A General Agent Model of Emotion and Trust using the BDI Structure

Acknowledgement

Before starting with the thesis itself I would like to thank four people for helping me through this project. First I want to thank dr. T Bosse for his extensive help throughout this project. Secondly, dr. M Hoogendoorn as second reader. Thirdly, I would like to thank P. Sannes for his nightlong help for making the RoboCup environment operatable on my computer. Last, but not least, my great and sincere thanks goes to dr. T Gabel. Firstly, and this is for the whole Brainstormers team, to make their source code publicly available. Furthermore, dr. T Gabel, who has never seen me, spent a lot of time helping me implement my model inside their team code. Besides helping me with the team’s source code, he also provided important information on the RoboCup Soccer environment, mainly about the possibilities within it.

Daniël Höhle
Student number: 1729896
E-mail: daniel_hohle@hotmail.com
Summary
To endow virtual agents with more human-like behaviour, the current project proposes a new model for decision making based on emotions and trust. The project focuses first on the conceptual design of the model, then on the implementation of this model. To try to reproduce human decision making, the model has, mostly, been based on psychologically grounded findings. This model is based on the BDI structure, and has been extended with submodels for emotions and trust. To further prove the implementability of the model, it was implemented first - at a conceptual level - in LeadsTo, and thereafter in the RoboCup soccer environment. The implementation in LeadsTo shows exactly how the connections and information flows of the model work. The results demonstrate that agents using emotions to bias their decisions act differently than the agents that use rational thinking. To investigate these results further, and in a more dynamic environment, the model was then successfully implemented in the RoboCup soccer environment, 2D simulation. Some of the results were significant differences in actions between players with different goals, and differences in some of the performed actions between a positive versus a negative emotional team.
## Contents

Summary ........................................................................................................................................... 1  

1.0 General Introduction .................................................................................................................. 5  

2.0 Theories on Emotions and Trust ............................................................................................... 6  

   2.1 Theories on Emotions ......................................................................................................... 6  

   2.2 Theories on Trust .............................................................................................................. 7  

3.0 History of Emotional BDI Systems ......................................................................................... 7  

4.0 Introduction to RoboCup ......................................................................................................... 8  

5.0 Model’s Overview ..................................................................................................................... 9  

   5.1 Traditional BDI model ....................................................................................................... 9  

   5.2 Extended eBDI model ....................................................................................................... 9  

      5.2.1 Why emotions? ........................................................................................................... 9  

      5.2.2 The model itself ......................................................................................................... 10  

      5.2.3 New state: Selected-Intention state ........................................................................... 10  

      5.2.4 New state: Emotion ................................................................................................. 11  

   5.3 Specifying the model’s lower levels ...................................................................................... 11  

      5.3.1 Emotions .................................................................................................................. 11  

      5.3.2 Beliefs ..................................................................................................................... 13  

6.0 Model in LeadsTo ..................................................................................................................... 14  

   6.1 Model without emotions ....................................................................................................... 15  

      6.1.1 Terms to be displayed .............................................................................................. 15  

      6.1.2 Sort declaration and initialization .............................................................................. 16  

      6.1.3 Weight initialization ................................................................................................. 16  

      6.1.4 Initial/start-off values ............................................................................................... 16  

      6.1.5 Simulation rules ........................................................................................................ 17  

      6.1.6 Simulation without emotions .................................................................................... 18  

   6.2 Model with emotion ............................................................................................................. 24  

   6.3 Scenarios ............................................................................................................................. 27  

      6.3.1 Examining action_outputs ...................................................................................... 28  

      6.3.2 Examining emotions ............................................................................................... 30  

   6.4 Final conclusion simulation results ....................................................................................... 31  

7.0 RoboCup: The implementation of the eBDI model ................................................................. 32  


 Daniël Höhle: A General Agent Model of Emotion and Trust Using the BDI Structure  

Page 3
7.1 Implementation in RoboCup…………………………………………………………………………………………………………………………..33
7.2 Basic eBDI model…………………………………………………………………………………………………………………………………………34
7.3 Differences between LeadsTo and RoboCup…………………………………………………………………………………………………………………34
7.4 Processing structure ……………………………………………………………………………………………………………………………………………35
   7.4.1 Can perform action …………………………………………………………………………………………………………………………………………37
   7.4.2 Action expectation …………………………………………………………………………………………………………………………………………37
   7.4.3 Intention generator …………………………………………………………………………………………………………………………………………37
   7.4.4 Selected Intention………………………………………………………………………………………………………………………………………39
   7.4.5 Action handler…………………………………………………………………………………………………………………………………………39
7.5 Simulation results ………………………………………………………………………………………………………………………………………………40
   7.5.1 Development of Mental States ………………………………………………………………………………………………………………………40
   7.5.2 Positive versus Negative teams ……………………………………………………………………………………………………………………43
7.6 Conclusion RoboCup ……………………………………………………………………………………………………………………………………………45
7.7 Future work RoboCup ………………………………………………………………………………………………………………………………………46
8.0 Conclusion and Future Work ……………………………………………………………………………………………………………………………47
References ………………………………………………………………………………………………………………………………………………………………49
Appendix 1 ……………………………………………………………………………………………………………………………………………………………51
Appendix 2 ……………………………………………………………………………………………………………………………………………………………52
1.0 General Introduction
Recently, an increasing amount of virtual agents have been produced for a vast range of applications. Some of these applications include (serious) gaming, web support agents and learning environments. In the 20th century, computer scientists only used exclusive algorithms to solve difficult decision making problems. Around the beginning of the 21st century emotions were introduced more frequently in computer systems. Within this research area a major division can be made. One group specializes on agents that present emotions in the form of facial expressions (Queiroz et al, 2009), whereas, the other group is more focused on the mental structure of virtual agents. It is, regardless of the purpose of the application, always the aim to produce an agent that can perform its task well. Nonetheless, a lot of agents, nowadays, are still based on ‘rational’ thinking. Rational, in this case, can be seen as some sort of ‘straight forward’ or predefined reasoning principle. This can be thought of as good, because there are no unexpected outcomes. But, this could hold the problem that the application is not suitable to cope with a vast amount of changes in the (virtual) environment. A study on patients with brain damage have shown that patients without the brain structure to ‘feel’ emotions are not ‘superrational’ (Damasio, 1996) but incapable of making any decisions at all (Meyer, 2004). Meyer further argues that emotions are necessary to stop reasoning. This indicates the importance of emotions for human decision making. Whereas humans have an (almost) infinite set of possible decisions they can make, computer applications usually have a limited set of decisions. Although, rational thinking (i.e. thinking without emotions) can only be appropriate for fairly small applications, when the set of possible actions/decisions (or combinations of them) becomes too large, even the fastest computers are incapable of ‘rationally’ calculate all the possible outcomes. For example, the chess playing computer created by IBM, Deep Blue, used a tree structure with all possible changes for the next cycle (Campbell et al., 2002). Although this system uses prior knowledge on known strategies it still processes the decision tree more broadly than human’s would do. Therefore, the decision has to be biased, based on satisfying results, during similar circumstances, from the past, instead of examining all the possible series of actions. To try to achieve this, a generic reusable model has to be created which can be used for numerous different applications.

Thus, the main goal of this thesis is to let virtual agents behave in a more human-like fashion. To achieve this goal I will use as basic model that originates from the folk psychology, called BDI model. The BDI model is thought of as to be a model on how people decide to perform certain actions. This BDI model is based purely on rational thinking (Rao & Georgeff, 1995), although it is, from the psychological perspective, clear that decision making is influenced by more than rational reasoning (Damasio, 1996). Some of the influences are: personality, hormones, sex, trust, and emotions. This article will start off with the existing BDI model and will then be upgraded with trust and emotions. This is not meant to be a final model of how human thinking works, but merely an update and kickoff in the direction of more human-like behaviour in virtual agents. This upgraded version of the BDI model will be demonstrated in a soccer environment, called RoboCup. The reason for this simulation environment is that RoboCup is designed for research purposes, with as goal to reflect the world’s environment realistically, on a two dimensional level. This application provides feedback on recent changes and, therefore, gives the possibility for observations and actions to interact from and within the environment. These observations also include basic properties of other players, depending on the direction the player is looking at. This
two dimensional simulation environment makes interaction between players a crucial property, since 22 players operate in an ever changing environment. This interaction makes the concepts such as trust and emotions possible and useful. The players update their trust and emotions based on the results of their actions, which can be done because the rules in soccer are ‘fairly’ simple.

An introduction on emotions and trust in virtual agents will be discussed next. The third chapter gives a brief introduction to existing emotional BDI implementations. The RoboCup soccer simulation will be handled in chapter 4. In chapter 5 the new eBDI model will be described. The workings of this model will first be displayed in the LeadsTo environment in chapter 6. In chapter 7 this conceptual model will be implemented in the RoboCup environment. Finally, some conclusions and final remarks will be depicted in the final section, chapter 8.

2.0 Theories on Emotions and Trust

2.1 Theories on Emotions

The importance of emotions was already emphasized by Darwin (1872). Darwin argued that each living creature uses emotions in decision making and adapting to the environment. To follow up with trying to describe how emotions operate different theories evolved. Adam (2007) listed the different trends of emotions. Thereby making a division in discrete, continuous, physiological, and cognitive theories. Discrete theories are based on a limited set of basic emotions (e.g. happy, sad) where the different emotions differ importantly from the others. Within the discrete models another subdivision is made between evolutionary, non-evolutionary, and building blocks. The evolutionary models are based on the theory posed by Darwin (1872) which is based on the evolution of species. Tomkins (1962) argues that emotions are organized in a physiological fashion without any interference of cognitive processes. Emotions as building blocks are proposed by Plutchik (1980) who presents emotions as a ‘coloured wheel’. There he defined eight basic emotions, where he argues that all other emotions are a combination between the eight basic ones.

Continuous theories, base their emotions on different ‘dimensions’ (e.g. valence, stance, arousal) where a particular feeling merely drives towards ‘basic’ emotion (for instance Breazeal, 2003). Physiological theories (James, 1884) state that emotions are solely created by physiological chances created as a result of incoming stimuli. The bodily changes warn for a recent negatively changed situation, where the resulting action is then based on. The cognitive theories focus on emotions as a combination of cognitive and physiological activation.

Arnold (1960) proposes the concept of cognitive appraisal. She proposes that incoming information is first categorized as either good or bad for the current individual. The physiological responses take place after deciding whether the event is good or bad for the individual. This line of reasoning will be used throughout this thesis. Lazarus (1984) adds to Arnold that the same stimulus can be categorized good and bad depending on the environment and the individual perceiving the stimulus (Marsella & Gratch, 2002). Lazarus (1966, 1991) further states that the triggered emotion is dependent on the individual’s motivation and the experiences with the environment. The previously triggered
emotion is then used to drive towards particular actions. Ortony, Clore and Collins (1988) made a model to incorporate emotions in the BDI (Beliefs, Desires, and Intention) structure. Their approach was to address emotions to certain consequences of events, actions of agents, and aspects of object. Whereby, the consequences for the agents result in an emotion, as well as for the agent itself as for the possible interacting agents (see Ortony et al, 1988, Figure 2.1).

2.2 Theories on Trust
Models using trust, as argued by Jonker et al (2004), are usually based on intuition and not tested on their validity. They further argue that trust is a function which is of great importance in our daily lives. Without trust our social lives will be in trouble because of the (possible) lack of cooperation. Nootenboom (2005) has built a model based on recent psychological knowledge in trust. He argues that trust is often used in a too broad fashion, where, for instance, also social expectations are usually embedded in the broad meaning of trust. Jonker and Treur (1999) made a formal model of adaptation in trust. They started with a model with 4 different values of trust. The highest form of trust is unconditional trust followed by conditional trust, conditional distrust, and unconditional distrust. When something good happens the trust is increased, per event, until the highest form of trust is reached. The same holds for negative experiences. They, furthermore, made a quantitative trust model where the trust values are between -1 and +1. In this case the events are not just positive or negative, but have a value corresponding to the trustworthiness of the event. This value is then used to update the old trust value. Jonker et al (2004) performed an experiment to prove the model described before. They used 238 subjects to make the model produced by Jonker and Treur (1999) experimentally valid. The results confirmed this hypothesis.

In the current model the same concept of trust is used. With as only different the range of the trust value, which is between 0 and 1, instead of between -1 and +1. On the contrary of the results discussed before, the trust used in this article is based on capabilities. Capabilities are updated in the same way as trust was in the article of Jonker and Treur (1999). Trust is then build up as a sum over all the capabilities. This will be discussed in more detail in section 5.3.2. Notice that this interaction between capability and trust is, for now, only an assumption.

3.0 History of Emotional BDI Systems
Different emotional BDI systems have been made for different purposes over the last decade (Adam, 2004; Steunebrink et al, 2007). Although they all introduced emotions to the BDI system their attempts and reasons are different. Steunebrink et al (2007) made heuristics to introduce 22 different emotions to the OCC model (Ortony et al, 1988). Bosse and Zwanenburg (2009) made a model to increase the believability of the virtual agent. The emotions used in their BDI model are only operatable for the action selection. Jiang et al (2007) showed in their article that emotions in their model are updated by a combination of beliefs and intentions, whereas desires are generated by the belief that an action can be executed. Adam (2004) argued for the misconception of desires versus goals. According to her criticisms, the desires used as in the article of Jiang et al (2007) are not desires but merely goals. Goals, as she
argues are more like sub-desires, smaller steps towards a desire, whereas desires are abstracter and more long term.

The current article proposes to use emotions on all the mental states (see Figure 1). Although, currently, only two kinds of emotions are used (positive and negative) the main purpose is to suggest a manageable eBDI model with its relevant connections. The graphical model used here is directly based on the model visualized by Bosse et al (2007).

4.0 Introduction to RoboCup

RoboCup is, generally speaking, made to promote robot research. The idea behind this concept is to promote research in the areas of robotics, machine learning, and multi-agent modelling, although other Special Interest Groups (SIGs) are always welcome to subscribe. The concept of RoboCup was first suggested in 1993. The first conference was held in 1997, when the RoboCup competition was first held. After this conference, this became an annual meeting with increasing popularity. Nowadays, more than 400 teams and 2000 participants spread out over 35 countries/regions are using RoboCup as main research area. The federation wants to accelerate research in the soccer branch by annual conferences with competitions. The RoboCup Federation has proposed a final goal, which is:

“By 2050, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying the official FIFA rules, against the winner of the most recent World Cup of Human Soccer” (www.robocup.org).

To try to achieve this, the federation tries to push the researches to share their approaches, so that different branches of research can be combined into something better.

RoboCup has three branches: soccer, rescue, and junior. Whereas the soccer branch is divided into three sub-groups: simulation, mixed-reality, and reality. The simulation branch can be divided into a two-dimensional and three-dimensional version. The former has as main interest the interaction between the players, where the latter is based on humanoid robots in a virtual physical environment. Mixed-reality was introduced due to pressure of a sponsor (Citizen) but never became a real league. It is currently only used for display purposes, but has almost no support from the RoboCup federation anymore.

In the real-world league (i.e. reality), small, middle-sized, standard platform, and humanoid robots use a physical environment to play on. The small and middle-sized robots use wheels to move around the environment and the robots in the standard-platform and the humanoid version use legs. The robots used for the standard-platform league were using four legs until 2008, nowadays they use two legs. The principle of this league is that all the teams have the same robots, so the teams can concentrate on their strategies instead of improving the physical qualities of the robots. Finally, in the humanoid version the teams have to build their own robots and they have to use human-like movement. Here the robots are divided into two categories: “KidSize” from 30 to 60 centimetres of height and “TeenSize” with heights between 100 and 160 centimetres.
5.0 Model’s Overview
As Figure 1 shows, five internal states are taken into account. These include the states from the BDI model proposed by Rao & Georgeff (1995). The additional parts of this model are the states; emotions and Selected-Intention. The numbers listed here are Identifiers which will be discussed later.

![General BDI Model with Emotion and Trust](Image)

5.1 Traditional BDI model
The original BDI model only included three modules. The belief module is responsible for collecting and remembering information from the environment. The desire module is used for storing goals. The intention module is for selecting an action that strives for or satisfies a desire. The intention is then executed when the agent has the belief that the intention can be performed.

5.2 Extended eBDI model
5.2.1 Why emotions?
Please notice that the most important difference here is that emotions are included. There are several reasons why this is done. The original model, for instance, is too rational. Other decision making techniques for virtual agents, like neural networks, are always based on making the agents as smart as possible (i.e. optimal decision making). The purpose of this model is ‘not’ to develop the most intelligent agents, but to make them behave more human-like. Since, as discussed before, human do not reason rationally and especially not in the most ‘intelligent’ way (although people will disagree here), but merely use gut feelings to make decisions (Damasio, 1998; Khatri, 2000). Gut feelings (Gigerenzer, 2008) appear quickly in consciousness without apparent reason and are strong enough to act upon. These gut feelings are triggered by emotional events, which can be characterized into four emotions (sad, anger,
fear, happy). Notice, though, that there is still much debate over the number of emotions humans have, some papers say just four, while other say over one hundred. To investigate emotions further (Vuilleumier et al, 2002) showed that even patients with visual extinction or spatial neglect (patients without the ability to, consciously, observe either the left or the right side with respect to the fixation point) responded to fearful faces regardless of whether the stimulus was presented in the visual or the invisible side of fixation. This shows that even though the subjects were not consciously aware of the stimulus it was still processed. This effect, although less apparent, has also been found on anger (Beaver et al, 2008). This indicates that emotions are processed before the rest of the environment is completely analyzed (i.e. aware). Therefore, see Figure 1, is the first link from the modules Emotion and Input to the module Beliefs.

5.2.2 The model itself
When an agent gets an input (e.g. from observation) this is first converted to a belief, but according to the emotion the agent has towards the belief, this belief is adjusted in strength. For instance, the capabilities and trust in player X is less negatively adjusted when an agent is happy and observes that a team mate makes an error (e.g. 'misses the ball completely') than when the former agent is angry or sad (note: that the mood as well as the emotion for such an event are taken into account). The generated belief with the emotion the agent has on that belief then updates the desires. Intentions are created by the desire in combination with the emotion for that specific generated action. When the intention is generated and the agent believes that this intention can be performed (either when this belief had already been persistent or when a new belief for the possibility of successfully executing the action is formed) the agent will send this intention to the next state, the Selected-Intention state.

5.2.3 New state: Selected-Intention state
Another adjustment with regard to the original model is an additional intention state. Since in this model there are a lot of possible actions, desires, and players this also means that multiple intentions can be generated at the same time (since multiple actions are possible). Therefore, we have chosen to include the selected-intention state. The Selected-Intention state is included as a state, which ‘determines’ the intention that has to be executed based on the selection chance value of this ‘intention’. Let’s, for example, say that there are three possible actions. Each possible action gets a value used for determining the chance of being selected. This is dependent on the expectation value, which is the chance of success plus the gain towards a certain goal. Let’s give action 1 and 2 a 25 percent chance of being selected and action 3 a 50 percent chance. Then, in the selected-intention state, a random number generator will provide a number that will be compared to the action corresponding to that number. This is the action, in the selected-intention state, the agent tries to execute. To conclude the example, action 1 is chosen if the number is between 0 and 0.25, action 2 if the number is between 0.25 and 0.5 and action 3 when the number is between 0.5 and 1. When one action is ‘selected’ the agent performs this action when there is the belief that this action can be executed. This state, like the other states, is updated constantly and therefore can be erroneous in wanting to perform the ‘wrong’ action at one time and when they are willing to perform another action, the first one could have been executed successfully.
5.2.4 New state: Emotion
Emotion can be seen as one of the most important aspects of the model because it influences the beliefs of the world, the agent’s desires, and influences the possible intentions. Emotions are used in two ways, first when information comes in and secondly when the information is used. A generated belief on an event influences the emotion the agent has for that event, in combination with the emotion the agent already has for that event. This effect is in some way the same as an emotion over another agent. The biggest adjustment in the higher layer is the emotion module, since it links to the entire BDI structure, each link will now be discussed. These links are assumed based on the knowledge that the emotional system is not a physic part of the brain, but merely the amount of specific neurotransmitters that flow through the brain, such as dopamine (Salgado-pineda et al, 2005).

*Emotions and input to belief.* This link is meant for updating the beliefs on capabilities based on the emotion an agent has for another agent. For example, the observer sees a teammate ‘shoot at goal’, but he misses. When the observer is feeling sad or angry, the event is interpreted far more negatively than when the observer was happy. How the event is interpreted is then used for updating the belief the observer has in the capability of ‘shoot at goal’ for his teammate, which was performing the action.

*Belief, desire, and emotion to emotions.* The purpose of this link is to update the emotion an agent has for another agent or for certain events. This works as follows: when a belief is generated on an event, and the agent believes that this event helps finalizing his goal, then this belief will, together with the mood the agent has, produce a new emotion for that agent.

*Belief and emotion to desire.* This is used for updating desire values. When, for instance, a belief is made that a desire can be finalized by an action, the strength of that desire to be completed will be increased, when the agent is happy (positive). But when the agent is sad (negative), this desire can be decreased because the agent does not care about anything at all.

*Belief, desire, and emotion to intention.* The function of this link is to generate intentions (actions) according to certain beliefs the agent has on the environment and the desires the agent has. The emotion here is used to decide what the expectation value (chance of intention execution) is for that intention. Thus, if the agent is happy, it will be more likely to perform an action than if it were sad.

5.3 Specifying the model’s lower levels

5.3.1 Emotions
Emotions have either a positive or negative state, for simplicity reasons. These states have values to give a rating on how much that state is true for a given agent at a given time. There are two kinds of emotions, mood and event/agent-related emotions. The former is the overall feeling an agent has at a given time, which is influenced by the beliefs generated earlier in time in combination with the emotion the agent has for himself, but it is not a feeling specific to one person/event. The latter kind of emotion is simply called ‘Emotion’. This type is purely based on a specific event or agent without interference from the other ones. Yet, when an emotion has generated pure anger for an event, this event might influence the agent’s mood in a drastic way. A pure emotion is, for instance, unconditional anger (i.e. a value of 1.0). People usually don’t feel complete unconditional (for instance) anger for a person or
event, but more or less a gradual strength of an emotion, like the feeling between neutral and the emotion anger. This is because the vmPFC (ventral medial PreFrontal Cortex) regulates emotional responses (Greene 2007). Thus, normally, the higher the rate of emotion a person feels, the more the value comes towards that corresponding specific emotion.

Both versions of emotions are generated via the same path, both have inputs from beliefs and earlier emotions, and differ only in meaning and output. Mood determines the actions an agent chooses, when (for example) an agent is fearful, this agent will keep its distance from other players (depending on the level of anger of other agents with respect to the former agent). Nonetheless, the influence of ‘emotion’ is greater than the ‘mood’ when it comes to generating trust. Both mood and emotion have a link to themselves in the model. This link is important because emotion is based on a series of events, thus the current emotion is important in calculating the new one.

![Figure 2](image)

Figure 2 shows the model for emotions. The two states mentioned before can be seen in the middle section of this figure. Here emotion is only generated when new information (beliefs) is generated. If the generated belief is on the performance of an agent then the Emotion for that agent/event is adjusted, which is also dependent on the emotion that agent X (the observer) already has for agent Y (the observed one). When agent X sees that agent Y is performing a ‘good’ action (i.e. an action that satisfies one of the agent’s desires), the emotional value for agent Y will be increased. This also implies that the mood of agent X will slightly be adjusted. There is a small positive causal correlation between emotion and mood. The rate of the correlation is dependent on the state of mind of agent X (noticed the loop at the mood state). The output of this model influences the beliefs, desires, and intentions.
The output towards the beliefs is dependent on both the mood and the emotion agent X has for agent Y. For the desires, only the mood is needed for setting the (sub)goal to be executed later. Finally, the intention generation is, again, dependent only on mood. The emotion is only used for the connection towards belief generation, for updating the agent specific trust and capability settings.

5.3.2 Beliefs

If an agent observes a state change of the environment (e.g. time, score) this will first come to the belief state and can later be used to set or adjust goals. The knowledge base will primarily be used for static information (e.g. player’s team name). Where, on the other hand, the belief state is used for dynamically changing information (e.g. observation from the external world). If an agent observes an action performed by another agent, the observer builds up a belief on the action performed by the other player. The trust in that player, as well as its beliefs on capabilities, will be adjusted. Here the trust is a value for individual agents (between 0 and +1). The trust one has about themselves can be considered to be ‘self confidence’ and is of importance in the subdivision for the emotions; fear and anger. Beliefs on capabilities are used for individual agents but also for individual action performances. Each capability an agent is capable of performing has a performance value. This performance value is a value between zero and one and is used to calculate how good an agent is in performing an action. For example, if the action is shoot_at_goal, the performance value is based on the combination between precision and power. For ‘running’ the performance value is based on speed and durance. Beliefs on capabilities are estimations made on the quality of the actions performed by an agent. These estimations are based on the generated beliefs on a certain event. When an agent has the intention for
an action to be performed (for instance ‘a shot on the goal’) in combination with a positive capability expectation (the most positive capability number found) an action is set in the final internal state, selected intention. In the selected-intention state, the agent has determined his next action and when the possibility arrives, the action will be transferred to the output for performing the specified action.

Trust is, within the belief state, the next state depending on the capability values. The value of trust is an ever changing value depending on the values of the capabilities. An agent decides what capabilities are important for a specific player in the form of weights. These weights are used to make certain capabilities more important for updating the trust value.

6.0 Model in LeadsTo
LeadsTo is a program written for specifying simulation models. It is a rule-based declarative language based on the Prolog programming language. A pro of using LeadsTo is that as soon as rules are writing, results are displayed. This simulation environment consists out of two programs. 1) The Iteditor is used as programming application, 2) and LeadsTo is used for displaying the results. Notepad was used instead of the Iteditor, because there are no limitations of the number of sub sorts (will be discussed later), whereas the Iteditor itself cannot handle this. Important to know, though, is how to read the results. An example of the output is, therefore, displayed in Figure 4. On the left hand side, the atom is show. On the right hand side the time-line is displayed. This is used to show when the atom is true. The dark blue line is ‘true’, the light blue line is ‘false’. For instance, current_score(0,0) is true from time point 0 until 21, current_score(1,0) from 21 until 26, and so forth. Note: in this example all the atoms are either true or false, while the application can also handle a third value, unknown.

![Figure 4](image)

The model is first implemented in LeadsTo software for the following reasons; firstly to work out the entire model on a rule-based level and show the internal relations, and to test how the rules/components work best with respect to each other. A pro in this case, is that all the events can, and have to be, explicitly specified (e.g. "input(is_event(pass_from_to(playerZ, playerX)))."). This rules out the possibility of interference from unexpected input parameters, like an extra ball on the field.

Two models have been written in LeadsTo. The first model is without emotions, and the second one is the model where emotions are included. The reason for this is, twofold; the first reason is for ‘simplicity’ to make the more ‘rational’ decision model first and then adjust the decision making to a more human-like way. The second reason is to show the differences in output (i.e. actions, desires, intentions) between the ‘rational’ and ‘human-like’ decision making, using the four scenarios created. The scenarios will be discussed later. The model without emotion will be discussed first, thereafter how
the emotions are implemented, following with the scenarios and the results from the four scenarios without emotions and the same scenarios with emotions included. An overview of the model without emotions is shown in Appendix 1. This appendix consists out of squares (i.e. modules) and connecting lines. These modules list the atoms and sorts used in that module. When an atom is preceded by ‘sort’, then the connection indicates which module is inherited by the module displayed above.

6.1 Model without emotions

The rules in the simulation are built-up as components and links in the same way as the new Emotional Beliefs, Desires, and Intention (eBDI) model shows, but in this section the upper part (emotions) of the model is not discussed yet.

The software can be divided in eight parts: file declaration, the terms to be displayed, initialization of sorts (sort naming), defining sorts (possible values, i.e. actions), weight values, initial/start-off values, scenario, and the simulation rules. The most interesting part here is the simulation rules section where the conceptual model is implemented. File declaration is used for specific simulation properties. In this simulation I am using a closed world assumption (CWA), so an argument is either true or false, never unknown “qterm(cwa('_')).” Another important simulation parameter defined in the file declaration is the length of the simulation. The simulation here will execute for 35 time points: “end_time(35).”. The terms to be displayed will be discussed now.

6.1.1 Terms to be displayed

Important in this simulation environment is to declare which atoms are shown, and how. Would this have been left out, then the simulation results will display an extensive amount of data, which is nearly impossible to comprehend. Two different type of code are used, the first is to display the active rules with the values. The following example displays, for each player, the belief of the capability/skill of each player:


gterm('display(_, show_atoms(has_belief(_, has_capability(_,_,_))))').

But some of the values of the atoms are ever changing, like the atom shown before. Normally, LeadsTo will print a different line for each different value, see Figure 5. Figures are used to get a clearer understanding in what is happening with the values.

![Figure 5](image)

The following example demonstrates how to get a figure in the LeadsTo code.

qterm('display_number_range(has_belief(playerY, has_trust(playerX, X)), X, \' playerX_has_trust \', \' in X "\')').

The number range displayed in Figure 6 shows the changes in amount of trust Player Y has in Player X.
6.1.2 Sort declaration and initialization

This part of the software is meant for assigning elements, or sub sorts, to a specific sort. This will be used to make the program use all elements in a sort, instead of specific independent items. For instance, for actions: “sortdef (action, [shoot_at_goal, receive_ball, run_free, dribble, tackle, intercept_ball, pass_forward, pass_back]).” Here ‘action’ is the name of the sort, and the values between brackets ‘[]’ are the items in the sort. In the case of the sort action, every item is a term. Instead of using terms it is also possible to use subsorts, like:

```plaintext
sortdef(event, ['SUBSORT'(ball_info_event), 'SUBSORT'(player_info_event), 'SUBSORT'(player_specific_event), 'SUBSORT'(other_event), 'SUBSORT'(player_ball_action_event))].
```

A complete overview of the most important sorts is defined in Appendix 1.

6.1.3 Weight initialization

Two types of weight are used, one concerning the personality of the players and one handling the other value updates. The personality parameter is used instead of using static dependency parameters over the importance of a certain value; see for instance the trust weight displayed below. These weight parameters, listed in the beginning of the source file, are clustered per rule (e.g. the trust updating rule). The sum over all the weight parameters, used per rule, always adds up to 1 (i.e. 100%). These weight parameters are used for values updates, such as updating the trust values, for example:

```plaintext
constant(weight_old_trust, 0.3).
constant(weight_new_trust, 0.7).
```

Here the old trust value is 30 percent and the updated value is 70 percent of the new trust level.

6.1.4 Initial/start-off values

Values are initialized, for example, for desires, links between actions and desires, and knowledge information (i.e. ‘which agent plays for which team’). These are values that are not part of the model but are prerequisites in order to make the model work. This can be thought of as prior information for the agents. For instance:
This rule gives the agent with the name ‘playerX’ the desire to win the game of 0.5. The number range here is between 0-1, so 0.5 is a ‘neutral’ desire. The argument ‘range(0,1)’ is used for the time steps where the predicate has_desire() holds. This is only for one time step, because other persistency rules take over. Notice also the empty squared brackets ‘[]’, these can be used for instance as follows:

“interval([], range(0,1), has_desire(playerX, win_game, 0.5)).”.

Where in this case ‘a:player’ (where ‘player’ is a sort including the three players used in the simulation) is between the squared brackets which means that for all the players the predicate ‘exist(player)’ holds.

### 6.1.5 Simulation rules

The simulation rules are divided in sections equal to the modules and links in the conceptual model, as described earlier. The rules are identified using an Identifier with a number, to keep the source code clear. Furthermore, since there are two simulation models, one without emotions and one including emotions, these two models will be described both. To include emotions new rules have to be written. Though, some of the rules written for the without emotion simulation had to be adjusted in some form to be able to include emotions.

After explaining the global parts of the simulation, a selection of rules that are important for the simulation program itself will be explained. Starting with the rules where emotions are, not yet, part of. The emotional model will be explained thereafter. Yet, some of the state properties are the same in both the models. The rules that are changed, depending on the model, will be pointed out using either ‘(without emotion)’ or ‘(with emotion)’ after the Identifier. The rules just for without emotions are also identified by the identifier number followed by the letter ‘a’, where the rule for emotions included gets the letter ‘b’. First the state properties and their interpretations are explained, see Table 1.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_belief(p:player, e:event)</td>
<td>player p believes event e</td>
</tr>
<tr>
<td>has_belief(p:player, has_capability(p1:player, ac:action, v:value))</td>
<td>player p believes that player p1 has the capability to perform action ac with skill-level v</td>
</tr>
<tr>
<td>has_belief(p:player, has_trust(p1:player, v:value))</td>
<td>player p trusts player p1 with value v</td>
</tr>
<tr>
<td>has_desire(p:player, g:goal, v:value)</td>
<td>player p desires goal g with value v</td>
</tr>
<tr>
<td>has_intention(p:player, q:player, ac:action, v:value)</td>
<td>player p wants player q to perform action ac with value v (p can be q)</td>
</tr>
<tr>
<td>has_selected_intention(p:player, q:player, ac:action, v:value)</td>
<td>player p has selected action ac to be performed for player q (p can be q)</td>
</tr>
<tr>
<td>has_mood(p:player, e:emotion, v:value)</td>
<td>player p’s mood with emotion-type e with value v (emotion-type is happiness)</td>
</tr>
<tr>
<td>has_emotion_on_player(p:player, q:player, e:emotion, v:value)</td>
<td>player p feels emotion e with value v about player q (p can be q)</td>
</tr>
<tr>
<td>action_output(p:player, ac:action)</td>
<td>player p is performing action ac</td>
</tr>
</tbody>
</table>

Table 1: State properties and their interpretations
6.1.6 Simulation without emotions

Notice in the following example the ‘Identifier x’ before the explanation of rules. These identifiers link to a certain module or connection explained in the conceptual model, see Figures 1, 2, and 3. Now, to start off explaining the simulation rules, first the persistency rules will be described. Such as, for example, the rules used for updating the current score:

```prolog
leadsto([v:'INTEGER', v1:'INTEGER'],
       and(current_score(v,v1), not(new_score)),
       current_score(v,v1), efgh(0,0,1,1)).

leadsto([v:'INTEGER', v1:'INTEGER'],
        new_score(v,v1),
        current_score(v,v1), efgh(0,0,1,1)).
```

The first rule is executed if there is not a ‘new_score’ (so no goal is scored) and so that the new score is the same as the old score. If there is a ‘new_score’ then the ‘current_score’ will become the ‘new_score’. This ‘new_score’ is executed if: the ball_is_in_goal and there is no_foul. For example:

```prolog
leadsto([t:team, v:'INTEGER', v1:'INTEGER', p:all_players],
       and(current_score(v,v1),
           input(is_event(ball_is_in_goal(p, no_foul))),
           has_belief(p, has_knowledge(is_in_team(p,t))),
           t='Team_1'),
       and(new_score(v+1,v1),
           new_score),efgh(0,0,1,1)).
```

The following rules are listed in the LeadsTo file as identifier 0 through 12. To identify the links and components described in the conceptual model. Where identifier 0 is the beginning of the model, so from input the belief, and 12 is to the output.

**Identifier 0: input -> belief -> global event**

This first line of code is used to convert global inputs to global beliefs. This part only has 1 rule, which is:

```prolog
leadsto([e:event, p:player],
        and(input(event(e)), exist(p)),  // antecedent
        has_belief(p, event(e)),  // consequent
        efgh(0, 0, 1, 1)).  // the consequent is valid for 1 time step as well.
```

**Identifier 1: belief -> global event -> belief -> specific event**

As can be seen according to the consequent of the previous rule, the event is still global. Meaning that the agent has no information, yet, on what it can do with the information. Six different kinds of events have been characterized (see Appendix 1). The following rule is executed when the global event is of type ‘player_specific_event’. To convert the global event to a characterized event the following rule is used:

```prolog
leadsto([player:player_specific_event],
        input(is_event(player)),
        player_specific_event(player), efgh(0, 0, 1, 1)).
```
**Identifier 2: event -> output**

This rule is purely used to analyse the chance an action can be performed. Information on the environment, such as ball possession, is analysed. This includes a vast amount of rules so that all the different possible scenarios are taken into account (e.g. self ball, opponent ball, teammate ball).

The purpose of the example given below is to generate beliefs over whether an action can be performed. The agent 'self' believes that player 'ai' has ball possession and is in the same team as player 'a' and 'a1'. 'ai' is not in the same team as player 'o'. Furthermore, the location of player 'ai' is larger than of all the other players, meaning that player 'ai' is closer to the opponent's goal than the other 3 players. Knowing this, the actions that can be performed are calculated. There is no possibility for this player 'ai' to 'pass_forward' (since no teammate is in front of 'ai'), so this player could perform the actions 'dribble' or 'shoot_at_goal' better than to tackle or intercept the ball, since it is already in possession.

The agent 'self' performs this form of reasoning for all players, so if the player 'self' should be one of the other teammates (thus not 'ai'), then it could be a good action to pass_forward towards player 'ai'. Hereafter the values are given to all possible action including the execution value; note that in the consequent, each value is multiplied by two weights consistent with the importance of the trust and the capability operator:

\[
\text{leadsto}([\text{self:} \text{player}, \ ai: \text{all_players}, a: \text{player}, a1: \text{player}, o: \text{all_players}, t: \text{team}, t1: \text{team}, l: \text{integer}, l1: \text{integer}, l2: \text{integer}, l3: \text{integer}, \text{trust_value: real}],
\text{and(has_belief(self, game_event(ball_in_possession(ai))),}
\text{has_belief(self, has_knowledge(is_in_team(ai, t)))),}
\text{has_belief(self, has_knowledge(is_in_team(a, t)))),}
\text{has_belief(self, has_knowledge(is_in_team(a1, t)))),}
\text{has_belief(self, has_knowledge(is_in_team(o, t1)))),}
\text{has_belief(self, has_knowledge(player_field_position(a, defender)))),}
\text{not(has_belief(self, has_knowledge(player_field_position(a1, defender)))),}
\text{has_belief(self, player_info_event(player_is_at_location(ai, l)))),}
\text{has_belief(self, player_info_event(player_is_at_location(a, l1)))),}
\text{has_belief(self, player_info_event(player_is_at_location(a1, l2)))),}
\text{has_belief(self, player_info_event(player_is_at_location(o, l3)))),}
\text{has_belief(self, has_trust(ai, trust_value))},
\text{t1=t, ai=a, a1=a1, a1=a, l>l1, l>l2, l>=l3),}
\text{and(has_belief(self, update_can_perform_action(ai, pass_forward,}
\text{(0.1*weight_new_update_can_perform_action)+}
\text{(trust_value*weight_has_trust_on_update_can_perform_action)))},
\text{has_belief(self, update_can_perform_action(ai, pass_forward)),}
\text{has_belief(self, update_can_perform_action(ai, dribble,}
\text{(0.9*weight_new_update_can_perform_action)+}
\text{(trust_value*weight_has_trust_on_update_can_perform_action)))},
\text{has_belief(self, update_can_perform_action(ai, dribble)),}
\text{has_belief(self, update_can_perform_action(ai, tackle,}
\text{(0.1*weight_new_update_can_perform_action)+}
\text{(trust_value*weight_has_trust_on_update_can_perform_action)))},
\text{has_belief(self, update_can_perform_action(ai, tackle)),}
\text{has_belief(self, update_can_perform_action(ai, shoot_at_goal,}
\text{(0.8*weight_new_update_can_perform_action)+}
\text{(trust_value*weight_has_trust_on_update_can_perform_action)))},
\text{has_belief(self, update_can_perform_action(ai, shoot_at_goal)),}
\text{has_belief(self, update_can_perform_action(ai, shoot_at_goal))},
\text{has_belief(self, update_can_perform_action(ai, shoot_at_goal))).}
\]
Identifier 3a: specific event -> capability update (without emotion)

Here the skills (capabilities) of the players are updated, for the player itself as well as for the other players. This is based on the previous skill level updated by the event which has occurred recently. The following example is on updating the ‘shoot_at_goal’ skill, and how it is updated when the agent of which the skill is being accessed shoots at the goal and scores.

```
leadsto([self:player, p:all_players, v:'REAL'],
    and(has_belief(self, player_ball_action_event(shoot_at_goal)),
        has_belief(self, game_event(ball_in_possession(p))),
        has_belief(self, game_event(ball_is_in_goal(p, no_foul))),
        has_belief(self, has_capability(p, shoot_at_goal, v))),
    and(new_capability_update(self, p, shoot_at_goal, v+(1-v)/
        personality_weight)),
    capability_update(self, p, shoot_at_goal)), efgh(0,0,1,1)).
```

Identifier 4: capability -> trust update

A total of eight capabilities have been described. The update in trust is dependent on the location of the field position (defender, midfielder, and attacker) of the player of which is thought about. So in the example provided hereunder the agent is a defender, therefore the properties such as intercept_ball are more important than shoot_at_goal. This is defined as:

```
leadsto([(self:player, p:all_players, v1:'REAL', v2:'REAL', v3:'REAL', v4:'REAL',
    v5:'REAL', v6:'REAL', v7:'REAL', v8:real],
    and(has_belief(self, has_capability(p, pass_forward, v1)),
        has_belief(self, has_capability(p, pass_back, v8)),
        has_belief(self, has_capability(p, run_free, v2)),
        has_belief(self, has_capability(p, receive_ball, v3)),
        has_belief(self, has_capability(p, dribble, v4)),
        has_belief(self, has_capability(p, tackle, v5)),
        has_belief(self, has_capability(p, intercept_ball, v6)),
        has_belief(self, has_knowledge(player_field_position(p, defender)))),
    and(has_belief(self, trust_update(p,
        ((v1*weight_importance_of_pass_forward_defender) +
        (v2*weight_importance_of_run_free_defender) +
        (v3*weight_importance_of_receive_ball_defender) +
        (v4*weight_importance_of_dribble_defender) +
```
trust_update(self, self, p)), efgh(0, 0, 1, 1)).

**Identifier 5: trust_update -> trust**

This is most probably the most straightforward rule of the simulation. What happens here is that the trust_update, defined in Identifier 4, is multiplied by the importance weight. This is added to the old trust value multiplied by the weight of the importance of the former trust level. This is defined as:

\[
\text{trust_update}(\text{self}, \text{self}, p) \times \text{weight_importance_of_trust} + \text{old_trust} \times \text{weight_old_trust}.
\]

**Identifier 6a: belief -> desire_update (without emotion)**

The line described below is used for updating the desires. This code is executed when an agent has the ball, which is in the same team as ‘self’ but is not self, thus a teammate. As can be seen, three desires are to be updated. Here the aimed values are given hard coded. This is done merely for testing the model’s links.

\[
\text{desire_update}(\text{self}, \text{ball_in_possession}, 0.4), \text{desire_update}(\text{self}, \text{ball_nearby_goal}, 0.6), \text{desire_update}(\text{self}, \text{not_to_be_tackled}, 0.7).
\]

**Identifier 7: desire_update -> desire**

The entire code for updating the desire value is:

\[
\text{desire_update}(\text{self}, \text{goal}, \text{v}) \times \text{weight_old_desire} + \text{new_desire} \times \text{weight_new_desire}.
\]
updated desire value, again, multiplied by the corresponding weight. Notice, as argued before, that the sum of all the weights in one rule always adds up to 1.

**Identifier 8a: belief & desire -> update intention (without emotion)**

Although the next line of code may look overwhelming, what happens here is pretty straightforward. In the antecedent, at first, all the corresponding values for the desires are collected (v2, v12, ..., v62). Then the values for how much a certain action 'i' contributes to finalizing a desire (v1, v11, ..., v61). Since in this simulation model the agent only thinks about itself, the probability that agent 'self' can perform the action 'i' is value: 'v'. That is, an intention to perform an action can be created, but (for instance) when an agent passes the ball forwards, but nobody is there, the action is executed but failed. The agent hereafter looks at its skill level for the action that it wants to execute (v3). And the final value is the trust the agent has in itself (v4). Notice that this line of code is executed for each possible i:action, which are all the actions that are defined as sort (i.e. sortdef(action, []).)

The consequent handles the update intention value. This is calculated by taken all the relevant values, five for each argument, of which trust is counted twice for its importance. After added these values, they are divided by six.

```plaintext
  and(has_desire(self, win_game, v2),
    has_desire(self, ball_in_possession, v12),
    has_desire(self, ball_nearby_goal, v22),
    has_desire(self, give_score_chance, v32),
    has_desire(self, not_to_be_tackled, v42),
    has_desire(self, score_other_goal, v52),
    has_desire(self, not_score_own_goal, v62),
    has_belief(self, can_perform_action(self, i, v)),
    has_belief(self, desire_can_be_finalized_by(i, win_game, v1)),
    has_belief(self, desire_can_be_finalized_by(i, ball_in_possession, v11)),
    has_belief(self, desire_can_be_finalized_by(i, ball_nearby_goal, v21)),
    has_belief(self, desire_can_be_finalized_by(i, give_score_chance, v31)),
    has_belief(self, desire_can_be_finalized_by(i, not_to_be_tackled, v41)),
    has_belief(self, desire_can_be_finalized_by(i, score_other_goal, v51)),
    has_belief(self, desire_can_be_finalized_by(i, not_score_own_goal, v61)),
    has_belief(self, has_capability(self, i, v3)),
    has_belief(self, has_trust(self, v4))),
  and(update_intention(self, self, win_game, i, ((v*weight_can_perform_action)+
    (v1*weight_desire_can_be_finalized_by)+(v2*weight_desire_importance)+
    (v3*weight_has_capability)+(v4*weight_has_trust))),
  update_intention(self, self, ball_in_possession, i, ((v*weight_can_perform_action)+
    (v11*weight_desire_can_be_finalized_by)+(v12*weight_desire_importance)+
    (v3*weight_has_capability)+(v4*weight_has_trust))),
  update_intention(self, self, ball_nearby_goal, i, ((v*weight_can_perform_action)+(v21*weight_desire_can_be_finalized_by)+
    (v2*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust))),
  update_intention(self, self, give_score_chance, i, ((v*weight_can_perform_action)+(v31*weight_desire_can_be_finalized_by)+
    (v32*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust))),
  update_intention(self, self, not_to_be_tackled, i,
```
{(v*weight_can_perform_action)+(v41*weight_desire_can_be_finalized_by)+
(v42*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)),
update_intention(self, self, score_other_goal, i,
{(v*weight_can_perform_action)+(v51*weight_desire_can_be_finalized_by)+
(v52*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust))),
update_intention(self, self, not_score_own_goal, i,
{(v*weight_can_perform_action)+(v61*weight_desire_can_be_finalized_by)+
(v62*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust))},
efgh(0,0,1,1)).

Identifier 9: update intention -> intention
The number of lines of code here is very limited. Yet, the number of generated results is extensive. What
happens here is that all the update_intention lines that have been generated are being transferred in
the higher_than(x) function. The second rule depicted below is the higher_than function. This is a so-
called forall rule that looks at all the lines that have the following properties: update_intention(self, self,
g:goal, action, real). Here ‘self’ is the player, action is an action, real is the value. The forall rule looks at
all the different ‘g:goal’ and selects what goal has the biggest value. That rule is then transferred to the
has_intention argument.

leadsto([self:player, g:goal, g1:goal, x:real, y:real, ac:action],
and(update_intention(self, self, g, ac, x),
update_intention(self, self, g1, ac, y), x>=y),
higher_than(update_intention(self, self, g, ac, x),
update_intention(self, self, g1, ac)), efgh(0,0,1,1)).

leadsto([g:goal, self:player, x:real, ac:action], //higher than function
forall([g1:goal],
higher_than(update_intention(self, self, g, ac, x),
update_intention(self, self, g1, ac))),
has_intention(self, self, ac, x),efgh(0,0,1,1)).

Identifier 10: intention -> selected intention
The rule described here is equal to the first rule in Identifier 9. The only difference is that ‘goal’ is no
longer taken into account, because the goal containing the highest value was already chosen. So this
rule gives the has_intention information to the higher_than2 function.

leadsto([self:player, i:action, il:action, v:real, vl:real],
and(has_intention(self, self, i, v), has_intention(self, self, il, vl), v>=vl),
higher_than2(has_intention(self, i, v), has_intention(self, il)),
efgh(0,0,1,1)).

This rule is directly accessed by the last one and works in a similar fashion as the second rule in Identifier
9. The only difference is, that here the forall function goes over the action argument instead of the goal,
as was the case in I9. The purpose of this rule is to choose the best action, that is, the action with the
highest probability of satisfying the desires.

leadsto([self:player, il:action, v:real],
forall([ac:action],
higher_than2(has_intention(self, il, v), has_intention(self, ac))),
has_selected_intention(self, self, il, v), efgh(0,0,1,1)).
**Identifier 11: belief & selected intention -> output**

The final line of code, at least in the software without emotion, is towards the output function. This rule uses the selected_intention, which is the intention containing the highest expectation value (so the one with the biggest gain). But the action chosen within this rule still needs to be executed. This is done as soon as the agent has the belief that the action can be executed, as is outlined below:

$$\text{leadsto}(\text{[self:player, ac:action, v:'REAL', v1:'REAL]},$$
$$\quad \text{and( has_belief(self, can_perform_action(self, ac, v)),}$$
$$\quad \text{has_selected_intention(self, self, ac, v1) ),}$$
$$\quad \text{action_output(self, ac), efgh(0,0,1,1)).}$$

### 6.2 Model with emotion

In this section, the updated eBDI model will be described, starting with information on emotion and mood, followed by the declaration, initialization, and persistency rules. Hereafter the rules necessary for updating mood and emotion are discussed. Then the updated versions of the rules described earlier in the without emotion section. In the conceptual model four different basic emotions have been described, but for simplicity reasons only a gradation in the emotion happiness has been modeled. Meaning that an agent can be happy or sad, or in between, but not one of the other emotions. (Notice, that when scrolling through the source code, that the rules purely for emotion are location at the bottom). The persistency and update rules, for emotion and mood, are handled in the same fashion as described before.

**Identifier 12: emotion -> mood update**

This code updates the mood. The emotions an agent has with respect to all the players are evaluated to give a new mood as an average of the emotions for the players. So, if the emotions for the players are, respectively 0.5, 0.7, and 0.9, then the mood update is 0.7 (given that the weights over the emotions are the same).

$$\quad \text{and(has_emotion_on_player(self, playerX, e, v1),}$$
$$\quad \text{has_emotion_on_player(self, playerY, e, v2),}$$
$$\quad \text{has_emotion_on_player(self, playerZ, e, v3)),}$$
$$\quad \text{and(has_mood_update(self, e, ((v1*weight_emotion_playerX)+}$$
$$\quad \text{(v2*weight_emotion_playerY)+ (v3*weight_emotion_playerZ))),}$$
$$\quad \text{has_mood_update(self, e)), efgh(0,0,1,1)).}$$

**Identifier 13: output -> emotion update part 1**

This emotion update rule is based on the performed action from some player a, where the current player is (as always) ‘self’. Player ‘a’ might as well be the same agent as ‘self’ so that the self agent is able to adjust its emotion towards all available agents. Notice that this is only the first part of the updating rule, which executed a lot of code including all the terms in the sort ‘goal’:

$$\text{leadsto}(\text{[self:player, a:player, action:action, goal:goal, value:real, value2:real]},$$
$$\quad \text{and(action_output(a, action),}$$
$$\quad \text{has_belief(self, desire_can_be_finalized_by(action, goal, value)),}$$
$$\quad \text{has_desire(self, goal, value2)),}$$
$$\quad \text{update2_emotion_on_player(self, a, happiness, goal,}$$

$$\quad (v1*weight_emotion_playerX)+ (v2*weight_emotion_playerY)+ (v3*weight_emotion_playerZ))),$$
$$\quad \text{has_mood_update(self, e)), efgh(0,0,1,1)).}$$
Identifier 14: emotion update part 1 -> emotion update part 2

There are three different versions written of this rule, one for each player_field_position (defender, midfielder, and attacker). The printed one below is the emotion update for if the agent is a defender. Here all the temporary emotion updates for all the different goals are multiplied by their corresponding importance/weight value. As an example, since a goal for a defender is mainly to protect its own goal and to provide chances for its teammates to score, the goals as not_score_own_goal is far more important than score_other_goal. Therefore, the skills level of the defender of shooting at goal is not as important as intercepting the ball, so the emotion an agent has over another agent has to do with how good the agent performs the tasks that are important for its current location.

    and(update2_emotion_on_player(self, a, happiness, win_game, v1),
    update2_emotion_on_player(self, a, happiness, ball_in POSsession, v2),
    update2_emotion_on_player(self, a, happiness, ball_nearby_goal, v3),
    update2_emotion_on_player(self, a, happiness, give_score_chance, v4),
    update2_emotion_on_player(self, a, happiness, not_to_be_tackled, v5),
    update2_emotion_on_player(self, a, happiness, score_other_goal, v6),
    has_belief(self, has_knowledge(player_field_position(a, defender))),
    update2_emotion_on_player(self, a, happiness, not_score_own_goal, v7)),
    and(update_emotion_on_player(self, a, happiness, 
        (v1*weight_importance_of_win_game_defender)+
        (v2*weight_importance_of_ball_in_POSsession_defender)+
        (v3*weight_importance_of_ball_nearby_goal_defender)+
        (v4*weight_importance_of_give_score_chance_defender)+
        (v5*weight_importance_of_not_to_be_tackled_defender)+
        (v6*weight_importance_of_score_other_goal_defender)+
        (v7*weight_importance_of_not_score_own_goal_defender)),
    update_emotion_on_player(self, a, happiness)), efgh(0,0,1,1)).

Identifier 3b: specific event -> capability update (with emotion)

In this version of the rule emotion plays a crucial role in adjusting the capability level. In this implementation there is only a representation of an emotion about an agent, either the agent itself or on other agents. There is no implementation of emotions on incoming beliefs (e.g. observation results).

Here you can see that the antecedent of this line of code is the same as in I3a. But there are two more lines, one containing the mood the current/self agent has, and one containing the emotion agent self has on the agent 'self' is thinking about. The other differences can be found in the consequent. Here the values for mood and emotion are taken into account for influencing the new capability value.

leadsto([self:player, p:all_players, v:'REAL', eV:real, eV2:real],
    and(has_belief(self, player_ball_action_event(shoot_at_goal))),
    has_belief(self, game_event(ball_in_ possession(p))),
    has_mood(self, happiness, eV), // new
    has_emotion_on_player(self, p, happiness, eV2), // new
    has_belief(self, game_event(ball_is_in_goal(p, no_foul))),
    has_belief(self, has_capability(p, shoot_at_goal, v)),
    and(new_capability_update(self, p, shoot_at_goal, (v+(((1-v) /
personality_weight)*(eV2*eV))

capability_update(self, p, shoot_at_goal)), efgh(0,0,1,1)).

Identifier 6b: belief -> desire_update (with emotion)

Here the emotion and mood values are used together with the weight parameters. Again, just as in Identifier 3b, there are not a lot of differences in the antecedent, except for two more lines to be able to include emotion and mood to the consequent (all the values that are used in the consequent are required to be in the antecedent as well). But instead of the fixed values in the desire updates as seen in Identifier 6a, here the values can be changed. This is dependent on the current agent’s mood and the emotion the current agent has on the other agent.

leadsto([self:player, a:all_players, t:team, eV:real, eV2:real],
   and(has_belief(self, game_event(ball_in_possession(a))),
       has_belief(self, has_knowledge(is_in_team(a,t))),
       has_belief(self, has_knowledge(is_in_team(self,t))),
       has_emotion_on_player(self, a, happiness, eV),
       has_mood(self, happiness, eV2),
       a\=self),
   and(desire_update(self, ball_in_possession, (0.4*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, ball_nearby_goal, (0.6*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, not_to_be_tackled, (0.7*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, ball_in_possession),
       desire_update(self, ball_nearby_goal),
       desire_update(self, not_to_be_tackled)), efgh(0,0,1,1)).

Identifier 8b: belief & desire -> update intention (with emotion)

Emotion is added to the source below, but here is an exception compared to the other rules where emotions were added. Meaning that, the only emotion that is modeled here is ‘mood’, no emotions over players are used here. The reason for this is that only the mood of the current agent is needed to choose what kind of intention is of importance. For instance, when the agent is sad, then the desires it has are of less importance, simply because it does not care about them, and the agent rather does nothing. On the other hand, when the agent is happy, then the agent wants to strive towards satisfying its desires, so that it wants to perform the best it can. In the consequent the weight parameters are assigned to the corresponding value functions. This makes the code harder to comprehend, at first glance, but when analyzing the value update better, there is not much difference from the old intention updater, Identifier 8a.

   and(has_desire(self, win_game, v2),
       has_desire(self, ball_in_possession, v12),
       has_desire(self, ball_nearby_goal, v22),
       has_desire(self, give_score_chance, v32),
       has_desire(self, not_to_be_tackled, v42),
       has_desire(self, score_other_goal, v52),
       has_desire(self, not_score_own_goal, v62),
       a\=self),
   and(desire_update(self, ball_in_possession, (0.4*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, ball_nearby_goal, (0.6*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, not_to_be_tackled, (0.7*weight_belief_importance)+
         (eV*weight_mood)+(eV2*weight_emotion_on_player)),
       desire_update(self, ball_in_possession),
       desire_update(self, ball_nearby_goal),
       desire_update(self, not_to_be_tackled)), efgh(0,0,1,1)).
has_belief(self, can_perform_action(self, i, v)),
has_belief(self, desire_can_be_finalized_by(i, win_game, v1)),
has_belief(self, desire_can_be_finalized_by(i, ball_in_possession, v11)),
has_belief(self, desire_can_be_finalized_by(i, ball_nearby_goal, v21)),
has_belief(self, desire_can_be_finalized_by(i, give_score_chance, v31)),
has_belief(self, desire_can_be_finalized_by(i, not_to_be_tackled, v41)),
has_belief(self, desire_can_be_finalized_by(i, score_other_goal, v51)),
has_belief(self, desire_can_be_finalized_by(i, not_score_own_goal, v61)),
has_belief(self, has_capability(self, i, v3)),
has_belief(self, has_trust(self, v4)),
has_mood(self, happiness, eV)),
and( update_intention(self, self, win_game, i, ((v*weight_can_perform_action)+
(v1*weight_desire_can_be_finalized_by)+(v2*weight_desire_importance)+
(v3*weight_has_capability)+(v4*weight_has_trust)+(eV*weight_has_mood))),
update_intention(self, self, ball_in_possession, i,
((v*weight_can_perform_action)+ (v11*weight_desire_can_be_finalized_by)+
(v12*weight_desire_importance) + (v3*weight_has_capability) + (v4*weight_has_trust)+
(eV*weight_has_mood))),
update_intention(self, self, ball_nearby_goal, i,
((v*weight_can_perform_action)+(v21*weight_desire_can_be_finalized_by)+
(v2*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)+
(eV*weight_has_mood))),
update_intention(self, self, give_score_chance, i,
((v*weight_can_perform_action)+(v31*weight_desire_can_be_finalized_by)+
(v32*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)+
(eV*weight_has_mood))),
update_intention(self, self, not_to_be_tackled, i,
((v*weight_can_perform_action)+(v41*weight_desire_can_be_finalized_by)+
(v42*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)+
(eV*weight_has_mood))),
update_intention(self, self, score_other_goal, i,
((v*weight_can_perform_action)+(v51*weight_desire_can_be_finalized_by)+
(v52*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)+
(eV*weight_has_mood))),
update_intention(self, self, not_score_own_goal, i,
((v*weight_can_perform_action)+(v61*weight_desire_can_be_finalized_by)+
(v62*weight_desire_importance)+(v3*weight_has_capability)+(v4*weight_has_trust)+
(eV*weight_has_mood)))),efgh(0,0,1,1)).

6.3 Scenarios
A total of four scenarios have been written to test the model previously described. Two scenarios have been designed to generate either more positive or more negative events. The other two are more or less neutral, in the sense that positive as well as negative events occur. In one of those scenarios the examined team wins, and loses in the other one. The purpose of these scenarios is to test the implemented model in LeadsTo with and without emotions, and to examine the differences in the results. Usually, if you happen to be watching virtual agents, you will only be able to examine the observations and the behaviour by the agents, reasoning is considered to be a ‘black box’. That is why the first comparison between the simulations results are the action_outputs. Most of the parameters used, like emotions and mood, are based on so-called weight parameters. It is, therefore, necessary to investigate how the results interact with those parameters.
6.3.1 Examining action_outputs

One of the expected results is a change in performed actions (i.e. action_output). To investigate the differences in performed action, a few scenarios are compared. Notable though, for both models, there is a delay between input (observation) and output (action_output) of more than 3 time points, depending on the path the information has to travel. From the belief to output is shorter than from belief to capability, trust, etc. This makes the analysis of the results more difficult, thus the results shown here can only be used as an indication.

We are starting off to compare the third scenario, which is designed to be positive, and look at the differences in performed outputs. Figure 7 displays two figures which are showing the action_outputs for three players, playerX, playerY, playerZ (all in the same team), where playerX is an attacker, playerY is defender, and playerZ is a midfielder.

![Figure 7: scenario 3, without emotions](image1)
![Figure 7: scenario 3, with emotions](image2)

As can be seen from the terms action_output, there is a (slight) difference in performed actions. Note that the differences should have been bigger when the original SI (Selected_Intention) function should have been used. Since in the original SI function a random function chooses the performed action, based on the values generated for the Intention. The higher the intention value, the bigger the chance of being selected. In this example, the highest value of the Intention is selected. Differences between Figure 7: without and with emotions can be found between the action_outputs of the players. One of the most noticeable differences is that the performed action results from the without emotion model change faster. Scenario 4 displayed in Figure 8 shows a reverse effect.

They output different actions, although the all information they gather, via observation, is the same (i.e. the same scenarios). So, the only difference is the emotion module. To examine why this generates differences, we have to take a look at the underlying structure, see Figure 1, 2, and 3.

To see why both models output different actions, we are going to take a look at time point 20-21 of playerX. The agent performs the action ‘tackle’, in the simulation without emotion, and action ‘run_free’ in the simulation with emotions. The simulation with emotion will be identified as “s+”, whereas the simulation without emotion is “s−”. Since LeadsTo takes one time step for each action, we have to look back for what information has been used.
Since action output is generated at time point 20-21, when looking back, we know; 
has_selected_intention is generated at time point 18-20 (took 2 time point to make, because of the 
forall rule). has_intention is generated at time point 16-18 (also, took 2 time point to make). Thus the 
information used to generate the output, from intention, comes from time point 15-16, and uses beliefs, 
desires, and in case of s+ also mood. To generate intentions, from the beliefs module, four atoms are 
used; desire_can_be_finalized_by, can_perform_action, trust, and capability. Although, 
desire_can_be_finalized_by is the belief the agent has, how much an action contributes to a desire, this 
attribute has a constant value, thus has nothing to do with the changing value of the action_output.

<table>
<thead>
<tr>
<th>Without emotions</th>
<th>With emotions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tp 19-20: has_selected_intention(tackle, 0.583603)</td>
<td>Tp 19-20: has_selected_intention(run_free, 0.580559)</td>
</tr>
<tr>
<td>Tp 17-18: has_intention(run_free, 0.550143)</td>
<td>Tp 17-18: has_intention(run_free, 0.580559)</td>
</tr>
<tr>
<td>Tp 17-18: has_intention(tackle, 0.583603)</td>
<td>Tp 17-18: has_intention(tackle, 0.563069)</td>
</tr>
</tbody>
</table>

Table 2: with and without emotions

In Table 2 it is apparent that the intentions of run_free and tackle have different strengths. This is due to 
desires, which are changed by the mood the agent has, the belief that the agent can performs the 
desired actions, and the capability/skill level the agent has for that specific action. The skill level is also, 
partly, based on the mood the agent has on the moment a previous action has been undertaken, and 
the result of the action (failed or succeeded). In this case, the agent’s mood does not contribute to the 
intention generation. In this simulation, the intention with the highest value is always chosen, so if we 
have three intentions, for example, with values 0.4, 0.5 and 0.6 and they are multiplied by a value, for 
example 1.2, the former highest value will always be the highest value, respectively 0.48, 0.6, 0.72.

To conclude, since playerX has a positive emotion towards his teammates, he would expect the 
other players to retrieve the ball, and therefore starts to run free.
6.3.2 Examining emotions

Take a look at Figure 9 and Figure 10 to start on examining the working of the emotions. Both Figures are executed using the second scenario. Figure 10 lists the action_outputs, so that these can be easily compared with the updated emotions, of Figure 9. In Figure 9 the emotions for the agents have been shown which are generated in the second scenario. Hereby it is important to notice the values written on the y-axis, because the steps are dependent on the minimum and maximum of the values’ atoms. Examining emotions is more straightforward than the action_outputs, at least in the modeled case, because the only dependencies of emotions are based on beliefs, desires, and performed actions. This is in contrast with action_output, because action_output is based on the entire model.

According to the conceptual model, emotions are updated depending on the performed action the agent has compared with the desires the agent has and the degree of contribution of that action to the desired goal. One obstacle of this model in LeadsTo, is that the scenario is predefined, meaning that the effects of the agents’ actions on the world are not modeled. Thus the output_actions are, most probably, not the same actions as the observation results of the agents. For example, an agent can decide to shoot at the goal, even when the scenario shows that agent is passing the ball to another agent.

To show exactly what happens, we have to look back to previous atoms, since emotions are generated based on desire, beliefs, and actions. Identifier 13 shows that the action_output is used for updating emotions, together with the belief that that action contributes to a certain desire, and the desire itself. Identifier 13 is used for each desire. The contribution of the performed action is calculated for each desire, by multiplying the value of which the action contributes to a goal, plus the value of the desire itself multiplied by its corresponding importance/weight (i.e. \((desire\_can\_be\_finalized\_value * weight\_desire\_can\_be\_finalized) + (desire\_value * weight\_desire\_value))\). Thus, if there happen to be three desires, for instance, win_game, score_other_goal, and ball_possession, then Identifier 13 will be fired three times. For example, the performed action is shoot_at_goal. How much does shoot_at_goal contribute to finalizing the goal win_game, plus the importance of the desire win_game. How much does shoot_at_goal contribute to the goal score_other_goal. And so forth.
After the first emotion update has been calculated, (Identifier 14) the weights over all those desires are taken, to calculate the updated emotion value. Thus to follow up on the example given before, the values calculated per desire are then summed up (each multiplied with its corresponding weight), to get to the updated emotion. The corresponding weight corresponds to each desire. This is linked with the sort player the current agent is thinking about, because particular goals are less important for defenders than for attackers (e.g. score_other_goal is more important for an attacker than for a defender).

When we take a look at Figure 9, the left hand side, we see the emotions playerX has, on playerX (self), playerY, and playerZ. Notice that there is a delay of three time steps between action_output and the updated emotion, because Identifier 13 and 14 have to be executed, as well as the update rule. The update rule is the old emotion multiplied by the importance weight of the old emotion plus the updated emotion multiplied by its importance weight.

So, take, for example TP 19-20 of action_output of playerZ. PlayerZ wants to perform the action pass_forward. As explained before, it takes three time steps to generate an emotion about the action that is performed. So we will see a difference in emotion at time point 22-23. The emotion that playerX has for playerY is close to 0.52 at time point 21-22, so 1 time point before the emotion update for the action pass_forward. At time point 22-23 we see a rapid increase in happiness, from 0.52 to 0.59. Why?

PlayerY is a midfielder, one of the major desires (meaning important goals) is to give another player a chance to score. If playerY passes the ball forward and thereby giving playerX (the attacker) a chance to score, playerY will be happier. Because doing that, he will be fulfilling a major role of its existence, his goal. Thus when an agent thinks he is performing actions that contribute to one of his important goals, he will be more satisfied and therefore happier.

6.4 Final conclusion simulation results
As shown, in the results reported above, the outputs from the rational model are different than those from the model with emotions. Emotions are updated, in this LeadsTo model, according to the actions of the agents. When the action contributes more to the goal the agent has, the more positive the updated emotion is. Thus when the agent performs actions that result in positive experiences, the agent will be happier. A positive experience is an event that fulfills, or strive to, a goal (or sub-goal), although, in real-life, the goals are not always that obvious.

Notably, though, the conceptual model also says that emotions are updated according to expectations the agent has, this has not been modeled in this simulation. Importantly, the missing aspect in updating emotions, the agent’s expectations of the next event, will contribute more to positive, as well as negative, emotions. This, unfortunately, was not an option in LeadsTo, because of the missing link between performed action and observation results. To test the entire model more extensively, it has been implemented in a real-time soccer simulation program called RoboCup. This will be discussed next.
7.0 RoboCup: The implementation of the eBDI model

The RoboCup soccer simulation 2D software can be divided into three sub programs; the RoboCup Soccer Server (rcssserver), the monitor (rcssmonitor), and the log player (rcsslogplayer). The rcssserver is the main playing field, where all the interaction and calculations takes place. This program prints all actions to an rcsslogplayer. The rcsslogplayer can load the log file making it possible to replay the game over and over again. The rcssmonitor is meant to make the game graphical to the spectator at the same time as the game is being played. Here it is also possible to manipulate the game, for instance by means of placing the ball in another location of the field.

Before implementing the model described before reusable team code had to be found. Since writing an operatable team took many teams several years and RoboCup, in this project, is primarily used to display the working of the decision model, an existing team code was reused. Fortunately, for the sake of this project, a few teams have made their source codes public. It could, for the purpose of choosing actions, be best to choose the most recently available team code. Because these teams would, probably, have more sophisticated information ready to be used by the eBDI model. WrightEagle, a team from China, was one of those teams that put there code online. Furthermore, they were the winning team in 2009. Yet, there are problems when using source code of teams that are too new. Therefore, it is important to know how the tournament itself is performed.

Each match, performed in tournament setting, will be executed on nine state-of-the-art computers. Each team gets four computers for their team’s decision model and one computer is used to perform alterations for the rcssserver. Each computer will then calculate four processes; each process is either one player or the coach. The performance of the computers is important, because each player only gets 100 ms to response. If the respond takes longer than 100 ms, no action will be executed. So each time step takes 100 ms and for this project only one computer was available. This computer is comparable to one of the nine computers used in the tournament that WrightEagle (version 2009) was playing in. It is, therefore, for the CPU-time needed to calculate the actions, better to use an older version of the rcssserver.

Fortunately, a team from Germany, The Brainstormers, recently made their 2005 team code public. However, there are also downsides to using such an older version. This has to do with the development of the RoboCup’s server. Since RoboCup is solely developed for the purpose of research the development of RoboCup is an ongoing project. The version of the Brainstormers (2005) was written for server version 11.x or earlier because version 12 had major upgrades, including a new log file system such that earlier team codes were not operatable anymore. One of the upgrades of rcssserver 12.x was team graphics such as stamina amount as a grey gradient for each player, and the displaying of cards the players have been given. The rcssserver versions before 12 had no foul, except for offline, included.

The publicly released source code was a demo version, meaning that the code responsible for winning the 2005’s tournament was replaced by three demos to show more basic behavioural types; with ball, without ball, and score behaviour. The reason for this is that it would, in the other case, be too easy to make a good team. Simply by adjusting a few lines of code a winning team could be created. Nonetheless, because the Brainstormer’s code is based more on strategy, instead of calling individual
actions, there would not have been an implementation difference between the original and the demo version of the team code. It was, also, never the goal to produce a tournament winning team. Instead, the purpose was to create a more human-like decision model.

The Brainstormers first created action classes (e.g. intercept_ball) by hand. After constructing these actions, they have tried to improve the actions by using Reinforcement Learning and Neural Networks. When the actions were improved by these learning operators, they used the generated classes. For each action, the best version was then chosen and used in their original decision making. Furthermore, decisions were made depending on team performance without the intention to reproduce human decision making.

Instead, this is exactly the purpose of this project. It is not about making a good or winning team, but for individual players to make understandable decisions where each action is unrelated to the action performed before. So, no team strategies are used. Exactly how the actions are chosen will be discussed later. Nonetheless, to prove the added value of emotions for making decisions, RoboCup is used, why?

The purpose, of RoboCup, is to create an environment that reproduces the reality, but just in two-dimensions. Playing soccer is primarily used in this project to give the players a purpose of interacting with others. Since mental states as, for instance, trust have to be based on something that happened before. Therefore, it is important to measure the action that is performed, how good is it performed?

7.1 Implementation in RoboCup
The implementation is based on the way the conceptual model has been implemented in LeadsTo. Therefore, the mental states and their interactions needed to be created first (i.e. Beliefs, Desires, Intentions, Selected_Intention, and Emotions). The most vital difference between the decision models in LeadsTo compared to RoboCup version appears after the intentions are created. In LeadsTo, the intention with the highest expectation value was chosen. In RoboCup, on the other hand, a probabilistic mechanism is used: the bigger the intention’s expectation value, the bigger the chance of that action to be selected. Furthermore, some formulae have been adjusted. This will be discussed more extensively later.

In the RoboCup version the model is divided into two parts. The first part, including the Beliefs, Desires, and Emotion states is called the information module. Whereas, the second part including the Intention and Selected_Intention states are together called the processing module. This processing module is purely for making decisions. But to make a decision, information is needed. Therefore, vital information will be recollected from the information module. The main reason for this division has to do with the decision file from the Brainstormers source code, BS03_bmc.c. This file is, together with client.c, the main decision part of the Brainstormer’s model. Here choices are made on which action the player will perform.
The first part of this file is used for collecting information for the eBDI model (information module). The second part is used for choosing an action. Here a division is made, by using a switch-statement which looks at the current play-mode. The value of that variable incorporates current information on the state of the game (e.g. goal_kick, play_on). Information on the current state needs to be updated each time step (i.e. each 100 ms). But decisions, from the eBDI model, are only made when the current play_mode is PM_PlayOn. In other cases the Brainstormers decision model takes over. The other play-modes have not been adjusted because the added value compared to the time needed to change this is not in proportion.

The information needed for the eBDI decision model, such as player and ball location, has to be collected from the existing Brainstormers source, instead of extracting information from the server directly. This would have made development noticeably longer. The same holds for most of the actions the agents have to perform. Most of the actions the eBDI model uses are pre-programmed actions written by the Brainstormers team. The only action that had to be created was the tackle function.

The information subtracted from the Brainstormers code comes mainly from three files, ws.x, ws_info.x, and ws_memory.x. Player specific information and ball information (e.g. position, velocity) are stored in ws.x. Current information on, for instance, who has ball position is located in ws_info.x. Information from the past, like what was the last player in ball possession, is stored in ws_memory.x.

### 7.2 Basic eBDI model

The basic components of the eBDI model are comparable to the conceptual model described before. Therefore the parts that are identical or very similar to the model described in LeadsTo will not be described. The main differences are the separation between the information (see Figure 11) and processing module (Figure 12). Some other differences from the current eBDI model with respect to the LeadsTo model have to do with different formulae and specific information needed to let the agents perform actions needed for a soccer player. These variables, files, and formulae will be discussed now.

### 7.3 Differences between LeadsTo and RoboCup

Two main classes with variables have been defined; goals.h (i.e. desires), and actions.h. The goals listed here are the same as in the LeadsTo version and will, therefore, not be explained here. Actions are used for calculating specific interaction with the environment (e.g. ‘go to location’, ‘kick ball towards location’). Capabilities are very much similar to actions and use the same variables, except for passing. Actions like direct_pass_to_player_2 through direct_pass_to_player_11 are set to be the same capability, namely direct_pass (i.e. number 10). This has been done because the action direct_pass_to is the same action regardless of the target player, at least for the purpose of capabilities. For the sake of performing an action, the destination player is of importance because that player should be the next player to be in ball possession. The same holds for the long_pass action (i.e. number 11). Notice, though, that some actions can only be executed when the player has ball possession (actions from ‘dribble’ through ‘long_pass_to_player_number_11’), and some are designed for when the player does not have ball possession (from ‘rest’ to ‘guarding’).
Another important file for global (reusable) variables is `model_parameters.h`. This file includes, for instance, the number of players an agent thinks about, the number of desires, and number of different actions. Like in the initialization part of the LeadsTo code, some weight parameters have been designed/ set to change certain importance factors more easily. By importance factors is meant, for instance when the expectation_value of an intention is calculated, the relative importance of desires, and the ability to execute the wished actions. The generation of intentions will be discussed in more detail later.

### 7.4 Processing structure

First the important modules used for decision making briefly will be discussed from input to output (see Figure 12 for the entire structure of the processing method). Thereafter, the five classes will be explained in more depth (can_perform_action, action_generator, intention_generator, selected_intention, and the action_handler). The standard eBDI model mainly includes the core modules
for thinking and deciding. But input and output information is needed to actually use the model. An agent needs to know, for example, whether an action can be executed (see file ‘can_perform.action.x’).

![Processing Method Diagram](image)

This is a part of the agent’s belief system, but it is apparently one that changes rapidly. When an agent, after the intention is chosen, wants to execute an action more information is needed on the type of low-level action and target vector. A target vector is the location where an action should be performed. For example, a player wants to pass the ball. Where is the current player? Where is the target player? And what is his position? So the transformation from the chosen action, `direct_pass_to_player_number_2`, to the low-level action could be, ‘kick the ball with speed X to position of player_number_2’ (see file ‘action_handler.x’).

After the execution of an action, players need to know whether the action has been succeeded, failed, or if it is still in progress. This is important because the trust and capability in the player itself and in other players need to be representative to the player to think about. Without the ‘action_expectation.x’ file, the beliefs over other players will never change.

Finally, as mentioned before, the mechanism for choosing actions has been changed, with respect to the LeadsTo model. In the LeadsTo model, the intention with the highest `expectation_value` was chosen. This would, possibly, result in the best possible actions but the version used in LeadsTo is obviously still a large simplification of decision making in reality. Meaning that all the players think about...
is what they have just seen, assuming no “life” outside the field. In real-life, decision makers base their actions on more variables (e.g. childhood experiences) which are, nearly, impossible to recreate. To simulate previous experiences and other unknown mental states we have chosen to use a more stochastic approach. This approach is, nonetheless, also based on the expectation_value but merely as a chance variable. An action with a higher expectation_value has a bigger chance of being selected. But if two actions have almost similar expectation_values, both should have almost an equal chance of being selected which is in contrast to the LeadsTo model. Therefore every action should in principle be considered. This is handled in the ‘selected_intention.x’ class. The expectation_value of an intention is generated in the ‘intention_generator.x’ class. This is based on current beliefs and desires. The classes mentioned in this paragraph, which are different from the ones implemented in LeadsTo, are explained next in more detail.

7.4.1 Can perform action
This module can be seen as a fairly low-level reasoning class. Here it will be checked whether the action that is currently under investigation can be executed at all. Actions with ball interaction cannot be executed when the player does not have the ball, and vice versa. When an action comes through this check some information is gathered, like player and ball position. A switch-statement follows, so that only relevant action information is issued. Each action has its own statement on when it can be executed. This, in some case, has to do with distance from the place where the action could be executed. For more information, look at the corresponding file ‘can_perform_action.c’.

7.4.2 Action expectation
When a player is performing an action it is important to know when that player can perform another action. With other words, when is an action completed? Has the action failed, succeeded, or might there be another reason why the player has to stop performing that again (for instance, because the ball has crossed the sideline). It is important to know whether an action has failed or succeeded for updating trust and beliefs over capabilities. Each action has different sets of rules of when it is finished. Each statement that determines that an action is finished returns a value. The range of return values is between -1 and 1. If the action is failed, the returned value is negative. The value is 0.0 when, for instance, the play-mode changed without interaction from the current player. And, finally, the value is positive if the action is succeeded.

7.4.3 Intention generator
The main method of the Intention_generator class includes two arguments; a pointer to the Model_Input (i.e. hardwired ‘brain’); and a pointer to players_intention. The players_intention pointer is an array where all the possible actions are included. Since there are only a limited number of possible actions all the actions can be thought off. The purpose of this class is to calculate the expectation value for each single action. So, what information is needed for each action? Is the current action a single player action, or is there an interaction between players (for instance, when player X passes to player Y, also information from player Y is important).
The first part of the reasoning process is to look at whether an action can be performed. Therefore a link is made with the `can_perform_action.x` class. When the action cannot be performed, because, for instance, current action is to *dribble* but the player is not in ball possession then there is no need to reason further over this action. If the chance the action can be performed is higher than 0, we will continue reasoning. Then the capability number, instead of action number, is chosen. As explained before, if the action is not a pass, then the capability number is the same as the action number. Only when the action is some sort of pass then the capability number changes with respect to the action number.

The next part is to get a value of how good the current player thinks he is in performing that action, which is based on previous experiences. Furthermore, the player needs to know how much it trusts itself. Now the information on whether the player can perform the action is present in its memory. But, does he really want to perform this? That is, does this action contribute to the desires the player has? Why should an action be performed, purely because it is possible but without added value? Thus let us take a look at the player’s desires.

A number of desires are designed. Here the desires themselves are static. That is, no new desires are created, only the value of how much the player wants to perform that desire is altered (from `desire.x`). There are three factors that contribute to the overall importance of a desire. 1) The one mentioned before, desire_value. 2) The importance of a desire (from `desire_importance.x`), for instance, the desire *keep ball possession* is of less importance than *win game*. 3) And, finally, (from `desire_can_be_finalized_by.x`) how much does the action contribute to achieving the goal? The product of these three factors gives an approximation of how much the current action can help satisfying the agent’s most important desires. Finally, the desire with the highest overall value will then contribute to the final intention’s expectation value with variable name `desire_contribution`, see the final section of this paragraph.

It is only useful to investigate the capabilities and trust for another player if there is an interaction between players. Those two factors (capability and trust) are important to think about here. The first factor is trust the current player has in a player, either self or another one. The second factor is the capability value that the current player thinks the other player has with respect to finalizing his part of the action. In the current usage of the model only the action *receive_ball* is accompanied as counteraction. So, when a playerX passes the ball to playerY, the capability that playerX thinks that playerY has in receiving the ball together with its trust in playerY are taken into consideration.

The final part of the intention_generator section is for calculating the final expectation_value based on all the information previously gathered. The function used is, the sum over all the results of each value multiplied with the corresponding weight parameter. The sum over all the corresponding weight parameters adds up to one, see `model_parameters.h` for all these parameters. Finally, the ‘expectation_value’ that was calculated is then squared and then divided by 100 (i.e. expectation_value = expectation_value * expectation_value / 100). As a result, when the expectation_value (before this function) is 100, it stays 100, but if it were, for instance, 50 then the new result will be 25 (since, 50 * (50/100) is 25). This last function results in a stronger differentiation between ‘good’ and ‘bad’ actions.
which implies that better actions (i.e. intention with higher ‘expectation_values’) are more likely to be chosen than the actions that contribute less to, for instance, desires. When all the actions have been considered, this class returns the variable to the processing module.

7.4.4 Selected Intention
As indicated before, the selected intention method has been changed in comparison with the one used in LeadsTo. The main reason for this difference is the decision to use random values. Additional information on the random function can be found later on. The main purpose of this class is to choose an action that will be executed later. This class is divided into two for-loops. The first is to sum up all the expectation_values of the intentions. The result of this sum is then used to decide the range of the random function. Then a random number is generated. This number will then be compared in the second for-loop where the corresponding action is set and the for-loop is interrupted. This action is thereafter collected by the model_output which calls the action_handler's method to get the information needed for the RoboCup server to perform that action.

7.4.4.1 Random function
Getting a random value is harder than it seems, especially since each programming language has its own vision of providing pseudo-random functions. For instance, Java’s function is fairly functional on its own. C#’s random function only works when the system ‘sleeps’ for at least 16 ms before the declaration of the random function. C++’s random function is less operatable than the other two languages. The best known, but pretty easy way to get a usable random number is using the system time for a seed from the random list. However, the system time C++ retrieves is per second. This is, usually, good enough, but the way it will be used in RoboCup more than one random number per second is needed. Ten agents think with a maximum of ten times a second over which action to take. Another option is to use an artificial random seed (i.e. srand(2000); ). The pseudo-random function can be seen as an array of billions of ‘random’ numbers. A seed is the identifier to a random number. But when an artificial seed is set on declaration of the agent’s brain, each agent will select, approximately, the same random number each time. This random ‘problem’ is finally resolved by making an integer variable. This variable points to a memory address. That memory address is used as the seed for the current agent. Since it is impossible to have multiple variables to point at the same memory address, unless explicitly coded, each agent will start off at a different point in the pseudo-random seed function.

7.4.5 Action handler
The purpose of this file is to provide the information necessary to perform the action previously chosen. This file is divided in a couple of methods. Each method is designed for a different action. Almost each action needs a target vector (i.e. a location on the field to perform the action at or to). This information, the target vector and type of action, together will be used by the Brainstormers main decision class (bs03_bmc.c) and will then be executed. I will, hereby, give two examples of how information for actions is chosen. Information on the direct_pass_to_player action will be discussed first and thereafter more on the tactic_tackle.
Direct_pass_to_player needs information on the current beliefs of the player. Furthermore, it needs to know who to pass to. Therefore, the first step is to locate the player to pass the ball to. To calculate the target position of the ball, also the current velocity of the other player has to be known. The final destination target to kick the ball to is the current location of the other player plus its velocity. In the case of the action long_pass_to_player the velocity of the other player is multiplied by five, if the player’s direction is towards the goal. Otherwise, the ball will be passed five meters in front of the player, that is, towards the goal.

For the tactic_tackle it is important to know which opponent currently has ball possession because that is the opponent that should be tackled. The target position will then be the position of the opponent currently in ball possession.

7.5 Simulation results
Based on the model described above, a number of simulation experiments have been performed. The parameter settings used within these experiments are shown in Appendix 2. The initial values for updating mental states are 1 divided by the number of arguments, for instance the relative contribution to update_intention is 0.5 for the old value and 0.5 for the new one. This holds for: update_mood, update_desire, intention_generator (see Appendix 2). Notice that the values presented in Appendix 2, for intention generator, are different than the original setting, since the original setting is that all the weights are of equal importance.

This paragraph is divided into two sections. The first one is meant to show how the mental states of the agents change, in different circumstances. The second part will be used to demonstrate the difference between a team with positive emotions and a team with negative emotions.

7.5.1 Development of Mental States
This paragraph gives a demonstration on the working of the eBDI model in the RoboCup environment. Here will be shown that, given the outcome of an action, the capabilities, trust, emotions and mood of the agents are changed. Noticeable, though, is that here only positive updates are referred to. The reason for this is that the effects of the negative experiences (i.e. failed actions) are less apparent than the positive ones. For instance, a few players can have the intention to perform ‘receive_ball’ at the same time, although only one can perform that action. The effects of positive experiences will be displayed in short movies that are shown on the website (still have to determine which website) and on YouTube. First the capability updates are demonstrated followed by trust, emotion and mood. In the final part of this section the desire ‘release_frustration’ will be shown. This desire is meant for the players to change the emotion on themselves positively by performing aggressive tackles. The emotion will only change positively when the aggressive tackle succeeds. When the aggressive tackle fails then the desire release_frustration will become more important, resulting in an increased desire to release their frustration. Small remark: when, in the explanation some mental state (e.g. trust) is mentioned, without mentioning a specific player, then these mental states are about what the agent feels or thinks about himself. Only paragraph 7.5.1.2 handles trust over in players. Furthermore, the player numbers, as displayed in the RoboCup simulator, are hexadecimal instead of decimal (so that player 10 is A, and player 11 is B).
7.5.1.1 Capabilities
To show the results that positive actions have on the players’ belief of their own capabilities, these updates are demonstrated in Movie 1. What happens is as follows. Player 4 intercepts the ball from the opponent and starts to dribble for a short moment of time. Then he passes the ball to player 10 who receives the ball and start dribbling towards the opponent’s goal. Then the keeper gets ball possession. Notice that the trust of player 10 increases more than player 4. This happened because player 10 increased his beliefs over its capabilities on two actions, whereas player 4 only increases one of its actions. See Table 3 for details or go for the movie to “http://www.youtube.com/watch?v=1XfnnExBag”. In the following tables five columns were used: “who” is the player whose mental state is being investigated, “what” is the mental state itself, “old value” is the mental state’s value before the action is finished, “new value” is the value after updating, and finally “why” is the reason for the value update.

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 4</td>
<td>Capability: dribble</td>
<td>0.500</td>
<td>0.549</td>
<td>Because dribble went good and the intention was changed to pass_ball</td>
</tr>
<tr>
<td>Player 4</td>
<td>Trust</td>
<td>0.488</td>
<td>0.493</td>
<td>increase because dribble succeeded and pass was received</td>
</tr>
<tr>
<td>Player 10</td>
<td>Capability: receive_ball</td>
<td>0.399</td>
<td>0.476</td>
<td>Increase because Player 10 receives the ball</td>
</tr>
<tr>
<td>Player 10</td>
<td>Capability: dribble</td>
<td>0.500</td>
<td>0.548</td>
<td>Correct dribbling</td>
</tr>
<tr>
<td>Player 10</td>
<td>Trust</td>
<td>0.472</td>
<td>0.480</td>
<td>Receive ball and dribble were good actions</td>
</tr>
</tbody>
</table>

Table 3: capability updates

7.5.1.2 Trust
As could be seen in Table 3 displayed in the section before, trust was also updated. Trust is not only present for the player itself but also for other players. To demonstrate the workings for this take a look at Movie 2 (see “http://www.youtube.com/watch?v=AdX9LPxKeI”). This movie shows that player 10 performs a good action so that it increases its self trust. Also, the trust in other players is displayed. Here can be seen that some players (players 8, 9, 11) have increased their trust in player 10, while player 7 did not see it happen and, therefore, did not increase its trust in player 10. Notice that the changes in trust are fairly small. This is because trust is the result of the sum over all the capabilities (weighed by their corresponding importance values). See Table 4 for the exact trust updates.

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 10</td>
<td>Trust in self</td>
<td>0.427</td>
<td>0.432</td>
<td>Player 10 receives the ball and increases its capability, therefore its trust in self increases</td>
</tr>
<tr>
<td>Player 7</td>
<td>Trust in player 10</td>
<td>0.548</td>
<td>0.548</td>
<td>Player 7 did not see player 10, so no increase in trust in player 10</td>
</tr>
<tr>
<td>Player 8</td>
<td>Trust in player 10</td>
<td>0.488</td>
<td>0.489</td>
<td>Player 8 saw player 10 receiving the ball, therefore increases its trust in player 10</td>
</tr>
<tr>
<td>Player 9</td>
<td>Trust in player 10</td>
<td>0.489</td>
<td>0.490</td>
<td>Player 9 saw player 10 receiving the ball, therefore increases its trust in player 10</td>
</tr>
<tr>
<td>Player 11</td>
<td>Trust in player 10</td>
<td>0.5003</td>
<td>0.5007</td>
<td>Player 11 saw player 10 receiving the ball, therefore increases its trust in player 10</td>
</tr>
</tbody>
</table>

Table 4: trust updates
7.5.1.3 Mood and Emotion_on_player

Movies 3 and 4 demonstrate the increase in mood and emotion. In Movie 3 it is shown that player 7 receives the ball and passes it to player 9. Player 9 receives the ball and passes it to player 8. Player 8 receives the ball and passes it to player 11 which receives the ball and starts dribbling when the updated parameters are displayed (see “http://www.youtube.com/watch?v=I-ZamC-louo”). This is displayed in Table 5, notice here that the first column is not sorted on name but on who performed the action first.

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 7</td>
<td>Emotion self</td>
<td>0.380</td>
<td>0.592</td>
<td>Receive ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 7</td>
<td>Mood</td>
<td>0.468</td>
<td>0.591</td>
<td>Emotion on self, emotion on player 9 increased</td>
</tr>
<tr>
<td>Player 9</td>
<td>Emotion self</td>
<td>0.661</td>
<td>0.745</td>
<td>Receive ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 9</td>
<td>Mood</td>
<td>0.589</td>
<td>0.660</td>
<td>Emotion on self, emotion on player 8 increased</td>
</tr>
<tr>
<td>Player 8</td>
<td>Emotion self</td>
<td>0.759</td>
<td>0.832</td>
<td>Receive ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 8</td>
<td>Mood</td>
<td>0.620</td>
<td>0.693</td>
<td>Emotion on self, emotion on player11 increased</td>
</tr>
<tr>
<td>Player 11</td>
<td>Emotion self</td>
<td>0.309</td>
<td>0.349</td>
<td>Receive ball succeeded (dribble is in process)</td>
</tr>
<tr>
<td>Player 11</td>
<td>Mood</td>
<td>0.431</td>
<td>0.448</td>
<td>Emotion on self increased</td>
</tr>
</tbody>
</table>

Table 5: mood and emotion update

In Movie 4 (see “http://www.youtube.com/watch?v=n-ZLuezGXGE”) the mood, emotion over self, capability of shoot_at_goal, and self trust are displayed. Player 6 kicks in to player 10. Player 10 receives the ball, dribbles a bit then shoots at the goal and scores. See Table 6 for the mental state changes.

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 10</td>
<td>Capability: Shoot_at_goal</td>
<td>0.509</td>
<td>0.602</td>
<td>He scores. Thus, his belief for his ability shoot_at_goal is increased</td>
</tr>
<tr>
<td>Player 10</td>
<td>Trust in self</td>
<td>0.522</td>
<td>0.552</td>
<td>His self trust increased because his belief in the capabilities shoot_at_goal and receive_ball increased</td>
</tr>
<tr>
<td>Player 10</td>
<td>Emotion self</td>
<td>0.366</td>
<td>0.686</td>
<td>His emotion on himself increased because he has performed an action successfully that contributes greatly to an important desire to ‘score_other_goal’</td>
</tr>
<tr>
<td>Player 10</td>
<td>Mood</td>
<td>0.398</td>
<td>0.557</td>
<td>Mood greatly increased because its self image has dramatically improved</td>
</tr>
</tbody>
</table>

Table 6: mental state changes after scoring

7.5.1.4 Desire

As mentioned before, only one desire will be updated in this version of the model. This desire is ‘release_frustration’ and its values are directly linked with either positive or negative experiences. The agent can release its frustration by performing an aggressive tackle. This is demonstrated in Movie 5 (see “http://www.youtube.com/watch?v=CUrgzSoE5I1”). This movie shows how several players perform an aggressive tackle and the result of that action on their desire to release their frustration. Players 4 and 5 both perform their tackle on player 9, from the other team. The opponent is not moving during the aggressive tackle and the action has, therefore, succeeded. Players 7 and 8 both tackle opponent player 10 which just keeps moving, implying a failed action. The desire changes are listed in Table 7.

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 7</td>
<td>Emotion self</td>
<td>0.380</td>
<td>0.592</td>
<td>Release ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 7</td>
<td>Mood</td>
<td>0.468</td>
<td>0.591</td>
<td>Emotion on self, emotion on player 9 increased</td>
</tr>
<tr>
<td>Player 9</td>
<td>Emotion self</td>
<td>0.661</td>
<td>0.745</td>
<td>Receive ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 9</td>
<td>Mood</td>
<td>0.589</td>
<td>0.660</td>
<td>Emotion on self, emotion on player 8 increased</td>
</tr>
<tr>
<td>Player 8</td>
<td>Emotion self</td>
<td>0.759</td>
<td>0.832</td>
<td>Receive ball and pass have succeeded</td>
</tr>
<tr>
<td>Player 8</td>
<td>Mood</td>
<td>0.620</td>
<td>0.693</td>
<td>Emotion on self, emotion on player11 increased</td>
</tr>
<tr>
<td>Player 11</td>
<td>Emotion self</td>
<td>0.309</td>
<td>0.349</td>
<td>Receive ball succeeded (dribble is in process)</td>
</tr>
<tr>
<td>Player 11</td>
<td>Mood</td>
<td>0.431</td>
<td>0.448</td>
<td>Emotion on self increased</td>
</tr>
</tbody>
</table>

Table 7: desire changes after aggressive tackle
### Table 7: Result of tackling on desire release_frustration

<table>
<thead>
<tr>
<th>Who</th>
<th>What</th>
<th>Old value</th>
<th>New value</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 4</td>
<td>Desire: Release_frustration</td>
<td>0.425</td>
<td>0.341</td>
<td>Tackle succeeded</td>
</tr>
<tr>
<td>Player 5</td>
<td>Desire: Release_frustration</td>
<td>0.611</td>
<td>0.542</td>
<td>Tackle succeeded</td>
</tr>
<tr>
<td>Player 7</td>
<td>Desire: Release_frustration</td>
<td>0.355</td>
<td>0.389</td>
<td>Tackle failed</td>
</tr>
<tr>
<td>Player 8</td>
<td>Desire: Release_frustration</td>
<td>0.420</td>
<td>0.474</td>
<td>Tackle failed</td>
</tr>
</tbody>
</table>

#### 7.5.2 Positive versus Negative teams

To demonstrate the importance of positive emotions with respect to decision making two teams have been produced. This has been done with the final version of the eBDI team code with as only difference that the emotions and mood are not dynamic. These values are set statically to 0.2 and 0.8, respectively for the negative and positive team. The results are described below, and will be discussed in the conclusion RoboCup section.

One aspect of the RoboCup soccer environment has to be prominent before starting with the results. RoboCup, like real soccer games, has a large amount of uncertainties. It is, therefore, impossible to reproduce an exact match for a second time. To try to cope with this massive amount of randomness, ten games were performed for each setting. The numbers of action occurrences have been printed at the end of each game. In paragraph 7.5.2.1 the differences in actions between ‘field positions’ will be investigated (i.e. defender-midfielder, defender-attacker, and midfielder-attacker). In paragraph 7.5.2.2 the differences between positive and negative emotional teams will be investigated further. In this paragraph, a couple of different settings will be described. First, the entire teams’ actions will be compared, so, for instance, positive versus negative with respect to ‘pass’. Second, the actions will be compared over the player’s field position (e.g. negative defender pass versus positive defender pass).

#### 7.5.2.1 Within team action difference

Here the results on differences between field positions will be described. Fifteen different paired sampled T-tests were produced for both the positive and negative teams, for each action and for each field position, respectively five (pass, dribble, shoot_at_goal, aggressive_tackle, and tactic_tackle) and three (defender, midfielder, and attacker). An example for this is the comparison between defender and midfielder (from the same team) with respect to the amount of times they perform an aggressive_tackle. Since the results of both the positive and the negative teams were alike, only the positive team results will be discussed. The results for the positive team produced significant differences on 7 of the 15 tests (the significance level is set at p=.05).

To give a closer look at the differences, first there were some significant values on the action ‘pass’; defender – midfielder (p=.495), defender – attacker (p=.000), midfielder – attacker (p=.002). The dribble results showed highly significant differences; defender – midfielder (p=.001), defender – attacker (p=.000), midfielder – attacker (p=.001). Shoot_at_goal showed no significant differences; defender – midfielder (p=.233), defender – attacker (p=.065) and midfielder – attacker (p=.657). Aggressive_tackle
was significant for both defender – midfielder (p=.000) and midfielder – attacker (p=.000), but not for defender – attacker (p=.730). Finally, no differences were found for tactic_tackles; defender – midfielder (p=.576), defender – attacker (p=.764), and midfielder – attacker (p=.455).

7.5.2.2 Between team action difference
To compare differences between the positive and negative emotional teams, paired sampled T-tests were performed over the actions. Five tests were performed, of which two gave significant results between team differences; pass p=.039, and aggressive_tackle p=.000. The other results were; shoot_at_goal p=.280, tactic_tackle p=.467, and dribble p=.516. To investigate the significant values more closely, let’s take a look at the passing differences. The mean of pos_team_pass (4.6) is higher than the neg_team_pass (3.88), where the mean indicates the number of time a player performs that action per match. The opposite is shown for aggressive_tackles, where the pos_team showed a mean of 9.83 tackles per player per game, the neg_team showed an average of 21.68 tackles. To explore the differences for performing actions more closely, the results will now be shown over the players’ field positions. Instead of comparing whole team actions, the results of the players’ field position will be compared next, such as: the action ‘pass’ of the player field position ‘defenders’ from the negative emotional team versus the action ‘pass’ of the player field position ‘defenders’ of the positive emotional team.

The results showed significant differences on 4 of the 15 tests. As was demonstrated before, passing and aggressive_tackles were significantly different among positive and negative teams. All the aggressive_tackles (defender, midfielder, attacker) showed significance values of p=.000, whereas the action ‘pass’ only showed a significant difference for defenders p=.037. Test values for passing for the other field positions were; midfielder p=.961, and attacker p=.182. None of the other actions showed significant differences. Now that the significant differences are shown, let’s look deeper into the values itself, that is, the means. First the action ‘pass’ for the defender, the positive team showed a mean of 5.65, whereas the negative emotional team, also for defenders, showed a mean value of 4.375. Secondly, the aggressive_tackles were significant for all the players’ field position. Here the average amount of times the aggressive_tackles are performed, per field position, are: defender (positive = 15.425, negative = 23.7), midfielder (positive = 10.83, negative = 17.367), and attacker (positive = 15.63, negative = 23.3).

To sum up, the results show that passing and aggressive tackles showed significant differences, where the positive emotional team, for passing, performed significantly more passes than the negative emotional team. For aggressive_tackles, the negative emotional team performed, for all the field positions, significantly more tackles than the positive emotional team. Most of the results for the other actions showed differences in their means, but, due to a huge factor of uncertainty, also showed relatively high standard deviations. This high amount of fluctuations resulted in higher p-values (lower significance) for the other actions. This might be resolved by including more simulation results to the variable pool.
7.6 Conclusion RoboCup
In the conclusion from LeadsTo was mentioned that there was no interaction between agents’ actions and their effect in the world. In the RoboCup implementation there is a direct mapping between performed actions and environmental changes. Therefore the performed action could be checked on performance (i.e. failed or succeeded) and the importance of that action (i.e. a contribution to a desire) to update the mental states.

Nonetheless, the implemented model is, in principle, a copy of the one designed before and previously implemented in LeadsTo. Other exceptions were discussed in paragraph 7.4. As demonstrated in paragraph 7.5.1, the mental states (i.e. Information Module) contain values that represent the importance or belief of that module. It has also been shown that when actions succeed that beliefs and emotions change, although, opposed to the LeadsTo model, most of the desires were not updated. The reason for this was partly because of a lack of time and because implementing that part of the model would not increase the model’s performance much. It was, therefore, of little importance in the current setting, since the operatability of the model within the RoboCup environment is also apparent without these updates. Nonetheless, these updates were done for the desire “release_frustration”. This is because this desire is directly connected with the action aggressive_tackle. Since all the others actions can contribute with some extent to other desires, aggressive_tackle has no added value to any of the other desires. Another reason for updating one desire, instead of none, is to show the structure of updating desires in the current implementation.

The results described before prove that there is a great difference in the executed actions for different players’ field positions. This is based on the desires the players have and the rate of contribution an action has with respect to the players’ desires. For example, it is more important to pass the ball, for defenders (mean positive 5.8) and midfielders (mean positive 5.2) than for attackers (mean positive 2.6). As shown in the results section, there were eight tests that were not conclusive. Three of them were on tactic_tackles, which could be due to fact that this action is not often executed (the mean in the positive emotional team is 1.42, and in the negative emotional team is 1.22 per player per game). It, furthermore, makes sense that the difference in passing, between defenders and midfielders, is not significantly different, because they both want to give the attackers a chance to score.

The results for the differences between entire teams (positive versus negative) were, as mentioned before, significant for both passing and aggressive_tackles. The same has been shown during the comparison between field positions. This indicates that positive emotional teams are far less likely to perform aggressive tackles, regardless of field position, because they are happier. Notice, though, that although the emotional states of the teams are different, the desire update of ‘release_frustration’ is updated, although this is coupled with the mood of the agent, as well as with the emotion the agent has over itself. The other significant difference was shown for defenders and the amount of times they perform the action ‘pass’. Passing is an action that contributes to desires, such as: keep_ball_possession, give_score_chance. The results indicate that the emotionally more positive teams perform more actions that contribute to desires, than the teams that have negative emotions. This is caused, as explained in paragraph 7.4.3, by taking the square of the expectation value. This square realized higher differences in expectation_values, whereby the ‘better’ actions have remarkably higher expectation_values than the
worse ones. Highly positive emotional teams, as opposed to negative ones, produce remarkably higher expectation_values, since the contribution of emotions is, approximately, 60 percent. Emotions, in this case, include the emotion_on_self, mood, and (dependent on whether there is an interaction) emotion_on_player_X. Thus, positive emotional teams are more likely to distinguish between ‘useful’ and ‘useless’ actions.

To sum up, the results show that players with different desires perform a significant amount of different actions. This is regardless of the emotional state of the player. In addition, the different emotional states (i.e. positive and negative) produced different actions. Here, only the actions ‘aggressive_tackle’ and ‘pass’ showed significant differences, the mean of some of the other actions indicates more differences between emotional states, although not sufficient to produce significant results. It is, noticeable, also, that other mental states, such as capabilities, trust, and beliefs over the environment, also contribute to the generation of intentions.

7.7 Future work RoboCup
It has now been shown that the conceptual model, designed earlier, produces satisfying results. In the results section two aspects have been examined. First, the contribution emotions have for action selection. Secondly, the differences in the (number of) actions players with different field positions perform. The contributions of the other mental states still have to be empirically tested.

The performance of this team is, probably, less than the other teams playing in the league, why? Some minor, model unrelated, abilities were not implemented. To make optimizing the team’s performance possible, a few shortcomings will be described next; 1) the agents base their intentions solely on the current environment, although they use previously learned information, they do not consider (possible) next world states. 2) They ignore the opponent while choosing most of their actions. They only consider the opponent when the opponent has ball possession. When, for instance, a player wants to pass to another player, the opponents’ locations are ignored. 3) The dribble function is basic. Meaning that the player dribbles straight ahead, until its next position is further away from the target location than the current position, then the dribble action fails. 4) The teams created, both the rational and the eBDI ones, will most probably lose all the games they play against teams that play in the league. The main reason for this is that most teams have one or multiple joint strategies, while the current modeled teams have none. This should be the work of the coach. But the players modeled here have no interaction with the coach, mainly, because it is outside the scope of this project to make a good team. 5) Aggressive tackle has no added value for the performance of the team. On the contrary, it costs time and stamina. This function is not implemented with the idea to improve the team’s performance, but merely to visualize anger and frustration in agents.

The model implemented in RoboCup using the Brainstormers team as starting point provided a closer look at the eBDI conceptual model described before. As discussed before, the number of actions players can choose from is limited. Because of this limitation the necessity of emotions, at least for fast decision making, is low since players in the current settings have enough time to think all the possible actions through. The importance of emotions will become more prominent when the actions are really “created”, meaning that only low level actions are pre-described (e.g. go_to) and where all the ‘higher-
level’ actions (e.g. intercept ball) use multiple low level actions (go_to(location), turn_body(degrees), collect_ball()). The actions used in this setting are reused from the Brainstormers team. It is, therefore, not necessary for the agents to think further on how to execute the action (because it is predefined). Emotions could also change performance when the agents would try to look in the future. In the current implementation the players only think about their current action (i.e. the agents do not plan their next actions). If the agents, for instance, start dribbling they do not perform this action with the intention to perform a pass or shoot at goal action later on, but only because dribbling is the action that contributes most to their desires (and action possibilities) at that moment. To sum up, their reasoning is still limited but the model permits more extensive reasoning.

As can be seen from one of the movies sometimes strange things happen. For instance, free kicks often fail; the same holds for goal kicks. The reason for this is simply because the player does not have enough time to ‘think’. Since only the play-mode ‘play-on’ has been modeled changing the other play-modes would not have made a difference for the performance of the model and was therefore out of the current project’s scope. This shows, nonetheless, how little time the players get to decide which actions to perform.

There are a few components/modules from the eBDI model that are not modelled. Emotion over events is not implemented due to a lack of time. Furthermore, most of the desire values are not updated. When, for instance, the current team scores, the desires for winning the game and for scoring should be adjusted. In reality, there are a vast amount of such events that in a way contribute to desire changes. This has now only been done for the desire ‘release_frustration’. Thus, this should be done, for the other desires, in the future. Furthermore, the current model can be considered as a player without the ability to look in the future. What should be added to the current belief system is some sort of hope. For example, hope that he scores, or hope that the chance at scoring for the other team fails.

8.0 Conclusion and Future Work
The main purpose of this project was to design and implement a general BDI model with emotions and trust based on (empirically proven) psychological theories. Although emotions are still a matter of research and debate, we have tried to configure the eBDI model so that it as psychologically valid as possible. Emotions and trust are used for how agents perceive themselves, as well as how they think or feel about others. Trust, in the current article is built up out of beliefs on capabilities. Each capability someone has, whether in a game or social environment, has some performance value to it. Notice that these ‘performance values’ are a belief a player has in the capability of himself, or someone else, and do not indicate to what extent an agent can actually execute an action.

With respect to the implementation of capabilities an assumption is made. Research by Jonker et al (2004) has shown that trust (in this article in a copy machine) is based on previous experiences. The assumption made here, is that the trust one has, is based on a combination of abilities. Whereas the copy machine only has one function (or little other functions), humans have more abilities. Therefore, the assumption that has been made is that ‘trust’ in that article can be compared with beliefs in multiple capabilities, as defined in the current model. Furthermore, some abilities/capabilities for machines or
humans are more important than others. Since this has not been tested empirically yet, this should be
done to prove the assumption made here.

The relations between the different mental states (emotion, belief, desire, and trust) are made
more apparent by implementing the eBDI model in two simulation environments. These relations were
explicitly displayed in the movies discussed in paragraph 7.5.1. It was shown here, how the dynamically
changing mental states are updated, depending on the results of the performed actions. This provides
observable relations between the mental states.

Another reason for implementing this model in RoboCup is to produce more human-like
decisions, so that the observer can ‘understand’ why the player is acting the way he does and therefore
feels to be better ‘emotionally bonded’ with the player the observer is tracking. This has not been tested
empirically, so that should be done in the future.

The framework is extensively described in Chapter 5. That chapter is good for inside information
on the purpose, the mental states, the connection between the mental states, and the assumptions
made for the current model. This makes the model better reproducible and testable for researchers. For
example, computer scientists can use the source code (for the entire source, programmed in C++, please
send a request e-mail to tboosse@few.vu.nl or daniel_hohle@hotmail.com) to have a good start for
implementing the model in whatever environment they want to use the model in. As such, the
presented model has provided a solid basis for follow-up research.
References


```c
#ifndef _MODELPARAMETERS_H_
#define _MODELPARAMETERS_H_

/* NOTE: THIS FILE IS FOR THE SIMULATION RESULTS */
/* personality */
static const int personality_value = 5; // used for updating capabilities (the lower the value the faster the update)
static const int emotional_value = 2; // used for updating emotions (the lower the value the faster the update)

/* number of players/events/desires/actions */
static const int numbe_of_different_actions = 30;
static const int number_of_capabilities = 12; //0-9 actions. 10: direct_pass, 11: long_pass
static const int number_of_players_to_think_about = 10; //excluding keeper
static const int number_of_desires = 8;
static const int number_of_different_events = 10; //not implemented (emotion_over_event)

/* begin parameters */
static const double beginMood = 0.2;
static const double beginEmotionOnPlayer = 0.2;
static const double beginEmotionOnEvent = 0.2;
/* trust update parameters (sum = 1!) */
static const double weight_importance_old_trust = 0.75;
static const double weight_importance_new_trust = 0.25;
/* intention update parameters (sum = 1!) */
static const double weight_importance_old_intention = 0.5;
static const double weight_importance_new_intention = 0.5;
/* mood update parameters (sum = 1!) */
static const double weight_importance_old_mood = 0.5;
static const double weight_importance_new_mood = 0.5;
/* desire update parameters (sum = 1!) */
static const double weight_importance_old_desire = 0.5;
static const double weight_importance_new_desire = 0.5;
/* IntentionGeneration */
/* without player interaction (add up to 1) */
static const double value_without_interaction_with_emotion_action_can_be_performed = 0.1;
static const double value_without_interaction_with_emotion_capability_value_self = 0.1;
static const double value_without_interaction_with_emotion_trust_in_self = 0.1;
static const double value_without_interaction_with_emotion_desire_contribution = 0.1;
static const double value_without_interaction_with_emotion_mood = 0.3;
static const double value_without_interaction_with_emotion_on_player_self = 0.3;
/* with player interaction (add up to 1) */
static const double value_with_interaction_with_emotion_action_can_be_performed = 0.05;
static const double value_with_interaction_with_emotion_capability_value_self = 0.05;
static const double value_with_interaction_with_emotion_trust_in_self = 0.05;
static const double value_with_interaction_with_emotion_desire_contribution = 0.05;
static const double value_with_interaction_with_emotion_mood = 0.23;
static const double value_with_interaction_with_emotion_on_player_self = 0.23;
static const double value_with_interaction_with_emotion_on_player_other = 0.24;
/* start desire values (nog niet verwerkt) defender */
static const double value_win_game_defender = 0.5;
static const double value_ball_in_possession_defender = 0.5;
static const double value_ball_nearby_goal_defender = 0.5;
static const double value_give_score_chance_defender = 0.5;
static const double value_score_other_goal_defender = 0.5;
static const double value_not_to_be_tackled_defender = 0.5;
static const double value_not_score_own_goal_defender = 0.5;
/* start desire values (nog niet verwerkt) midfielder */
static const double value_win_game_midfielder = 0.5;
static const double value_ball_in_possession_midfielder = 0.5;
static const double value_ball_nearby_goal_midfielder = 0.5;
```
static const double value_give_score_chance_midfielder = 0.5;
static const double value_score_other_goal_midfielder = 0.5;
static const double value_not_to_be_tackled_midfielder = 0.5;
static const double value_not_score_own_goal_midfielder = 0.5;
/* start desire values (nog niet verwerkt) attacker */
static const double value_win_game_attacker = 0.5;
static const double value_ball_in_possession_attacker = 0.5;
static const double value_ball_nearby_goal_attacker = 0.5;
static const double value_give_score_chance_attacker = 0.5;
static const double value_score_other_goal_attacker = 0.5;
static const double value_not_to_be_tackled_attacker = 0.5;
static const double value_not_score_own_goal_attacker = 0.5;
/* start capability values (nog niet verwerkt) defender */
static const double value_direct_pass_to_player_defender = 0.5;
static const double value_long_pass_to_player_defender = 0.5;
static const double value_receive_ball_defender = 0.5;
static const double value_shoot_at_goal_defender = 0.5;
static const double value_intercept_ball_defender = 0.5;
static const double value_dribble_defender = 0.5;
static const double value_tackle_defender = 0.5;
static const double value_run_free_defender = 0.5;
static const double value_search_ball_defender = 0.5;
static const double value_guarding_defender = 0.5;
/* start capability values (nog niet verwerkt) midfielder */
static const double value_direct_pass_to_player_midfielder = 0.5;
static const double value_long_pass_to_player_midfielder = 0.5;
static const double value_receive_ball_midfielder = 0.5;
static const double value_shoot_at_goal_midfielder = 0.5;
static const double value_intercept_ball_midfielder = 0.5;
static const double value_dribble_midfielder = 0.5;
static const double value_tackle_midfielder = 0.5;
static const double value_run_free_midfielder = 0.5;
static const double value_search_ball_midfielder = 0.5;
static const double value_guarding_midfielder = 0.5;
/* capability contributes to trust self */
/* defender */
static const double capability_contributes_to_trust_receive_ball_defender_self = 0.125;
static const double capability_contributes_to_trust_intercept_ball_defender_self = 0.225;
static const double capability_contributes_to_trust_shoot_at_goal_defender_self = 0.025;
static const double capability_contributes_to_trust_direct_pass_defender_self = 0.175;
static const double capability_contributes_to_trust_run_free_defender_self = 0.025;
static const double capability_contributes_to_trust_tactic_tackle_defender_self = 0.15;
static const double capability_contributes_to_trust_long_pass_defender_self = 0.075;
/* midfielder */
static const double capability_contributes_to_trust_receive_ball_midfielder_self = 0.15;
static const double capability_contributes_to_trust_intercept_ball_midfielder_self = 0.15;
static const double capability_contributes_to_trust_shoot_at_goal_midfielder_self = 0.1;
static const double capability_contributes_to_trust_direct_pass_midfielder_self = 0.075;
static const double capability_contributes_to_trust_run_free_midfielder_self = 0.025;
static const double capability_contributes_to_trust_tactic_tackle_midfielder_self = 0.175;
static const double capability_contributes_to_trust_long_pass_midfielder_self = 0.125;
static const double capability_contributes_to_trust_dribble_midfielder_self = 0.1;
static const double capability_contributes_to_trust_guarding_midfielder_self = 0.1;
/* attacker */
static const double capability_contributes_to_trust_receive_ball_attacker_self = 0.175;
static const double capability_contributes_to_trust_intercept_ball_attacker_self = 0.1;
static const double capability_contributes_to_trust_shoot_at_goal_attacker_self = 0.25;
static const double capability_contributes_to_trust_direct_pass_attacker_self = 0.025;
static const double capability_contributes_to_trust_run_free_attacker_self = 0.125;
static const double capability_contributes_to_trust_tactic_tackle_attacker_self = 0.05;
static const double capability_contributes_to_trust_long_pass_attacker_self = 0.075;
static const double capability_contributes_to_trust_dribble_attacker_self = 0.175;
static const double capability_contributes_to_trust_guarding_attacker_self = 0.025;
/* capability contributes to trust others */
/* defender */
static const double capability_contributes_to_trust_receive_ball_defender_other = 0.15;
static const double capability_contributes_to_trust_intercept_ball_defender_other = 0.275;
static const double capability_contributes_to_trust_shoot_at_goal_defender_other = 0.025;
static const double capability_contributes_to_trust_direct_pass_defender_other = 0.25;
static const double capability_contributes_to_trust_run_free_defender_other = 0.025;
static const double capability_contributes_to_trust_tactic_tackle_defender_other = 0.275;
/* midfielder */
static const double capability_contributes_to_trust_receive_ball_midfielder_other = 0.2;
static const double capability_contributes_to_trust_intercept_ball_midfielder_other = 0.2;
static const double capability_contributes_to_trust_shoot_at_goal_midfielder_other = 0.1;
static const double capability_contributes_to_trust_direct_pass_midfielder_other = 0.2;
static const double capability_contributes_to_trust_run_free_midfielder_other = 0.1;
static const double capability_contributes_to_trust_tactic_tackle_midfielder_other = 0.2;
/* attacker */
static const double capability_contributes_to_trust_receive_ball_attacker_other = 0.225;
static const double capability_contributes_to_trust_intercept_ball_attacker_other = 0.05;
static const double capability_contributes_to_trust_shoot_at_goal_attacker_other = 0.275;
static const double capability_contributes_to_trust_direct_pass_attacker_other = 0.1;
static const double capability_contributes_to_trust_run_free_attacker_other = 0.2;
static const double capability_contributes_to_trust_tactic_tackle_attacker_other = 0.15;

#endif