The application of personality and emotion in artificial agents

THESIS

By John E Armstrong

Department of Information Science and Media Studies

University of Bergen

Norway

Acknowledgments

The author would firstly like to acknowledge Weiqin Chen for her guidance and patience through the project it honestly wouldn't have been done without you. He would also like to acknowledge his family for the great support, specially his mother Ingunn, who though sometimes stressful helped motivate the development of it. Additionally, he would like to acknowledge the great testers who showed a willingness to help him get results on a moments notice and even shared sent the test out to prospective testers. Finally, he would like to acknowledge the memory of the Bodum french press, it served him well through the writing of this thesis. He really wouldn't have made it through this without you people, thank you for staying with him.

\

Table of Contents

Introduction

Since humanity's beginning we have dreamed of creating sentient life out of inanimate material. From the clay golem of Jewish folklore to Metropolis' Maria Robot; people have tried to imagine what such an intelligence might be and how it might be created. With the advent of modern computers, and processor clock speeds, storage- and random-access memory ever increasing; agents display decision making abilities in ever more advanced manners; these factors mean that the reality of realistically acting artificial agents, is coming ever closer to fruition.

On the first page of the Disney animation reference begins with these words: “Disney animation makes audiences really believe in … characters, whose adventures and misfortunes make people laugh – and even cry. There is a special ingredient in our type of animation that produces drawings that appear to think and make decisions and act of their own volition; it is what creates the “illusion of life” [i]. The aim of creating the “illusion of life”, can also be attributed to the field of artificial intelligence.

Artificial intelligence is in many ways a concept within the field of cognitive science, specially when fields regarding mimicking the human brain are involved. Cognitive science is the interdisciplinary scientific study of the mind and its processes, in the search of a model on how the human brain works and how it can be recreated. The fundamental idea behind cognitive science is "that thinking can best be understood in terms of representational structures in the mind and computational procedures that operate on those structures.” [ii].

Two growing fields in the pursuit of realistically acting agents, are the fields of decision making based on personality and emotion. These two subjects are being researched and applied to a multitude applications within many fields; Such as robotic AI, conversational agents and video game agents.

The main motivation for these fields is that emotion and personality can be useful in aiding interaction between the agents and human users. Through expressions of personality and emotions, the agent can create engaging and believable interactions. Additionally, there are also theories that, personality and emotion can play parts in information processing architecture that is not designed for human interaction.iii

Focus and Scope

While the focus of the project is on personality and emotions in artificial agents, it must be noted that the focus of this project does not seek to further research into psychology, but rather to explain how common models of personality and emotion can be used to benefit artificial agents.

It will firstly seek to test whether emotional and personality based expression can be noticeable to a normal user in comparison to the users intrinsic instinct of personification.

It will secondly seek to develop a versatile and reusable system that could be usable in a large range of contexts.

It will seek to do this by searching for models that can potentially be generalized to modify a large number of various systems.

Though the aim of the development is to create a largely reusable system, that can be used to apply personality and emotions in applications as far ranging as games of various genres to applications within talk-bots; It will not be tested as such, but rather in the form of one specific system that will test it's merits, both as an addition to a larger test environment and it's viability as a basis for expressive personalities and emotions. Due to the reusability aim of the project, it's important that both models of personality and emotions are versatile and expansive, both in taxonomy and function.

Literary Review

About this chapter

Due to the fields rarely coalescing into a truly unified model; there are no techniques on how to specifically combine the aspects of both personality and emotion. As such, this section will mainly focus on the current techniques in their respective fields. It will then try to create a picture, of the possible options that might be derived from these individual techniques.

Emotion

What are emotions?

What are emotions? Is there such a thing as a basic emotion[1]? iv What actually can be called an emotion?

- Would for instance surprise be an emotion?
- Or is it a non-emotional response to an unexpected event? v

"In order to delineate the concept of an emotion, we should try to define it. However, the large number of definitions which can be found in the literature is convincing evidence for the ill-definedness of the concept". vi

A. Sloman vii suggested that the difficulty of defining the phenomena of emotions comes down to the lack of deeper theories about the underlying mechanisms of emotion. There are however applicable definitions of emotion. Based on a review of over 100 definitions of emotions,viii proposed that:
"*Emotion is a complex set of interactions among subjective and objective factors, mediated by neural hormonal systems, which can*

a) *give rise to active experiences such as feelings of arousal,*

1 e.g an emotion that can't be broken down further

pleasure/displeasure;

b) *generate cognitive processes such as emotionally relevant perceptual effects, appraisals, labeling processes;*

c) *activate widespread physiological adjustments to the arousing conditions; and*

d) *lead to behavior that is often, but not always, expressive, goal directed, and adaptive."*

Most theorists agree that some emotions can be defined as basic[2] emotions. However, there are as many opinions about the number of basic emotions as there are opinions about their identity. ix Some claim that, in order to be referred to as a *basic emotion*, an emotion should have its own distinct facial expression across cultures. However, there are many things we do not consider to be emotions that has its own facial expression across cultures.[3] x

Ortony and Turner[4] argue that perhaps the only basic emotions are those that are experienced by both humans and animals.

For example it is easy to see that fear and anger are being experienced by a monkey or dog,

but how is it possible to know if a monkey or dog experiences emotions such as envy or shame?

One could argue that fear and anger are more basic since it is more plausible that animals also experience these emotions. The most frequently occurring basic emotions among emotion theorists are anger, happiness, sadness and fear. However, these emotions also seem to be the emotions that are most frequently referred to in Western culture, xi and this might bias some theorists into giving these emotions a special status.

Though the existences of basic emotions are a heavily discussed prospect, most agree that the concept is beneficial in terms of computer science to classify emotions in a certain way. xii

---

2or primary
3e.g. lifting a heavy object
41990

Models

While research into emotions in the cognitive respect by far predates the existence of computers, the field of emotions in regards to artificial intelligence architecture was in it's infancy in large part defined by the work of Herbert A. Simon in his 1967 paper "Motivational and emotional controls of cognition". The paper broke new ground in being the first paper to try and integrate affect with the information processing view of human cognition. The paper focused on goal-terminating mechanism and interruption mechanism, based on emotion and concluded that 1967's processing power could respond to urgent needs in real time and handle goals in a satisfactory manner. xiii

Ira J. Roseman

The theory of Roseman was first presented in 1979, in which he postulated an approach of appraisal theory. It has however, changed multiple times to accommodate newer information regarding the subject of emotions and tests of the model. Roseman started by studying 200 written reports of emotional experiences, which were later used as a basis for which the model could be derived.
The key component of the model is it's six cognitive dimensions, which determine whether an emotion arises, based on a pair of alternate states for each dimension.
The current model of the theory states that these are:

1. The first dimension describes whether a person possesses a motivation to a desired situational state or to an undesired state; xiv
2. The second dimension describes whether the situation agrees with the motivational state of the person; xv
3. The third dimension describes whether an event is noticed as certain or as only a possibility;
4. The fourth dimension describes whether the persons relational appraisal of the potential to control the situation;
5. The fifth dimension describes, from whom the event originates;

6. Finally, the sixth dimension describes whether the is noticed as negative because it blocks a goal or because it is negative in its nature; xvi

Because of the dimensions simple structure which can quickly translated into rues which exactly define which appraisals elicit which emotions. Roseman's models were received very positively in the AI community. With Picard writing: "*Overall, it shows promise for implementation in a computer, for both reasoning about emotion generation, and for generating emotions based on cognitive appraisals."* xvii
The model does, however, contain one major weakness in that it struggles to deal with situations in which a person might make two appraisals. For instance, in a case where a chess player has the opinion that the opposing player is cheating, however knows that the opponent is more skilled.

Aaron Sloman & Monica Croucher

In 1981 Sloman, Croucher published their paper *"Why Robots Will Have Emotions"*. In the paper, they postulate that *"emotions involve complex processes produced by interactions between motives, beliefs, percepts, etc."*, which means that to understand emotion, you need to understand the motivation for these emotions. The paper further discusses a model based on Constraints on intelligent systems.[5]
The computational architecture of the model is similar in many ways to other decision making models, including stores of possible actions, used resources, possible resources, a central administrative process and real-time information on goals, current effects and status of constraints. The computation of the model is therefor also very similar computationally to many other decision making algorithms:

- particularly the "Markov decision process",xviii where the reward can be interpreted as the emotion the motive triggers,
- however it also adds extra computation for the constant

5 e.g complexity, restraints (time, obstacles, goals), non-static environments, speed, etc.

possibility of interruptions and suppression of motives.

It considers the motives, on which it bases it's decisions, on an external *"motive generator"*. Motives in the motive-generator are in many ways defined in a way that overlaps with personality. While they are constituted as either logical or desires, they are ordered by goals and possible goals and may vary in intensity; they are still described as genetically programmed and might change intensity based on the agent's underlying sense of concerns, pleasure and preservation of a state. It is therefore postulated that a priority system is needed to run the motive-generator. xix

Nico Henri Frijda

Frijda's theory postulates that the word *"emotion"* does not refer to a *"natural class"* and that it's not able to refer to a well-defined class of phenomena which are clearly distinguishable from other mental and behavior events. The process of emotion emergence is of larger interest.

At the center of the theory is the term concern. A concern is the disposition of a system to prefer certain states of the environment and of the own organism over the absence of such conditions. Concerns produce goals and preferences for a system. If the system has problems to realize these concerns, emotions develop.

The theory defines six substantial characteristics of the emotion system, which describe its function: The first is the *"concern relevance detection"*, which announces the meaning of events for the concerns of the overall system to all other components; The next is *"appraisal"*, which check the meaning of the stimulus in the concerns of the system. This process has two sub-processes which test appraisal in terms of relevance and context; The *"control precedence"* changes the perception, attention and processing, to affect the behavior of the system, if the relevance signal is strong enough; The fourth is the *"Action readiness changes"*. It affects the system's dispatching of processing and attention resources, which affect the tendency towards specific kinds of actions; *"Regulation", the emotion system, not only activates certain forms of action readiness, but also monitors all processes of the overall system and events of the environment which can affect the action readiness;* The final characteristic of the

emotional system is the “social nature of the environment”, this adjusts the system to the social environment of the surrounding system;
For Frijda, emotions are necessary for systems which realize multiple concerns in an uncertain environment. If a situation occurs, in which the realization of these concerns appears endangered, so-called action tendencies develop. These action tendencies are linked closely with emotional states and serve as a safety device for what Frijda calls concern realization.
Action tendencies and associated emotions are as follows:

- **Approach:** Desire
- **Avoidance:** Fear
- **Being-with:** Enjoyment, Confidence
- **Attending:** Interest
- **Rejecting:** Disgust
- **Nonattending:** Indifference
- **Agnostic:** Anger
- **Interrupting:** Shock, Surprise
- **Dominating:** Arrogance
- **Submitting:** Humility, Resignation

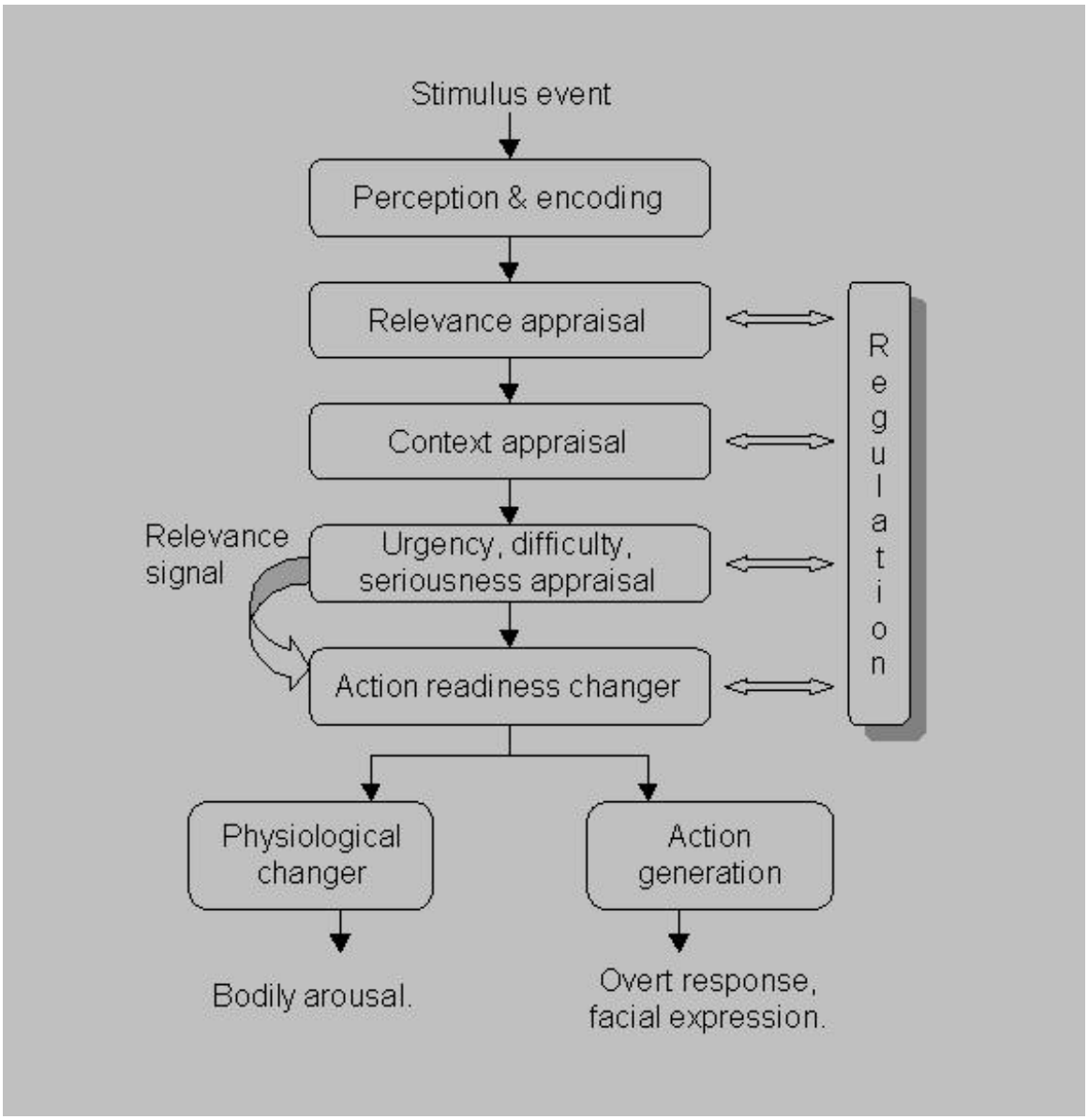
Frijda postulates that a functioning emotional system must have the following components:
Concerns: Internal representations, to which existing conditions are tested.
Action Repertoire: Actions, reactions, social signals and planning mechanisms inherent to the sytem.
Analyser: Analyses the incoming information and subsequent implications and consequences.
Comparator: A test of all information on concern relevance. This results in relevance signals which activate the action system, the Diagnoser and causes attentional arousal.
Diagnoser: Evaluates the context and scans the information for action-relevant references, which results in an appraisal profile.

Evaluator: Agreement or discrepancy signals of the Comparator and the profile of the Diagnoser are combined into the final relevance signal and its intensity parameter. The relevance signal constitutes the so-called control precedence signal.

Action Proposer: Preprocesses the action by selecting a suitable course of action and releases the resources necessary for it.

Actor: Generates the action.

This description of the emotional system was later formalized into the form of a computermodel. xx



Golf Pfeifer

In 1988, Dr. Golf Pfeifer released his, paper on *"Artificial Intelligence models of emotion"*. The paper xxi is a general discussion on the systems involved in emotion and a basis for future models.

In it,

he argues that emotion can not only be regarded as a binary of positive and negative emotions,[6] but also as a binary of intrinsic and extrinsic.[7]

He further argues the next *"dimension"* is the *"locus of causality"*, as in the reaction on another agent's actions and that action's proximity in relation to yourself.[8]

Where further complexity is involved in regard to how agent perceives the intention of the other agent's actions.[9]

He further argues that though these values are in many ways regard as binary opposites, they should not be treated as a Boolean states.

Further he discusses the dynamics of emotion. Arguing the importance of decay of emotion in any emotional system, and how many of the preceding theoretical architectures of emotion do not even consider the fact that it is an important factor in both the strength and longevity of an emotion.

He contrasts this problem with the way many emotions arrive in bundles,[10] and how each emotion is affected by the feedback of it's subsystems. Where, for instance, an emotion's effect on the agent's body might increase the sensation of the emotion itself.

He next argues that the emotion should follow three rules;

one concerning the effect of the emotion and

the other is meant to reflect the tendency of the emotion on enacting an action upon another agent.

The final rule regard the ways the emotion can be lessened.[11]

He argued a heuristics value for the proximity to any given emotion in the taxonomy is of great importance to determine the rule state. xxii

---

6e.g. happiness being a positive emotion, while anger being a negative emotion
7e.g. pain being intrinsic negativity and unfulfilled goals being extrinsic negativity
8e.g. it's effect on the agent
9e.g. did the other agent purposely hinder agent or not
10e.g. frustration and anger
11e.g. happy thoughts for anger

Klaus R. Scherer

In Scherer's theory it is postulated that there are five functionally defined subsystems involved in the emotional process.

1) An information-processing system, that evaluates the stimulus through perception, memory, forecast and evaluation of available information;
2) A supporting system, that adjusts the internal condition through control of neuroendocrine, somatic and autonomous states;
3) A leading system that plans, prepares and selects between competing motives;
4) An acting system that controls motor expression and visible behavior; and
5) A monitor system that controls the attention which is assigned to the present states and passes the resulting feedback on to the other systems.

Of special interest to the model is the information-processing system's involvement with the system. According to Scherer this subsystem is based on appraisals which he calls stimulus evaluation checks. Out of the five stimulus evaluation checks named in the theory, four contain further sub-checks.

A ***"novelty check"*** decides whether external of internal stimuli have changed. It sub-checks suddenness, confidence and predictability.

An ***"intrinsic pleasantness check"*** activates approximation or avoidance tendencies, depending on whether it specifies the attraction as pleasant or unpleasant;

A ***"goal significance check"***, checks whether the event is positive or negative to the goals of the person. It sub-checks goal relevance, probability of result, expectation, support character and urgency;

A ***"coping potential check"***, which determines what extent the person believes to have the person under control. It sub-checks agent motive, control power and adaptability;

A ***"compatibility check"***, finally compares the event with internal and external standards. It sub-checks externality and internality;

The results of these stimulus evaluation checks modify the other subsystems in the system for the final result. According to Scherer, each emotion can thus be clearly determined by a combination of the stimulus evaluation checks and sub-checks.

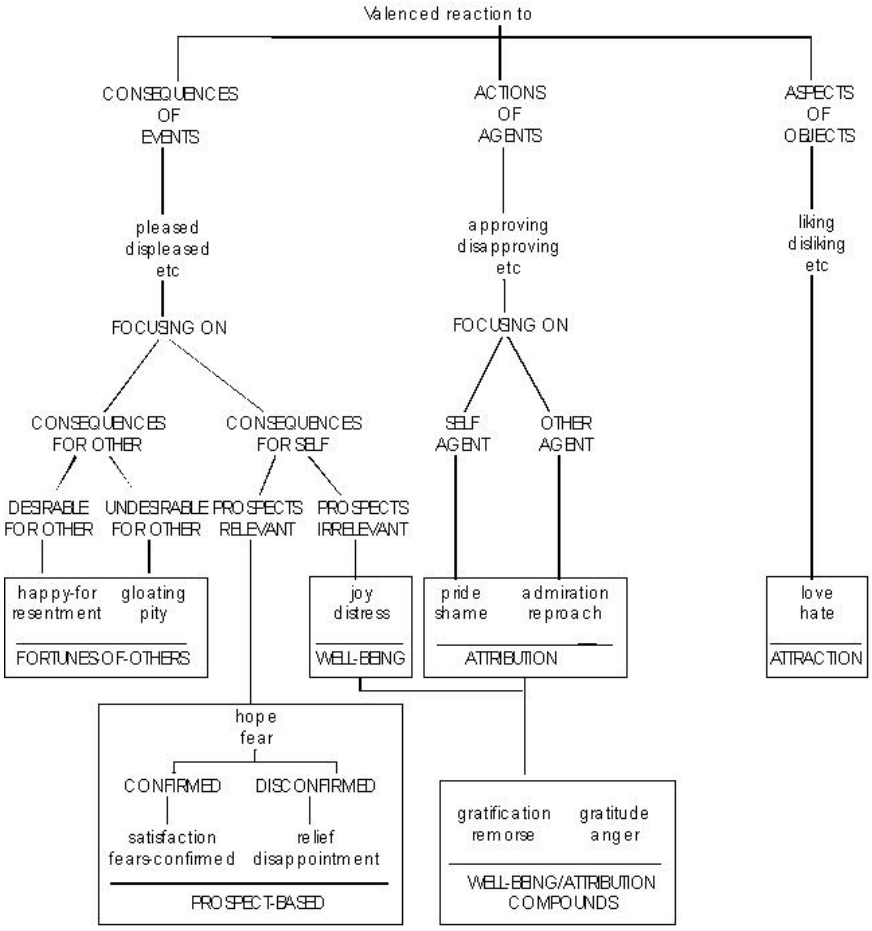
Andrew Ortony, Gerald L. Clore & Allan Collins

One of the first papers to propose an implementable model of emotion in artificial intelligence, was Ortony, Clore and Collins' paper on *"The cognitive structure of emotions"*. [12] It's outlined is one of the most commonly employed models of emotions, still in use to this day.

In the paper Ortony, Clore and Collins defined emotions as: *"[...] valenced reactions to events, agents or objects, with their particular nature being determined by the way in which the eliciting situation is construed"*. The model taxonomy, divides emotion into 20 separate emotional types, which are further divided into groups of correlation, either to a reaction to events, other agents or objects.

These traits are based on having either a positive and negative[13] associative connotation, hence excluding neutral emotions.[14] The emotion's nature is then dependent on the agent's construal of the situation that elicits the emotion.[15] Agent's approval or disapproval of, for example, an event are decided by the agent's preset *"core values"*. Lastly, emotions can also be put into compound emotions. This means that multiple emotional responses can be elicited from one factor.

121988
13valenced
14i.e. surprise
15i.e. if the agent perceives an event as positive of negative



The intensity of an emotional feeling is determined by a set of variables. These are firstly divided into the variability of the eliciting factors.[16] Secondly there are a set of global variables to determine the agents sense of reality, proximity, unexpectedness of an eliciting factor. In a concrete case, each of these variables are assigned a value and a weighting, where the overall intensity of the factor is measured against a threshold value for each emotion, below which the emotion will not be elicited. [17] [18]

Events	**Agents**	**Objects**
Desirability	Praiseworthyness	Appealingness
Desirability for other	Strength of cognitive unit	Familiarity
Deservingness	Expectation deviation	
Liking		

16i.e. the desirability of an event; the expectation deviation of an agent; or the appealingness of an object
17Structure of emotion types in the theory of Ortony, Clore and Collins (after Ortony, Clore, Collins, 1988, p.19)
18Local variables in the theory of Ortony, Clore and Collins (after Ortony, Clore and Collins, 1988, p. 68ff.)

Events	Agents	Objects
Likelihood		
Effort		
Realization		

An example of this model in progress can be described in formal language in terms of:
D(p, e, t) being the desirability **(D)** of event **(e)** for person **(p)** at time **(t)**;
$\mathbf{I}_{g}$**(p, e, t)** being the combination of global intensity **(I)** variables; and $\mathbf{P}_{j}$ **(p, e, t)** being the potential of a state of joy.
The resulting function $\mathbf{f}_{j}$ then represents the combined effects of the local and global variables affecting the emotion joy.

```
IF D(p,e,t) > 0
THEN set P_j(p,e,t) = f_j(D(p,e,t), I_g(p,e,t))
```

This is then checked against
the threshold $\mathbf{T}_{j}$ which results in the intensity of joy being set to 0 (no feeling)
if $\mathbf{P}_{j}$ is below the threshold,
however if $\mathbf{P}_{j}$ is larger than the threshold the resulting intensity equals $\mathbf{P}_{j} - \mathbf{T}_{j}$.

```
IF P_j(p,e,t) > T_j(p,t)
THEN set I_j(p,e,t) = P_j(p,e,t) - T_j(p,t)
ELSE set I_j(p,e,t) = 0 [19] xxiii
```

Ortony, Clore and Collins do not supply formalization for all of their defined emotions, but give a few further examples. They postulate, however, that every emotion can be described using a formal notation, although with many emotions this is by far more complex than with the presented example**.** The end goal of the thesis is not to create a model that will give agents feelings, but for a computer to draw conclusions about emotional episodes presented to it.
"Our interest in emotion in the context of AI is not an interest in questions such as "Can computers feel?" or "Can computers have emotions?" There are those who think that such questions can be answered in the affirmative..., however, our view is that the subjective experience of emotion is central, and we do not consider it possible for

19 Joy example in the theory of Ortony, Clore and Collins

computers to experience anything until and unless they are conscious. Our suspicion is that machines are simply not the kinds of things that can be conscious. However, our skepticism over the possibility of machines having emotions certainly does not mean that we think the topic of emotions is irrelevant for AI..... There are many AI endeavors in which the ability to understand and reason about emotions or aspects of emotions could be important." xxiv

Rosalind W. Picard

The modern branch of the field is also in large part influenced by Picard's 1995 paper *"Affective Computing"*.

In the paper she discusses the importance of emotion in human reasoning, through the relationship between the cerebral cortex and the climbing system and concludes that emotion plays a large role; contrary to models like the Myers-Briggs type indicator which divides thinking and feeling into two separate fields. She further describes models of how emotion can be applied to artificial agents.

She states that the most common *"prototype emotions"* are:

fear,

anger,

sadness

and **joy**,

and that these can further be controlled by three dimensions of emotion:

arousal calm/excited,

valence negative/positive

and **attention** internal/external, contempt/surprise.

She further concludes that emotions work well in a "Sentic state model", as described by Clynes and Manfred. xxv

However she argues that, contrary to the Clynes, Manfred model of pure emotions[20]; Emotional states are more or less fuzzy, based on intensity of the emotion[21].

---

20 i.e emotions are mutually exclusive

21 i.e stronger emotions are less fuzzy

Further, in the paper, Picard discusses the architecture of emotion processing, where she discusses the possibility of using models like Kharkov's or partially observable Markov. In contrast with both Pfeifer and Simon, Croucher; Picard argues that the architecture should be able to recognize the current explicit emotion rather than the motive or an implicit representation of the emotion. xxvi

Keith Oatley & Johnson-Laird

Oatley and Johnson-Laird developed their theory expressly *"communicative theory of emotion"* in a form which can be implemented as a computer model, even if they did not carry out this step. They see the necessity for their model in the fact that almost all computer models of the human mind did not consider emotions, while they regard this as a central component for the organization of cognitive processes.
Oatley and Johnson-Laird 's theory assumes a hierarchy of parallel working processing instances, which work on asynchronously different tasks. These instances are coordinated by a central control system, which contains a model of the entire system. Each individual module of the system has to communicate with each other module, in order for the system to function.
In the system there are two different kinds of communication.
The first is semantic signals[22], which conveys information about the environment.
The second is control signals[23], it does not convey information, but shifts the entire system of modules into a state of increased emotion, called "emotion mode". This function acts to interrupt processes in action in order for the system to focus.
According to Oatley, the central postulate of the theory follows these terms: "Each goal and plan has a monitoring mechanism that evaluates events relevant to it. When a substantial change of probability occurs of achieving an important goal or subgoal, the monitoring mechanism broadcasts to the whole cognitive system a signal that can set it into readiness to respond to this change. Humans experience these signals and the states of readiness they induce as emotions." xxvii

22earier called *"propositional"*
23earlier called "nonpropositional"

Emotions coordinate quasi-autonomous processes in the nervous system by communicating significant way marks of current plans, called “plan junctures”.
Such plan junctures work in conjunction with elementary emotions[24]:

Plan juncture	Emotion
Sub-goals being achieved	Happiness
Failure of major plan	Sadness
Self-preservation goal violated	Anxiety
Active plan frustrated	Anger
Gustatory goal violated	Disgust

Since they arise at plan junctures, emotions are a design solution for problems caused by plan changes in systems with a multiplicity of goals.

Implementations of emotion

Emotions in Statecraft

On the concept of specific application of emotion in a multi-process environment (such as video games), there are two papers of note; C. Carlson and M. Hellevang's, 2010 paper on *“Improving user experience in StateCraft”*, and the 2012 paper on *“Modeling Emotions with EEG-data in StateCraft”*, by A. N. Slinde.

Cristoph. Carlson and Mathias Hellevang

C. Carlson and M. Hellevang's implementation concerns emotions in the agents in the turn-based strategy game Statecraft. In it they decide that the OCC model is the most viable model for their implementation of the *“emotion module”* and a *“Prisoner's Dilema Module”,* however, this discussion will only concern the “emotion module”. For the model, they utilize the fear emotion of the prospect leaf, the joy/distress valence of the well-being leaf, anger from the compound leaf and the admiration/reproach valence of the attribution leaf, while replacing the pride/shame valence with a guilt emotion. All

24 Plan junctures and associated emotions in the theory of Oatley & Johnson-Laird (after Oatley, 1992, p. 55)

emotions in the system are between the numeric values of 0 and 100, with a default value of 0. All have a different intensity directed towards each player, except joy, which only applies to the character. From this they derive an algorithm for the compound of anger.

```
IF joyValue < 0 AND admirationValue < 0
  IF |joyValue| > joyThreshold AND |admirationValue| > admirationThreshold
     THEN set anger = sqrt(joyValue * admirationValue)
```

Their findings for the “Emotion module” are that the agents performed at a decreased performance in terms of supply centres in the game. They were unable to find signifigant differences between their agent and the standard unit utilized by the game. In their final conclusion they argue that the results were due to their limited data collection on the lack of signifigant data on the subject. **xxviii**

Anders Njøs Slinde

Slinde's model is based on the “emotion module” developed by Carlson and Hellevang. However, they differ in one key aspect; namely, how they process the emotion in conjunction with the different layers of the game's AI. While Carlson and Hellevang uses an altered form of the OCC model tree, implemented directly into the strategical layer of StateCraft. A. N. Slinde, implemented a neural network into utilizing training based on an EEG headset, while comparing it to the model of C. Carlson and H. Hellevang's previously created emotion module. The neural network serves as a way for the emotion module to train itself, into improved states, which further serve to improve the system's emotional parameters for later runs of the software. Slinde found the same decrease in performance in terms of supply centres as Carlson and Hellevang. He concludes that though there are few findings, the EEG data indicates improved results over that of Carlson and Hellevang. It is however noted that the Emotion module using both game state based and EEG-based emotions perform better with the general emotions than with the country-specific emotions. In direct contradiction to this the configuration using only the EEG-based emotions performed best using the country specific emotions. The performance difference is bigger in the EEG- 85 based

simulations, indicating that country specific emotions do perform better than general emotions. xxix

Personality

Personality is often seen as a pillar of future artificial intelligence. With robots like R2D2 from Star Wars, to 2001: A space geodesy's HAL, media's vision of artificial agents have been brimming with personality for years. But what research has been done into the prospect of how a personality works?

Early Personality Inquiry

The search for a definition of personality is often credited to have started with Hippocrates (460–370 BC) and his version of the four humors (fluids). Though originally an ancient Egyptian or Mesopotamian model, Hippocrates formalized the model where the emotions and behavior could all be explained through the excess or lack of the bodily fluids.

The model was further developed by Claudius Galenus (AD 129 – c. 200), who further likened the four humors with the four elements, where each could be explained as hot or cold, dry or wet. He further related the four elements to characteristics viewed to affect the temperaments of a person and as such created the four temperaments xxx.

The four humors, and their respective elements and temperaments were as such defined as:

Humor	Element	Temperament	Temperament Characteristics
Blood	Air	Sanguine	Courageous, hopeful, playful, carefree
Yellow bile	Fire	Choleric	Ambitious, leader-like, restless, easily angered
Black bile	Earth	Melancholic	Despondent, quiet, analytical, serious
Phlegm	Water	Plegmatic	Calm, thoughtful, patient, peaceful

This dualistic and attribute based view of personality was a strong basis for the models we see as personality today. It did, however, not address a sense of an individual personality which is a major tenant of the modern view of personality. This view came as a result of the cultural shifts that originated as Renaissance xxxi, however was not formalized in a meaningful way until relative modernity.

Carl Jung and Myers-Briggs

Jung's interest in typology started with his desire to reconcile the theories of Sigmund Freud and Alfred Adler, and to define how his own perspective that differed from theirs. This led him to eventually conclude that Freud's theory was inherently extraverted, while Adler's theory was inherently introverted. Jung became convinced that the animosity between the Freudian and Adlerian groups was due to the inherent contradiction between the two thoughts fundamental to the two models. He argued that each side can demonstrate the truth embodied in its theory. However, it is only partial truth and not generally valid because it excludes the principle and truth embodied in the other. Based on this feud Jung postulated the basics tenants of his model, which was published in his work in German in the book *"Psychologische Typen"*, released in 1921.
In 1917, Kathrine Cook Briggs observed marked differences between her future son-in-law and his other family members, this event is credited to have started her research into personality. Briggs embarked on a projects of analyzing biographies and developed her first typology of personality, consisting of four temperaments: meditative, spontaneous, executive and social. After the publishing of the English translation of Jung's work in 1923, Briggs recognized that the model was similar, however further developed than her own. This led her to develop her own types to correspond with Jung's.
Eventually, development of the model started transitioning to Briggs' daughter Isabel Briggs Myers. Due to Myers and Briggs lack of credentials and inexperience in psychology and psychometric testing, Myers decided to go into an apprenticeship for Edward N. Hay. During her time with Hay, Myers learned rudimentary test construction, scoring, validation, and statistical methods.
Myers and Briggs further developed the type-indicator during World War 2, eventually

publishing the Briggs Myers Type Indicator Handbook in 1944. The model started gaining traction with further development and the first edition of the MBTI Manual was published in 1962, with subsequent editions in 1980 and 1998.
The basic tenants of the Jung and Myers-Briggs models are their four cognitive functions, where each comprised of two polar orientations, hence giving a total of eight dominant functions. The four types of Jung's theory are labeled as extraversion, sensing, thinking and judgment, generally abbreviated to ESTJ; while the four types of Myers-Briggs are labeled as Introversion, Intuition, Feeling Perception, generally abbreviated to INFP.

Carl Jung's four dichotomies	
	Subjective
Perception	Intuition Sensing
Judging	Feeling Thinking

Myers-Briggs' four dichotomies	
	Subjective
Deductive	Intuition Sensing
Inductive	Feeling Thinking

In the context of both models, the four types are divided into two dichotomies, whereof none of these can be seen as inherently positive, nor inherently negative, hence giving the models 16 possible combinations and results.

Challenges to personality

Due to mounting arguments against the prospect of defining personality, the research into finding definitions fell silent to a large extent from the 1960s until the 1980s. Researchers like Walter Michel argued that personality instruments could not predict behavior with a correlation above 0.3, and that attitudes and behavior were not stable, but varied with the situation. Predicting behavior from personality instruments was claimed to be impossible, and as such the prospect of personality became uncertain in psychology. These assertments, were however, subsequently been demonstrated

empirically incorrect in terms of the magnitude of predictive correlation with relation to real-life in conditions of stressful emotional states. Hence giving personality a significantly greater proportion of the predictive variance. xxxii

In the 1980s, emerging methodologies challenged this point of view even further. Instead of trying to predict single instances of behavior, which was unreliable, researchers found that they could predict patterns of behavior by aggregating large numbers of observations. xxxiii This resulted in the view of the correlations between personality and behavior being revised and the revival of the view that "personality" does in fact exist. xxxiv Trait theories became justified, and there was a resurgence of interest in this area. xxxv

The Lexical Hypothesis

The first modern inquiry into deriving a comprehensive taxonomy was performed by Sir Francis Galton in 1884. His Lexical Hypothesis was based on deriving words with descriptions of personality attributes from dictionaries. xxxvi

In 1936, Gordon Allport and Henry S. Odbert put Sir Francis Galton's lexical hypothesis into practice by extracting 4,504 adjectives which they believed were descriptive of observable and relatively permanent traits from the dictionaries at that time. xxxvii

Raymond Cattell further developed the model in 1940, by eliminating the synonyms which reduced the model to 171 adjectives.xxxviii He constructed a self-report instrument for the clusters of personality traits he found from the adjectives, which he called the Sixteen Personality Factor Questionnaire.

In 1963, Warren Norman started an independent analysis of Allport and Odbert's original list to create a more precisely structured taxonomy of terms. Using the 1961 edition of Webster's International Dictionary, Norman added relevant terms and removed those from Allport and Odbert's list that were no longer in use. This resulted in a source list of approximately 40,000 potential trait-descriptive terms. This resulted in a source list of approximately 40,000 potential trait-descriptive terms. He further

developed the model by removing terms that were deemed archaic/obsolete, overly obscure, dialect-specific, solely evaluative, loosely related to personality or purely physical. Which reduced the list to 2,797 unique trait-descriptive terms. xxxix

During its time of development the lexical method has become one of the most influential scientific theories for guiding personality psychology.

Big Five

The basis for the big five were given by Ernest Tupes and Raymond Christal who found five broad factors in a subset of 20 of the 36 dimensions that Cattell had previously identified. They labeled these as *"surgency"*, *"agreeableness"*, *"dependability"*, *"emotional stability"*, and *"culture"*. xl

In his research Warren Norman related the factors to his research and relabeled *"dependability"* as *"conscientiousness"* .xli

Following the lack of research into personality that happened up until the 1980s, Lewis Goldberg started developing his own inquiry into a lexical hypothesis. He reemphasized the concept of the five broad personality factors and coined the term "*Big Five"* as a label for the factors. xlii

The five factor model gained traction throughout the 80s following a symposium in 1980. xliii

The model was further developed through various iterations, such as the five-factor "Pentagon" model and the NEO five-factor personality inventory. xliv

The five factors of the Big Five can be summarized by the acronym OCEAN. Unlike the theories of Jung and Myers-Briggs the factors are not two sided, but rather a scale from high to low.
The five factors and their constituent degrees are **Openness to experience -** inventive/curious or consistent/cautious, **Conscientiousness -** efficient/organized or easy-going/careless, **Extraversion -** outgoing/energetic or solitary/reserved respectively, **Agreeablenes -** friendly/compassionate or analytical/detached and

Neuroticism - sensitive/nervous or secure/confident. xlv

Openness to experience reflects the degree of intellectual curiosity, creativity and a preference for novelty and variety a person has. A person with high openness can be perceived as unpredictable and unfocused; while low openness can be perceived as pragmatic and data-driven, but can also be perceived as dogmatic and closed-minded.

Conscientiousness reflects the degree to which a person is organized, dependable, self-disciplined, dutiful and achievement focused. A person with high conscientiousness can be perceived as stubborn and obsessive; while low conscientiousness can be perceived as flexible and spontaneous, but can also be perceived as sloppy and unreliable.

Extraversion reflects the degree of energy, surgency, assertiveness, sociability and talkativeness a person has. A person with high extraversion can be perceived as attention-seeking and domineering; while low extraversion can be perceived as reserved, reflective personality, but can also be perceived as aloof or self-absorbed.

Agreeableness reflects the degree to which a person is compassionate and cooperative. A person with high agreeableness can be seen as naive or submissive; while low agreeableness personalities are often competitive or challenging people, but can also be seen as argumentative or untrustworthy.

Neuroticism reflects the degree to which a person a person is compelled to experience unpleasant emotions (e.g. anger, anxiety, depression, vulnerability). A high need for stability manifest as stable and calm personality, but can be seen as uninspiring and unconcerned. A low need for stability causes a reactive and excitable personality, often very dynamic individuals, but they can be perceived as unstable or insecure.xlvi

Zamora

Antonio Zamora defines personality as *"the totality of character attributes and behavioral traits of a person"*.xlvii

The Zamora personality taxonomy was developed for use in the Zamora personality test. The test was developed by Antonio Zamora for the purpose of estimating the

compatibility between two people. Zamora compounded a collection of characteristics people looked for in their ideal mate from an extensive compilation of personal advertisements from newspapers. The characteristics that were deemed desirable by people in the adverts were judged as *"desirable"*, while *"undesirable"* traits were derived from a list of antonyms. Upon further analysis, these were further divided into two groups, one individual and one social, each comprised of ten attributes.

The model shares similarities to the Big Five in two major ways. The first is that it incorporates the five factors defined in the model. The second is that the attributes act as scales from positive to the negative. In addition each attribute has two polar orientations similar to the types of the Myers-Briggs model.

In a concrete case individual attributes can be displayed in any situation and may only be apparent to the individual, while the social attributes can only manifest themselves in a social situation. xlviii

Though the individual attributes have polar orientations designated as positive of negative, this does not reflect on the desirability or undesirability of any of the attributes. In these cases the extremes of each pole is the negative factor, while any moderate value is considered a grey-zone. Two people are compatible if most of their attributes align.

Individual attributes, for the most part, are that part of our personality that cannot be altered. Zamora states that *"We cannot become more intelligent, but we can become more educated. We cannot become more attentive, or less impatient, or more optimistic. These are physical characteristics that are determined by our brain structure and our body chemistry"*. It is however noted that individual attributes can change in extreme cases such as addition, strokes and head injuries. xlix

The individual attributes of Zamora are as follows:

INDIVIDUAL ATTRIBUTES	CHARACTERISTICS
	POSITIVE +
i1. Achievement attitudes – degree of motivation.	persistent, ambitious, obsessive
i2. Emotional temperament – emotions that rule our lives.	confident, stable, calm, relaxed, patient
i3. Energy level – pace of our daily life.	active, energetic, fast

INDIVIDUAL ATTRIBUTES	CHARACTERISTICS
	POSITIVE +
i4. Intellectual factors – characteristics of our minds.	alert, inquisitive, intelligent
i5. Material attitudes – how we regard our environment.	frugal, thrifty, materialistic
i6. Maturity – our level of experience and wisdom.	mature, knowledgeable, wise
i7. Philosophical attitudes – our ways of thinking.	optimistic, positive, flexible
i8. Physical attributes – how we regard our body.	youthful, healthy, strong, sane
i9. Risk attitudes – degree of concern for oneself.	conservative, cautious, calculating
i10. Task performance – attitudes toward problem solving	organized, accurate, skillful, methodical

Unlike the individual attributes, the social attributes two equal poles, but rather a scale from desirable to undesirable. Social attributes are only compatible in cases where most of the social attributes are positive. Every negative value will detract from the positive view of the person. As such the value of compatibility is not determined by similarities between the two persons attributes, but rather on the overall positive value of the social attributes.

The social attributes of Zamora are as follows:

SOCIAL ATTRIBUTES	CHARACTERISTICS
	SOCIABLE +
s1. Aggressiveness – our demeanor toward people.	friendly, courteous, thoughtful
s2. Control attitudes – mechanisms by which we influence others.	persuasive, conciliatory, submissive, gentle, yielding
s3. Dependability – factors that affect trust in others.	dependable, trusting, honest, truthful
s4. Egocentrism – our degree of selfishness.	generous, humble, forgiving, modest
s5. Emotional expression – our ways of expressing feelings.	congenial, funny, extroverted, talkative
s6. Fairness – how we judge others.	appreciative, impartial, tolerant
s7. Leadership – how we interact in a group.	brave, leader, independent
s8. Physical appearance – how we view ourselves physically.	attractive, stylish, tidy
s9. Regard for Rules – obedience for the laws of society.	ethical, honest, law-abiding
s10. Team Spirit – how we fit in society.	social, family-oriented, patriotic

Implementation of Personality

Arild Johan Jensen and Håvard Nes

One paper that show this specific use of taxonomy, is the 2008 paper by J. Jensen and H. Nes, “The Personality Module”. The paper concerns the specific application of personality to a player agent in the strategical video game “StateCraft”. In the paper, they conclude that the best taxonomy for their use is the Zamora taxonomy. This decision is based on the complexity needed to convey, a full spectrum of personality based actions. However, their taxonomy does not constitute the entirety of the Zamora spectrum, as they opt instead to utilize two individual (i.e. emotional temperament and risk attitudes) and two social (aggressiveness and regard for rules). The rest of the taxonomy is deemed to not have any way of being performed in a system such as StateCraft. Jensen and Nes' results conclude that personality and specifically their implementation of the Zamora model were a success, showing a noticable difference between the AI with and without the personality module. l

Test environment

The test environment for the project is the game 0 A.D and it's bot the petrabot. 0 A.D. Is a real time strategy inspired by the age of empire series. The game takes place in a fictionalized version of the time period surrounding the year 0 A.D.

Originally developed fully by Wildfire Games, the game whent open-source on the July 10th, 2009 and also went completely free to download, both in it's source and binary forms. The game is also a freely licensed software, to allow for young developers to practice and earn credentials to provide an entry point into the industry.

Work on 0 A.D. began in 2001, first as a mod concept for Age of Empires II. With limited design capabilities, the team soon turned to trying to create a full independent game based on their ideas. Wildfire Games released source code for 0 A.D. under the Gnu public license model. In 2013, Wildfire Games started a crowdfunding campaign to raise money to hire a larger team. Though they didn't get the amount they had wished

for they hoped for, they were able to hire a programmer.

As of this date, the game is still considered to be in alpha, with constant development by the open-source fan community that has gathered around the game, as well as Wildfire games.li

As stated before, the game is a real-time strategy game. The game is based of controlling and expanding a settlement, while being at war with other civilizations. The game focuses to a large part in growing the settlement into a large city, while also controlling your armies in skirmishes. This continues until one side has won by defeating the enemy player to some capacity.

There are twelve civilizations in the game, each represented by the development and style they had at they're greatest period. Each civilization, has a range of unique unit characters, buildings as well as both land and naval, vehicle units.

The civilizations featured in the game are the:

- Athenians, Macedonia, Spartans, representing the major Mediterranean factions of 900 – 100 B.C.
- The Britons, Gauls and Iberian representing England and the surrounding area in the era of 200 – 100 B.C.
- The Romans, Seleucid and Egyptians representing the Mediterranean around the Roman era of 500 B.C. - 27 B.C.
- And the Carthaginians, Persians and Mauryan around the same time and are as the Athenians.[lii]

The game features both a single player and a multi player mode. The map types differ from computer generated to predesigned maps by either Wildfire Games or the open-source community.

The game is developed in using a C++ engine called Pyrogenesis. Pyrogenesis was developed by the developers at Wildfire Games and is designed to support both real-time strategy and third person role-playing games. It features an OpenGL-based rendering engine, support for scripting in JavaScript, Data files in XML, Peer to Peer capabilities, a level editor and support for scripted agents in the form of A* pathfinding. [liii]

Pyrogenesis utilizes the JavaScript middle-ware SpiderMonkey for all communications between the two layers of the game. It is developed by Mozilla and is used in many of their products. It is normally used for communication between the back and front end of browsers and websites. [liv]

The PetraBot

In the single player mode 0 A.D. can be player against one of multiple possible bot iterations. The newest and most notable of these being the 'Petrabot'. The Petrabot was developed in 2014, by a user using the forum pseudonym 'Duplicarius' and was an iteration on the earlier 'Aegis' bot. [lv]

The agent is designed to vary in strategy, based on the difficulty, as well as three randomly valued personality modifiers. The first of these is the agent's aggressiveness,

this determines how frequently and quickly the agent chooses to attack it's opponents. The second is cooperativeness, which determines how helpful the agent will be if an allied in trouble. This value is also modified based on tributes the agent might receive from the other players. The third and final one is the defensiveness of the character, which determines how prone the agent is to build defensive structures.

The script layer of the Petrabot utilizes two main modules. The first being the headquarter module, which controls the major functions of the agent. It utilizes multiple sub-modules, where the three most notable are: the attack manager, which deals with the strategical and skirmished based sections of the game, it is solely responsible for the soldier units; the base manager, which organizes the worker units and the development of the settlement; and the diplomacy manager which controls how the agent deals with it's allies. The second of the main modules is the agent is it's queue manager. It handles the many queues of the of actions the agent will make in the various aspects of the gameplay round. The queues are performed in order of a dynamically adjustable priority value, that adjusts based on the perceived importance of each task at the current state in the game. lvi

Design and Implementation

About this chapter

This chapter concerns the design and implementation of the emotion and personality system. The main subsystems will first be discussed individually in regards to the models discussed in the literary section and the requirements of the system. They will then be discussed in regards to specific implementable designs derived from the models. This section includes code, which will be encapsulated and colored as such:

```
double val = 0; //Comment
if(val == enum){ MethodCall(); }
```

Project development language, paradigm and format

The nature of the project demands that it needs to perform well in conjunction with the functionality of multiple other systems. As systems incorporating the techniques will at the bare minimum require a user interface and in some circumstances (such as video games), require a large amount of concurrent calculations, to make the artificial agents as well as the surrounding systems run. Additionally, individual sections of the prototypes will have to be easily accessible from different parts of any given system. It will also serve it's further development possibilities, if the prototype is easily portable between systems and software. As such the prototype will require speed, segmentation and portability.

Language

To satisfy a few needs of the project, it will best be developed in C++. The language is an iteration of the C language and supports a large range of. Originally named C with classes, it's purpose was to be an iteration that supported the Object-oriented paradigm; while still retaining the low-level utility and speed of it's predecessor. It was developed

with a bias towards system development (such as operating system kernels), but has also been found useful in many higher-level applications.

The language's standard is now on the iteration known as C++ 11 (named by year of release), and has multiple additions and variations (including a .Net implementation).[lvii] The language has been the basis of many higher level language's, such as Java and C#, and has even become a basis for newer iterations of it's predecessor C. It is today, regarded as one of the more complex languages one can; however, it's status as one of the more powerful languages still retains it's position in use today.

Due to the many calculations needed to perform the real-time execution of emotions and the necessity of the system to perform well in conjunction with other systems, the speed of the C++ language is a perfect fit for the project. Additionally, C++'s portability, means that emotions and personality variables can be compiled to a multitude of formats, across multiple platforms, such as: .lib (Windows Static Library), .dll (Windows Dynamic Link Library), .a (Archive, Unix Static Library), .so (Shared Object, Unix Dynamic Link Library), as well as a large range of executable files, which means that the project can potentially be implemented as, either a peripheral library or as part in a standalone application. As previously mentioned, C++ was originally developed to be an iteration of C, utilizing an object-oriented paradigm. However, C++ is also a multi-paradigm language (which means that it does not necessarily suit the segmentation criteria), and in addition to the object-oriented, also allows development in paradigms, such as: procedural, functional and generic paradigms.

Paradigm

However, to suit the segmentation need of the project, the project should be developed, using the object-oriented paradigm. The object-oriented paradigm works by segmenting functionality into classes, from which objects (software bundles of related state and behavior) can be derived. The functionality of these classes are further segmented within the class as methods that can either be called through a static reference of the class, or through an object of the class. The paradigm is designed to easily be able to

mimic and recreate real-world environments where a system can be a sum of multiple parts or an entity contains attributes. [lviii]

Format

To improve the portability of the project prototype should be compiled and structured; down to and as a library. This allows the project to take form not only as part of one software, but also as a component to future software. Additionally, it means that the components and methods of this project, can easily be shared, expanded and improved, beyond the life-time of the project itself. A library will also allow the project to be ported between operating systems (given the correct design circumstances), to be utilized in systems ranging from video games to communication systems.

Libraries can be developed by two models; either as a “dynamic linked library” or a “static library”. The main difference between these, is how the library includes it's dependencies on compilation. Where a static library will compile both the library and it's external dependencies are linked into a final binary; a dynamic library will compile any external dependencies only by an embedded name reference. This in turn leads to a multitude of pros, cons associated with their use, development and maintenance.

A dynamic library requires all dependencies to have the same version present the for the library to function. Positively, this means that a dynamic library will both have a smaller size than it's static counterpart and that dependencies don't need to be loaded multiple times for multiple libraries that require the same dependencies. Additionally, out of date dependencies can simply be changed without recompilation of either the library or the code dependent on it. Static libraries, however, require recompilation for each dependency change and each software that requires it needs to be recompiled to accommodate, however it's pros are a dynamic library's cons.[lix]

Due to the low number of external dependencies required by a mainly mathematically based library, such as this; the library will need few to none external dependencies other than the C++ standard library. Because of this dependency on the C++ standard library it can be be assumed that a dynamic library will be more suited than a static library for

this project. This also has the added advantage of allowing the library to be called from languages, such as C#, where static libraries are not supported.

System Overview

The core system of the personality and emotions system is comprised of three main subsystems, as well as a series of data management and controllers specific to each system. The three main subsystems of the system are the 'emotion', 'personality', and 'control and interfacing' systems respectively.

The system is designed to be self contained and generalized, as it's designed to be utilized across a wide range of systems. this section will contain details that are implementation specific to the test environment implementation section, followed by an example from the test environment of the system.

Emotions System

The first core subsystem of the project is the emotion system. The emotion model selected for this was the Ortony, Clore, Collins model.

There are multiple advantages to the OCC model that make it uniquely suited for a project such as this, however the main reason for its selection was due to their view on the emotion calculations themselves.

Their system of values, weights and thresholds, as well as local and global intensity modifying variables allow for a broad range of possible reactions to pretty much any given situation. Additionally, it allows for a degree of variability in reactions different characters can have to the same situation.

The model has a strong and broad taxonomy, with many differing emotions that can serve to bring variability in many different circumstances. These are divided into an architecture that works to underline their intended outcome and use, in

such a way as to be extremely userfriendly.

Emotional Taxonomical Type Divide

The emotional taxonomy of the OCC model is divided into three implicit type divisions; where each is integral to how the specific emotion is computed and utilized in the system. It has to be noted, however, that though the divisions do not directly interact with each other, they are implicitly linked.

The first is a division in the type of the eliciting factor the reaction is attributed to; whether it's consequences of events, actions of agents or aspects of objects. This division is important to note because it is intrinsically linked to how the emotion is selected. As previously stated, this implementation of the model only concerns the first two (i.e. consequences of events and actions of agents).

The two other divisions can, in varying ways, be seen as divisions in the focus of the reaction.

The first and most noticeable of these can be clearly seen in the tree of the OCC model and represents the perspective the focus takes and the emotional groups representing the perspective; whether it focuses on: the fortunes of others, the prospect of an event, ones own well-being, the attribution of an action or a compound of the latter two. These groups are important to denote a shared interaction between the formulation of various emotional intensities. Emotions in these groups may utilize the same intensity modification variables, be directly affected by one and another, or simply calculate the final intensity in the same way.

The final division represents who the reaction is focused at; whether it's personal or social. As such the application of an emotion can be seen to either affect the current agent character or the current agent characters opinion towards another character. Following this divide, the emotions gratification, hope, satisfaction, relief, joy and pride, and their respective counterparts are applied only to ones self; while emotions such as happy-for, gloating, admiration, gratitude and their respective counterparts

apply to only to a characters opinion of another. In the context of emotion, the social emotions define the character's opinion towards another.

The cases where these divisions become relevant will be discussed in further detail in the sections to come.

Emotion Handler

The emotion system of the OCC model can be seen as three subsystems, the main tree, modification factors, and eliciting factors. In addition to this the emotion system need to interface towards other subsystems in the model. The emotions system is comprised of two subsystems; an emotion handler system and a value modification behavior tree system. The emotion handler system's main functions are to contain the definitions of the emotional intensities, their related thresholds and to provide inputs and outputs of the emotions and for the emotion. The emotion handler also contains sorting and queuing systems for the emotion and the value modification behavior tree system. The emotion handler take the arguments of it's own identification and a definitional struct of the emotional thresholds for the character. Upon instantioation it first defines and builds the emotion tree. It then sets the emotional thresholds defined in the definition struct and instantiates it's own emotions.

```
//Behavior tree as defined in “EmotionHandler.h”
//Takes arguments of an emotion map reference and self identification
EmoteTree emoteTree = EmoteTree(emControl, thisc);
EmoteTree::Selector selectors[3];
EmoteTree::Foo foo = EmoteTree::Foo(emoteTree);
EmoteTree::WellBeing wellBeing = EmoteTree::WellBeing(emoteTree);
EmoteTree::Attribution attri = EmoteTree::Attribution(emoteTree);
EmoteTree::CompoundEmotions compSelf = EmoteTree::CompoundEmotions(emoteTree);
EmoteTree::ProspectEmotions proEmo = EmoteTree::ProspectEmotions(emoteTree);
```

The emotion handler system organizes the emotions towards itself as well as all other characters in an unordered map utilizing the key defined by the users upon character instantiation along with a vector containing all emotions aimed at the character. Upon the creation of another character the emotion handler creates a map entry of netural

emotions towards it.
Eliciting factors meant for the tree are put in a deque, which is then popped each update tick and inserted into the tree.
Other than controlling the inputs to the emotion tree, the emotion handler also returns the strongest emotional intensity for the character, for use in the system upon the user's request. The final intensity return will be discussed in detail after the calculations have been explained.

Eliciting Factors

Eliciting factors serve as three purposes in and outside the context of the system. The first is to inform the system that something has happened that might elicit an emotional response. The second is to encapsulate the context specific variables that are used to calculate the emotional potential and intensities. The thrid is to serve as a reference to what an emotion is in reaction to.

Eliciting factors are defined outside the parameters of the system, to then be passed to the specific character instance that's supposed to respond to the eliciting factor. The eliciting factor should be instantiated when a recognized event or action is recognized by the agent. All values contained in the individual eliciting factors are by necessity set outside the context of the system when the specific eliciting factor is instantiated.

The input variables of each eliciting factor serve the same purpose as the modifier values defined by the OCC model for each model group. All values are represented by both a value and a weight contained in a pair.

There are 4 main eliciting factor classes, where each is representative of the context of the individual emotional group of the OCC model (e.g. Fortunes for other, Attribution, etc). Each of the 4 eliciting factors derive from a base eliciting factor class that contains the general intensity variables as well as other variables with similar applicability to all emotions (such as desirability and praiseworthiness).

```
//Example of eliciting factor for joy/distress valence
class ElicitWellBeing : public ElicitingFactor
{
public:
	ElicitWellBeing(std::pair<double, double> desirability, std::pair<double, double> senseofreality,
		std::pair<double, double> proximity, std::pair<double, double> unexpectedness)
		: ElicitingFactor(desirability, senseofreality, proximity, unexpectedness){}

	~ElicitWellBeing(){}

	int getType() override { return eWellBeing; };
};
```

Three emotional groups are treated differently in regards to eliciting factors, these are the Compound Emotions and the Prospect Emotions. The first of which acts entirely on the result of two other groups (i.e. Well-being and Attribution), as such the group is not represented by an eliciting factor. The latter emotional group acts in two stages and as such is represented by a special eliciting factor that has two representative states. One represents the pre-response to a prospective event (i.e. Hope/Fear), while the other represents a response to the event once it's happened (e.g. Relief/Disappointment).

Emotion Tree

The hierarchical structure of the OCC can clearly be seen as a tree. To suit this the most fitting and tested tree architecture is that of a behavior tree. A behavior tree is in essence a serialized serialized form of a finite state machine, commonly used in video game AI. Unlike serialized state machines, behavior trees use tasks rather than states. This makes behavior trees powerful in that they can model complex behavior through a combination of smaller tasks rather than using static states as behavior. Similar to other tree architectures, the states are represented as nodes of the tree, with a series of nodes that decide the flow towards the varying functions. The tree flows from the predefined root following the tree until it reaches an end (a leaf node) or fails at a node prior to the end. This results in three primary statuses (Success, Failure and Running), that define the

operations status of the tree throughout the operation. Though any number of statuses are possible depending on the implementation needs. The nodes are divided into three main archetypes composite nodes, decorator nodes and leaf nodes. As previously mentioned the leaf node represents the end of a branch and as such does not have children to which it can continue and cover the end functionality of the trees selection. While the composite and decorator nodes both serve to direct the flow of the tree towards an end in various ways. Composite nodes can have multiple children. Composite nodes can have a few varieties where the most prominent are selectors and sequences. Selectors act to select between multiple paths towards a leaf, where it in serialized or randomized order, returns the first (if any) value to be true. Sequences act to run multiple leafs in a sequence, this allows trees to exhibit complex behaviors the require multiple leafs. Unlike composite nodes, decorator nodes can only have one child and acts to modify its result (by for instance inverting it). lx

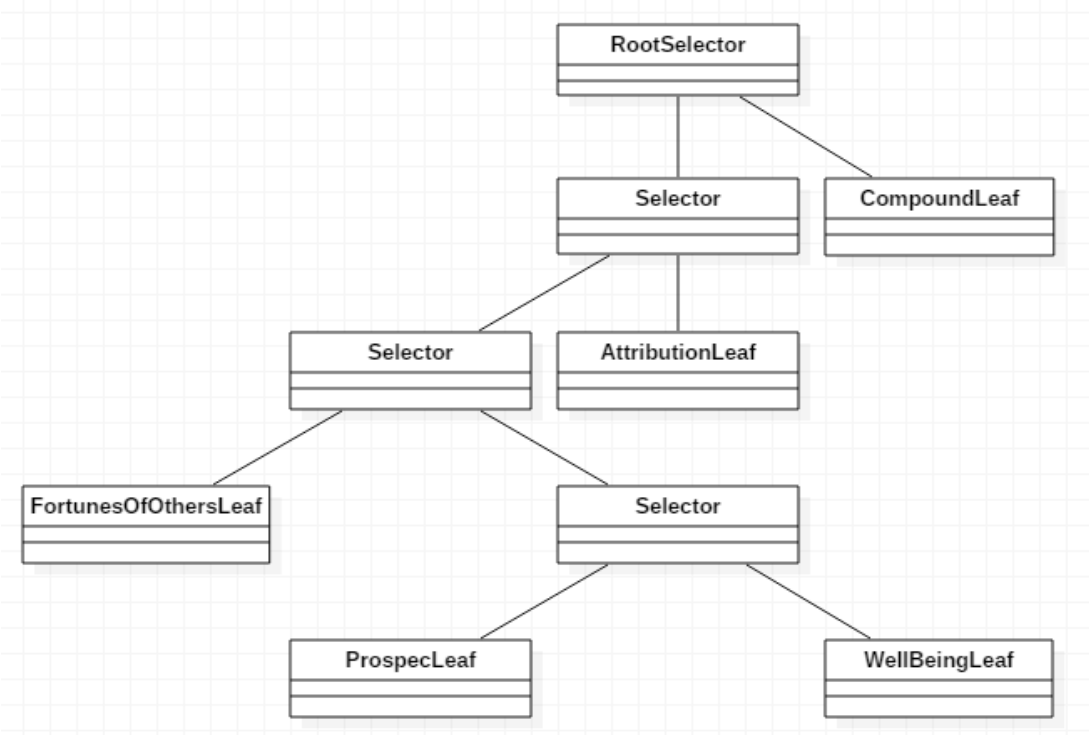
The tree of the emotion system is defined, instantiated and run from the emotion handler.Though the tree architecture is fairly simple, it acts as an integral part of the emotion handler. It provides a framework and an order in which the emotions that are modified are controlled. The tree follows a left to right priority order where it first checks all emotions that utilize the eliciting factors. The left to right order is dictated by the part of the tree OCC model utilized by this implementation (i.e. excluding the 'aspects of objects' branch), as such the leaf call order executes the leafs in the order of:

1. Fortunes of others
2. Prospect
3. Well-being
4. Attribution
5. Compound

Only one of the leafs can be executed in a run through of the tree, as such the compound emotions will only be executed if there are no new eliciting factors available. Though there is never a direct interaction between the outside system and the tree, the actions of the tree are dictated by the situational information of the eliciting factor given.

The importance of the eliciting factor and behavior tree system is to make complex

emotion elicitation easily and intuitively differentiable from one and other in the respects to the event or action that elicits the emotion. Rather than the user of the dll for instance calling and modifying an anger variable directly, it is important to have a clear connection between the eliciting factor and the specific emotion elicit. This is especially beneficial in relation to emotions that can be semantically similar in modern vocabulary (e.g. gratification and gratitude), and serves to show the situation an emotion should be used. It also allows for emergent behavior through the elicitation of emotions that differ based on user defined criteria.



Emotional Value Ranges and Variations

As valenced emotions are considered as mutually exclusive lxi the two sides of an emotion share one variable. The variable is a double value, where the value is a default at 0 until an appropriate emotion at which point, the value set, is normalized between the values of 1 and -1, where positive values represent a positive valence and negative values represent negative valence. Due to the nature of this project, value scaling of the emotional is seen as case specific. The initial value of an emotion is as such set through the eliciting factor.

As explained in the literature section, the value of the emotion is also checked towards a weighting from which an emotion can be modified. Like the intensity value is represented by a double value. The value is also normalized between 1 and -1, however differ in that it stays constant as long as the character is active. The threshold is set at the instantiation of the character class as a user preset as defined in the character.

A special case in the subject of range is in regards to the emotional potentials. Rather than following a single range between -1 and 1, the potential is represented by two individual ranges; a positive range between 0 and 1, and a negative range between 0 and -1. The details of this will further be discussed in the context of the potential calculations.

Emotional Potentials

The next step for the algorithm is deciding the potential intensity of the emotion based on the desirability of the result as well as any other intensity modifiers. In the original model, the potential intensity is defined in terms of $Pe(p,e,t)=fe(|D(p,e,t)|,Ig(p,e,t))$ (usually including emotion specific variables as well). This does however not take into account how the intensity modifier weight is to be used nor how the global intensity modifiers are added into the result. Furthermore, it does not describe what kind of function the emotions should be used for the calculation. Therefore, unlike the standard variation of the OCC model, the potential intensity is calculated by taking the mean of a function applied to each intensity modifiers.

Each individual intensity modifiers of the OCC model are rated on a logistic response curve function. The curve is based on a modified Sigmoid Function (i.e. $s(t)=\frac{1}{1+e^{-t}}$), as described by Dave Mark (Architecture Tricks: Managing Behaviors in Time, Space, and Depth, 2013, 40:25).

$$y=\left(K\left(\frac{1}{\left(1+\left(\left(1000\,em\right)^{-x+c}\right)\right)}\right)\right)+b$$

The curve is modified by four additional variables (m, k, b and c) as well an additional

constant value (1000).

- 'm' sets the slope of the line at the inflection point.
- 'k' affects the vertical size and direction of the curve, hence a value of 1 would result in an upwards slope, while -1 would result in a downwards slope.
- 'b' shifts the y-intercept of the curve, hence shifting the curve vertically from said value.
- 'c' shifts the x-intercept of the curve, hence shifting the curve horizontally by said value.
- The additional constant (1000) acts to correct the slope in the cases of large 'm' values and keep the result within the given range.

```
static double const LogCurve(double m, double k, double c, double b, double x)
{
        //Calculate e taking into account slope of line at inflection point m
        double em = (1000 * loge * m);
        //Calculate exponent for slope taking into account horizontal shift
        double xc = -x + c;
        //Create curve base
        double pw = 1 + pow(em, xc);
        //Truncation of decimals to allow for values such as 1.0 and 0.0
        double tr = std::floor(pw * 100) / 100;
        //Calculate the final slope taking into account vertical size and shift of slope
        return (k * (1 / tr)) + b;
}
```

In the context of the implementation, the two inputs of the curve algorithm are the value and weight of an intensity modifier, where each variables need to be normalized within the range of -1 and 1.

The value of the intensity modifier is used in two contexts within the calculation. The first is that the absolute of the value represents the 'x' to which the corresponding result can be derived. The second is that the value's inflection affects the 'k' value, hence an intensity modifier value that is less than 0 will result in a 'k' that is set to -1, otherwise it

will be set to 1.

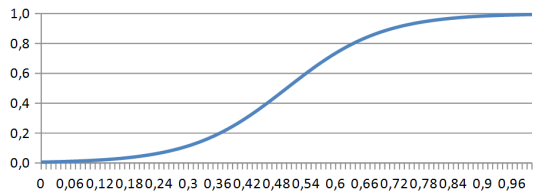
The intensity modifier weight is used to affect the 'c' and acts value to make it easier or harder to reach the end value of the curve. However, in order for the correlation between the weight and the 'c' value to be concise (e.g. a high weight makes it easy to achieve a high value on an increasing curve), the remainder of the weight subtracted from the normalization's maximum value (in this case 1, i.e. $c=1-|(weight)|$) is used rather than exact weight.

Due to the x and c value of the curve being rated on a positive scale between 0 and 1 no matter the direction of the curve, they will always be calculated in terms of absolute values.

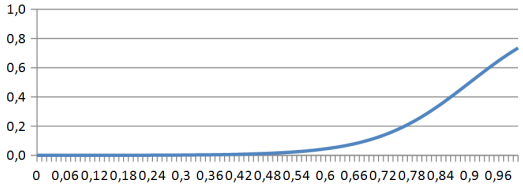
```
double resCurve(double x, double c)
{
        double invc = 1 – abs(c);
        return AddMath::LogCurve(10, x < 0 ? -1 : 1, invc, 0, abs(x));
}
```

The remaining variables ('m' and 'b') are in this context preset constants. Where: 'm' is set to a value of 10, hence giving the curve an angle of approximately 68 degrees at the inflection point, and 'b' remains as a constant 0 to allow for a slope that can range either between 0 and -1 or 0 and 1 in correspondence with 'k'.

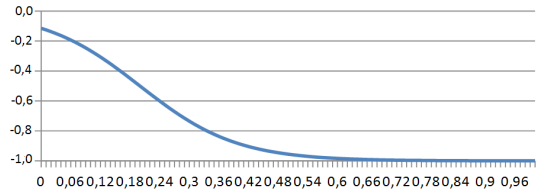
The resulting curve can be visually represented in terms such as these:



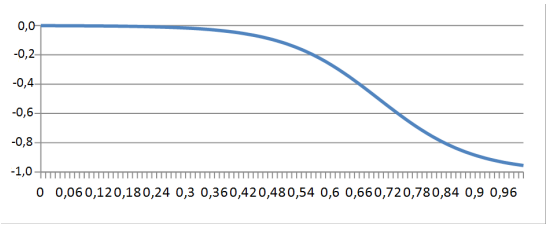
m = 10, k = 1 b = 0 c = 0.5



m = 10, k = 1, b = 0, c = 0.9



m = 10, k = -1, b = 0, c = 0.2



m = 10, k = -1, b = 0, c = 0.7

The resulting y value (correspondent to the x value) is thus representative of that variable's contribution to the potential intensity of emotions (normalized between -1 and 1). Though most cases the unaltered result (y value) of the curve is used in the mean calculation, there is a deviation in the case of the 'Liking' modifier of the Gloating/Pity valence. For this potential to be calculated, the desirability of the event for the other needs to be negative. As such, the agents liking of the other is inverted to reinforce the feeling the other has towards the event (e.g. if the other dislikes an event and the agent likes the agent it will either reinforce how much the agent dislikes the event). The mean of the results of the various intensity variable curves affecting the emotion are then passed to a function that adds the value to both the positive and negative potentials.

```
//p is applied to emotion i, directed at person per
void modPotential(int per, int i, double p)
{
        double pos = *aRef.emotes.at(per)[i].emotePot.posPotential + p;
        *emotes.at(per)[i].emotePot.posPotential = AddMath::clamp(pos, 0.0, 1.0);
        double neg = *aRef.emotes.at(per)[i].emotePot.negPotential + p;
        *emotes.at(per)[i].emotePot.negPotential = AddMath::clamp(neg, -1.0, 0.0);
}
```

As previously discussed, the potential intensities are represented by two independent value ranges, one representing the positive valence and the negative valence. This is firstly done to differentiate between the potential of the positive and negative valence, but secondly to allow for changes to happen more rapidly. The potential is added to both sides of the valence to account for the relative decrease in one valence on the increase of

the other.

```
//Example of attribution leaf algorithm
//Calculate the curve of desirability (same as praiseworthiness)
double D = resCurve(eF->Desirability().first, eF->Desirability().second);
 //If the praiseworthiness of the action is not neutral (0)
if (D != 0)
{
        //Calculate the curves of expectation deviation and strength of cognitive unit
        double Exd = resCurve(eF->ExpDev().first, eF->ExpDev().second);
        double Scu = resCurve(eF->StrCogU().first, eF->StrCogU().second);
         //If scu (unit affiliation) is greater neutral or the action is attributed to this unit
        if (Scu > 0 || eF->CharID() == thisID)
        {
                //Set potentials and test and mod final intensity of pride/shame valence
                modPotential(thisID, PS,(D + Exd + Scu + Ig()) / 6);
                intensityCalc(thisID, PS, PrideShame);
         }
         else
         {
                //Else set potentials and test and mod final intensity of admiration/reproach valence
                modPotential(eA->CharID(), AR, (D + Exd + Ig()) / 5);
                ntensityCalc(eA->CharID(), AR, AdmirationReproach);
         }
}
```

Due to their reliance on the intensities of other emotions, rather than modifying values and thesholds set by an eliciting factor, the compound emotion's potential calculations deviate from that of the other emotions. The compound emotions are all calculated in iteration through a loop, regardless of who they're aimed at.

The algorithm starts by checking what character the iteration of the loop applies to. It then takes the values of the joy/distress valence and the relevant valence for the character (pride/shame if it applies to the current agent, if not admiration/reproach). It then checks whether the two intensities share the same valence. It then uses the valence of the joy/distress valence to determine the inflection of the resulting potential.

Rather than calculating the potentials through the sum of the results of the modifier

value curves, the compound emotion potentials are calculated on a varied version of Carlson and Hellevang's anger calculations. The alteration made to the formula have been made to take into accout negative emotions, as Carlson and Hellevang don't address this point. The potential is as such defined by the formula $\left(i * \sqrt{(|(Ij * Ip)|)}\right)$. The complete algorithm for the leaf is formulated as such:

```
//Compound emotion algorithm example
//For all characters
for (auto kv : emotes)
{
	//Check if self
	bool self = kv.first == thisID;
	//Get intensity of joy/distress valence
	double Ij = *emotes.at(kv.firsts)[JD].intensity;
	//If self get intensity of pride/shame valence else get intensity of admiration/reproach
	double Ip = *emotes.at(kv.first)[self ? PS : AR].intensity;
	//If the two intensities share valence
	if ((Ij * Ip) > 0)
	{
		//Get inflection of joy/distress valence
		int i = ( Ij > 0 ? 1 : -1);
		//Modify potentials, test and calculate intestity for emotions cooresponding to person
		modPotential(kv.first, self ? GR : GA, (i * sqrt(abs(Ij * Ip))));
		intensityCalc(kv.first, self ? GR : GA, self ? GratificationRemorse : GratitudeAnger);
	}
}
```

Resulting Intensity

Though Ortony, Clore and Collins don't define explicit formalizations for all emotions, they do define multiple from which further formalizations can be derived. The structures of the various emotional algorithms are similar to one and other. Where most of the variation between them comes from the intensity modifiers to the parameters of the algorithm.

The final emotional intensity is tested directly after an emotion's potential calculation. The inflection of the emotion is decided by the highest relative potential (of the negative

or positive inflection) that is also above it's respective threshold. If one of the potentials for that emotion meets it's respective criteria, the final intensity is calculated for the emotion. If neither of the potentials meet the criteria of the test, the intensity is set to neutral (0).

Unlike the convention defined by the OCC model, the final intensity of the emotion is calculated by another curve, rather than the difference between the potential and the threshold. The curve uses the same parameters as the potentials to calculate a similar curve, with the potential of the emotion representing 'x' and the threshold of the emotion representing 'c'. This deviation from the OCC model is to allow for the possibility of a full range of emotional intensity values regardless of the value of the respective threshold.

The resulting emotion value is represents the emotional intensity for the specific instance.

Personality System

The second core subsystem of the project is the personality system. For the purposes of this system the most suited personality model has been deemed to be the Zamora model.

The Zamora model also has the bonus of embodying a lot of the positive aspects of the other models, while discarding a lot of the negative aspects.

Similarly to the scalarity of the Big-Five model, the scalarity of the individual attributes of the Zamora model allow for greater variation in personality combinations, and gives the attributes the possibility of being used as modifier values to calculations on the probability of actions. Simultaneously, the dual poles the attributes (similar to Myers-Briggs) allow for the inflections of the variable to be used in the capacity of boolean states.

One of the main points of criticism leveraged towards the Big-Five models are their lack of taxonomical breadth (Big-Five 5 moderates and 5 extremes). This has led to psychologists like Dan P. McAdams to dub the model the “psychology of the stranger”

lxii. The increased number of attributes of the Zamora mode (20 attributes), not only makes it simpler to derive behavioral effects based on it's attributes, but also an increased breadth of combinatorial affects. It also addresses the point made by McAdams by incorporating both socially and personally aimed attributes, hence also giving the use of these attributes, a specific metric differentiation for variations between social and individual actions.

Finally, the findings reported by Jensen and Nes, show a noticeable advantage in agents that implements the Zamora model from those that do not. As such showing the viability of the Zamora taxonomy in a game setting.

Personality Taxonomy and Attribute Values

As stated in the literary section**,** the Zamora taxonomy includes 20 attributes in total (10 personal and 10 social). Though the taxonomy of the breadth gives a good description of many aspects of a human personality, there are certain personality attributes that don't suit an agent. The personality attributes deemed suited by this model are: Achievement attributes, emotional temperament, intellectual factors, risk factors and task performance (for the individual attributes), and aggressiveness, control attitudes, dependability, egocentrism, emotional expression. Fairness, leadership skills, regard for rules and team spirit (for the social attributes).

For the individual personality attributes the attributes that are deemed to be suited and unsuited are:

1. Achievement attitudes – The goal motivation is beneficial for any goal based architecture it can give a strong indication as to whether the agent will focus on a single goal or scatter to achieve multiple.
2. Emotional temperament – The emotional temperament is important for the relation to the emotional system and it's reaction to eliciting factors as it can serve to determine the chance that an agent will pay attention to it.
3. Energy level – The energy level is not important in the system as of now as the

energy of an agent can be determined quite easily by external factors that define the activity of the character.

4. Intellectual factors – Intellectual factors can be beneficial for creating differentiation in how efficiently an agent will be able to solve tasks layed before it, such as solving a puzzle.
5. Material attitudes – Though not beneficial to the system at this stage the material attributes can serve as a factor in augmenting the emotions from the 'aspects of objects' branch, as well as serve to modify more advanced agents.
6. Maturity – Maturity is deemed unusable by this system because it serves little purpose without an extension of time, from which a character could mature. Additionally, the functions of this attribute can in large part be parted on to other attributes of the model.
7. Philosophical attitudes – Philosophical attitudes is beneficial to how a character will see an eliciting factor or task.
8. Physical attributes – Physical attributes is non-beneficial to the system as they represent factors that are more easily defined through visual means than behavioral means.
9. Risk attitudes – Risk attitudes are beneficial to determine whether the agent assesses the risk of a situation before acting upon it. An agent with negative risk attitudes might be more inclined to do something without considering the outcome, while an agent with a positive might be more inclined to consider the possible outcomes before acting.
10. Task performance attitudes – The task performance of a character is a good indicator to whether the agent will be able to perform the task it has been set to do, it is hence beneficial for determining the probability of a desirable outcome to the task the agent is performing.

For the social personality attributes the attributes that are deemed to be suited and

unsuited are:

1. Aggressiveness – The aggressiveness of a character can be used to define a wide range of behavior. An aggressiveness character might be more inclined to attack someone and elicit negative emotions.
2. Control attitudes – Control attitudes, along with dependability and egocentrism can work as a good indicator to whether an agent is likely to be a good ally or team mate. In the case of control attitudes it can define the chance of an agent acting domineering towards it's fellow allies.
3. Dependability – The dependability of a character is good indicator for whether an agent is likely to help an ally in need. It can also determine whether an agent is likely to break agreements made to serve it's own goals.
4. Egocentrism – An egocentric agent might be less inclined to share resources and assist others through virtue. It can also be an indicator on whether an agent is likely to break agreements made to serve it's own goals.
5. Emotional expression – The emotional expression of a character defines in large part how the emotions intensity of an emotion can unfold. Following
6. Fairness - The fairness of an agent helps determine whether a character is trustworthy in a team context and how likely the agent is to cheat in a game.
7. Leadership attributes – The leadership attributes can determine a lot more than just the ability an agent has to lead. It represents the tendencies the character has to lead in a positive or negative way. Hence, it is beneficial for the
8. Physical appearance – Similarly to the physical attributes, the physical appearance of a character is more characteristic of the visual design of an agent and as such excluded from the model. It can however, be developed to be used with systems that allow the agent to act to how it looks, such as an agent with a procedurally generated visual design.
9. Regard for Rules – The agent's regards for rules is important for determining whether the agent will play fairly in regards to it's opponents and allies. It is an

important factor in how the agent will deal with a situation where it can play unfairly to gain an advantage.

10. Team Spirit – Team spirit is the most important in regards to diplomacy. It serves as the main variable determining whether an agent is likely to stay in an alliance and whether to be helpful towards it's allies.

Further descriptions of how these personality attributes can be utilized by an agent is described in further detail later in this paper. All traits are clamped between the maximum value of 1 implying a positive version of the trait, a minimum value of -1 implying a negative version of the trait and a value of 0 representing the positive or negative neutrality of the trait.

Personality Classes

In the balance of the system, the personality of a character is defined as the static factor, to balance the dynamic factor of the emotions. The personalities are to a certain extent a behavioral baseline, able of determining both emotion dependent behavior and non-emotion dependent behavior. The personality of the character is defined throughout three classes. The main of these is called a personality template. The personality template's main function is as a wrapper class for the two personality attribute container classes. The personality attribute classes serve only one purpose in the system, to define and hold the values of the personality attributes in an encapsulated form.

Personality Based Compatibility Score

In regards to the opinion of another character the personality taxonomy acts as a modifier for emotions. Additionally, this can work as a general indicator of how well a character is compatible with another without regarding the allegiance of the other character and without taking emotions towards the character into account.

As the theory of the Zamora personality attributes states, individual attributes clash with their opposites and negative social personality attributes act negatively towards how the

character is perceived by others.[25] [26]

Based on this it can be assumed that the opinion of someone else can be based on how much of the two persons personalities differentiate from each other. As such, the individual attribute half of the compatibility score can be defined by taking the remainder of the absolute distance value between absolute values of the two corresponding individual attribute values. The resulting value can then added or subtracted from the resulting sum based whether the inflection of the two character's attributes are different from one and another.

```
void PersonalityTemplate::CompatiblityScore(int j, PersonalityTemplate* oT)
{
	double pers = 0;
	//Loop through the individual attributes of this character (p) and (o)
	for (int i = 0; i < TotalPersonal; i++)
	{
		double p = Personal()->GetByNumber(i);
		double o = oT->Personal()->GetByNumber(i);
		//Take the
		double op = 1 - (abs(p) - abs(o));
		pers += (p * o > 0 ? op : -1 * op);
	}
	pers /= TotalPersonal;

	double soc = 0;
	for (int i = 0; i < TotalSocial; i++)
	{
		double o = oT->Personal()->GetByNumber(i);
		if (o < 0){   soc += o; }
	}
	double compS =(pers + (soc / TotalSocial)) / 2;
	compScore.emplace(j, compS);
}
```

25 Personal attributes in the theory of Zamora (Retrieved 10/06/2015) http://www.scientificpsychic.com/workbook/person1.html

26 Social attributes in the theory of Zamora (Retrieved 10/06/2015) http://www.scientificpsychic.com/workbook/person2.html

The second part of this calculation works differently from the first in that the social attributes of the two characters are not compared, but rather the character making the compatibility score takes the sum of all the negative attributes the other character has and divides it by the total number of social attributes, to create the mean value.

Finally, the resulting means of the individual and social calculations are added together and divided by two to create the final mean. This final addition means that the value can not exceed 1 or be beneath -1, due to the same limitations enforced on the personality attribute values.

The personality's compatibility score is a determination of how well two agents get can get along. It should not however, be seen as an analogue for how much an agent likes another, though it can be used as part of the determination. Though such a definition is outside the scope of this system.

This model does however, not take into account how well a character knows another and does additionally not allow for miss-attribution of personality attributes a character might make about another.

Personality Based Emotional Thresholds

The thresholds of the emotional system are defined through the combination of personality attributes.

Two thresholds are calculated individually for each emotional valence of the emotion system, one per side. Due to the either positive or negatively charged connotations of the names of the traits (e.g. aggressiveness has a negative connotation) and the input values that they logically entail. Each of the traits are first inverted to suit their intended meaning in the calculation. This means that traits with negative connotations will affect positive valence by giving them a higher value (hence making the emotion harder to achieve) if the value is negative.

As a rule, most personal emotional thresholds are calculated purely from the personal

personality traits, while most the social emotional thresholds are calculated purely from the personal personality traits. This was decided to most accurately reflect the emotional direction as both the emotion and personality taxonomies direct themselves towards either personal or social reflections. There are some exceptions to this rule as some personal emotions are often affected by the actions of others and some social emotions are affected by personal ones.

All thresholds use either emotional temperament (for personal emotions) or emotional expression (for social emotions) as a baseline to represent a characters positive or negative tendencies when emotions are elicited. As such, their inclusion in emotional thresholds will not be discussed in the various examples.

There are also some further exceptions in terms of prospects of events or compound emotion, however these will be discussed in further detail within their relative examples.

Each personality trait is first calculated to represent the 'positive' valence of the of the trait. This means that any negative valence will be calculated as the remainder of the trait's absolute value the maximum value (1). The positive valenced thresholds are then calculated by an average of the personality attributes deemed relevant for the given emotion. Once the positive emotional threshold for a valence has been calculated, the result is inverted to the negative valence by a similar method as the used for the traits in the positive.

This allows for even personality traits of a valence of opposite value to that of the valence being calculated affect the resulting value of the threshold calculation to some degree. This is done under the understanding that for instance someone who is prone to anger might be less prone to gratitude, however that what it lacks it lacks in anger might affect it's chance of gratitude and vice versa.

The emotional thresholds are calculated with the following values:

- The Joy/Distress valence thresholds are decided by just the Emotional

Temperament of the character. This is based on the OCC theory, which lists it as a prospect irrelevant. As such, this emotion is based little outside of purely the specific event that makes the character feel either joy or distress.

- The Pride/Shame valence thresholds are special, because they can apply both to the character or characters associated to it. As such, they are decided by a combination of both personal and social personality traits; these are the Emotional Temperament, Achievement Attributes, Task Performance and Egocentrism of the character. This is done to represent how much stock the character puts into achievements made, how much pride it takes in its work, and how much and highly the character thinks of itself and its associated group.

- The Happy-For/Resentment valence thresholds are decided by the Emotional Expression, Team Spirit and Egocentrism of the character. A character with high team spirit might be more inclined to support an ally or resent a foe for the outcome of a desirable event. This is then offset by the characters egocentrism to show how likely it is to think favorably for friends and negatively for non-friends.

- The Gloating/Pity valence thresholds are decided by the Emotional Expression, Team Spirit, Egocentrism, Aggressiveness and Fairness of the character. Similarly to previous emotional thresholds team spirit and egocentrism represents pretty much a similar function as within the Happy-For/Resentment valence thresholds. Aggressiveness is added to represent the characters tendency to aggressively gloat to non-friends or a lack of empathy towards friends. Fairness represents how fairly a character tends to look at events, if the character is it will be less likely to gloat, while if it's not fair it will be less lightly to pity.

- The Admiration/Reproach valence thresholds are decided by the Emotional Expression, Team Spirit, Control Attitudes, Aggressiveness and Fairness of the character. Team Spirit, Aggressiveness and Fairness all represent the same

function as the previous valence thresholds. Control Attitudes serves to represent whether the character has healthy or unhealthy control attitudes towards others. Someone who is demeaning and controlling is less likely to feel admiration towards another characters actions and achievements, while more likely to feel reproach at any slight or action seen as negative to the character.

- The Hope/Fear valence thresholds are decided by the characters Emotional Temperament, Philosophical Attitudes and the Risk Factor traits. This is done to reflect how the character analyzes an event, whether it has a positive or negative outlook, and whether it is willing to take risks or risk averse in the prospect of said event.

- The Satisfaction/Fears-Confirmed and Relief/Disappointment valence thresholds are both represented by variations of the Hope/Fear valence. Both valences are interconnected can be seen as direct reflections of each other; in that satisfaction and disappointment can both be seen as results of hope, while fears-confirmed and relief can both be seen as results of fear. As such, the Satisfaction/Fears-Confirmed valence thresholds have the same values as the threshold for the Hope/Fear valence, while the Relief/Disappointment valence thresholds represents a direct inversion of the same values. This follows the understanding that for instance a character with a high chance of feeling a high value of hope at the prospect of an event, would also be inclined to feel a high level of satisfaction or disappointment given the outcome of said event.

- The thresholds of the two compound valences (Gratification/Remorse and Gratitude/Anger) are decided in the same way. Both represent the average of the thresholds of the emotional valences that affect them in the OCC model. As such, Gratification/Remorse represents the average of the thresholds for Pride/Shame and Joy/Distress, while Gratitude/Anger represents the average of the thresholds for Admiration/Reproach.

It can be argued that the social emotional thresholds should vary depending on who the

emotion is directed at. Though the personality based compatibility score could be used to fill this function, it was decided to not be a feasible option as the author sees this as a value that must be paired with another to give an accurate understanding of one characters 'liking' of another. Otherwise, no feasible way was found to achieve this within the constraints of the current system.

System Interfaces

The system contains a number of internal and external controllers and interface that act to relay information inside and to outside systems. Some of these have been detailed within previous chapters, as such they will not be described in detail here. To allow for easy interfacing with the system, a wrapper class was created to handle internal calls, data collection and outputs.

Characters System

The first of these systems and the first class is the character handler class. The class acts to instantiate, destruct and maintain the character classes. It also acts as an interface towards outside systems (wrapper classes notwithstanding) and as such is an important, yet non-complex class of the system.

When the initiation or a termination of a character occurs the character handler informs the all characters and also their subsystems that they need to update in regards to the specific character. Upon the instantiation of a character the handler loops through all other characters and mediates as the characters calculate their respective opinions of each other.

The handler identifies each individual character by a user defined integer identification, which also carries similar importance throughout the other systems. The characters and the identification are organized in an unordered map.

```
//Instantiate new character i
Character c = Character(i , EmotionHandler(i, chr.emoThresh), PersonalityTemplate(*chr.pers, *chr.soc));
for (auto kv : chars)
{
        // Cross set opinions for all characters
        c.instOpinion(&kv.second, kv.first);
        kv.second.instOpinion(&c, i);
}
//Add character to map
chars.emplace(i, c);
```

The character handler also serves to update all active characters to update their internal functions and tell them about other characters about one characters removal.

In addition to adding and removing characters, the character handler allows access to check whether a character exists, the number of characters, a function to get a specific character and the possibility to fetch all character IDs currently active in the system.

Character

The character serves a similar system to the handler in that it serves to handle other classes, therefore it is in large part a wrapper class with functionality specific towards the emotion and personality systems.

The character The character opinion is the total of all things a character feels towards a character (including one self). Opinions in the context of the character class are references to the other character and their respective ids. As such the data for said opinion is stored within the personality and emotion handler classes inherent to the character.

```
//Instantiate opinion of character i
void Character::instOpinion(Character* i, int o)
{
	Emotion()->InitSocEmotions(o);
	Personality()->CompatiblityScore(o, i->Personality());
}
```

The characters only function as an interface is to pass pointers to its emotion and personality functions.

Emotion Interfaces

The emotion handler itself contains a multitude of interfaces both for internal and external use. Internal to the system the emotion handler allows for initialization of a set of social emotions towards another character, the removal of emotions towards a given character and updating of the emotions (which runs the calculations). Though these functions can be called outside of the system, it is advised to let the character and character handler classes organize their functions.

Outside the system, the user has three alternative definitions for what emotion is to be returned at a given call as well as functions to add eliciting factors to the emotional system.
The three emotional return methods either return the strongest emotion, the strongest emotion aimed at a given character (including self) or a specific emotion.
These three varieties are important to different situations where various emotions are relevant.
The strongest emotion aimed at a character is important for cases where a character might need to act on an emotion that is not necessarily the strongest overall. In a concrete case, the character might need to interact with another character that the character feels mostly anger towards. In this interaction the character would likely act

on the anger rather than it's overall strongest emotion (if different).

In other cases the character would want act on the strongest overall emotion. Finally, all other cases can be defined by the specific emotion. In general one can act on any one of the emotions, one might be feeling. This case is important to note, because it allows for situational emotional modification outside the bounds of the strongest emotions. As such a character might be able to act on multiple emotions for various functions.

The emotion is returned in the form of an emote struct defined by:

- Who the emotion is directed at through the character ID defined for the character upon instantiation.
- An integer to the enum name of the emotion that was elicited (for easy lookup in an an array).
- The value intensity of the emotion.

This combination of variables give the possibility of a large variety of use cases for the returned emotion.

```
struct Emote
{
        int directedAt;
        int name;
        double intensity;
};
```

All output intensities for valenced emotions are normalized within the range of -1 and 1.

Personality Interfaces

The personality system includes similar internal and external interfaces of similar functionality as those use for the emotion handler. Internally the character handler can make the personality template calculate or remove compatibility scores, and calculate emotional thresholds to be sent to the emotion handler. Externally, the personality template allows for the user to get the compatibility to each other character, as well as pointers to the personal and social personality values.

Convenience Interface

During test environment implementation, it was decided to create another interface for convenience in data retrieval to be sent to the front-end layer of the game. Though this interface can be fully ignored for full utilization of the system, it serves as a convenient way to gather data in cases where constant fetching of data might be a constraint, while also being a good way to access functionality the entirety of the system. Hence making easier to use for less experienced users.

The interface was originally designed for the use express use of transferring data to the test environment, however, after some development it started to take on a form that would be useful in a large amount of settings. As such it was decided that it should be a part of the personality and emotion system in it's own right.

The interfaces key functionality is to act as as an outer interface to all the functionality of the library, while also adding some useful additions to make fetching of the personality and emotions of a character easier for systems where they can't be called at will at any point (such as the test environment). In addition to providing abstractions of many of the functions of the character handler (e.g. start character, end character, update character); it also has the possibility to gather complete emotional profiles (emotion states), personality taxonomies, compatibility scores and add eliciting factors to a given character. Thus making it possible to get full use of the system through just one interface.

The emotion states are a full context of what emotions a character is feeling. They act to send all information of the various emotions in batch, which is convenient when multiple data transfers can be a hindrance to performance; such as when sending data between two asynchronous layers.

Test Environment Implementation

As discussed in the introductory section**,** the personality and emotion system is designed to act and be utilized outside one specific system. It's use case may as such contain details that are non-specific to the test environment, however the majority of its functionality is included in some way to prove the viability of implementation. Some possible functionality was excluded due to a high possibility that they would give away the test, these as well as further possibilities for the system will be discussed in the further development section at the end this thesis.

As previously discussed in the literary section; the architecture of the 0 A.D. game contains two layers: a back-end engine called Pyrogenesis developed in C++ and a front-end scripting system developed in JavaScript; with the decision making parts of the Petrabot developed in the latter. These two layers use the SpiderMonkey engine for intercommunication.

As the personality and emotion system was developed in C++, the natural entry point was through the Pyrogenesis engine. The engine is comprised of 15 individual component libraries that each serve a main function within the system and a main project to run the solution. The personality and emotion system was developed into the Simulation2 library, which handles the back-end logic and calculations for the individual AI.

The Simulation2 library was modified to include additional methods to convert emotion and personality specific data to and from JavaScript, as well as multiple interfaces that can be called by the bots to trigger the conversion and transference of data.

All data transfers happen on the request of the currently active bot. There are two rationals behind this approach. The first is to avoid lag caused by timing mismatches, due to the asynchronicity of the two layers. The second is to avoid cases where the data might be sent to the wrong bot; which could cause crashes of the game in cases where non-modified bots (which don't have systems to maintain the data) are used or cause a mismatch between the data being sent and the bot receiving it in cases where modified bots are used. Both of these cases would render the emotion and personality systems

unusable in this game.

PetraBot Modifications

The implementation into the PetraBot aimed to expand the current functionality of the bot using the emotion and personality inputs. The modifications focus mainly on three functions of play: defense, attack and diplomacy. The additions to the PetraBot mainly came in the form of modifying already programmed functions to allow for greater variability in respect to the emotional and personality outputs.

At the start of a game a modified PetraBot will prompt the system to instantiate a character is instantiated for each player involved. The system will then instantiate a character for each player, even if it's the player or one a non-modified bots. This is done to allow for modified bots to have opinions of all the characters play, even though they might not have them about it. This is only done in games where one or more modified bots are involved.

Personality in the bot

Once the characters have been instantiated, the modified bot will proceed to fetch the personality attributes and the compatibility the character has to all other players. Though the personality attributes don't serve a prominent role directly in the actions of the bot, they do serve a large indirect role through the personality variables defined by the PetraBot. This was a design decision that was made for two main reasons. The first is that the personality traits already defined in the PetraBot serve a very similar purpose as the ones from the Zamora would serve in a game such as 0 A.D. In effect this means that the personality attributes are in large part used to instantiate other behavior modifying variables.

In this case the base values of the internal agressiveness trait is defined by the average of the inverse of the character's risk factor and the inverse aggressiveness of the Zamora system. Defensiveness is defined by the same characteristics, however not inverted.

The cooperative trait is defined as the average of the character's teamspirit, egocentrism, dependability and the compatibility it has to it's ally. Local personality traits are normalized between the values of 0 and 1.

```
this.personality.cooperative = Math.max(Math.min(1,(allyComp +
this.personality.teamSpirit + this.personality.egocentrism +
this.personality.dependable)/4), 0);
```

Emotions in the bot

The emotions are controlled and held locally in a custom emotion handler module. The emotion handler object is the last module to be instantiated, updated and destructed in the modified Petrabot. Any function in the bot has access the emotion handler through the bots headquarter module. Its main function is to hold the emotional values calculated by the system and to control the logic that sends eliciting factors, as well as fetches and processes the emotional states. When an eliciting factor is sent to the emotion handler, it will be held in an array until the emotion handler is updated. If the array contains one or more eliciting factors upon update, it will first send the entirety array to the emotion system of the relevant character. It will next call the engine to fetch the complete emotional state of the character, which then updates all the emotions of the character.

In addition to the emotion, the emotions serve to update the cooperative personality trait. This replaces the updates to the cooperative trait made elsewhere in the bot, with emotion based updates. The cooperative trait is made to both be able to increase and decrease during the battle based on an average of a characters Joy/Distress, Pride/Shame valences and the valence of Admiration/Reproach that the character has for an ally. The actual value of a valence will only be used if they have been updated in the current update, if it hasn't it will be represented in the calculation by 0. Hence the cooperative trait will only increase or decrease if one of the relevant valences have been updated during the current and will only update when updating the emotions towards an ally. The cooperative trait is capped between 1 and 0.

Only three of the four eliciting factor types have been included in the bot. These are the Well-being, Prospect and Agent eliciting factors. Fortunes of Others was excluded because the only expression of the emotional valences (i.e. Happy-For/Resentment and Gloating/Pitty) found was through the bots chat system. Though it would have been simple to design and implement such functionality to possibly great results, it was decided that this would likely skew the test results as it would be too easy to notice; hence trivializing the results of the other emotional valences as a result.

Eliciting factors are instantiated and sent to the emotion handler from a 4 other modules in the bot. These are the headquarter, defense manager, diplomacy manager and the base manager modules. Each one of these modules have the possibility to instantiate and send multiple eliciting factor types, depending on their individual functions.

In general eliciting factors are sent as a result of various in-game events:

- Well-being eliciting factors will be instantiated within most events which elicit an emotional response. A positive elicit will be sent in events of a beneficial nature to the character or its ally, while a negative elicit will be sent for events of a negative nature to the character or its ally. In an explicit sense it will be instantiated by events of diplomacy, allys assisting in battle, the loss or capturing of territory., or attacks.
- Agent eliciting factors come in two variants, either they elicit Pride/Shame or they elicit Admiration/Reproach. Because the emotions elicited by agents can be seen ad personal and social respectively, many events will elicit two agent eliciting factors to modify both the characters view of it's ally or enemy, and its view on its and its allys general progress. In an explicit sense it will be instantiated by events of the same types as the well-being elicits.
- Prospects are unique as they can be instantiated based on the expectation of events, as well as the result of events. Prospects for Fear/Hope will generally be elicited from approaching enemies or threatening structures being built. In an explicit sense finished prospects will be instantiated by events if the threat has passed or if the threat has been confirmed. A finished prospect elicit can only be

sent if the player has more unfinished elicits.

Though emotional eliciting factors generally include static preset values to increase or decrease their values, some eliciting factors will act to modify based on values relevant to the event or action being elicited.

```
let elicit =
{
        "type": "Prospect",
        “desireabilityVal": -1, "desireabilityWei": -0.1,
        "senseofrealityVal": ((((army.ownStrength - army.foeStrength) - (-100)) * (1 - (-1))) / (100 - (-100))) + -1,
        "senseofrealityWei": -0.5,
        "proximityVal": 0, "proximityWei": 0,
        "unexpectednessVal": 0, "unexpectednessWei": 0,
        "prospectDone": false,
        "likelihoodVal": -1 - (enemyDistance * -0.000025),  ”likelihoodWei":-0.1,
        "effortVal": 0,"effortWei": 0,
        "realizationVal": 0, "realizationWei": 0
};
```

As mentioned at the start of this section, the emotional intensities are designed to affect the character's diplomacy, defense and attack strategy. These all vary in utilization to a certain degree, however the method is similar for all.

Diplomacy is affected by the admiration a character has for it's ally. While the PetraBot gives tributes unconditionally and at a flat rate, the modified bot first needs to have admiration for it's ally to give tributes and at an increased rate depending on on high the character's admiration is for another.

```
if (allyPop < Math.min(30, 0.5*gameState.getPopulation()) &&
totalResources[res] > 500 && allyResources[res] < 100)
{
        if(gameState.ai.HQ.emotionHandler.SocialEmoState[i - (1 +(i > PlayerID?
1 : 0))].AdmirationReproach > 0.4)
        {
                tribute[res] = Math.min(100 *
(gameState.ai.HQ.emotionHandler.SocialEmoState[i - (1 +(i > PlayerID? 1 :
0))].AdmirationReproach*5), 0.3*availableResources[res]);
                toSend = true;
        }
}
```

The modified PetraBot has been modified to prioritize attack order differently depending on anger and reproach towards a character. The character will prioritize characters its really angry towards (-0.6 < anger) over the strongest character (by army

strength and wealth) and the character it's angriest at above that. If not sufficiently angry at any given charater it will prioritize the strongest character, beyond that it will either prioritize it's highest reproach (under -0.6 < reproach). Anger and Reproach will also modify the base aggressiveness of the bot over time, to allow for varying patterns of attack based on the agents current emotional state.

Finally the character will prioritize its defenses mainly on the prospect of fear. Two things can and will happen given a high degree of fear towards an enemy. The first of these is that the character will over calculate how large its defensive army should be at a given base. The second is that it will start prioritizing research and building of defensive structures.

Testing

About this chapter

This chapter concerns the evaluation of the system, once implemented in 0 A.D. It will discuss the criteria, the forms of evaluation and the final results of the model.

Testing Procedure

Testing of of the test environment utilized a user testing based approach. The test environment was distributed to multiple users along side test instructions and a set of questionnaires. Multiple users were sent the test remotely through a dropbox link, they were not observed taking it.

Each tester was instructed to play a total of three single player matches, against three different bots; PetraBot, Aristotle and Hippocrates. While PetraBot and Aristotle were in reality the exact same bot; Hippocrates was the one that been modified to include the personality and emotion functionality.

The first match the users faced the standard PetraBot. This was a testing bot meant to make you acquainted with 0 A.D. This was to allow the testers a chance to learn about the game, the game mechanics and the baseline behavior of the bots. This proved highly beneficial as testers would often improve significantly in skill and understand of the game between the first and second match, more so than between the second and third match. A second the training round proved beneficial was that multiple testers reported higher interest and enthusiasm for the game and playing the next couple of matches. This resulted in users who reported that they generally saw themselves as more attentive of the various bots behavior throughout the matches

For the second match the tester were asked to choose between playing against either the Aristotle or Hippocrates bots. After which they would have to answer a series of 10 questions specific to that bot. This was directly followed by the third match in which

they were asked to play against the bot the hadn't faced in the last match. After which they were asked to answer the same 11 questions they had been about the second match for the third. After these sets of questions they were asked a further 6 questions that were generalized about the two final matches.

All matches were standardized by the testers playing by the same predefined civilization, game-type, map and number of players for all matches.

The testers were asked to set all their own, as well as all the bots civilization to the Britons. Though the specific choice of the Britons was in large part trivial, the standardization of it was to avoid varying difficulty caused by randomized civilizations with varying advantages and disadvantages. It allowed players to get acquainted with the specific play style of the civilization, without having to relearn how to play each match. This also had the additional advantage of putting all testers on equal footing when discounting skill in strategy games.

The testers were assigned with one ally and two enemy bots for all matches. This was mainly to allow for testing of diplomacy functionality of the bots and cooperation gained or lost through the actions or lack of action of the tester, however it also had the additional benefit of giving the player assistance against bots of generally higher skill than themselves. Most of the testers stated that they were in favor of this because it allowed them to stay in the game longer and to get a feel for the various bots in two different ways.

The map for the matches was set to the Caspian Sea. This map is distinct because it's not only one of the maps that allow two versus two player matches, but also because both teams start clearly divided by a large lake between them. This gave both team the advantage of being able to build their base and forces before interaction. The added time before initial confrontation also gave the bots ample time to rush in on the player and show their varied attack tendencies and play styles overall.

The matches were set to skirmish mainly because it's a pretty standard game type in Real-time strategy games. The goal of skirmish matches is to purely defeat your enemies, whether it be through force or building a wonder (which is a winning

condition in 0 A.D.). The lack of additional goals and aims for the match means that the tester was allowed to focus completely on the match at hand and as a result on the bots involved.

The testers were allowed to decide their own difficulty setting, with the stipulation that they kept it consistent thru-out all the matches they played. This was to allow for testers of varying skill levels in regards to real-time strategy games to stay in the match for a longer time and as such observe as much o the bots behavior as possible. The consistency rule was put in place to assure that the testing environment parameters did not deviate by a large amount throughout the test.

Questionnaire

As stated above, the tester were asked a series of 28 questions (17 not total counting duplicates). The questions were aimed to gauge whether the tester observed varying behavior in the various bots and whether some of the confirmation of emotions and personality could be attributed to personification of the bots. To this end the questions included a few 'red herring' questions created to see how much the testers observed patterns that weren't there.

The 11 questions asked for the specific bots were answerable only by yes or no, however also allowed for testers to answer not applicable as not confirmed. All questions included a comments section to which each user could add their thoughts and additional details to how they experienced the situation asked about.

The questions for the individual matches were:

1. Did any of the bots in the previous group deviate from expectations you had on it's approach based on previously observed behavior?
2. Did any of the bots seem to change their strategy over time?
3. Did the various bots seem to act differently from one another in terms of play style?
4. Did any of the bots seem to prioritize attacking one of its enemies over the other, possible to an irrational degree?
5. Did any of the bots seem more aggressive than the others?
6. Did any of the bots seem overly defensive?
7. Did your team-mate communicate its intent well?
8. Did your team-mate seem more willing to help you fight enemies over time?
9. Did your team-mate seem more inclined to give you tributes over time?
10. Did your enemies seem to react to actions you made?
11. Did your team win the match?

The final 6 questions were a combination of yes/no questions and questions where the tester had to write an answer.

1. Did think the inclusion of the emotions and personality was noticeable in how one of the bots played?
2. Which set of bots do you think was modded to include personality and emotions? (name the bot in comments)
3. Which set of bots did you was the most difficult to face? (name the bot in comments)
4. Would you call yourself experienced in real-time strategy games?
5. Have you played 0 A.D. before?
6. What difficulty did you play your matches at? (name in comments)

Results

Fourteen people completed the test and sent in their results, however one set of results had two entire incomplete and as such was disregarded in the test. Other testers who have answered all sections, but not answered all questions, have however, been included.

The results for the two match specific sections were measured in the total number of occurrences. Though the test itself only included yes and no questions for the match specific sections, a third option has been added to reflect that some behavior went unobserved, this includes both explicit answers of unobserved behavior and unanswered questions in accepted tests.

Aristotle			
	Total Yes	Total No	Not Observed
Did any of the bots in the previous group deviate from expectations you had on it's approach based on previously observed behavior?	8	4	0
Did any of the bots seem to change their strategy over time?	5	6	1
Did the various bots seem to act differently from one another in terms of play style?	7	4	1
Did any of the bots seem to prioritize attacking one of its enemies over the other, possible to an irrational degree?	6	5	1
Did any of the bots seem more aggressive than the others?	6	4	2
Did any of the bots seem overly defensive?	4	7	1
Did your team-mate communicate its intent well?	9	3	0
Did your team-mate seem more willing to help you fight enemies over time?	7	5	0
Did your team-mate seem more inclined to give you tributes over time?	8	4	0
Did your enemies seem to react to actions you made?	7	4	0
Did your team win the match?	5	6	

Hippocrates			
	Total Yes	Total No	Not Observed
Did any of the bots in the previous group deviate from expectations you had on it's approach based on previously observed behavior?	8	4	0
Did any of the bots seem to change their strategy over time?	9	1	2
Did the various bots seem to act differently from one another in terms of play style?	10	1	1
Did any of the bots seem to prioritize attacking one of its enemies over the other, possible to an irrational degree?	8	2	2
Did any of the bots seem more aggressive than the others?	9	2	1
Did any of the bots seem overly defensive?	6	5	1
Did your team-mate communicate its intent well?	5	7	0
Did your team-mate seem more willing to help you fight enemies over time?	6	4	2
Did your team-mate seem more inclined to give you tributes over time?	4	8	
Did your enemies seem to react to actions you made?	10	1	1
Did your team win the match?	1	11	0

The results of the match specific show some interesting statistical outliers. For one it can be seen clearly that the Hippocrates bot got a higher score on average on the questions concerning emotive and personality based reactions.

For instance 9 out of the 13 testers rated one or more individual Hippocrates bots more aggressive than other Hippocrates bots in its set, while only 6 of the 13 testers rated the Aristotle. Additionally the Hippocrates bot showed a higher average score (though by a small margin in some cases) in variability in strategy over time, variability in play styles between the bots and reactions in response to tester actions.

The bot also rates higher on a tendency to prioritize attacking a player over another.

It must be noted however that the bots share the exact same rate of deviation from the testers expectation. Additionally, the Aristotle rated higher in terms of communication, and sending of tributes. Additionally, the distribution of yes to no on the various questions are very similar; with Hippocrates gaining a sum of 76 to Aristotle 72 yeses.

	Yes / Aristotle	No / Hippocrates	No difference / Both
Did think the inclusion of the emotions and personality was noticeable in how one of the bots played?	10	2	
Which set of bots do you think was modded to include personality and emotions? (name the bot in comments)	7	5	
Which set of bots think did you was the most difficult to face? (name the bot in comments)	4	6	2
Would you call yourself experienced in real-time strategy games?	5	7	
Have you played 0 A.D. before?	0	12	

The generalized test scores show a slightly different story. While nearly all participants listed that the inclusion of personality and emotions were noticeable, a majority of them selected Aristotle as the bot they expected to include the modifications.

One interesting statistic of the test is that the Hippocrates bot had a much higher average of victories than Aristotle, however the testers listed it as nearly the same in terms of difficulty. This could tie into the higher average in aggressiveness perceived in the Hippocrates bot, which could lead to victory without making the bot harder; alternatively this could tie into the higher cooperation testers reported about the Aristotle. The data is however, not conclusive enough that one could argue either point. It might even be something completely different.

In addition to the stats shown in the table, a total of 9 testers opted to stay at the normal difficulty setting (which is standard), while 2 more testers selected to play the game on easy and a last tester played through the matches on Sandbox difficulty (the easiest setting)

Conclusion, Discussion and Further Development

This thesis firstly sought to test whether personality and emotionally based expression can be noticeable to a normal user in comparison to the users intrinsic instinct of personification. It secondly sought to develop a versatile and reusable system that could be usable in a large range of contexts.
On the first account, the results of the test show an either negligible or inconclusive tendency to be noticeable. Though the individual matches seem to show a slight inclination towards noticeable expression, the test as a whole shows a mainly inconsequential difference in the testers reaction to the two bots being tested. Additionally, the higher number of testers who listed the Aristotle bot as the one that had been modded for personality and emotion, showed a tendency towards the notion being false. There might be many reasons for this. One could possibly correlate a tendency to to see the more cooperative bot as the one with more emotional expression, while the other is seen as cold and distant. The more frequent communication could also be a factor. On this question as a whole, the tests have been inconclusive.
On the second account, there might be a partial yet wholly untested potential for the emotion and personality system to act as an addition to a large range of systems. However, implementation into multiple systems has as of yet not been attempted.

Further Development

Though the project is completed, the system can still have a number of improvements. This section will focus on what development makes sense to improve the model from it's current state.

Multiple additions that can be postulated to improve the combination of the models discussed in this paper. These additions were for the included in the project either due to being conceived too late in the project to be added or because their addition would be outside the scope of the project. As a system such as this can have any number of expansions and improvements.
One of note might be the utilization of a social system in conjunction with the

personality and emotion system. This could potentially fit nicely with the social aspects of the models, as well as the compatibility score mechanic.
The emotion system could potentially be improved with a metric of decrease over time if the emotion is not updated to better replicate real emotions that don't stick around forever. Additionally, emotional thresholds might need to adjust over time and in response to a culmination of emotional responses updating the emotional state of the character.

iBates, J. (1994). The Role of Emotion in Believable Agents Joseph Bates, (April).

iiThagard, P. (n.d.). Cognitive Science. Retrieved from http://plato.stanford.edu/archives/fall2008/entries/cognitive-science/

iiiSloman, A. (2001). Beyond Shallow Models of Emotion, *2*(1).

iv R. Pfeifer (1994). *The Fungus Eater Approach" to Emotion*. Retrieved March 9, 2014, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.4405&rep=rep1&type=pdf

v Sloman, A. (2001). *Beyond Shallow Models of Emotion, 2*(1).

vi R. Pfeifer (1994)

vii A.Sloman (1993)

viii Kleinginna and Kleinginna (1981), p. 355

ix Ortony, Clore, & Collins, (1988)

x Carlson, & Hellevang (2010)

xi Conway and Bekerian, 1987

xii Ortony & Turner, (1990)

xiii Herbert A. Simon (1967). *Motivational and emotional controls of cognition.*

xiv Roseman (1979)

xv Roseman (1984)

xvi Roseman et al. (1996)

xviiPicard (1997) p. 209

xviiiHollinger, G. (2007). *Partially Observable Markov Decision Processes* (POMDPs) Outline for POMDP Lecture.

xixSloman, A., & Croucher, M. (1981). *Why robots will have emotions*,

xxFrijda and Moffat (1994) *Frijda's emotion system*

xxiDr. Golf Pfeifer (1988) paper *Artificial Intelligence models of emotion*.

xxiiR. Pfeifer. (1988). *Artificial intelligence models of emotion*. Retrieved May 28, 2014, from http://folk.uib.no/simwc/papers/emotion.pdf

xxiiiOrtony, Clore and Collins (1988), p. 182f

xxivOrtony, A., Clore, G., & Collins, A. (1988). *The cognitive structure of emotions.*

xxvClynes, M. (n.d.). " SENTICS : *The Touch of Emotions*.

xxviPicard, R. W. (n.d.). *Affective Computing,* (321).

xxviiOatley(1992), p. 50

xxviiiCarlson, C., & Hellevang, M. (2010). *Improving user experience in StateCraft*.

xxixSlinde, A. N. (2012). *Modeling Emotions with EEG-data in StateCraft.* Retrieved June 04, 2014, from https://bora.uib.no/bitstream/handle/1956/5972/97132993.pdf?sequence=1

xxxhttp://webspace.ship.edu/cgboer/neurophysio.html (fetched 19.09.2015)

xxxiGélis, *"The Child: from anonymity to individuality",* in Philippe Ariès and Georges Duby, A History of Private Life III: *Passions of the Renaissance* 1989:309.

xxxii Boyle, G. J. (1983). *Effects on academic learning of manipulating emotional states and motivational dynamics.* British Journal of Educational Psychology, 53, 347-357.

xxxiii Epstein, S. & O'Brien, E.J. (1985). *The person-situation debate in historical and current perspective*. Psychological Bulletin

xxxiv Kenrick, D.T. & Funder, D.C. (1988). *Profiting from controversy: Lessons from the person-situation debate*. American Psychologist

xxxv Eysenck, M. W., & Eysenck, H. J. (1980). *Mischel and the concept of personality.* British Journal of Psychology,

xxxvi Atkinson, Rita, L.; Richard C. Atkinson; Edward E. Smith; Daryl J. Bem; Susan Nolen-Hoeksema (2000). *Hilgard's Introduction to Psychology* (13 ed.). Orlando, Florida: Harcourt College Publishers. p. 437.

xxxvii Allport, G.W; Odbert, H. S (1936). *Trait names: A psycholexical study.* - Psychological Monographs 47: 211

xxxviii Cattell, R. B.; Marshall, MB; Georgiades, S (1957). *Personality and motivation: Structure and measurement.* J Journal of Personality Disorders 19 (1): 53–67. doi:10.1521/pedi.19.1.53.62180. PMID 15899720.

xxxix Norman, W. T. (1967). 2800 personality trait descriptors: *Normative operating characteristics for a university population.* Ann Arbor, MI: University of Michigan, Dept. of Psychology.

xl Tupes, E. C., & Christal, R. E. (1961). *Recurrent personality factors based on trait ratings.* USAF ASD Tech. Rep. No. 61-97, Lackland Airforce Base, TX: U. S. Air Force.

xli Norman, W. T. (1963). *Toward an adequate taxonomy of personality attributes: Replicated factor structure in peer nomination personality ratings*. Journal of Abnormal and Social Psychology 66 (6)

xlii Goldberg, L. R. (1981). *Language and individual differences: The search for universals in personality lexicons.* In Wheeler (ed.), Review of Personality and social psychology, vol. 1, 141–165. Beverly Hills, CA: Sage.

xliii Goldberg, L. R. (1980, May). *Some ruminations about the structure of individual differences: Developing a common lexicon for the major characteristics of human personality*. Symposium presentation at the meeting of the Western Psychological Association, Honolulu, HI.

xliv Boyle, G. J., Stankov, L., & Cattell, R. B. (1995). *Measurement and statistical models in the study of personality and intelligence.* In D. H. Saklofske & M. Zeidner (Eds.), International Handbook of Personality and Intelligence (pp.431-433).

xlv Atkinson, Rita, L.; Richard C. Atkinson; Edward E. Smith; Daryl J. Bem; Susan Nolen-Hoeksema (2000). *Hilgard's Introduction to Psychology.* Orlando, Florida: Harcourt College Publishers. p. 437.

xlvi http://sloanreview.mit.edu/article/how-to-become-a-better-leader/ Toegel, G., & Barsoux, J. L. (2012). *How to become a better leader.* MIT Sloan Management Review, 53(3), 51-60

xlvii *Personality Analysis Exercises.* (n.d.). Retrieved March 18, 2014, from http://www.scientificpsychic.com/workbook/chapter8.htm

xlviii Article about the background of the Zamora model retrieved 23.11.2014 from http://www.scientificpsychic.com/workbook/person.html

xlix Article about the individual attributes of the Zamora model retrieved November 20th 2014 from
t htttp://www.scientificpsychic.com/workbook/person1.html

l Jensen, A. J., & Nes, H. (2008). The Personality Module.

li 0 A.D. Philosphical project overview (n.d.) Retrieved August 20th 2015 from http://play0ad.com/game-info/project-overview/

lii Civilization of 0 A.D (n.d.). Retrieved August 20th 2015 from http://0ad.wikia.com/wiki/Civilizations

liii 0 A.D. game technical overview (02.26.2014) Retrieved August 20th 2015 from http://trac.wildfiregames.com/wiki/WfgAcademiaInto

liv SpiderMonkey documentation, Retrieved August 20th 2015 from https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey

lv Forum post by 'Duplicarius' announcing the Petrabot (03.2014) retrieved August 20th 2015 from http://wildfiregames.com/forum/index.php?showtopic=18425

lvi Generall structure of the Petrabot (01 29, 2015). Retrieved August 20th 2015 from http://trac.wildfiregames.com/wiki/PetraBot

lvii History of C++ - C++ Information. (n.d.). Retrieved May 27, 2014, from http://www.cplusplus.com/info/history/

lviii Introduction to Object Oriented Programming Concepts (OOP) and More - CodeProject. (n.d.). Retrieved May 28, 2014, from http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep

lix Pae, A. (2005, August 10). Static Libraries Versus Dynamic Libraries. alan pae. Retrieved from http://www.ilkda.com/compile/Static_Versus_Dynamic.htm

lx Behavior trees for AI: How they work, C. Simpson Retrieved January 10th 2015 http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php#

lxi Giovanna Colombetti, Appraising Valence http://people.exeter.ac.uk/gc243/index/GC_AppraisingValence05.pdf

lxii McAdams, D. (1995). What Do We Know When We Know a Person? Journal of Personality, 63(3), 365-396. doi:10.1111/j.1467-6494.1995.tb00500.

Appendix A – Personality and Emotion System Code

Appendix A 1.1 – Character Handler Header

```
#pragma once
#include <unordered_map>
#include "Character.h"
#include "PersonalityTemplate.h"
#include "PersonalityAttributes.h"
#include "CharacterDefinition.h"

class CharacterHandler
{
public:
	CharacterHandler::CharacterHandler();
	CharacterHandler::~CharacterHandler();

private:
	std::unordered_map<int, Character> chars;

public:
	bool CharacterHandler::NumberExists(int i);
	bool CharacterHandler::EndCharacter(int i);
	bool CharacterHandler::StartCharacter(int i, CharacterDefinition chr);
	Character* CharacterHandler::GetCharacter(int i);

	void Update();

}
```

Appendix A 1.2 – Character Handler Source

```
#include "stdafx.h"
#include "PersonalityTemplate.h"
#include "EmotionHandler.h"
#include "CharacterHandler.h"

bool CharacterHandler::NumberExists(int i)
{
	if (chars.count(i) > 0)
	{
		return true;
	}
	return false;
}

void CharacterHandler::Update()
{
	for (auto kv : chars)
	{
		kv.second.Update();
	}
}

bool CharacterHandler::EndCharacter(int i)
{
	if (NumberExists(i))
	{
		chars.erase(i);
		for (auto kv : chars)
		{
			kv.second.remOpinion(i);
		}
		return true;
	}
	return false;
}

bool CharacterHandler::StartCharacter(int i, CharacterDefinition chrdef)
{
	if (!NumberExists(i))
	{
		Character c = Character(i, EmotionHandler(i, chrdef.e), PersonalityTemplate(chrdef.p));
		for (auto kv : chars)
		{
			c.instOpinion(&kv.second, kv.first);
			kv.second.instOpinion(&c, i);
		}
		chars.emplace(i, c);
		return true;
	}
	return false;
}

Character* CharacterHandler::GetCharacter(int i) { return &chars.at(i); }

CharacterHandler::CharacterHandler(){}

CharacterHandler::~CharacterHandler(){}
```

Appendix A 1.3 – Character Header

```
#pragma once
#include <vector>
#include <unordered_map>
#include "Emote.h"
#include "EmotionHandler.h"
#include "PersonalityTemplate.h"

class Character
{
	friend class CharacterHandler;
private:
	EmotionHandler eH;
	PersonalityTemplate pT;

	void instOpinion(Character* i, int o);
	void remOpinion(int i);
	void Update();
	double augmentEmotion(ValencedEmotions e, double frEm);

public:
	Character(int i, EmotionHandler emotion, PersonalityTemplate personality);

	~Character();

	EmotionHandler* Emotion();

	PersonalityTemplate* Personality();

};
```

Appendix A 1.4 – Character Source

```
#include "stdafx.h"
#include "Character.h"

Character::Character(int i, EmotionHandler emotion, PersonalityTemplate personality) : eH(emotion), pT(personality){}

void Character::instOpinion(Character* i, int o)
{
	Emotion()->InitSocEmotions(o);
	Personality()->CompatiblityScore(o, i->Personality());
}

void Character::Update()
{
	eH.Update();
}

void Character::remOpinion(int i)
{
	Emotion()->remOpinion(i);
	Personality()->RemCompatibility(i);
}

Character::~Character(){}

EmotionHandler* Character::Emotion(){ return &eH; }

PersonalityTemplate* Character::Personality(){ return &pT; }
```

Appendix A 2.1 – Personality Template Header

```
#pragma once
#include <unordered_map>
#include <cmath>
#include "AddMath.h"
#include "IPersonalityAttributes.h"
#include "PersonalityAttributes.h"

class PersonalityTemplate
{
	friend class Character;
private:
	PersonalAttributes pA;
	SocialAttributes sA;

	void PersonalityTemplate::RemCompatibility(int i);
	void PersonalityTemplate::CompatiblityScore(int j, PersonalityTemplate* oT);

	//compatibility Scores
	std::unordered_map<int, double> compScore;

public:
	PersonalityTemplate(IPersonalityAttributes i);
	~PersonalityTemplate();

	double PersonalityTemplate::GetCompatibility(int i);

	PersonalAttributes* PersonalityTemplate::Personal();
	SocialAttributes* PersonalityTemplate::Social();
};
```

Appendix A 2.2 – Personality Template Source

```
#include "stdafx.h"
#include "PersonalityTemplate.h"

PersonalityTemplate::PersonalityTemplate(IPersonalityAttributes i)
	: pA(i.achievement_attributes, i.emotional_temperament, i.intellectual_factors, i.philosophical_attitudes,
i.risk_factors, i.task_performance),
	sA(i.aggressiveness,i.control_attitudes, i.dependability, i.egocentrism, i.emotional_expression, i.fairness,
i.leadership, i.regard_for_rules, i.team_spirit){}

void PersonalityTemplate::CompatiblityScore(int j, PersonalityTemplate* oT)
{
	double pers = 0;
	for (int i = 0; i < TotalPersonal; i++)
	{
		double p = Personal()->GetByNumber(i);
		double o = oT->Personal()->GetByNumber(i);
		double op = 1 - abs(abs(p) - abs(o));
		pers += (p * o > 0 ? op : -1 * op);
	}
	pers /= TotalPersonal;

	double soc = 0;
	for (int i = 0; i < TotalSocial; i++)
	{
		double o = oT->Personal()->GetByNumber(i);
		if (o < 0){		soc += o; }
	}
	double compS = (pers + (soc / TotalSocial) / 2);
	compScore.emplace(j, compS);
}

double PersonalityTemplate::GetCompatibility(int i){ return compScore.at(i); }

void PersonalityTemplate::RemCompatibility(int i){ compScore.erase(i); }

PersonalAttributes* PersonalityTemplate::Personal() { return &pA; }

SocialAttributes* PersonalityTemplate::Social() { return &sA; }

PersonalityTemplate::~PersonalityTemplate(){}


#pragma once
enum PersAtt{ achievement_attributes, emotional_temprament, intellectual_factors, philosophical_attitudes, risk_factors,
task_performance, TotalPersonal, StartPersonal = achievement_attributes};
class PersonalAttributes
{
	friend class PersonalityTemplate;

private:
	double p[TotalPersonal];
	double GetByNumber(int i) const;

public:
	PersonalAttributes(double achievement_attributes, double emotional_temperament, double intellectual_factors,
double philosophical_attitudes, double risk_factors, double task_performance);
	~PersonalAttributes();

	double AchievementAttributes() const;
	double EmotionalTemperament() const;
	double IntellectualFactors() const;
	double PhilosophicalAttitudes() const;
	double RiskFactors() const;
	double TaskPerformance() const;

};
```

Appendix A 2.3 – Personality Attributes Header

```
#pragma once
enum PersAtt{ achievement_attributes, emotional_temprament, intellectual_factors, philosophical_attitudes, risk_factors,
task_performance, TotalPersonal, StartPersonal = achievement_attributes};
class PersonalAttributes
{
	friend class PersonalityTemplate;

private:
	double p[TotalPersonal];
	double GetByNumber(int i) const;

public:
	PersonalAttributes(double achievement_attributes, double emotional_temperament, double intellectual_factors,
double philosophical_attitudes, double risk_factors, double task_performance);
	~PersonalAttributes();

	double AchievementAttributes() const;
	double EmotionalTemperament() const;
	double IntellectualFactors() const;
	double PhilosophicalAttitudes() const;
	double RiskFactors() const;
	double TaskPerformance() const;

};

enum SocAtt{ aggressiveness, control_attitudes, dependability, egocentrism, emotional_expression, fairness, leadership,
regard_for_rules, team_spirit, TotalSocial, StartSocial = aggressiveness };
class SocialAttributes
{
	friend class PersonalityTemplate;

private:
	double s[TotalSocial];
	double GetByNumber(int i) const;

public:
	SocialAttributes(double aggressiveness, double control_attitudes, double dependability, double egocentrism,
double emotional_expression, double fairness, double leadership, double regard_for_rules, double team_spirit);
	~SocialAttributes();

	double Aggressiveness() const;
	double ControlAttitudes() const;
	double Dependability() const;
	double Egocentrism() const;
	double EmotionalExpression() const;
	double Fairness() const;
	double Leadership() const;
	double RegardForRules() const;
	double TeamSpirit() const;

};
```

Appendix A 2.4 – Personality Attributes Source

```
#include "stdafx.h"
#include <array>
#include "AddMath.h"
#include "PersonalityAttributes.h"

double clamp(double n)
{
	return AddMath::clamp(n, -1.0, 1.0);
}

PersonalAttributes::PersonalAttributes(double achievement_attributes, double emotional_temperament, double
intellectual_factors, double philosophical_attitudes, double risk_factors, double task_performance)
{
	p[0] = clamp(achievement_attributes);
	p[1] = clamp(emotional_temperament);
	p[2] = clamp(intellectual_factors);
	p[3] = clamp(philosophical_attitudes);
	p[4] = clamp(risk_factors);
	p[6] = clamp(task_performance);
}

PersonalAttributes::~PersonalAttributes(){}

double PersonalAttributes::AchievementAttributes() const { return p[achievement_attributes]; }
double PersonalAttributes::EmotionalTemperament() const { return p[emotional_temprament]; }
double PersonalAttributes::IntellectualFactors() const { return p[intellectual_factors]; }
double PersonalAttributes::PhilosophicalAttitudes() const { return p[philosophical_attitudes]; }
double PersonalAttributes::RiskFactors() const { return p[risk_factors]; }
double PersonalAttributes::TaskPerformance() const { return p[task_performance]; }
double PersonalAttributes::GetByNumber(int i) const { return p[i]; };


SocialAttributes::SocialAttributes(double aggressiveness, double control_attitudes, double dependability, double
egocentrism, double emotional_expression, double fairness, double leadership, double regard_for_rules, double
team_spirit)
{
	s[0] = clamp(aggressiveness);
	s[1] = clamp(control_attitudes);
	s[2] = clamp(dependability);
	s[3] = clamp(egocentrism);
	s[4] = clamp(emotional_expression);
	s[5] = clamp(fairness);
	s[6] = clamp(leadership);
	s[7] = clamp(regard_for_rules);
	s[8] = clamp(team_spirit);
}
SocialAttributes::~SocialAttributes(){}

double SocialAttributes::Aggressiveness() const { return s[aggressiveness]; }
double SocialAttributes::ControlAttitudes() const { return s[control_attitudes]; }
double SocialAttributes::Dependability() const { return s[dependability]; }
double SocialAttributes::Egocentrism() const { return s[egocentrism]; }
double SocialAttributes::EmotionalExpression() const { return s[emotional_expression]; }
double SocialAttributes::Fairness() const { return s[fairness]; }
double SocialAttributes::Leadership() const { return s[leadership]; }
double SocialAttributes::RegardForRules() const { return s[regard_for_rules]; }
double SocialAttributes::TeamSpirit() const { return s[team_spirit]; }
double SocialAttributes::GetByNumber(int i) const { return s[i]; }
```

Appendix A 3.1 – Emotion Handler Header

```
#pragma once
#include <deque>
#include <vector>
#include <iostream>
#include <unordered_map>
#include "Emote.h"
#include "EmotionController.h"
#include "ElicitingFactor.h"
#include "EmoteTree.h"
#include "EmotionalThresholdDefinition.h"

class EmotionHandler
{
private:
	int thisc;
	std::unordered_map<int, std::vector<Em>> emControl;

	EmoteTree emoteTree = EmoteTree(emControl, thisc);
	EmoteTree::Selector selectors[3];
	EmoteTree::Foo foo = EmoteTree::Foo(emoteTree);
	EmoteTree::WellBeing wellBeing = EmoteTree::WellBeing(emoteTree);
	EmoteTree::Attribution attri = EmoteTree::Attribution(emoteTree);
	EmoteTree::CompoundEmotions compSelf = EmoteTree::CompoundEmotions(emoteTree);
	EmoteTree::ProspectEmotions proEmo = EmoteTree::ProspectEmotions(emoteTree);

	std::deque<ElicitingFactor*> eFQ;

	void EmotionHandler::BuildEmoteTree();

	void EmotionHandler::RunTree();

	//Used for setup of the emotional unit
	void EmotionHandler::SetFOOThresh(double happyForThreshold, double resentMentThreshold, double
gloatingThreshold, double pityThreshold);
	void EmotionHandler::SetWBThresh(double joyThreshold, double distressThreshold);
	void EmotionHandler::SetProspectThresh(double hopeThreshold, double fearThreshold, double
satisfactionThreshold, double fearsconfirmedThreshold, double reliefThreshold, double disappointmentThreshold);
	void EmotionHandler::SetAttributThresh(double prideThreshold, double shameThreshold, double
admirationThreshold, double reproachThreshold);
	void EmotionHandler::SetCompoundThresh(double gratificationThreshold, double remorseThreshold, double
gratitudeThreshold, double angerThreshold);

public:
	EmotionHandler(int i, EmotionThresholdsDef eT);
	~EmotionHandler();

	//used to instantiate emotions directed at i
	void EmotionHandler::InitSocEmotions(int i);

	void EmotionHandler::remOpinion(int i);

	//Used for dll emotion elicitation
	void EmotionHandler::AddElicitingFactor(ElicitingFactor& ef);

	void EmotionHandler::Update();

	//Mainly used outside dll scope
	Emote EmotionHandler::GetStrongestEmotion();

};
```

Appendix A 3.1 – **Emotion Handler** Header

```
#include "stdafx.h"
#include "EmotionHandler.h"

void EmotionHandler::AddElicitingFactor(ElicitingFactor& ef){ eFQ.push_back(&ef); }

Emote EmotionHandler::GetStrongestEmotion()
{
	Emote e = { };
	for (auto kval : emControl)
	{
		for (auto kv : kval.second)
		{
			if (kv.intensity > e.intensity)
			{
				*e.directedAt = kval.first;
				e.elicitID = kv.elicitID;
				*e.name = kv.name;
				e.intensity = kv.intensity;
			}
		}
	}
	return e;
}

void EmotionHandler::Update()
{
	RunTree();
}

void EmotionHandler::RunTree()
{
	if (!eFQ.empty())
	{
		emoteTree.SetElicitingFactor(eFQ.front());
		eFQ.pop_front();
	}

	if (emoteTree.run())
	{
		std::cout << "Emotion modified." << std::endl;
	}
	else
	{
		std::cout << "No emotion modified." << std::endl;

	}
}

void EmotionHandler::remOpinion(int i)
{
	emControl.erase(i);
}

void EmotionHandler::InitSocEmotions(int i)
{
	std::vector<Em> emotions;
	emotions.reserve(TotalSocialEmotions);
	for (int j = 0; j < TotalSocialEmotions; j++)
	{
		Em emotion;
		emotion.name = static_cast<ValencedEmotions>(j);
		emotion.intensity = new (double);
		emotion.emotePot.negPotential = new (double);
		emotion.emotePot.posPotential = new (double);

		emotion.intensity = 0;
		emotion.emotePot.negPotential = 0;
		emotion.emotePot.posPotential = 0;
		emotions.push_back(emotion);
```

```
		}
		emControl.emplace(i, emotions);
}

void EmotionHandler::BuildEmoteTree()
{
		emoteTree.setRootChild(&selectors[0]);
		selectors[0].addChildren({ &selectors[1], &compSelf });
		selectors[1].addChildren({ &selectors[2], &attri });
		selectors[2].addChildren({ &foo, &selectors[3] });
		selectors[3].addChildren({ &proEmo, &wellBeing });
}

EmotionHandler::EmotionHandler(int i, EmotionThresholdsDef eT) : thisc(i)
{
		BuildEmoteTree();

		SetFOOThresh(eT.happyForThreshold, eT.resentmentThreshold, eT.gloatingThreshold, eT.pityThreshold);
		SetWBThresh(eT.joyThreshold, eT.distressThreshold);
		SetProspectThresh(eT.hopeThreshold, eT.fearThreshold, eT.satisfactionThreshold, eT.fearsconfirmedThreshold,
eT.reliefThreshold, eT.disappointmentThreshold);
		SetAttributThresh(eT.prideThreshold, eT.shameThreshold, eT.admirationThreshold, eT.reproachThreshold);
		SetCompoundThresh(eT.gratificationThreshold, eT.remorseThreshold, eT.gratitudeThreshold,
eT.angerThreshold);

		std::vector<Em> emotions;
		emotions.reserve(TotalPersonalEmotions);
		for (int j = 0; j < TotalPersonalEmotions; j++)
		{
			Em emotion;
			emotion.name = static_cast<ValencedEmotions>(j + TotalSocialEmotions);
			emotion.intensity = new (double);
			emotion.emotePot.negPotential = new (double);
			emotion.emotePot.posPotential = new (double);

			emotion.intensity = 0;
			emotion.emotePot.negPotential = 0;
			emotion.emotePot.posPotential = 0;
			emotions.push_back(emotion);
		}
		emControl.emplace(i, emotions);
}

void EmotionHandler::SetFOOThresh(double happyForThreshold, double resentMentThreshold, double
gloatingThreshold, double pityThreshold)
{
		emoteTree.SetFOOThresh(happyForThreshold, resentMentThreshold, gloatingThreshold, pityThreshold);
}
void EmotionHandler::SetWBThresh(double joyThreshold, double distressThreshold)
{
		emoteTree.SetWBThresh(joyThreshold, distressThreshold);
}
void EmotionHandler::SetProspectThresh(double hopeThreshold, double fearThreshold, double satisfactionThreshold,
double fearsconfirmedThreshold, double reliefThreshold, double disappointmentThreshold)
{
		emoteTree.SetProspectThresh(hopeThreshold, fearThreshold, satisfactionThreshold, fearsconfirmedThreshold,
reliefThreshold, disappointmentThreshold);
}
void EmotionHandler::SetAttributThresh(double prideThreshold, double shameThreshold, double admirationThreshold,
double reproachThreshold)
{
		emoteTree.SetAttributThresh(prideThreshold, shameThreshold, admirationThreshold, reproachThreshold);
}
void EmotionHandler::SetCompoundThresh(double gratificationThreshold, double remorseThreshold, double
gratitudeThreshold, double angerThreshold)
{
		emoteTree.SetCompoundThresh(gratificationThreshold, remorseThreshold, gratitudeThreshold,
angerThreshold);
}
```

```
EmotionHandler::~EmotionHandler()
{
	if (!eFQ.empty()){ eFQ.clear(); }
}
```

Appendix A 3.2 – **Emotion Handler** Header

```
#include "stdafx.h"
#include "EmotionHandler.h"

void EmotionHandler::AddElicitingFactor(ElicitingFactor& ef){ eFQ.push_back(&ef); }

Emote EmotionHandler::GetStrongestEmotion()
{
	Emote e = { };
	for (auto kval : emControl)
	{
		for (auto kv : kval.second)
		{
			if (kv.intensity > e.intensity)
			{
				*e.directedAt = kval.first;
				e.elicitID = kv.elicitID;
				*e.name = kv.name;
				e.intensity = kv.intensity;
			}
		}
	}
	return e;
}

void EmotionHandler::Update()
{
	RunTree();
}

void EmotionHandler::RunTree()
{
	if (!eFQ.empty())
	{
		emoteTree.SetElicitingFactor(eFQ.front());
		eFQ.pop_front();
	}

	if (emoteTree.run())
	{
		std::cout << "Emotion modified." << std::endl;
	}
	else
	{
		std::cout << "No emotion modified." << std::endl;

	}
}

void EmotionHandler::remOpinion(int i)
{
	emControl.erase(i);
}

void EmotionHandler::InitSocEmotions(int i)
{
	std::vector<Em> emotions;
	emotions.reserve(TotalSocialEmotions);
	for (int j = 0; j < TotalSocialEmotions; j++)
	{
		Em emotion;
		emotion.name = static_cast<ValencedEmotions>(j);
		emotion.intensity = new (double);
		emotion.emotePot.negPotential = new (double);
		emotion.emotePot.posPotential = new (double);

		emotion.intensity = 0;
		emotion.emotePot.negPotential = 0;
		emotion.emotePot.posPotential = 0;
		emotions.push_back(emotion);
```

```
		}
		emControl.emplace(i, emotions);
}

void EmotionHandler::BuildEmoteTree()
{
		emoteTree.setRootChild(&selectors[0]);
		selectors[0].addChildren({ &selectors[1], &compSelf });
		selectors[1].addChildren({ &selectors[2], &attri });
		selectors[2].addChildren({ &foo, &selectors[3] });
		selectors[3].addChildren({ &proEmo, &wellBeing });
}

EmotionHandler::EmotionHandler(int i, EmotionThresholdsDef eT) : thisc(i)
{
		BuildEmoteTree();

		SetFOOThresh(eT.happyForThreshold, eT.resentmentThreshold, eT.gloatingThreshold, eT.pityThreshold);
		SetWBThresh(eT.joyThreshold, eT.distressThreshold);
		SetProspectThresh(eT.hopeThreshold, eT.fearThreshold, eT.satisfactionThreshold, eT.fearsconfirmedThreshold,
eT.reliefThreshold, eT.disappointmentThreshold);
		SetAttributThresh(eT.prideThreshold, eT.shameThreshold, eT.admirationThreshold, eT.reproachThreshold);
		SetCompoundThresh(eT.gratificationThreshold, eT.remorseThreshold, eT.gratitudeThreshold,
eT.angerThreshold);

		std::vector<Em> emotions;
		emotions.reserve(TotalPersonalEmotions);
		for (int j = 0; j < TotalPersonalEmotions; j++)
		{
			Em emotion;
			emotion.name = static_cast<ValencedEmotions>(j + TotalSocialEmotions);
			emotion.intensity = new (double);
			emotion.emotePot.negPotential = new (double);
			emotion.emotePot.posPotential = new (double);

			emotion.intensity = 0;
			emotion.emotePot.negPotential = 0;
			emotion.emotePot.posPotential = 0;
			emotions.push_back(emotion);
		}
		emControl.emplace(i, emotions);
}

void EmotionHandler::SetFOOThresh(double happyForThreshold, double resentMentThreshold, double
gloatingThreshold, double pityThreshold)
{
		emoteTree.SetFOOThresh(happyForThreshold, resentMentThreshold, gloatingThreshold, pityThreshold);
}
void EmotionHandler::SetWBThresh(double joyThreshold, double distressThreshold)
{
		emoteTree.SetWBThresh(joyThreshold, distressThreshold);
}
void EmotionHandler::SetProspectThresh(double hopeThreshold, double fearThreshold, double satisfactionThreshold,
double fearsconfirmedThreshold, double reliefThreshold, double disappointmentThreshold)
{
		emoteTree.SetProspectThresh(hopeThreshold, fearThreshold, satisfactionThreshold, fearsconfirmedThreshold,
reliefThreshold, disappointmentThreshold);
}
void EmotionHandler::SetAttributThresh(double prideThreshold, double shameThreshold, double admirationThreshold,
double reproachThreshold)
{
		emoteTree.SetAttributThresh(prideThreshold, shameThreshold, admirationThreshold, reproachThreshold);
}
void EmotionHandler::SetCompoundThresh(double gratificationThreshold, double remorseThreshold, double
gratitudeThreshold, double angerThreshold)
{
		emoteTree.SetCompoundThresh(gratificationThreshold, remorseThreshold, gratitudeThreshold,
angerThreshold);
}
```

```
EmotionHandler::~EmotionHandler()
{
	if (!eFQ.empty()){ eFQ.clear(); }
}
```

Appendix A 3.3 – **Emotion Tree** Class

```
#include <vector>
#include <unordered_map>
#include "Emote.h"
#include "EmotionController.h"
#include "AddMath.h"
#include "BehaviourTree.h"

class EmoteTree : public BehaviourTree
{
protected:
	int thisID;
	std::unordered_map<int, std::vector<Em>> & emotes;
	ElicitingFactor* eF;
	ElicitingFactor* last;
	EmotionalThresholds emoteThresh[(TotalPersonalEmotions + TotalSocialEmotions)];

public:
	//General Modifiers
	class EmoTreeBase : public BehaviourTree::Node
	{
	public:
		EmoTreeBase(const EmoteTree& aObj) : aRef(aObj) {}

	protected:
		const EmoteTree& aRef;

		double lg()
		{
			return resCurve(aRef.eF->SenseOfReality().first, aRef.eF->SenseOfReality().second)
				+ resCurve(aRef.eF->Proximity().first, aRef.eF->Proximity().second)
				+ resCurve(aRef.eF->Unexpectedness().first, aRef.eF->Unexpectedness().second);
		}

		double clamp(double n)
		{
			return AddMath::clamp(n, -1.0, 1.0);
		}

		double resCurve(double x, double c)
		{
			x = clamp(x);
			double invc = 1 - abs(clamp(c));
			return AddMath::LogCurve(10, x < 0 ? -1 : 1, invc, 0, abs(x));
		}

		void modPotential(int per, int i, double p)
		{
			double pos = *aRef.emotes.at(per)[i].emotePot.posPotential + p;
			*aRef.emotes.at(per)[i].emotePot.posPotential = AddMath::clamp(pos, 0.0, 1.0);
			double neg = *aRef.emotes.at(per)[i].emotePot.negPotential + p;
			*aRef.emotes.at(per)[i].emotePot.negPotential = AddMath::clamp(neg, -1.0, 0.0);
		}

		void intensityCalc(int per, int i, int j)
		{
			if (*aRef.emotes.at(per)[i].emotePot.posPotential > *aRef.emoteThresh[j].posThreshold
				&& abs(*aRef.emotes.at(per)[i].emotePot.negPotential) < *aRef.emotes.at(per)
[i].emotePot.posPotential)
			{
				*aRef.emotes.at(per)[i].intensity = clamp(resCurve(*aRef.emotes.at(per)
[i].emotePot.posPotential, *aRef.emoteThresh[j].posThreshold));
			}
			else if (*aRef.emotes.at(per)[i].emotePot.negPotential < *aRef.emoteThresh[j].negThreshold
				&& abs(*aRef.emotes.at(per)[i].emotePot.negPotential) > *aRef.emotes.at(per)
[i].emotePot.posPotential)
			{
				*aRef.emotes.at(per)[i].intensity = clamp(resCurve(*aRef.emotes.at(per)
[i].emotePot.negPotential, *aRef.emoteThresh[j].negThreshold));
			}
```

```
                        else
                        {
                                *aRef.emotes.at(per)[i].intensity = 0;
                        }
                        aRef.emotes.at(per)[i].elicitID = aRef.eF;
                        *aRef.last = *aRef.eF;
                }

        };

        //LEAF fortunes of others
        class Foo : public EmoTreeBase
        {
        public:
                Foo(const EmoteTree& aObj) : EmoTreeBase(aObj) {}

                bool run() override
                {
                        if (aRef.eF->getType() == eFortunes && aRef.eF != aRef.last)
                        {
                                ElicitFortunesOfOther* eF = static_cast<ElicitFortunesOfOther*>(aRef.eF);
                                double D = resCurve(eF->Desirability().first, eF->Desirability().second);
                                if (D != 0)
                                {
                                        double Dfo = resCurve(eF->DesiFO().first, eF->DesiFO().second);
                                        double Des = resCurve(eF->Deserve().first, eF->Deserve().second);
                                        double Like = resCurve(eF->Liking().first, eF->Liking().second);

                                        if (Dfo > 0){ modPotential(eF->CharID(), HR, (D + Dfo + Des + Like + Ig()) /
7); intensityCalc(eF->CharID(),HR, HappyforResentment); }
                                        else if (Dfo < 0){ modPotential(eF->CharID(), GP, (D + Dfo + Des + (-1 *
Like) + Ig()) / 7); intensityCalc(eF->CharID(), GP, GloatingPity); }
                                        return true;
                                }
                        }
                        return false;
                }
        };

        //LEAF fear based emotions
        class ProspectEmotions : public EmoTreeBase
        {
        public:
                ProspectEmotions(const EmoteTree& aObj) : EmoTreeBase(aObj) {}

                bool run() override
                {
                        if (aRef.eF->getType() == eProspect && aRef.eF != aRef.last)
                        {
                                ElicitProspect* eP = static_cast<ElicitProspect*>(aRef.eF);
                                double D = resCurve(eP->Desirability().first, eP->Desirability().second);
                                if (D != 0)
                                {
                                        double lik = resCurve(eP->Likelihood().first, eP->Likelihood().second);
                                        modPotential(aRef.thisID, HF, (D + lik + Ig()) / 5);

                                        //Intensity modification
                                        intensityCalc(aRef.thisID, HF, HopeFear);

                                        return true;
                                }
                        }
                        else if (aRef.eF->getType() == eProsDone && aRef.eF != aRef.last)
                        {
                                ElicitProspect* eP = static_cast<ElicitProspect*>(aRef.eF);
                                if (*aRef.emotes.at(aRef.thisID)[HF].emotePot.posPotential > 0 ||
*aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential < 0)
                                {
                                        double effort = resCurve(eP->Effort().first, eP->Effort().second);
                                        double real = resCurve(eP->Realization().first, eP->Realization().second);
```

```
                                        double hfPot = (*aRef.emotes.at(aRef.thisID)[HF].emotePot.posPotential >
abs(*aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential)
                                                ? *aRef.emotes.at(aRef.thisID)[HF].emotePot.posPotential :
*aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential);

                                        if (real > 0)
                                        {
                                                modPotential(aRef.thisID, SFc, (hfPot + effort + real + Ig()) / 6);
                                                intensityCalc(aRef.thisID, SFc, SatisfactionFearsconfirmed);

                                                if (*aRef.emoteThresh[SatisfactionFearsconfirmed].posThreshold <
*aRef.emotes.at(aRef.thisID)[SFc].emotePot.posPotential
                                                        ||
*aRef.emoteThresh[SatisfactionFearsconfirmed].negThreshold < *aRef.emotes.at(aRef.thisID)
[SFc].emotePot.posPotential)
                                                {
                                                        double D = resCurve(eP->Desirability().first, eP-
>Desirability().second);
                                                        double lik = resCurve(eP->Likelihood().first, eP-
>Likelihood().second);
                                                        (*aRef.emotes.at(aRef.thisID)[HF].emotePot.posPotential
> *aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential
                                                                ? *aRef.emotes.at(aRef.thisID)
[HF].emotePot.posPotential : *aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential)
                                                                = ((D + lik + Ig()) / 5);
                                                }
                                        }
                                        else if (real < 0)
                                        {
                                                modPotential(aRef.thisID, RD, (hfPot + effort + real + Ig()) / 6);
                                                intensityCalc(aRef.thisID, RD, ReliefDisappointment);

                                                if (*aRef.emoteThresh[ReliefDisappointment].posThreshold <
*aRef.emotes.at(aRef.thisID)[RD].emotePot.posPotential
                                                        || *aRef.emoteThresh[ReliefDisappointment].negThreshold
< *aRef.emotes.at(aRef.thisID)[RD].emotePot.posPotential)
                                                {
                                                        double D = resCurve(eP->Desirability().first, eP-
>Desirability().second);
                                                        double lik = resCurve(eP->Likelihood().first, eP-
>Likelihood().second);
                                                        (*aRef.emotes.at(aRef.thisID)[HF].emotePot.posPotential
> *aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential
                                                                ? *aRef.emotes.at(aRef.thisID)
[HF].emotePot.posPotential : *aRef.emotes.at(aRef.thisID)[HF].emotePot.negPotential)
                                                                = ((D + lik + Ig()) / 5);
                                                }
                                        }

                                        return true;
                                }
                        }
                        return false;
                }
        };

        //LEAF Well-Being
        class WellBeing : public EmoTreeBase
        {
        public:
                WellBeing(const EmoteTree& aObj) : EmoTreeBase(aObj) {}

                bool run() override
                {
                        if (aRef.eF->getType() == eWellBeing && aRef.eF != aRef.last)
                        {
                                double D = resCurve(aRef.eF->Desirability().first, aRef.eF->Desirability().second);
                                if (D != 0)
                                {
                                        //joy & distress potential calc
```

```
                                        modPotential(aRef.thisID, JD, (D + Ig()) / 4);

                                        //Intensity modification
                                        intensityCalc(aRef.thisID,JD, JoyDistress);

                                        return true;
                                }
                        }
                        return false;
                }
        };

        //LEAF attribution towards self
        class Attribution : public EmoTreeBase
        {
        public:
                Attribution(const EmoteTree& aObj) : EmoTreeBase(aObj) {}

                bool run() override
                {
                        if (aRef.eF->getType() == eAgent && aRef.eF != aRef.last)
                        {
                                ElicitAgent* eA = static_cast<ElicitAgent*>(aRef.eF);
                                double D = resCurve(eA->Desirability().first, eA->Desirability().second);
                                if (D != 0)
                                {
                                        double Exd = resCurve(eA->ExpDev().first, eA->ExpDev().second);
                                        double Scu = resCurve(eA->StrCogU().first, eA->StrCogU().second);

                                        if (Scu > 0 || eA->CharID() == aRef.thisID){ modPotential(aRef.thisID, PS, (D
+ Exd + Scu + Ig()) / 6); intensityCalc(aRef.thisID, PS, PrideShame); }
                                        else{ modPotential(eA->CharID(), AR, (D + Exd + Ig()) / 5); intensityCalc(eA-
>CharID(), AR, AdmirationReproach); }

                                        return true;
                                }
                        }
                        return false;
                }
        };

        //LEAF Well-Being, self attribution compound
        class CompoundEmotions : public EmoTreeBase
        {
        public:
                CompoundEmotions(const EmoteTree& aObj) : EmoTreeBase(aObj) {}

                bool run() override
                {
                        bool found = false;
                        for (auto kv : aRef.emotes)
                        {
                                bool self = kv.first == aRef.thisID;
                                double Ij = *aRef.emotes.at(kv.first)[JD].intensity;
                                double Ip = *aRef.emotes.at(kv.first)[self ? PS : AR].intensity;
                                if ((Ij * Ip) > 0)
                                {
                                        int i = (Ij > 0 ? 1 : -1);
                                        modPotential(kv.first, self ? GR : GA, (i * sqrt(abs(Ij * Ip))));
                                        intensityCalc(kv.first, self ? GR : GA, self ? GratificationRemorse :
GratitudeAnger);
                                        found = true;
                                }
                        }
                        return found;
                }
        };

        EmoteTree(std::unordered_map<int, std::vector<Em>>& em, int thisID) : emotes(em), thisID(thisID)
        {
```

```
		last = eF;
		for (int i = 0; i < TotalValencedEmotions; i++)
		{
			emoteThresh[i].negThreshold = new (double);
			emoteThresh[i].posThreshold = new (double);
			emoteThresh[i].negThreshold = 0;
			emoteThresh[i].posThreshold = 0;
		}
	}

	void SetElicitingFactor(const ElicitingFactor* elF){ *eF = *elF; }

	//Threshold instantiation
	void SetFOOThresh(double happyForThreshold, double resentMentThreshold, double gloatingThreshold,
double pityThreshold)
	{
		*emoteThresh[HappyforResentment].posThreshold = abs(happyForThreshold);
		*emoteThresh[HappyforResentment].negThreshold = -1 * abs(resentMentThreshold);
		*emoteThresh[GloatingPity].posThreshold = abs(gloatingThreshold);
		*emoteThresh[GloatingPity].negThreshold = -1 * abs(pityThreshold);
	}
	void SetWBThresh(double joyThreshold, double distressThreshold)
	{
		*emoteThresh[JoyDistress].posThreshold = abs(joyThreshold);
		*emoteThresh[JoyDistress].negThreshold = -1 * abs(distressThreshold);
	}
	void SetProspectThresh(double hopeThreshold, double fearThreshold, double satisfactionThreshold, double
fearsconfirmedThreshold, double reliefThreshold, double disappointmentThreshold)
	{
		*emoteThresh[HopeFear].posThreshold = abs(hopeThreshold);
		*emoteThresh[HopeFear].negThreshold = -1 * abs(fearThreshold);
		*emoteThresh[SatisfactionFearsconfirmed].posThreshold = abs(satisfactionThreshold);
		*emoteThresh[SatisfactionFearsconfirmed].negThreshold = -1 * abs(fearsconfirmedThreshold);
		*emoteThresh[ReliefDisappointment].posThreshold = abs(reliefThreshold);
		*emoteThresh[ReliefDisappointment].negThreshold = -1 * abs(disappointmentThreshold);
	}
	void SetAttributThresh(double prideThreshold, double shameThreshold, double admirationThreshold, double
reproachThreshold)
	{
		*emoteThresh[PrideShame].posThreshold = abs(prideThreshold);
		*emoteThresh[PrideShame].negThreshold = -1 * abs(shameThreshold);
		*emoteThresh[AdmirationReproach].posThreshold = abs(admirationThreshold);
		*emoteThresh[AdmirationReproach].negThreshold = -1 * abs(reproachThreshold);
	}
	void SetCompoundThresh(double gratificationThreshold, double remorseThreshold, double gratitudeThreshold,
double angerThreshold)
	{
		*emoteThresh[GratificationRemorse].posThreshold = abs(gratificationThreshold);
		*emoteThresh[GratificationRemorse].negThreshold = -1 * abs(remorseThreshold);
		*emoteThresh[GratitudeAnger].posThreshold = abs(gratitudeThreshold);
		*emoteThresh[GratitudeAnger].negThreshold = -1 * abs(angerThreshold);
	}

	EmoteTree::~EmoteTree()
	{
		for (int i = 0; i < (TotalPersonalEmotions + TotalSocialEmotions); i++)
		{
			delete(emoteThresh[i].negThreshold);
			emoteThresh[i].negThreshold = NULL;
			delete(emoteThresh[i].posThreshold);
			emoteThresh[i].posThreshold = NULL;
		}
	}

};
```

Appendix A 3.4 – **Eliciting Factor** Classes

```
#pragma once
#include <tuple>

enum ElicitType{eAgent, eFortunes, eWellBeing, eProspect, eProsDone};

class ElicitingFactor
{
protected:
	ElicitingFactor(std::pair<double, double> desirability, std::pair<double, double> senseofreality, std::pair<double, double> proximity, std::pair<double, double> unexpectedness)
		: desireability(desirability), sR(senseofreality), prox(proximity), unexp(unexpectedness){}

	~ElicitingFactor(){}

private:
	//Applicable as power for all
	std::pair<double, double> desireability;

	//Global
	std::pair<double, double> sR;
	std::pair<double, double> prox;
	std::pair<double, double> unexp;

public:
	virtual int getType() = 0;

	std::pair<double, double> Desirability(){ return desireability; }
	std::pair<double, double> SenseOfReality(){ return sR; }
	std::pair<double, double> Proximity(){ return prox; }
	std::pair<double, double> Unexpectedness(){ return unexp; }
};

//Elicit Admiration or Reproach, Pride or Shame
class ElicitAgent : public ElicitingFactor
{
private:
	int charID;
	std::pair<double, double> expDev;
	std::pair<double, double> strCogU;

public:
	ElicitAgent(int charID, std::pair<double, double> praiseworthy, std::pair<double, double> expectationDeviation, std::pair<double, double> actionassociation, std::pair<double, double> senseofreality, std::pair<double, double> proximity, std::pair<double, double> unexpectedness)
		: ElicitingFactor(praiseworthy, senseofreality, proximity, unexpectedness), charID(charID), expDev(expectationDeviation), strCogU(actionassociation){}

	~ElicitAgent(){}

	int getType() override { return eAgent; };

	int CharID(){ return charID; }
	//ExpectationDeviation
	std::pair<double, double> ExpDev(){ return expDev; }
	//StrengthOfCognitiveUnit
	std::pair<double, double> StrCogU(){ return strCogU; }
};

//Elicit Gloating or Pity, Happyfor or Resentment
class ElicitFortunesOfOther : public ElicitingFactor
{
private:
	int charID;
	std::pair<double, double> desiFO;
	std::pair<double, double> deserve;
	std::pair<double, double> liking;

public:
	ElicitFortunesOfOther(int charID, std::pair<double, double> desirability, std::pair<double, double>
```

```
desirabilityForOther, std::pair<double, double> deserving, std::pair<double, double> liking, std::pair<double, double>
senseofreality, std::pair<double, double> proximity, std::pair<double, double> unexpectedness)
                        : ElicitingFactor(desirability, senseofreality, proximity, unexpectedness), charID(charID),
desiFO(desirabilityForOther), deserve(deserving), liking(liking){}

        ~ElicitFortunesOfOther(){}

        int getType() override { return eFortunes; };

        int CharID(){ return charID; }
        //DesireableForOther
        std::pair<double, double> DesiFO(){ return desiFO; }
        //Deserving
        std::pair<double, double> Deserve(){ return deserve; }
        //Liking
        std::pair<double, double> Liking(){ return liking; }
};

//Elicit Joy or Distress
class ElicitWellBeing : public ElicitingFactor
{
public:
        ElicitWellBeing(std::pair<double, double> desirability, std::pair<double, double> senseofreality, std::pair<double,
double> proximity, std::pair<double, double> unexpectedness)
                        : ElicitingFactor(desirability, senseofreality, proximity, unexpectedness){}

        ~ElicitWellBeing(){}

        int getType() override { return eWellBeing; };
};

//Elicit Hope or Fear
class ElicitProspect : public ElicitingFactor
{
private:
        bool prospectDone;
        std::pair<double, double> likelihood;
        std::pair<double, double> effort;
        std::pair<double, double> realization;

public:
        ElicitProspect(bool prospectDone, std::pair<double, double> desirability, std::pair<double, double> likelihood,
std::pair<double, double> effort, std::pair<double, double> realization, std::pair<double, double> senseofreality,
std::pair<double, double> proximity, std::pair<double, double> unexpectedness)
                        : prospectDone(prospectDone), ElicitingFactor(desirability, senseofreality, proximity, unexpectedness),
likelihood(likelihood), effort(effort), realization(realization){}

        ~ElicitProspect(){}

        int getType() override { return (prospectDone ? eProsDone : eProspect); };

        //Likelihood
        std::pair<double, double> Likelihood(){ return likelihood; }
        //Effort
        std::pair<double, double> Effort(){ return effort; }
        //Realization
        std::pair<double, double> Realization(){ return realization; }
};
```

Appendix A 3.4 – Emotion Controller Structs

```
#pragma once
#include "Emote.h"
#include "ElicitingFactor.h"

enum SocialEmotions{ AR, HR, GP, GA, TotalSocialEmotions };
enum PersonalEmotions{ PS, JD, HF, SFc, RD, GR, TotalPersonalEmotions };

struct EmotionalPotentials
{
        double* posPotential;
```

```
        double* negPotential;
};

struct EmotionalThresholds
{
        double* posThreshold;
        double* negThreshold;
};

struct Em
{
        ValencedEmotions name;
        ElicitingFactor* elicitID;
        double* intensity;

        EmotionalPotentials emotePot;
};
```

Appendix A 3.6 – **Emotional Output Struct**

```
#pragma once

#include "ElicitingFactor.h"

enum ValencedEmotions{ AdmirationReproach, HappyforResentment, GloatingPity, GratitudeAnger, PrideShame,
JoyDistress, HopeFear, SatisfactionFearsconfirmed, ReliefDisappointment, GratificationRemorse,
TotalValencedEmotions };

struct Emote
{
	int* directedAt;
	ValencedEmotions* name;
	ElicitingFactor* elicitID;
	double* intensity;
};
```

Appendix A 4.1 – Additional Math

```
#pragma once
#include <cmath>
#include <algorithm>

//logarithmic base-e
const double loge = 2.718281828459046;

class AddMath
{
public:

	template <typename T>
	static T clamp(const T& n, const T& lower, const T& upper) {
		return (std::max)(lower, (std::min)(n, upper));
	}

	/*****************************************************************************************
		Based on modification of Slope Intercept Form by Dave Mark (Intrinsic Algorithm)
	*****************************************************************************************/
	static double const QuadCurve(double m, double k, double c, double b, double x)
	{
		return m * pow(x - c, k) + b;
	}

	/*****************************************************************************************
		Based on modification of Sigmoid Function by Dave Mark (Intrinsic Algorithm)
	*****************************************************************************************/
	static double const LogCurve(double m, double k, double c, double b, double x)
	{
		//Calculate e taking into account slope of line at inflection point m
		double em = (1000 * loge * m);

		//Calculate exponent for slope taking into account horizontal shift
		double xc = -x + c;

		//Create curve base
		double pw = 1 + pow(em, xc);

		//Truncation of decimals to allow for values such as 1.0 and 0.0
		double tr = std::floor(pw * 100) / 100;

		//Calculate the final slope taking into account vertical size and shift of slope
		return (k * (1 / tr)) + b;
	}

private:

};
```

Appendix A 4.2 – Behavior Tree Implementation

```
#include <list>
#include <vector>
#include <stack>
#include <initializer_list>
#include <cstdlib>
#include <algorithm>
#include <sstream>

class  BehaviourTree
{
public:
	//Base node
	class Node
	{
	public:
		virtual bool run(){ return false; }
	};

	//Composite nodes allow multiple children
	class CompositeNode : public Node
	{
	private:
		std::vector<Node*> nChildren;
	public:
		const std::vector<Node*>& getChildren() const{ return nChildren;}
		void addChild(Node* child){ nChildren.emplace_back(child); }
		void addChildren(std::initializer_list<Node*>&& children){ for (Node* child : children){ addChild(child); }
}
		template <typename CONTAINER>
		void addChildren(const CONTAINER& newChildren) { for (Node* child : newChildren) addChild(child); }
	protected:
		std::vector<Node*> shuffledChildren() const{ std::vector<Node*> tNode = nChildren;
random_shuffle(tNode.begin(), tNode.end()); return tNode; }
	};

	//Node succeeds, if one child succeeds
	class Selector : public CompositeNode
	{
	public:
		virtual bool run() override
		{
			for (BehaviourTree::Node* child : getChildren()){ if (child->run()){ return true; } }
			return false;
		}
	};

	//Node succeeds if one or more child succeeds
	class SequenceSelector : public CompositeNode
	{
	public:
		virtual bool run() override
		{
			bool *b = false;
			for (BehaviourTree::Node* child : getChildren()){ if (child->run()){ *b = true; } }
			return *b;
		}
	};

	//Same as above, shuffled order
	class RandomSelector : public CompositeNode
	{
	public:
		virtual bool run() override
		{
			for (Node* child : shuffledChildren()){ if (child->run()){ return true; } }
			return false;
		}
	};
```

```
	//Node succeeds, if both children succeed
	class Sequence : public CompositeNode
	{
	public:
		virtual bool run() override
		{
			for (Node* child : getChildren()){ if (!child->run()){ return false; } }
			return true;
		}
	};

	//Nodes that either transforms the result received from child node, terminates the child, or repeats processing of the child
	class DecoratorNode : public Node
	{
	private:
		Node* nChild;
	protected:
		Node* getChild() const { return nChild; }
	public:
		void setChild(Node* child){ nChild = child; }
	};

	//single child root node
	class Root : public DecoratorNode
	{
	private:
		friend class BehaviourTree;
		virtual bool run() override { return getChild()->run(); }
	};

	//Inverts result of child
	class Inverter : public DecoratorNode
	{
	private:
		virtual bool run() override { return !getChild()->run(); }
	};

	//Succeeds regardless of child
	class Succeeder : public DecoratorNode
	{
	private:
		virtual bool run() override { getChild()->run();  return true; }
	};

	//Fails regardless of child
	class Failer : public DecoratorNode
	{
	private:
		virtual bool run() override { getChild()->run();  return false; }
	};

	//Repeats infinitly or for N iterations
	class Repeater : public DecoratorNode
	{
	private:
		int numRepeats;
		Repeater(int num = -1) : numRepeats(num) {}
		virtual bool run() override
		{
			if (numRepeats == -1)
				while (true) getChild()->run();
			else {
				for (int i = 0; i < numRepeats - 1; i++)
					getChild()->run();
				return getChild()->run();
			}
		}
	};
```

```
	//Repeats until operation fails
	class RepeatUntilFail : public DecoratorNode
	{
	private:
		virtual bool run() override{ while (getChild()->run()){} return true; }
	};

	template<typename T>
	class StackNode : public Node
	{
	protected:
		std::stack<T*>& stck;
		StackNode(std::stack<T*>& s) : stck(s){}
	};

	template<typename T>
	class PushToStack : public StackNode<T>
	{
	private:
		T*& item;
	public:
		PushToStack(T* t, std::stack<T*>& s) : StackNode<T>(s), item(t){}
	private:
		virtual bool run() override
		{
			this->stck.push(item);
		}
	};

	template<typename T>
	class GetStack : public StackNode<T>
	{
	private:
		const std::stack<T*>& obtainedStack;
		T* object;
	public:
		GetStack(std::stack<T*>& s, const std::stack<T*>& o, T* t = nullptr) : StackNode<T>(s),
obtainedStack(o), object(t){}
	private:
		virtual bool run() override
		{
			this->stck = obtainedStack;
			if (object)
			{
				this->stck.push(object);
			}
			return true;
		}
	};

	template<typename T>
	class PopFromStack : public StackNode<T>
	{
	private:
		T*& item;
	public:
		PopFromStack(std::stack<T*>& s, T*& t) :StackNode<T>(s), item(t){}
	private:
		virtual bool run() override
		{
			if (this->stck.empty())
			{
				return false;
			}
			item = this->stck.top();
			this->stck.pop();
			return true;
		}
	};
```

```
	template<typename T>
	class StackIsEmpty : public StackNode<T>
	{
	public:
		StackIsEmpty(std::stack<T*>& s) : StackNode<T>(s){}
	private:
		virtual bool run() override
		{
			return this->stck.empty();
		}
	};

	template<typename T>
	class SetVariable : public Node
	{
	private:
		T *&variable, *&object;
	public:
		SetVariable(T*& t, T*& o) : variable(t), object(o){}
		virtual bool run() override
		{
			variable = object;
			return true;
		}
	};

	template<typename T>
	class IsNull : public Node
	{
	private:
		T*& object;
	public:
		IsNull(T*& t) : object(t){}
		virtual bool run() override { return !object; }
	};

private:
	Root* root;
public:
	BehaviourTree() : root(new Root){}
	void setRootChild(Node* rootChild) const{ root->setChild(rootChild);}
	bool run() const { return root->run(); }
};
```

Appendix B – Test results