# ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

## Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Η εφαρμογή ενός μοντέλου που προσδοκούμε να επηρεάσει τη συμπεριφορά των γνωστικών πρακτόρων σε ένα σενάριο αποφυγής συγκρούσεων

Ιωάννα Παντελοπούλου
iopadelo@inf.uth.gr
Μαδρίτη, Ιούλιος 2014

Επιβλέποντες: Ricardo Imbert Paredes Δημήτριος Κατσαρός

**Η διπλωματική μου εργασία συντάχθηκε κατά τη διάρκεια της συμμετοχής μου στο πρόγραμμα Erasmus του εαρινού εξαμήνου του 2014, στο τμήμα Computer Science and Engineering School - Universidad Politécnica de Madrid, υπό την επίβλεψη του Director of the European Master in Software Engineering, Ricardo Imbert Paredes.**

UNIVERSIDAD POLITÉCNICA
DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS**


# Application of an expectation model to influence cognitive agents behaviours in a collision avoidance scenario

B.Sc. Thesis

Ioanna Pantelopoulou

Madrid, July 2014

This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of Bachelor of Science on Computer Engineering.

*B.S.c Thesis*
*Thesis Title:* Application of an expectation model to influence cognitive agents behaviours in a collision avoidance scenario
July 2014

*Author:*Ioanna Pantelopoulou
Erasmus student at Universidad Politécnica de Madrid
Student from Greece, University of Thessaly

Supervisor:
Ricardo Imbert Paredes
Ph.D.
Universidad Politécnica de Madrid
Languages, Systems and Software
Engineering Department
ETSI Informáticos
Universidad Politécnica de Madrid

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain

Given the completion of my present project, I would like to thank my supervising professor, mr.Ricardo Imbert Paredes, who helped me with his guidance and valuable advice. During the course of my effort he was of great support, suggesting solutions to any problems that came up, listening to my ideas and propositions and always showing his interest and confidence in my work. By cooperating with him, I gained a great deal of knowledge and this collaboration contributed to my responding to the needs of this project the best way I could. In addition, I would like to thank my second supervisor, mr.Demetrios Katsaros.

Without any doubt, the most important help I received was from my family. They were always by my side, in good but mostly in bad times, never stopping to show me their love, faith and support during all these years I have been studying.

I definitely owe gratitude to my friends as well, who always stood by me, available whenever I needed them, willing to listen to my problems and offer their advice and encouragement.

In the end, I would like to thank Nikos for all his psychological and technical support, without which it would be impossible to complete the present thesis.

# ΠΕΡΙΛΗΨΗ

Ο στόχος αυτής της έρευνας είναι η δημιουργία ευφυών πρακτόρων λογισμικού. Η έρευνα αυτή βασίζεται στην ανάπτυξη ενός πράκτορα, ενός χαρακτήρα που ελέγχεται από τις νοητικές του ικανότητες, από τα συναισθήματα και από την προσωπικότητα του. Προσομοιώσαμε ανθρώπινα χαρακτηριστικά στον πράκτορα. Για την έρευνα μας η ανθρωπινή εγκεφαλική λειτουργία είναι το κλειδί της, το σημείο αναφοράς της. Η έρευνα είναι εμπνευσμένη από το μοντέλο που δημιουργήθηκε από τον prof Ricardo Imbert Paredes[1], το μοντέλο αυτό ονομάζεται COGNITIVA και εμείς επεκτείναμε ένα μέρος του μοντέλου. Η έρευνα μας επικεντρώθηκε στις προσδοκίες του πράκτορα. Τέλος, αυτή η έρευνα είναι ένα βήμα για να ενθαρρύνει την περαιτέρω έρευνα παρόμοιων ζητημάτων, που έχουν άμεση εφαρμογή στον καθημερινό κόσμο μας.

# ABSTRACT

The aim of this Work is the creation of intelligent software agents. This work is based on the development of an agent, a character controlled by its mental abilities, the moods and his personality. We have simulated human qualities to the agent. We try to reproduce believable behaviours. This present work applies by Prof. Ricardo Imbert Paredes' model (Imbert,2005), called **COGNITIVA**. We have implemented a part of his model. Our research emphasized the expectations of the Agent. Lastly, this work is a step to encourage further research on similar issues which can be directly applied to everyday situations.

## Table of Contents

**ABSTRACT** ..... **6**
**1.INTRODUCTION** ..... **10**
**2.THEORETICAL CONCEPTS** ..... **12**
2.1 Intelligent Agents ..... 12
2.2 Multi-Agent Systems ..... 13
2.3 Rational-Emotional Agents ..... 14
2.4 Environment of simulation :Avoid up Collisions ..... 15
2.5 Expectations ..... 15
2.5.1 What we mean with the term “Expectations” ..... 15
2.5.2 How the expectations affect the mood ..... 16
**3. STATE OF THE ART** ..... **17**
3.1 Abstract ..... 17
3.2 Emotional architectures for agents ..... 17
3.3 Our Cognitive Architecture to Manage Emotions ..... 18
3.4 State of the Agent ..... 20
3.4.1 Current State of the world: Beliefs ..... 20
3.4.2 Manage of the past history ..... 22
3.5 The emotional Behaviour ..... 23
3.5.1 The Effect of Perceptions on Emotions ..... 23
3.5.2 The Effect of Emotions on Actuation ..... 24
3.6 Expectations ..... 26
**4. DEVELOPMENT** ..... **27**
4.1 Description of the Problem ..... 27
4.2 Description of the design ..... 30
4.2.1 Description of the movement ..... 30
4.2.1.1 Perceptions ..... 31
4.2.2 Relations between Personality Trait and Mood ..... 33
4.2.3 Avoid up Collision ..... 35
4.2.4 Description of the Action ..... 39
**5.EVALUATIONS CASES** ..... **42**
1 EXPERIMENT ..... 42
2nd Experiment ..... 51
**6. FUTURE WORK** ..... **59**
6.1 Introduction of parameters ..... 59
6.2 Modifications in the existing algorithm ..... 60
**BIBLIOGRAPHY** ..... **61**
**ANNEX A** ..... **64**
Description of the software Jade ..... 64
Platform ..... 68
JADE Agent ..... 69
Agents Behaviour ..... 70

Unlock an Agent....70
ACL Messages....70

# 1.INTRODUCTION

Expectation is what is considered most likely to occur provided there is an uncertain environment. It is a belief focusing on the future and can or cannot be completed. In cases of lower levels of realization, it leads to a sense of **disappointment.** If a scenario which is not at all expected occurs, it constitutes a **"surprise"**. Expectations and how these contribute to the changes in the emotional state has been the subject of research in most scientific fields, such as psychology, philosophy and economy, to name but a few. Expectations are an emotional state people face in every aspect of their lives. Not only does it apply to an individual's daily activities, but it is also of general scientific interest. Expectations Theory is a problem which has met with numerous studies conducted by psychologists and other scientists towards its solution.

In the present project, the problem of expectations and its solution is being studied through a multi-agent system, within a properly simulated environment where there are agents, some of which do have feelings and personality and others no. In this case, our main goal is to create agents able to move in a certain environment and how they will react to certain unpredictable events, depending on their emotional state. This is why we have implemented an algorithm through which we can create agents moving within a limited environment, some of whom have certain personality traits while others conduct themselves as mere robots. Our focus is to try and study what reactions these agents will have when they collide (against each other), as well as the way they will deal with an unpleasant situation to them based on their emotional state and what expectations derive from this collision. These scenarios are based on a good and effective communication among the agents. The algorithm was implemented using Java and was based on the agent development platform JADE.

In the **2nd Chapter**, there is a brief introduction to the agents and the multi-agent systems and some references are made on basic concepts relating to the believable behaviours, the expectations and how these can be affected by the mood state. A general reference to the simulation of the environment is also made.

In the **3rd Chapter**, there is an extensive elaboration on the development of COGNITIVA, prof.Ricardo Imbert Paredes' model (Imbert, 2005), which has been the inspiration of this study.

In the **4th Chapter**, a general description of the algorithm used towards the solution of the problem in this study is provided. There is a reference to the three stages which essentially compose the algorithm and we analyze what happens in each of them in order to reach the desirable result.

In the **5th Chapter**, the execution of the algorithm is presented. There is a detailed account of the realization process, the admissions that were made, the simulation of the environment of the problem as well as the elements which constitute it, the communication among the agents,

while there is also a presentation of the roles of the classes and the activities taking place in each of these.

In the **6th Chapter** we present the experiments carried out after the implementation of the algorithm.

In the **7th Chapter**, there are some observation made during the course of the present project concerning improvements that could be made in the algorithm used to resolve the problem which has been undertaken. In addition, some ideas and suggestions are offered for further expansions which could broaden the possibilities of the algorithm in the future, so that it could be used/applied in a wider range of problems, which represent applications encountered in a variety of human activities.

# 2.THEORETICAL CONCEPTS

## 2.1 Intelligent Agents

In artificial intelligence, an **intelligent agent** (**IA**) (wikipedia) is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals (i.e. it is rational).Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent, as is a human being, as is a community of human beings working together towards a goal.

Intelligent agents are often described schematically as an abstract functional system similar to a computer program. For this reason, intelligent agents are sometimes called **abstract intelligent agent**s (AIA) to distinguish them from their real world implementations as computer systems, biological systems, or organizations. Some definitions of intelligent agents emphasize their autonomy, and so prefer the term **autonomous intelligent agents**. Still others (notably Russell & Norvig (2003)) considered goal-directed behavior as the essence of intelligence and so prefer a term borrowed from economics, "rational agent".

Intelligent agents in artificial intelligence are closely related to agents in economics, and versions of the intelligent agent paradigm are studied in cognitive science, ethics, the philosophy of practical reason, as well as in many interdisciplinary socio-cognitive modeling and computer social simulations.

Intelligent agents are also closely related to software agents (an autonomous computer program that carries out tasks on behalf of users). In computer science, the term **intelligent agent** may be used to refer to a software agent that has some intelligence, regardless if it is not a rational agent by Russell and Norvig's definition. For example, autonomous programs used for operator assistance or data mining (sometimes referred to as *bots*) are also called "intelligent agents".

Apparently, the most widely accepted definition is that of Wooldridge (Wooldridge & Jennings, 1995) according which :

**An agent is computer system that is situated in some enviroment ,and that is capable of autonomous action in this enviroment in order to meet its delegated objectives.**

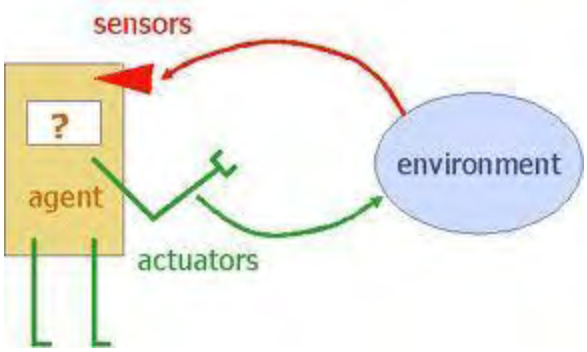



*Fig.1 This is a depiction of the agent and the environment where it functions. The Agent, as/being an autonomous entity, observes the environment and receives inputs through sensors and produces, as exits, actions which affect the environment.*

### 2.2 Multi-Agent Systems

A multi-agent system is one designed and executed as a sum of agents which interact, meaning they cooperate, coordinate, negotiate etc. Mutli-agent systems along with the distributed problem solving form the main fields of the distributed artificial intelligence.
The distributed problem solving deals with how a specific problem can be resolved from a number of processing units which collaborate by sharing knowledge concerning the matter and the individual solutions. The multi-agent systems are essentially a network of “leniently-linked “ agents acting together to resolve problems beyond the capabilities and the knowledge of a single agent. Even though at first it would seem that there is no difference between the two areas , yet the collaboration in the multi-agent systems is dynamic, meaning that the interacting entities are autonomous, and they consequently decide on when and how they will cooperate. This way, a multi-agent system can have as its main aims:

- The resolution of problems otherwise too complex to be resolved efficiently by a single agent
- The resolution of problems by nature distributed, such as ones where collection of data from various sources is needed, like sensor networks, distributed databases, air-traffic control etc.
- The resolution of problems where expertise is distributed, like the flow of tasks in a work(ing) environment
- The connection and operation of already existing legacy systems in a way to render their implementation without any mediator systems efficient.

In multi-agent systems, the agents either work independently by exchanging data or services and try to achieve their individual goals,or cooperate by solving minor problems so that the combination of the partial solutions leads to the final solution. In the general case, the environments of this category are open, there is no central design of the whole system, the

agents operating within these are possible to cooperate or negotiate the accomplishment of their personal goals and offer a possibility of communication to the agents .
The main attribute of the cooperating agents is their coordination possibility through a communication language so that they reach mutually accepted agreements and resolve potential collisions that might derive from the accomplishment of their individual goals. The most inclusive definition of what constitutes coordination (of what coordination is) is likely to have been given by Huhns and Stephens :

**Coordination is a property of a system of agents performing some activity in a shared environment**

### 2.3 Rational-Emotional Agents

According to traditional definitions stemming from Artificial Intelligence and Multi-Agent Systems, **Rational Agents** are associated with programs capable of Practical Reasoning,i.e. building plans and choosing actions to be executed, in order to achieve their goals. For example, SOAR-based architectures are one of the first attempts at modeling the cognitive reasoning process of an agent (Laird, Newell, and Rosenbloom 1987) by means of explicit IF-THEN rules.More recently, the BDI approach of Bratman (1990), Rao and Georgeff (1995) is a theory of practical reasoning (deciding what to do next) directed towards situated reasoning about actions and plans (Allen et al. 1991). Recently, authors have proposed to integrate into rational agent architectures psychological notions, in order to propose: 1)a more complete cognitive models of agents; 2) agents capable of sustaining more human-like interactions with people, especially ordinary people involved in conversational activities with assistant agents.

For example, Gratch and Marsella (2004) have proposed a model of emotions based on SOAR, with a significant impact upon the SOAR architecture. Using the agent creation platform JACK that implements the BDI theory, CoJACK (Norling and Ritter 2004; Evertsz et al. 2008) is an extension layer intended to simulate physiological human constraints like the duration taken for cognition, working memory limitations (e.g. “loosing a belief” if the activation is low or “forgetting the next step” of a procedure), fuzzy retrieval of beliefs, limited focus of attention or the use of moderators to alter cognition.

Emotions have also been integrated to the BDI framework, for instance with eBDI (Jiang, Vidal, and Huhns 2007) or KARO (Steunebrink,Dastani, and Meyer 2007). All those works provide a good introduction about the history of the necessity to implement emotions, and more generally psychological notions, into rational agents.

Although there has also been a lot of research works about the effect of personality on agents’ behaviors in the virtual agents community (one of the most recent one being the SEMAINE project (Bevacqua et al. 2010)), they generally focus more on their impact on the animated agent (e.g. gaze or facial expressions) than on the rational decision process.

## 2.4 Environment of simulation :Avoid up Collisions

The problem we are trying to resolve in this study is one encountered in several aspects of daily life. So, motivated by examples regarding our everyday life and the problems we face in each and every aspect of it, we have tried to simulate one such example in our problem. For example, if one is in a central avenue running to catch the subway to get to work as fast as possible, during this particular route they may encounter lots of people heading towards them and an effort is made to try and avoid them in order not to collide and be late arriving to the destination. Yet, one can never be sure towards which direction the others are moving; they just make assumptions in their effort to avoid collision. A similar example could be when a car crash is about to occur. Such problems taken from our daily life were the inspiration to conduct this kind of research.

The simulation environment we have created concerns the movement of two or more people. More specifically, we have created an environment where an intelligent Agent and Dummies are interacting. To be exact, the Dummies are moving around the environment alone, with no particular purpose, aimlessly. One the other hand, our Agent is moving within the environment with a clear destination and direction toward it. It decides how it wants to move by itself. Consequently, there is a chance that they will collide during their movement in this world. So, the intelligent Agent, since it possesses perceptions and feelings, will react uniquely to this event whereas the Dummy which has no perception will not display a unique reaction because it conducts itself as a robot. In Chapter 5 there is a more detailed description of the simulation environment

## 2.5 Expectations

### 2.5.1 What we mean with the term “Expectations”

In the introduction we mentioned the term “expectations”, in this one we are going to describe it in detail so that we can perceive what we are trying to accomplish in this project. In case we are not sure of something, meaning there is a feeling of uncertainty, using the term expectations we refer to what we consider most likely to happen. An **expectation** is a belief centered on the future, which may or may not be realistic. For instance, an unexpected result or one we would not prefer will provoke a sense of disappointment, namely the Agent after an undesirable collision with another agent will become more nervous and less happy. On the other hand, if something we did not anticipate occurs, it will cause us the feeling of surprise. For example, if the agent sees another dummy heading towards them and it is likely that they will collide, if the dummy ultimately avoids the agent and they do not collide, this will provoke a feeling of surprise to the agent. An expectation related to the behaviour or the performance of another person, expressed according to that person, could be a strong request or a command.

Expectations lead to consequences in the beliefs.. If an expectation is true or not has a direct effect on whether or not a result is going to occur. Regarding the beliefs, it is a situation where a person, according to what is happening to them and to the results of their actions, considers these real, a part of their reality.

Consequently, in this project, we executed a model in which our agent, in accordance with its behaviour and the scenarios of its movements, adopts the respective perceptions and beliefs.In compliance with which it acts and behaves in every occasion. Each of its behaviours is immediately affected by these. We are going to elaborate on all these in the following paragraphs.

### 2.5.2 How the expectations affect the mood

Mood is a certain emotional state. Mood in general differs from emotions in that it is less specific, less intense and less likely to be caused by a specific simulation or event. Generally speaking, mood could have either positive or negative effect. For instance, people generally in a good or bad mood.

Mood is different from temperament and personality, however, there are certain personality traits which intensely affect the mood. Mood is an internal, subjective situation.

A person's expectations can have a considerable effect on their emotional state, if caused under intense circumstances. If for example the feeling of happiness or fear is a a strong feature of the person's psychosynthesis, it is inevitable that their expectations will also be affected. The emotion and the mood constitute a determinant factor concerning a person's expectations and behaviour.

# 3. STATE OF THE ART

## 3.1 Abstract

**Emotions** are an important aspect of human intelligence and have been shown to play a significant role in the human decision-making process. Researchers in areas such as cognitive science, philosophy, and artificial intelligence have proposed a variety of models of emotions. Most of the previous models focus on an agent's reactive behavior, for which they often generate emotions according to static rules or pre-determined domain knowledge. However, throughout the history of research on emotions, memory and experience have been emphasized to have a major influence on the emotional process.

The model I have designed in this study can be considered a part of the model (originally) created by prof. Ricardo Imbert Paredes (Imbert, 2005). In the following paragraphs there will be a detailed description of this model, which presents the mechanisms proposed by a generic cognitive architecture for agents with emotional influenced behaviors, called **COGNITIVA**, to maintain behavior control at will without giving up the richness provided by emotions. This architecture have been used successfully to model intelligent agents with emotions, expectations and intelligence.

## 3.2 Emotional architectures for agents

When trying to incorporate this emotional dimension into computer systems, most of the theoretical models are very hard to be applied directly, because their psychological formulation has a difficult fitting on computer restrictions.In fact, most of the current emotionally inspired computational systems (almost always agent-oriented) match one of a very small group of emotional model types: appraisal models, motivational models, dimensional models. The empirical results of these approaches reveal that including an emotional influence in the agent's reasoning model helps to better explain and understand behaviors observed in real life. However, neither these models nor the architectures and systems developed from them, provide a definitive solution —if there is one— to the inclusion of emotions into the general process of intelligent reasoning. Some deficiency or drawback is always imputed to everyone, although, depending on the contexts and problems, they also prove sometimes to be acceptably adequate.

The structure underlaying emotional architectures is, frequently, very complex. Sometimes, emotional elements and mechanisms are interwoven with the restrictions and particularities of the application context and with the problem to be solved, mingling with them, and making them very difficult and costly to be reused in different contexts (J. Gratch & S. Marsella, 2004), (S. Gadanho, 2003).

In other situations, emotional architectures are very generic, independent from any specific problem (S. Allen, 2001), (D. Canamero, 1997). However, usually the lack of an orientation to the particular necessities of the problem originates less-efficient, computationally

demanding mechanisms. In the end, the need to produce feasible applications usually forces the designers to reconsider their structure and simplify some of their inherent features.

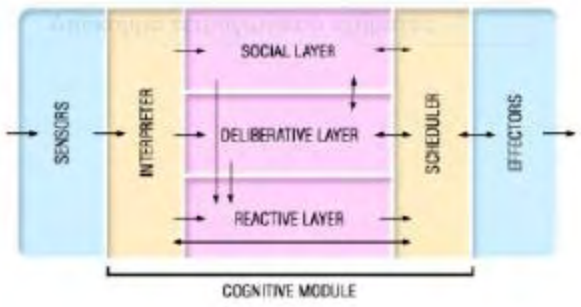
Our hypothesis is that current solutions are not as satisfactory as they should be because they fail, precisely, in the “attitude” with which they cope with complexity: instead of betting on specificity or generality, we believe that the key to the solution lies in adaptivity.
The complexity must be faced from a new perspective to allow the development of both reusable and efficient systems; a new approach adaptable to the specific necessities of the application context and problem, but without losing a generic nature; and a new architectural focus able to provide coherent and explainable structures, components and processes.

### 3.3 Our Cognitive Architecture to Manage Emotions

Our proposal is to model agents using a truly emotionally-oriented architecture, not a conventional architecture with an emotion component.In our opinion, explainable and elaborated emotion-based behaviors can only emerge when the whole architecture has an emotional vocation.

The architecture that we propose, called COGNITIVA, is an agent-based one. Agents are a common choice to model autonomous agents.Considering an agent as a continuous **perception-cognition-action** cycle, we have restricted the scope of our proposal to the “cognitive” activity, although no constraint on the other two modules (perceptual and actuation) is imposed.This is the reason why this architecture will be sometimes qualified as “cognitive”.

In cognitiva, emotions are not considered just as a component that provides the system with some “emotional” attributes, but all the components and processes of the architecture have been designed to deal naturally with emotions. Cognitiva is a multilayered architecture: it offers three possible layers to the actor  designer, each one corresponding to a different kind of behavior, viz reactive, deliberative and social (see Fig. 2). The interaction of these three layers with the other two modules of the actor, the sensors (perceptual module) and the effectors (actuation module), is made through two specific components, the interpreter and the scheduler, respectively.



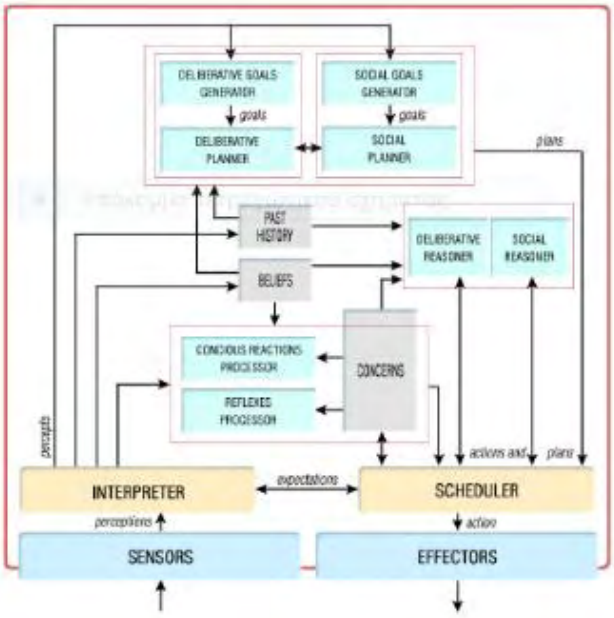

General structure of COGNITIVA.

*Fig2. Structure of COGNITIVA*

The cognitive module described by cognitiva receives from the perceptual module (the actor's sensors) **perceptions** of the environment. This input may not be directly manipulable by most of the processes of the cognitive module and must be interpreted (for instance, sensors might provide measures about light wavelengths, but the cognitive module could only be able to manage directly colors). In other situations, many input may be irrelevant for the agent, and should be filtered.

Cognitiva provides a component, called interpreter, which acts as an interface between sensors and the rest of the cognitive module, receiving the perceptions coming from the perceptual module, filtering and discarding those noninteresting to the agent, and translating them into **percepts**, intelligible by the rest of the components and processes of the cognitive module.

On the other hand, each time some internal reasoning process, in any of the three layers of the architecture, proposes an action to be executed, it must be properly sequenced with other previous and simultaneous action proposals. This should not be a responsibility of the actuation module (effectors), since this module should not need to have any information about the origin and final goal of the actions it receives. Cognitiva proposes a component, the **scheduler**, to act as interface between the cognitive module and the effectors, managing an internal agenda in which action proposals are conveniently sequenced and ordered. The scheduler organizes the actions according to their priority, established by the reasoning process that generated them, and their concurrence restrictions. Once it has decided which is/are the most adequate action/s to be executed, it sends it/them to the effectors.




Internal components and processes of COGNITIVA.

*The dynamics of the architecture follow a continuous cycle, represented in the Fig. 3*

### 3.4 State of the Agent

#### 3.4.1 Current State of the world: Beliefs

Beliefs represent the information managed by the agent about the most probable state of the environment, considering all the individuals in it. Among all the beliefs managed by the agent, there is a small group specially related to the emotional behavior. This set, that has been called the agent's **personal model**, is composed by the beliefs that the agent has about itself. More precisely, this personal model consists on personality traits, moods, physical states, attitudes and concerns.

Cognitiva defines a taxonomy of beliefs, depending on their object and their nature. On one hand, a belief may refer to a **place** in the environment, to **objects** located in the environment, and to other **individuals**. Besides, the agent maintains beliefs concerning the current situation, for instance a belief of my Agent about the **current situation** may be the fact that the Agent walk in the environment for finding his goal and his destination and during his walk maybe the Agent found dummies and there is a collision.That is not information about the Agent, nor about the dummies, but about the situation that is taking place. Beliefs about places, objects and individuals may include:

- **Defining characteristics** (DCs), traits that mark out the fundamental features of places, objects or individuals. DCs will hardly change in time, and if they do, it will happen very slowly. For instance, the distance from my Agent to find his destination and his location may be DCs about the place; a DC about CodeWalker (individual) and the Dummies are their names. Among all the DCs that an agent can manage, cognitiva prescribes the existence of a set of **personality traits** (P) for individuals. Personality traits will mark out the general lines for the agent's behavior. For instance, my Agent can be provided with two personality traits, **self-control** and **positiveness.**

- **Transitory states** (TSs), characteristics whose values represent the current state of the environment's places, objects or individuals. Unlike the Dcs, whose values are, practically, static in time, the TSs values have a much more dynamic nature. Cognitiva considers essential two kinds of TSs for individuals: their **moods**(M), which reflect the emotional internal state of the agents; and their **physicalstates** (F), which represent the external state of the actors . In our example, CodeWalker could have as moods **happiness, nervousness** and **disgust**.

- **Attitudes** (As), which determine the predisposition of the agent towards the environment's components (places, objects and individuals). Attitudes are less variable in time than TSs, but more than DCs. Attitudes are important to guide the agent's decision making, action selection and, above all, to keep coherence and consistency in the agent's interactions.

The elements of the personal model in our architecture have been modeled with a fuzzy logic representation. Fuzzy logic linguistic labels are nearer to the way in which humans qualify these kind of concepts (it is usual to hear "I am **very** happy", instead of "My happiness is **0.8**"). Besides, fuzzy logic is a good approach to manage imprecision. Relationships among personal model elements are a key point in cognitiva. Many of these beliefs are conceptually closely related, and have a direct influence on each other:

- Personality traits exert an important influence determining emotions. For instance, in a similar situation, the value of the mood **happiness** will be different for a **positiveness** agent than for a **negative** one.

- The set of attitudes of an agent has some influence on the emotions that it experiences. For instance, the **collision** of our **Codewalker** with the dummy will produce an increment on the agent's **nervousness.**

- Personality traits, in turn, have influence on attitudes. The **colision** towards the dummy will be different depending on the value for the personality trait **positiveness:** a **negative** agent will feel absolute rejection towards dummies, whereas a **happiness** one just will not like them.

- Physical states have also influence on emotions. For instance, when the agent is **positiveness**, its **happiness** will increase.

- Finally, personality traits exert some influence on concerns. This influence will be explained later on.

All these relationships have been designed and implemented through special fuzzy rules and fuzzy operators. The result is a set of fuzzy relationships, which might include the following:

```
positiveness DECREASES <much> nervousness
positiveness INCREASES <few> happiness
positiveness DECREASES <some> disgust
self-control DECREASES <much> happiness
```

### 3.4.2 Manage of the past history

The agent's past history maintains propositions related to any significant event —from the agent's point of view— that happened. Past history is a key tool to include, in the agent reasoning, considerations on events occurred in past moments. Behaviors that do not take into account past events are disappointing to human observers, specially in storytelling. Cognitiva considers two mechanisms to maintain the agent's past history information:

- **Accumulative effect of the past:** this is an implicit mechanism, related to the way in which beliefs are managed. External changes in the environment or internal modifications in the agent's internal state may produce an update of the agent's beliefs. In the case of transitory states, this update is performed as a variation —on higher or lower intensity— on the previous value of the belief, avoiding abrupt alterations in the individual's state.

- **Explicit management of the past state:** an accumulative effect of the past events may not be enough to manage efficiently the past state, because it does not consider information related to the events themselves or to the temporal instant in which they took place. Cognitiva maintains explicit propositions related to any significant event —to the agent— that happened. In the agent's past history we could find events such as the dummies returned left for avoiding our agent, or in a previous moment the dummies returned right and they had a colision with our agent. Past history allows the agent to reason considering facts occurred in past moments. As a possible way to implement it, an inverse delta based mechanism has been developed to manage past events.

### 3.5 The emotional Behaviour

Emotions, in particular moods, may be a strong force to drive the agent's behavior. As it was seen before, emotions are part of the state of the agent. If their values are properly updated and their effects are reasonable, the outcomes of the emotionally based behavior will not be unpredictable, but coherent responses.

#### 3.5.1 The Effect of Perceptions on Emotions

Cognitiva provides some mechanisms to update and control the internal state of the agent and, in particular, to control the values of the components of the personal model.
In the first place, the interpreter will direct the interpreted precepts to the convenient processes in every layer of the architecture. The interpreter also feeds information for updating past history and beliefs. Most of that updating may be more or less automatic, and needs no further processing. For instance, when the agent perceives the new position of the dummy, the interpreter will update automatically the beliefs about his new position that he wants to move.
However, that is not the case for moods, and moods are the core of emotional behavior. The new value of moods depends on their old value and on the perceptions, but also on what was expected to happen and to which degree that occurrence was desired. Moods need a new factor to be conveniently generated. With this aim, cognitiva includes the mechanism of the **expectations**, inspired on the proposal of Seif El-Nasr , which has been adapted, in turn, from the OCC Model.

Expectations capture the predisposition of the actor toward the events — confirmed or potential. In cognitiva, expectations are valuated on:

- Their **expectancy:** Expressing how probably the occurrence of the event is expected

- Their **desire:** Indicating the degree of desirability of the event

Through expectations, the interpreter has a mechanism to update moods from perception:

- When the event **occurrence has not yet been confirmed.** Moods will be updated depending on the degrees of expectancy and desire for the event. For example, if the Agent knows that the Dummy was going towards to him he may elaborate the expectation "maybe happened a collision". That is an undesirable event, whose occurrence has not been confirmed yet, that produces a sensation of distress, increasing the value of agent's "nervousness" and decreasing the value of his "happiness".

- When the event **occurrence has already been confirmed.** Again, depending on the degrees of expectancy and desire for the event, moods will be updated. For instance, if someone (World Agent) informed our Agent that there is one collision between our Agent and the Dummy, his fears would be confirmed, and his distress would transform into disgust, decreasing considerably the value of her "happiness".

- When the event **non-occurrence has already been confirmed.** The degree of expectancy and desire of the event will determine the updating of moods. For instance, when our Agent see the dummy to go for an other way and to avoid him, he believes that there is no collision, so his expectation about collision vanishes, and distress would give way to relief by increasing “happiness” and decreasing nervousness and disgust.

This is how expectations are used to update moods, but, what is the origin of those expectations? Actions will have a set of associated expectations. When the scheduler selects an action to be executed, it informs the interpreter about what is expected to occur in the future, according to that action.

### 3.5.2 The Effect of Emotions on Actuation

It is reasonable that actions proposed by the reactive layer have a higher priority in the queue of actions to be executed than those coming from the deliberative or social layers. Even more, it makes sense that deliberative or social action executions are interrupted when a reactive action emerges. Then, does not that mean that reactive, instinctive, passional actions will always take the control of the actuation, leaving out higher level behaviors? Is not that, in fact, losing control? In fact, humans have, to some extent, the ability to control the reactions that logically follow from their emotional state.

The mechanism proposed in Cognitiva to allow higher level control the agent’s actuation is the use of **concerns**. Beliefs represent information about what the agent thinks is the most probable state of the environment, including itself and the rest of the agents. For instance, when our Agent (CodeWalker) has a collision with others dummies, he will know that he his feeling nervousness but as this is not at all a desired state for him, he should try to do something to change that state.

**Concerns** express the desirable/acceptable values for the TSs of an agent anytime, in particular, for **emotions** and **physical states**. Concerns restrict the range of values of the TSs of the agent, expressing the acceptable limits in a certain moment. With this aim, concerns provide two **thresholds**, **lower** and **upper**, for every TS. All the values among them will be considered as desired by the agent; those values out of this range will be unpleasant, and the agent will be inclined to act to avoid them and let them move to the desired range.

Then, a reaction, besides some triggering conditions, the operator to be executed, the consequences of its execution, and some other parameters, such as its priority or its expiry time, will be provided with **justifiers,** i.e., emotional restrictions that must be satisfied to execute the action. Justifiers are expressed in terms of restrictions related to the value of the agent’s concerns, that is, restrictions on the desirable state of the actor. For instance, a justifier to trigger a reaction to change direction or change mood because of the nervousness produced by the collision with the Dummy will be:

```
nervousness > upper threshold concern(nervousness)
```

Whenever some agent wants to be able to stand a bit more nervousness, it first must raise the value of the upper threshold of this concern. If nervousness does not surpass the value of the upper threshold, the reaction will not be justified and it will not be triggered.

Depending on the personality traits of the individual, which have some influence on concerns as it was mentioned before, the real new value for that upper threshold will be higher or lower. Coming back to the scenario , if we have two agent with personality trait, one positiveness and another one easily negative, and they have the both a collision with one Dummy, their nervousness will raise and a reaction of escape would be triggered. Once they are far enough and their nervousness has descended under the value of the upper threshold of its corresponding concern, but still they will feel worried if it will happen a collision again. However, the new information included in their beliefs, the position of the Dummy, prevents them from generating a plan to change direction and not happen collision again. Then the two agents decide to increase their nervousness tolerance (the upper threshold of their concern about nervousness), each one according to their possibilities (their personality traits). They tried again to walk in the road, perceive the Dummy and, again, their nervousness raises. But, this time, the level of the nervousness of the positive agent does not surpass its nervousness tolerance upper threshold, and if it happen collision he isn't so upset. The other agent, less positive, cannot raise enough its nervousness tolerance upper threshold, and, again, he feels so upset and disgust.

In this way, higher processes of the architecture (deliberative and social) can adjust the value of the thresholds of the agent's concerns to control the instinctive actuation whenever it is not desirable.

### 3.6 Expectations

During the research there were a lot of different models that each focus in a different parts. For example, Bolles and Fanslow proposed a model to account for the effect of pain on fear and vice versa (Bolles and Fanslow 1980). Other models focus on the process by which events trigger certain emotions; these models are called “event appraisal” models. For example, Roseman et. al. developed a model to describe emotions in terms of distinct event categories, taking into account the certainty of the occurrence and the causes of an event (Roseman et al. 1990). Other models examine the influence of expectations on emotions (Price et al. 1985). While none of these models presents a complete view, taken as a whole, they suggest that emotions are mental states that are selected on the basis of a mapping that includes a variety of environmental conditions (e.g., events) and internal conditions (e.g., expectations, motivational states).

In this papar, we will  focus on the part of the **expecations** of our model COGNITIVA. Then we will try to implement the part of the expectations of this model.Besides, to update moods, COGNITIVA allows the definition of expectations, inspired on the proposal of Seif El-Nasr, adapted, in turn, from the OCC Model. Expectations capture the predisposition of the agent towards events—confirmed or potential.

# 4. DEVELOPMENT

### 4.1 Description of the Problem

For the problem we have resolved in this project, we have used an algorithm whose aim is to research an Agent's behaviours, reactions and expectations in case of a collision.

The design of the algorithm can be divided into three phases:

**Phase 1:** We have created an agent(**CodeWalker**) who has a certain **personality** according to which a mood is created respectively. In fact, what we wanted to achieve was to create an Agent who responds to human characteristics and behaviours, because it is far more interesting to create and simulate the CodeWalker with feelings, personality and behavior rather that spend time on one who behaves like a robot, meaning it has a predefined **actions** and **behaviours**. When designing a robot, the results are entirely expected and the experiment lacks interest. Additionally, apart from our Agent, we create **Dummies** as well, which/who behave like **robots**, in other words they have a certain predefined behavior. With this in mind, we create the following scenario : We have got an Agent moving in an environment and having already decided set a destination where it intends to arrive fast, staying relaxed and without encountering any further problems.

**Phase 2:** This next phase is about their **colliding**. We add the Dummies in the environment so that we can observe how the Agent will react, based on its personality, when it stumbles upon obstacles which will make its task harder, make it sad, then angry and in the end frustrated, resulting in no letting it reach its goal easily. Their role is to cause collision in the Agent's movement. This means that when they meet in the same position, they will collide against each other and we will observe the way our Agents will react.

**Phase 3:** This is the most interesting part of the project, the CodeWalker's **reaction**. Its reaction involves numerous elements. For instance, if the CodeWalker collides with one or more Dummies, the likeliest and most reasonable scenario is that it will want to change its direction in order to avoid them and reach its destination without any consequences. This incident will also alter the CodeWalker's mood, which is directly related to its personality. There is also a function inextricably linking the CodeWalker's personality and mood (which we are going to describe in more detail later on). More specifically, we have added the following personality traits: self-control and positiveness. And as mood there are the following ones: happiness, nervousness and disgust. Therefore in the example given, if our agent is not positive enough and happens to collide with another one, it will experience a decrease in its happiness, an increase in its nervousness and a considerable increase in its disgust. On the other hand, the opposite scenario will take place if our agent could be described as rather positive/positive enough. A variety of other kinds of reactions may also be experienced by our agent, reactions we will focus

more on in the next sections and which depend on the collision scenarios we are going to examine. Thus, after the Agent has reacted, we are going to evaluate the results according to its expectations as to whether these were expected and desired or not. In our example, this could be explained by if the  Agent had foreseen the way the Dummy would move and if eventually that was the desirable one.

Summing up, our study focus on the collision between the Agent and the Dummies, on the Agent's reaction, as well as on its expectations regarding the desirability of these expectations

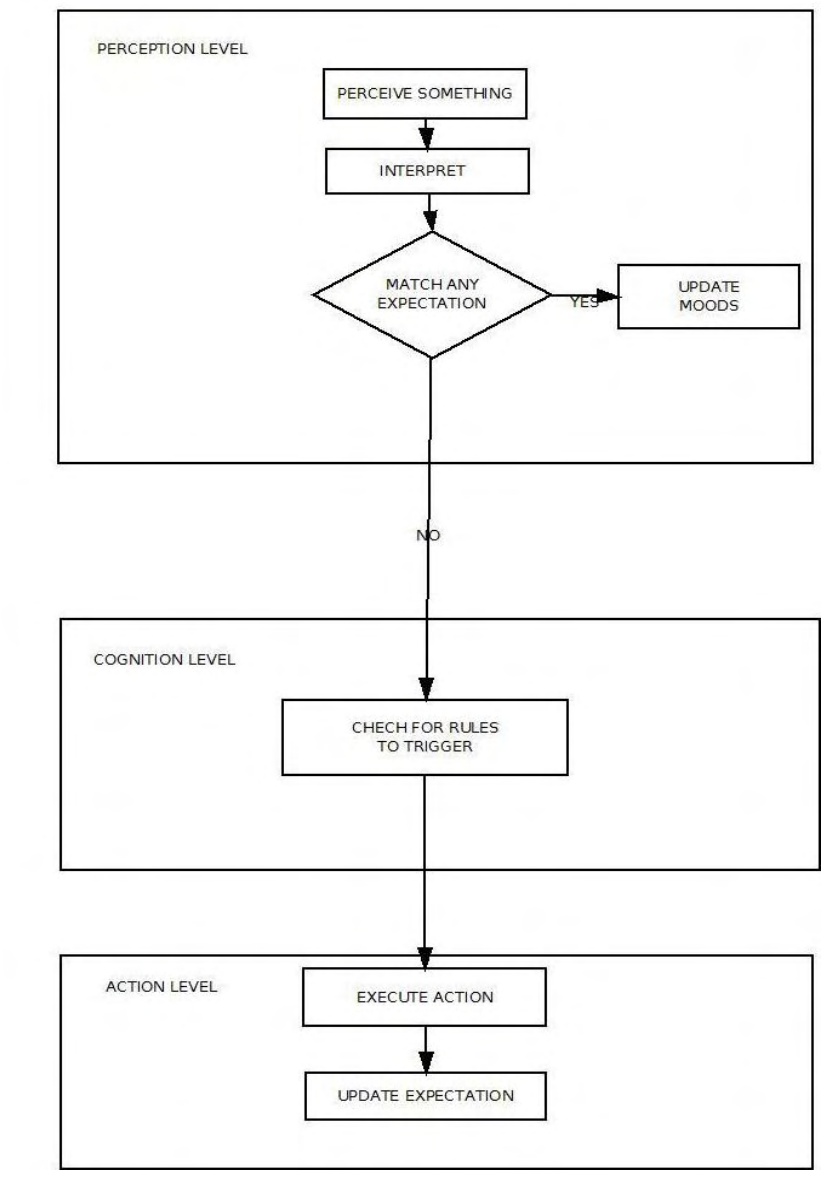
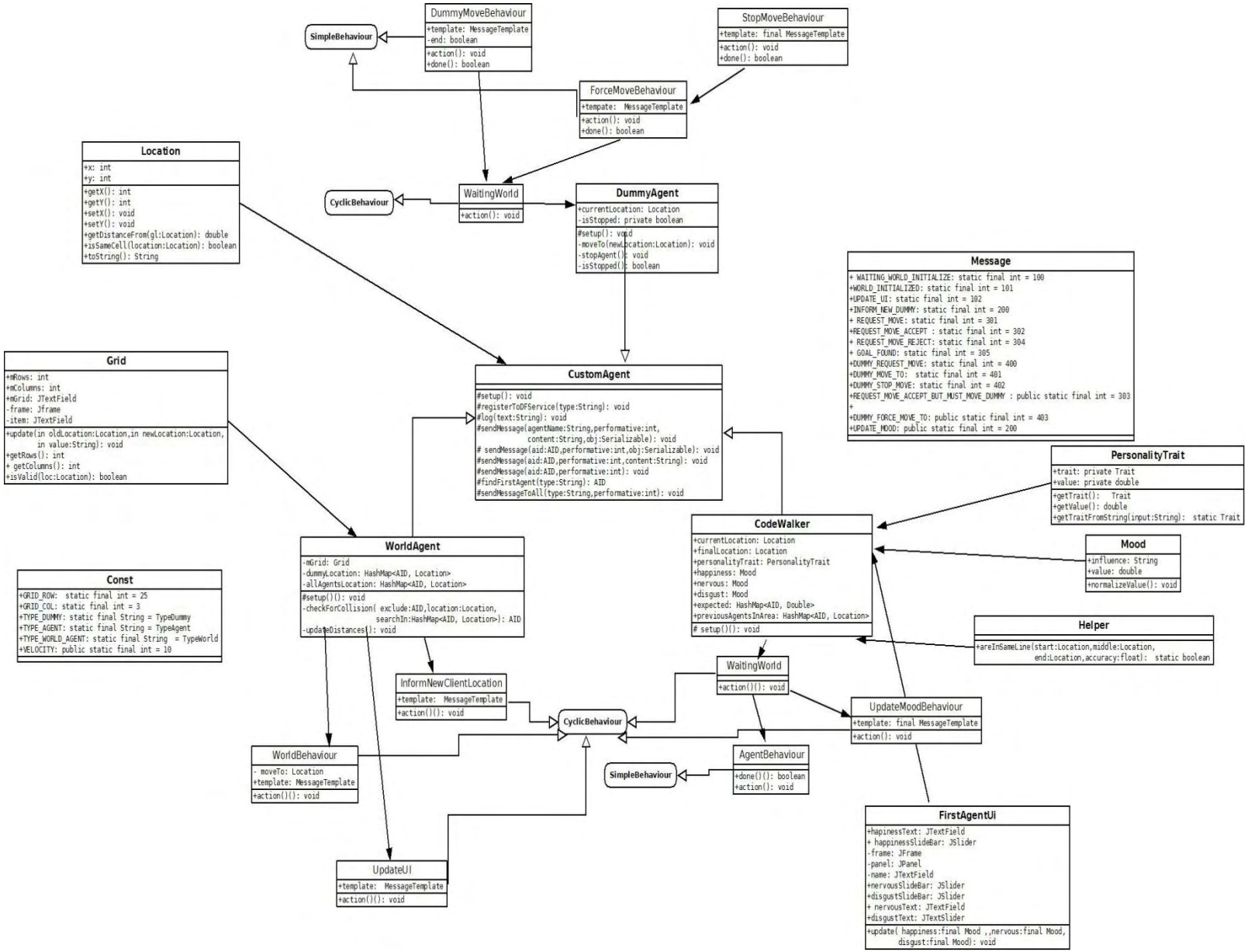



*Fig.4 The architecture of the Algorithm*


DummyMoveBehaviour
+template: MessageTemplate
-end: boolean
+action(): void
+done(): boolean
SimpleBehaviour
StopMoveBehaviour
+template: final MessageTemplate
+action(): void
+done(): boolean
ForceMoveBehaviour
+tempate: MessageTemplate
+action(): void
+done(): boolean
Location
+x: int
+y: int
+getX(): int
+getY(): int
+setX(): void
+setY(): void
+getDistanceFrom(gl:Location): double
+isSameCell(location:Location): boolean
+toString(): String
CyclicBehaviour
WaitingWorld
+action(): void
DummyAgent
+currentLocation: Location
-isStopped: private boolean
#setup(): void
-moveTo(newLocation:Location): void
-stopAgent(): void
-isStopped(): boolean
Message
+ WAITING_WORLD_INITIALIZE: static final int = 100
+WORLD_INITIALIZED: static final int = 101
+UPDATE_UI: static final int = 102
+INFORM_NEW_DUMMY: static final int = 200
+ REQUEST_MOVE: static final int = 301
+REQUEST_MOVE_ACCEPT : static final int = 302
+ REQUEST_MOVE_REJECT: static final int = 304
+ GOAL_FOUND: static final int = 305
+DUMMY_REQUEST_MOVE: static final int = 400
+DUMMY_MOVE_TO: static final int = 401
+DUMMY_STOP_MOVE: static final int = 402
+REQUEST_MOVE_ACCEPT_BUT_MUST_MOVE_DUMMY : public static final int = 303
+
+DUMMY_FORCE_MOVE_TO: public static final int = 403
+UPDATE_MOOD: public static final int = 200
Grid
+mRows: int
+mColumns: int
+mGrid: JTextField
-frame: Jframe
-item: JTextField
+update(in oldLocation:Location,in newLocation:Location,
in value:String): void
+getRows(): int
+ getColumns(): int
+isValid(loc:Location): boolean
CustomAgent
#setup(): void
#registerToDFService(type:String): void
#log(text:String): void
#sendMessage(agentName:String,performative:int,
content:String,obj:Serializable): void
# sendMessage(aid:AID,performative:int,obj:Serializable): void
#sendMessage(aid:AID,performative:int,content:String): void
#sendMessage(aid:AID,performative:int): void
#findFirstAgent(type:String): AID
#sendMessageToAll(type:String,performative:int): void
PersonalityTrait
+trait: private Trait
+value: private double
+getTrait(): Trait
+getValue(): double
+getTraitFromString(input:String): static Trait
CodeWalker
+currentLocation: Location
+finalLocation: Location
+personalityTrait: PersonalityTrait
+happiness: Mood
+nervous: Mood
+disgust: Mood
+expected: HashMap<AID, Double>
+previousAgentsInArea: HashMap<AID, Location>
# setup(): void
Mood
+influence: String
+value: double
+normalizeValue(): void
WorldAgent
-mGrid: Grid
-dummyLocation: HashMap<AID, Location>
-allAgentsLocation: HashMap<AID, Location>
#setup(): void
-checkForCollision( exclude:AID,location:Location,
searchIn:HashMap<AID, Location>): AID
-updateDistances(): void
Const
+GRID_ROW: static final int = 25
+GRID_COL: static final int = 3
+TYPE_DUMMY: static final String = TypeDummy
+TYPE_AGENT: static final String = TypeAgent
+TYPE_WORLD_AGENT: static final String = TypeWorld
+VELOCITY: public static final int = 10
Helper
+areInSameLine(start:Location,middle:Location,
end:Location,accuracy:float): static boolean
WaitingWorld
+action(): void
InformNewClientLocation
+template: MessageTemplate
+action(): void
UpdateMoodBehaviour
+template: final MessageTemplate
+action(): void
CyclicBehaviour
WorldBehaviour
- moveTo: Location
+template: MessageTemplate
+action(): void
AgentBehaviour
+done(): boolean
+action(): void
SimpleBehaviour
FirstAgentUi
+hapinessText: JTextField
+ happinessSlideBar: JSlider
-frame: JFrame
-panel: JPanel
-name: JTextField
+nervousSlideBar: JSlider
+disgustSlideBar: JSlider
+ nervousText: JTextField
+disgustText: JTextSlider
+update( happiness:final Mood ,,nervous:final Mood,
disgust:final Mood): void
UpdateUI
+template: MessageTemplate
+action(): void

## 4.2 Description of the design

### 4.2.1 Description of the movement

Concerning the Agents and Dummies' movement, we have created a **grid**, a two-dimensional framework where our Agents move, which simulates the environment(world) of the problem. For example, this grid may be 25x3. Each cell of the grid represents a possible position either for a dummy or for an agent. We take for assume that the agents' movement goes towards a single direction each time – upwards, downwards, right or left. What this means is that an agent needs to fulfill two movements to access a cell located in a diagonal position to its own: A horizontal (right or left) and a vertical one (up or down). To recreate this kind of movement in a more understandable form, we have created a GUI through which we can see exactly how the agents move.

The agents development is provided through the Jade library. We have developed a class, the CustomAgent, which offers its own functions as well, ones used by the Agents for their various functions, but this method also creates each time a snapshot of the Agents class. As a result, it is through this class that we design our main 3 agents (CodeWalker, Dummy, World Agent). We define the location of each agent (CodeWalker, Dummies) in the environment with the line arguments.

Initially we create the CodeWalker and as many Dummies as we want. Later, we create a third agent (called World Agent) which has an assisting role and acts as an administrator, namely it is the one responsible and which adjusts the CodeWalker and Dummies' movement. They do not know their mutual movements, but it is only the World Agent which informs the CodeWalker of the Dummies' movements each time, so that the CodeWalker knows if there is going to occur a collision among them. This happens only after and if the CodeWalker asks the World Agent for permission to move appropriately and in turn, when ready, the World Agent sends the appropriate message where to move. The same things happens to the case of the Dummies accordingly.

This is a necessary process so that the CodeWalker, depending on its personality and experience, decide on which is the appropriate next move to make, with the intention of rendering the arrival at its destination easier. The movement scenarios are the following:

- The CodeWalker moves along the way, meaning up (y+1) - down (y+1) - left (x+1) – right (x-1), aiming to reach its destination.
- On the other hand, the Dummies move completely randomly within their environment.

In this point, it should be noted that the choice to create dummies from the same class is due to the fact that they are identical in this particular application and they perform exactly the same actions towards the accomplishment of a common goal. If each of them had a distinct role

in the problem-solving process, which would also bring about different actions for each dummy, then they would have to be taken from different classes.

#### 4.2.1.1 Perceptions

All the steps described above represent the perceptions agents receive from the environment – their world. In the tables below we will try and recreate how perceptions are linked/connected to behaviours.

| **Perception** | *WAITING_WORLD_INITIALIZE* |
|---|---|
| **Behaviour** | All agents are waiting for the environment of the World Agent to initialize and then proceed to their next move(ment)s. |

| **Perception** | *GOAL_FOUND* |
|---|---|
| **Behaviour** | When the Agent reaches its final destination, the execution of the algorithm no longer continues |

| **Perception** | *DUMMY_REQUEST_MOVE* |
|---|---|
| **Behaviour** | The Dummy asks the World Agent for permission to move |

| **Perception** | *REQUEST_MOVE* |
|---|---|
| **Behaviour** | The Agent asks the World Agent for permission to move |

| **Perception** | *DUMMY_STOP_MOVE* |
|---|---|
| **Behaviour** | The Dummy stops moving as well as soon as the Agent reaches its final destination. |

| | |
|---|---|
| **Perception** | *REQUEST_MOVE_ACCEPT* |
| **Behaviour** | The World Agent grants the Agent permission to move |

| | |
|---|---|
| **Perception** | *REQUEST_MOVE_REJECT* |
| **Behaviour** | The World Agent doesn't allow the Agent to move |

| | |
|---|---|
| **Perception** | *DUMMY_MOVE_TO* |
| **Behaviour** | The World Agent informs/instructs the Dummy as to where to move |

| | |
|---|---|
| **Perception** | *INFORM_NEW_DUMMY* |
| **Behaviour** | We inform the World Agent of the existence of a new Dummy. |

| | |
|---|---|
| **Perception** | *UPDATE_UI* |
| **Behaviour** | When one of the Agents moves, it also notifies the World Agent so that it can refresh the GUI with their new/current locations/positions. |

| | |
|---|---|
| **Perception** | *REQUEST_MOVE_ACCEPT_BUT_MUST_MOVE_DUMMY* |
| **Behaviour** | The World Agent informs the Agent that it can/may move, but that the Dummy also has to move. |

| | |
|---|---|
| **Perception** | *DUMMY_FORCE_MOVE_TO* |
| **Behaviour** | The World Agent forces towards a specific direction. |

| | |
|---|---|
| **Perception** | *WORLD_INITIALIZED* |
| **Behaviour** | The grid initializes |

| | |
|---|---|
| **Perception** | *UPDATE_MOOD* |
| **Behaviour** | The Agent's behavior is updated (once at a time …) |

### 4.2.2 Relations between Personality Trait and Mood

As it has already stated above, the Agent has certain personality traits, namely self-control and positiveness. It also possesses certain moods, having three parameters, meaning happiness, nervousness and disgust. There are two variables in prof. Ricardo Impert Paredes' model for the implementation of rules between personality and mood. These variables are interrelated and refer to the beliefs. The influencer representes the personality trait and the influenced the mood respectively:

- **Influencer:** A feature whose value (directly) affects another features value.

- **Influenced:** The value of influence the influencer exerts on the influenced.

An additional parameter related to the two values and connect them is **degree of influence:**

- **degree of influence:** it is the value the influencer affects the influenced.

Thus we use two functions to express the level of influence of the influencer feature on the influenced:

- **INCREASE:** According to this function, the influencer trait causes an increase in the value of the influenced. This value, which will also be estimated through this function, is the maximum value which can express the degree of influence of the beliefs. The way this function works is as follows: Suppose X and Y are the values of the attributes of our model (for a given X personality trait and a Y mood), (inter)related with the following relation:

*X INCREASE <deg> Y*

"deg" is the degree of influence. Therefore, the new value representing the amount of mood is going to be:

$$< Y' > = < Y > + < X > ⓧ <deg>$$

- **DECREASE:** According to this function, the influencer attribute causes a decrease in the value of the influenced. This value, which will also be estimated/calculated through this function, is the minimum value which can express the degree of influence of the beliefs. The way this function works is as follows: Suppose X and Y are the values of the attributes of our model (for a given X personality trait and a Y mood), (inter)related with the following relation:

*X DECREASE <deg> Y*

"deg" is the degree of influence. Therefore, the new value representing the amount of mood is going to be:

$$< Y' > = < Y > - < X > ⓧ <deg>$$

In our example, then, these are potential rules to make after the collision:

***Rule: influencer WAY of Influence <degree of influence> influenced***

Where the degree of influence can be the following values: **none, some, medium, much, completely**. In our algorithm, the degree of influence is represented by three values: much, some, none.

Accordingly, all the aforementioned stages in our algorithm have been carried out as described here: First, we initialize the personality and the mood from the arguments line with random values from 0 to 1 and we have created two classes: one for the mood and one more for the personality. Within the CodeWalker class we caal for a new behavior, which updates every time the Agent's mood. We have also defined cleared an area within whose limits the Agent expects that a collision will occur if one or more Dummies enter, thus changing its mood since it no longer feels safe. This distance is equal or less than 5 (referring to a distance of 5 cells between them). This results in the following scenarios:

- The Dummy is outside the critical point, so the mood increases (function Increase) in a positive way and the degree of influence is between the values (0.5,1) and can be represented with the expression <much>.

- If it already is within the critical area but is on its way out, there is also a function increase, though not that intense. The degree of influence is between the values (0.15,0.5) and can be represented with the expression <some>.

- The Dummy has just entered the critical area, so there is a function decrease in the Agent's mood, but not dramatic one. The degree of influence is between the values (0,0.15) and can be described with the expression <some>.

- The Dummy is just beside the Agent and a collision is bound to occur, so a rapid function decrease is experienced. The degree of influence is between the values (0.15,0.5) and can be represented with the expression <none>.

Which function is the one we use each time depends on whether the Agent is within the critical area and on the respective features. Furthermore, the Degree of Influence takes a variety of values because it also depends on the Agent's personality and on the area it is in.

We have created as well a GUI in the CodeWalker class to recreate the CodeWalker's mood and make the way it behaves more understandable to the user. In the GUI, we initially created 3 bars (JSlider type) which go up and down oscillate according to the mood changes. More specifically, in the CodeWalker class we call for the class in order to create the feelings frame. The bars represent the mood fluctuations of our Agent during the course of its motion. Finally, we have added textFiled (JTextField type) where we can see the Agent's mood and name.

### 4.2.3 Avoid up Collision

During the agents' movement, it is very likely that a collision will take place. Thus, we are trying to apply different scenarios so as to avoid the collision. A collision is achieved when(ever) the CodeWalker, in its next step, tries to occupy a place already occupied by someone else. As a result, when the CodeWalker wants to claim this kind of place in its next step, the main goal during this phase is to avoid colliding, though in most cases such an event is impossible due to our inability to predict another's movements. We simply try to guess which the dummy's possible movements are. The possible scenarios in case both want the exact same position are the following:

- When the Dummy is already in the position the Agent wants to claim, we let the Agent select the way it will move, giving him priority, as it has both perception and intelligence. As long as the Agent selects the way it is going to move, the Dummy has 2 possible choices on its part. Since its movement is controlled by the World Agent, it will either <<push>> him towards an alternative direction (left or right for example) or it weill remain in the same position.

- On the other hand, as long as the CodeWalker has received all the refreshed movements of the Dummies within the particular environment and observed a collision is about to take place, it will try to decide upon where to move so that the collision can be avoided. As a consequence, its choices could be either move towards a certain direction (up, down, left or right for example) or remain still in the already occupied position without taking any steps. The decision that will prevail is immediately dependent on the perceptions the CodeWalker receives from its environment.

The scenario that will prevail each time depends exclusively on the Agent because it has the perception to decide on what its next step is going to be. The World Agent sends the updated movements of the Dummy and as a result, the CodeWalker has in its memory the way the Dummy moved in previous occasions. As a result, the CodeWalker's decision as to where to move relies on where it thinks the Dummy is going to move, even though it can never be sure as to where the Dummy will really move.

In our code, this algorithm design is developed in the administrator's code, meaning the World Agent's, because it is responsible for the agents' movements as it has already been stated. Therefore, we store the id and location of all agents in a hash map type variable and what we have to do every time is detect whether there is another in the same position. If the answer is yes, then it removes this position and selects one from the rest.

Whenever there is a collision and the dummy can move, we send the following messages DUMMY_FORCE_MOVE_TO και REQUEST_MOVE_ACCEPT_ BUT_MUST_MOVE_DUMMY.If on the other hand there is a collision and we cannot move the dummy elsewhere, we send REQUEST_MOVE_REJECT. If then there is no collision, the CodeWalker moves normally and the World Agent sends it the REQUEST_MOVE_ACCEPT message. A similar detection takes place in case two dummies collide. In such case, we let one of these move randomly toward a direction and the other remains still. The Dummy's movements are decided within the World Agent's class, but they take place within its own class. In the same respect, the CodeWalker decides on its own moves within its own class but these take place within tat of the World Agent.

This check is performed at the level of perceptions, when we receive a message like DUMMY_REQUEST_MOVE or REQUEST_MOVE.

The possible collision scenarios are described in the following figures.

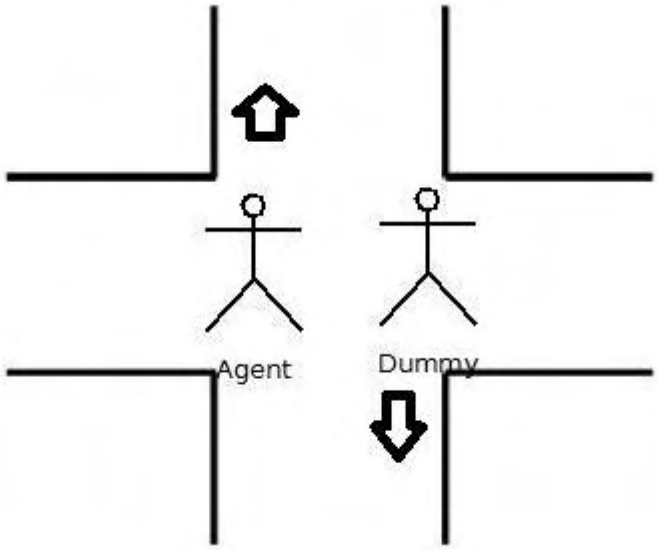



*Fig 5 .No collision. The CodeWalker is in the Critical Area and he avoids the collision*

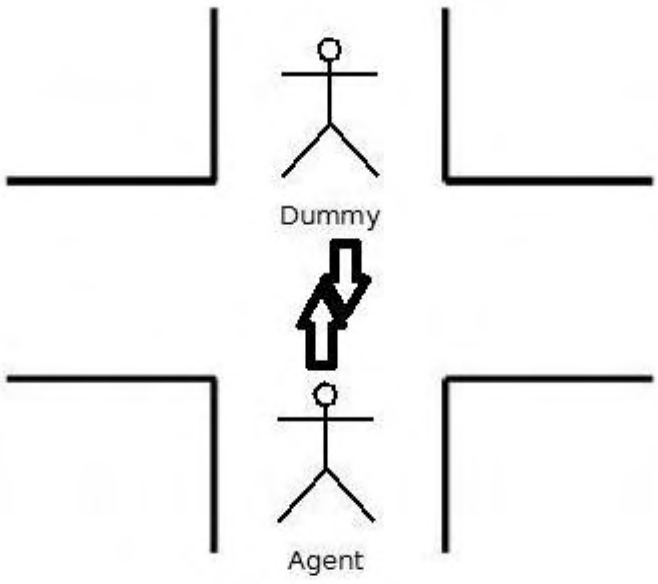



*Fig 6. Collision. The CodeWalker is in the Critical Area and it going to be happened collision*

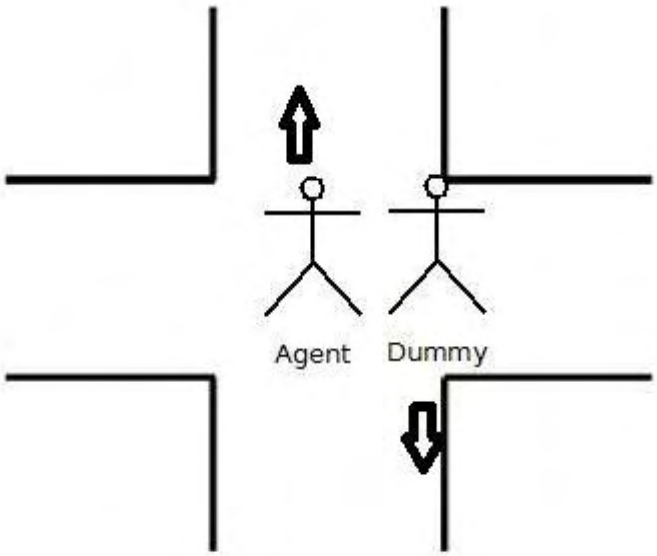



*Fig 7 No Collision. The CodeWalker is in the Critical Area and avoidance of Collision*

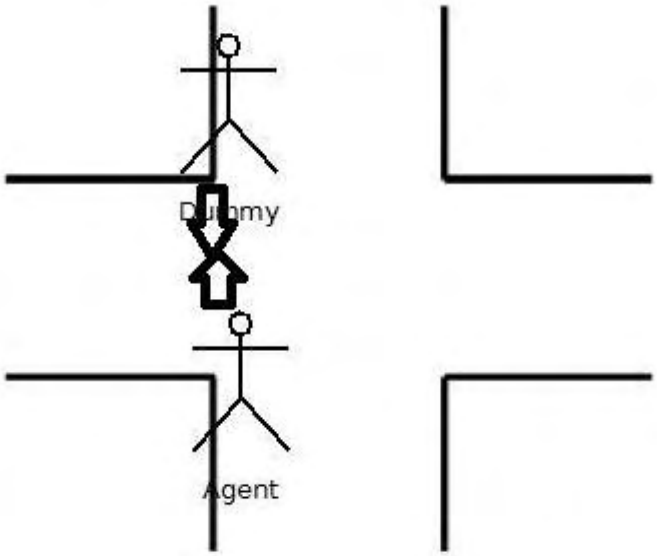



*Fig 8 .Collision. The CodeWalker is in the Critical Area with the Dummy and they go towards each other.*

### 4.2.4 Description of the Action

In our model, COGNITIVA, the interpreter initially takes as entry the position of each agent, interprets it according to the moods, personality, beliefs and acts-reacts accordingly(reactive level). If it acts appropriately, we judge if the result of this reaction is expected or desired. The results of the action are :

- The mood is a feature immediately by the Agent's reaction. More specifically, every mood attribute may increase or decrease depending on whether there is a collision or not . The mood plays a decisive role in the Agent's reaction.

- The change of the direction of the movement. While the agent had an initial plan regarding its direction, this changes whenever there is a collision. This results in the change of the initial movement plan.

- If the Agent avoids colliding, then it will be happy and smile, whereas it will feel angry in case of a collision.

Therefore, according to the way the CodeWalker reacts, there are two variables describing its motion:

- **Expected:** Based on the perception and mood, the CodeWalker judges whether the reaction was expected, meaning if it anticipated the way the Dummy moved so that it could react accordingly in its effort to avoid the collision. If for example they collided and this was not expected, the value of this variable decreases, while it increases in the opposite scenario.

- **Desired:** The CodeWalker considers of its motion is desired or not. For example, if there is a collision, this is not a desired scenario and the value of this variable decreases, whereas it increases as well in the opposite one.

In a previous paragraph we elaborated on haw we design the change in the mood within the code. As for the opposite case, meaning how the direction of the movement changes, it is performed within each Agent's code individually. To be more precise, it is in the DummyAgent class where we design the moveTo function, the one which pushes the Agent towards its new position. When it receives the message DUMMY_FORCE_MOVE_TO or DUMMY_MOVE_TO , it calls for this particular function which stores the dummy's new location. Similarly, within the CodeWalker's class, we develop a Location type moveTo variable, where we have created all the appropriate methods to move the CodeWalker through this class.

In a previous section, we also that the expectations and the mood are inextricably linked to one another and we developed our algorithm as follows: As we have previously stated, we have determined a certain distance. If the Dummy enters its limits, meaning it approaches the

Agent, the latter expects there will be a collision between them. As a result, it changes its mood according to the rules we described in a previous section, but it simultaneously changes its expectations. More precisely, 4 cases take place in this area, and the two variables (expected and desired) are affected by these. In our algorithm we defined our variables as either positive (expected, desired) or negative (unexpected, undesired), in an effort to simplify the values of the variables. More specifically, whenever the CodeWalker is placed in the grid, these variables initialize based on the circumstances of the environment (for example if there are Dummies, in what distance, how many dummies are next to it…). If the expected variable is positive, we expect a collision, whereas we do not expect one in case it is negative. If the Desired variable is positive too, the existing situation is the desired/desirable one for the Agent, while it is not in the case of a negative value. Additionally, there may be more than just one Dummy in that particular space, so the possible scenarios are the following:

- A Dummy enters this area but is just on the borders, so the CodeWalker's mood is starting to vhange because it believes a collision is going to take place. The Expected value is positive and the Desired negative.

- The Dummy is already inside the critical area and is really close to the CodeWalker, so the CodeWalker's mood and expectations change dramatically. Positive Expected and negative Desired.

- While the Dummy is inside this area, it is starting to getting further, so there is a slight positive change in the mood variables. Negative Expected and positive Desired.

- The Dummy is getting away from the CodeWalker's vicinity, but is still within the confined area, so a collision is still likely and the CodeWalker cannot rest. Negative Expected and positive Desired.

Reaching this point, I feel useful to point out the use of a helping class (Helper) to develop the Expected (value). This means that, in this class, we took three positions (the CodeWalker's actual position, the Dummy's position and the final destination of CodeWalker) and we estimate the deviation. If the three positions together with the deviation have a true result, this means the dummies are aligned with the CodeWalker and a collision is expected. The opposite applies in case it is false. Then we estimate/calculate the ExpectationDelta. If it is little, the CodeWalker has just entered the critical area and it increases as the Dummy approaches the CodeWalker.

In the table below we can observe the connection between mood and expectation.

| | ***Desirable*** | ***Not Desirable*** |
|---|---|---|
| ***Expected*** | Positiveness Increase much happiness and decrease much nervousness and disgust.(reaction = smile)<br><br>Self-control Increase some happiness and decrease some nervousness and disgust.(reaction = smile) | Positiveness Decrease some Happiness and Increase some Nervousness and Disgust.(reaction = sad<br><br>Self-Control not change Happiness and not change nervousness and disgust. (reaction = sad) |
| ***Not Expected*** | Positiveness Increase much Happiness and Decrease much Nervousness and Disgust(reaction = surprise)<br><br>Self-Control Increase some Happiness and Decrease some Nervousness and Disgust(reaction = surprise) | Positiveness Decrease some Happiness and Increase some Nervousness and Disgust(reaction = angry)<br><br>Self-Control Decrease much Happiness and Increase much Nervousness and Disgust(reaction = angry) |

*Table 2: Display of the relation between expectations and mood*

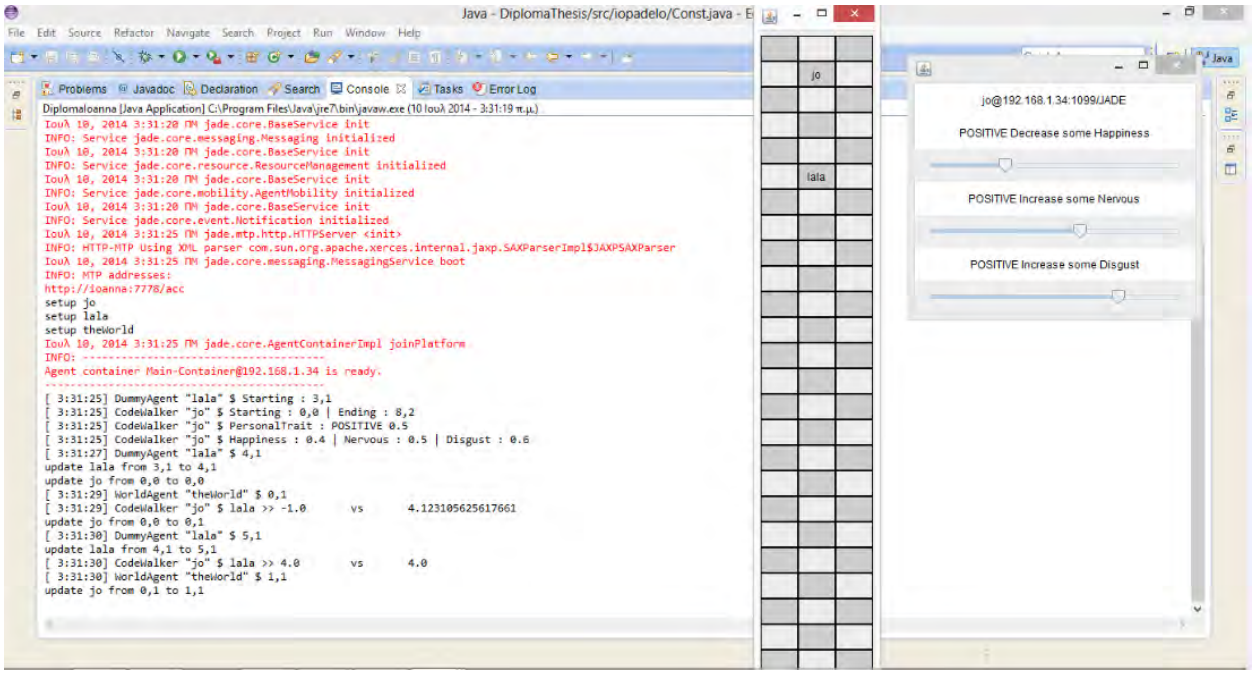
# 5.EVALUATIONS CASES

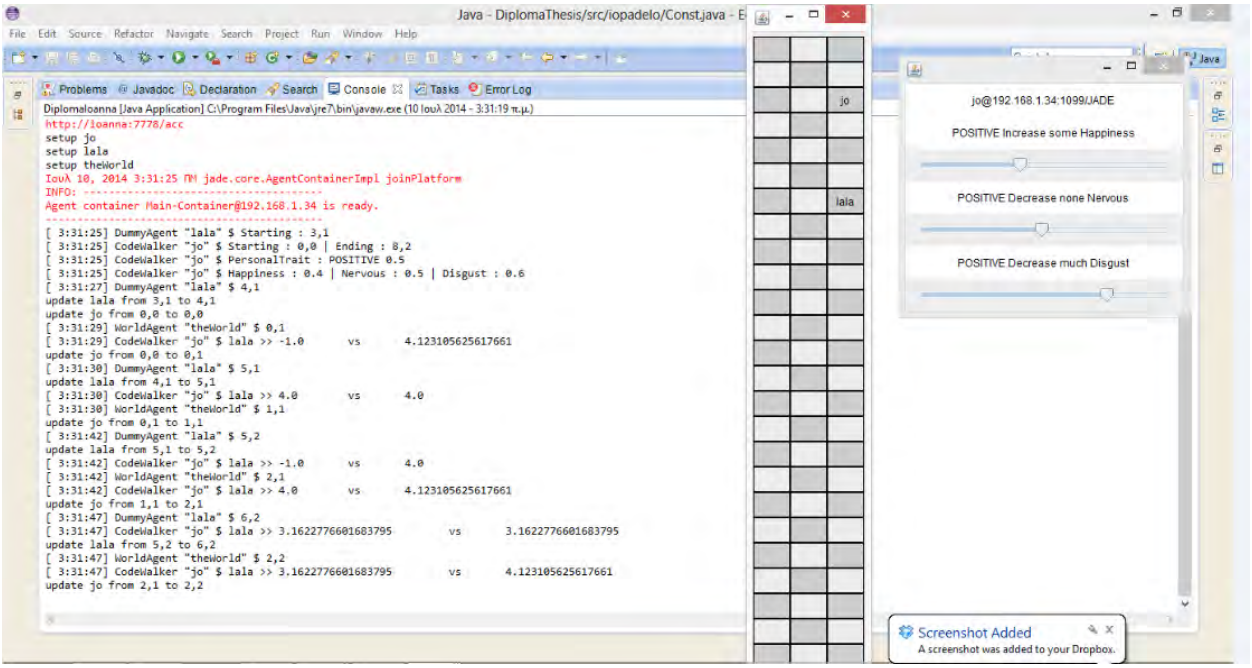
I have applied experiments of two sorts. In the first one, I decided that our CodeWalker has just one personality trait, the positiveness. You will see in detail the way the CodeWalker reacts and makes decisions in two cases, when there is only one Dummy and when there are several.

### 1 EXPERIMENT

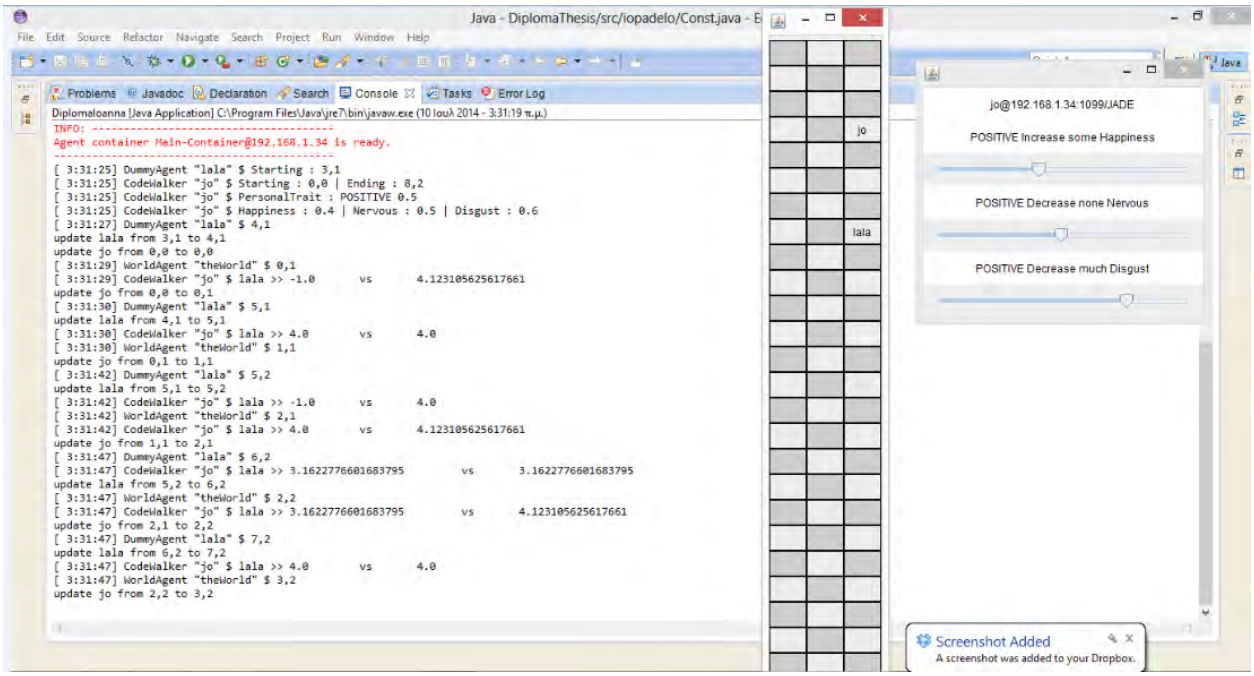
**1st case**: One Dummy. We initialized the positiveness at 0.5, happiness at 0.4, nervous(ness) at 0.5 and disgust at 0.6. The Agent is in the position 0,0 moving towards 15,2.

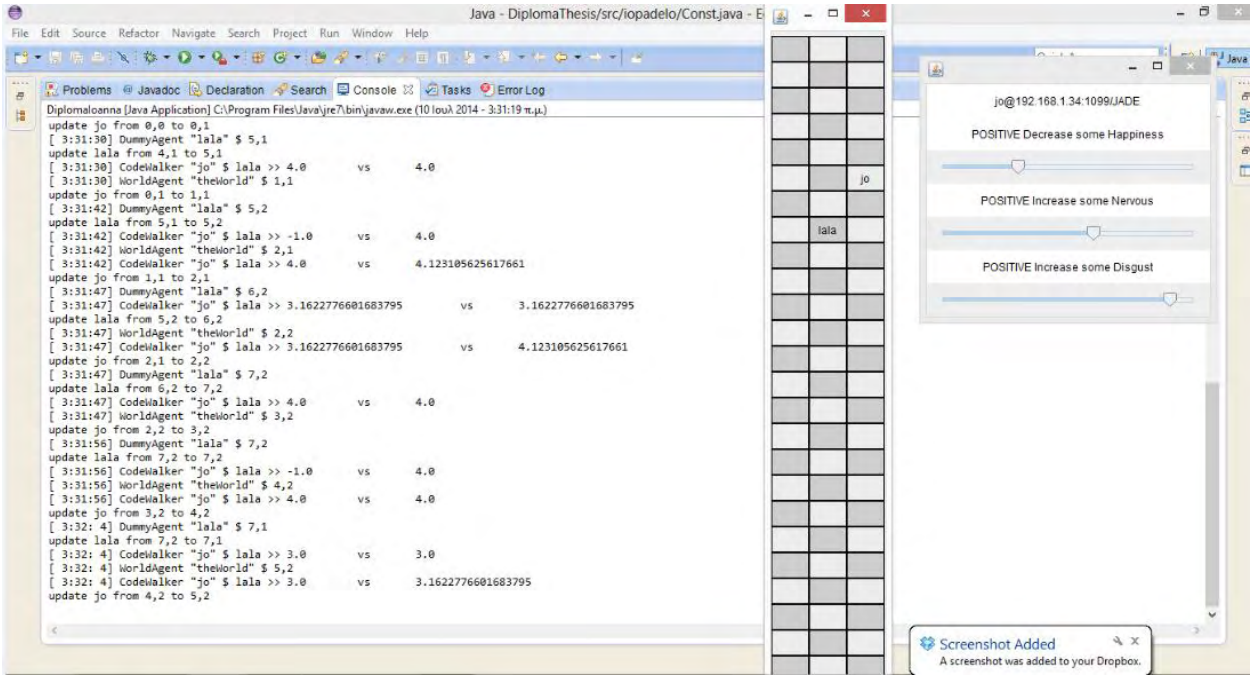
In this screenshot, we see the CodeWalker named Jo moving towards to the Dummy (lala). We see also how the values of the variable Mood change depending on the Personality. We represent these moves in a grid, so we implement a Gui. We also implement a second Gui for the moods. Finally we print some messages from the console: how the CodeWalker moves and how he avoids the collision.



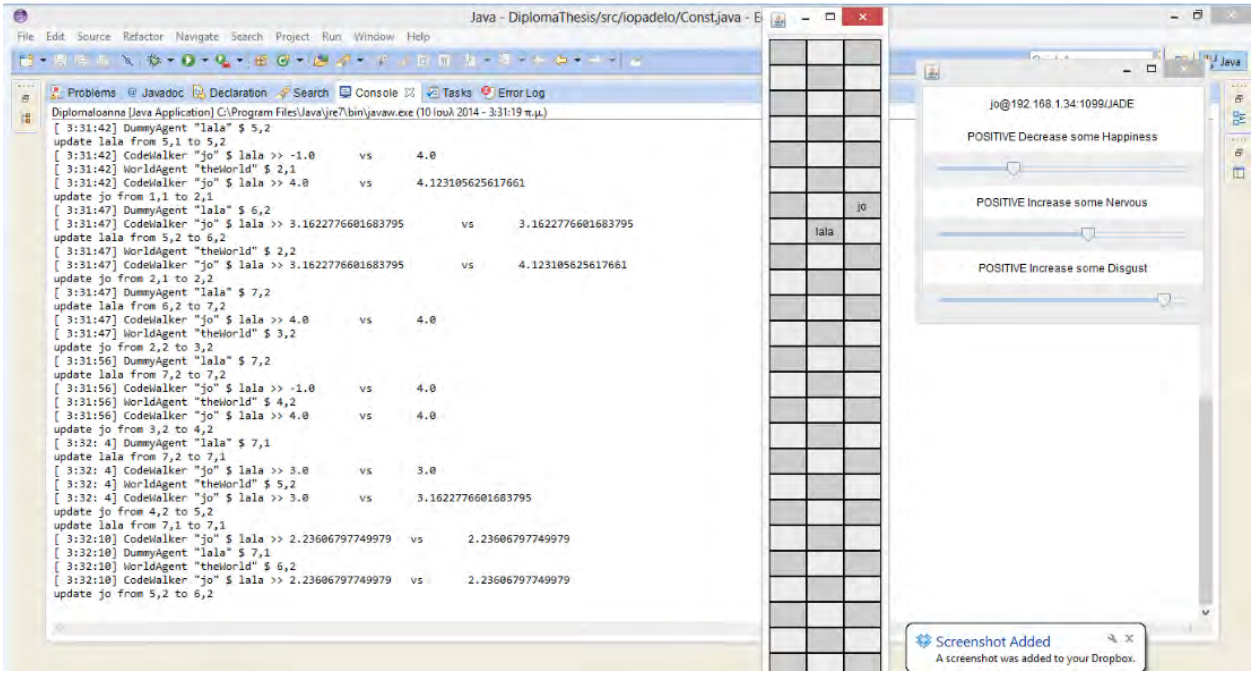


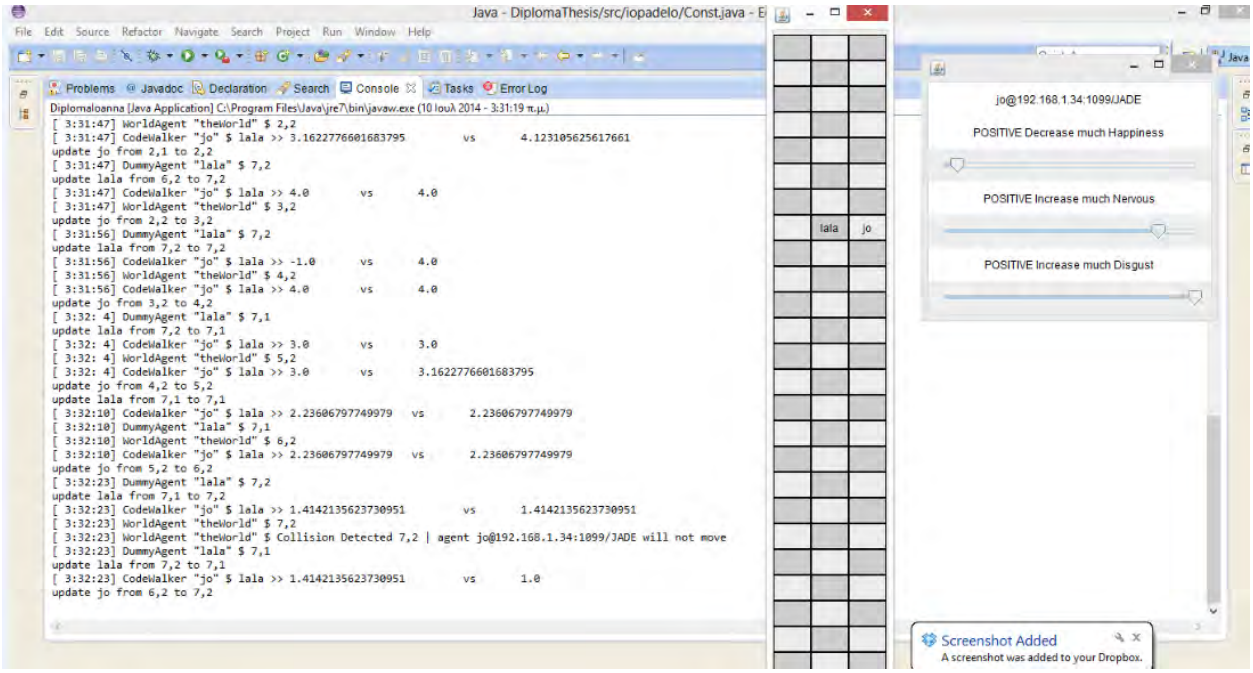

In these screenshot, we observe that the Dummy is going out of the Critical Area, so The CodeWalker Increase his happiness and Decrease his nervous and disgust.






In these screenshot, the Dummy approach the CodeWalker and we see how the mood changes again.

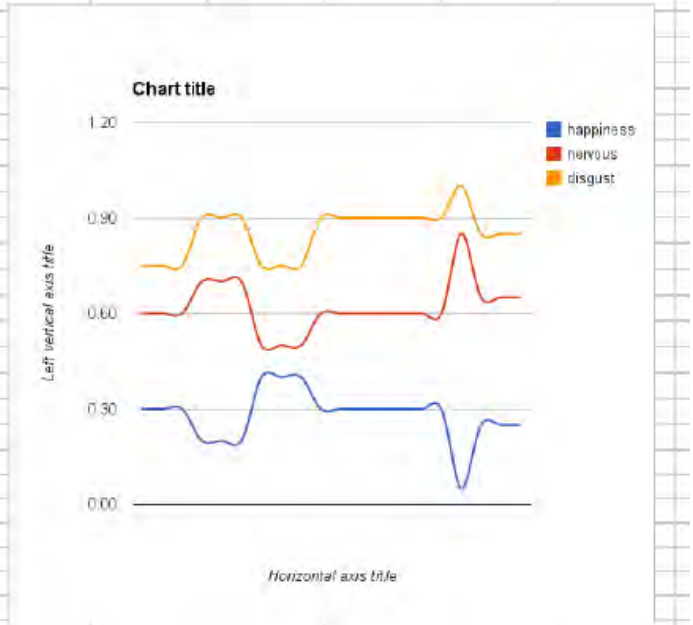





In the last screenshot, the Dummy is next to CodeWalker, so it is very possible to be happened collision, so he Decrease his happiness and he Increase the nervous and disgust.

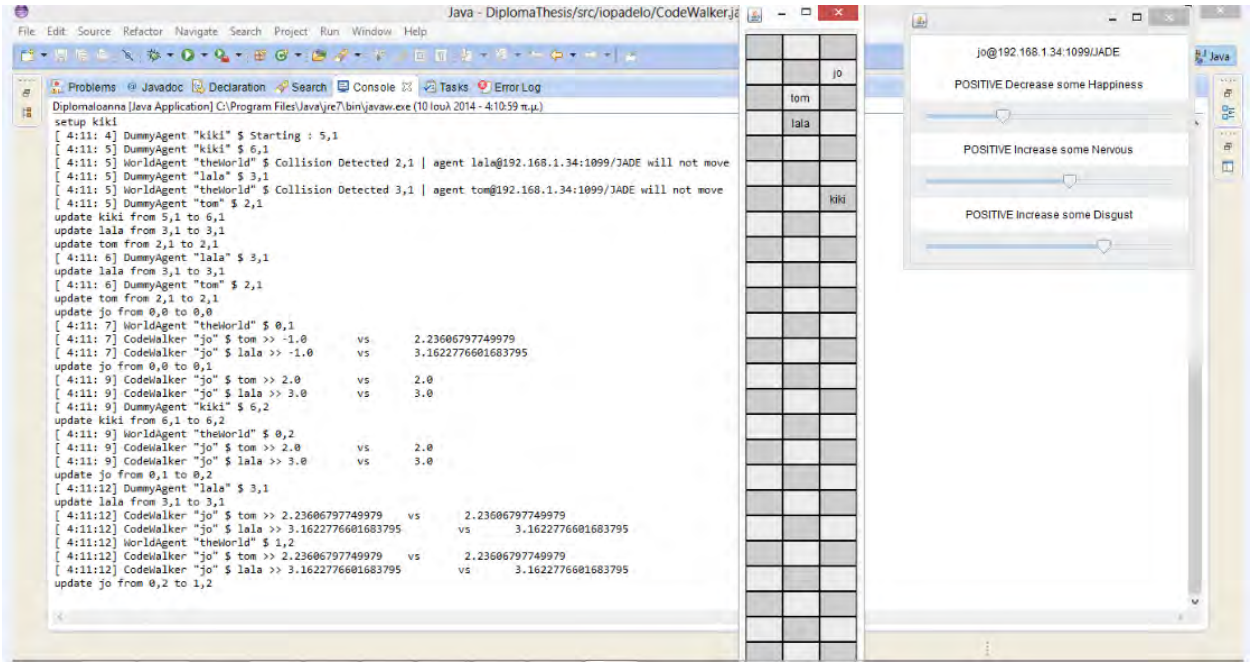
**Datagram of Moods:** How the mood changes in time.

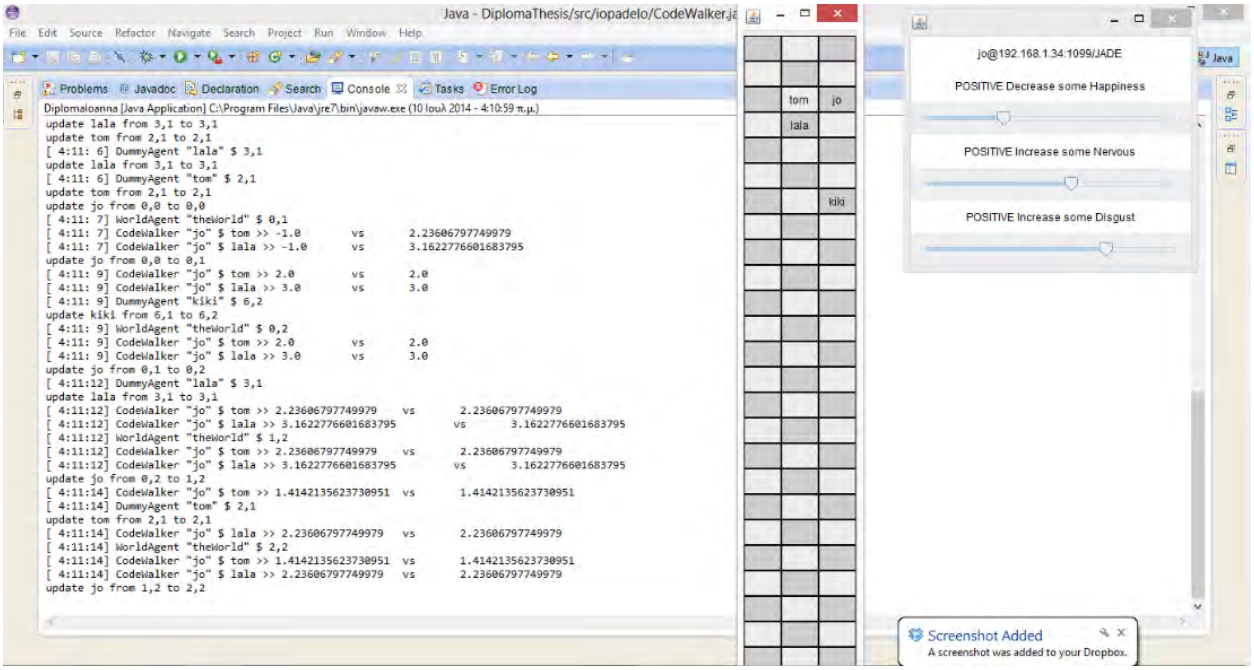
| happiness | nervous | disgust |
|---|---|---|
| 0,30 | 0,60 | 0,75 |
| 0,30 | 0,60 | 0,75 |
| 0,30 | 0,60 | 0,75 |
| 0,20 | 0,70 | 0,90 |
| 0,20 | 0,70 | 0,90 |
| 0,20 | 0,70 | 0,90 |
| 0,40 | 0,50 | 0,75 |
| 0,40 | 0,50 | 0,75 |
| 0,40 | 0,50 | 0,75 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,30 | 0,60 | 0,90 |
| 0,05 | 0,85 | 1,00 |
| 0,25 | 0,65 | 0,85 |
| 0,25 | 0,65 | 0,85 |
| 0,25 | 0,65 | 0,85 |

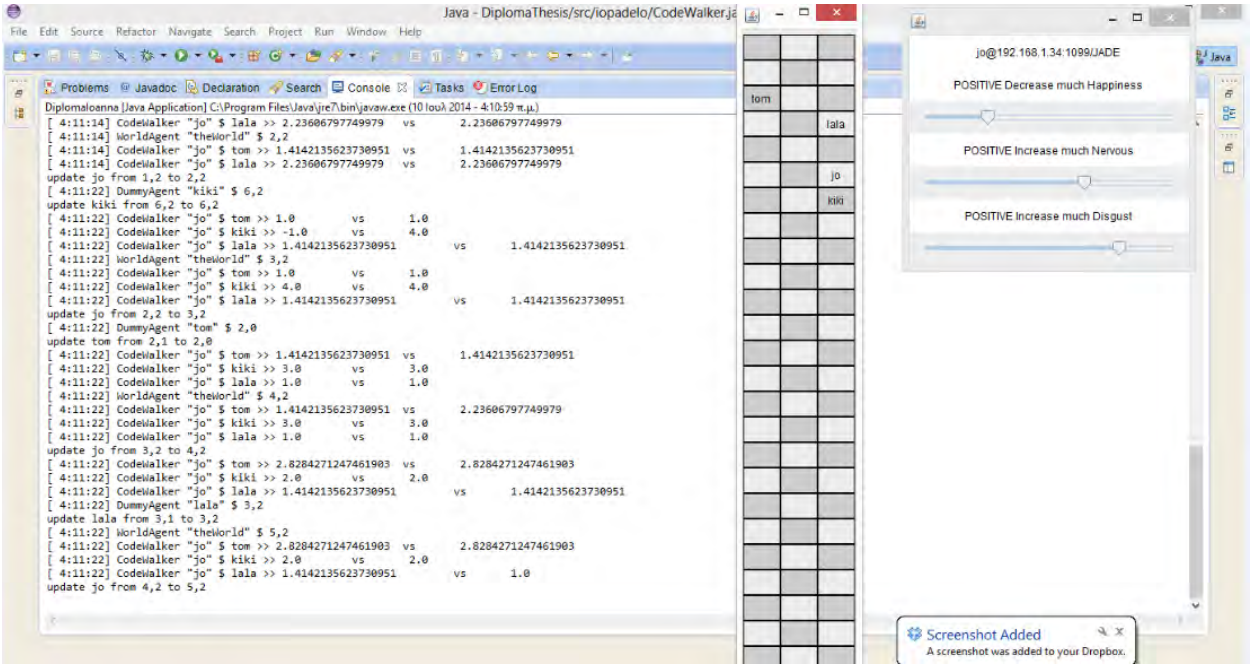


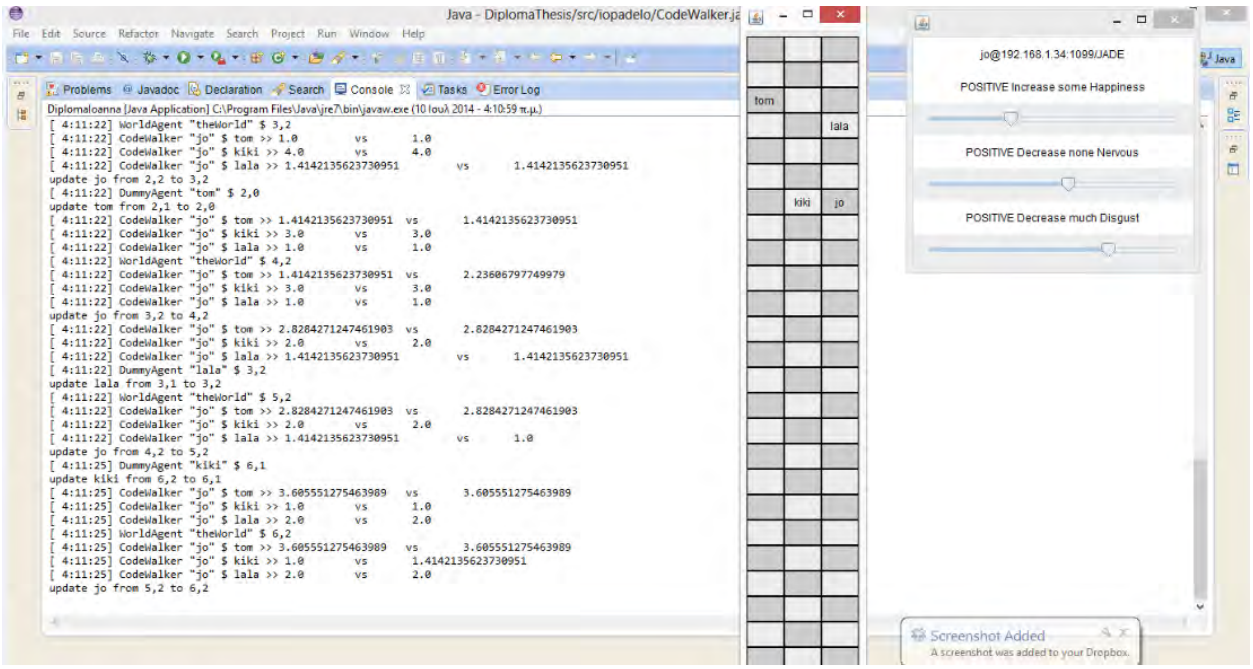
**2nd case:** Three Dummies. We initialized the positiveness at 0.2, happiness at 0.4, nervous(ness) at 0.5 and disgust at 0.6. The Agent is in the position 0.0 heading for the 8.2.

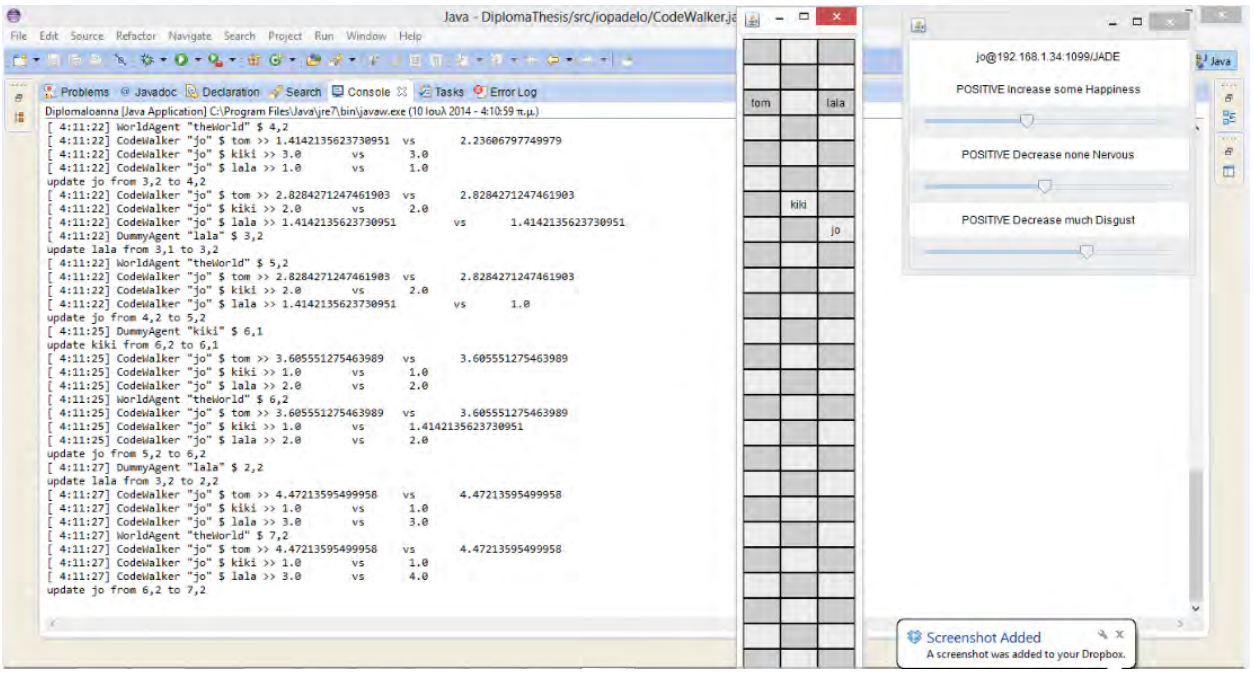
In these screenshot, we represent the environment as a grid and how the mood changes in a Gui. The CodeWalker named jo walks in the Critical Area with three Dummies. So we can see how he changes his mood during his movement. We can also see how he avoids the collision.

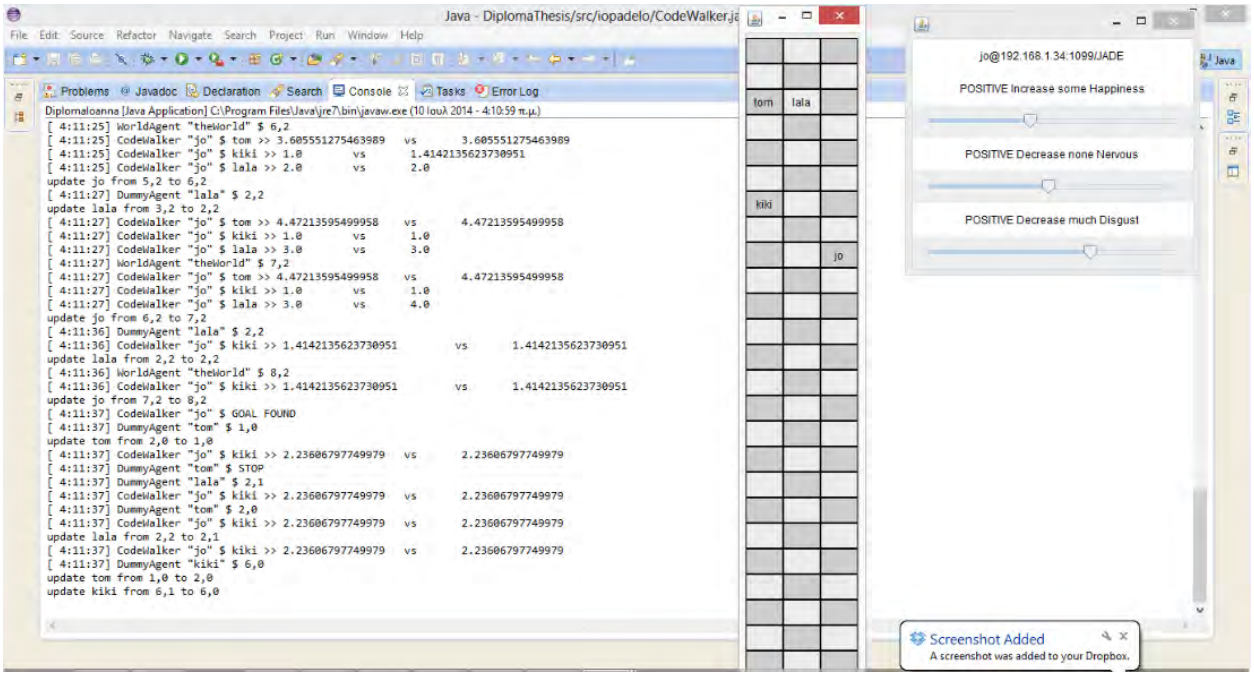



Java - DiplomaThesis/src/iopadelo/CodeWalker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:10:59 π.μ.)
[ 4:11:14] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
[ 4:11:14] WorldAgent "theWorld" $ 2,2
[ 4:11:14] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:14] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
update jo from 1,2 to 2,2
[ 4:11:22] DummyAgent "kiki" $ 6,2
update kiki from 6,2 to 6,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.0 vs 1.0
[ 4:11:22] CodeWalker "jo" $ kiki >> -1.0 vs 4.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] WorldAgent "theWorld" $ 3,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.0 vs 1.0
[ 4:11:22] CodeWalker "jo" $ kiki >> 4.0 vs 4.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
update jo from 2,2 to 3,2
[ 4:11:22] DummyAgent "tom" $ 2,0
update tom from 2,1 to 2,0
[ 4:11:22] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] CodeWalker "jo" $ kiki >> 3.0 vs 3.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:11:22] WorldAgent "theWorld" $ 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 2.23606797749979
[ 4:11:22] CodeWalker "jo" $ kiki >> 3.0 vs 3.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 3,2 to 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] DummyAgent "lala" $ 3,2
update lala from 3,1 to 3,2
[ 4:11:22] WorldAgent "theWorld" $ 5,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.0
update jo from 4,2 to 5,2
tom
lala
jo
kiki
jo@192.168.1.34:1099/JADE
POSITIVE Decrease much Happiness
POSITIVE Increase much Nervous
POSITIVE Increase much Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:10:59 π.μ.)
[ 4:11:22] WorldAgent "theWorld" $ 3,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.0 vs 1.0
[ 4:11:22] CodeWalker "jo" $ kiki >> 4.0 vs 4.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
update jo from 2,2 to 3,2
[ 4:11:22] DummyAgent "tom" $ 2,0
update tom from 2,1 to 2,0
[ 4:11:22] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] CodeWalker "jo" $ kiki >> 3.0 vs 3.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:11:22] WorldAgent "theWorld" $ 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 2.23606797749979
[ 4:11:22] CodeWalker "jo" $ kiki >> 3.0 vs 3.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 3,2 to 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] DummyAgent "lala" $ 3,2
update lala from 3,1 to 3,2
[ 4:11:22] WorldAgent "theWorld" $ 5,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.0
update jo from 4,2 to 5,2
[ 4:11:25] DummyAgent "kiki" $ 6,1
update kiki from 6,2 to 6,1
[ 4:11:25] CodeWalker "jo" $ tom >> 3.605551275463989 vs 3.605551275463989
[ 4:11:25] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:25] CodeWalker "jo" $ lala >> 2.0 vs 2.0
[ 4:11:25] WorldAgent "theWorld" $ 6,2
[ 4:11:25] CodeWalker "jo" $ tom >> 3.605551275463989 vs 3.605551275463989
[ 4:11:25] CodeWalker "jo" $ kiki >> 1.0 vs 1.4142135623730951
[ 4:11:25] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 5,2 to 6,2
tom
lala
kiki
jo
jo@192.168.1.34:1099/JADE
POSITIVE Increase some Happiness
POSITIVE Decrease none Nervous
POSITIVE Decrease much Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:10:59 π.μ.)
[ 4:11:22] WorldAgent "theWorld" $ 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 1.4142135623730951 vs 2.23606797749979
[ 4:11:22] CodeWalker "jo" $ kiki >> 3.0 vs 3.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 3,2 to 4,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:11:22] DummyAgent "lala" $ 3,2
update lala from 3,1 to 3,2
[ 4:11:22] WorldAgent "theWorld" $ 5,2
[ 4:11:22] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:11:22] CodeWalker "jo" $ kiki >> 2.0 vs 2.0
[ 4:11:22] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.0
update jo from 4,2 to 5,2
[ 4:11:25] DummyAgent "kiki" $ 6,1
update kiki from 6,2 to 6,1
[ 4:11:25] CodeWalker "jo" $ tom >> 3.605551275463989 vs 3.605551275463989
[ 4:11:25] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:25] CodeWalker "jo" $ lala >> 2.0 vs 2.0
[ 4:11:25] WorldAgent "theWorld" $ 6,2
[ 4:11:25] CodeWalker "jo" $ tom >> 3.605551275463989 vs 3.605551275463989
[ 4:11:25] CodeWalker "jo" $ kiki >> 1.0 vs 1.4142135623730951
[ 4:11:25] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 5,2 to 6,2
[ 4:11:27] DummyAgent "lala" $ 2,2
update lala from 3,2 to 2,2
[ 4:11:27] CodeWalker "jo" $ tom >> 4.47213595499958 vs 4.47213595499958
[ 4:11:27] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:27] CodeWalker "jo" $ lala >> 3.0 vs 3.0
[ 4:11:27] WorldAgent "theWorld" $ 7,2
[ 4:11:27] CodeWalker "jo" $ tom >> 4.47213595499958 vs 4.47213595499958
[ 4:11:27] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:27] CodeWalker "jo" $ lala >> 3.0 vs 4.0
update jo from 6,2 to 7,2
tom
lala
kiki
jo
jo@192.168.1.34:1099/JADE
POSITIVE Increase some Happiness
POSITIVE Decrease none Nervous
POSITIVE Decrease much Disgust
Java
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.ja
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:10:59 π.μ.)
[ 4:11:25] WorldAgent "theWorld" $ 6,2
[ 4:11:25] CodeWalker "jo" $ tom >> 3.605551275463989 vs 3.605551275463989
[ 4:11:25] CodeWalker "jo" $ kiki >> 1.0 vs 1.4142135623730951
[ 4:11:25] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 5,2 to 6,2
[ 4:11:27] DummyAgent "lala" $ 2,2
update lala from 3,2 to 2,2
[ 4:11:27] CodeWalker "jo" $ tom >> 4.47213595499958 vs 4.47213595499958
[ 4:11:27] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:27] CodeWalker "jo" $ lala >> 3.0 vs 3.0
[ 4:11:27] WorldAgent "theWorld" $ 7,2
[ 4:11:27] CodeWalker "jo" $ tom >> 4.47213595499958 vs 4.47213595499958
[ 4:11:27] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:11:27] CodeWalker "jo" $ lala >> 3.0 vs 4.0
update jo from 6,2 to 7,2
[ 4:11:36] DummyAgent "lala" $ 2,2
[ 4:11:36] CodeWalker "jo" $ kiki >> 1.4142135623730951 vs 1.4142135623730951
update lala from 2,2 to 2,2
[ 4:11:36] WorldAgent "theWorld" $ 8,2
[ 4:11:36] CodeWalker "jo" $ kiki >> 1.4142135623730951 vs 1.4142135623730951
update jo from 7,2 to 8,2
[ 4:11:37] CodeWalker "jo" $ GOAL FOUND
[ 4:11:37] DummyAgent "tom" $ 1,0
update tom from 2,0 to 1,0
[ 4:11:37] CodeWalker "jo" $ kiki >> 2.23606797749979 vs 2.23606797749979
[ 4:11:37] DummyAgent "tom" $ STOP
[ 4:11:37] DummyAgent "lala" $ 2,1
[ 4:11:37] CodeWalker "jo" $ kiki >> 2.23606797749979 vs 2.23606797749979
[ 4:11:37] DummyAgent "tom" $ 2,0
[ 4:11:37] CodeWalker "jo" $ kiki >> 2.23606797749979 vs 2.23606797749979
update lala from 2,2 to 2,1
[ 4:11:37] CodeWalker "jo" $ kiki >> 2.23606797749979 vs 2.23606797749979
[ 4:11:37] DummyAgent "kiki" $ 6,0
update tom from 1,0 to 2,0
update kiki from 6,1 to 6,0
tom
lala
kiki
jo
jo@192.168.1.34:1099/JADE
POSITIVE Increase some Happiness
POSITIVE Decrease none Nervous
POSITIVE Decrease much Disgust
Java
Screenshot Added
A screenshot was added to your Dropbox.

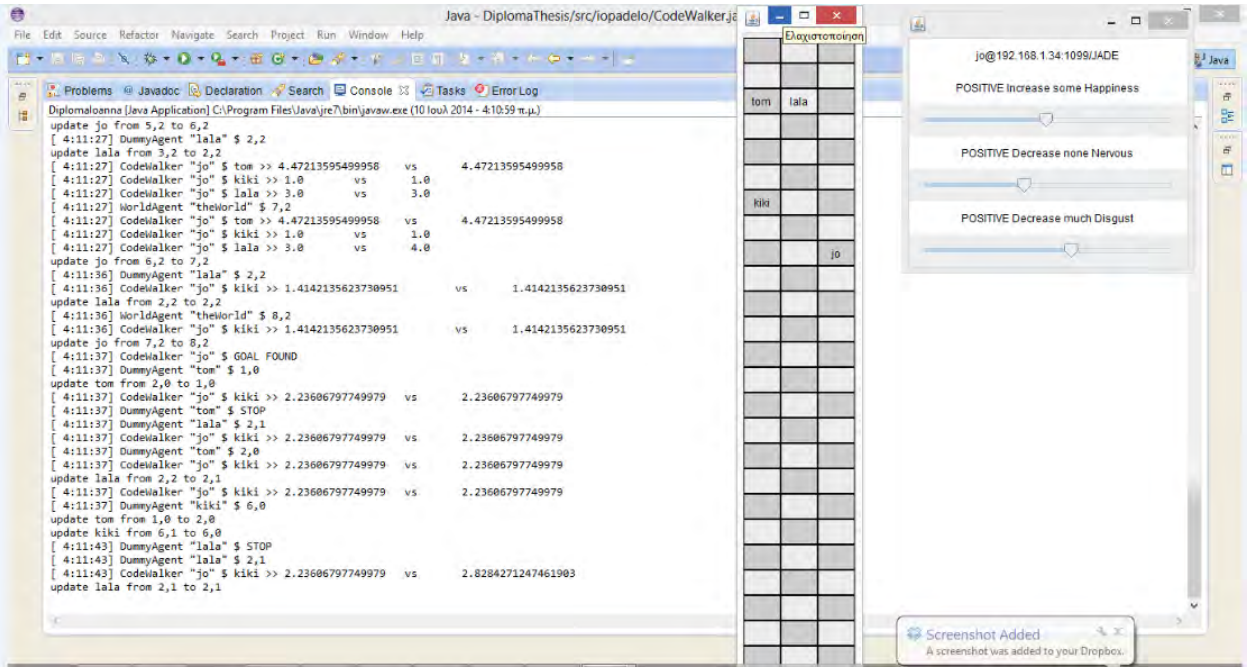
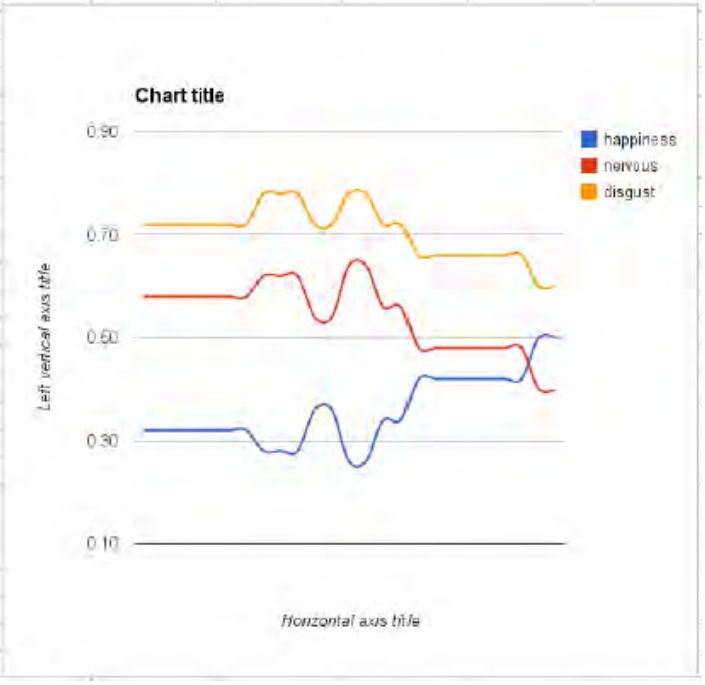



Diagram Of Moods: How the mood changes in time.

| happiness | nervous | disgust |
|---|---|---|
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,32 | 0,58 | 0,72 |
| 0,28 | 0,62 | 0,78 |
| 0,28 | 0,62 | 0,78 |
| 0,28 | 0,62 | 0,78 |
| 0,36 | 0,54 | 0,72 |
| 0,36 | 0,54 | 0,72 |
| 0,26 | 0,64 | 0,78 |
| 0,26 | 0,64 | 0,78 |
| 0,34 | 0,56 | 0,72 |
| 0,34 | 0,56 | 0,72 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,42 | 0,48 | 0,66 |
| 0,50 | 0,40 | 0,60 |
| 0,50 | 0,40 | 0,60 |

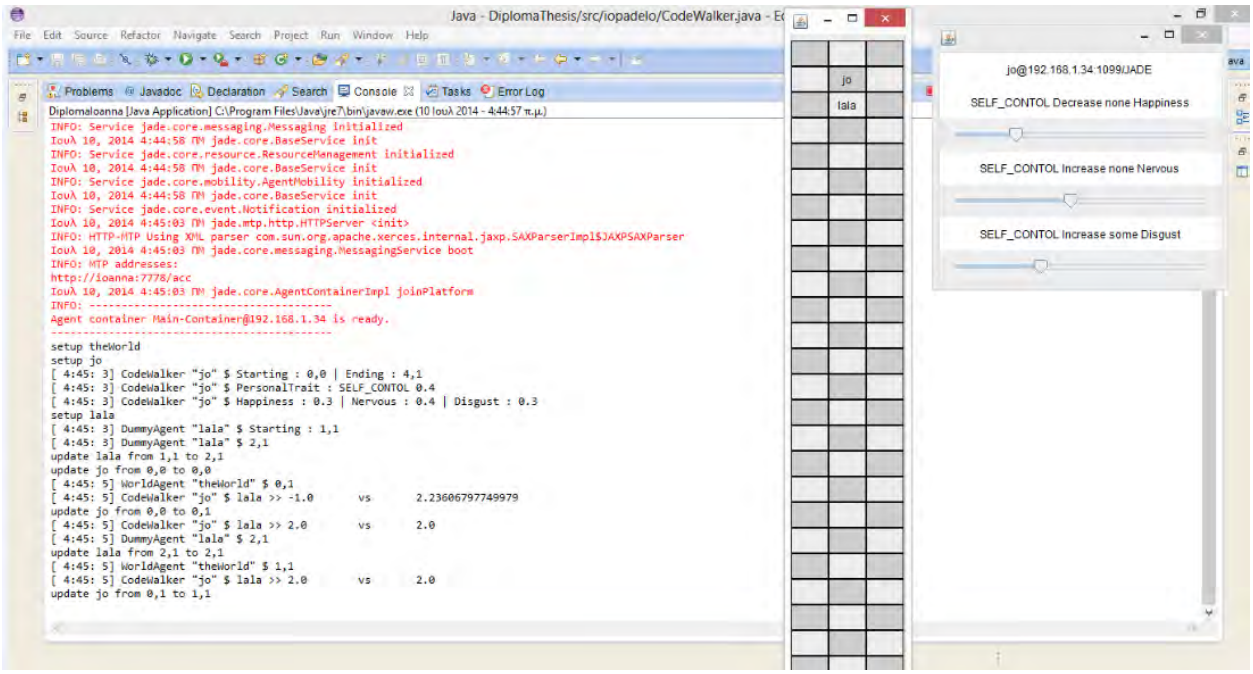


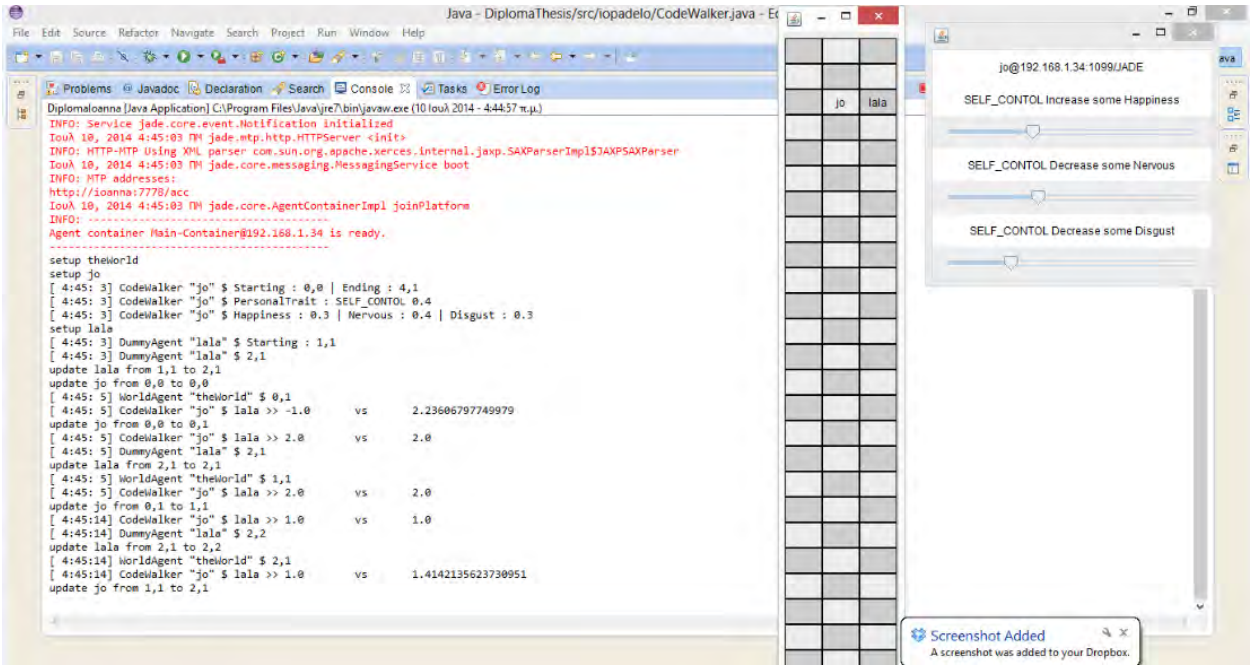
## 2nd Experiment

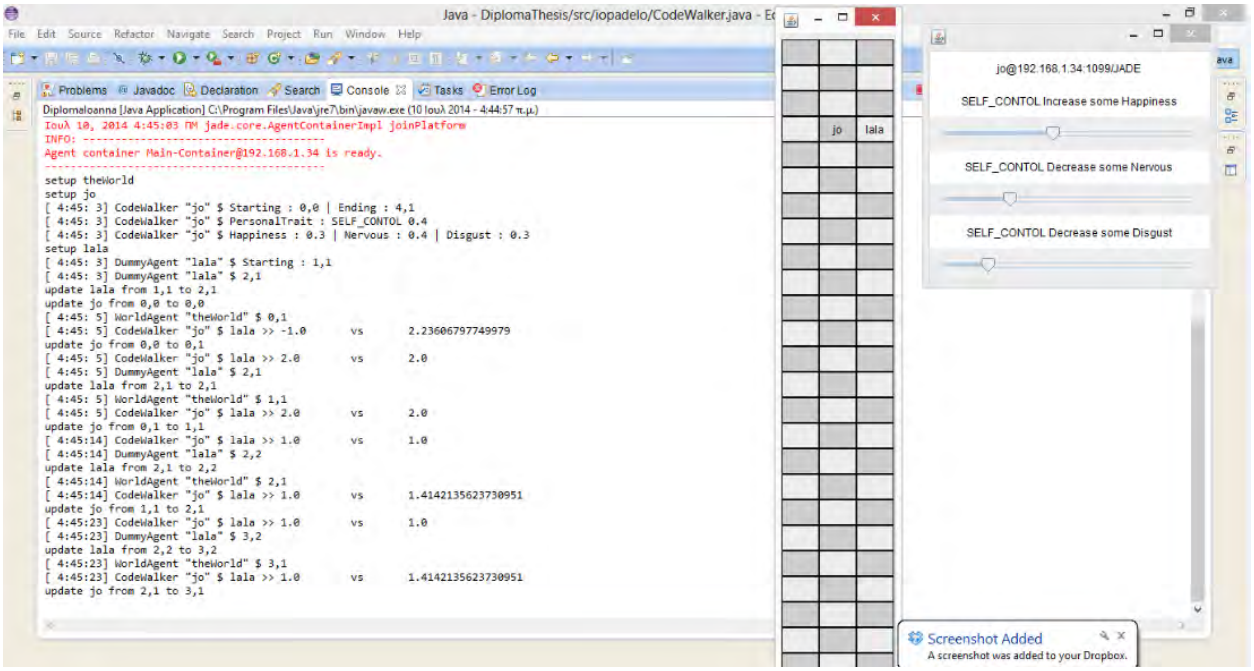
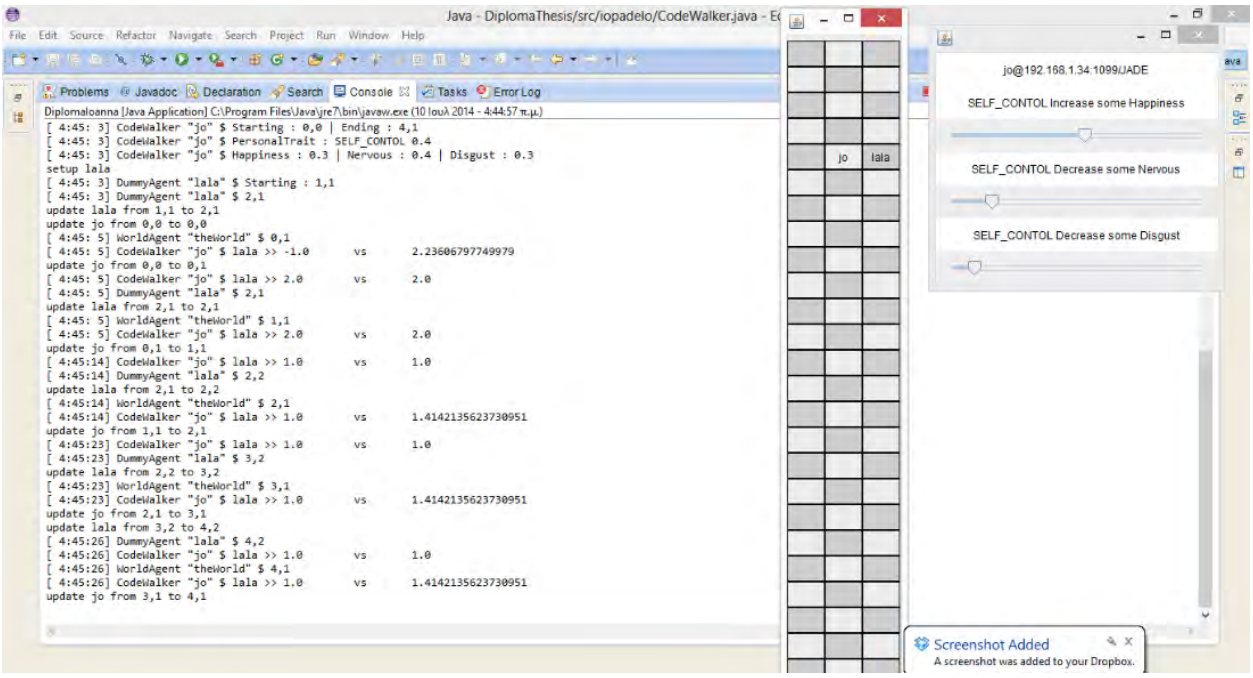
:In this experiment, we examine CodeWalker with personality trait: self_control.

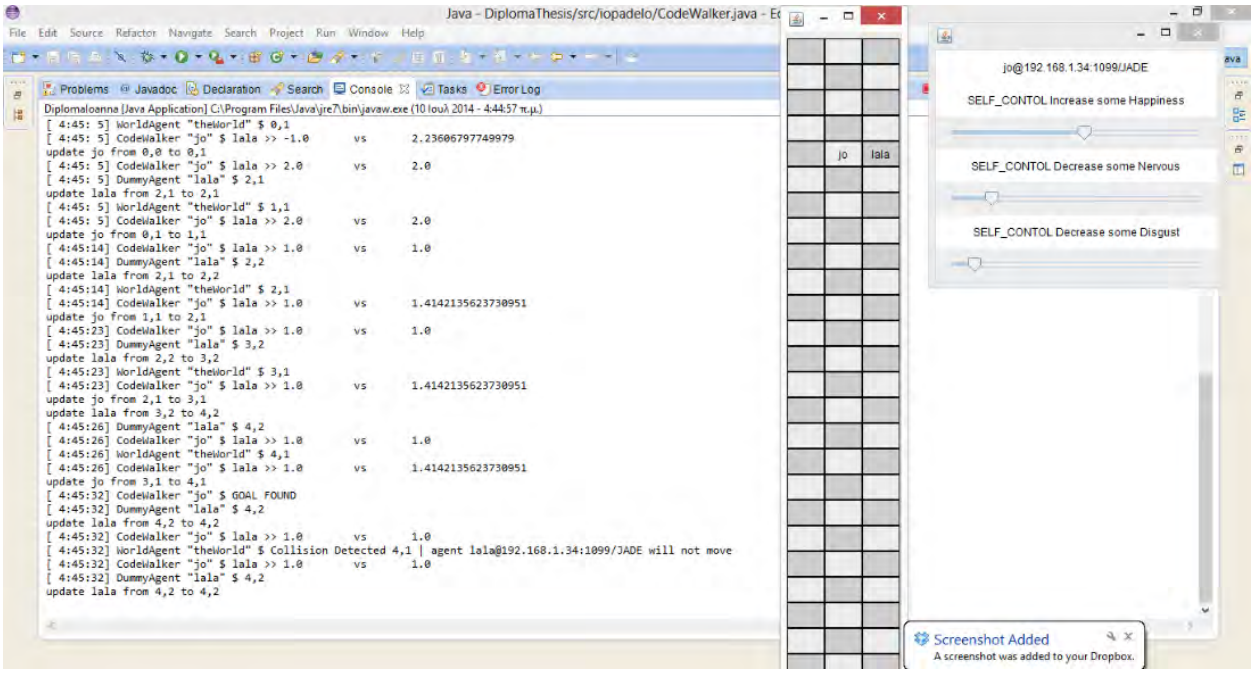
**1st case:** one dummy. We initialized the Self_control variable at 0.4, happiness at 0.3, nervous(ness) at 0.4 and disgust at 0.3. The Agent is in the position 0,0 heading for the 4,1.

In the following screenshots, we represent the movement of the agents in a grid. The CodeWalker named Jo is walking across this grid and one dummy approach him. So we observe in an other gui how he changes the mood during his movement. We also print message in the left side how he avoid the collision.




Java - DiplomaThesis/src/iopadelo/CodeWalker.java - Ec
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:44:57 π.μ.)
Ιουλ 10, 2014 4:45:03 ΠΜ jade.core.AgentContainerImpl joinPlatform
INFO: -----------------------------------
Agent container Main-Container@192.168.1.34 is ready.
-----------------------------------
setup theWorld
setup jo
[ 4:45: 3] CodeWalker "jo" $ Starting : 0,0 | Ending : 4,1
[ 4:45: 3] CodeWalker "jo" $ PersonalTrait : SELF_CONTOL 0.4
[ 4:45: 3] CodeWalker "jo" $ Happiness : 0.3 | Nervous : 0.4 | Disgust : 0.3
setup lala
[ 4:45: 3] DummyAgent "lala" $ Starting : 1,1
[ 4:45: 3] DummyAgent "lala" $ 2,1
update lala from 1,1 to 2,1
update jo from 0,0 to 0,0
[ 4:45: 5] WorldAgent "theWorld" $ 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> -1.0 vs 2.23606797749979
update jo from 0,0 to 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
[ 4:45: 5] DummyAgent "lala" $ 2,1
update lala from 2,1 to 2,1
[ 4:45: 5] WorldAgent "theWorld" $ 1,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 0,1 to 1,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:14] DummyAgent "lala" $ 2,2
update lala from 2,1 to 2,2
[ 4:45:14] WorldAgent "theWorld" $ 2,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 1,1 to 2,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:23] DummyAgent "lala" $ 3,2
update lala from 2,2 to 3,2
[ 4:45:23] WorldAgent "theWorld" $ 3,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 2,1 to 3,1
jo
lala
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease some Disgust
Screenshot Added
A screenshot was added to your Dropbox.

Java - DiplomaThesis/src/iopadelo/CodeWalker.java - Ec
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:44:57 π.μ.)
[ 4:45: 3] CodeWalker "jo" $ Starting : 0,0 | Ending : 4,1
[ 4:45: 3] CodeWalker "jo" $ PersonalTrait : SELF_CONTOL 0.4
[ 4:45: 3] CodeWalker "jo" $ Happiness : 0.3 | Nervous : 0.4 | Disgust : 0.3
setup lala
[ 4:45: 3] DummyAgent "lala" $ Starting : 1,1
[ 4:45: 3] DummyAgent "lala" $ 2,1
update lala from 1,1 to 2,1
update jo from 0,0 to 0,0
[ 4:45: 5] WorldAgent "theWorld" $ 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> -1.0 vs 2.23606797749979
update jo from 0,0 to 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
[ 4:45: 5] DummyAgent "lala" $ 2,1
update lala from 2,1 to 2,1
[ 4:45: 5] WorldAgent "theWorld" $ 1,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 0,1 to 1,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:14] DummyAgent "lala" $ 2,2
update lala from 2,1 to 2,2
[ 4:45:14] WorldAgent "theWorld" $ 2,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 1,1 to 2,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:23] DummyAgent "lala" $ 3,2
update lala from 2,2 to 3,2
[ 4:45:23] WorldAgent "theWorld" $ 3,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 2,1 to 3,1
update lala from 3,2 to 4,2
[ 4:45:26] DummyAgent "lala" $ 4,2
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:26] WorldAgent "theWorld" $ 4,1
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 3,1 to 4,1
jo
lala
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease some Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.java - Ec
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:44:57 π.μ.)
[ 4:45: 5] WorldAgent "theWorld" $ 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> -1.0 vs 2.23606797749979
update jo from 0,0 to 0,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
[ 4:45: 5] DummyAgent "lala" $ 2,1
update lala from 2,1 to 2,1
[ 4:45: 5] WorldAgent "theWorld" $ 1,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 0,1 to 1,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:14] DummyAgent "lala" $ 2,2
update lala from 2,1 to 2,2
[ 4:45:14] WorldAgent "theWorld" $ 2,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 1,1 to 2,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:23] DummyAgent "lala" $ 3,2
update lala from 2,2 to 3,2
[ 4:45:23] WorldAgent "theWorld" $ 3,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 2,1 to 3,1
update lala from 3,2 to 4,2
[ 4:45:26] DummyAgent "lala" $ 4,2
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:26] WorldAgent "theWorld" $ 4,1
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 3,1 to 4,1
[ 4:45:32] CodeWalker "jo" $ GOAL FOUND
[ 4:45:32] DummyAgent "lala" $ 4,2
update lala from 4,2 to 4,2
[ 4:45:32] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:32] WorldAgent "theWorld" $ Collision Detected 4,1 | agent lala@192.168.1.34:1099/JADE will not move
[ 4:45:32] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:32] DummyAgent "lala" $ 4,2
update lala from 4,2 to 4,2
jo
lala
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease some Disgust
Screenshot Added
A screenshot was added to your Dropbox.

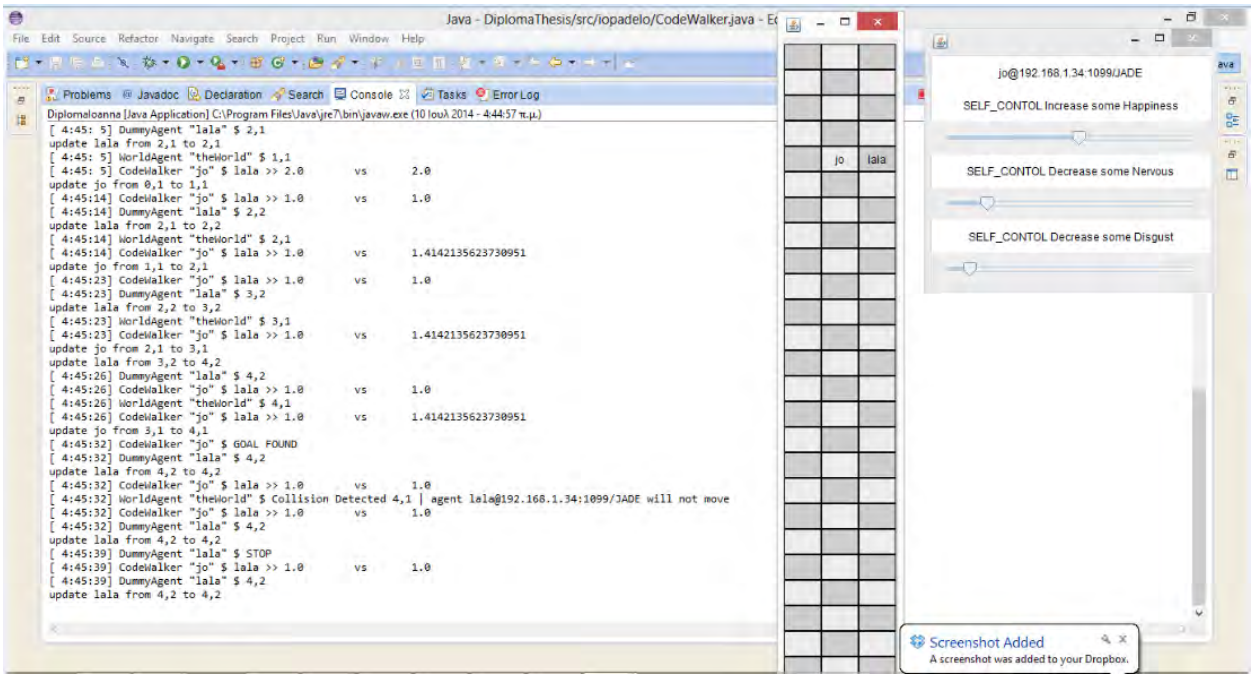
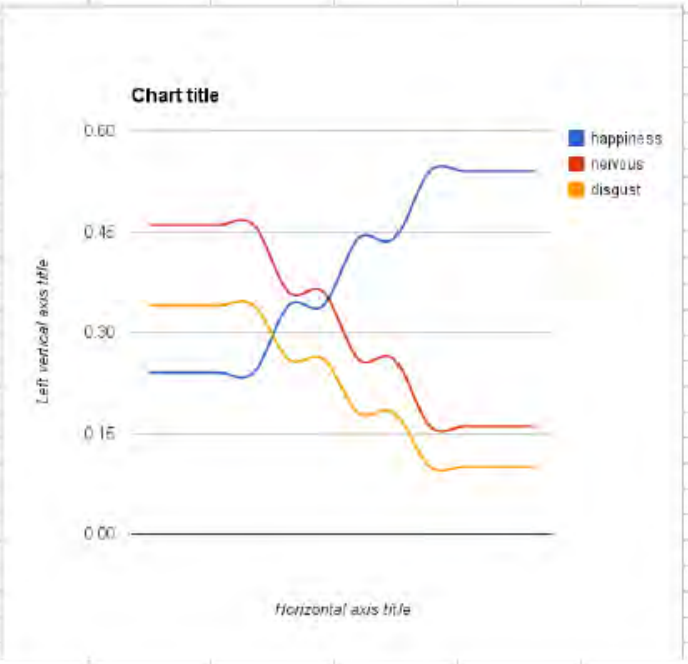

Java - DiplomaThesis/src/iopadelo/CodeWalker.java - Ec
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:44:57 π.μ.)
[ 4:45: 5] DummyAgent "lala" $ 2,1
update lala from 2,1 to 2,1
[ 4:45: 5] WorldAgent "theWorld" $ 1,1
[ 4:45: 5] CodeWalker "jo" $ lala >> 2.0 vs 2.0
update jo from 0,1 to 1,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:14] DummyAgent "lala" $ 2,2
update lala from 2,1 to 2,2
[ 4:45:14] WorldAgent "theWorld" $ 2,1
[ 4:45:14] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 1,1 to 2,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:23] DummyAgent "lala" $ 3,2
update lala from 2,2 to 3,2
[ 4:45:23] WorldAgent "theWorld" $ 3,1
[ 4:45:23] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 2,1 to 3,1
update lala from 3,2 to 4,2
[ 4:45:26] DummyAgent "lala" $ 4,2
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:26] WorldAgent "theWorld" $ 4,1
[ 4:45:26] CodeWalker "jo" $ lala >> 1.0 vs 1.4142135623730951
update jo from 3,1 to 4,1
[ 4:45:32] CodeWalker "jo" $ GOAL FOUND
[ 4:45:32] DummyAgent "lala" $ 4,2
update lala from 4,2 to 4,2
[ 4:45:32] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:32] WorldAgent "theWorld" $ Collision Detected 4,1 | agent lala@192.168.1.34:1099/JADE will not move
[ 4:45:32] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:32] DummyAgent "lala" $ 4,2
update lala from 4,2 to 4,2
[ 4:45:39] DummyAgent "lala" $ STOP
[ 4:45:39] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:45:39] DummyAgent "lala" $ 4,2
update lala from 4,2 to 4,2
jo
lala
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease some Disgust
Screenshot Added
A screenshot was added to your Dropbox.

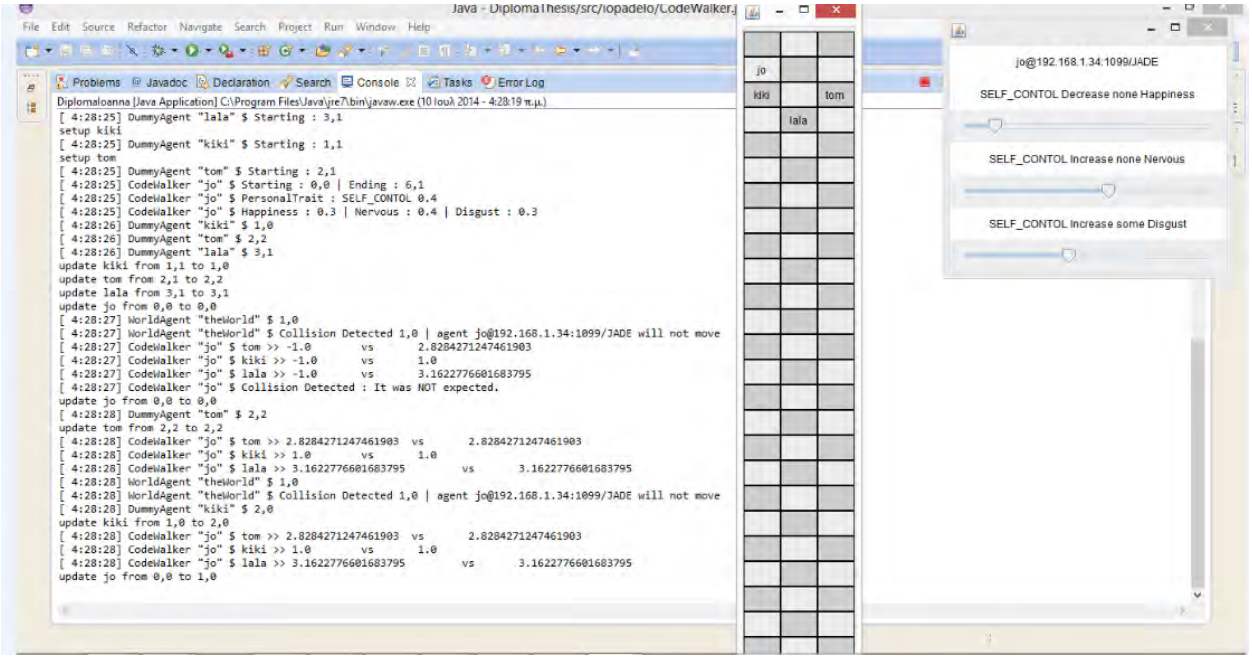
**Diagram Of Moods:** How the mood change in time.

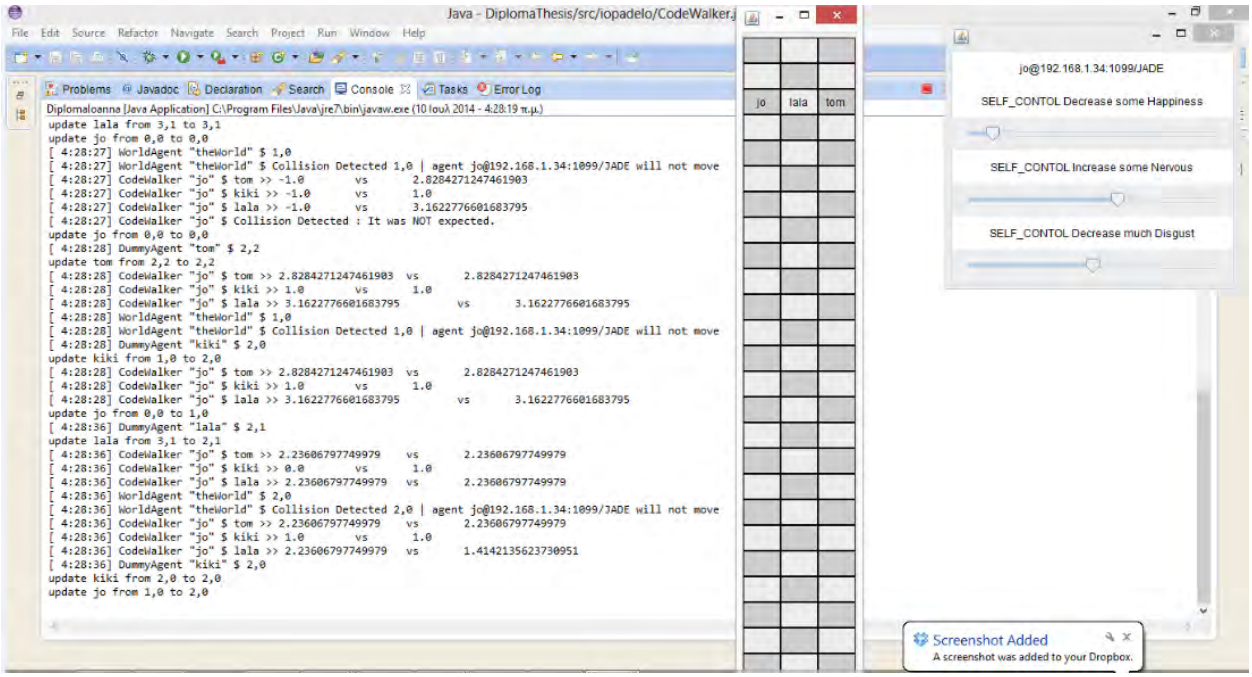
| happiness | nervous | disgust |
|---|---|---|
| 0,24 | 0,46 | 0,34 |
| 0,24 | 0,46 | 0,34 |
| 0,24 | 0,46 | 0,34 |
| 0,24 | 0,46 | 0,34 |
| 0,34 | 0,36 | 0,26 |
| 0,34 | 0,36 | 0,26 |
| 0,44 | 0,26 | 0,18 |
| 0,44 | 0,26 | 0,18 |
| 0,54 | 0,16 | 0,10 |
| 0,54 | 0,16 | 0,10 |
| 0,54 | 0,16 | 0,10 |
| 0,54 | 0,16 | 0,10 |

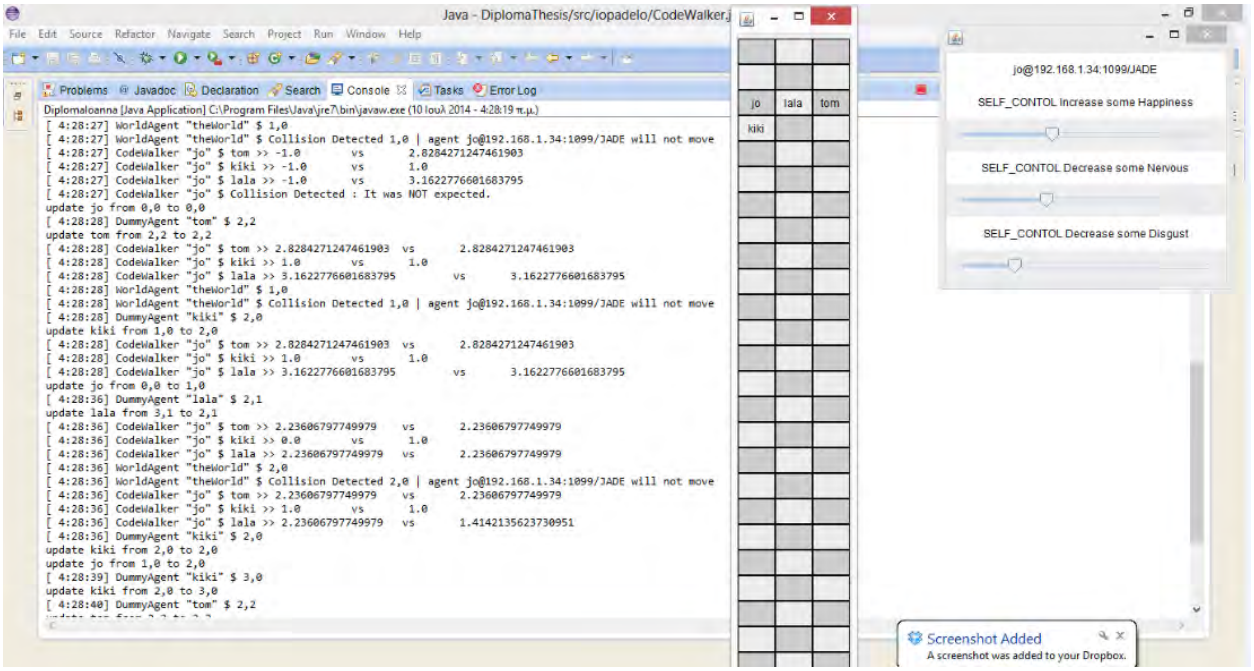


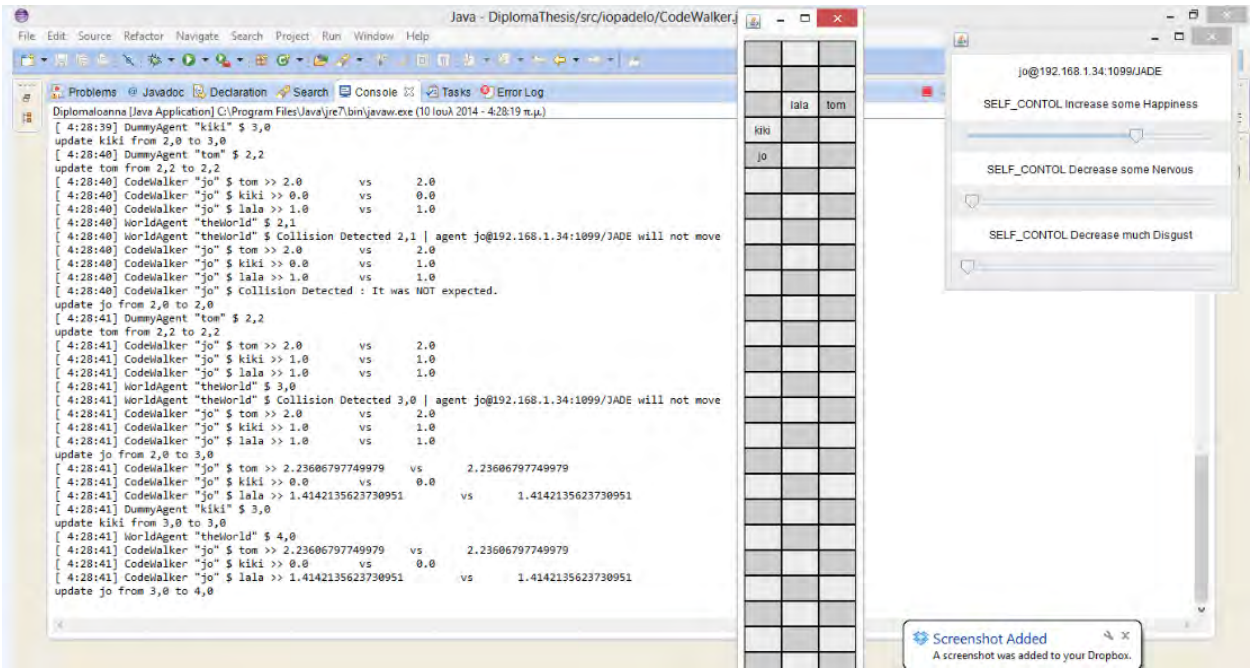
**2nd case**: three dummies. We initialized the Self_control variable at 0.4, happiness at 0.3, nervous(ness) at 0.4 and disgust at 0.3. Our Agent is in position 0,0 heading for the 6,1.

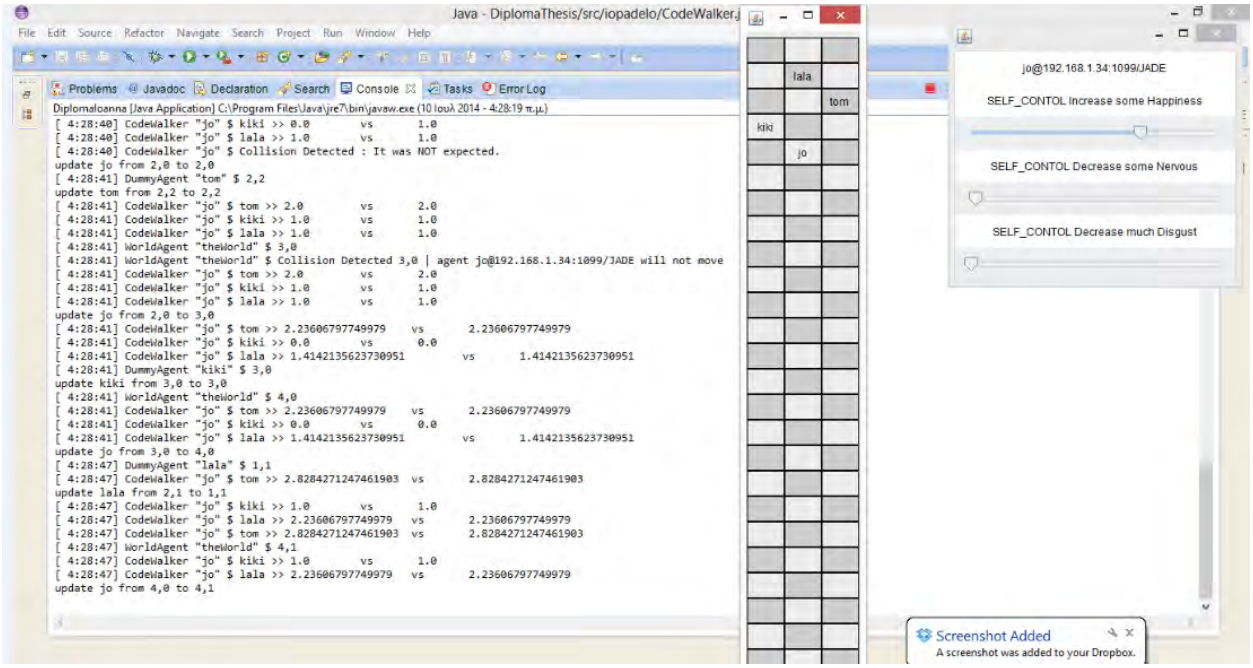
In the following screenshots, we create the CodeWalker named jo and three other dummies. The CodeWalker tries to avoid the collision with them. In the console, we print messages how the CodeWalker avoids the Dummies. First, we represent in a Gui the road where the agents walk and secondly, we represent in a second Gui how the mood changes.

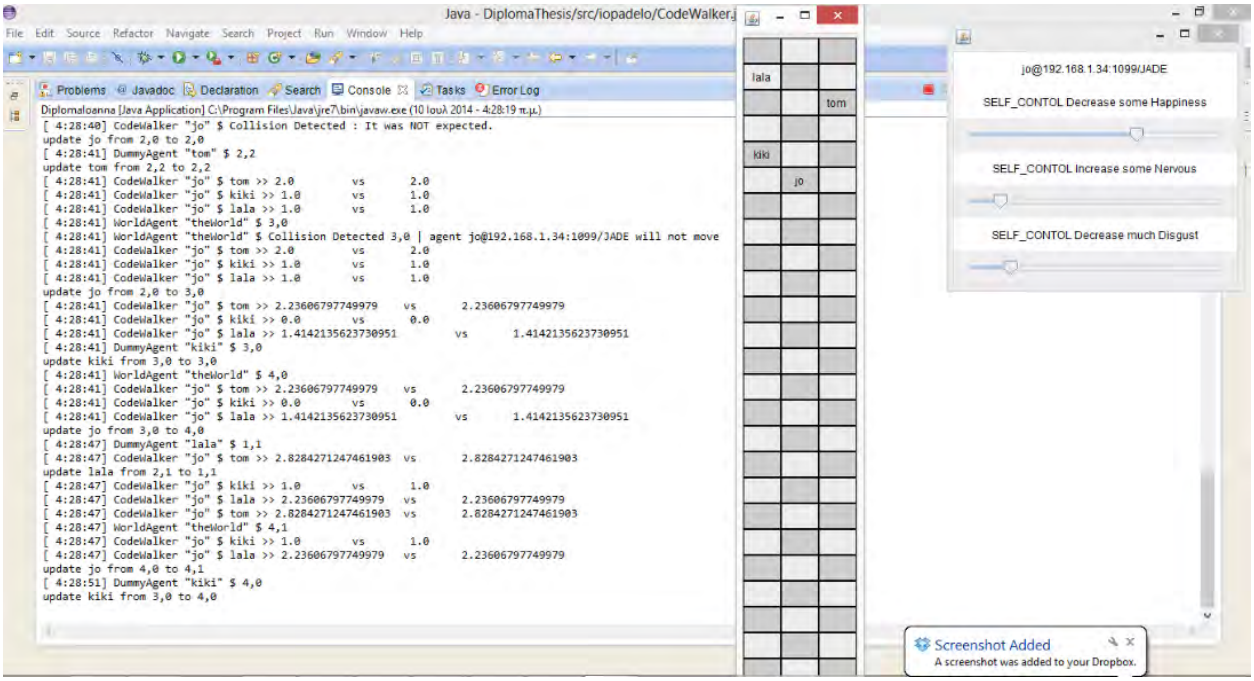



Java - DiplomaThesis/src/iopadelo/CodeWalker.j
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:28:19 π.μ.)
[ 4:28:27] WorldAgent "theWorld" $ 1,0
[ 4:28:27] WorldAgent "theWorld" $ Collision Detected 1,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:27] CodeWalker "jo" $ tom >> -1.0 vs 2.8284271247461903
[ 4:28:27] CodeWalker "jo" $ kiki >> -1.0 vs 1.0
[ 4:28:27] CodeWalker "jo" $ lala >> -1.0 vs 3.1622776601683795
[ 4:28:27] CodeWalker "jo" $ Collision Detected : It was NOT expected.
update jo from 0,0 to 0,0
[ 4:28:28] DummyAgent "tom" $ 2,2
update tom from 2,2 to 2,2
[ 4:28:28] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:28:28] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:28] CodeWalker "jo" $ lala >> 3.1622776601683795 vs 3.1622776601683795
[ 4:28:28] WorldAgent "theWorld" $ 1,0
[ 4:28:28] WorldAgent "theWorld" $ Collision Detected 1,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:28] DummyAgent "kiki" $ 2,0
update kiki from 1,0 to 2,0
[ 4:28:28] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:28:28] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:28] CodeWalker "jo" $ lala >> 3.1622776601683795 vs 3.1622776601683795
update jo from 0,0 to 1,0
[ 4:28:36] DummyAgent "lala" $ 2,1
update lala from 3,1 to 2,1
[ 4:28:36] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:36] CodeWalker "jo" $ kiki >> 0.0 vs 1.0
[ 4:28:36] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
[ 4:28:36] WorldAgent "theWorld" $ 2,0
[ 4:28:36] WorldAgent "theWorld" $ Collision Detected 2,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:36] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:36] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:36] CodeWalker "jo" $ lala >> 2.23606797749979 vs 1.4142135623730951
[ 4:28:36] DummyAgent "kiki" $ 2,0
update kiki from 2,0 to 2,0
update jo from 1,0 to 2,0
[ 4:28:39] DummyAgent "kiki" $ 3,0
update kiki from 2,0 to 3,0
[ 4:28:40] DummyAgent "tom" $ 2,2
jo
lala
tom
kiki
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease some Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.j
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:28:19 π.μ.)
[ 4:28:39] DummyAgent "kiki" $ 3,0
update kiki from 2,0 to 3,0
[ 4:28:40] DummyAgent "tom" $ 2,2
update tom from 2,2 to 2,2
[ 4:28:40] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:40] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:40] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:40] WorldAgent "theWorld" $ 2,1
[ 4:28:40] WorldAgent "theWorld" $ Collision Detected 2,1 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:40] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:40] CodeWalker "jo" $ kiki >> 0.0 vs 1.0
[ 4:28:40] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:40] CodeWalker "jo" $ Collision Detected : It was NOT expected.
update jo from 2,0 to 2,0
[ 4:28:41] DummyAgent "tom" $ 2,2
update tom from 2,2 to 2,2
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:41] WorldAgent "theWorld" $ 3,0
[ 4:28:41] WorldAgent "theWorld" $ Collision Detected 3,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 2,0 to 3,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:28:41] DummyAgent "kiki" $ 3,0
update kiki from 3,0 to 3,0
[ 4:28:41] WorldAgent "theWorld" $ 4,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
update jo from 3,0 to 4,0
lala
tom
kiki
jo
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease much Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.j
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:28:19 π.μ.)
[ 4:28:40] CodeWalker "jo" $ kiki >> 0.0 vs 1.0
[ 4:28:40] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:40] CodeWalker "jo" $ Collision Detected : It was NOT expected.
update jo from 2,0 to 2,0
[ 4:28:41] DummyAgent "tom" $ 2,2
update tom from 2,2 to 2,2
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:41] WorldAgent "theWorld" $ 3,0
[ 4:28:41] WorldAgent "theWorld" $ Collision Detected 3,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 2,0 to 3,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:28:41] DummyAgent "kiki" $ 3,0
update kiki from 3,0 to 3,0
[ 4:28:41] WorldAgent "theWorld" $ 4,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
update jo from 3,0 to 4,0
[ 4:28:47] DummyAgent "lala" $ 1,1
[ 4:28:47] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
update lala from 2,1 to 1,1
[ 4:28:47] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:47] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
[ 4:28:47] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:28:47] WorldAgent "theWorld" $ 4,1
[ 4:28:47] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:47] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
update jo from 4,0 to 4,1
lala
tom
kiki
jo
jo@192.168.1.34:1099/JADE
SELF_CONTOL Increase some Happiness
SELF_CONTOL Decrease some Nervous
SELF_CONTOL Decrease much Disgust
Screenshot Added
A screenshot was added to your Dropbox.


Java - DiplomaThesis/src/iopadelo/CodeWalker.j
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Search Console Tasks Error Log
Diplomaloanna [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10 Ιουλ 2014 - 4:28:19 π.μ.)
[ 4:28:40] CodeWalker "jo" $ Collision Detected : It was NOT expected.
update jo from 2,0 to 2,0
[ 4:28:41] DummyAgent "tom" $ 2,2
update tom from 2,2 to 2,2
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
[ 4:28:41] WorldAgent "theWorld" $ 3,0
[ 4:28:41] WorldAgent "theWorld" $ Collision Detected 3,0 | agent jo@192.168.1.34:1099/JADE will not move
[ 4:28:41] CodeWalker "jo" $ tom >> 2.0 vs 2.0
[ 4:28:41] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.0 vs 1.0
update jo from 2,0 to 3,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
[ 4:28:41] DummyAgent "kiki" $ 3,0
update kiki from 3,0 to 3,0
[ 4:28:41] WorldAgent "theWorld" $ 4,0
[ 4:28:41] CodeWalker "jo" $ tom >> 2.23606797749979 vs 2.23606797749979
[ 4:28:41] CodeWalker "jo" $ kiki >> 0.0 vs 0.0
[ 4:28:41] CodeWalker "jo" $ lala >> 1.4142135623730951 vs 1.4142135623730951
update jo from 3,0 to 4,0
[ 4:28:47] DummyAgent "lala" $ 1,1
[ 4:28:47] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
update lala from 2,1 to 1,1
[ 4:28:47] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:47] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
[ 4:28:47] CodeWalker "jo" $ tom >> 2.8284271247461903 vs 2.8284271247461903
[ 4:28:47] WorldAgent "theWorld" $ 4,1
[ 4:28:47] CodeWalker "jo" $ kiki >> 1.0 vs 1.0
[ 4:28:47] CodeWalker "jo" $ lala >> 2.23606797749979 vs 2.23606797749979
update jo from 4,0 to 4,1
[ 4:28:51] DummyAgent "kiki" $ 4,0
update kiki from 3,0 to 4,0
lala
tom
kiki
jo
jo@192.168.1.34:1099/JADE
SELF_CONTOL Decrease some Happiness
SELF_CONTOL Increase some Nervous
SELF_CONTOL Decrease much Disgust
Screenshot Added
A screenshot was added to your Dropbox.

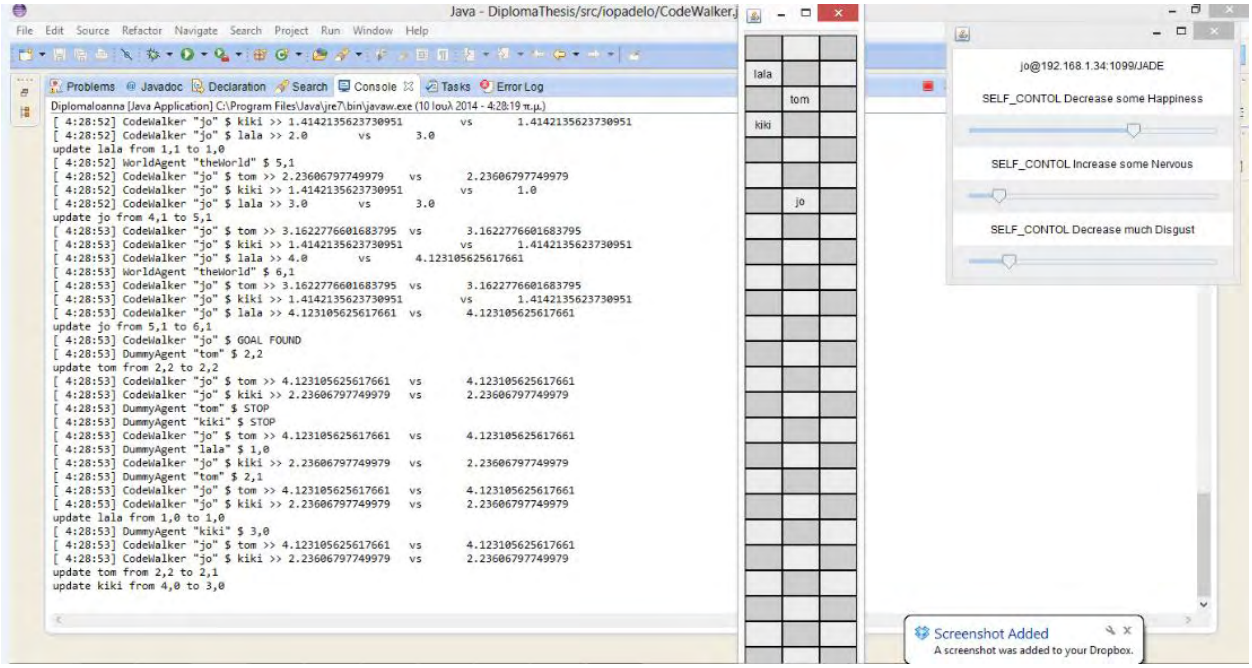
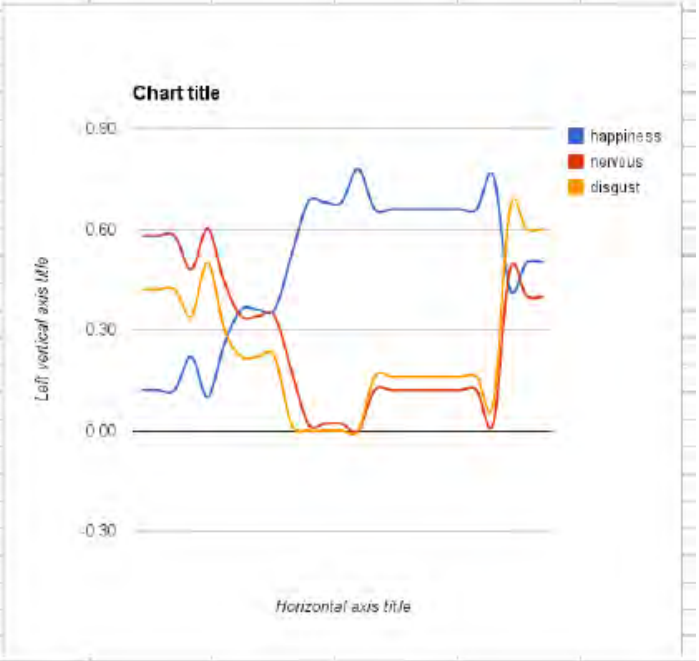



**Diagram of Moods:** How the mood changes in time.

| happiness | nervous | disgust |
|---|---|---|
| 0,12 | 0,58 | 0,42 |
| 0,12 | 0,58 | 0,42 |
| 0,12 | 0,58 | 0,42 |
| 0,22 | 0,48 | 0,34 |
| 0,10 | 0,60 | 0,50 |
| 0,26 | 0,44 | 0,30 |
| 0,36 | 0,34 | 0,22 |
| 0,36 | 0,34 | 0,22 |
| 0,36 | 0,34 | 0,22 |
| 0,52 | 0,18 | 0,02 |
| 0,68 | 0,02 | 0,00 |
| 0,68 | 0,02 | 0,00 |
| 0,68 | 0,02 | 0,00 |
| 0,78 | 0,00 | 0,00 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,66 | 0,12 | 0,16 |
| 0,76 | 0,02 | 0,08 |
| 0,42 | 0,48 | 0,66 |
| 0,50 | 0,40 | 0,60 |
| 0,50 | 0,40 | 0,60 |



# 6. FUTURE WORK

Having already presented the general design of an algorithm to resolve the expectations problem in a multi-agent system as well as the results deriving from its application with certain data and assumptions, it is worth noting some modifications or expansions that can apply. Some of these concern different acceptances for the same problem and the way it is formed into words, while othersare linked to the way it can be studied and resolved. Under any circumstances, such changes increase the intricate nature of the problem and render its solution much more demanding, but at the same time motivate more thorough research on it and are of great research interest.

### 6.1 Introduction of parameters

The initial problem can be studied under various scopes of view, changing the assumptions that have already been made for this particular development. This happens by developing a series of scenarios. In each of these scenarios, one or more of the initial assumptions for the research of the problem in the present project are rescinded**.**

- **Participation and impact of external factors**

Each problem consists of certain elements which should be taken into account towards finding a way to resolve it. There are numerous cases, though, where external factors (elements not belonging to the immediate environment of the problem) may increase or decrease the complexity of the problem and therefore play a significant role in the design of its solution. The problem of avoiding collision and the Agent's expectations deriving from this is the focus of the present project without taking under consideration external circumstances that may have an impact on it. The sole changes which occur concern the entities existing inside the environment of the problem and these are tested during the execution of the algorithm to find a solution. On the other hand, if this general view of the problem, with the specific acceptances that have been made, is assigned to a particular problem of the real world, than there will be an amountl of external events which should be taken under consideration for a complete solution from every single aspect.

- **Existence of multiple agents and priority assignment**

The problem we have examined involves only one intelligent agent. Therefore, an expansion of the algorithm vould be to add more intelligent agents or convert the Dummies into Agents with personality, expectations and perception of their environment.As a result, their reaction and their new movement/motion plan during their movement would increase the complexity of the problem. A nice/interesting scenario would be to add priorities to the agents and make the one with the highest priority decide where to move and let the others decide afterwards. We should pay attention though for there not to be starvation. This would happen to an agent if it has very

low priority and the rest, having a higher one, would not be right to keep overruling it. There should be proper conditions where our problem will run smoothly.

- As it has already been stated in a previous chapter, avoiding collisions is a fundamental criterion of the Agents' movement. As a consequence, the decision where the agent is going to move depends on whether the next position it wants to claim is already occupied by another dummy. Another way of expanding this scenario would be for the agent to be willing to check if in the position where there has already been a collision with a dummy a new collision would take place . The best case scenario each time for the agent would be to select positions that would cause as few collisions and problems as possible. This way its feelings will not be constantly disturbed and as a result the moves it makes each time will approach the ideal scenario for it, meaning that they are both desired and expected

### 6.2 Modifications in the existing algorithm

Apart from the modifications mentioned in the previous section which have to be made in the (re)solution algorithm in case some of the acceptances are rescinded or modified, there are also some optimizations or expansions that can be made in the existing algorithm.

- The JADE platform offers the possibility of designing mobile agents which can "move" within a network environment in order to achieve their goals. Essentially, these are software processes which, during their execution, are transferred to the computers constituting the network-environment.

There is a possibility the expectations problem is not found solely locally, but that it has a distributed character. This means that the environment of the problem may correspond to a JADE platform which is distributed among numerous hosts (Jade distributed Agent Platform), which results in all the agents not being in a single host. Only a particular kind of agents can be provided in each host (agents and dummies for example). At this point, the possibility Agents have to "move" will be of help/assistance, so that they can get into in another host and find their ideal/appropriate position in order to move right according to the relevant criteria. In such a scenario, the matter of the agents' communication and message exchange over the network is out of question. There will be the possibility only for agents in the same host to communicate between one another, so that they decide which is to move first at the time. In a different view of the same problem, it is likely that the communication or even the transfer of great amounts of data over the network will be considered essential. In any case, when the problem is based on a distributed JADE platform, there are certain difficulties. These also concern the fact that the remote control of mobile agents is more complex and demanding.

# BIBLIOGRAPHY

Ricardo Impert Paredes, Tesis Doctoral:Una Arquitectura Cognitiva Multinivel para Agentes con Comportamiento Influido por Caracter´ısticas Individuales y Emociones, Propias y de Otros Agentes

An Objective Character Believability Evaluation Procedure for Multi-Agent Story Generation Systems,Mark O. Riedl and R. Michael Young
http://liquidnarrative.csc.ncsu.edu/pubs/iva05-01.pdf

(Seif El-Nasr et al., 2000) - FLAME - Fuzzy Logic Adaptive Model of Emotions

Wikipedia(http://en.wikipedia.org/wiki/Intelligent_agent#CITEREFRussellNorvig2003)

S. Gadanho. Learning behavior-selection by emotions and cognition in a multi-goal robot task. Journal of Machine Learning Research, 4:385–412, 2003.

J. Gratch and S. Marsella. Evaluating the modeling and use of emotion in virtual humans.Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004), pages 320–327, New York, 2004.

S. Allen. Concern Processing in Autonomous Agents. PhD thesis, Faculty of Science of The University of Birmingham, School of Computer Science, UK, 2001.

D. Ca˜namero. Modeling motivations and emotions as a basis for intelligent behavior. Procs. of the First International Symposium on Autonomous Agents (Agents'97), pages 148–155, New York, 1997.

Ricardo Imbert Paredes and Angelica de Antonio,Using progressive adaptability against the complexity of modeling emotionally influenced virtual agents.

Ricardo Imbert and Angelica de Antonio Facultad de Informatica, Universidad Politecnica de Madrid, An Emotional Architecture for Virtual Characters.

When Emotion Does Not Mean Loss of Control Ricardo Imbert and Angelica de Antonio Facultad de Informatica, Universidad Politecnica de Madrid.

FIPA: Foundation for Intelligent Physical Agents (1999) Specification part 2 – agent communication language. The text refers to the specification dated 16 April 1999
http://www.fipa.org/

FIPA ACL Message Structure Specification:

http://www.fipa.org/specs/fipa00061/SC00061G.html

Wooldridge, M. και Jennings, N. R. (1995), Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, 10(2), pp. 115-152

Wooldrigde, M. και Jennings, N. R. (1999) The Cooperative Problem-Solving Process. J. Logic Computat., Vol. 9, No. 4, pp. 563-592, Oxford University Press

Wooldridge, M. (2002) An introduction to multi-agent systems. John Wiley & Sons

Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), http://en.wikipedia.org/wiki/Template:Russell_Norvig_2003

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: an architecture for general intelligence. Artif. Intell. 33(1):1–64.

Rao, A. S., and Georgeff, M. P. 1995. BDI agents: From theory to practice. In Proc. First Int. Conference on Multi-agent Systems (ICMAS-95), 312–319.

Bratman, M. E. 1990. What is intention? In Cohen, P. R.; Morgan, J.; and Pollack, M. E., eds., Intentions in Communication. The MIT Press. 15–32.

Gratch, J., and Marsella, S. 2004. A domain-independent framework for modeling emotion. Journal of Cognitive Systems Research 5(4):269–306.

Norling, E., and Ritter, F. E. 2004. Towards supporting psychologically plausible variability in Agent-Based human modelling. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems.

Evertsz, R.; Ritter, F. E.; Busetta, P.; and Pedrotti, M. 2008. Realistic behaviour variation in a BDI-based cognitive architecture. In Proc. of SimTecT'08.

Jiang, H.; Vidal, J. M.; and Huhns, M. N. 2007. eBDI: an architecture for emotional agents. In AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, 1–3. New York, NY, USA: ACM.

R. C. Bolles and M. S. Fanselow. (1980). A Perceptual Defensive Recuperative Model of Fear and Pain. *Behavioral and Brain Sciences*, 3, 291-301.

D. D. Price, J. E. Barrell, and J. J. Barrell. (1985). A Quantitative-Experiential Analysis of Human Emotions. *Motivation and Emotion*, 9 (1).

I. J. Roseman, P. E. Jose, and M. S. Spindel. (1990). Appraisals of Emotion-Eliciting Events: Testing a Theory of

Discrete Emotions. *Journal of Personality and Social Psychology*, 59 (5), 899-915.

# ANNEX A

**Description of the software Jade**

**Definition of Jade**

We will try and describe in brief the software we used in the development of our code, JADE, which has been essential in the design of our agents. **JADE** coordinates the various Agents among themselves. It also provides them with mutual communication using the ACL-message standard as well as services detecting the system.

**JADE** (Java Agent Development Framework) is an middleware application fully developed in Java by Telecom Italia Lab. It is used as the development environment of distributed multi-agent applications based on the peer-to-peer communication architecture. The information, the resources and the control can be distributed entirely in mobile terminals, as well as in fixed network computers. The environment may be developing dynamically with peers, called Agents in the JADE, which appear and disappear from the system depending on the needs and demands of each application. The communication among the peers, regardless of whether they run on a wired or a wireless network, is absolutely symmetrical, therefore giving each the opportunity to play the role of either the starter of a conversation or of the one responding to a conversation invitation received by another.

JADE provides:

- An environment where JADE agents are executed.
- Class Libraries to create agents using heritage and redefinition of behaviors.
- A graphical toolkit to monitoring and managing the platform of Intelligent Agent agents.

JADE's (main) purpose/aim/goal is to simplify the development of multi-agent systems, ensuring compatibility with the FIPA standard, through a set of system services and agents. To reach this goal, JADE offers the programmer the following list of features and useful tools:

- An agent platform, compatible with the FIPA standard, which includes 3 agents automatically activated upon launching the platform: **AMS** (Agent Management System), **DF** (Directory Facilitator) and **ACC** (Agent Communication Channel).
- Ability to distribute the platform in a number of hosts. Only a Java application and more specifically a **JVM** (Java Virtual Machine) runs in each host. The Agents are designed as Java threads and the Java events are used for effective/efficient communication among the Agents within the same host.
- Support and administration of each Agent's life cycle through a graphic(s) environment.
- A transfer mechanism and an interface for message exchanges among the Agents.
- Support of various communication protocols **(FIPA ACL**) and negotiation protocols (Contract Net).

- ACL message transfer within the same agents platform. These messages are transferred encoded as Java objects instead of strings. If the sender and the receiver do not belong to the same platform, the message is automatically converted into a form compatible with the FIPA standard. In this way, this conversion is not visible to the agent developers who always need to manage the same class of Java objects.
- White/Yellow Pages services to (search and) find agents and applications.
- Naming service in compliance with the FIPA standard: On launch, the agents obtain a GUID (Globally Unique Identifier) from the platform, meaning a feature that identifies them.
- Graphic(s) environment for the user **(GUI)** allowing the management of multiple agents and platforms by the same agent. The activity of each platform can be monitored and recorded.
- Support of agent transfers (code, performance status) in diverse platforms.

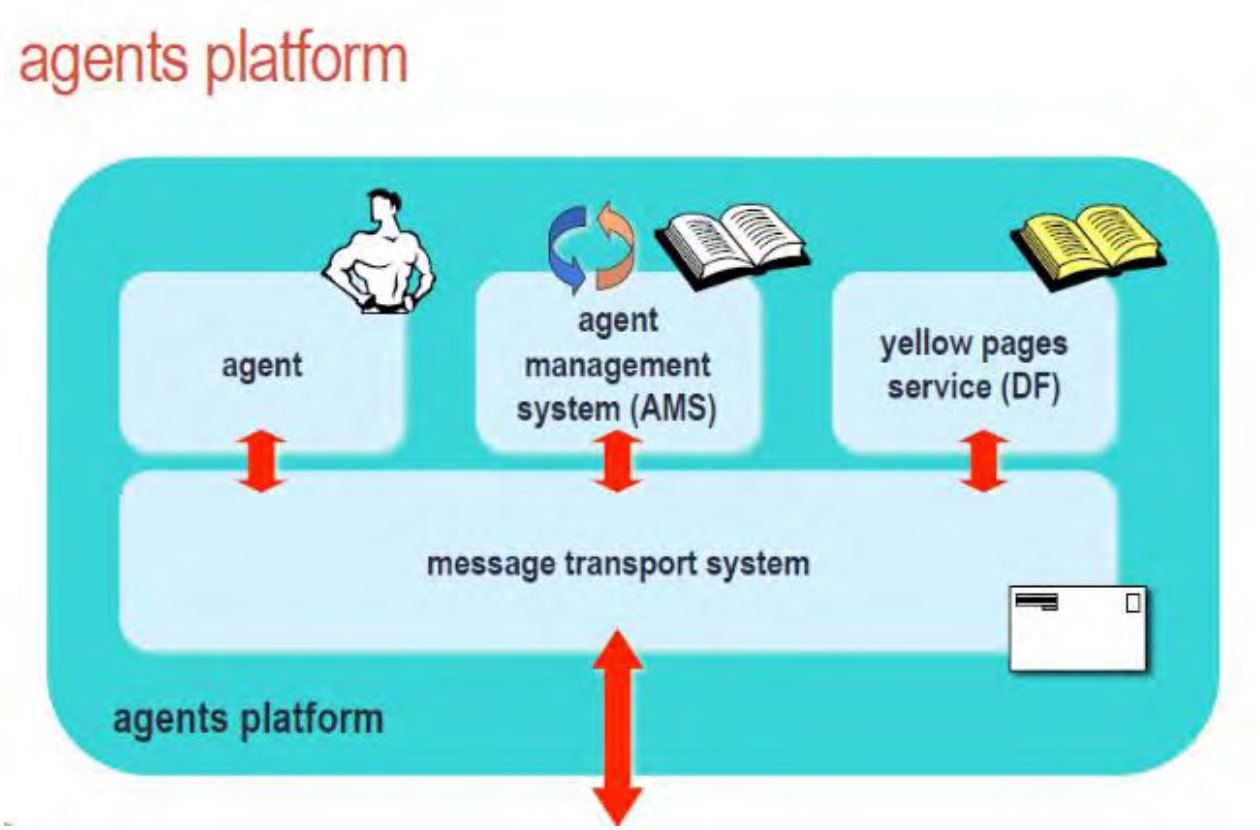



Fig 9. Display of the internal architecture of JADE

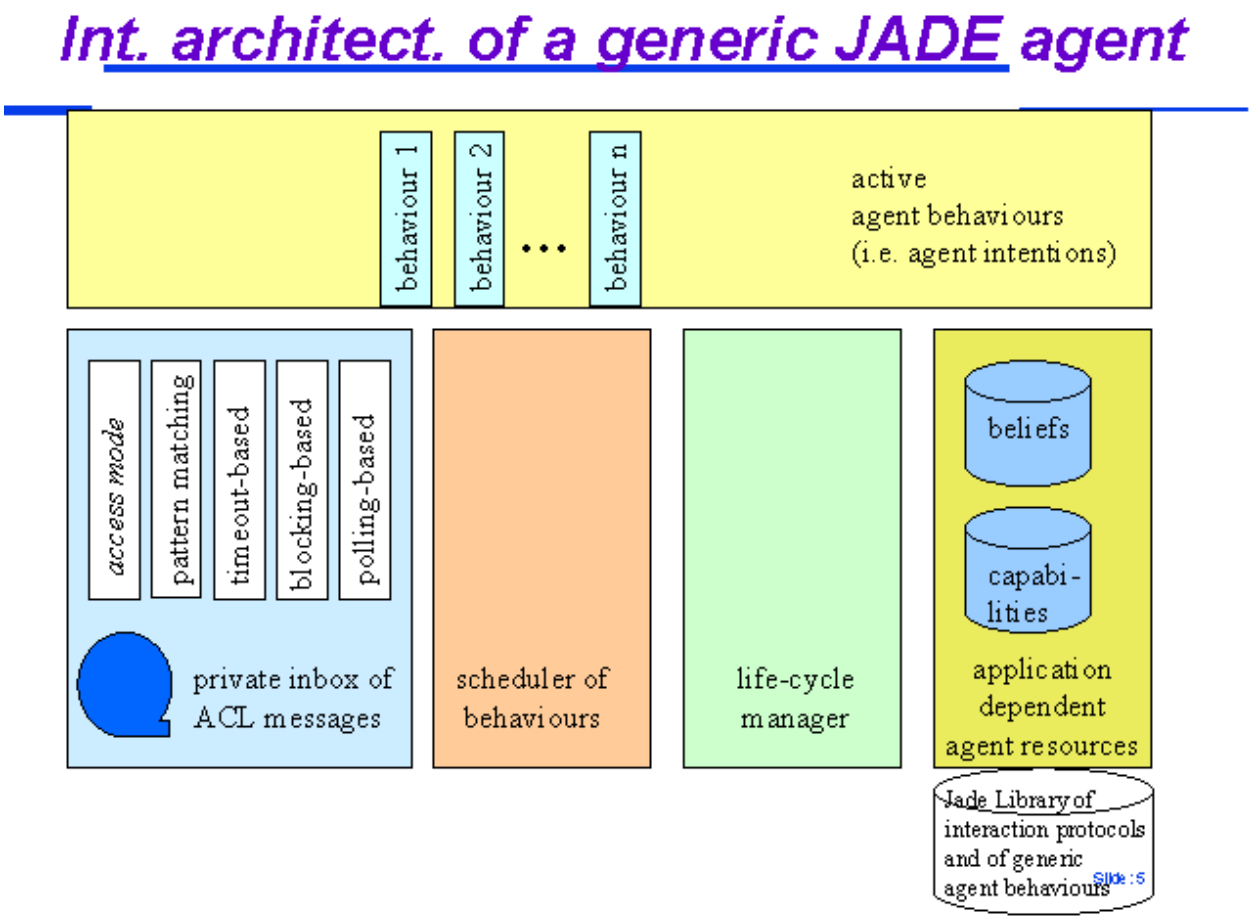
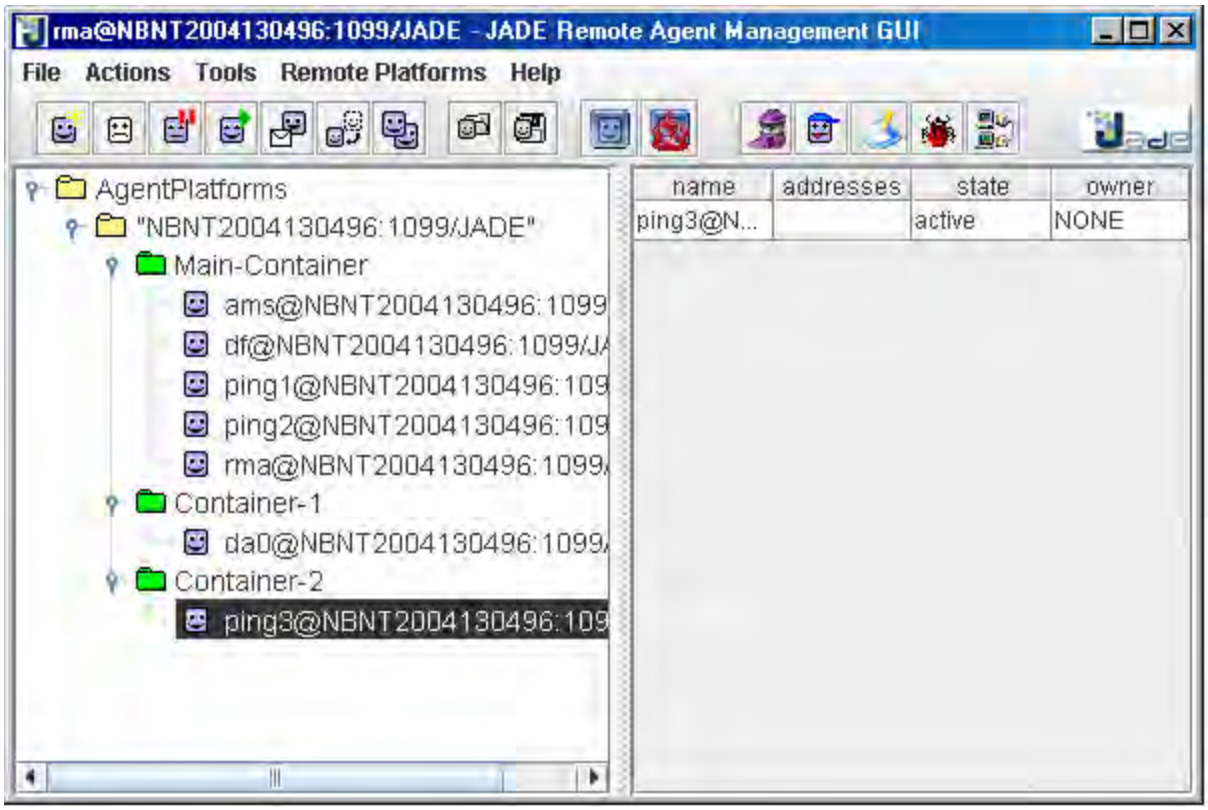
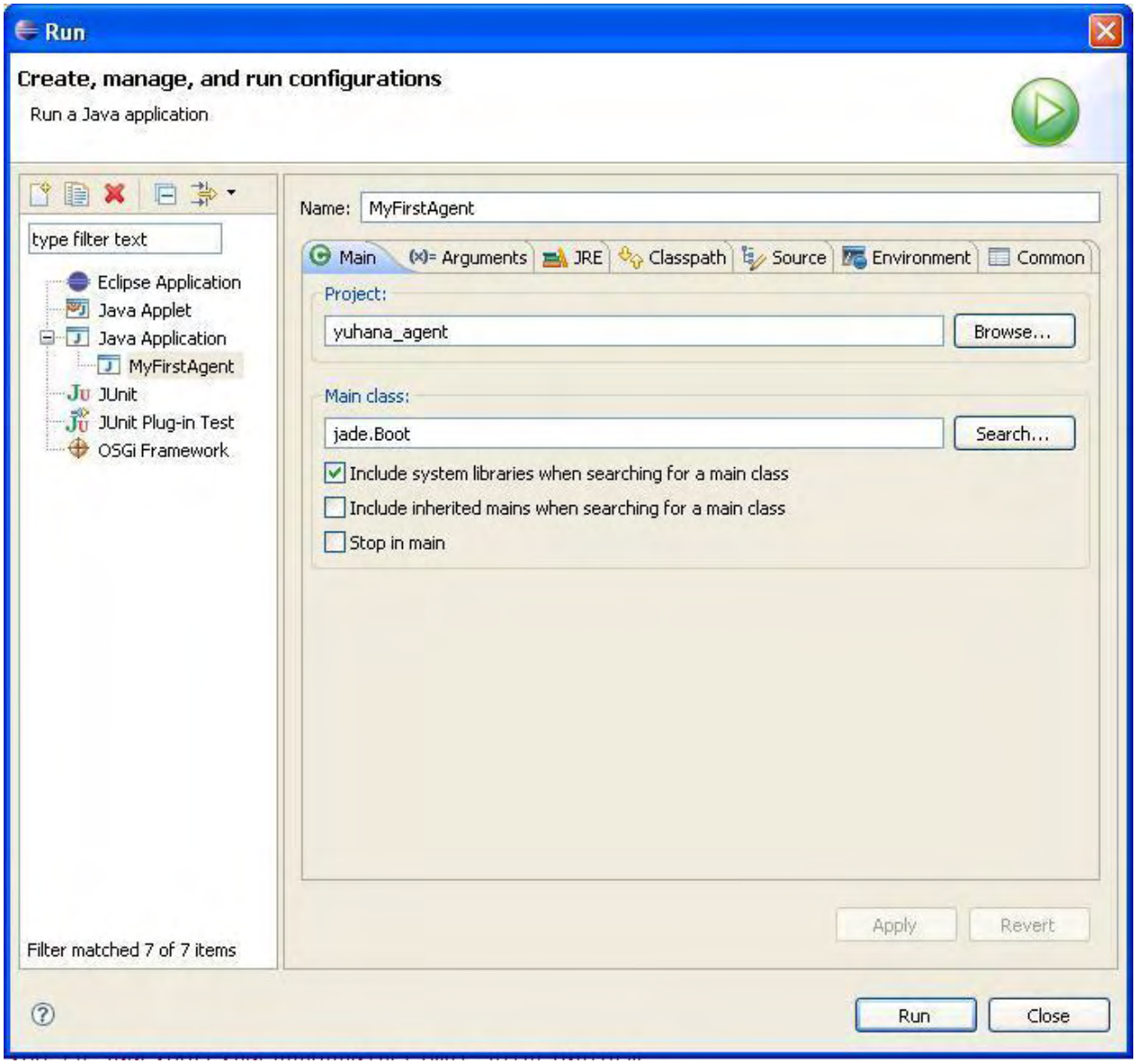



Fig.10 Architecture of a generic Jade Agent

THE GUI which is provided by the platform (fig 11, fig 12) facilitates the remote management and status control o the agents, allowing their stop and restart. Moreover, it allows the creation and beginning of an agent's performance in a remote host, as well as the control of remote platforms compatible with the FIPA standard.


rma@NBNT2004130496:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
AgentPlatforms
"NBNT2004130496:1099/JADE"
Main-Container
ams@NBNT2004130496:1099
df@NBNT2004130496:1099/JA
ping1@NBNT2004130496:109
ping2@NBNT2004130496:109
rma@NBNT2004130496:1099/
Container-1
da0@NBNT2004130496:1099/
Container-2
ping3@NBNT2004130496:109
name
addresses
state
owner
ping3@N...
active
NONE




**Platform**

(More specifically), regarding the JADE platgorm,it is a distributed agents platform, which has a container for each host where you are running the agents. Additionally the platform has various debugging tools, mobility of code and content agents, the possibility of parallel execution of the behavior of agents, as well as support for the definition of languages and ontologies. Each platform must have a parent container that has two special agents called AMS and DF.

- The **DF** (Directory Facilitator) provides a directory which announces which agents are available on the platform.

- The **AMS** (Agent Management System) controls the platform. Is the only one who can create and destroy other agents, destroy containers and stop the platform.

DF Agent:

To access the **DF** agent the class "jade.domain.DFService" and its static methods are used: *register, deregistrer, modify* and *Search*.

AMS Agent:

To access the **AMS Service** an agent is created which automatically runs the *register* method of the **AMS** by default before executing the method *setup* from the new agent. When an agent is destroyed it executes its *takedown()* method by default and automatically calls the *deregister* method of the AMS.

**Description of the Agent Class**

The Agent class is a super class which allows the users to create JADE agents. To create an agent one needs to inherit directly from *Agent*. Normally, each agent recorder several services which they should be implemented by one or more behaviours.

This class provides methods to perform the basic tasks of the agents as:

- Pass messages by objects ACLMessage, with pattern matching.
- Support the life cycle of an agent.
- Plan and execute multiple activities at the same time.

**JADE Agent**

The cycle of life of a JADE agent follows the cycle proposed by FIPA. These agents go through different states defined as:

1. Initiated: The agent has been created but has not registered yet the AMS.
2. Active: The agent has been registered and has a name. In this state it can communicate with other agents.
3. Suspended: The agent is stopped because its thread is suspended.
4. Waiting: The agent is blocked waiting for an event.
5. Deleted: The agent has finished and his thread ended his execute and there is not any more in the *AMS*.
6. Transit: The agent is moving to a new location.

**Agents Behaviour**

The behavior defines the actions under a given event. This behaviour of the agent is defined in the method ***setup*** using the method ***addBehaviour***. The different behaviors that the agent will adopt are defined from the abstract class Behaviour. The class Behaviour contains the abstract methods:

- **action()**: Is executed when the action takes place.
- **done()**: Is executed at the end of the performance.

A user can override the methods ***onStart ()*** and ***OnEnd ()*** property. Additionally, there are other methods such as **block *()*** and ***restart ()*** used for modifying the agent's behavior. When an agent is locked can be unlocked in different ways. Otherwise the user can override the methods *onStart()* and *onEnd()* the agent possess.

**Unlock an Agent**

1. Receiving a message.
2. When the timeout happens associated with block ().
3. Calling restart.

**ACL Messages**

Message passing ACL (Agent Communication Language) is the base of communication between agents. Sending messages is done by the method *send* of the class Agent. In this method you have to pass an object of type ***'ACLMessage'*** that contains the recipient information, language, coding and content of the message. These messages are sent asynchronously, while messages are received they will be stored in a message queue. There are two types of receiving ACL messages, blocking or non-blocking. For this provide methods **blockingReceive *()*** and ***receive ()*** respectively. In both methods you can make filtering messages to be retrieved from the queue by setting different templates.