extroverted and non-neurotic person, while the last one was his exact opposite, a highly neurotic and introverted conservative.

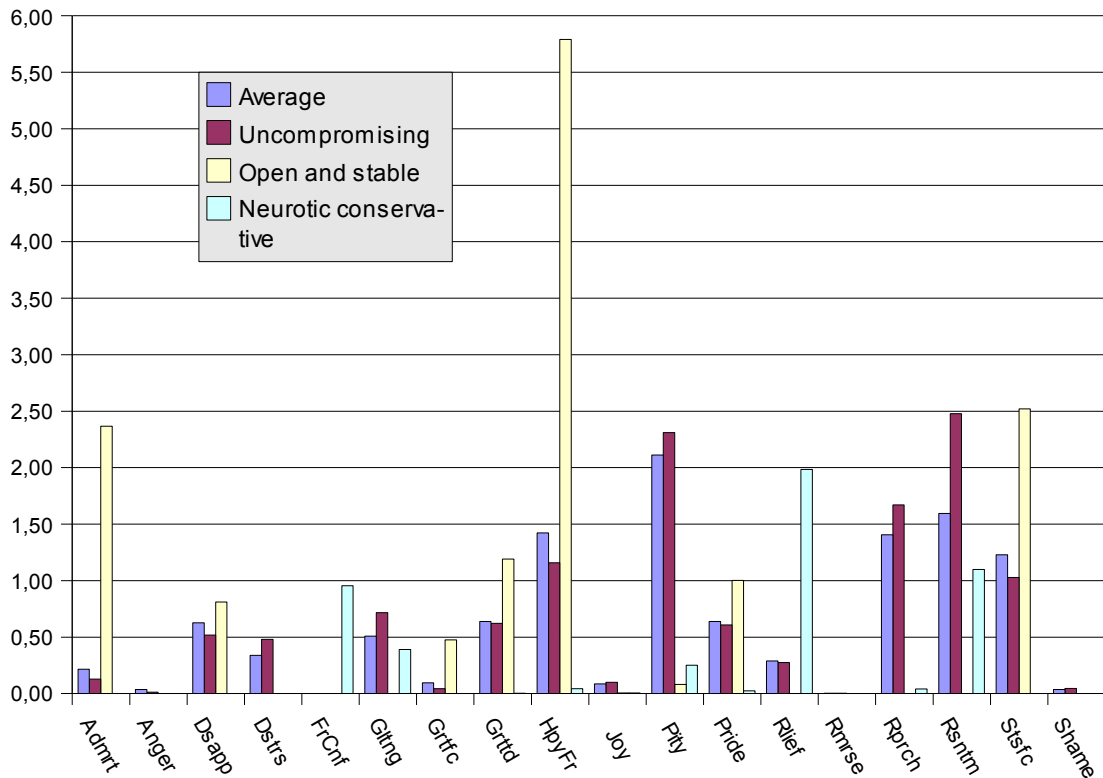


Illustration 25: Emotions by personality

So far, we noticed that the negative emotions were usually experienced a lot more often. Finally, we found a player that experienced almost nothing but positive emotions. The extroverted one. The only exception is disappointment, but as he receives a questionable bonus on his optimism, this is just a natural consequence of having high hopes in a 50:50 situation. *Happy-for* is triggered extremely often and requires to remember that the emotions are counted per move, but one move consists of multiple events. Target selection is good for two players, asking for help, independently of the answer is good for one player and the outcome will also be good for one of them. These four opportunities and the moving threshold are most likely responsible for the high result.

In the same way, the neurotic person experienced nothing but negative emotions, again with the exception of relief, as his pessimism wouldn't always turn out to be appropriate. Else he seems to be full of gloating and resentment, a spiteful one with a surprising capability for pity. His lack in extroversion might also explain why only the

most intense negative emotions came to the surface.

Unfortunately, the strict and principled one only demonstrates the unsatisfying implementation of praiseworthiness. Gloating, resentment, reproach and pity are the emotions where he scores the highest. Reproach makes sense, nothing one can do will be good enough in his eyes, so the demonstrations of dislike like gloating and resentment aren't surprising. Yet, he also scores highest in pity. It would make sense, that he pities those he finds being treated unfairly, but as no explanation based on the implementation comes to mind, as both personality factors only affect the calculation of praiseworthiness, another test run was made with the personalities distributed differently. This caused some slight changes in the results, but nothing outside an expectable variance. Yet it was enough to let the average and strict personality swap ranks here and there, confirming that with their results so close together, it is just coincidence who scores higher.

8 Conclusion

The original goal was to create a universal emotion module that could create believable behavior for virtual agents. Hope was that the agents might act realistically enough to let patients play a game with them to learn what kind of behavior would help him in succeeding or would cause everybody else to work against him. But different problems were encountered, showing the difficulties of such a generic solution and of realistically simulating emotions in general.

Unlike artificial intelligence, which is typically based on logic, artificial emotion has the enormous disadvantage of not being unambiguously verifiable. It is much simpler to confirm whether a path finding algorithm found the shortest path than confirming if a person with a certain personality would actually react in exactly the same way that the virtual agent reacted. In fact, when even people that have known each other very well for many years can still fail to predict the others reaction, how would it be possible to determine whether an artificial agent's reaction is wrong or simply unexpected? There are usually too many factors involved that are impossible to be handled and considered in a program. Fortunately it is often easier to find an explanation for observed behavior than it is to make a prediction.

With a more modest goal, settling for simulated emotions that are convincing rather than realistic, less ambitious implementations of artificial emotions can already be used to improve a user's experience. It can be seen that, even incomplete and with only improvisational solutions to some problems, the module is still able to produce comprehensible reactions that vary based on the agents personality. While not all goals could be achieved, the results are still promising and hint at the possibilities for future applications.

FLAME was demonstrating how learning can solve many problems and even partially replace personality, while at the same time relieving the programmer from creating huge knowledge bases. An agent would still need to be trained and collect experiences to learn from, but much of that could be automated. Learning is an important factor that needs to be included in any model supposed to be complete, but the issue of state descriptions in complex scenarios has to be solved as well.

However, it must be pointed out that separating AI and AE is not always a good idea. For once, the factual analysis of a situation, which would belong into the realm of AI is also necessary for the appraisal process in AE. This mirrors the way how humans work. Without first extracting information from our perceptions, nothing would be there to trigger any emotions. But once emotions arise, they influence our perceptions and thought processes, so that even when we're rational this rationality is still colored by our emotions [3]. Reason and emotion are strongly intertwined and affect each other. A circumstance that would have to be simulated in any convincing virtual agent.

It is this connection, that makes the module not as easy to use as it was intended. The interface between module and application suggests a clear line between logic and

Conclusion

feelings that doesn't really exist. As this interface was supposed to be small and simple, there is no way to have the kind of interaction between these two that is going on in a real human, without causing significant extra work for the programmer.

In addition to that, he is facing other problems as well. First, the definition of events isn't based on any objective rules. Second and worse, there is an infinite number of ways to let personality, mood and emotion affect the decision making of an agent and both parts are entirely left open to the programmer. Fortunately even here, simple rules can lead to sufficiently colorful behavior, even if it is not entirely realistic.

Machines that are perfectly simulating real human behavior will be reserved to Hollywood for many more years, but even today artificial emotions can easily be used to add flavor and variety.

9 Downloads

EmoSettlers can be downloaded under <http://festini.device-zero.de/emosea.zip>. Note that the console client is not included, as later changes (adding buildings and options) let it become incompatible with the server.

SIMPLEX is available under <http://festini.device-zero.de/simplex.zip>.

This document is also available as PDF: <http://festini.device-zero.de/diplom.pdf>.

References

- [1] Bartneck, C. (2002). Integrating the OCC Model of Emotions in Embodied Characters. Proceedings of the Workshop on Virtual Conversational Characters 2002.
- [2] Becker, C. (2003). der Emotionsdynamik eines künstlichen humanoiden Agenten. http://www.techfak.uni-bielefeld.de/~cbecker/DA_Emotionsdynamik_cbecker_pdf.zip
- [3] Damasio, A. R. (1996). The somatic marker hypothesis and the possible functions of the prefrontal cortex. *Philos Trans R Soc Lond B Biol Sci*, 351(1346), 1413-1420.
- [4] Gebhard, P. (2005). ALMA: A layered model of affect. Proceedings of the Fourth International Joint Conference on Autonomous Agents & Multi Agent Systems 2005, 29-36.
- [5] Gebhard, P., & Klesen, M., & Rist, T. (2004). Coloring Multi-character Conversations through the Expression of Emotions. Tutorial and Research Workshop on Affective Dialogue Systems (ADS04).
- [6] McCrae, R.R., & John, O.P. (1992). An Introduction to the Five-Factor Model and Its Applications. *Journal of Personality*, 60, 175–215.
- [7] Mehrabian, A. (1996). Analysis of the Big-five Personality Factors in Terms of the PAD Temperament Model. *Australian Journal of Psychology*, 48, 86-92.
- [8] Mehrabian, A. (2000). Beyond IQ: Broad-Based Measurement of Individual Success Potential or "Emotional Intelligence". *Genetic, Social and General Psychology Monographs*, 126(2), 133–239.
- [9] Mehrabian, A. (1996). Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14, 261-292.
- [10] Ortony, A., & Clore, G., & Collins, A. (1988). *The cognitive structure of emotions*. Cambridge University Press.
- [11] Ruebenstrunk, G. (1998). Emotional Computers: models of emotions and their meaning for emotion-psychological research. <http://www.ruebenstrunk.de/emeocomp/content.HTM>
- [12] Seif, M., & Ioerger, T., & Yen, J. (2000). FLAME: Fuzzy Logic Adaptive Model of Emotions. *Autonomous Agents and Multi-Agent Systems*, 3, 219-257.
- [13] Wilson, I. (2000). The Artificial Emotion Engine, Driving Emotional Behavior. *AI and Interactive Entertainment*, AAAI Spring Symposium 2000.

EIDESSTATTLICHE ERKLÄRUNG
DIPLOMARBEIT IM STUDIENGANG INFORMATIK

Hiermit bestätige ich, dass ich die Diplomarbeit mit dem Thema

“Implementierung und empirische Evaluierung eines Emotionsmodelles in einer Spielumgebung”

selbständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Ort und Datum

Unterschrift