4.1.2.1 Properties ........................................ 31
4.1.3 Declaration ........................................ 32
4.1.4 Informal Description ............................... 33
4.1.5 Related functions ................................ 34
4.1.6 Detailed Example ................................. 34
  4.1.6.1 Initial Data .................................. 34
  4.1.6.2 Agent Properties ............................... 35
  4.1.6.3 Actions Available ............................. 35
  4.1.6.4 Actions Description ......................... 35
4.1.7 Representation .................................... 35

4.2 Current Planning Criticism ......................... 35

4.3 New Prolog Planning ................................. 36
  4.3.1 Planning Methods ................................ 36
  4.3.2 Architecture .................................... 38
  4.3.3 Java to Prolog .................................. 39
    4.3.3.1 Prolog side ................................ 39
    4.3.3.2 Java side .................................. 43
  4.3.4 Prolog Algorithm ............................... 44
    4.3.4.1 Algorithm .................................. 44
    4.3.4.2 Prolog Side ................................ 44
      4.3.4.2.1 main .................................. 45
      4.3.4.2.2 body.pl ................................ 46
      4.3.4.2.3 actions.pl ............................... 47
      4.3.4.2.4 util.pl ................................ 48
    4.3.4.3 Java side .................................. 49
  4.3.5 Prolog to Java .................................. 50
  4.3.6 Packages ........................................ 51
  4.3.7 Library .......................................... 52

5 Anticipation under SACECA ......................... 53
  5.1 The beginning of Anticipation under SACECA .... 53
  5.2 Proposal of Anticipation under Planning .......... 54
    5.2.1 Overview ...................................... 54
    5.2.2 Details ........................................ 55
CONTENTS

5.2.3 Architecture ......................................................... 57
  5.2.3.1 Code .......................................................... 57
  5.2.3.2 User Interface ............................................... 58
5.2.4 Design ............................................................. 59
  5.2.4.1 Analyzer ...................................................... 59
  5.2.4.2 Memory ....................................................... 60
  5.2.4.3 Statistics ..................................................... 62
  5.2.4.4 Drawer ....................................................... 63
  5.2.4.5 Simulation ................................................... 65
  5.2.4.6 User Interface ............................................... 66

6 Anticipation Experiments under Planning .......................... 69
  6.1 Experiment Results ............................................... 69
  6.2 Critical Review of the Experiment ................................ 69

7 Project Planning .......................................................... 73

IV Perspectives and conclusion ......................................... 77

8 Perspectives ............................................................... 79

9 Conclusion ................................................................. 81

Appendices ....................................................................... 83

A Module Planning ........................................................... 85
  A.1 DefaultPlan .......................................................... 85
  A.2 DefaultPlanningModule ........................................... 89
  A.3 Plan ................................................................. 99
  A.4 PlanElement ......................................................... 100
  A.5 PlanningModule ..................................................... 105

B Module Planning Prolog ................................................. 109
  B.1 PrologWrapper ....................................................... 109
CONTENTS

C Module Planning Implementation 119
C.1 PrologDFSAlgorithm ........................................ 119
C.2 DFSAlgorithm .................................................. 121

D Module Prolog 125
D.1 Lock ........................................................ 125
D.2 Path .......................................................... 126
D.3 PrologInit ...................................................... 126
D.4 SinglePl ......................................................... 128

E Module Prolog Output 131
E.1 PIAction ......................................................... 131
E.2 PIActionMultiple .............................................. 132
E.3 PrologResult .................................................... 133

F Module Anticipation 139
F.1 Heuristic ...................................................... 139
F.2 LongAnticipation ............................................ 141

G Module Anticipation Analyzer 147
G.1 AbstractAlgorithm ............................................. 147
G.2 Analyzer ....................................................... 148
G.3 AnticipationGoal ............................................... 154
G.4 DefaultAlgorithm ........................................... 156
G.5 ResultWrapper ................................................ 160

H Module Anticipation Drawer 163
H.1 AbstractMemoryLayout ..................................... 163
H.2 CircleMemoryLayout ......................................... 164
H.3 Painter ........................................................ 166
H.4 Position ......................................................... 170

I Module Anticipation Memory 171
I.1 Memory .......................................................... 171
I.2 Node ............................................................. 179
I.3 VectorGeneric .................................................. 183
I. Module Anticipation Memory Statistics 185
   J.1 Statistics .................................................. 185

II. Module Anticipation Simulation 195
   K.1 Simulation ............................................. 195

III. Module View 201
    L.1 Anticipation Window ............................... 201
    L.2 SimulationWindow .................................. 212

IV. Module Controller Anticipation 251
    M.1 DisplayGraphController ............................. 251
    M.2 ObjectSelectionController .......................... 252

V. Module Controller Simulation 255
    N.1 Agent Selection Controller ......................... 255
    N.2 DisplayAnticipationGraphController ............... 256
    N.3 DisplayAnticipationWindowController .......... 257

VI. Prolog 259
    O.1 main .................................................. 259
    O.2 body ................................................... 260
    O.3 actions ............................................... 264
    O.4 util .................................................... 267
Part I

Introduction
Chapter 1

The project

This project's main idea is to improve SACECA adding new functionalities in planning and giving a basic anticipation to avatars. To do so, we split the project into serveral parts to describe with a maximum detail all steps.

1.1 Starting Point

SACECA means Société d'Agents Cognitifs Emotionnels Communicants Autonomes and is the base to start this project. The longterm aim of the project is to simulate a city with all its inhabitants. They interact with each other as well as with the environment. It's not an easy task. At the end of the project would give the following abilities to avatars.

- Plan and anticipate
- Reason
- Social skills
- Communication skills
- Perception skills
- Emotion skills
- Reaction
- Adaption
- Evolution
1.2 Why This Project?

When SACECA finishes it could be used for many industrial fields such as: security, video games, urbanism, transportation, city management, research, ... So, it has a wide range of end applications, all of them up to date.

There is also a personal component to work in it. The long-term goals are ambitious and involve a number of challenges which they stimulate my interest. It’s necessary to apply different knowledges that I have learned during my university years. Not only for a specific field, but several.

1.3 Goals

Here the project’s main objectives are summarized:

- Integrate Prolog to SACECA and implement planning algorithm. It’s about to find out an algorithm that improves the current one. In addition, it should be effective because the avatar won’t have to do useless actions. It’s a short-term planning that should solve the day to day life. From Prolog’s side, it has to be completely transparent to the user and it should represent internal changes.

- Add a first anticipation to avatars. Which it should be able to improves avatars life, to give them a better welfare. The idea is that this anticipation should take decisions to medium and long term and, as a consequence, to affect the day to day avatar.

1.4 Summary

This document is distributed in the following chunks:

1. Illustrates the theoretical bases of the project. It is necessary to demostrate that decisions have foundations.

Chapter 2. State of the art Planning and Anticipation are taken into account. In the first case, several methods have been investigated to see whether they are feasible to implement. The other, it has been necessary to search for basic theory. Not algorithms, but rather basic theory as it is not easy in this field. An other interesting point, it’s about to enumerate the challenges that represents each point.
2. All project details are explained. From the conception to the implementation phase. However, there is a small introduction to SACECA.

Chapter 3. SACECA platform Brief introduction to SACECA project’s terms. It is necessary to understand them to follow the next chapters.

Chapter 4. Planning under SACECA It is necessary to explain the basis to justify the new planning. This way, it analyzes the current SACECA algorithm. Because once determined the weak points, it is going to justify the new implementation. The other goal is to integrate Prolog with SACECA. For this reason, it explains in detail how it has been done and which guidelines has followed.

Chapter 5. Anticipation under SACECA Like the chapter above, anticipation must be justified with theory and questions like, what, how and why must be answered. UML diagram classes have been added in order to increase the comprehension, because they help to represent architecture and design.

Chapter 6. Anticipation experiments under Planning A part from implementation, results must be validated. It’s required to test the software to assure it works and goals are achieved.

Chapter 7. Project Planning To make the project, it’s set a series of dates indicating when the project phases begin and end. It is also done an assessment about of what really happen with project itself.

3. Those chapters take into account different perspectives and impressions about the project. In the last one, it needs to objective.

Chapter 8. Perspectives The months dedicated to the project gives you, at the end, a deep perspective about new ideas for future iterations. As well as, to do it in other way because some ideas were not good.

Chapter 9. Conclusion Exposes a balance about goals. If they have been accomplished or not. It also gives personal impressions about it.

4. The document also provides several appendices that shows the final code implemented in SACECA. The majority of them are coded in Java except for the last ones which It’s coded in Prolog.

Appendices A - N. Java code Include all classes implemented in Java. Each appendix represents a Java package.
Appendix O. Prolog Code Prolog code implemented in planning.
Part II

Methods and Problems
Chapter 2

State of the art

This chapter tries to give an overview about planning and anticipation. In addition, it also explains real problems which they naturally impose on this project.

2.1 Planning Problematics

First of all, it’s necessary to understand what is a plan. For that reason, there is a collection of definitions found from different sources -dictionaries, papers, ...-. This will give us a better comprehension and a bigger perspective.

- The determination of an action plan designed to achieve a goal.
- A scheme, program, or method worked out beforehand for the accomplishment of an objective: a plan of attack.
- A proposed or tentative project or course of action: had no plans for the evening.
- A systematic arrangement of elements or important parts; a configuration or outline: a seating plan; the plan of a story.
- A drawing or diagram made to scale showing the structure or arrangement of something.

All those definitions belong to different fields of study. But for second and third are close to Artificial Intelligence sense. Nevertheless, the next point clearly exposes it.

- Planning is about how an agent achieves its goals. To achieve anything but the simplest goals, an agent must reason about its future. Because an agent does not usually achieve
its goals in one step, what it should do at any time depends on what it will do in the future. What it will do in the future depends on the state it is in, which, in turn, depends on what it has done in the past.

At this point, it is distinguished that there are two different planning types: Classical versus non-classical. The next paragraphs describe both of them.

Classical planning has the following assumptions:

- the agent’s actions are deterministic; that is, the agent can predict the consequences of its actions.
- there are no external events beyond the control of the agent that change the state of the world.
- the world is fully observable; thus, the agent can observe the current state of the world.
- time progresses discretely from one state to the next.
- goals are predicates of states that must be achieved or maintained.

However, non-classical planning is totally different. The properties that describe planning become more complex. So those are the assumptions:

**Dynamic** the world keeps changing.

**Stochastic** the behavior isn’t determinist. That is, there is a random component. Or that it’s possible to get different results for a particular state.

**Partially observable** only known a part of the world

**Take time** require time to be satisfied. For example: 1 hour, 20 minutes, ...

**have continuous effects** perform an action which has effects that occur indefinitely.

There is comparison about both types in table 2.1. So we can affirm that non-classical algorithms have a bigger difficulty to solve. In all mentioned points the complexity is always bigger. Either because possibilities are bigger -more elements- or there is uncertainty, which they are not easy to deal with it.
Table 2.1: Classical vs Non-classical planning algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Classical</th>
<th>Non-Classical</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Scope</td>
<td>Finite</td>
<td>Non-Finite</td>
</tr>
<tr>
<td>Action Determinism</td>
<td>Deterministic</td>
<td>Non-Deterministic</td>
</tr>
<tr>
<td>Action Duration</td>
<td>Instantaneous</td>
<td>Durative</td>
</tr>
<tr>
<td>World Observality</td>
<td>Full</td>
<td>Partial</td>
</tr>
<tr>
<td>World Dynamics</td>
<td>Static</td>
<td>Exogenous events</td>
</tr>
<tr>
<td>Goal Attainment</td>
<td>Full</td>
<td>Partial</td>
</tr>
<tr>
<td>Time</td>
<td>No Time Points</td>
<td>Rich model of Time</td>
</tr>
</tbody>
</table>

2.2 Anticipation Problem

Like in planning, in this section is going to follow the same pattern to describe anticipation. But in this concrete case, first it’s described in a more philosophical way. After that, the section focus in the Artificial Intelligence field, which it isn’t an easy concept to identify.

- ... a process or behavior that does not only depend on past and present but also on predictions, expectations, or beliefs about the future.[13]

- For Husserl, a mathematician and philosopher from Czech Republic born in 1859, anticipation is an essential feature of human action. 'In every action we know the goal in advance in the form of an anticipation that is "empty", in the sense of vague...and [we] seek by our action to bring it step by step to concrete realization'. [4]

But what about Artificial Intelligence? It’s necessary to understand how this human concept is defined in this field. En aquest sentit, trobem la definicio següent:

1. In 1985, Robert Rosen: A system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model’s predictions pertaining to a later instant. [12]

Besides, anticipation can be defined as the text above. It’s important to understand them before create a statement about it. To all this and find a method to anticipate, that one must be generic enough to work with all types of problems or situations. Pero cal saber perque volem una anticipacio. Quins avantatges poden suposar. Tal i com es diu en [13] s’han observat tres tipus diferents d’avantatges que es passen a descriure seguidament.

**Multiple representations** Anticipation enhances adaptivity of living organisms and artificial systems. While reactive agents are certainly able to adapt to some environments, there
are limitations in the complexity of the behavior they can develop. Another important feature of anticipatory systems that enables several cognitive functions is the necessary capability to represent multiple states in parallel. Systems that have implicit forms of anticipation can certainly coordinate with their environment, but it is unnecessary to represent more than one (the current) state. Anticipatory systems, by definition, need to represent the present state and at least one potential future state. The consequent capability to represent (and act on) multiple representations and to engage in mental simulation of alternative perspectives is required for several cognitive tasks. One tenet of purposive action is that a representation of at least two states, the present and the desired one, is necessary for goal-oriented behavior. It is also important that these states share the same representational format, since they need to be compared to be able to decide if the goal is already achieved or, at least, if progress is being made. Representing future alternatives explicitly (such as an upcoming event, the existence of an object, etc.) opens up the possibility to perform complex mental operations: for example, it allows the comparison of multiple options for action decision making, the comparison of available affordances in the present and in the future, the reasoning about the state of mind of other agents (theory of mind, Buckner and Carroll, 2007). All these capabilities share the prerequisite of imagining or simulating what is not here-and-now, that is, being able to detach potential future states from the current state as well as to detach other agents’ states of mind from the current own state of mind.

**Future-oriented capabilities** Some capabilities are eminently future-oriented, since they are carried out for the sake of future and not present needs. Specific mechanisms for predicting and representing the future can facilitate them. In general, the capability to conceive the future enables the selection of actions to establish future and not (only) present immediate outcomes, selecting among multiple candidate futures, coordinating one’s own actions in the present and the future in order to realize intentions and goals that go beyond satisfaction of immediate needs, or producing and selecting future affordances instead of simply exploiting present ones.

**Bootstrapping complex cognitive capabilities** Anticipatory capabilities may not only be useful by themselves, but they may also serve as the neural basis for the formation of increasingly complex cognitive capabilities and abstract representations. Based on simple sensorimotor representations, hierarchical, abstract representations may emerge that could be highly useful for the development of grounded, symbolic representations.
2.2. ANTICIPATION PROBLEM

of the environment, such as object representations or event representations (such as touching an object). Recent machine learning literature indicates that anticipation-based mechanisms that depend on estimations of current prediction error, such as curiosity and surprise mechanisms, can play an important role in the autonomous development of a repertoire of increasingly sophisticated, hierarchical representations and behaviors (Butz et al., 2004b; Oudeyer et al., 2007; Schmidhuber, 2002, 1991b; Singh et al., 2005). Overall, anticipation might play a role not only in enhancing individual cognitive functions, but also in extending a cognitive agent’s capabilities to learn more complex, abstract, and symbolic concepts.

Predictions are typically based on prior knowledge, for example, on our past experience in similar situations or also information acquired by others. However, the ‘past’ can be represented and processed in different ways. Thus, predictions often depend on different sources of prior information and the exploitation of these to learn suitable predictions effectively.

**Statistical Regularities** The most popular mechanisms for prediction are based on statistical information that is accumulated. These mechanisms are then often mimicked or bootstrapped with Bayesian prediction or soft computing algorithms, such as neural networks or fuzzy logic. Typically these methods require a lot of information to be trained. Basically, all these mechanisms establish (implicitly or explicitly) a similarity between a set of past experiences and a novel experience, and use this similarity to generate predictions.

**Analogy** Another source for prediction is analogy, that is based on mapping of knowledge into multiple distinct domains, which permits predictions in a given domain without being trained in it. Analogical reasoners usually assume that a mapping with another domain (in which there has been some training) can be found. In this sense, the similarity is not established within events in a single domain, but in distinct domains. Analogy then focuses on abstract commonalities between domains, such as big dogs are dangerous, thus, also big tigers are dangerous or since I usually find milk close to butter I also expect to find flour close to bread. In a sense, any categorization is an analogy with a past situation; here we restrict the term ‘analogy’ to inferences involving mapping between at least two distinct domains and establishing ‘profound’ or ‘structural’ similarities and not simply surface similarities.

**Inference** Predictions can be done on the basis of inference rules. For example, our ‘ingenious physics’ model can tell us that all objects fall due to gravity. We can then predict that
if we throw an object, it will fall, independently of the object and independently of any observation. On the basis of categorical knowledge (for example, ‘all fragile objects that fall down will break’), inferences such as modus ponens can be done (for example, ‘my glass is falling down’ → ‘my glass will break’) that depend on the structure and not the content of information. Of course, in order to make such an inference it is required that information is first categorized (for example, deciding that a glass is a fragile object).

Although there are more methods to model predictions, they are not related with this project. For this reason they haven’t been included.

2.3 AI Planning Techniques

It’s worth to say there are so many planning methods. En this section, several options are explained and analyzed to know which one could be useful to, at the end, pick one. However, it’s necessary to define to which problem faces SACECA. That way, it’d be possible to restrict several options and focus in those ones which, beforehand they could be more feasible.

2.3.1 Problem Classification

As it’s the brief statement says, an inhabitant of SACECA lives in a simple environment, not like ours. So when you plan some tasks, you are sure if you will succeed in 100% because there are no independent events neither an unexpected effects on you. As well as, actions can not fail. Also, do not exists external events which they have no potential to modify our original plan. Finally, there is a full view of the world around us.

That said, now we can theoretically affirm that this problem requires to apply a classical algorithm.

At this point we already know to what kind of problem we’re facing. So now we need to look in a more detailed way because is a statement requirement.

2.3.2 Methods

[6] [11] The following options are presented as possible solutions for planning. They all solve the classical problem types. Mes concretament, exploren l’espai d’estats del problema a partir d’un estat inicial fins a trobar l’objectiu designat.
2.4. THE PROBLEM OF ANTICIPATION INTEGRATION UNDER PLANNING

**Forward Chaining** is one of the two main methods of reasoning when using inference rules (in artificial intelligence) and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. D’alta banda, aquest metode tendeix a explorar totes les opcions, fins i tot aquelles que no son pertinents. Aixo el fa poc eficaç quan l’espai a explorar es enorme.

**Backward Chaining** is an inference method that can be described (in lay terms) as working backward from the goal(s). It is used in automated theorem provers, proof assistants and other artificial intelligence applications, but it has also been observed in primates.

**Heuristics under Planning** Forward and backward chaining no son eficaços sense una bona funcio heuristica, el qual es dedica a evaluar la distancia entre un estat s i l’objectiu final. Pero per definir-lo es necessari que un individu huma el dissenyi i l’adapti al problema en questio.

**Partial-Order Planning** Forward and Backward State-Space Search Solution path imposes total ordering on actions in a plan. e.g. do X then Y then Z ... ( a sequence of actions ). But with Partial-Order Planning, solutions define multiple action orderings (partial ordering: a set of sequences achieving goals). Allows agent to use alternate concrete plan if an action becomes unavailable. Tackle subgoals independently, merge resulting plans achieved by searching in a space of partially-ordered plans, rather than states of the environment.

2.4 The Problem of Anticipation Integration under Planning

Anticipation it’s already itself a big challenge. Even whether it’s basic. But in addition, to integrate with planning it’s even worse given the SACECA.

The next points explain different problem which the project could face while working on it:

- Small testing area: SACECA doesn’t have a lot of objects to be tested. There are gauges but not with different options. Apparently, it can difficult to demonstrate that works for all cases.

- Modifications on SACECA project: can be necessary to change SACECA skeleton to add anticipation. Which could lead to a difficult implementation.
• How anticipation affects planning: decide whether anticipation modifies in some way planning algorithm or not. It’d be easier to not to but it depends on many things that needs to take into account.

• Low performance risk: avoid an slow SACECA performance. The final implementation must be fast enough in the case of intensive calculations.
Part III

SACECA Project
This part describes SACECA project and which modifications have been done. That is, how planning and anticipation have been carried out, during the proposal and implementation stages. Finally, some results are presented to demonstrate if the results were as expected or, conversely, was not to. In consequence, it has been organized in four chapters.
Chapter 3

SACECA Platform

This chapter describes which SACECA parameters are necessary to know. Because they’ll conditionate planning and anticipation.

**Avatar** this concept does reference to SACECA inhabitants. They represent a person.

**Goal** defines which objective must satisfy the avatar. So, planning algorithm must obey this Goal.

**Gauge** avatars have several gauges to satisfy, otherwise they would die. At the time of the project there are four: Hunger, Tiredness, Happiness and Thirsty. It’s an integer number which ranges from 0 to 100. In addition, it has a threshold that when Gauge value underpass it, a Goal is created to satisfy Gauge. It decreases depending on each Avatar. It’s easy to know how it works. In a nutshell, every X clock ”ticks” the value decreases by one unit.

**Services** every object found in SACECA can offer a service. Those services let an avatar to satisfy their needs directly or indirectly -intermediate steps-. Some services have prerequisites that they must be accomplished before execution. For example: have enough money to buy a can.

**Properties** avatar have properties which define them. Those values are dynamic. They have a name, a type and a default value.

**DefaultPlanningModule.java** this file contains the first planning algorithm used which used SACECA in the first version. In the next part, It’s explained carefully.
“Ticks” this concept is related to every time that SACECA application is updated. That is, all parameters are activated to let SACECA change and evolve.

**Behavior** every time there is a "tick" in SACECA, each object executes its behavior. Depending on each type behaves differently. In avatars, they have several steps which they execute sequentially: perceive, emotion, communication, reason and planning. Other process are done, such as: updates gauge value, check if it’s dead, ...
Chapter 4

Planning under SACECA

4.1 Current Planning

In this chapter it is studied the current SACECA algorithm. This way, once all details are known it will be easier to improve it.

4.1.1 Start Point

First of all, it’s necessary to show the current algorithm implementation. The function itself is called buildplan and is found in DefaultPlanningModule.java inside the n7.saceca.u3du.model.ia.planning package. The following Java source code corresponds to original implementation:

```java
public Plan buildPlan(Agent agent, Goal goal, \n    int currentDepth, int maxDepth) {

    if (maxDepth < currentDepth) {
        return DefaultPlanningModule.NO_PLAN_FOUND;
    }
    if (!goal.seemsReached(new MemoryAwareU3duJexlContext(agent))) {
        // According to the memory, the goal seems not yet reached

        // Objects order randomization to avoid having always the same scene
        Collection<WorldObject> objects = agent.getMemory().getKnowledges();
        if (COLLECTION_SSHUFFLE_ENABLED) {
            objects = new ArrayList<WorldObject>(objects);
            Collections.shuffle((List<WorldObject>) objects);
        }
```
for (WorldObject object : objects) {
    // Services order randomization
    Collection<Service> services = object.getServices();
    if (COLLECTION_SHUFFLE_ENABLED) {
        services = new ArrayList<Service>(services);
        Collections.shuffle((List<Service>) services);
    }
    for (Service service : services) {
        Agent clonedAgent = agent.deepDataClone();
        Agent clonedAgentMemory = clonedAgent.clearMemory().getKnowledgeAboutOwner();
        if (service.getName().equals("walkTo")) {
            continue;
        }
        if (service.isUsable(object, clonedAgentMemory, null)) {
            // The service is usable, according to the memory
            // What about its accessibility?
            try {
                PlanElement walkTo = this.handleWalkTo(object, clonedAgent, service.getMaxDistanceForUsage());
                service.execute(object, clonedAgentMemory, null, ExecutionMode.VIRTUAL);
                Goal goalClone = new Goal(goal.getSuccessCondition(), goal.getPriority());
                // Computing the next step of the plan
                Plan plan = this.buildPlan(clonedAgent, goalClone, currentDepth + 1, maxDepth);
                if (plan != NO_PLAN_FOUND) {
                    // The computed plan works
                    plan.cons(new PlanElement(service, object, null));
                    if (walkTo != null) {
                        plan.cons(walkTo);
                    }
                    return plan;
                }
            }
        }
    }
}
4.1. **CURRENT PLANNING**

```java
} catch (UnreachableObjectExpection e) {
    // The object cannot be reached
    continue;
}
}
return NO_PLAN_FOUND;
} else {
    return new DefaultPlan();
}
```

With a some knowledge of algorithms we can identify it as a Depth First Search. For this reason, the next section formally describes it.

### 4.1.2 Depth-First Search

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hasn’t finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration [2].

#### 4.1.2.1 Properties

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|V| + |E|)$, linear in the size of the graph. In these applications it also uses space $O(|V|)$ in the worst case to store the stack of vertices’s on the current search path as well as the set of already-visited vertices’s. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of which of these two algorithms to use depends less on their complexity and more on the different properties of the vertex orderings the two algorithms produce.

For applications of DFS to search problems in artificial intelligence, however, the graph to be searched is often either too large to visit in its entirety or even infinite, and DFS may suffer from non-termination when the length of a path in the search tree is infinite. Therefore, the search is only performed to a limited depth, and due to limited memory availability one
typically does not use data structures that keep track of the set of all previously visited vertices. In this case, the time is still linear in the number of expanded vertices and edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once and others not at all) but the space complexity of this variant of DFS is only proportional to the depth limit, much smaller than the space needed for searching to the same depth using breadth-first search. For such applications, DFS also lends itself much better to heuristic methods of choosing a likely-looking branch. When an appropriate depth limit is not known a priori, iterative deepening depth-first search applies DFS repeatedly with a sequence of increasing limits; in the artificial intelligence mode of analysis, with a branching factor greater than one, iterative deepening increases the running time by only a constant factor over the case in which the correct depth limit is known due to the geometric growth of the number of nodes per level.

DFS may be also used to collect a sample of graph nodes. However, incomplete DFS, similarly to incomplete BFS, is biased towards nodes of high degree [2].

### 4.1.3 Declaration

The following line is the function header. There are 4 parameters which are described in this section.

```java
public Plan buildPlan(Agent agent, Goal goal,
                      int currentDepth, int maxDepth) {...}
```

Now, let’s take a look to function declaration and what every parameter does.

**Agent** _agent_ It’s the agent instance to analyze. It contains all data about itself.

**Goal** _goal_ The algorithm must satisfy this goal.

**int currentDepth** The current number of plans found while executing the algorithm.

**int maxDepth** The maximum number of plans allowed to search. It the worst case, a Plan will have maxDepth actions.

**return Plan** It returns a Plan which contains a PlanElement class. Each one has a reference to the consumer and provider objects.
4.1. CURRENT PLANNING

4.1.4 Informal Description

This part explains informally how the algorithm works. It's a first view to start to understand what is doing. So every line has an effective correlation with code.

Check the depth is not surpassed.
Check the goal is not yet reached. If the condition is finally accepted, we end up with an empty plan:
Iterate all over the objects randomly sorted in memory
Iterate all over the services offered by each object (randomly sorted)
Now clone the agent as well as its memory
If the service is "walkTo" then, it analyzes the next one
If not usable (checks service preconditions), then analyze the next service.
Else
   It checks its accessibility (the object could be surrounded by other objects). If it fails it examines the next service.
With function walkTo we know how to get to the service (it returns null if it’s not possible)
Now we apply the effects of the execution of the service to the agent cloned memory. (+100 cash, -50 cash, ...)
Recursive call. we use cloned agent and its cloned memory.
It returns a plan or null value if it’s not possible to find something.
If we have a plan:
   It adds this new service to the returned plan
   If it’s accessible we add the action "walkTo" to reach to
the objects position
It finally returns the plan

With a high probability it won’t find an efficient plan but is quite sure the goal will be accomplished (With not so many actions defined). In addition, some nonsense actions will be realized before to achieve the goal.
4.1.5 Related functions

The purpose of this part is to describe what the most important functions do. As the idea is to help to improve the comprehension of the DFS algorithm in SACECA.

Goal: seemsReached(JexlContent) This function evaluates the agents goals with the help of Jexl library. Those goals are defined by object service conditions. if (!goal.seemsReached(new MemoryAwareU3duJexlContext(agent)))

Service: isUsable(object, clonedAgentMemory, null) As it’s name says, according to agent (important) memory, it tries to know if it’s possible to use the desired object. This means the agent satisfies the preconditions imposed by the service. For example, it has enough money to buy a can.

handleWalkTo(object, clonedAgent, service.getMaxDistanceForUsage()) Imagine that given an object, an agent and a maximum distance we try to figure out if the place is reachable. As it has been said before, the object could be surrounded by an other. Which it doesn’t let you use it. Finally, it returns a PlanElement or null.

service: execute(object, clonedAgentMemory, null, ExecutionMode.VIRTUAL) Giving the object and the agents memory it applies the effects of a service on the agent. For example, if the agents goes to a restaurant, the amount of money have to be reduced and his hunger bar set to maximum. The Virtual execution mode only modifies the agents memory. In normal way, the agent would execute it in real life.

4.1.6 Detailed Example

A complete example is provided to give a better comprehension. So, the next chapter is a complete explanation of what planning involves in SACECA. Therefore, to provide an example it’s necessary to define which properties and data have the agent, as well as what kind of actions can be performed.

4.1.6.1 Initial Data

Those properties can be found in SACECA application, in Memory tab. After that, it requires to search Me field. Agent properties defined by SACECA are shown there.
4.2. CURRENT PLANNING CRITICISM

4.1.6.2 Agent Properties

The following properties are defined because they are mandatory to run the example:

- c_Human_Cash: 100
- c_AccountOwner_cash: 250
- i_gauge_hunger: 65 (he’s starving)
- c_Worker_Salary: 300

4.1.6.3 Actions Available

On this example we only use 3 actions to avoid a big tree expansion. This way, it’ll be easier to explain and it will also be possible to show all possible results.

- ATM: withdrawMoney (cash: +50)
- Restaurant: eatAt (cash: -100)
- Pavement: walkTo

4.1.6.4 Actions Description

Actions definitions are available in \code\u3du-4-saceca\ud3u-4-saceca\data\ai\services folder. They have an intuitive syntax and a simple structure.

4.1.7 Representation

The figures 4.1 show the expansion tree when the algorithm is executed. All possible casuistries are shown given the initial conditions.

The figure 4.2 represents the first two branches of the 4.1 when first S3 fails, and then when it finds a solution S5.

4.2 Current Planning Criticism

What we can see is a naive algorithm. It’s not efficient in terms of the solution quality because it can produce plans where unnecessary actions are set. I.e. the example shown in figure 4.1 and 4.2 returns a solution where the avatar withdraw money twice at the ATM. However, this isn’t necessary because he has enough cash in his pocket.
4.3 New Prolog Planning

The purpose of this chapter is to explain the implementation carried out to integrate Prolog and SACECA, as well as, to justify all technical questions. Consequently, it has been splitted into several parts.

4.3.1 Planning Methods

In this subsection it is justified which method is choosen and why others are discarded.

At this point, SACECA has not so many states to explore because there is a small amount of options. In addition, it is not a goal to create a very fast algorithm with $N \times K \times L$ states. But only to achieve a result with an acceptable search speed with a set of useful actions.

So, it depends more on how the algorithm is implemented rather than which one is choosed. Consequently, any of the options proposed are acceptable. But it is necessary to decide one because there is not enough time to implement all of them. At this point, other algorithm properties must be evaluated in order to pick the best choice. It must adapt to statement requirements. Not too complex in the implementation phase neither too naive for useless results.

On the other side, it is not the intention to add an expert knowledge to the algorithm because it would become less generic. That would be the case with an heuristic. Accordingly, this method as a solution is discarded.
Hypothetical Case of: WithdrawMoney(W), WithdrawMoney(W), eatAt(A), waitTo(WT)

As a precondition we assume we can reach to all those actions. So we are sure the imposed algorithm conditions are satisfied. Which means:
- all those services are accessibles
- they aren’t “waitTo” actions

Plan: A-WT-W-WT-W-WT
Plan: A-WT-W-WT-W
Plan: A-WT-W-WT
Plan: A-WT-W
Plan: A-WT
Plan: A
Plan: [ ]

As we can see, this is not the best plan we could get. But it’s a good example as it shows how the algorithm works.

Figure 4.2: Algorithm schematic representation.
Finally, the project’s supervisor Evrard F. has created a first implementation of Forward Chaining for Prolog language. It is a good idea to start working with it because it’s a solid base. For the statement requirements is enough but it is going to be necessary to adapt to SACECA. Nevertheless, if during the implementation and test phase show that isn’t enough, first, it will be changed to forward chaining. It will have less states to search. Even so, Partial-Order planning will be deeply considered.

4.3.2 Architecture

All Prolog files are located in the folders data\ai\prolog and data\ai\tmp\avatars\. In the first one is found the source code of the implementation and all those auxiliary functions. The second, is a folder where temporal files are created by SACECA.

The figure 4.3 represents an schematic layout for Prolog files found in SACECA and how they relation themselves.

First step is to load main Prolog files at the beginning of SACECA. After that, a whole process starts to plan for an avatar.

1. Writes avatar data to files in a specified tmp folder
   
   (a) Memory.pl
   
   (b) Goal.pl

2. Calls planning function from the main Prolog file

3. Execution
4.3. NEW PROLOG PLANNING

4. Retrieves from java the final Prolog output
5. Transforms to a Java structure
6. Finally creates a SACECA Plan from the Java structure

Each step is widely explained in the following sections. But not necessarily in the same way as the schematic overview found above. Which it’s only intended to give a general view of the entire process.

4.3.3 Java to Prolog

The first step consists to transfer data from SACECA to Prolog. Several ways have been studied but in this document it’s only detailed the implemented one. The technique writes directly to files into Prolog format -predicates-. Thanks to this, data is directly loaded. This gives to Prolog flexibility in terms of access because it’s always available. In other words, there is no need to establish a connection between SACECA and Prolog when planning is being carried out.

4.3.3.1 Prolog side

Then two files are created for each avatar. Those are memory and goal. First, as it says its file name, contains memory data. The other, the current state of Gauges and money and the Goal to achieve. In a nutshell, those files are simple but so important. Figure 4.4 shows SACECA layout of all files. They are all created into the folder data/ai/tmp/avatarId. For each avatar a new folder is automatically created when necessary. For that reason, before the first SACECA execution any file nor folder are found there.

The following Prolog source code shows the real context of memory.pl. Therefore, all lines are predicates. Each defines a different property of the avatar or the content of memory.

```prolog
worldobject(avatar4246, obj8, house, restathome).
worldobject(avatar4246, obj20, restaurant, eatat).
worldobject(avatar4246, obj21, cinema, watchmovie).
...

service(avatar4246, watchmovie).
```
service(avatar4246, restathome).
service(avatar4246, eatat).
...

position(avatar4246, obj8, 27, -61).
position(avatar4246, obj20, -16, -53).
position(avatar4246, obj4246, -23, -6).
position(avatar4246, obj21, 64, 52).
...

%Object properties
object_properties(avatar4246, obj8, i_threaded, 'false').
object_properties(avatar4246, obj8, i_isMisc, 'false').
object_properties(avatar4246, obj8, c_Building_address, '3 Beach road').
object_properties(avatar4246, obj8, i_mass, 42).
object_properties(avatar4246, obj8, i_graphics_conf, 'House1').

object_properties(avatar4246, obj20, i_threaded, 'false').
object_properties(avatar4246, obj20, c_Restaurant_mealDuration, 15).
object_properties(avatar4246, obj20, i_isMisc, 'false').
object_properties(avatar4246, obj20, c_Building_address, '2 Beach road').
object_properties(avatar4246, obj20, i_mass, 42).
object_properties(avatar4246, obj20, i_graphics_conf, 'MacDo').
% Avatar properties

object_properties(avatar4246, obj4246, c_AccountOwner_account, 250).
object_properties(avatar4246, obj4246, c_Human_cash, 100).
object_properties(avatar4246, obj4246, c_BuildingOwner_address, '15 Beach road').
object_properties(avatar4246, obj4246, c_Worker_companyName, 'The World Company').
object_properties(avatar4246, obj4246, i_gauge_hunger, 80).
object_properties(avatar4246, obj4246, i_decrement_period_default, 14).
object_properties(avatar4246, obj4246, i_perception_maxHearingDistance, 10).
object_properties(avatar4246, obj4246, i_perception_maxEyesightDistance, 10).
object_properties(avatar4246, obj4246, i_name, 'Aurelien').
object_properties(avatar4246, obj4246, i_mass, 75).
object_properties(avatar4246, obj4246, i_decrement_period_thirst, 8).
object_properties(avatar4246, obj4246, i_graphics_conf, 'Man').
object_properties(avatar4246, obj4246, i_gauge_thirst, 78).
object_properties(avatar4246, obj4246, i_gauge_tiredness, 50).
object_properties(avatar4246, obj4246, i_gauge_happiness, 87).
object_properties(avatar4246, obj4246, c_Worker_workingTime, 40).
object_properties(avatar4246, obj4246, c_Human_restDuration, 50).
object_properties(avatar4246, obj4246, c_Human_maxCarriedMass, 10).
object_properties(avatar4246, obj4246, c_Human_walk_speed, 0).
object_properties(avatar4246, obj4246, i_threaded, 'true').
object_properties(avatar4246, obj4246, i_isMisc, 'false').
object_properties(avatar4246, obj4246, c_Worker_salary, 2000).

object_properties(avatar4246, obj21, c_Cinema_filmDuration, 30).
object_properties(avatar4246, obj21, i_threaded, 'false').
object_properties(avatar4246, obj21, i_isMisc, 'false').
object_properties(avatar4246, obj21, c_Building_address, '3 Time street').
object_properties(avatar4246, obj21, i_mass, 42).
object_properties(avatar4246, obj21, i_graphics_conf, 'CinemaGaumont').

priority(avatar4246, obj8, 0).
priority(avatar4246, obj20, 1).
priority(avatar4246, obj4246, 0).
priority(avatar4246, obj21, 0).
...

There are 5 different types and each describes a fact. Part of the code is removed. Otherwise it’d be too long and those lines are enough to explain it. In the example, 3 different objects are shown as well as the avatar itself. For that reason, the are more properties for the object obj4246 than the rest. Note that all data is represented. The names of the predicates are intended to be self descriptive. However, each predicate is described next:

**worldobject** Describes an object from memory. Avatar identification to which belongs, object identification, the type and which service offers. If there are more than one service, the line is written again but with different service.

**service** All avatar services, with the identification avatar and its service name.

**position** Where an object is located. Again there are the identification avatar and object and its position X, Y. Currently not used but has been added for future SACECA iterations.

**object_properties** Describes all object properties found in memory. Avatar’s and object identification, the name and its value.

**priority** It’s the priority of an object to be chosen while planning. A part from the identifiers there is also an integer number. 0 is the lower priority with no maximum value. This value is modified by anticipation module.

At this point, let’s analyze Goal.pl, which contains only two predicates. It’s a simple and small file but it defines current Gauge values and its money. The other predicates indicate the amount of units to satisfy. By order of appearance, each field represents:

1. Avatar cash
2. Account money
3. Tiredness gauge
4. Thirsty gauge
5. Hunger gauge
4.3. NEW PROLOG PLANNING

6. Happiness gauge

% cash, account, tiredness, thirsty, hunger, happiness
current(avatar4246,100,250,50,78,80,87).
caim(avatar4246,0,0,80,0,0,0).

Following the example above, this text would correspond for goal.pl at the same time. It's also the same avatar. In addition, SACECA must satisfy tiredness gauge because if we compare predicates values of current and caim, the second is higher. For this reason, the algorithm will try to match or exceed it. The other values are already satisfied because they are bigger than 0. By default, zero is written when there isn't a defined goal for that gauge.

The text below describes each predicate:

current  It has the current avatar values.
caim  It defines the goal. As well as, which values of current will have to match or overpass.

4.3.3.2 Java side

To generate the files above, it's necessary to code some functions in SACECA responsible for the translation. A package fr.n7.saceca.u3du.model.ai.agent.module.planning.prolog has been added in order to maintain the consistency and organization of SACECA code.

In the figure 4.5, attributes and functions aren't shown in DefaultPlanningModule because they are almost the same version as the last one. Except that now is an abstract class and it must be implemented by another. In that case is PrologDFSAlgorithm. The next functions produce the real work for translation to Prolog. Their names try to be self descriptive but they are describe below to remove any doubt.
CHAPTER 4. PLANNING UNDER SACECA

GoalToProlog(Goal) Generates goal.pl predicates, ccurrent and caim.

PropertiesToProlog(String, WorldObject) Generates the property predicates of an object.

MemoryToProlog(List<WorldObject>) Generate objects predicates, services, object position and their priorities.

At last, all generated data is written in the corresponding file and directory. PrologDF-Algorithm is in charge of that job, inside buildPlan method. Consequently, those files are modified just before planning is called. The reason is that memory is dynamic because the avatar can interact with the environment. Then, it’s necessary to update to let the algorithm find new plans.

4.3.4 Prolog Algorithm

This section describes the algorithm used and the implementation carried out. Specially important is to show the important files and their aim. For this, we tried to achieve a logical structure, as well as, intuitive. The new algorithm uses Forward Chaining. This way, we’ve added a formal description of the method because it’s necessary to understand how it works.

4.3.4.1 Algorithm

Forward chaining is one of the two main methods of reasoning when using inference rules (in artificial intelligence) and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems.

Forward chaining starts with the available data and uses inference rules to extract more data (from an end user for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When found it can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data. [7]

4.3.4.2 Prolog Side

In the folder \data\ai\prolog\ there are those files: main.pl, body.pl, actins.pl and util.pl. All of them are necessary for the execution and have their specific function. The following points explain in detail each file and their important parts. On the other hand, the files itself contains comments to improve their comprehension.
4.3. **NEW PROLOG PLANNING**

4.3.4.2.1 **main**  The most important file because from this one the algorithm is called. It must be executed every time we need to plan. It takes care to load all necessary files. It has two functions with a specific goal.

- getplan(MyIdent, Plan, Hour, MaxDepth):-
- preload(Path):-

The first function calls the algorithm to create a new plan. It also loads goal.pl and memory.pl files for the current avatar. Both files are mandatory for the execution and they are generated just before the getplan call. As it has been already said, memory is dynamic so it’s necessary to update information every time. Then it tries to find a plan with the minimum number of actions.

**MyIdent**  Integer/Long, Avatar’s identifier.

**Plan**  Variable, return the plan found.

**Hour**  In minutes, It tells to what time starts planning. Because each actions has a duration. Nevertheless, it’s no used. But it’s there for future revisions.

**MaxDepth**  Integer, the maximum number of elements a plan can have.

preload is the other function found in the file. It takes care to load util.pl, actions.pl and body.pl given the folder path. Those files are common to all avatars. Therefore, it’s necessary to load them once. The class PrologInit.java takes care of it and it is found in the package fr.n7.saceca.u3du.prolog.

Singleton pattern is used in this problem because it has to deal with multiple threads but it’s only need to load once. For this reason, it’s only required one instance. It’s also an exclusion area because it only lets to one thread to execute the code at the same time. Moreover, it’s necessary to know when SACECA call the Prolog function and under which conditions. The next function shows PrologInit call, which is found in DefaultPlanningModule.java class, in
the package fr.n7.saceca.u3du.model.ai.agent.module.planning. The initialization of this class is only done at the beginning of SACECA.

```java
public DefaultPlanningModule(Agent agent) {
    //...
    try {
      PrologInit.getInstance(this.agent.getId());
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
}
```

4.3.4.2.2 body.pl  It has the algorithm code. So here, takes place the real work. At this point, all avatar data is loaded. Defined functions are related to the recursive call, which defines the starting point and check that the final condition is satisfied. Unlike the main file, here the algorithm is called for forcing a plane with N elements. Otherwise it returns empty.

```prolog
mplan(I,B,P,N,F,CData,Avatar) :-
  M is N,
  writeln(CData),
  generf(I,F,P,M,CData, Avatar),
  inclus(B,F),

  % extracts information which must be satisfied
  caim(Avatar, CashAim, BankAim, TirednessAim, ThristyAim, HungerAim, HappinessAim),

  % algorithm data output
  member(cash(robo,FinalCash),F),
  member(account(robo,FinalBank),F),
  member(tiredness(robo,FinalTiredness),F),
  member(thristy(robo,FinalThristy),F),
  member(hunger(robo,FinalHunger),F),
  member(happiness(robo,FinalHappiness),F),

  % check defined aim vs output
```
FinalCash >= CashAim,
FinalBank >= BankAim,
FinalTiredness >= TirednessAim,
FinalThristy >= ThristyAim,
FinalHunger >= HungerAim,
FinalHappiness >= HappinessAim,

The code above describes when the condition is satisfied. We see how data is extracted from caim predicate and then, it’s compared with current gauges data. generf tries to generate a plan.

4.3.4.2.3 actions.pl All actions are described here. They all have an equal in SACECA and they must added manually. It’s important that they have the same requirements and effects. Otherwise, there would be non expected plannings. All of them have a pattern which is describe here.

```
action(work(robo, CurrTime, NewTime, IdObject),
    [heurecourante(CurrTime),account(robo, CBank)],
    [heurecourante(CurrTime),account(robo, CBank)],
    [heurecourante(NewTime),account(robo, PlusBank)],
    [CurrCash, CBank, CTired, CThrist, CHunger, CHappy, CurrTime],
    [CurrCash, PlusBank, CTired, CThrist, CHunger, CHappy, NewTime],
    Avatar) :-
    getOffices(Avatar, IdObject),
    getProperty(Avatar, c_Worker_workingTime, Duree),
    NewTime is CurrTime + Duree,
    getProperty(Avatar, c_Worker_salary, Salary), % avatar salary
    PlusBank is CBank + Salary.

action(withdrawmoney(robo, Duree, IdObject),
    [heurecourante(CurrTime),cash(robo, CurrCash), account(robo, CurrBank)],
    [heurecourante(CurrTime),cash(robo, CurrCash), account(robo, CurrBank)],
    [heurecourante(NewTime),cash(robo, NewCash), account(robo, LessBank)],
    [CurrCash, CurrBank, CTired, CThrist, CHunger, CHappy, CurrTime],
```
[NewCash, LessBank, CTired, CThrist, CHunger, CHappy, NewTime],
Avatar) :-
getObjectByPriorityId(Avatar, ObjectId, withdrawmoney),
Duree is 10,
NewTime is CurrTime + Duree,
NewCash is CurrCash + 50,
LessBank is CurrBank - 50,
LessBank >= 0.

Two of the multiple actions are shown above. Those are work and withdraw money from an ATM. Due to the algorithm implementation it's necessary to write long function headers. First, we have which action is and its arguments -Inbound and outbound-. The second line has which conditions must satisfy before it can be executed. The third, which predicates delete from list. Fourth which new predicates to add with updated data. The next list has the current data which is used to fill those inbound variables of other lists. Thanks to this, it isn't necessary to find these values, it could take a lot of time. The last list, it has the new updated data. It's outbound and it will be used for the next calls. It saves the current avatar state. Finally, there is the avatar identification code. In consequence, the following lines have the proper code to change avatar properties. For example: where he lives or where he works, ... All those instructions are quite simple due to the Prolog nature of inference rules.

Work action updates his bank account and until what time ends that task. So to get such information there are available multiple rules. For example: getOffice, which it returns the office identification code and getProperty, which it returns the value of a property. As all other actions, withdraw money follows the same scheme. But as it have different effects it uses other rules. The rule getObjectByPriorityId selects an object which offers the service withdrawmoney. Moreover, it also gives more priority to the same objects with the highest value. In case of multiple priorities with the same value, it chooses them randomly -it's actually Java the responsible of randomness due to the simplicity-.

Those actions which are instantaneous take 1 minute.

4.3.4.2.4 util.pl Auxiliary rules which don't have a direct attachment with the algorithm but are necessary because of their dependency are found here. A part from rules there are also directives to modify predicates behavior.

:- multifile object_properties/4.
4.3. NEW PROLOG PLANNING

:- multifile position/4.
:- multifile service/2.
:- multifile worldobject/4.
:- multifile ccurrent/7.
:- multifile caim/7.

The directive is multiple. Given a predicate and the arity it lets to load the same predicates from different files. Otherwise, it would overwrite the last definitions. Thanks to this, Prolog can have all avatar memories loaded at the same time.

The most important rules found are explained to give a better comprehension.

**getOffices(Av, IdObject)** Given the avatar, it returns the id object which is an office. It's expected that it have one office.

**getHome(Av, IdObjects)** The same as above, but with avatar home. Also it has the same precondition.

**getObjectsId(Av, Property, Value)** For a certain avatar and a property name, it returns the value.

**getPropertyObj(Av, Obj, Property, Value)** The same as above but it also return the identification object.

**getObjectsByPriorityId(Avatar, IdObject, Service)** It isn’t a basic function. But it gets the object with highest priority for a given service.

### 4.3.4.3 Java side

It’s necessary to call the rule getplan from Java to Prolog. In addition, arguments are required. To say the least, JPL library imposes a restriction in which it’s necessary to take into account when implementation is carried out. A direct effect is the penalization of SACECA efficiency.

PrologDFSAlogrithm class is in charge of Java Prolog call, specifically, the function build-Plan. It has been chosen the most interesting part of the code to show. There is an exclusion area which only one thread can be there. The other code is about Prolog call from Java.

```java
try {
    SinglePl.getInstance().lock();
    Variable P = new Variable("P");
```
In this case, inbound parameters are set as integers and the outbound variable P is declared. At the end of the planning execution, PrologToSacecaPlan change Prolog P output in Plan SACECA class.

It’s necessary to justify why is needed an exclusion area at this point. It’s because JPL library it can only manage at the same time one Prolog Virtual Machine. It doesn’t matter the number of threads neither different instances. Due to library implementation they will share the same Prolog VM. So to avoid conflicts it’s necessary to isolate calls. Even so, for future SACECA revisions, swipl can handle threads inside. It would be possible to exploit the power of new multiprocessors machines. The following link confirms the information http://www.swi-prolog.org/packages/jpl/java_api/high-level_interface.html#Multi-Threaded%20Queries.

### 4.3.5 Prolog to Java

Once data is processed and a plan is found -or not-, Java needs to recover and convert the information from Prolog to SACECA plan. To do so, Prolog data is transformed into a middle stage Java structure. Finally, a SACECA plan is created. This way it will be able to execute the tasks successfully.
4.3. NEW PROLOG PLANNING

Inside the package fr.n7.saceca.u3du.prolog.output there is the class PrologResult.java. It’s in charge to do the first step. The figure 4.7 describes the classes and their dependencies.

PlAction represents an action from the Prolog output. Data is extracted from Prolog to create an instance. If the action has a specific duration then, PlActionMultiple wraps it. So when all actions are read then, it comes the second step. For this, let’s take a look to PrologWrapper class. The function PrologToSacecaPlan does the final adaptation.

4.3.6 Packages

During Prolog integration it has been added new packages to keep a good code organization. The figure 4.8 show them and it’s dependencies.
4.3.7 Library

To communicate Java and Prolog there is a library called Java-Prolog Library [3]. It is found in the directory lib\swipl-5.10.1\lib. There, you find all files to execute SACECA in stand-alone. Each OS (Windows and Linux) need to configure their own environment variables. This way, jpl.jar dependencies are found. Due to some compatibility problems with OS, it's used an old version. In fact, under Linux jpl.jar need swipl shared libraries. But in the last versions isn’t possible for some unknown reason. The solution was found in this website http://code.google.com/p/javanaproche/wiki/HowToJPL. Otherwise, the last version is used under Windows 7 32 bits. At the time of the project, there is the version 6.0.2. Finally, for each version you can find library files under swipl lib subfolders.
Chapter 5

Anticipation under SACECA

The second part of the project is about to give a basic anticipation to avatars. It doesn’t seem an easy task, so in the first version it’s proposed a method that it’ll be necessary to check if it works. For that reason the chapter has been divided in 2 parts.

5.1 The beginning of Anticipation under SACECA

Due to state of the art chapter, it is desired to anticipate through past experiences. For this reason, there is the need to conceive a system that let to acquire this knowledge. As it requires time to do it due to data recopilation, we though it could be a great idea to accelerate time in a simulation environment. Moreover, it should also let to modify the environment itself to force certain conditions. We can consider as a simulation engine for avatars which it would let to extract the desired information. Therefore, the final solution of a basic anticipation should work over this.

We propose this method to put in practice the anticipation engine: it is desired that avatars use those services that are cheaper in terms of time/money for a gauge. In other words, if it exists multiple options to chose a restaurant, then it would pick the cheapest or the one who takes less time to execute it. Consequently, it is necessary to make a balance between cost and time to pick the cheapest. It is intented that avatar saves money or time, depending on the situation. So it would let him:

- Have more time to do other activities or, at least, more free time. Under SACECA with the current implementation would mean to do more actions per day.
• Spend less money. Conseguently, avatar should work less time -to put the example in SACECA context-. As well as the point above, he would let to focus with actions which directly satify gauges. The work actions is considered to satisfy them indirectly because it doesn’t gauge values.

To apply the idea and know which object is the best for a specific gauge it’s necessary to force the use of that object during the simulation. Specifically, for each one found that modifies the gauge. It is required to do it for long time to get a good average. Finally, the captured data must give a result for each object to get the best one according to mentioned standards. It’s not intended to discard any object because it seems evident it won’t help. But that would add an expert knowledge to the solution. On the other hand, it opens the door to unexpected results, as well as, it won’t invalidate future project iteration as it keeps abstract enough.

For example, on table 5.1 there are several restaurants with its properties. All of them satisfy Hunger Gauge.

Each element is simulated for the avatar with a constraint to use that one. At the end, there will be available data extracted from simulation that will let SACECA find the best. As it is said before, it requires to be long to have trustable information and avoid anomalies. So once found, SACECA will pick it up everytime it needs to satisfy that gauge.

### 5.2 Proposal of Anticipation under Planning

As far as possible, a non technical explanation is done about the statement to implement.

#### 5.2.1 Overview

To achieve this assessment, a simulation is done with all avatar knowledge, inside its memory. This means it will behave as if it was the real world but with accelerated time and no 3D
5.2. PROPOSAL OF ANTICIPATION UNDER PLANNING

interface. Otherwise it’d take too much time. In doing so, N plans are found to recreate the avatars routine. Next step is to identify all those objects in memory that modify the selected Gauge. Once done, a new simulation is done for each element with the following condition:

- It can only use that object to satisfy the selected Gauge. The others are discarded.

At this point, there are \( K + 1 \) different objects -those found and the avatar itself with no constraints- with N plans each. It remains to decide which one is the best. Fortunately, there is a database with all simulated data in which a lot of data can be extracted. Thanks to them it will let SACECA find the best.

5.2.2 Details

Of all N plans found, they are grouped all those who are equal. Two plans are considered the same when all object services are the same. Consequently, a plan with different restaurants is considered the same. It has been decided to act like this to simplify. This only affects the first simulation of each avatar -with no constraints- because there is a loss of precision of time. On the other hand, plans simulations with objects are not affected because there is only one possible object to satisfy the gauge. As we’ve already defined in the previous point, each simulation knows how many times a plan has been executed. In addition, it’s also interesting to know how long takes to execute a plan. As SACECA already knows the execution time of an action, it only remains to calculate travel time from one place to an other. For this, we’ve simplified the calculation with a given estimation: the shortest length between two points. The formula is found below.

\[
\frac{1}{\text{walkspeed} \times 40} \times \text{length}
\]

The final value doesn’t represent a big part of total action time. Consequently, there is a small penalization when there are long distances between objects. To achieve a relative small number in comparison total time, it has been necessary to add constant values to adjust it. Nevertheless, they can be modified through code.

Table 5.2 has three plans with their execution time calculated and how many times is repeated by each branch. To calculate the average time of multiple plans is used a basic formula shown at school. But, any other method to get an average times of multiple plans or another way to quantify many plans would be valid as well.
Table 5.2: Same plan with different branches.

\[
\left( \frac{1}{5} \times 56 \text{ min.} \right) + \left( \frac{3}{5} \times 16 \text{ min.} \right) + \left( \frac{1}{5} \times 15 \text{ min.} \right) = 20 \text{ minutes}
\]

The final estimation to go to the Cinema is of 20 minutes of average time. The last operation can be expressed this way:

\[
\sum_{i=0}^{N} p_i \cdot \text{min}_i
\]

\( N \) number of branches

\( p(i) \) probability of execution for \( i \) branch

\( \text{min}(i) \) minutes of execution for \( i \) branch

Moreover, for each avatar Gauge there is a value with the total amount of units gained. Also, an index value is created with the following variables:

\[
\frac{\text{total gained units of a gauge}}{\text{average total time}}
\]

This value represents how efficient has been the avatar to gain units of a Gauge. Maybe because each executed service earns more units than other or the execution time is short. To sum up, SACECA is interested in looking for the highest possible value because it wants to maximize the mentioned properties.

Each avatar has an anticipation Goal defined at the beginning of SACECA, through this value it knows which gauge improve when possible. With the index found, it’s only necessary to compare with the original one. Finally, the one which surpass with the highest value will be selected. In consequence, that object will have higher priority over the rest of the same type when planning. But priority it’s just a number from 0 to a non-specified maximum. So the object with highest priority will be elected. As it has been already explained, if different
5.2. **PROPOSAL OF ANTICIPATION UNDER PLANNING**

objects have the same priority, then it’s randomly picked.

### 5.2.3 Architecture

The architecture of the proposed idea is explained in this section. All involved packages with their dependencies are represented in the following subsections.

#### 5.2.3.1 Code

All Java code is organized in packages inside SACECA. The figure 5.1 represents all packages used in Anticipation.

![Architecture of Anticipation](image)

**Figure 5.1: Architecture of Anticipation.**

Each module and class is in charge of specific task. Submodules have been thought to have little couple between them. All names try to be self-describing in order to facilitate code reading. The text below describes all code units created:

**Anticipation** This is considered the main package. It contains all subpackages and basic files necessary to run the Anticipation.

**LongAnticipation class** Main file to instantiate anticipation. It is in charge of everything because subpackages work for this class. It can be considered as a wrapper that communicates with the other elements. So there, there are all necessary methods and functions to work with anticipation.
Heuristic class  Contains auxiliar methods for Statistics module, found in Memory sub-package. But heuristic class doesn’t belong to anticipation memory. It has been conceived to be generic.

Memory  Contains all generated plans. La considerem una base de dades d’accions fetes. Per tant, totes les classes necessaries s’han agrupat en aquet package. Es considera prou important per la rellevancia que te de tenir el seu propi espai. A mes a mes, s’utilitzen estructures de dades no basiques i uniques dins del codi i el qual es bo no separar.

Memory Statistics  It creates the statistics about the memory information. Its purpose is to quantify anticipation memory. It is expected that more code will be added here. Consequently, it has its own package.

Drawer  It draws a graph of the memory. It’s used to represent memory and provide a easier interpretation of generated data. It seems a good way to show data as a circle graph.

Simulation  It simulates the reality. Takes care of all simulation processes. It also saves important data for the next step. Although it is only one class, it’s expected to grow in future iterations. For this reason, it has its own package. In addition, it has the appropriate functions to interact with the simulation.

Analyzer  Analyze and gives the results of the simulation. Once the simulation is done, there is a need to analyze output data. It groups all those classes with this finality. At the end, it returns the best object.

5.2.3.2 User Interface

On the other hand, new user interfaces have been added to integrate anticipation with SACECA. The user is able to explore the generated information. The figure 5.2 shows the architecture.

Each module and class is in charge of a specific task:

View package  Contains all SACECA user interfaces.

AnticipationWindow class  Shows the anticipation data in a Window.

Controller.Anticipation package  It has all the anticipation window actions.
5.2. PROPOSAL OF ANTICIPATION UNDER PLANNING

5.2.4 Design

In this section, it’s explained how things are done in a more detailed way. All classes are explained. It has been organized in accordance with the new packages created.

5.2.4.1 Analyzer

The package takes care to process and get a result. For that reason an algorithm class has been included. It pretends to be the basic class for, in a future, change it. Analyzer class is the main file, which calls the other ones because they are available to him. AnticipationGoal defines an anticipation Goal to satisfy, which is important to call. In fact, it’s inside the LongAnticipation constructor where AnticipationGoal is defined for each avatar. The next chunk of code shows this part.

```java
this.reasoningModule = new DefaultReasoningModule(this.agent);
this.agent.setReasoningModule(this.reasoningModule);
```
**Anticipation Goal**

```java
AnticipationGoal goal = new AnticipationGoal( 
    this.agent.getGauges().get(0).Objective.INCREASE, 0.05);
this.analyzer = new Analyzer( this.memory, this.agent, goal);

this.initiate();
```

ResultWrapper is a structure that groups all output simulation data. It’s a list because it lets to save different options.

### 5.2.4.2 Memory

Memory has two structures to save the given plans: VectorGeneric and WeightedGraph. The first saves all given plans. It also detects when a plan has been already saved. This way, it counts the repeated ones. The second saves plans as a graph. Relations between them, how many times some actions have been repeated -integer number over each edge- and, for the same action, the number of occurrences -it is showed in the graphs nodes with a numeric...
Figure 5.4: UML of Anticipation Memory package and its classes.
identifier after the name-. To deal with this data it has been created the Node class. The figure 5.5 is a graphical representation of the explained graph.

5.2.4.3 Statistics

It's in charge to create the statistical data of a given anticipation memory. It's necessary to defines concepts such as money and time, a part from avatar Gauges. Because it's not capable to infer those concepts. Average time and total time are used to calculate final output. But not money which it's only calculated for user purposes. updateData method is responsible to update this values. For that reason, every time it's necessary to display this data, it must be
5.2. PROPOSAL OF ANTICIPATION UNDER PLANNING

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Times</th>
<th>Index(Total/Av.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_time</td>
<td>1889 units</td>
<td>0</td>
<td>17.820755</td>
</tr>
<tr>
<td>gauge_time</td>
<td>540 units</td>
<td>27</td>
<td>5.09434</td>
</tr>
<tr>
<td>gauge_height</td>
<td>2300 units</td>
<td>23</td>
<td>21.698112</td>
</tr>
<tr>
<td>average_time</td>
<td>106 units</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Figure 5.7: Displayed statistics data.

Figure 5.8: UML of drawer package and its classes.

called. getIndex function is an other important method, because there is generated the value to compare with other plans. Finally, the class itself it automatically detects all gauges. This way, nobody has to take care of it.

5.2.4.4 Drawer

The package is in charge to calculate coordinates given a layout. In SACECA it’s implemented the circle layout but it’s possible to extend code subclassing super class. At every request it’s returned a following coordinate. This way, the anticipation memory can be represented schematically. In addition, it’s also an intuitive way the display data. The figure 5.9 is a screenshot of the circle layout.
Figure 5.9: Example Circle Layout.

Figure 5.10: UML Simulation.
5.2.4.5 Simulation

It makes the simulation until it finds N plans. It works with a copy of the original avatar. This way, any potential bug with the real avatar is removed. No inconsistencies will raise inside SACECA due to anticipation. It’s copied a big part of SACECA system. This means that it works with ticks. Each one represents one minute and all data is computed again. So, it’s increased by T ticks every time -despite of only a single unit in the real world-. At the end of it, it’s called the reason module which is expected to produce a new Goal. In consequence, a new plan is created and the result is gathered. The simulation class implements the Iterator interface -hasNext and Next functions-. So, it’s only necessary to call next function to request a new plan.

```java
public boolean hasNext() {
    // Evaluate Gauges
    this.evaluate();

    // No Goals, then advance time N times
    if (!this.hasGoals()) {
        this.advanceSteps(500);

        // Evaluate Gauges
        this.evaluate();
    }

    // Avatar is dead
    if (this.isDead()) {
        return false;
    }

    // Conclude if there are goals found
    if (!this.hasGoals()) {
        return false;
    }

    // Apply effects on the plan
    try {
        this.executePlan();
        this.advanceSteps(10);
    } catch (UnreachableObjectException e) {
```
At the code, 500 units are advanced for every requested plan. This value guarantees that one of the gauges will cross their threshold. Once it has been planned and solved, those effects are applied to the simulated avatar.

5.2.4.6 User Interface

The figure 5.11 shows a screenshot with the new tab available in the main SACECA window. It has two buttons. First shows anticipation memory in a graph. The other opens a new window with the anticipation data.

The figure 5.12 shows the anticipation window. All generated data during the anticipation process is displayed here. Each element is explained with more details in the following list:

1. The list contains all objects which satisfy the gauge. In this example, it is hunger. The last
5.2. PROPOSAL OF ANTICIPATION UNDER PLANNING

Figure 5.12: Avatar Anticipation Window.

1. One is the avatar with the original simulated data -no restrictions-. There is also the * symbol which points the best result found. Otherwise, the symbol is not shown.

2. It shows the anticipation memory graph in a new window for the selected avatar.

3. The table contains all found plans during the simulations and the identification of which object serves the services. The first column shows how many times has been executed.

4. The statistics table shows all stats calculated. Each column means from the first to the last: the variable name, accumulated units, how many times have been executed or 0 if it isn’t applicable and the index value -accumulated units divided by the total average time-.

5. The object properties selected in the list.
Chapter 6

Anticipation Experiments under Planning

Once everything is coded and all bugs fixed, it’s important to prove that the statement is valid. Or at least, try to understand the results. For this, we’ve run several tests given certain conditions explained below.

6.1 Experiment Results

Anticipation has been tested with all avatars. But it has only been tested with Goal Hunger because there is only one Gauge which offers different ways to satisfy it. Specifically, three different restaurants. Each has a different execution time for the meal.

- obj20 takes 15 minutes
- obj3030 takes 40 minutes
- obj6 takes 60 minutes

All avatars gain the same amount of units, one hundred. Consequently, the best option is the object obj20 because it’s the one who offers the fastest meal by far.

6.2 Critical Review of the Experiment

However, SACECA results show that sometimes obj3030 is the best. Other times, it isn’t found the best. Despite of those non expected results, the majority of them chose obj20 -considered
the right answer. Figure 6.1 is a screenshot where any object is selected. SACECA marks the best result with an * in objects list. Therefore, when there isn't any symbol means that the original is the best one.

![Figure 6.1: Anticipation Window Fail.](image)

There is an exception with avatar 1 because it can only execute one plan while simulating. Apparently, it’s a problem related with its memory. At the beginning of SACECA he doesn’t know enough elements which surround him. Therefore, he isn’t capable to satisfy his needs at that time. But this changes when perception module detects the environment surrounding him. So then, he is able to live like other avatars. At that time, the anticipation is too late. However, in the next SACECA iteration may be reviewed to call anticipation in other circumstances.

Figure 6.2 is a real example which has succeeded finding which is the best object to use.
Figure 6.2: Anticipation Window Success.
Chapter 7

Project Planning

It was not easy to plan how this project would evolve in the future, mostly because there was some research involved where the timeline was just an approximation. In figure 7.1 we have the initial schedule.

<table>
<thead>
<tr>
<th>Nom</th>
<th>Finis</th>
<th>2012: M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>130h</td>
<td></td>
</tr>
<tr>
<td>SACECA</td>
<td>19h</td>
<td></td>
</tr>
<tr>
<td>Platform INT</td>
<td>14d</td>
<td></td>
</tr>
<tr>
<td>Current Planning Algorithm</td>
<td>5d</td>
<td></td>
</tr>
<tr>
<td>Planning</td>
<td>47d</td>
<td></td>
</tr>
<tr>
<td>Conception</td>
<td>20d</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>27d</td>
<td></td>
</tr>
<tr>
<td>Java adaptation</td>
<td>14d</td>
<td></td>
</tr>
<tr>
<td>Prolog adaptation</td>
<td>7d</td>
<td></td>
</tr>
<tr>
<td>P. Unit Test</td>
<td>6d</td>
<td></td>
</tr>
<tr>
<td>Anticipation</td>
<td>50d</td>
<td></td>
</tr>
<tr>
<td>A. Conception</td>
<td>25d</td>
<td></td>
</tr>
<tr>
<td>A. Implementation</td>
<td>15d</td>
<td></td>
</tr>
<tr>
<td>A. Unit Test</td>
<td>10d</td>
<td></td>
</tr>
<tr>
<td>Final Document</td>
<td>14d</td>
<td></td>
</tr>
<tr>
<td>Bug Fixing</td>
<td>130h</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.1: Prevision of dedication in Project

But of course reality is seldom like we expected it to be, here’s some representation of what it turned out to be in the end. See figure 7.2.

If we compare both Gantt diagrams -prediction and final dedication Gantts-, we realize that project length has extended for almost 4 months more.
In part, it comes from data lose which it took almost one month to recover all work because it was necessary to write planning again. Actually both parts, Java and Prolog. The projects hard drive broke and there weren’t any copy. That wasn’t a smart decision.

On the other side, planning phase under SACECA broke all previsions because it has doubled the length. The final goal was too open and the thematic too large. It was conducted a papers research about planning. We mainly focus on those papers with planning uncertainty. However, it hasn’t been used with SACECA. From practical point of view we could consider a waste of time because. At then end, It has been used a classical algorithm.

Documentation phase has taken more time than the expected. Like planning, it has been doubled as well. In part, it took several improvements because it wasn’t satisfying the minimum requirements to deliver. From another side, it was necessary to specify which documents were necessary to create and how because during that time there was enough free time to do them.

Although it is not specifically shown, it took time to solve non expected bugs. All throughout the project it was necessary to fix them because they were appearing at the time that new features were added. Having to work in very specific parts of SACECA led to appear. However, this is inherent to all softwares because programmers are persons which, unfortunately, make mistakes.

But not all stages have gone wrong. Anticipation has worked on time. Maybe because the
project wasn’t on schedule. It took to work hard. At the end, there was a good performance. Like in planning, there have been a research. Nevertheless, final implementation phase has been basic and it hasn’t been required a high complex tests to validate it.
Part IV

 Perspectives and conclusion
Chapter 8

Perspectives

For next iterations it would be interesting to add more anticipation examples. This way the test range would be bigger and, it’d help to confirm or refute the correctness of the anticipation in a more generic way. On the other hand, it remains to fix the mistake. Where the result isn’t always the best option but, at least, isn’t the worth one.

Moreover, there is a certain need to find a way to speed up the calculations in advanced at the beginning of SACECA. Because at certain point, it will require a lot of time when SACECA options growth.

Another point, anticipation could be executed again while avatar is sleeping. It would make sense because it could update those actions that, after a while, they aren’t useful anymore due to new learned objects.
Chapter 9

Conclusion

Planning part works at 100% so, avatars have efficient plans thanks to Prolog integration. Multi-threading part doesn’t work under SACECA through Prolog. Subsection 4.3.7 justifies it. But to summarize, it’s due to Java to Prolog library used.

With Anticipation, the range of the experiment itself isn’t wide enough to show how generic is the proposal. However, with the obtained results they clearly show there is a problem with the statement. My guess is about how a plan is considered equal to another. Maybe it has been a bad idea to simplify plans and not to distinguish them by their identification object code. At the time of the experiment, plans are considered equal when their services are the same. It doesn’t matter which object serves it. A first implication, is that an original plan will believe that an avatar has used N times the same restaurant. For this reason, when it compares with simulated data of other objects there isn’t an improvement because it has already executed the original plan with the first restaurant used -in this example the obj20 which is the fastest-. Although the results are not the desired, the results are still showing most of the times that the best object is obj20.

However, planning nor anticipation phases haven’t been easy. Mainly because it was necessary to do a research. Which it never guarantees that is going to work as expected. Several problems have appeared and they were necessary to be solved.

New bugs have found and fixed. Some has been difficult to solve but, at the end, always fixed. In consequence, new code has been tried to be free of errors because in other itera-
tions they don’t have to deal with it. But in this field it’s always difficult and it’s part of the job.

The project has gone out of schedule. Maybe, it was necessary to define specific and detailed goals and dates because the thematic was too open. Nevertheless, data lose didn’t help because it took almost one month to recover it.

At the end, I have a good impression about the project because it allowed me to become part a part of it. So I could participate in different stages of what it could be a real industry job. It also let me to put in practice all my acquired knowledges of my university.
Appendices
Appendix A

Module Planning

A.1 DefaultPlan

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

/**
 * A plan implemented using a LinkedList.
 */
public final class DefaultPlan implements Plan {

    private List<PlanElement> backingCollection;

    /**
     * Instantiates a new list plan.
     */
    public DefaultPlan() {
        super();
        this.backingCollection = new LinkedList<PlanElement>();
    }

    @Override
    public Iterator<PlanElement> iterator() {
        return this.backingCollection.iterator();
    }
}
```
@Override
defaulDefaultPlan cons(PlanElement planElement) {
    (LinkedList<PlanElement>) this.backingCollection).addFirst(planElement);
    return this;
}

@Override
defaulDefaultPlan append(PlanElement planElement) {
    (LinkedList<PlanElement>) this.backingCollection).addLast(planElement);
    return this;
}

@Override
public String toString() {
    StringBuilder builder = new StringBuilder("Plan:\n");
    int num = 0;
    for (PlanElement planElement : this.backingCollection) {
        builder.append("\t");
        builder.append(num);
        builder.append(' ');
        builder.append(planElement.toString());
        builder.append('\n');
        num++;
    }
    return builder.toString();
}

@Override
public PlanElement get(int i) {
    return this.backingCollection.get(i);
}

@Override
public int size() {
    return this.backingCollection.size();
}

/**
 * Compare to plans in generic way (only services)
 */
@Override
public boolean equalsGeneric(Object obj) {
    if (obj == null) {
        return false;
    }
    if (obj == this) {
        return true;
    }
    if (! (obj instanceof DefaultPlan)) {
        return false;
    }
    DefaultPlan plan = (DefaultPlan) obj;

    // if different sizes, then they are not equal
    if (this.backingCollection.size() != plan.size()) {
        return false;
    }

    // Retrieves iterators
    Iterator<PlanElement> source = this.backingCollection.iterator();
    Iterator<PlanElement> toCompare = plan.iterator();

    // compare each element of the plan
    while (source.hasNext() && toCompare.hasNext()) {
        PlanElement planElementSource = source.next();
        PlanElement planElementToCompare = toCompare.next();

        // Not equal, then plans are different
        if (!planElementSource.equalsGeneric(planElementToCompare)) {
            return false;
        }
    }

    return true;
}
* Returns unique code for the plan.
*/

@Override
public int hashCode() {
    int value = 0;

    for (PlanElement planElement : this.backingCollection) {
        if (planElement.getService().getName().equals("walkTo")) {
            continue;
        }
        value += planElement.hashCode();
    }
    return value;
}

/**
 * DIRTY. It call equalsGeneric method.
 */

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (obj == this) {
        return true;
    }
    if (! (obj instanceof DefaultPlan)) {
        return false;
    }

    return this.equalsGeneric(obj);
}

/**
 * The number of actions of the action "ActionName"
 * @param ActionName String (Case sensitive)
 * @return int \geq 0
 */
A.2. DEFAULTPLANNINGMODULE

*/
@Override
public int getNumberOfActions(String ActionName) {
    int counter = 0;
    for (PlanElement planElement : this.backingCollection) {
        if (planElement.getService().getName().equals(ActionName)) {
            counter++;
        }
    }
    return counter;
}

A.2 DefaultPlanningModule

class DefaultPlanningModule {

    package fr.n7.saceca.u3du.model.ai.agent.module.planning;

    import java.util.HashMap;
    import java.util.List;
    import java.util.Map;
    import org.apache.log4j.Logger;
    import edu.uci.ics.jung.graph.UndirectedSparseGraph;

    import fr.n7.saceca.u3du.model.Model;
    import fr.n7.saceca.u3du.model.ai.AI;
    import fr.n7.saceca.u3du.model.ai.agent.Agent;
    import fr.n7.saceca.u3du.model.ai.agent.
        module.perception.PerceptionModule;
    import fr.n7.saceca.u3du.model.ai.agent.
        module.planning.anticipation.LongAnticipation;
    import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.Goal;
    import fr.n7.saceca.u3du.model.ai.graph.GraphSolver;
    import fr.n7.saceca.u3du.model.ai.graph.WeightedEdge;
    import fr.n7.saceca.u3du.model.ai.object.WorldObject;
    import fr.n7.saceca.u3du.model.ai.service.ExecutionStatus;
    import fr.n7.saceca.u3du.model.ai.service.Service;
    import fr.n7.saceca.u3du.model.ai.statement.ExecutionMode;
import fr.n7.saceca.u3du.model.console.CommandException;
import fr.n7.saceca.u3du.model.util.Couple;
import fr.n7.saceca.u3du.model.util.Oriented2DPosition;
import fr.n7.saceca.u3du.prolog.PrologInit;
import fr.n7.saceca.u3du.util.Log;

/**
 * The Class DefaultPlanningModule.
 *
 * @author Sylvain Cambon
 */
public abstract class DefaultPlanningModule implements PlanningModule {

    /** The Constant logger. */
    private static Logger logger = Logger.getLogger(DefaultPlanningModule.class);

    /** The Constant COLLECTION_SHUFFLE_ENABLED. */
    public static final boolean COLLECTION_SHUFFLE_ENABLED = true;

    /** The Constant NO_PLAN_FOUND. */
    protected static final Plan NO_PLAN_FOUND = null;

    /** The Constant NOT_POSSIBLE_TO_FIND_A_PLAN. */
    protected static final Plan NOT_POSSIBLE_TO_FIND_A_PLAN = null;

    /** The Constant PLAN_MAX_DEPTH. */
    public static final int PLAN_MAX_DEPTH = 50;

    /**
     * The agent.
     * @uml.property name="agent"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    protected Agent agent;

    /**
     * The current goal.
     * @uml.property name="currentGoal"
     * @uml.associationEnd
     */
    protected Goal currentGoal = null;
A.2. DEFAULTPLANNINGMODULE

```java
/**
 * The current plan.
 * @uml.property name="currentPlan"
 * @uml.associationEnd
 */
protected Plan currentPlan = null;

/**
 * The current index.
 * @uml.property name="currentIndex"
 */
private int currentIndex = 0;

/**
 * The anticipation mechanism
 * @uml.property name="longAnticipation"
 * @uml.associationEnd
 */
private LongAnticipation longAnticipation = null;

/**
 * @uml.property name="memoryKnowledge"
 */
private List<WorldObject> MemoryKnowledge = null;

private boolean forcePlan = false;

/**
 * Instantiates a new default planning module.
 * @param agent the agent
 */
public DefaultPlanningModule(Agent agent) {
    this.agent = agent;

    this.setMemoryKnowledges(this.agent.getMemory().getKnowledges());

    // Connects to Prolog VM
    try {
```
PrologInit.getInstance(this.agent.getId());
}

catch (InterruptedException e) {
   e.printStackTrace();
}

/**
 * @return
 * @uml.property name="longAnticipation"
 */
@ Override
public LongAnticipation getLongAnticipation() {
   return this.longAnticipation;
}

/**
 * @param longAnticipation
 * @uml.property name="longAnticipation"
 */
@ Override
public void setLongAnticipation(LongAnticipation longAnticipation) {
   this.longAnticipation = longAnticipation;
}

/**
 * Plans and executes.
 */
@ Override
public synchronized void planAndExecute() {
   // We forced a plan execution and there is no element in it
   if (this.forcePlan && this.currentPlan == null) {
      return;
   }
   // There is an element in the plan
   else if (this.forcePlan &&
            this.currentGoal != null &&
            !this.agent.getMemory().getGoals().isEmpty() &&
            this.currentGoal.equals(this.agent.getMemory().getGoals().get(0)) &&
            this.currentPlan != null) {
   } else {
A.2. DEFAULTPLANNINGMODULE

&& this.currentPlan.size() >= this.currentIndex) {
    this.execute();
} else if (!this.agent.getMemory().getGoals().isEmpty()) {
    this.currentGoal = this.agent.getMemory().getGoals().get(0);

    // Java algorithm
    this.currentPlan = this.buildPlan(this.agent, this.currentGoal, 0, PLAN_MAX_DEPTH);
}
}

/**
 * Handles walk to service setup, if needed.
 * The memory consumer is directly accessed. The agent
 * as well as its memory are moved (if needed).
 * @param provider
 * the provider
 * @param consumer
 * the consumer
 * @param maxDist
 * the max distance
 * @return The plan element to be added to move
 * to the next place. Null means there is no need
 * to move.
 * @throws UnreachableObjectException
 * If no path is found
 */
public PlanElement handleWalkTo(WorldObject provider, Agent consumer, int maxDist)
    throws UnreachableObjectException {
    Oriented2DPosition initialConsumerPosition = \\ consumer.getMemory().getKnowledgeAboutOwner().getPosition();
    Oriented2DPosition providerPosition = provider.getPosition();

    final float distance = initialConsumerPosition.distance(providerPosition);
    if (maxDist > distance) {
        // The service can directly be used
        return null;
    } else {
        // The service cannot directly be used, a path has to be found first
AI ai = Model.getInstance().getAI();

UndirectedSparseGraph<WorldObject, WeightedEdge> walkableGraph = ai.getWorld().getWalkableGraph();

// Looking for the closest walkable item to the agent
WorldObject closestWalkableToAgent = \n    ai.getWorld().getClosestWalkable(consumer.getPosition());

// Looking for the closest walkable item to the object
WorldObject closestWalkableToObject = \n    ai.getWorld().getClosestWalkable(provider.getPosition());

try {
    boolean existsPath = \n        GraphSolver.forGraph(walkableGraph).\n    existPath(closestWalkableToAgent, closestWalkableToObject);
    final float distanceAfterWalk = \n        closestWalkableToObject.getPosition().distance(providerPosition);

    final boolean distanceOk = distanceAfterWalk <= maxDist;

    if (existsPath && distanceOk) {
        // A path exists to a place close enough to the object
        Service walkTo = \n            ai.getEntitiesFactoryMaterials().getServiceRepository().get("walkTo");
        // constant
        Map<String, Object> parameters = new HashMap<String, Object>();
        // parameters.put("source", closestWalkableToAgent);
        parameters.put("destination", closestWalkableToObject);
        PlanElement planElement = \n            new PlanElement(walkTo, closestWalkableToAgent, parameters);
        walkTo.execute(closestWalkableToAgent, consumer, \n            parameters, ExecutionMode.VIRTUAL);
        // Ensures that the job is well done
        consumer.setPosition(closestWalkableToObject.getPosition());

        consumer.getMemory().getKnowledgeAboutOwner().\n            setPosition(closestWalkableToObject.getPosition());

        return planElement;
    }
} catch (IllegalArgumentException iae) {
    // Should not occur
}
A.2. DEFAULTPLANNINGMODULE

```java
logger.error("A node was not in the sidewalk graph...", iae);
}

// Too far and unreachable or error

throw new UnreachableObjectException();

/**
 * Execution part.
 */

public void execute() {
    PlanElement planElement = this.currentPlan.get(this.currentIndex);
    // FIXME : pas d'exec
    ExecutionStatus status = planElement.execute(this.agent, ExecutionMode.REAL);

    switch (status) {
        case CONTINUE_NEXT_TIME:
            // NOP
            break;

        case SUCCESSFUL_TERMINATION:
            this.currentIndex++;
            if (this.currentIndex == this.currentPlan.size()) {
                if (!this.forcePlan) {
                    this.currentGoal.setReachable(true);
                    this.currentGoal.setReached(true);
                    this.agent.getMemory().getPastPlans().put(this.currentGoal, this.currentPlan);
                }
                this.currentGoal = null;
                this.currentPlan = null;
            }
            this.currentIndex = 0;
            break;

        case FAILURE:
            this.currentPlan = null;
            this.currentIndex = 0;
            Log.debug("SERVICE UNUSABLE");
            break;
    }
}
```
APPENDIX A. MODULE PLANNING

```java
break;
default:
    break;
}

/**
 * Gets the storage label.
 * @return the storage label
 */
@Override
public String getStorageLabel() {
    return DefaultPlanningModule.class.getName();
}

/**
 * To string.
 * @return the string
 */
@Override
public String toString() {
    return "DefaultPlanningModule [currentGoal=" + this.currentGoal + ", currentPlan=" + this.currentPlan + ", currentIndex=" + this.currentIndex + "]";
}

@Override
public Plan getPlan() {
    return this.currentPlan;
}

/**
 * Gets the current goal.
 * @return the current goal
 * @uml.property name="currentGoal"
 */
@Override
public final Goal getCurrentGoal() {
    return this.currentGoal;
}
```


A.2. DEFAULTPLANNINGMODULE

```java
/**
 * Gets the current plan.
 * @return the current plan
 * @uml.property name="currentPlan"
 */
@Override
public final Plan getCurrentPlan() {
    return this.currentPlan;
}

/**
 * Gets the current index.
 * @return the current index
 * @uml.property name="currentIndex"
 */
@Override
public final int getCurrentIndex() {
    return this.currentIndex;
}

@Override
public synchronized void forceExecution(String serviceName, \List<Object> params, boolean clearPreviousPlan)
    throws CommandException {
    Agent agent = this.agent;
    // If previous forced services were added to the current plan, we virtually apply their effects to the "agent" variable.
    if (this.forcePlan && this.currentPlan != null && \this.currentPlan.size() > 0) {
        agent = this.agent.deepDataClone();

        for (PlanElement planElement : this.currentPlan) {
            planElement.getService().execute(planElement.getProvider(), \agent, planElement.getParameters(), ExecutionModeVIRTUAL);
        }
    }
    // We find the nearest service around the "agent" variable
```
Couple<Service, WorldObject> serviceAndProvider = \[
    Model.getInstance().getAI().getWorld().getFirstNearServiceAndProvider(serviceName, agent, PerceptionModule.NEARBY_OBJECTS_RANGE);
if (serviceAndProvider == null) {
    throw new CommandException("There is no service called \"" + serviceName + \"\ near the agent.");
}

Service service = serviceAndProvider.getFirstElement();
WorldObject provider = serviceAndProvider.getSecondElement();

// We fill its parameters
Map<String, Object> parameters = new HashMap<String, Object>();
if (serviceName.equalsIgnoreCase("walkto")) {
    if (params.isEmpty()) {
        throw new CommandException("The service \"" + serviceName + \"\ must have a position parameter.\"");
    }
    parameters.put("destination", params.get(0));
}

// We interrupt the possible current agent animation
agent.interruptAnimation();

// We clear the current plan, and we add the service
this.forcePlan = true;
if (clearPreviousPlan) {
    this.currentPlan = new DefaultPlan();
    this.currentIndex = 0;
    this.currentGoal = null;
}

this.currentPlan.append(new PlanElement(service, provider, parameters));

@Override
public synchronized void enablePlanning() {
    this.forcePlan = false;
}
@Override
public void setMemoryKnowledges(List<WorldObject> knowledges) {
    this.MemoryKnowledge = knowledges;
}

public List<WorldObject> getMemoryKnowledges() {
    return this.MemoryKnowledge;
}

A.3 Plan

package fr.n7.saceca.u3du.model.ai.agent.module.planning;

public interface Plan extends Iterable<PlanElement> {

    /**
     * Adds an element at the beginning of the plan.
     * @param element the element
     * @return "this"
     */
    public Plan cons(PlanElement element);

    /**
     * Adds an element at the ends of the plan.
     * @param element the element
     * @return "this"
     */
    public Plan append(PlanElement element);

    /**
     * Gets an element
     * @param i
     */
* the i
* @return the service
*/
public PlanElement get(int i);

/**
 * Gets the size.
 * @return the int
 */
public int size();

/**
 * Compare to two plans with generic way
 * @param obj
 * @return true if equal or false it not.
 */
public boolean equalsGeneric(Object obj);

public int getNumberOfActions(String ActionName);
public class PlanElement {

  /*
   * @author Sylvain Cambon
   */

  /**
   * The service.
   * @uml.property name="service"
   * @uml.associationEnd multiplicity="(1 1)"
   */
  private Service service;

  /**
   * The provider.
   * @uml.property name="provider"
   * @uml.associationEnd multiplicity="(1 1)"
   */
  private WorldObject provider;

  /**
   * The parameters.
   * @uml.property name="parameters"
   * @uml.associationEnd qualifier="key:java.lang.String fr.n7.saceca.u3du.model.ai.object"
   */
  private Map<String, Object> parameters;

  /**
   * Gets the service.
   * @return the service
   */
  public final Service getService() {
    return this.service;
  }

  /**
   * Sets the service.
   * @param service the new service
   */
  public final void setService(Service service) {
this.service = service;
}

/**
 * Gets the provider.
 * @return the provider
 * @uml.property name="provider"
 */
public final WorldObject getProvider() {
    return this.provider;
}

/**
 * Sets the provider.
 * @param provider the new provider
 * @uml.property name="provider"
 */
public final void setProvider(WorldObject provider) {
    this.provider = provider;
}

/**
 * Gets the parameters.
 * @return the parameters
 */
public final Map<String, Object> getParameters() {
    return this.parameters;
}

/**
 * Sets the parameters.
 * @param parameters the new parameters
 */
public final void setParameters(Map<String, Object> parameters) {
    this.parameters = parameters;
}
/**
A.4. PLANELEMENT

* Instantiates a new plan element.
*
* @param service
*     the service
* @param provider
*     the provider
* @param parameters
*     the parameters
*/

public PlanElement (Service service, WorldObject provider, 
Map<String, Object> parameters) {
    super ();
    this . service = service;
    this . provider = provider;
    this . parameters = parameters;
}

/**
* To string.
* *
* @return the string
*/
@Override
public String toString () {
    String providerString = this . provider == null ? null : this . provider . toShortString () ;
    return "PlanElement [service=" + this . service . getName () + ", provider=" + providerString + ", parameters=" + (this . parameters == null ? "{}" : "{...}") + "]" ;
}

/**
* Executes the elements, i.e. its service.
* *
* @param consumer
*     the consumer
* @param mode
*     the mode
* @return the execution status of this element.
*/
public ExecutionStatus execute (WorldObject consumer, ExecutionMode mode) {
    return this . service . execute (this . provider, consumer, this . parameters, mode);
}
```java
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (obj == this) {
        return true;
    }
    if (!(obj instanceof PlanElement)) {
        return false;
    }
    PlanElement element = (PlanElement) obj;
    boolean parameters = false;
    if (this.parameters != null) {
        // There was a bug
        parameters = element.getParameters() != null &&
                    this.parameters.equals(element.getParameters());
    } else if (element.getParameters() == null) {
        // To compare parameters are null in both objects
        parameters = true;
    }
    return this.service.equals(element.getService()) &&
           this.provider.equals(element.getProvider()) && parameters;
}

/**
 * Only compares name service as we want to compare generic
 * @param obj PlanElement
 * @return boolean, true is equal
 */
public boolean equalsGeneric(Object obj) {
    if (obj == null) {
        return false;
    }
    if (obj == this) {
        return true;
    }
```
A.5. PLANNINGMODULE

```java
if (!(obj instanceof PlanElement)) {
    return false;
}
PlanElement element = (PlanElement) obj;

return this.service.getName().equals(element.getService().getName());

@Override
public int hashCode() {

    // MD5 of service name
    byte[] thedigest = null;
    try {
        byte[] bytesOfString = this.getService().getName().getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("MD5");
        thedigest = md.digest(bytesOfString);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }

    // byte to int
    long value = 0;
    for (int i = 0; i < thedigest.length; i++) {
        value += (thedigest[i] & 0xff) << (8 * i);
    }

    // We want to identify only the action. No matter to which object give service.
    // long serviceProviderHash = this.getPlanElement().getProvider().getId();

    return ((int) value);
}
```

A.5 PlanningModule

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning;
```
import java.util.List;

import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.LongAnticipation;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.Goal;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;
import fr.n7.saceca.u3du.model.console.CommandException;
import fr.n7.saceca.u3du.model.util.io.storage.Storable;

/**
 * The Interface PlanningModule.
 * @author Sylvain Cambon & Jerome Dalbert
 */
public interface PlanningModule extends Storable {

/**
 * Plan and execute.
 */
public void planAndExecute();

/**
 * Builds the plan.
 * @param agent the agent
 * @param goal the goal
 * @param currentDepth the current depth
 * @param maxDepth the max depth
 * @return the plan
 */
public Plan buildPlan(Agent agent, Goal goal, int currentDepth, int maxDepth);

/**
 * Gets the plan.
 * @return the plan
 * @uml.property name="plan"
 * @uml.associationEnd
*/
public Plan getPlan();

/**
 * Gets the current goal.
 * @return the current goal
 * @uml.property name="currentGoal"
 * @uml.associationEnd
 */
public Goal getCurrentGoal();

/**
 * Gets the current plan.
 * @return the current plan
 * @uml.property name="currentPlan"
 * @uml.associationEnd
 */
public Plan getCurrentPlan();

/**
 * Gets the current index.
 * @return the current index
 */
public int getCurrentIndex();

/**
 * Forces the execution of a service, disabling the planning.
 * @param serviceName the service name
 * @param params the params
 * @param clearPreviousPlan the clear previous plan
 * @throws CommandException the command exception
 */
public void forceExecution(String serviceName, List<Object> params, boolean clearPreviousPlan, throws CommandException;
/**
 * Enables the planning.
 */

public void enablePlanning();

public LongAnticipation getLongAnticipation();

/**
 * @param longAnticipation
 * @uml.property name="longAnticipation"
 */
public void setLongAnticipation(LongAnticipation longAnticipation);

/**
 * Sets the memory to use in planning
 * @param knowledges
 * @uml.property  name="knowledges"
 */
public void setMemoryKnowledges(List<WorldObject> knowledges);
Appendix B

Module Planning Prolog

B.1 PrologWrapper

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.prolog;

import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.DefaultPlan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.UnreachableObjectException;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PrologDFSAlgorithm;
import fr.n7.saceca.u3du.model.ai.object.properties.Property;
import fr.n7.saceca.u3du.model.ai.object.properties.UnknownPropertyException;
import fr.n7.saceca.u3du.model.ai.service.Service;
import fr.n7.saceca.u3du.model.ai.statement.ExecutionMode;
import fr.n7.saceca.u3du.prolog.output.PIAction;
import fr.n7.saceca.u3du.prolog.output.PrologResult;
import jpl.Term;
```
public class PrologWrapper {

  /** The class communicates Prolog <-> Java. */
  @author Albert Jornet Puig
  */

  public class PrologWrapper {

    /** The Name of the Prolog function to call planning. */
    public final static String PROLOG_FUNCTION = "getplan";

    /**
     * @uml.property name="agent"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    private Agent agent = null;

    private PrologDFSAlgorithm prologDFS = null;

    /** The Constant NO_PLAN_FOUND. */
    private static final Plan NO_PROLOG_PLAN_FOUND = null;

    public PrologWrapper(Agent agent, PrologDFSAlgorithm prologDFS) {
      this.agent = agent;
      this.prologDFS = prologDFS;
    }

    /**
     * Transforms Avatar memory to prolog.
     */
    @return String
    */

    public String MemoryToProlog(List<WorldObject> agentKnowledges) {
      /**
       * object(obj<id>, modelname, service)
       * service(name service)
       * position(obj<id>, x, y)
       */
    }
  }
}
B.1. PROLOGWRAPPER

```java
String objectFacts = "", positionFacts = "", servicesFacts = "";
String propertiesAvatar = "", propertiesObjects = "";
String objectsPriority = "";
Set<String> allservices = new HashSet<String>();

for (WorldObject object : agentKnowledges) {

    // Apply constraints
    Collection<Service> services = object.getServices();
    String tmplID = "avatar" + Long.toString(this.agent.getId()) + \"", obj" + Long.toString(object.getId());
    String modelName = object.getModelName().toLowerCase();

    // position facts in prolog
    positionFacts += "position(" + tmplID + ", " + (int)object.getPosition().x +\"
    ", " + (int)object.getPosition().y + ").\n"

    // objects planning priority
    objectsPriority += "priority(" + tmplID + ", " + object.getPriority() + ").\n"

    // object properties
    propertiesObjects += this.PropertiesToProlog("object.properties", object) + "\n\n"

    for (Service service : services) {
        if (service.getName().equals("walkTo")) {
            continue;
        }

        // object facts in prolog
        objectFacts += "worldobject(" +tmplID + ", " + modelName + ", " + service.getName() + ").\n"
        allservices.add(service.getName().toLowerCase()); // set of services
    }
}

// List of services in prolog
for (String service : allservices) {
    servicesFacts += "service(avatar" + this.agent.getId() + ", " + service + ").\n";
}
```
APPENDIX B. MODULE PLANNING PROLOG

// List of avatar properties to prolog
// propertiesAvatar += this.PropertiesToProlog("avatar.properties", this.agent);

return "\n" + objectFacts + "\n" + servicesFacts + "\n" \n + positionFacts + "\n" + propertiesAvatar \n + "\n" + propertiesObjects + "\n" + objectsPriority;
}

/**
 * Turns object properties to prolog facts
 * @param Tag
 * @param object
 * @return String to prolog facts
 */
public String PropertiesToProlog(String Tag, WorldObject object) {
    String out = "";

    // List of object properites to prolog
    Collection<Property<?>> properties = object.\
        getPropertiesContainer().getProperties();
    String tmp = "";

    for (Property<?> property : properties) {
        // To know if it's a string or a number, must treat this case for prolog
        try {
            double value = Double.parseDouble(property.getValue().toString());
            int val = (int) value;
            tmp = Integer.toString(val);
        } catch (Exception nFE) {
            tmp = "'" + property.getValue() + "'";
        }

        out += Tag + "(avatar" + this.agent.getId() + ", obj" +\n            object.getId() + "," + property.getModel().getName() +\n            "," + tmp + ").\n";
    }
}
B.1. PROLOGWRAPPER

```java
return out;
}

/**
 * Transforms goal gauge to Prolog
 * @param currentGoal
 * @return String
 */
private String AimGaugeToProlog(Goal currentGoal) {
    String out = "";
    String goal = currentGoal.getSuccessCondition().replace("id "+
    this.agent.getld() + "." , "").replace("i_gauge", "");

    String[] rawdata = goal.split("=>");
    String avatar = "avatar" + this.agent.getld();

    if (rawdata[0].equals("tiredness")) {
        out += "caim(" + avatar + ",0,0," + rawdata[1] + ",0,0,0).";
    } else if (rawdata[0].equals("thirst")) {
        out += "caim(" + avatar + ",0,0,0," + rawdata[1] + ",0,0).";
    } else if (rawdata[0].equals("hunger")) {
        out += "caim(" + avatar + ",0,0,0,0," + rawdata[1] + ",0).";
    } else if (rawdata[0].equals("happiness")) {
        out += "caim(" + avatar + ",0,0,0,0,0," + rawdata[1] + ").";
    } else {
        // Should never achieve here
        out += "caim(" + avatar + ",0,0,-1,-1,-1,0).";
    }

    return out;
}

/**
 * Transforms current SACECA gauges to Prolog
 * @return String
 */
private String currentGaugesToProlog() {
    return out;
}
```

String out = "current(avatar" + this.agent.getId() + "," +
    this.getCurrentProperty(this.agent, "c.Human_cash")
    + "," + this.getCurrentProperty(this.agent, "c.AccountOwner.account") + ","
    + this.getCurrentProperty(this.agent, "i.gauge_tiredness") + ","
    + this.getCurrentProperty(this.agent, "i.gauge_thirst") + ","
    + this.getCurrentProperty(this.agent, "i.gauge_hunger") + ","
    + this.getCurrentProperty(this.agent, "i.gauge_happiness") + ")");

    return out;
}

/**
 * Get int value of a property
 *
 * @param property
 *   String
 * @return int
 */
private int getCurrentProperty(WorldObject obj, String property) {
    int out;

    try {
        Property<?> prop = obj.getPropertiesContainer().getProperty(property);
        out = (int) Double.parseDouble(prop.getValue().toString());
    } catch (UnknownPropertyException e1) {
        out = -1;
    }

    return out;
}

/**
 * Transforms from prolog output to Saceca plan
 *
 * @param result
 *   JPL.Term
 * @param agent
 *   Agent
 * @param _goal
 *   Goal
 *
* @return Plan or null if there isn’t any plan
*/

public Plan PrologToSacecaPlan(Term result, Agent agent, Goal _goal) {

    // input is null, any plan can be generated
    if (result == null) {
        return NO_PROLOG_PLAN_FOUND;
    }

    Plan plan = new DefaultPlan(); // new empty plan
    Agent clonedAgent = this.agent.deepDataClone();

    Goal goal = _goal.clone();

    // From prolog output to java intermediate state
    PrologResult plOuput = new PrologResult(result);

    for (PIAction action : plOuput.getJavaPlan()) {

        // empty class isn’t treated
        if (action.getName().toLowerCase().equals("empty")) {
            continue;
        }

        // Search the object instance in memory
        WorldObject WOtmp = new WorldObject(action.getName(), action.getIdObject());
        WorldObject WOobject = this.agent.getMemory().getKnowledgeAbout(WOtmp);
        Goal goalClone = null;

        for (Service service : WOobject.getServices()) {

            // service the object memory is the same as in prolog
            if (service.getName().toLowerCase().equals(action.getName().toLowerCase())) {

                // the action is repeated as many times is needed

                // Data is cloned
                // Agent tmpClonedAgent = clonedAgent.deepDataClone(); // Don’t use it
                // Agent tmpClonedAgentMemory =
                // tmpClonedAgent.getMemory().getKnowledgeAboutOwner();
            }
        }
    }
}
if (service.getName().equals("walkTo")) {
    continue;
}

if (service.isUsable(WOobject, clonedAgent, null)) {
    // The service is usable, according to the memory
    // What about its accessibility?
    try {
        PlanElement walkTo = this.prologDFS.handleWalkTo(WOobject, clonedAgent, service.getMaxDistanceForUsage());

        // is the last action of a multiple, walkTo must be at the end
        if (walkTo != null) {
            plan.append(walkTo);
        }

        service.execute(WOobject, clonedAgent, null, ExecutionMode.VIRTUAL);
        goalClone = new Goal(goal.getSuccessCondition(), goal.getPriority());

        plan.append(new PlanElement(service, WOobject, null));
    }
    catch (UnreachableObjectException e) {
        e.printStackTrace();
        continue;
    }

    goal = goalClone;
    // clonedAgent = tmpClonedAgent;
}
// once the service is executed there is no need to keep checking more services
continue;
}

// Return null if no plan found.
if (plan != null && plan.size() == 0) {
    return null;
}
/**
 * Transforms Avatar Goals and Gauges to Prolog
 *
 * @param currentGoal
 *     Current Goal
 * @return String
 */
public String GoalToProlog(Goal currentGoal) {

    String out = this.currentGaugesToProlog() + "
        cash, account,\\
        tiredness, thirsty, hunger, happiness\n";
    out += this.AimGaugeToProlog(currentGoal) + "\n\n";

    return out;
}
Appendix C

Module Planning Implementation

C.1 PrologDFSArgorithm

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.implementation;

import java.util.Collections;
import java.util.Hashtable;

import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.DefaultPlanningModule;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.prolog.PrologWrapper;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.Goal;
import fr.n7.saceca.u3du.model.util.io.TextFileIO;
import fr.n7.saceca.u3du.prolog.Path;
import fr.n7.saceca.u3du.prolog.SinglePl;

import jpl.Atom;
import jpl.Query;
import jpl.Term;
import jpl.Variable;

/**
 * This class is the hook to Prolog implementation
 * @author Albert Jornet Puig
 */
```
public class PrologDFSAlgorithm extends DefaultPlanningModule {

    PrologWrapper prologBinding = null;

    public PrologDFSAlgorithm(Agent agent) {
        super(agent);
        this.prologBinding = new PrologWrapper(agent, this);
    }

    @SuppressWarnings("unchecked")
    @Override
    public Plan buildPlan(Agent agent, Goal goal, int currentDepth, int maxDepth) {
        Plan plan = null;

        // For prolog purposes. To pick different options.
        Collections.shuffle(this.getMemoryKnowledges());

        // Gauges to prolog
        String goalprolog = this.prologBinding.GoalToProlog(this.currentGoal);
        // Memory to prolog
        String memoryprolog = this.prologBinding.MemoryToProlog(this.getMemoryKnowledges());

        // Save files
        TextFileIO.writeFileWithPath(Path.TMP_AVATAR_DIR + this.agent.getId() + Path.GOAL_FILENAME, goalprolog);
        TextFileIO.writeFileWithPath(Path.TMP_AVATAR_DIR + this.agent.getId() + Path.MEMORY_FILENAME, memoryprolog);

        // Exclusion area
        *
        * Prolog Libraries don’t let you have multiple
        * prolog virtual machines at the same time.
        * So, in this implementation we’ve created an
        * exclusion area because of the threads. This
        * way, there isn’t any conflict between them
        * but it suppose a lose of efficiency.
        */
Term ploutput = \texttt{null};
Hashtable<
Term, Term> solution = \texttt{null};

\begin{verbatim}
try {
    SinglePl.getInstance().lock();

    Variable P = \texttt{new Variable("P");}
    String id = Long.toString(this.agent.getId());

    \// execute query
    Query q2 = \texttt{new Query(PrologWrapper.PROLOG_FUNCTION, \}
                new Term[] { \texttt{new Atom(id), P, new jpl.Integer(10), \}
                            \texttt{new jpl.Integer(7) } });

    \// real execution
    solution = q2.oneSolution();
    if (solution != \texttt{null}) {
        ploutput = solution.get("P");
    }

    q2.close();

    plan = \texttt{this.prologBinding.PrologToSacecaPlan(ploutput, \}
             \texttt{this.agent, this.currentGoal});
}
\end{verbatim}

\catch (Interrupted Exception e1) {

    e1.printStackTrace();
}\finally {

    SinglePl.getInstance().unlock();
}

\texttt{// \ldots}

\texttt{return plan;}
\}

\section{C.2 DFSAlgorithm}

\texttt{package fr.n7.saceca.u3du.model.ai.agent.module.planning.implementation;}

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.DefaultPlan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.DefaultPlanningModule;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.UnreachableObjectException;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.Goal;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;
import fr.n7.saceca.u3du.model.ai.service.Service;
import fr.n7.saceca.u3du.model.ai.statement.ExecutionMode;
import fr.n7.saceca.u3du.model.ai.statement.MemoryAwareU3duJexlContext;

public class DFSAlgorithm extends DefaultPlanningModule {

    public DFSAlgorithm(Agent agent) {
        super(agent);
    }

    /**
     * Builds a plan.
     * @param agent the agent
     * @param goal the goal
     * @param currentDepth the current depth
     * @param maxDepth the max depth
     * @return the plan
     */
    @Override
    public Plan buildPlan(Agent agent, Goal goal, int currentDepth, int maxDepth) {
        if (maxDepth < currentDepth) {
            return null;
        }
    }
}
return DefaultPlanningModule.NO_PLAN_FOUND;
}
if (!goal.seemsReached(new MemoryAwareU3duJexlContext(agent))) {
    // According to the memory, the goal seems not yet reached

    // Objects order randomization to avoid having always the same scene
    Collection<WorldObject> objects = agent.getMemory().getKnowledges();
    if (COLLECTION_SHUFFLE_ENABLED) {
        objects = new ArrayList<WorldObject>(objects);
        Collections.shuffle((List<WorldObject>) objects);
    }
}
for (WorldObject object : objects) {
    // Services order randomization
    Collection<Service> services = object.getServices();
    if (COLLECTION_SHUFFLE_ENABLED) {
        services = new ArrayList<Service>(services);
        Collections.shuffle((List<Service>) services);
    }
    for (Service service : services) {

        Agent clonedAgent = agent.deepDataClone();
        Agent clonedAgentMemory = clonedAgent.getMemory().getKnowledgeAboutOwner();
        if (service.getName().equals("walkTo")) {
            continue;
        }
        if (service.isUsable(object, clonedAgentMemory, null)) {
            // The service is usable, according to the memory
            // What about its accessibility ?
            try {
                PlanElement walkTo = this.handleWalkTo(object, clonedAgent,
                    service.getMaxDistanceForUsage());
                service.execute(object, clonedAgentMemory, null,
                    ExecutionMode.VIRTUAL);
                Goal goalClone = new Goal(goal.getSuccessCondition(),
                    goal.getPriority());
                // Computing the next step of the plan
                Plan plan = this.buildPlan(clonedAgent, goalClone,
                    currentDepth + 1, maxDepth);
                if (plan != NO_PLAN_FOUND) {
                    // The computed plan works
                    plan.cons(new PlanElement(service, object, null));
                    if (walkTo != null) {
\begin{verbatim}
    plan.cons( walkTo );
    } return plan;
} catch ( UnreachableObjectException e ) {
    // The object cannot be reached
    continue;
} return NO_PLAN_FOUND;
} else {
    return new DefaultPlan();
}
\end{verbatim}
Appendix D

Module Prolog

D.1 Lock

```java
package fr.n7.saceca.u3du.prolog;

/**
 * Lock between agent threads. Slow performance.
 *
 * @author Albert Jornet Puig
 */
public class Lock {

    /**
     * @uml.property name="isLocked"
     */
    private boolean isLocked = false;

    public synchronized void lock() throws InterruptedException {
        while (this.isLocked) {
            this.wait();
        }
        this.isLocked = true;
    }

    public synchronized void unlock() {
        this.isLocked = false;
        this.notify();
    }
}
```
D.2 Path

```java
package fr.n7.saceca.u3du.prolog;

/**
 * Filenames and dir paths to prolog
 * @author Albert Jornet Puig
 */
public final class Path {

    /**
     * Path to main files: body.pl, util.pl, actions.pl */
    public final static String MAIN_DIR_PATH = "data/ai/prolog/";

    /**
     * Path to main prolog planning algorithm file */
    public final static String MAIN_PROLOG_FILE = "data/ai/prolog/main.pl";

    /**
     * The temporal prolog folder path */
    public final static String TMP_AVATAR_DIR = "data/ai/tmp/avatars/";

    /**
     * The goal filename */
    public final static String GOAL_FILENAME = "/goal.pl";

    /**
     * The avatar memory filename */
    public final static String MEMORY_FILENAME = "/memory.pl";

    /**
     * Name of the function to load common Prolog files */
    public final static String PROLOG_PRELOAD_NAME = "preload";

    /**
     * Name of the function to call plannig algorithm */
    public final static String PROLOG_ALGORITHM_NAME = "getplan";
}
```

D.3 PrologInit

```java
package fr.n7.saceca.u3du.prolog;
```
import jpl.Atom;
import jpl.Query;
import jpl.Term;

/**
 * This class initializes connection to Swipl. It's only
 * required to load once because JPL can only
 * work with one instance.
 *
 * @author Albert Jornet Puig
 */
public class PrologInit {

    private static PrologInit instance = null;

    /**
     * Singleton class
     *
     * @param agent
     * @return instance of PrologInit
     * @throws InterruptedException
     */
    public static synchronized PrologInit getInstance(long agent) throws InterruptedException {
        if (instance == null) {

            instance = new PrologInit();
            init(agent);
        }
        return instance;
    }

    /**
     * Loads common files to execute the algorithm.
     * Must be done once, query closes itself as there
     * is only one option (JPL doc)
     *
     * @param agent
     */
APPENDIX D. MODULE PROLOG

* long
* @throws InterruptedException
* There is a problem connecting to Prolog Virtual Machine.
* /
public static void init(long agent) throws InterruptedException {
    // Creates a connection to main prolog file.
    Query qconnection = new Query("consult", new Term[] {
        new Atom(Path.MAIN_PROLOG_FILE)
    });
    qconnection.hasSolution();

    // Calls preload function
    Query qlibs = new Query(Path.PROLOG_PRELOAD_NAME, \n        new Term[] { new Atom(Path.MAIN_DIR_PATH) });
    qlibs.hasSolution();
}

D.4 SinglePl

package fr.n7.saceca.u3du.prolog;

/**
 * Singleton pattern. <br/>
 * There is only one reference to Prolog VM because
 * more isn't allowed due to it's limitations.
 * *
 * @author Albert Jornet Puig
 * *
 */
public class SinglePl {

    private static SinglePl instance = null;
    /**
     * @uml.property name="lock"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    private Lock lock = new Lock();

    public static synchronized SinglePl getInstance() {
        if (instance == null) {
            return null;
        }
        return instance;
    }
}
instance = new SinglePl();
}
return instance;
}

public void lock() throws InterruptedException {
    this.lock.lock();
}

public void unlock() {
    this.lock.unlock();
}
Appendix E

Module Prolog Output

E.1 PLAction

```java
package fr.n7.saceca.u3du.prolog.output;

/**
 * It represents an action from Prolog to Saceca.
 * This action is simple, which means it takes 1 unit
 * of time to be realized.
 *
 * @author Albert Jornet Puig
 */

public class PLAction {

    /**
     * @uml.property name="name"
     */
    private String name;

    /**
     * @uml.property name="idObject"
     */
    private long idObject;

    public PLAction(String _name, String _object) {
        this.name = _name;

        try {
            // Code...
        }
    }

    // Other methods...
}
```
```java
this.idObject = Long.parseLong(_object.replace("obj", ":"));
}
} catch (Exception e) {
    // if error, is an empty action
    this.idObject = -1;
}

public int getDuration() {
    return 10;
}

@Override
public String toString() {
    return "Action: " + this.name + " with object " +
            this.idObject + ", duration: " + this.getDuration();
}

/**
 * @return
 * @uml.property name="name"
 */
public String getName() {
    return this.name;
}

/**
 * @return
 * @uml.property name="idObject"
 */
public long getIdObject() {
    return this.idObject;
}
}

E.2 PIActionMultiple

package fr.n7.saceca.u3du.prolog.output;

public class PIActionMultiple extends PIAction {
```
E.3. PROLOGRESULT

*/
   @uml.property name="hourStart"
*/
private int hourStart;
/**
   @uml.property name="hourEnd"
*/
private int hourEnd;

public PIActionMultiple(String _name, String _object, int _Start, int _End) {
   super(_name, _object);

   this.hourStart = _Start;
   this.hourEnd = _End;
}

@Override
public int getDuration() {
   return hourEnd - hourStart;
}

}

E.3 PrologResult

package fr.n7.saceca.u3du.prolog.output;

import java.util.ArrayList;
import java.util.List;

import jpl.Term;

/**
   * This class transforms a Prolog result to a java structure.
   *
   * @author Albert Jornet Puig
   *
   */
public class PrologResult {

private List<PIAction> results;
/**
 * @uml.property name="listTerm"
 * @uml.associationEnd multiplicity="(1 1)"
 */

private Term listTerm;

private final static int PROLOG_ACTION_ARGS = 3;
private final static int PROLOG_MULTIPLE_ARGS = 4;

/**
 * It is expected to recieve a Term structure, more precisely a Prolog list
 * @param listTerm
 */
public PrologResult(Term _listTerm) {
    this.results = new ArrayList<PIAction>();
    this.listTerm = _listTerm;
    this.prologToJava();
}

/**
 * It translates the result from Prolog to Saceca
 */
private void prologToJava() {
    Term element = null;

    int length = this.listTerm.listLength();

    // iteration through result prolog list
    for (int i = 0; i < length; i++) {
        element = this.listTerm.arg(1);
        this.results.add(this.build(element));
        this.listTerm = this.listTerm.arg(2); // next item in the list
    }
}
/**
E.3. PROLOGRESULT

* Given a Prolog term it translates to Java action
* @param action
* Term, it contains action plus parameters
* @return Java Action
*/
private PlAction build(Term action) {
    String nameAction = action.name(); // name
    System.out.println(action.arg(1)); // params

    Term[] arguments = action.args();
    PlAction out = null;

    switch (arguments.length) {
        case PrologResult.PROLOG_ACTION_ARGS:
            out = this.buildAction(nameAction, arguments);
            break;
        case PrologResult.PROLOG_MULTIPLE_ARGS:
            out = this.buildMultiple(nameAction, arguments);
            break;
        default:
            System.out.println("PrologResult.java: Should not be reach here. " +\"nameAction");
            break;
    }

    return out;
}

/**
 * Creates class Action with simple duration, only 1 time per action
 * @param name
 * String
 * @param args
 * Term []
 * @return returns Action class
 */
private PlAction buildAction(String name, Term[] args) {

/**
 * Create class ActionMultiple, which means it can take more than one time
 * @param name
 * @param String
 * @param args
 * @param Term []
 * @return ActionMultiple
 */
private PIActionMultiple buildMultiple(String name, Term[] args) {

    int start = args[1].intValue();
    int end = args[2].intValue();
    String object = args[3].name();

    System.out.println(object);

    return new PIActionMultiple(name, object, start, end);
}

@Override
public String toString() {

    String out = "Result:\n";

    for (PIAction act : this.results) {
        out += act.toString() + "\n";
    }

    return out;
}

/**
 * Planning in Java (It's not yet transformed to SACECA)
 * @return List<Action> or null
E.3. PROLOGRESULT

```java
/**
 * public List<PIAction> getJavaPlan() {
 *     return this.results;
 * }
 */
```
Appendix F

Module Anticipation

F.1 Heuristic

```java
package fr.n7.saceca.u3du.model.ai.module.planning.anticipation;

import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.object.properties.UnknownPropertyException;
import fr.n7.saceca.u3du.model.util.Oriented2DPosition;

/**
 * It gives a numeric representation of a plan.
 *
 * @author Albert Jornet Puig
 *
 */
public class Heuristic {

    /**
     * Given a plan, it returns an approximation
     * execution time. Which is the sum of all actions.
     *
     * @param plan Plan
     * @param agent
     * @return float >= 0
     */
```
/

public static float planExecutionTime(Plan plan, Agent agent) {
    int value = 0;
    Oriented2DPosition agentPosition = agent.getPosition();
    boolean walkNeeded = false;

    for (PlanElement planElement : plan) {
        // We need to walk
        if (planElement.getService().getName().equals("walkTo")) {
            walkNeeded = true;
            continue;
        }

        if (walkNeeded) {
            // get avatar walk speed
            double walkSpeed;
            try {
                walkSpeed = agent.getPropertiesContainer().getDouble("c_Human_walk_speed");
            } catch (UnknownPropertyException e) {
                // If there is any problem, set walk speed to 10. Bigger is worse.
                e.printStackTrace();
                walkSpeed = 10.0;
            }

            // Calculate distance
            float length = planElement.getProvider().getPosition().distance(agentPosition);
            value += (1 / walkSpeed / 40) * length;
            agentPosition = planElement.getProvider().getPosition();

            walkNeeded = false;
        }

        // Duration action
        value += planElement.getService().getDuration(planElement.getProvider(), null);
    }

    return value;
}
F.2 LongAnticipation

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.emotion.EmptyEmotionModule;
import fr.n7.saceca.u3du.model.ai.agent.module.perception.DefaultPerceptionModule;
import fr.n7.saceca.u3du.model.ai.agent.module.perception.PerceptionModule;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanningModule;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer.Analyzer;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer.anticipationGoal.Objective;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer.anticipationGoal.ResultWrapper;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.memory.Statistics;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.DefaultReasoningModule;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.ReasoningModule;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;

/**
 * This class anticipates as it's explained
 * @author Albert Jornet Puig
 */
public class LongAnticipation {

/**
 * It let's to this memory to save plans.
 * @uml.property name="memory"
 * @uml.associationEnd multiplicity="(1 1)"
 */
APPENDIX F. MODULE ANTICIPATION

private Memory memory;

/** The max number of plans to remember at initialization */
public final static int ANALYZE_PLANS = 100;

/** The simulation agent class */
// private Simulation simulation = null;

/**
 * The analyzer class
 * @uml.property name="analyzer"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private Analyzer analyzer = null;

private PlanningModule planningModule = null;

/**
 * @uml.property name="reasoningModule"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private ReasoningModule reasoningModule = null;

/**
 * @uml.property name="emotionModule"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private EmptyEmotionModule emotionModule = null;

/**
 * @uml.property name="perceptionModule"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private PerceptionModule perceptionModule = null;

/**
 * The agent to work with
 * @uml.property name="agent"
 * @uml.associationEnd multiplicity="(1 1)"
 */
LongAnticipation

public LongAnticipation(Agent agent) {
    this.agent = agent;
    this.memory = new Memory(this.agent);
    this.emotionModule = new EmptyEmotionModule();
    this.agent.setEmotionModule(this.emotionModule);
    this.perceptionModule = new DefaultPerceptionModule(this.agent);
    this.agent.setPerceptionModule(this.perceptionModule);
    this.planningModule = new PrologDFSAlgorithm(this.agent);
    this.agent.setPlanningModule(this.planningModule);
    this.reasoningModule = new DefaultReasoningModule(this.agent);
    this.agent.setReasoningModule(this.reasoningModule);
    // TODO - get an anticipation goal. This should be done by reason module.
    AnticipationGoal goal = new AnticipationGoal(
        this.agent.getGauges().get(0), Objective.INCREASE, 0.05);
    this.analyzer = new Analyzer(this.memory, this.agent, goal);
    this.initiate();
}

/**
 * WorldObject which helps to improve avatars living.
 * @return WorldObject or null if isn't accomplish.
 */
private WorldObject initiate() {
    // Fills anticipation memory
    this.memory.fillAnticipationMemory();

    // Thinks
    this.execute();

    // Does exist a result? No--> return null
if (this.analyzer.getResults() == null || this.analyzer.getResults().size() == 0) {
    return null;
}

// Returns the object which improves the goal
if (this.analyzer.getBetterResult() != null) {
    return this.analyzer.getBetterResult().specialObject;
} else {
    return null;
}

/**
 * Executes the algorithm to anticipate
 */
public void execute() {
    this.analyzer.start();
}

/**
 * Print to console
 */
@ SuppressWarnings("unused")
private void print() {
    // Print data
    for (ResultWrapper resultWrapper : this.analyzer.getResults()) {
        Statistics stats = new Statistics(resultWrapper.result, this.agent);
        stats.updateData();
        System.out.println(resultWrapper.specialObject.getModelName() + " \
                          " + resultWrapper.specialObject.getId() + ");
        System.out.println("Knowledge avatar size: " + \n                          resultWrapper.agentKnowledge.size());
        System.out.println("Satisfy? " + this.analyzer.isSatisfied(resultWrapper));
        System.out.println("--------------------\n");
    }
    System.out.println("--------------------");
    System.out.println("_________ Original __") ;
F.2. LONGANTICIPATION

```java
this.memory.getStatisticsModule().updateData();
System.out.println(this.memory.getStatisticsModule().toString());
System.out.println("\n\n-------------------------------");
System.out.println("BEST: " + this.analyzer.getBetterResult().toString());
}

/**
 * Returns the best result found.
 * @return ResultWrapper or null if no result is found
 */
public ResultWrapper getResult() {
    return this.analyzer.getBetterResult();
}

/**
 * Get the anticipation memory
 * @return Memory
 * @uml.property name="memory"
 */
public Memory getMemory() {
    return this.memory;
}

/**
 * Get the analyzer.
 * @return Analyzer
 * @uml.property name="analyzer"
 */
public Analyzer getAnalyzer() {
    return this.analyzer;
}
```
Appendix G

Module Anticipation Analyzer

G.1 AbstractAlgorithm

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer;

import java.util.Iterator;
import java.util.List;
import java.util.Set;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;

/**
 * Base class to create an algorithm to anticipate
 * @author Albert Jornet Puig
 */
public abstract class AbstractAlgorithm implements Iterator<List<WorldObject>> {

    /**
     * @uml.property name="memory"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    protected Memory memory;

    /**
    */
```
protected Agent agent;

/**
 * Instantiate class
 * @param memory Memory to deal
 * @param agent Agent to deal
 */
public AbstractAlgorithm(Memory memory, Agent agent) {
    this.memory = memory;
    this.agent = agent;
}

/**
 * The number of different options found.
 * @return int
 */
public abstract int getNumberOfFoundOptions();

class Analyzer
{
    package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer;

    import java.util.ArrayList;
    import java.util.List;
}
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.Gauge;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.LongAnticipation;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer.AnticipationGoal.Objective;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.simulation.Simulation;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;

/**
 * Implements algorithm to find new ways to increase standards living of an avatar.
 * @author Albert Jornet Puig
 */
public class Analyzer {

    /**
     * Anticipation memory avatar
     * @uml.property name="memory"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    private Memory memory = null;

    /**
     * Agent
     * @uml.property name="agent"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    private Agent agent = null;

    /**
     * Goal to satisfy
     * @uml.property name="goal"
     * @uml.associationEnd multiplicity="(1 1)"
     */
APPENDIX G. MODULE ANTICIPATION ANALYZER

private AnticipationGoal goal = null;

/**
 * Algorithm to analyze data
 * @uml.property name="algorithm"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private AbstractAlgorithm algorithm = null;

/**
 * Simulation class
 * @uml.property name="simulation"
 * @uml.associationEnd
 */
private Simulation simulation = null;

private List<ResultWrapper> results = null;

public Analyzer(Memory memory, Agent agent, AnticipationGoal goal) {
    this.memory = memory;
    this.agent = agent;
    this.goal = goal;
    this.algorithm = new DefaultAlgorithm(this.memory, this.agent, this.goal);
    this.results = new ArrayList<ResultWrapper>();
}

/**
 * Starts the analysis.
 */
public void start() {
    // Temporal Agent memory
    List<WorldObject> simulatedAgentMemory = null;

    // Current Agent memory
    List<WorldObject> originalAgentMemory = this.agent.getMemory().getKnowledges();
    // Agent clonedAgent = this.agent.deepDataClone();
    System.out.println("Analyzing the " + " different options found");
    System.out.println(this.algorithm.getNumberOptions() + " different options found");
    System.out.println(this.algorithm.getModifyingActions().toString());
for (int i = 0; i < this.algorithm.getNumberOfFoundOptions() &&
    this.algorithm.hasNext(); i++) {

    // Restore original memory for agent
    this.agent.getPlanningModule().setMemoryKnowledges(originalAgentMemory);

    // Instantiate new anticipation memory
    Memory memorySimulation = new Memory(this.agent);

    // Changes agent memory to erase those objects
    // that his services can modify agent gauge
    simulatedAgentMemory = this.algorithm.next();

    System.out.println(simulatedAgentMemory.toString());

    // Set new avatars memory
    this.agent.getPlanningModule().setMemoryKnowledges(simulatedAgentMemory);

    // Simulates with new memory
    this.simulation = new Simulation(this.agent);

    // Fills new memory with plans
    for (int j = 0; j < LongAnticipation.ANALYZE_PLANS &&
        this.simulation.hasNext(); j++)
        Plan plan = this.simulation.next();

        try {
            memorySimulation.remember(plan, -1);
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    // Adds result
    ResultWrapper resultWrapper = new ResultWrapper(this.algorithm.getCurrent()\
        , this.goal, memorySimulation, simulatedAgentMemory);
    this.results.add(resultWrapper);

    // Displays memory
    // memorySimulation.paint();
/**
 * Gets the results of the execution
 * @return List<ResultWrapper>
 */
public List<ResultWrapper> getResults() {
    return this.results;
}

/**
 * The result satisfy the goal condition
 * @param resultWrapper ResultWrapper
 * @return boolean
 */
public boolean isSatisfied(ResultWrapper resultWrapper) {
    resultWrapper.result.getStatisticsModule().updateData();
    this.memory.getStatisticsModule().updateData();

    // Gauge units gained / total time = index
    float newValue = resultWrapper.result.getStatisticsModule().getIndex(this.goal.getGauge());
    float originalValue = this.memory.getStatisticsModule().getIndex(this.goal.getGauge());

    boolean increase = this.goal.getAim() == Objective.INCREASE;

    // Checks if the goal is satisfied
    if (originalValue > newValue && !increase) {
        return true;
    } else if (originalValue > newValue && increase) {
        return false;
    } else if (originalValue < newValue && !increase) {
        return false;
    } else if (originalValue < newValue && increase) {
        return true;
    }
return false;
}

/**
 * Return the result with the best index
 * @return ResultWrapper or null if there isn't a valid result.
 */
public ResultWrapper getBetterResult() {

float best = -1;
ResultWrapper bestWrap = null;
Gauge goalGauge = this.goal.getGauge();

for (ResultWrapper wrap : this.results) {
    // Goal satisfied
    if (this.isSatisfied(wrap)) {
        float index = wrap.result.getStatisticsModule().getIndex(goalGauge);

        // Compares with the last found the check if it's bigger/lower
        if ((index > best && this.goal.getAim() == Objective.INCREASE) || (index < best && this.goal.getAim() == Objective.DECREASE)) {
            best = wrap.result.getStatisticsModule().getIndex(goalGauge);
            bestWrap = wrap;
        }
    }
}

return bestWrap;
}

/**
 * The algorithm used.
 * @return AbstractAlgorithm
 * @uml.property name="algorithm"
 */
public AbstractAlgorithm getAlgorithm() {
    return this.algorithm;
}
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer;

import fr.n7.saceca.u3du.model.ai.agent.Gauge;

/**
 * Defines an anticipation Goal.
 *
 * @author Albert Jornet Puig
 */

public class AnticipationGoal {

/**
 * Gauge to satisfy
 * @uml.property name="gauge"
 * @uml.associationEnd multiplicity="(1 1)"
 */

private Gauge gauge = null;

/**
 * @author Sylvain Cambon & Jerome Dalbert &
 * Anthony Foulfoin & Johann Legaye & Aurelien Chabot
 */

public enum Objective {

/**
 * @uml.property name="INCREASE"
 * @uml.associationEnd
 */

INCREASE,  /**
 * @uml.property name="DECREASE"
 * @uml.associationEnd
 */

DECREASE
}

/**
 * What to do
 * @uml.property name="aim"
 * @uml.associationEnd multiplicity="(1 1)"
 */


```java
private Objective aim;

/**
 * Range to consider when a goal is achieved
 * @uml.property name="alpha"
 */
private double alpha;

/**
 * Instantiate AnticipationGoal
 *
 * @param gauge Gauge
 * @param aim Objective Increase/decrease the gauge
 * @param alpha Factor to consider the goal succeeds
 */
public AnticipationGoal(Gauge gauge, Objective aim, double alpha) {
    this.gauge = gauge;
    this.aim = aim;
    this.alpha = alpha;
}

/**
 * Returns Goal aim
 * @return Objective
 * @uml.property name="aim"
 */
public Objective getAim() {
    return this.aim;
}

/**
 * @return
 * @uml.property name="gauge"
 */
public Gauge getGauge() {
    return this.gauge;
}
```
G.4 DefaultAlgorithm

package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.Gauge;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;

/**
 * Algorithm implementation.
 * @author Albert Jornet Puig
 */
public class DefaultAlgorithm extends AbstractAlgorithm {

/**
G.4. DEFAULTALGORITHM

* @uml.property name="modifyingActions"
*/
private Set<WorldObject> modifyingActions = null;

private Iterator<WorldObject> iterator = null;

/**
 * @uml.property name="goal"
 * @uml.associationEnd multiplicity="(1 1)"
*/
private AnticipationGoal goal = null;

/**
 * The object to analyze
 * @uml.property name="currentObject"
 * @uml.associationEnd multiplicity="(1 1)"
*/
private WorldObject currentObject = null;

public DefaultAlgorithm(Memory memory, Agent agent, AnticipationGoal goal) {
    super(memory, agent);

    this.goal = goal;
    this.modifyingActions = this.getActionsModifyGauge(this.goal.getGauge());

    if (this.modifyingActions != null && this.modifyingActions.size() > 0) {
        this.currentObject = (WorldObject) this.modifyingActions.toArray()[0];
    }

    this.iterator = this.modifyingActions.iterator();
}

/**
 * Returns all those actions in avatar memory which directly effect the gauge param.
 *
 * @param gauge
 * @return Set<WorldObject>
 */
private Set<WorldObject> getActionsModifyGauge(Gauge gauge) {
Set<WorldObject> output = new LinkedHashSet<WorldObject>();

// Iterate all over avatars memories objects
for (WorldObject worldObject : this.agent.getMemory().getKnowledges()) {

    // Add to output variable if its effects change the gauge
    if (worldObject.isServicesDirectlyEffectGauge(gauge)) {
        output.add(worldObject);
        System.out.println(worldObject.getModelName());
    }
}

return output;

/**
 * Returns the avatar memory excluding those Objects that modify the indicated gauges.
 * @param excludingEffects List<Gauge> Gauges to avoid
 * @return List<WorldObject> Returns filtered list
 */
private List<WorldObject> getFilteredKnowledges(Set<Gauge> excludingEffects) {
    List<WorldObject> output = new ArrayList<WorldObject>();
    Set<Gauge> modifiedGauges = null;
    List<Gauge> excludingEffectsList = null;

    for (WorldObject object : this.agent.getMemory().getKnowledges()) {
        // Modified gauges of the object
        excludingEffectsList = new ArrayList<Gauge>(excludingEffects);
        modifiedGauges = object.getModifiedGauges(excludingEffectsList);

        // To check if modified gauges are relative to this object
        if (!this.containsGauge(excludingEffectsList, new ArrayList<Gauge>(modifiedGauges))) {
            output.add(object);
        }
    }

    return output;
G.4. DEFAULTALGORITHM

```java
@Override
public boolean hasNext() {
    // No possible to find any solution
    if (this.currentObject == null) {
        return false;
    }
    return this.iterator.hasNext();
}

@Override
public List<WorldObject> next() {
    this.currentObject = this.iterator.next();
    System.out.println("Object to analyze: " + this.currentObject.toString());
    List<Gauge> listGauge = new ArrayList<>();
    listGauge.add(this.goal.getGauge());
    // Which Gauges are modified by this provider. Ex: MacDo -> hunger gauge
    Set<Gauge> modifiedGauges = this.currentObject.getModifiedGauges(listGauge);
    // Delete all other actions which change the same gauge. Ex: exclude KFC, Bistrot, ...
    List<WorldObject> filteredKnowledge = this.getFilteredKnowledges(modifiedGauges);
    // Add the object we want to force to simulate as a default option
    filteredKnowledge.add(this.currentObject);
    return filteredKnowledge;
}

@Override
public void remove() {
}

@Override
public int getNumberOfFoundOptions() {
    return this.modifyingActions.size();
}
```
@Override
public Set<WorldObject> getModifyingActions() {
    return this.modifyingActions;
}

/**
 * toCompare gauge is included in the source list
 * @param source
 * List<Gauge>
 * @param toCompare
 * List<Gauge>
 * @return Boolean true is included, otherwise false
 */
private boolean containsGauge(List<Gauge> source, List<Gauge> toCompare) {

    for (Gauge gaugeSource : source) {
        for (Gauge gaugeCompare : toCompare) {
            if (gaugeSource.getName().equals(gaugeCompare.getName())) {
                return true;
            }
        }
    }

    return false;
}

/**
 * Retrieves the current object to analyze
 * @return WorldObject
 */
@Override
public WorldObject getCurrent() {
    return this.currentObject;
}

G.5 ResultWrapper
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.analyzer;

import java.util.List;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;

/**
 * This class wraps a result of an anticipation simulation
 *
 * @author Albert Jornet Puig
 *
 */
public class ResultWrapper {

/**
 * Special object to analyze
 * @uml.property name="specialObject"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public WorldObject specialObject;

/**
 * Gauge to analyze
 * @uml.property name="goal"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public AnticipationGoal goal;

/**
 * The anticipation memory
 * @uml.property name="result"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public Memory result;

/**
 * The agent knowledge used to generate result variable
 * @uml.property name="agentKnowledge"
 */
public List<WorldObject> agentKnowledge;
/*
 * Instantiate class
 *
 * @param object
 * WorldObject Special object to analyze
 * @param anticipationGoal
 * AnticipationGoal
 * @param memoryResult
 * Memory which contains the simulation
 */

public ResultWrapper(WorldObject object,
                      AnticipationGoal anticipationGoal,
                      Memory memoryResult,
                      List<WorldObject> agentKnowledge) {
    this.specialObject = object;
    this.goal = anticipationGoal;
    this.result = memoryResult;
    this.agentKnowledge = agentKnowledge;
}

@Override
public String toString() {

    return this.specialObject.getModelName() + " " +
    this.goal.toString() + " " + this.specialObject.getModelName() + " ( " +
    this.specialObject.getId() + ")";
}
}
Appendix H

Module Anticipation Drawer

H.1 AbstractMemoryLayout

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer;

/**
 * Abstract class to implement new layouts for memory representation.
 * @author Albert Jornet Puig
 */
public abstract class AbstractMemoryLayout {
    /**
     * Window width size
     * @uml.property name="width"
     */
    protected int width;

    /**
     * Window height size
     * @uml.property name="height"
     */
    protected int height;

    /**
     * Number of elements to print
     * @uml.property name="elements"
     */
}```
protected int elements;

/**
* All calculated positions
* @uml.property name="nodes"
* @uml.associationEnd multiplicity="(0 −1)"
*/
protected Position[] nodes;

/**
* The next position to show
* @uml.property name="currentPosition"
*/
protected int currentPosition;

/**
* Get a new position for an element
* @return Position
*/
public abstract Position nextPosition();

H.2 CircleMemoryLayout

package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer;

import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer.AbstractMemoryLayout;

/**
* Generate N coordinates for a circle layout
* @author Albert Jornet Puig
*/
public class CircleMemoryLayout extends AbstractMemoryLayout {

/**
* Instantiate the class
public CircleMemoryLayout(int width, int height, int elements) {
    this.width = width;
    this.height = height;
    this.elements = elements;

    this.nodes = new Position[elements];

    this.calculatePositions();
}

/**
 * Fills nodes array with a position class for each element.
 */
private void calculatePositions() {
    // Calculates circle diameter
    int radius = Math.min(this.width, this.height) / 2 - 40;

    // Calculates the center
    Position center = new Position();
    center.X = this.width / 2;
    center.Y = this.height / 2;

    // Calculate every point of the circle
    double circumference = 2 * Math.PI;
    double inc = circumference / this.elements;

    /*
     * Circle parametric form:
     *
     * x = a + r * cos(t), y = b + r * sin(t),
     * where: a,b is the center and t is a value between [0, 2*pi] r is the radius
     */
}
H.3 Painter

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer;
import java.util.HashMap;
import javax.swing.JFrame;
import org.jgrapht.WeightedGraph;
import org.jgrapht.graph.DefaultWeightedEdge;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Node;

/**
 * This class render anticipation memory into an image
 * @author Albert Jornet Puig
 */
```
public class Painter {

/**
 * Represents the memory, nodes and edges
 * @uml.property name="weightedGraph"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private WeightedGraph<Node, DefaultWeightedEdge> weightedGraph;

/**
 * Sets the Window size
 * @uml.property name="frameWidth"
 */
private int frameWidth;

/**
 * Sets the Window size
 * @uml.property name="frameHeight"
 */
private int frameHeight;

/**
 * Layout of the elements to show
 * @uml.property name="memoryLayout"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private AbstractMemoryLayout memoryLayout;

/**
 * The avatar which belongs this memory
 * @uml.property name="agent"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private Agent agent;

public Painter(WeightedGraph<Node, DefaultWeightedEdge> \$
    weightedGraph, int frameWidth, int frameHeight,
    AbstractMemoryLayout memoryLayout, Agent agent) {
    this.weightedGraph = weightedGraph;
    this.frameHeight = frameHeight;
}
```java
this.frameWidth = frameWidth;
this.memoryLayout = memoryLayout;
this.agent = agent;
}

/**
* Retrieves mxGraph instance if it already exists
* @param graph
* @param hashMap
* @param node
* @param parent
* @return */
private Object getGraphXNode(mxGraph graph, HashMap<Integer, Object> hashMap, Node node, Object parent) {
    Object v1 = null;

    // To check if instance exists
    if (!hashMap.containsKey(node.hashCode())) {
        // Retrieves a position
        Position vertexPosition = this.memoryLayout.nextPosition();

        // Set node identification by hashcode
        String node_id = Integer.toString(node.hashCode());

        v1 = graph.insertVertex(parent, node_id,
            node.toString(), vertexPosition.X, vertexPosition.Y, 80, 30);

        // Insert new vertex in cache
        hashMap.put(node.hashCode(), v1);
    } else {
        // To retrieve from cache (hashmap)
        v1 = hashMap.get(node.hashCode());
    }

    return v1;
```
/**
 * To visualize graph
 */
public void getRepresentation() {

/** Sets Layout */
// this.circleMemoryLayout = new CircleMemoryLayout(Memory.FRAME_WIDTH, \ 
    Memory.FRAME_HEIGHT, 
// this.weightedGraph
// .vertexSet().size());
mxGraph graph = new mxGraph();
Object parent = graph.getDefaultParent();

// Cache created vertexs
HashMap<Integer, Object> hashMap = new HashMap<Integer, Object>();
Object v1, v2;

// Adds cells to the model in a single step
graph.getModel().beginUpdate();
try {
    for (DefaultWeightedEdge edge : this.weightedGraph.edgeSet()) {
        // Get nodes from edge
        Node start = this.weightedGraph.getEdgeSource(edge);
        Node end = this.weightedGraph.getEdgeTarget(edge);

        // If already created, get the instance. Else create a new one.
        v1 = this.getGraphXNode(graph, hashMap, start, parent);
        v2 = this.getGraphXNode(graph, hashMap, end, parent);

        graph.insertEdge(parent, null, this.weightedGraph.getEdgeWeight(edge), v1, v2);
    }
}
finally {
    // Updates the display
    graph.getModel().endUpdate();
}
mxGraphComponent graphComponent = new mxGraphComponent(graph);
new mxParallelEdgeLayout(graph).execute(graph.getDefaultParent());

JFrame frame = new JFrame();
frame.add(graphComponent);
frame.pack();
frame.setVisible(true);
frame.setSize(this.frameWidth, this.frameHeight);
frame.setTitle("Agent " + this.agent.getModelName() + " (" + this.agent.getId() + ")");
}

H.4 Position

package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer;

/**
 * Represents a position of 2 coords
 * @author Albert Jornet Puig
 */
public class Position {
    /**
     * @uml.property name="x"
     */
    public int X;
    /**
     * @uml.property name="y"
     */
    public int Y;

    @Override
    public String toString() {
        return this.X + " " + this.Y;
    }
}
Appendix I

Module Anticipation Memory

I.1 Memory

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory;

import java.util.Arrays;
import java.util.Vector;

import org.jgrapht.WeightedGraph;
import org.jgrapht.graph.DefaultDirectedWeightedGraph;
import org.jgrapht.graph.DefaultWeightedEdge;

import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.Heuristic;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.LongAnticipation;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer.CircleMemoryLayout;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.drawer.Painter;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.statistics.Statistics;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.simulation;

public class Memory {
    
    171
```
/**
   * Represents the memory, nodes and edges
   * @uml.property name="weightedGraph"
   * @uml.associationEnd multiplicity="(1 1)"
   */
   private WeightedGraph<Node, DefaultWeightedEdge> weightedGraph;

/**
   * To have all plans
   * @uml.property name="rememberedPlans"
   * @uml.associationEnd multiplicity="(1 1)"
   */
   private VectorGeneric RememberedPlans = null;

/**
   * Initial and ending nodes
   * @uml.property name="initElement"
   * @uml.associationEnd multiplicity="(1 1)"
   */
   private Node initElement;

/**
   * Initial and ending nodes
   * @uml.property name="endElement"
   * @uml.associationEnd multiplicity="(1 1)"
   */
   private Node endElement;

/**
   * Agent
   * @uml.property name="agent"
   * @uml.associationEnd multiplicity="(1 1)"
   */
   private Agent agent;

/**
   * Graphically represents the memory
   * @uml.property name="painter"
   * @uml.associationEnd
   */


private Painter painter;

/**
 * The simulation agent class
 * @uml.property name="simulation"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private Simulation simulation = null;

/**
 * The statistics class
 * @uml.property name="stats"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private Statistics stats = null;

/** Window width */
private static final int WINDOW_WIDTH = 640;

/** Window height */
private static final int WINDOW_HEIGHT = 480;

public Memory(Agent agent) {
    this.agent = agent;

    this.initElement = new Node(null, 0, true, false);
    this.endElement = new Node(null, 0, false, true);

    this.weightedGraph = new DefaultDirectedWeightedGraph<Node, DefaultWeightedEdge>(DefaultWeightedEdge.class);
    this.weightedGraph.addVertex(this.initElement);
    this.weightedGraph.addVertex(this.endElement);

    this.RememberedPlans = new VectorGeneric();
    this.stats = new Statistics(this, this.agent);
    this.simulation = new Simulation(this.agent);
}

/**
 * To add a plan to memory

APPENDIX I. MODULE ANTICIPATION MEMORY

```java
public void remember(Plan plan, int time) throws Exception {
    if (plan == null) {
        throw new Exception("Plan is null");
    }

    Node previousNode = this.initElement;
    int occurrences = 0;

    for (PlanElement planElement : plan) {
        // Skip walkTo action.
        if (planElement.getService().getName().equals("walkTo")) {
            continue;
        }

        // It Checks If the previous planElement and the new one have the same service
        if (previousNode.getPlanElement() != null &&
            planElement != null &&
            previousNode.getPlanElement().equalsGeneric(planElement)) {
            occurrences++;
        } else {
            occurrences = 0;
        }

        Node node = new Node(planElement, occurrences, false, false);
        this.weightedGraph.addVertex(node);

        DefaultWeightedEdge new_edge = this.weightedGraph.
            addEdge(previousNode, node);

        // null: it means edge already exists, It's only need to update edge counter.
        if (new_edge == null) {
            this.increaseEdgeWeight(previousNode, node);
        }
    }
}
```
previousNode = node;
}

// Create edge from last action to the end action
DefaultWeightedEdge new_edge = this.weightedGraph.addEdge(previousNode, this.endElement);
if (new_edge == null) {
    this.increaseEdgeWeight(previousNode, this.endElement);
}

// Set new value or update in memory
int value = this.RememberedPlans.containsKey(plan) ? this.RememberedPlans.get(plan) + 1 : 1;
this.RememberedPlans.put(plan, value);

/**
 * Updates edge weight to +1 if it already exists or initialize to 1
 * @param edge
 * DefaultWeightedEdge, edge to update or initialize. If null it means it already exists.
 */
private void increaseEdgeWeight(Node init, Node end) {
    // Edge already exists
    DefaultWeightedEdge edge = this.weightedGraph.getEdge(init, end);
    double currentWeight = this.weightedGraph.getEdgeWeight(edge);
    this.weightedGraph.setEdgeWeight(edge, currentWeight + 1);
}

public void forget(Plan plan) {
}

@Override
public String toString() {
    String output = "", tmp = "";
output = "Memory:\n";
double frequency;
int totalPlans = this.getTotalNumberOfPlans();

for (Plan plan : this.getRememberedPlans()) {
    frequency = (double) this.getFrequency(plan) / (double)
                this.getTotalNumberOfPlans();

    tmp = "";
    // Print each plan element with service action
    for (PlanElement planElement : plan) {
        tmp += planElement.getService().getName() + " ( " + planElement.getProvider().getModelName() + " " + planElement.getProvider().getId() + " ) -> ";
    }
    tmp += Heuristic.planExecutionTime(plan, this.agent) + " minutes";

    // Print frequency for each plan
    output += frequency + " ( " + this.getFrequency(plan) + " / " + totalPlans + " ) " + " : " + tmp + "\n";
}

return output;
}

/**
 * Fills anticipation memory with the simulation plans found.
 */
public void initiate() {
    /** TODO – fill memory with experiences. */

    /*
    * Simulation to iterate over an avatar
    * to get next goal Get a plan Add to anticipation
    * memory
    */
}

/**
 * The number of plans in the memory
** © return int >= 0 */

public int getTotalNumberOfPlans() {
    int sum = 0;
    for (DefaultWeightedEdge edge : this.weightedGraph.edgesOf(this.initElement)) {
        sum += this.weightedGraph.getEdgeWeight(edge);
    }
    return sum;
}

/**
 * Given a plan it returns the number of executions
 * @param plan Plan
 * @return int number of times, but –1 if plan contains no element
 */
public int getFrequency(Plan plan) {
    // Memory doesn’t contain this plan
    if (this.RememberedPlans.get(plan) == null) {
        return -1;
    }

    return this.RememberedPlans.get(plan);
}

/**
 * Retrieves remembered plans in memory
 * @return Vector<Plan>
 */
@SuppressWarnings({"rawtypes", "unchecked"})
public Vector<Plan> getRememberedPlans() {
    return new Vector(Arrays.asList(this.RememberedPlans.keySet().toArray()));
}

/**
Get statistics module
@return Statistics

public Statistics getStatisticsModule() {
    return this.stats;
}

Get simulation module
@return Simulation

public Simulation getSimulationModule() {
    return this.simulation;
}

Fill anticipation memory with N plans

public void fillAnticipationMemory() {

    // If possible adds a maximum of 40 plans to memory
    for (int i = 0; i < LongAnticipation.ANALYZE_PLANS &&
        this.simulation.hasNext(); i++) {
        Plan plan = this.simulation.next();

        try {
            this.remember(plan, -1);
        }
        catch (Exception e) {
            System.out.println(this.agent.getId());
            e.printStackTrace();
        }
    }
}

Draw the graph
public void paint() {

  // Nothing to show
  if (this.weightedGraph == null ||
      this.weightedGraph.vertexSet().size() == 0) {
    return;
  }

  // Circle layout
  CircleMemoryLayout circleLayout = new CircleMemoryLayout(
      Memory.WINDOW_WIDTH, Memory.WINDOW_HEIGHT,
      this.weightedGraph.vertexSet().size());
  // Initialize painter
  this.painter = new Painter(this.weightedGraph,
      Memory.WINDOW_WIDTH, Memory.WINDOW_HEIGHT, circleLayout,
      this.agent);
  // Show
  this.painter.getRepresentation();
}

1.2 Node

package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory;

import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;

/**
 * Wrapper to PlanElement class
 * @author Albert Jornet Puig
 */
public class Node {

  /* Defined as initial node */
  @uml.property name="initNode"
  private boolean initNode;
}
APPENDIX I. MODULE ANTICIPATION MEMORY

/* Defined as ending node */
/**
 * @uml.property name="endNode"
 */
private boolean endNode;

/* Plan Element to reference */
/**
 * @uml.property name="planElement"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private PlanElement planElement;

/* It tells which occurrence in the same plan is. To treat repeated actions. */
/**
 * @uml.property name="occurrence"
 */
private int occurrence;

/**
 * Constructor
 *
 * @param planElement
 * @param initNode
 * boolean, defines start node
 * @param endNode
 * boolean, defines end node
 */
public Node(PlanElement planElement, int occurrence, \
    boolean initNode, boolean endNode) {
    this.planElement = planElement;
    this.initNode = initNode;
    this.endNode = endNode;
    this.occurrence = occurrence;
}

/**
 * @return
 * @uml.property name="initNode"
 */
1.2. NODE

```java
public boolean isInitNode() {
    return this.initNode;
}
/**
 * @return
 * @uml.property name="endNode"
 */
public boolean isEndNode() {
    return this.endNode;
}
/**
 * @return
 * @uml.property name="planElement"
 */
public PlanElement getPlanElement() {
    return this.planElement;
}
@override
public String toString() {
    if (this.initNode) {
        return "Node_initNode."
    } else if (this.endNode) {
        return "Node_endNode."
    } else {
        return "Node." + this.planElement.getService().getName() +\".to. + this.planElement.getProvider().getId\" +\"." + this.occurrence;
    }
}
/**
 * Retrieves the occurrence of the node
 * @return int >= 0
 * @uml.property name="occurrence"
 */
public int getOccurrence() {
    return this.occurrence;
}
```
@Override
public boolean equals(Object other) {
    if (other == null) {
        return false;
    }
    if (other == this) {
        return true;
    }
    if (!(other instanceof Node)) {
        return false;
    }
    Node otherNode = (Node) other;

    if (this.initNode != otherNode.isInitNode() || this.endNode != otherNode.isEndNode()) {
        return false;
    }

    // To compare with the generic planElement (Only same Service)
    if (!this.planElement.equalsGeneric(otherNode.getPlanElement())) {
        return false;
    }

    // Different occurrence
    if (this.occurrence != otherNode.occurrence) {
        return false;
    }

    return true;
}

@Override
public int hashCode() {

    // unique code for init and end node
    int initValue = !this.initNode ? 0 : 10000;
    int endValue = !this.endNode ? 0 : 100000;

    // if no plan element, return the sum of both values
    if (this.planElement == null) {
        return initValue + endValue;
    }

    // if plan element is not null, create a code based on it
    // ...
I.3. VECTORGENERIC

```java
return initValue + endValue;
}

int value = this.planElement.hashCode();

// We want to identify only the action. No matter to which object give service.
// long serviceProviderHash = this.getPlanElement().getProvider().getId();

return value + initValue + endValue + this.occurence;
}
}

1.3 VectorGeneric

package fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory;

import java.util.HashMap;

import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;

/**
 * Extends IdentityHashMap class with containsGeneric.
 *
 * @author Albert Jornet Puig
 */
public class VectorGeneric extends HashMap<Plan, Integer> {

private static final long serialVersionUID = 1L;

/**
 * Looks for an element inside the vector
 * with equalsGeneric action (Only compare services
 * between elements).
 *
 * @param toCompare
 * @return true if toCompare plan is in VectorGeneric
 */
public boolean containsGeneric(Plan toCompare) {

```
for (Plan vectorPlan : this.keySet()) {
    if (vectorPlan.equalsGeneric(toCompare)) {
        return true;
    }
}
return false;
Appendix J

Module Anticipation Memory Statistics

J.1 Statistics

```java
package fr.n7.saceca.u3du.model.ai.agent.module.planning.\\
    anticipation.memory.statistics;

import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.util.HashMap;
import java.util.Vector;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.Gauge;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.Heuristic;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.anticipation.memory.Memory;
import fr.n7.saceca.u3du.model.ai.object.properties.UnknownPropertyException;
import fr.n7.saceca.u3du.model.ai.statement.Effect;

/**
 * This class contains statistics extracted from memory.
 * This way it's possible to quantify the quality of all plans.
 * @author Albert Jornet Puig
 */
```
public class Statistics {

  /** Describes average time */
  private final static String STATS_AVERAGE_TIME = "average.time";

  /** Describes total time */
  private final static String STATS>Total_TIME = "total.time";

  /** Money gained, in the future should be automatized */
  private final static String STATS_MONEY_SPENT = "money.spent";
  private final static String STATS_MONEY_GAINED = "money.gained";

  /**
   * Analyzed properties.
   * @uml.property name="properties"
   */
  private HashMap<String, Integer> properties;

  /** How many times a action has been executed */
  private HashMap<String, Integer> actionExecuted;

  /**
   * @uml.property name="memory"
   * @uml.associationEnd
   */
  private Memory memory = null;

  /**
   * @uml.property name="agent"
   * @uml.associationEnd
   */
  private Agent agent = null;

  public Statistics(Memory memory, Agent agent) {
    this.memory = memory;
    this.agent = agent;

    // Init hashmaps
    this.properties = new HashMap<String, Integer>();
    this.actionExecuted = new HashMap<String, Integer>();

    // Init for gauge properties
for (Gauge gauge : this.agent.getGauges()) {
    this.properties.put(gauge.getName(), 0);
    this.actionExecuted.put(gauge.getName(), 0);
}

// Init for static fields
for (String stat : this.getStaticProperties()) {
    this.properties.put(stat, 0);
    this.actionExecuted.put(stat, 0);
}

/**
 * Discover all static variables. Using reflection technique.
 */
private Vector<String> getStaticProperties() {
    Vector<String> output = new Vector<String>();
    Field[] fields = Statistics.class.getDeclaredFields();
    String value = null;

    // Iterate all over found fields
    for (Field f : fields) {
        if (Modifier.isStatic(f.getModifiers())) {
            try {
                // Get the name of the field
                value = (String) f.get(null);
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
            output.add(value);
        }
    }
    return output;
}
return output;
}

/**
 * Updates all statistics attributes.
 */
public void updateData() {
    this.calculateTotalTime();
    this.calculateAverageTime();

    // TODO - delete those lines because money isn't a direct gauge. It should be dynamic.
    this.calculateMoneyGained();
    this.calculateMoneySpent();

    for (Gauge gauge : this.agent.getGauges()) {
        this.calculateGainedGaugeUnitsOf(gauge.getName());
    }
}

/**
 * The number of units gained in all plans.
 * @param gaugeName String, it corresponds to a gauge name
 * @return int
 */
public int getGainedGaugeOf(String gaugeName) {
    return this.properties.get(gaugeName);
}

/**
 * Calculates the total execution time of all plans.
 */
private void calculateTotalTime() {
    float total = 0;

    for (Plan plan : this.memory.getRememberedPlans()) {
        // Execution plan time
        total += Heuristic.planExecutionTime(plan, this.agent);
    }
private void calculateAverageTime() {

    float frequency;
    float minutes, average = 0, total = this.memory.getTotalNumberOfPlans();

    for (Plan plan : this.memory.getRememberedPlans()) {
        // Plan frequency
        frequency = this.memory.getFrequency(plan) / total;

        // Execution plan time
        minutes = Heuristic.planExecutionTime(plan, this.agent);

        // Average
        average += frequency * minutes;
    }

    // this.average_time = average;
    this.properties.put(Statistics.STATS_AVERAGE_TIME, (int) average);
}

private void calculateMoneyGained() {

    int repetitions = 0;
    int wage = -1;

    try {
        // Get avatar wage
        wage = this.agent.getPropertiesContainer().getInt("c.Worker.salary");
    } catch (UnknownPropertyException e) {
        e.printStackTrace();
    }
wage = 0;
}

// Iterate all over plans
for (Plan plan : this.memory.getRememberedPlans()) {
    // Accumulate work
    repetitions += plan.getNumberOfActions("work") *\
        this.memory.getFrequency(plan);
}

// Number of worked times per wage
// this.money.gained = wage * repetitions;
int gained = wage * repetitions;
this.properties.put(Statistics.STATS_MONEY_GAINED, gained);
}

/**
 * Calculates the value for a given Gauge Name
 * @param gaugeName
 * String Contains the gauge to update
 */
private void calculateGainedGaugeUnitsOf(String gaugeName) {
    this.properties.put(gaugeName, this.getTotalUnitsOf(gaugeName + "+= ", "\+= ");
}

/**
 * Calculates how many money spend an avatar.
 */
private void calculateMoneySpent() {

    // Retrives all remembered actions where money is spent
    int money_spent = this.getTotalUnitsOf("c_Human_cash-= ", "-= ");
    this.properties.put(Statistics.STATS_MONEY_SPENT, money_spent);
}

/**
 * Find the total units of the desired condition in the memory.
 * @param condition
 * String with the conditions to look for as a pattern
### J.1. STATISTICS

```java
private int getTotalUnitsOf(final String condition, final String splitCondition) {

    // Retrieves all remembered actions which spend money
    HashMap<String, Integer> spendingActions = this.getActionWithEffect(condition, splitCondition);
    HashMap<String, Integer> repetitions = new HashMap<String, Integer>();

    int amount = 0, unitsSpent = 0;

    // Examine all spending actions
    for (String NameAction : spendingActions.keySet()) {
        amount = 0;

        // Search in the memory how many occurrences of the action
        for (Plan plan : this.memory.getRememberedPlans()) {
            amount += plan.getNumberOfActions(NameAction) * this.memory.getFrequency(plan);
        }
        repetitions.put(NameAction, amount);
    }

    // Multiply all results: # occurrences * amount of money spent
    for (String NameAction : spendingActions.keySet()) {
        unitsSpent += spendingActions.get(NameAction) * repetitions.get(NameAction);
    }

    return unitsSpent;
}
```

/**
 * Given an action name it returns how many times has been executed.
 * @param String
 * @param String
 * @return int >= 0
 */
public int executedTimes(String gaugeName) {

    String condition = gaugeName + "+=";
    String splitCondition = "\+=";

    // Retrieves all remembered actions which spend money
    HashMap<String, Integer> spendingActions = this.getActionWithEffect("\n    condition, splitCondition");
    HashMap<String, Integer> repetitions = new HashMap<String, Integer>();

    int amount = 0, unitsSpent = 0;

    // Examine all spending actions
    for (String NameAction : spendingActions.keySet()) {
        amount = 0;

        // Search in the memory how many occurrences of the action
        for (Plan plan : this.memory.getRememberedPlans()) {
            amount += plan.getNumberOfActions(NameAction) * \n            this.memory.getFrequency(plan);
        }
        repetitions.put(NameAction, amount);
    }

    // Multiply all results: # occurrences * amount money spent
    for (String NameAction : repetitions.keySet()) {
        unitsSpent += repetitions.get(NameAction);
    }

    return unitsSpent;
}

/**
 * Retrieves actions with the specified effect
 * @param condition to identify the effect String
 * @param splitEquivalence String, tells how to divide the condition to get the number
 * @return HashMap<String, Integer>, where String
 * is the name of the action and Integer the
∗ amount of units which change.
∗ /
private HashMap<String, Integer> getActionWithEffect(String \ condition, String splitEquivalence) {

HashMap<String, Integer> actions = new HashMap<String, Integer>();
String actionName = null;
int amount = -1;

// Iterate all plans
for (Plan plan : this.memory.getRememberedPlans()) {

  // Iterate all plan elements
  for (PlanElement planElement : plan) {

    // Action name
    actionName = planElement.getService().getName();

    // To Avoid walkTo action and already known actions
    if (actionName.equals("walkTo") || actions.containsKey(actionName)) {
      continue;
    }

    // Iterate all over the effects and find the indicated pattern
    for (Effect effect :
      planElement.getService().getStatements().getActions()) {

      if (effect.toString().contains(condition)) {
        amount = Integer.parseInt(effect.toString().\ split(splitEquivalence)[1]);

        // To add action name and the amount of units
        actions.put(actionName, amount);
      }
    }
  }

  return actions;
}
```java
@Override
public String toString() {
    String output = "";
    for (Gauge gauge : this.agent.getGauges()) {
        output += gauge.getName() + " : " + this.getGainedGaugeOf(gauge.getName()) + " ( " + (this.getGainedGaugeOf(gauge.getName()) / this.properties.get(Statistics.STATS_TOTAL_TIME)) + ")\n";
    }

    return output;
}

/**
 * Calculates a index to value the gauge
 * @param gauge Gauge
 * @return float @param gauge
 * @return float >= 0
 */
public float getIndex(Gauge gauge) {
    return this.getIndex(gauge.getName());
}

public float getIndex(String gaugeName) {
    float value = this.properties.get(gaugeName);

    return value / this.properties.get(Statistics.STATS_AVERAGE_TIME);
}

/**
 * Get properties
 * @return HashMap<String, Integer>
 * @uml.property name="properties"
 */
public HashMap<String, Integer> getProperties() {
    return this.properties;
}
```

---

**APPENDIX J. MODULE ANTICIPATION MEMORY STATISTICS**

@Override
public String toString() {
    String output = "";
    for (Gauge gauge : this.agent.getGauges()) {
        output += gauge.getName() + " : " + this.getGainedGaugeOf(gauge.getName()) + " ( " + (this.getGainedGaugeOf(gauge.getName()) / this.properties.get(Statistics.STATS_TOTAL_TIME)) + ")\n";
    }

    return output;
}

/**
 * Calculates a index to value the gauge
 * @param gauge Gauge
 * @return float @param gauge
 * @return float >= 0
 */
public float getIndex(Gauge gauge) {
    return this.getIndex(gauge.getName());
}

public float getIndex(String gaugeName) {
    float value = this.properties.get(gaugeName);

    return value / this.properties.get(Statistics.STATS_AVERAGE_TIME);
}

/**
 * Get properties
 * @return HashMap<String, Integer>
 * @uml.property name="properties"
 */
public HashMap<String, Integer> getProperties() {
    return this.properties;
}
Appendix K

Module Anticipation Simulation

K.1 Simulation

```java
package fr.n7.saceca.u3du.model.ai.module.planning.anticipation.simulation;

import java.util.Iterator;
import fr.n7.saceca.u3du.model.ai.agent.Agent;
import fr.n7.saceca.u3du.model.ai.agent.Gauge;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.Plan;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.PlanElement;
import fr.n7.saceca.u3du.model.ai.agent.module.planning.UnreachableObjectException;
import fr.n7.saceca.u3du.model.ai.agent.module.reasoning.Goal;
import fr.n7.saceca.u3du.model.ai.service.Service;
import fr.n7.saceca.u3du.model.ai.statement.ExecutionMode;

/**
 * @author Albert Jornet Puig
 */
public class Simulation implements Iterator<Plan> {

    /**
     * The Constant logger.
     * @uml.property name="agent"
     * @uml.associationEnd multiplicity="(1 1)"
     */
```
/ **  // private static Logger logger = Logger.getLogger(DefaultPlanningModule.class);  

private Agent agent = null;

/**
 * @uml.property name="currentPlan"
 * @umlassociationEnd
 */
private Plan currentPlan = null;

/**
 * @uml.property name="currentGoal"
 * @umlassociationEnd
 */
private Goal currentGoal = null;

public Simulation(Agent agent) {
    this.agent = agent;
}

/**
 * If exists a plan then it returns it
 * @return Plan
 */
public Plan getPlan() {
    return this.currentPlan;
}

/**
 * Concludes if there is a result
 * @return boolean
 */
private boolean hasGoals() {

    // Plan not found or not executed
    if (this.agent.getPlanningModule().getCurrentPlan() == null) {
        return false;
    }
K.1. SIMULATION

// Empty Plan, which means it cannot be solved.
if (this.agent.getPlanningModule().getCurrentPlan().size() == 0) {
    return false;
}

// Save current plan and goal
this.currentPlan = this.agent.getPlanningModule().getCurrentPlan();
this.currentGoal = this.agent.getMemory().getGoals().get(0);

return true;
}

/**
 * Decrement Gauges N times
 */
private void advanceSteps(int ticks) {

    // Decrement Gauges ticks times
    for (int i = 0; i < ticks; i++) {
        this.periodicGaugesDecrement(this.agent);
    }
}

@Override
public boolean hasNext() {

    // Evaluate Gauges
    this.evaluate();

    // No Goals, then advance time N times
    if (!this.hasGoals()) {
        this.advanceSteps(500);

        // Evaluate Gauges
        this.evaluate();
    }

    // Avatar is dead
    if (this.isDead()) {
        return false;
    }
}
// Conclude if there are goals found
if (!this.hasGoals()) {
    return false;
}

// Apply effects on the plan
try {
    this.executePlan();
    this.advanceSteps(10);
} catch (UnreachableObjectException e) {
    e.printStackTrace();
}

return true;

@Override
public Plan next() {
    return this.currentPlan;
}

@Override
public void remove() {
    throw new UnsupportedOperationException();
}

/**
 * Check gauges and plan when needed
 */
public void evaluate() {
    /* Intelligence modules execution */
    this.agent.getPerceptionModule().perceive();
    this.agent.getEmotionModule().detectEmotions();
    this.agent.getReasoningModule().reason();
    this.agent.getPlanningModule().planAndExecute();
}

/**
 * Updates gauges periodically.
 */
* @param agent
* the agent
*/
private void periodicGaugesDecrement(Agent agent) {
   for (Gauge gauge : agent.getGauges()) {
      gauge.periodicDecrement(agent);
   }
}

/**
 * Checks if the agent is dead.
 * @param agent
 * @return true, if it is dead
 */
public boolean isDead() {
   for (Gauge gauge : this.agent.getGauges()) {
      if (gauge.getValue() == 0 && gauge.isSurvival()) {
         return true;
      }
   }

   return false;
}

/**
 * Sleep if pause.
 * @param a
 * the a
 */
public void sleepIfPause(Agent a) {
   if (a.isPause()) {
      a.sleep();
   }
}

/**
 * Executes current plan to bring its effects to agent.
 * @param
 */
public void executePlan() throws UnreachableObjectException {

    if (this.currentPlan == null) {
        return;
    }

    Iterator<PlanElement> iterator = this.currentPlan.iterator();
    Service service = null;

    while (iterator.hasNext()) {
        PlanElement planElement = iterator.next();
        service = planElement.getService();

        // Execute
        service.execute(planElement.getProvider(), this.agent, null, ExecutionMode.VIRTUAL);
    }

    this.currentGoal.setReachable(true);
    this.currentGoal.setReached(true);
}
Appendix L

Module View

L.1 Anticipation Window

```java
package fr.n7.saceca.u3du.view;

import java.util.Collection;

/**
 * Display anticipation data.
 * Basic layout has been generated with netbeans.
 * @author Albert Jornet Puig
 */
public class AnticipationWindow extends javax.swing.JPanel {

    private static final long serialVersionUID = 5176469157406632485L;

    /**
     * The ID agent
     * @uml.property name="id"
     */
    private long id;

    /**
     * The plan table model.
     * @uml.property name="plansTableModel"
     * @uml.associationEnd multiplicity="(1 1)"
     */
```
private TableModelDef plansTableModel;

/**
 * The statistics table model
 * @uml.property name="statisticsTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef statisticsTableModel;

/**
 * The object properties table model
 * @uml.property name="propertiesTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef propertiesTableModel;

public Long currentSelection;

/**
 * Parent frame
 * @uml.property name="parent"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JFrame parent;

/**
 * Creates new form AnticipationWindow
 * @param id
 * @param parent
 * @param JFrame
 */
public AnticipationWindow(long id, JFrame parent) {
    this.id = id;
    this.parent = parent;
    this.initComponents();
L.1. **ANTICIPATION WINDOW**

```java
private void initComponents() {

    this.parent.setTitle("Avatar anticipation " + this.id);
    this.jScrollPane = new javax.swing.JScrollPane();
    this.simulationsList = new javax.swing.JList();
    this.showsimulationGraphButton = new javax.swing.JButton();
    this.jScrollPane2 = new javax.swing.JScrollPane();
    this.objectDescriptionTable = new javax.swing.JTable();
    this.jScrollPane3 = new javax.swing.JScrollPane();
    this.statisticsTable = new javax.swing.JTable();
    this.jScrollPane4 = new javax.swing.JScrollPane();
    this.generatedPlansTable = new javax.swing.JTable();

    this.plansTableModel = new TableModelDef(new String[] {
        "#", "Plan"
    });
    this.statisticsTableModel = new TableModelDef(new String[] {
        "Name", "Value", "Times", "Index(Total/Av.)"
    });
    this.propertiesTableModel = new TableModelDef(new String[] {
        "Property", "Value"
    });

    this.simulationsList.setBorder(javax.swing.BorderFactory.
createTitledBorder("Simulations"));
    this.simulationsList.setModel(new javax.swing.AbstractListModel() {
        /**
         *
         */
        private static final long serialVersionUID = 1687790421296526042L;
        String[] strings = AnticipationWindow.this.listExaminedObject();

        @Override
        public int getSize() {
            return this.strings.length;
        }

        @Override
        public Object getElementAt(int i) {
            return this.strings[i];
        }
    });
```
```java
this.simulationsList.addListSelectionListener(new ObjectSelectionController(this));
this.jScrollPane1.setViewportView(this.simulationsList);

this.showSimulationGraphButton.setText("Show Graph");
this.showSimulationGraphButton.addActionListener(new DisplayGraphController(this));

this.jscrollPane2.setBorder(javax.swing.BorderFactory.createTitledBorder("Tested Object Properties"));

this.objectDescriptionTable.setBorder(null);
this.objectDescriptionTable.setModel(this.propertiesTableModel);
this.jscrollPane2.setViewportView(this.objectDescriptionTable);

this.jscrollPane3.setBorder(javax.swing.BorderFactory.createTitledBorder("Statistics"));

this.statisticstable.setModel(this.statisticstableModel);
this.jscrollPane3.setViewportView(this.statisticstable);

this.jscrollPane4.setBorder(javax.swing.BorderFactory.createTitledBorder("Generated Plans"));

this.generatedPlantsTable.setModel(this.plantsTableModel);
this.jscrollPane4.setViewportView(this.generatedPlantsTable);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
this.setLayout(layout);
layout.setHorizontalGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()                          
        .addComponent(this.jscrollPane1)                         
        .addComponent(this.showSimulationGraphButton)            
    ));
```
L.1. ANTICIPATION WINDOW

```java
...addComponent(t this .jScrollPane2 , javax.swing.GroupLayout.DEFAULT_SIZE , 537 , Short.MAX_VALUE)
...addComponent(t this .jScrollPane3 ,
     javax.swing.GroupLayout.Alignment.TRAILING)
...addComponent(t this .jScrollPane4 ))));
layout.setVerticalGroup(layout
   .createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addComponent(t this .jScrollPane1)
   .addGroup(
     layout.createSequentialGroup ()
     .addComponent(t this .showSimulationGraphButton)
     .addPreferredGap(javax.swing.GroupLayout.PREFERRED_SIZE , 1)
     .addComponent(t this .jScrollPane4 , javax.swing.GroupLayout.PREFERRED_SIZE , 115 ,
        javax.swing.GroupLayout.PREFERRED_SIZE)
     .addPreferredGap(javax.swing.GroupLayout.PREFERRED_SIZE , 1)
     .addComponent(t this .jScrollPane3 , javax.swing.GroupLayout.PREFERRED_SIZE , 147 ,
        javax.swing.GroupLayout.PREFERRED_SIZE)
     .addPreferredGap(javax.swing.GroupLayout.PREFERRED_SIZE , 1)
     .addComponent(t this .jScrollPane2 , javax.swing.GroupLayout.PREFERRED_SIZE , 148 ,
        Short.MAX_VALUE)))));
```
Get all those objects which has been analyzed.

```java
private String[] listExaminedObject() {
    Agent agent = this.getAgent();

    // LongAnticipation hasn't been executed
    if (agent.getPlanningModule().getLongAnticipation() == null) {
        return new String[0];
    }
```
L.1. ANTICIPATION WINDOW

Set<WorldObject> objects = agent.getPlanningModule().getLongAnticipation().\\getAnalyzer().getAlgorithm().getModifyingActions();
ResultWrapper analyzeResult = \\agent.getPlanningModule().getLongAnticipation().\\getAnalyzer().getBetterResult();

String[] output = new String[objects.size() + 1];
int counter = 0;

for (WorldObject object : objects) {
    output[counter] = object.getModelName() + \\
        " (" + object.getID() + ")";

    // Analyzer has picked this action as the best
    if (analyzeResult != null && analyzeResult.\\specialObject.getID() == object.getID()) {
        output[counter] += "∗";
    }

    counter++;
}
output[counter] = "Avatar";

return output;

/**
 * Get the agent instance
 *
 * @return Agent
 */
private Agent getAgent() {
    Map<Long, WorldObject> worldObjects = Model.getInstance().\\getAI().getWorld().getWorldObjects();
    return (Agent) worldObjects.get(this.id);
}

/**
 * Get wrapper instance of a given object
private ResultWrapper findResultInstance(Agent agent, long objectId) {

    for (ResultWrapper wrapper : agent.getPlanningModule().\getLongAnticipation().getAnalyzer().getResults()) {
        // Object found
        if (wrapper.specialObject.getId() == objectId) {
            return wrapper;
        }
    }
    return null;
}

/**
 * Display in JTable
 *
 * @param memory
 * @param Memory
 */
private void showAgentSimulatedPlans(Memory memory) {
    this.plansTableModel.clear();
    String textPlan = "";
    int row = 0;

    for (Plan plan : memory.getRememberedPlans()) {
        for (PlanElement planElement : plan) {
            // Exclude walkto
            if (planElement.service().getName().equals("walkTo")) {
                continue;
            }
            textPlan += planElement.service().getName() +\"" + planElement.getProvider().getId() + ") \->";
        }
    }
}
L.1. **ANTICIPATION WINDOW**

```java
} 
this.plansTableModel.setValueAt(textPlan, row, 1);
this.generatedPlansTable.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);
this.generatedPlansTable.getColumnModel().getColumn(0).setPreferredWidth(20);
this.generatedPlansTable.getColumnModel().getColumn(0).setWidth(20);
this.generatedPlansTable.getColumnModel().getColumn(0).setMinWidth(5);
this.generatedPlansTable.getColumnModel().getColumn(0).setMaxWidth(30);
this.plansTableModel.setValueAt(memory.getFrequency(plan), row, 0);
textPlan = "";
row++;
}
*/

/**
 * Display statistics table
 * @param stats Statistics
 */
private void showAgentStatistics(Statistics stats) {
    this.statisticsTableModel.clear();
    int row = 0;

    // Get instance
    // ResultWrapper wrapper = this.findResultInstance(
    //     agent, objectld);

    Iterator<Entry<String, Integer>> it = stats.getProperties().entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry<String, Integer> pairs = it.next();
        this.statisticsTableModel.setValueAt(pairs.getKey(), row, 0);
        // result = pairs.getValue() + " units (" +
```
```java
stats.executedTimes(pairs.getKey()) +
// " times, index="
// + stats.getIndex(pairs.getKey()) + ")";

this.statisticsTableModel.setValueAt(pairs.getValue() + " units", row, 1);
this.statisticsTableModel.setValueAt(stats.executedTimes(pairs.getKey()), row, 2);
this.statisticsTableModel.setValueAt(stats.getIndex(pairs.getKey()), row, 3);

row++;
}

/**
 * Display object properties
 *
 * @param object
 * WorldObject
 */
private void showObjectProperties(WorldObject object) {
    this.propertiesTableModel.clear();

    if (object == null) {
        return;
    }

    int row = 0;

    Collection<Property<?>> properties = object.getPropertiesContainer().getProperties();

    for (Property<?> property : properties) {
        this.propertiesTableModel.setValueAt(property.getModel().getName(), row, 0);
        this.propertiesTableModel.setValueAt(property.getValue(), row, 1);
        row++;
    }
```
public void setSelection(long objectId, boolean b) {

    WorldObject object = Model.getInstance().getAI().getWorld().getWorldObjects().get(objectId);

    // Object not found
    if (object == null) {
        return;
    }

    Agent agent = this.getAgent();

    this.currentSelection = this.id;
    // this.showAgentSimulatedPlans(agent, objectId);
    // Display plan

    ResultWrapper wrap = this.findResultInstance(agent, objectId);

    this.showAgentSimulatedPlans(wrap.result);
    this.showAgentStatistics(wrap.result.getStatisticsModule());
    this.showObjectProperties(wrap.specialObject);
}

/**
 * Display the selected simulated object memory
 * @param objectId long
 */
public void displayGraph(long objectId) {
Agent agent = (Agent) Model.getInstance().getAI().getWorld().getWorldObjects().get(this.id);

for (ResultWrapper wrapper : agent.getPlanningModule().getLongAnticipation().getAnalyzer().getResults()) {
    // Object found
    if (wrapper.specialObject.getId() == objectId) {
        wrapper.result.paint();
        // No need to keep iterating
        return;
    }
}

/**
 * Display the original planning simulation
 */
public void displayGraph() {
    Agent agent = this.getAgent();

    agent.getPlanningModule().getLongAnticipation().getMemory().paint();
}

/**
 * Display avatar information
 */
public void setCurrentAvatarSelection() {
    Agent agent = this.getAgent();

    this.currentSelection = this.id;
    this.showAgentSimulatedPlans(agent.getPlanningModule().getLongAnticipation().getMemory());

    this.showAgentStatistics(agent.getPlanningModule().getLongAnticipation().getMemory().getStatisticsModule());
    this.showObjectProperties(null);
}

L.2 SimulationWindow
L.2. SIMULATIONWINDOW

import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.Set;

import javax.swing.DefaultListModel;
import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JList;
import javax.swing.JMenuBar;
import javax.swing.JMenuBarItem;
import javax.swing.JPanel;
import fr.n7.saceca.u3du.model.ai.category.Category;
import fr.n7.saceca.u3du.model.ai.object.WorldObject;
import fr.n7.saceca.u3du.model.ai.object.properties.Property;
import fr.n7.saceca.u3du.model.ai.object.properties.UnknownPropertyException;
import fr.n7.saceca.u3du.model.ai.service.Service;

// TODO: Auto-generated Javadoc
/**
 * This class represents the 2D swing view of the application.
 * @author Sylvain Cambon & Jerome Dalbert &
 * Anthony Foulfoin & Johann Legaye & Aurelien Chabot
 */
public class SimulationWindow {

/**
 * The edition window.
 * @uml.property name="editionWindow"
 * @uml.associationEnd inverse="simulation:fr.n7.saceca.u3du.view.EditionWindow"
 */
public EditionWindow editionWindow;

/**
 * The frame.
 * @uml.property name="frame"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JFrame frame;

/**
 * The btn start.
 * @uml.property name="btnStart"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JButton btnStart;

/**
 * The time.
 * @uml.property name="time"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JLabel time;

/**
 * The canvas.
 * @uml.property name="canvas"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JPanel canvas;

private DefaultListModel agentsModel;

/**
 * The properties table model.
 * @uml.property name="propertiesTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef propertiesTableModel;

/**
 * The agents list.
 * @uml.property name="agentsList"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JList agentsList;

/**
 * The south Object infos pane.
 * @uml.property name="objectInfosPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JTabbedPane objectInfosPane;

/**
 * The categories table model.
 * @uml.property name="categoriesTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef categoriesTableModel;

/**
 * The services table model.
L.2. SIMULATIONWINDOW

* @uml.property name="servicesTableModel"
* @uml.associationEnd multiplicity="(1 1)"
*/
private TableModelDef servicesTableModel;

/**
 * The gauges panel.
 * @uml.property name="gaugesPanel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JPanel gaugesPanel;

/**
 * The gauges scroll pane.
 * @uml.property name="gaugesScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane gaugesScrollPane;

/**
 * The general infos table model.
 * @uml.property name="generalInfosTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef generalInfosTableModel;

public JTree objectsTree;

/**
 * The objects tree root.
 * @uml.property name="objectsTreeNode"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private DefaultMutableTreeNode objectsTreeNode;

/**
 * The objects tree categories.
 * @uml.property name="objectsTreeNodeCategories"
 */
private Map<String, DefaultMutableTreeNode> objectsTreeNodeCategories;
private Map<Long, DefaultMutableTreeNode> objectsTreeLeavs;

/**
 * The left agents objects tabbed pane.
 * @uml.property name="agentsobjectsTabbedPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JTabbedPane agentsObjectsTabbedPane;

/**
 * The general infos scroll pane.
 * @uml.property name="generalInfosScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane generalInfosScrollPane;

/**
 * The properties scroll pane.
 * @uml.property name="propertiesScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane propertiesScrollPane;

/**
 * The services scroll pane.
 * @uml.property name="servicesScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane servicesScrollPane;

/**
 * The categories scroll pane.
 * @uml.property name="categoriesScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane categoriesScrollPane;

/**
 * The console panel.
 */
L.2. SIMULATIONWINDOW

```java
/*
 * @uml.property name="consolePanel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JPanel consolePanel;

/**
 * The belonging table model.
 * @uml.property name="belongingsTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef belongingsTableModel;

/**
 * The belonging scroll pane.
 * @uml.property name="belongingsScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane belongingsScrollPane;

/**
 * The display panel.
 * @uml.property name="displayPanel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JPanel displayPanel;

/**
 * The display scroll pane.
 * @uml.property name="displayScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane displayScrollPane;

/**
 * The memory scroll pane.
 * @uml.property name="memoryScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane memoryScrollPane;

/**
 * The display scroll pane.
 * @uml.property name="displayScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane displayScrollPane;
```
* The messages inbox table model.
  * @uml.property name="messagesInboxTableModel"
  * @uml.associationEnd multiplicity="(1 1)"
  */
private TableModelDef messagesInboxTableModel;

/**
 * The messages inbox scroll pane.
 * @uml.property name="messagesInboxScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane messagesInboxScrollPane;

/**
 * The planning table model.
 * @uml.property name="goalsTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef goalsTableModel;

/**
 * The planning scroll pane.
 * @uml.property name="goalsScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane goalsScrollPane;

private AgentSelectionController agentSelectionController;

private ObjectSelectionController objectSelectionController;

/**
 * The gauges layout.
 * @uml.property name="gaugesGridBagLayout"
 */
private GridBagLayout gaugesGridBagLayout;

/**
 * The emitted messages table model.

L.2. SIMULATIONWINDOW

```java
/* @uml.property name="emittedMessagesTableModel"
   @uml.associationEnd multiplicity="(1 1)"
*/
private TableModelDef emittedMessagesTableModel;

/**
 * The emitted messages scroll pane.
 * @uml.property name="emittedMessagesScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane emittedMessagesScrollPane;

/**
 * The display grid bag layout.
 * @uml.property name="displayGridBagLayout"
 */
private GridBagLayout displayGridBagLayout;

/**
 * The agents scroll pane.
 * @uml.property name="agentsScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane agentsScrollPane;

/**
 * The objects scroll pane.
 * @uml.property name="objectsScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane objectsScrollPane;

/**
 * The memory tree root.
 * @uml.property name="memoryTreeRoot"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private DefaultMutableTreeNode memoryTreeRoot;

/**
 * The memory tree.
 */
```
/**
 * Contains the id of the current selected object.
 * Null if there is no selected object.
 * @uml.property name="currentSelection"
 */
public Long currentSelection;

/**
 * The plan table model.
 * @uml.property name="planTableModel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private TableModelDef planTableModel;

/**
 * The plan scroll pane.
 * @uml.property name="planScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane planScrollPane;

/**
 * The console text field.
 * @uml.property name="consoleTextField"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public JTextField consoleTextField;

/**
 * The console.
 * @uml.property name="console"
 * @uml.associationEnd multiplicity="(1 1)"
 */
public Console console;
The console text pane scroll.
* @uml.property name="consoleTextPaneScroll"
* @uml.associationEnd multiplicity="(1 1)"
*/
private JScrollPane consoleTextPaneScroll;

/**
 * The anticipation pane.
 * @uml.property name="anticipationPanel"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JPanel anticipationPanel;

/**
 * @uml.property name="anticipationScrollPane"
 * @uml.associationEnd multiplicity="(1 1)"
 */
private JScrollPane anticipationScrollPane;

/**
 * @uml.property name="anticipationGridLayout"
 */
private GridBagLayout anticipationGridLayout;

/** The logger. */
private static Logger logger = Logger.getLogger(SimulationWindow.class);

/**
 * Launch the application.
 *
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            SimulationWindow window = new SimulationWindow();
            window.frame.setVisible(true);
        }
    });
}
```java
/**
 * Display.
 */
public void display() {
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            SimulationWindow.this.initialize();
            SimulationWindow.this.getFrame().setVisible(true);
        }
    });
}

/**
 * Create the application.
 */
public SimulationWindow() {
    this.initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    this.frame = new JFrame("U3DU-4-SACECA - Simulation");
    this.frame.setMinimumSize(new Dimension(900, 762));
    this.frame.setBounds(100, 100, 900, 762);
    // Closing operation
    this.frame.addWindowListener(new ExitController());

    JMenuBar menuBar = new JMenuBar();
    this.frame.setJMenuBar(menuBar);
    // Window menu
    JMenu mnMenu = new JMenu("Mode");
    menuBar.add(mnMenu);
    JMenuItem edition = new JMenuItem("Edition");
    mnMenu.add(edition);
```
edition.addActionListener(new LaunchEditionWindowController(this));

// Help menu
JMenu helpMenu = new JMenu("?");
menuBar.add(helpMenu);
JMenuItem about = new JMenuItem("About");
helpMenu.add(about);
about.addActionListener(new AboutController(this.frame));

// $hide
if (this.canvas == null) {
    // $hide
    this.canvas = new JPanel();
    // $hide
    this.canvas = new JMECanvas();
    // $hide
}
// $hide

// Icon
java.awt.Image icon = Toolkit.getDefaultToolkit().getImage("./data/gui/agent_icon.png");
this.frame.setIconImage(icon);

// We register the controller in the AI in order to receive Aichanged notifications
Model.getInstance().getAI().getSimulation().addListener(new AIChangedController(this));
// We register the controller in the engine3D in order to receive object picking notifications
Model.getInstance().getGraphics().getEngine3D().addPickingObserver(new PickingController(this));
// We register the controller in the engine3D in order to receive new object creation
Model.getInstance().getGraphics().getEngine3D().addNewObjectObserver(new NewOrRemovedObjectController(this));

this.btnClose = new JButton("Start simulation");
APPENDIX L. MODULE VIEW

```java
this.btnStart.addActionListener(new StartSimulationController());

// The left pane
this.agentsObjectsTabbedPane = new JTabbedPane(SwingConstants.TOP);
// The south pane
this.objectInfosPane = new JTabbedPane(SwingConstants.TOP);

// SIMULATION TIME
JLabel lblNewLabel = new JLabel("Simulation Time: ");
this.time = new JLabel("0h 0m");

// THE GAUGES
this.gaugesPanel = new JPanel();
this.gaugesScrollPane = new JScrollPane(this.gaugesPanel);
this.gaugesGridLayout = new GridLayout();
this.gaugesGridLayout.columnWidths = new int[]{200, 200, 200, 200};
this.gaugesPanel.setLayout(this.gaugesGridLayout);

// The display
this.displayPanel = new JPanel();
this.displayScrollPane = new JScrollPane(this.displayPanel);
this.displayGridLayout = new GridLayout();
this.displayGridLayout.columnWidths = new int[]{200, 200, 200};
this.displayPanel.setLayout(this.displayGridLayout);

// The anticipation
this.anticipationPanel = new JPanel();
this.anticipationScrollPane = new JScrollPane(this.anticipationPanel);
this.anticipationGridLayout = new GridLayout();
this.anticipationGridLayout.columnWidths = new int[]{200, 200, 200};
this.anticipationPanel.setLayout(this.anticipationGridLayout);

// GENERAL INFOS
this.generalInfosTableModel = new TableModelDef(new String[]{
    "Name", "Value"
});
JTable generalInfosTable = new JTable(this.generalInfosTableModel);
this.generalInfosScrollPane = new JScrollPane(generalInfosTable);
```
// CATEGORIES
this.categoriesTableModel = new TableModelDef(new String[] {
    "Name", "Storage label" });
JTable categoriesTable = new JTable(this.categoriesTableModel);
this.categoriesScrollPane = new JScrollPane(categoriesTable);

// SERVICES
this.servicesTableModel = new TableModelDef(new String[] {
    "Name", "Is active" });
JTable servicesTable = new JTable(this.servicesTableModel);
this.servicesScrollPane = new JScrollPane(servicesTable);

// PROPERTIES
this.propertiesTableModel = new TableModelDef(new String[] {
    "Property", "Value" });
JTable propertiesTable = new JTable(this.propertiesTableModel);
this.propertiesScrollPane = new JScrollPane(propertiesTable);

// Belongings
this.belongingsTableModel = new TableModelDef(new String[] {
    "Object model", "Id" });
JTable belongingsTable = new JTable(this.belongingsTableModel);
this.belongingsScrollPane = new JScrollPane(belongingsTable);

// MEMORY
this.memoryTreeRoot = new DefaultMutableTreeNode();
this.memoryTree = new JTree(this.memoryTreeRoot);
this.memoryScrollPane = new JScrollPane(this.memoryTree);

// Goals
this.goalsTableModel = new TableModelDef(new String[] {
    "Goal", "Priority", "Reachable", "Reached" });
JTable goalsTable = new JTable(this.goalsTableModel);
this.goalsScrollPane = new JScrollPane(goalsTable);

// Planning
this.planTableModel = new TableModelDef(new String[] {
    "Service", "Parameters", "Provider" });
JTable planTable = new JTable(this.planTableModel);
this.planScrollPane = new JScrollPane(planTable);
// inbox messages
this.messagesInBoxTableModel = new TableModelDef(new String[] { "Sender", "Message" });
JTable messagesInBoxTable = new JTable(this.messagesInBoxTableModel);
this.messagesInBoxScrollPane = new JScrollPane(messagesInBoxTable);

// emmited messages
this.emmitedMessagesTableModel = new TableModelDef(new String[] { "Message" });
JTable emmitedMessagesTable = new JTable(this.emmitedMessagesTableModel);
this.emmitedMessagesScrollPane = new JScrollPane(emmitedMessagesTable);

// Agents list
this.agentsModel = new DefaultListModel();
this.agentsList = new JList(this.agentsModel);
this.agentsScrollPane = new JScrollPane(this.agentsList);
this.agentSelectionController = new AgentSelectionController(this);
this.agentsList.addListSelectionListener(this.agentSelectionController);
this.agentsObjectsTabbedPane.addTab("Agents", null, this.agentsScrollPane, null);

// Objects Tree
this.objectsTreeRoot = new DefaultMutableTreeNode();
this.objectsTree = new JTree(this.objectsTreeRoot);
this.objectsTree.setSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
this.objectSelectionController = new ObjectSelectionController(this);
this.objectsTree.addTreeSelectionListener(this.objectSelectionController);
this.objectsScrollPane = new JScrollPane(this.objectsTree);
this.objectsTreeCategories = new HashMap<String, DefaultMutableTreeNode>();
DefaultMutableTreeNode >();
this.objectsTreeLeafs = new HashMap<Long, DefaultMutableTreeNode >();
this.agentsObjectsTabbedPane.addTab("Objects", null, this.objectsScrollPane, null);

// CONSOLE
this.consolePanel = new JPanel();
JTextPane consoleTextPane = new JTextPane();
L.2. SIMULATIONWINDOW

```java
this.consoleTextPaneScroll = new JScrollPane(consoleTextPane);
this.consoleTextField = new JTextField();
this.consoleTextField.setColumns(10);
ConsoleSendController csc = new ConsoleSendController(this);
this.consoleTextField.addActionListener(csc);
JButton consoleBtnSend = new JButton("SEND");
consoleBtnSend.addActionListener(csc);
this.console = new Console(consoleTextPane);

GroupLayout groupLayout = new GroupLayout(this.frame.getContentPane());
groupLayout.setHorizontalGroup(groupLayout
    .createParallelGroup(Alignment.LEADING)
    .addGroup(groupLayout
        .createSequentialGroup()
        .addComponent(this.agentsObjectsTabbedPane, GroupLayout.PREFERRED_SIZE, 198,
        GroupLayout.PREFERRED_SIZE)
        .addGap(6)
        .addGroup(groupLayout
            .createParallelGroup(Alignment.LEADING)
            .addComponent(this.canvas, GroupLayout.PREFERRED_SIZE, 680, GroupLayout.PREFERRED_SIZE)
            .addGroup(groupLayout
                .createSequentialGroup().addGap(10).addComponent(this.time, GroupLayout.PREFERRED_SIZE, 288,
                GroupLayout.PREFERRED_SIZE)
            ))
        .addComponent(this.objectInfosPane, GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
        .addGap(10)
    )
    .setVerticalGroup(groupLayout
        .createParallelGroup(Alignment.LEADING)
        .addGroup(groupLayout
            .createSequentialGroup()
            .addComponent(this.agentsObjectsTabbedPane, GroupLayout.PREFERRED_SIZE, 509,
            GroupLayout.PREFERRED_SIZE)
            .addGroup(groupLayout
                .createSequentialGroup().addGap(10).addComponent(this.canvas,
                GroupLayout.PREFERRED_SIZE, GroupLayout.PREFERRED_SIZE)
            )
        ))
```
```java
GroupLayout.setLayout(this.frame.getContentPane(), groupLayout);

GridLayout glConsolePanel = new GridLayout(this.consolePanel);
glConsolePanel.setHorizontalGroup(glConsolePanel
    .createParallelGroup(Alignment.LEADING)
    .addComponent(this.consoleTextField, GroupLayout.PREFERRED_SIZE, 403,
    GroupLayout.PREFERRED_SIZE).addPreferredGap(ComponentPlacement.UNRELATED)
    .addComponent(consoleBtnSend).addContainerGap(377, Short.MAX_VALUE))
    .addComponent(this.consolePaneScroll, GroupLayout.DEFAULT_SIZE, 879, Short.MAX_VALUE)
glConsolePanel.setVerticalGroup(glConsolePanel.createParallelGroup(Alignment.TRAILING).addGroup(Alignment.LEADING,
    .addComponent(this.consolePaneScroll, GroupLayout.PREFERRED_SIZE, 403,
    GroupLayout.PREFERRED_SIZE).addPreferredGap(377, Short.MAX_VALUE))
    .addComponent(this.consolePaneScroll).addContainerGap(377, Short.MAX_VALUE))
    .addComponent(this.consolePaneScroll).addContainerGap(377, Short.MAX_VALUE))
```

L.2. SIMULATIONWINDOW

```java
private JFrame getFrame() {
    return this.frame;
}

public void initAgentsList() {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            SimulationWindow.this.agentsModel.clear();

            Collection<Agent> agents = Model.getInstance().getAI().getWorld().getAgents();
```

for (Agent agent : agents) {
    try {
        SimulationWindow.this.agentsModel.addElement(agent.
        getPropertiesContainer().getString(
            Internal.Agent.NAME) + " (#" + agent.getId() + ")");
    } catch (UnknownPropertyException e) {
        logger.error("Cannot properly select the agent.", e);
    }
}

/**
 * Initialize or reinitialize the object selection tree. The objects are sorted by model.
 * Invoked later
 */
public void initObjectsTree() {

    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            // We remove the object of its category node
            SimulationWindow.this.objectsTreeRoot = new
            DefaultMutableTreeNode();
            SimulationWindow.this.objectsTreeCategories.clear();
            SimulationWindow.this.objectsTreeLeafs.clear();
            ((DefaultTreeModel) SimulationWindow.this.objectsTree.getModel()).
            setRoot(SimulationWindow.this.objectsTreeRoot);

            Collection<WorldObject> objects = Model.getInstance().getAI().getWorld().getReactiveObjec

            for (WorldObject worldObject : objects) {
                // We check whether the model
                // node has already been created or not
                DefaultMutableTreeNode modelTreeNode = 
                SimulationWindow.this.objectsTreeCategories.get(worldObject.getModelName());

            }
        }
    });
}
if (modelTreeNode == null) {
    modelTreeNode = new DefaultMutableTreeNode(worldObject.getModelName());
    SimulationWindow.this.objectsTreeRoot.add(modelTreeNode);
    SimulationWindow.this.objectsTreeCategories.put(worldObject.getModelName(), modelTreeNode);
}
DefaultMutableTreeNode objectTreeNode = new DefaultMutableTreeNode(worldObject.getId());
modelTreeNode.add(objectTreeNode);
SimulationWindow.this.objectsTreeLeafs.put(worldObject.getId(), objectTreeNode);
}

// We expand the root node so that
// the categories become visible
SimulationWindow.this.objectsTree.expandRow(0);
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {

        if (!update) {
            SimulationWindow.this.objectInfosPane.addTab("Gauges", null,
                SimulationWindow.this.gaugesScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Memory", null,
                SimulationWindow.this.memoryScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Message inbox", null,
                SimulationWindow.this.messagesInboxScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Emmited messages", null,
                SimulationWindow.this.emmitedMessagesScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Goals", null,
                SimulationWindow.this.goalsScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Planning", null,
                SimulationWindow.this.planScrollPane, null);
        }

        // We show the gauges
        SimulationWindow.this.showAgentGauges(agent);

        // We show the memory
        SimulationWindow.this.showAgentMemory(agent);

        // We show the message inbox
        SimulationWindow.this.showAgentMessageInbox(agent);

        // We show the emmited messages
        SimulationWindow.this.showAgentEmmitedMessages(agent);

        // We show the goals
        SimulationWindow.this.showAgentGoals(agent);

        // We show the planning
        SimulationWindow.this.showAgentPlan(agent);
    }

});

/**}
* Display all the informations tabs existing
* for a given object. This method automatically
* removes all the tabs from the objectInfos tabs pane, and rebuild the tabs
*
* @param object
* the object for which we want to display the information tabs
* @param update
* False if the tabs must be removed and
* recreated (to display a different object).
* True if the tabs must just be updated
* (the object is still the same).
*/
private void showObjectInfos(final WorldObject object, final boolean update) {

SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        // We remove all the tabs
        if (!update) {
            // Remove tabs
            SimulationWindow.this.objectInfosPane.removeAll();

            // Recreate tabs
            SimulationWindow.this.objectInfosPane.addTab("General infos", null, SimulationWindow.this.generalInfosScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Properties", null, SimulationWindow.this.propertiesScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Categories", null, SimulationWindow.this.categoriesScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Services", null, SimulationWindow.this.servicesScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Belongings", null, SimulationWindow.this.belongingsScrollPane, null);

            SimulationWindow.this.objectInfosPane.addTab("Console", null, SimulationWindow.this.consolePanel, null);
            SimulationWindow.this.objectInfosPane.addTab("Display", null, SimulationWindow.this.displayScrollPane, null);
            SimulationWindow.this.objectInfosPane.addTab("Anticipation", null, SimulationWindow.this.anticipationScrollPane, null);
        }
    }
});
// We show the edit
SimulationWindow.this.showObjectEdit(object);

// We show the console
SimulationWindow.this.showObjectConsole(object);

// We show the general infos
SimulationWindow.this.showObjectGeneralInfos(object);

// We show the properties
SimulationWindow.this.showObjectProperties(object);

// We show the categories
SimulationWindow.this.showObjectCategories(object);

// We show the services
SimulationWindow.this.showObjectServices(object);

// We show the belongings
SimulationWindow.this.showObjectBelongings(object);

/**
 * Display the goals tab of an agent.
 * @param agent The agent for which we want to display the planning
 */
private void showAgentGoals(Agent agent) {
    this.goalsTableModel.clear();

    // We make a copy of goals list to avoid concurrent exceptions
    Object[] goals = agent.getMemory().getGoals().toArray();
    int row = 0;
for (Object obj : goals) {
    Goal goal = (Goal) obj;
    this.goalsTableModel.setValueAt(goal.getSuccessCondition(), row, 0);
    this.goalsTableModel.setValueAt(goal.getPriority(), row, 1);
    this.goalsTableModel.setValueAt(goal.isReachable(), row, 2);
    this.goalsTableModel.setValueAt(goal.isReached(), row, 3);
    row++;
}

/**
 * Display the plan tab of an agent.
 * @param agent The agent for which we want to display the planning
 */
private void showAgentPlan(Agent agent) {
    this.planTableModel.clear();
    Plan plan = agent.getPlanningModule().getPlan();
    int row = 0;
    if (plan != null) {
        for (PlanElement planElement : plan) {
            String parameters = "";
            if (planElement.getParameters() != null) {
                Set<String> parametersKey = planElement.getParameters().keySet();
                for (String key : parametersKey) {
                    if (!parameters.equals("")) {
                        parameters += ",";
                    }
                    Object param = planElement.getParameters().get(key);
                    if (param instanceof WorldObject) {
                        WorldObject wo = (WorldObject) param;
                        param = wo.getModelName() + "," + wo.getModelId() + "," + wo.getPosition().x + wo.getPosition().y + ""
                    }
                    parameters += key + ":" + param;
                }
            }
        }
    }
}
```java
APPENDIX L. MODULE VIEW

String serviceName = planElement.getService().getName();
// If the service is the service that is currently executed, we place a star after its name
serviceName = agent.getPlanningModule().getCurrentIndex() == row ? serviceName + " ∗" : serviceName;
this.planTableModel.getValueAt(serviceName, row, 0);
this.planTableModel.getValueAt(parameters, row, 1);
WorldObject provider = planElement.getProvider();
this.planTableModel.setValueAt(
    provider.getModelName() + " #" + provider.getId() + " @(" + provider.getPosition().x + ", " + provider.getPosition().y + ")", row, 2);
row++;
}

/**
 * Display the planning tab of an agent.
 * @param agent
 * The agent for which we want to display the planning
 */
private void showAgentEmmitedMessages(Agent agent) {
    this.emmitedMessagesTableModel.clear();
    Set<String> messages = Model.getInstance().getGraphics().getEngine3D().agents.get(agent.getId()).getEmmitedMessages();
    int row = 0;
    for (String message : messages) {
        this.emmitedMessagesTableModel.setValueAt(message, row, 0);
        row++;
    }
}

/**
 * Display the memory tab of an agent.
 * @param agent
 */
```
private void showAgentMemory(Agent agent) {
    this.memoryTreeRoot = new DefaultMutableTreeNode("Memory elements");
    ((DefaultTreeModel) this.memoryTree.getModel()).setRoot(this.memoryTreeRoot);

    Memory m = agent.getMemory();
    Map<Long, MemoryElement> memoryElements = m.getMemoryElements();

    for (MemoryElement memoryElement : memoryElements.values().toArray(new MemoryElement[0])){
        WorldObject worldObject = memoryElement.getWorldObject();
        String objectNodeName = "#" + worldObject.getId();

        if (worldObject.getId() == agent.getId()) {
            objectNodeName += " (Me)";
        } else {
            objectNodeName += "−" + worldObject.getModelName();
        }

        DefaultMutableTreeNode objectNode = new DefaultMutableTreeNode(objectNodeName);
        this.memoryTreeRoot.add(objectNode);
        objectNode.add(new DefaultMutableTreeNode("Forgettable : " + memoryElement.isForgettable));

        DefaultMutableTreeNode propertiesNode = new DefaultMutableTreeNode("Properties");
        objectNode.add(propertiesNode);

        Collection<Property<?>> properties = worldObject.getPropertiesContainer().getProperties();
        for (Property<?> property : properties) {
            propertiesNode.add(new DefaultMutableTreeNode(property.getModel().getName() + " : " + property.getValue()));
        }

        DefaultMutableTreeNode belongingsNode = new DefaultMutableTreeNode("Belongings");
        objectNode.add(belongingsNode);

        Collection<WorldObject> belongings = worldObject.getBelongings();
    }
}
for (WorldObject belonging : belongings) {
    belongingsNode.add(new DefaultMutableTreeNode(belonging.getModelName() + " (#" + belonging.getId() + ")" ));
}

DefaultMutableTreeNode servicesNode = new DefaultMutableTreeNode("Services");
objectNode.add(servicesNode);
Collection<Service> services = worldObject.getServices();

for (Service service : services) {
    servicesNode.add(new DefaultMutableTreeNode(service.getName()));
}

DefaultMutableTreeNode positionNode = new DefaultMutableTreeNode("Position : " + worldObject.getPosition().toString());
objectNode.add(positionNode);

this.memoryTree.expandRow(0);

/**
 * Display the messages inbox tab of an agent.
 * @param agent
 * The agent for which we want to display the messages inbox
 */
private void showAgentMessageInbox(Agent agent) {

    this.messagesInboxTableModel.clear();

    Queue<Message> messages = agent.getMemory().getMessageInbox();

    int row = 0;
    for (Message element : messages) {
        try {
            this.messagesInboxTableModel.setValueAt(
                element.getSenderId().getPropertiesContainer().getString(Internal.Agent.NAME) + " (#" +
                element.getSenderId().getId() + ")" , row, 0);
        } catch (UnknownPropertyException upe) {
            logger.error("Cannot properly select the agent.", upe);
        }
    }
private void showObjectEdit(WorldObject object) {

    this.displayPanel.removeAll();
    this.anticipationPanel.removeAll();

    JLabel graphLabel = new JLabel("Graph");
    String buttonText = Model.getInstance().getGraphics().getEngine3D().isWalkableGraphDisplayed() ? "Mask" : "Draw";
    JButton graphButton = new JButton(buttonText);
    graphButton.addActionListener(new GraphDisplayController());

    this.displayPanel.add(graphLabel);
    this.displayPanel.add(graphButton);

    GridBagConstraints graphLabelConstraint = new GridBagConstraints();
    graphLabelConstraint.gridy = 0;
    graphLabelConstraint.gridx = 0;
    graphLabelConstraint.gridwidth = 1;
    graphLabelConstraint.gridheight = 1;
    this.displayGridBagLayout.setConstraints(graphLabel, graphLabelConstraint);

    GridBagConstraints graphButtonConstraint = new GridBagConstraints();
    graphButtonConstraint.gridy = 0;
    graphButtonConstraint.gridx = 1;
    graphButtonConstraint.gridwidth = 1;
    graphButtonConstraint.gridheight = 1;
    this.displayGridBagLayout.setConstraints(graphButton, graphButtonConstraint);

    if (object instanceof Agent) {

    }
}
JLabel visionFieldLabel = new JLabel("Vision field");
JButton visionFieldButton = new JButton("Display");
visionFieldButton.addActionListener(new DisplayVisionFieldController(object.getId()));
this.displayPanel.add(visionFieldLabel);
this.displayPanel.add(visionFieldButton);

JLabel anticipationGraphLabel = new JLabel("Graph");
JButton anticipationGraphButton = new JButton("Show");
anticipationGraphButton.addActionListener(new DisplayAnticipationGraphController(object.getId()));
this.anticipationPanel.add(anticipationGraphLabel);
this.anticipationPanel.add(anticipationGraphButton);

JButton anticipationShowButton = new JButton("Show Simulation Window");
anticipationShowButton.addActionListener(new DisplayAnticipationWindowController(object.getId()));
this.anticipationPanel.add(anticipationShowButton);

GridBagConstraints visionFieldLabelConstraint = new GridBagConstraints();
visionFieldLabelConstraint.gridy = 0;
visionFieldLabelConstraint.gridx = 2;
visionFieldLabelConstraint.gridwidth = 1;
visionFieldLabelConstraint.gridheight = 1;
this.displayGridBagLayout.setConstraints(visionFieldLabel, visionFieldLabelConstraint);

GridBagConstraints visionFieldButtonConstraint = new GridBagConstraints();
visionFieldButtonConstraint.gridy = 0;
visionFieldButtonConstraint.gridx = 3;
visionFieldButtonConstraint.gridwidth = 1;
visionFieldButtonConstraint.gridheight = 1;
this.displayGridBagLayout.setConstraints(visionFieldButton, visionFieldButtonConstraint);
}

/**
 * Display the console tab of an object.
 * @param object
 * The object for which we want to display the informations
 */
private void showObjectConsole(WorldObject object) {

```java
private void showObjectBelongings(WorldObject object) {
    // We show the objects
    this.belongingsTableModel.clear();

    Collection<WorldObject> objects = object.getBelongings();
    int row = 0;
    for (WorldObject wObject : objects) {
        this.belongingsTableModel.setValueAt(wObject.getModelName(), row, 0);
        this.belongingsTableModel.setValueAt(wObject.getId(), row, 1);
        row++;
    }
}

private void showObjectGeneralInfos(WorldObject object) {
    // We show the info
    this.generalInfosTableModel.clear();

    this.generalInfosTableModel.setValueAt("Model name", 0, 0);
    this.generalInfosTableModel.setValueAt(object.getModelName(), 0, 1);

    this.generalInfosTableModel.setValueAt("Position", 1, 0);
    this.generalInfosTableModel.setValueAt(object.getPosition().toString(), 1, 1);

    this.generalInfosTableModel.setValueAt("Is paused", 2, 0);
    this.generalInfosTableModel.setValueAt(object.isPause(), 2, 1);
```
this.generAllInfosTableModel.setValueAt("Object id", 3, 0);
this.generAllInfosTableModel.setValueAt(object.getId(), 3, 1);
}

/**
 * Display the categories tab of an object.
 * @param object The object for which we want to display the informations
 */
private void showObjectCategories(WorldObject object) {

    // We show the categories
    this.categoriesTableModel.clear();

    Collection<Category> categories = object.getCategories();
    int row = 0;
    for (Category category : categories) {
        this.categoriesTableModel.setValueAt(category.getName(), row, 0);
        this.categoriesTableModel.setValueAt(category.getStorageLabel(), row, 1);
        row++;
    }
}

/**
 * Display the services tab of an object.
 * @param object The object for which we want to display the informations
 */
private void showObjectServices(WorldObject object) {

    // We show the services
    this.servicesTableModel.clear();

    Collection<Service> services = object.getServices();
    int row = 0;
    for (Service service : services) {
        this.servicesTableModel.setValueAt(service.getName(), row, 0);
        this.servicesTableModel.setValueAt(service.isActive(), row, 1);
        row++;
    }
/**
 * Display the gauges tab of an agent.
 * @param agent The agent for which we want to display the gauges
 */
private void showAgentGauges(Agent agent) {
    // We show the gauges
    this.gaugesPanel.removeAll();

    List<Gauge> gauges = agent.getGauges();

    for (int i = 0; i < gauges.size(); i++) {
        Gauge gauge = gauges.get(i);
        JLabel label = new JLabel(gauge.getName());
        JProgressBar progressBar = new JProgressBar((int) gauge.getMinValue(), (int) gauge.getMaxValue());
        progressBar.setValue(gauge.getValue().intValue());
        progressBar.setStringPainted(true);

        GridBagConstraints labelConstraint = new GridBagConstraints();
        labelConstraint.gridx = 0;
        labelConstraint.gridy = 0;
        labelConstraint.gridwidth = 1;
        labelConstraint.gridheight = 1;

        GridBagConstraints barConstraint = new GridBagConstraints();
        barConstraint.gridy = 0;
        barConstraint.gridx = 1;
        barConstraint.gridwidth = 1;
        barConstraint.gridheight = 1;

        if (i != 0) {
            if (i % 2 == 0) {
                labelConstraint.gridy = i / 2;
                labelConstraint.gridx = 0;
            } else {
                labelConstraint.gridy = i / 2;
                labelConstraint.gridx = 0;
            }
        }
    }
}
```java
} else {
    labelConstraint.gridy = i / 2;
    labelConstraint.gridx = 2;

    barConstraint.gridy = i / 2;
    barConstraint.gridx = 3;
}
this.gaugesGridBagLayout.setConstraints(label, labelConstraint);
this.gaugesGridBagLayout.setConstraints(progressBar, barConstraint);

this.gaugesPanel.add(label);
this.gaugesPanel.add(progressBar);
}
}

/**
 * Display the general properties tab of an object.
 * @param object
 * The object for which we want to display the informations
 */
private void showObjectProperties(WorldObject object) {
    // We show the properties
    this.propertiesTableModel.clear();

    Collection<Property<?>> properties = object.getPropertiesContainer().getProperties();
    int row = 0;
    for (Property<?> property : properties) {
        this.propertiesTableModel.setValueAt(property.getModel().getName(), row, 0);
        this.propertiesTableModel.setValueAt(property.getValue(), row, 1);
        row++;
    }
}

/**
 * This method is the only way for an external class to show tabs informations on a given object. This method has to be called from both 2D and 3D controllers if a new object is selected. If the given object is not selected in the swing agent or object list, this method selects it. The selection of the object in the agent list or the object tree does not produce events. The tabs corresponding to the given object are automatically displayed. If the method
 */
```
public void setSelection(long id, boolean focusJMEOnSelection) {

    WorldObject worldObject = Model.getInstance().getAI().getWorld().getWorldObjects().get(id);

    boolean update = false; // True if we have update the current object tabs, false if a new object is selected

    if (this.currentSelection != null && this.currentSelection == id) {
        update = true;
    }

    this.currentSelection = id;
    if (worldObject instanceof Agent) {
        // If this is a new selection
        if (!update) {
            // If an object is selected, we unselect it
            this.objectsTree.clearSelection();
            // We disable the listener otherwise the new selection in the list will throw an new event, and then, the controller will ask JME to focus on the selected object even if user just clicked on it in the JME canvas
            this.agentsList.removeListSelectionListener(SimulationWindow.this.agentSelectionController);
            // We select the agent
            try {
                this.agentsList.setSelectedValue(
                    worldObject.getPropertiesContainer().getString(Internai.Agent.NAME) + " (#" + worldObject.getId() + ")", true);
                } catch (UnknownPropertyException upe) {
                    logger.error("Cannot properly select the agent.", upe);
                }
            // We enable the listener so as the controller can detect the selection of a new agent from a user click.
            this.agentsList.addListSelectionListener(SimulationWindow.this.agentSelectionController);
            // We open the agent tab
APPENDIX L. MODULE VIEW

```java
    this.agentsObjectsTabbedPane.setSelectedIndex(0);
}

// We show the agent infos
this.showAgentInfos((Agent) worldObject, update);
}

else {
    // If this is a new selection
    if (!update) {
        // If an agent is selected, we unselect it
        this.agentsList.clearSelection();
        // We disable the listener otherwise the new selection in the list will
        // throw
        // an new event, and then, the controller will ask JME to focus on the
        // selected
        // object
        // even if user just clicked on it in the JME canvas
        this.objectsTree.removeTreeSelectionListener(this.objectSelectionController);
        // We select the object
        TreePath path = new TreePath(this.objectsTreeLeaves.get(id).getPath());
        this.objectsTree.setSelectionPath(path);
        // We enable the listener so as the controller can detect the selection
        // of
        // a new object from a user clic.
        this.objectsTree.addTreeSelectionListener(SimulationWindow.this.objectSelectionController);
        // We open the object tab
        this.agentsObjectsTabbedPane.setSelectedIndex(1);
    }

    // We show the agent infos
    this.showObjectInfos(worldObject, update);
}

if (focusJMEOnSelection) {
    // We set the focus on the object id
    Model.getInstance().getGraphics().getEngine3D().setFocusOn(id);
}
```

/**
 * Remove the specified object from the swing interface. Must be called before the AI
 */
public void removeObject(long id) {

    WorldObject worldObject = Model.getInstance().getAI().getWorld().getWorldObjects().get(this.getInstance().getAI().getWorld().getWorldObjects().get(id));
    this.currentSelection = null;

    if (worldObject instanceof Agent) {
        // We remove the agent from the agent list
        try {
            this.agentsModel.removeElement(worldObject.getPropertiesContainer().getString(Internal.Agent.NAME) + "(#" + worldObject.getId() + ")");
        } catch (UnknownPropertyException e) {
            logger.error("Cannot properly select the agent.", e);
        }
    } else {
        // We remove the object from the object tree
        DefaultMutableTreeNode objectNode = this.objectsTreeLeafs.get(id);

        // If it is the only child of its parent (the category), we also remove the
        // parent.
        DefaultMutableTreeNode category = (DefaultMutableTreeNode) objectNode.getParent();
        if (category.getChildCount() == 1) {
            // We remove the model category
            ((DefaultTreeModel) this.objectsTree.getModel()).removeNodeFromParent(category);
            this.objectsTreeCategories.remove(category.getUserObject());
        }

        // We remove the object of its category node
        ((DefaultTreeModel) this.objectsTree.getModel()).removeNodeFromParent(objectNode);

        // Of the nodes map
        this.objectsTreeLeafs.remove(id);
    }
    this.objectInfosPane.removeAll();
}

/**
  * Sets the edition window.
  *
  * @param editionWindow the new edition window
  */
    this.editionWindow = editionWindow;
}
Appendix M

Module Controller Anticipation

M.1 DisplayGraphController

```java
package fr.n7.saceca/u3du.controller.anticipation;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import fr.n7.saceca/u3du.view.AnticipationWindow;

/**
 * Displays a window which shows the simulated memory of an agent.
 * @author Albert Jornet Puig
 */
public class DisplayGraphController implements ActionListener {

    /**
     * The anticipation window instance.
     * @uml.property name="aw"
     * @uml.associationEnd multiplicity="(1 1)"
     */
    private AnticipationWindow aw;

    public DisplayGraphController(AnticipationWindow aw) {
        super();
        this.aw = aw;
    }
}
```
@Override
public void actionPerformed(ActionEvent e) {
    // We get the selected item
    Object selectedValue = this.aw.simulationsList.getSelectedValue();
    if (selectedValue != null) {
        // Alternative simulation
        String value = selectedValue.toString();
        if (!value.contains("Avatar")) {
            value = value.split("\(\)[1].split("\)\)[0];

            final long objectId = Long.parseLong(value);
            DisplayGraphController.this.aw.displayGraph(objectId);
        } else {
            // Original simulation
            DisplayGraphController.this.aw.displayGraph();
        }
    }
}

M.2 ObjectSelectionController

package fr.n7.saceca.u3du.controller.anticipation;

import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import fr.n7.saceca.u3du.view.AnticipationWindow;

/**
 * The Class AgentSelectionController.
 */
public class ObjectSelectionController implements ListSelectionListener {

    /**
     * The sw.
M.2. OBJECTSELECTIONCONTROLLER

```java
private AnticipationWindow aw;

/**
 * Instantiates a new agent selection controller.
 * @param sw the sw
 */
public ObjectSelectionController(AnticipationWindow aw) {
    this.aw = aw;
}

/**
 * Value changed.
 * @param arg0 the arg0
 */
@Override
public void valueChanged(ListSelectionEvent arg0) {
    Object selectedValue = this.aw.simulationsList.getSelectedValue();
    // Selects a solution
    if (selectedValue != null) {
        String value = selectedValue.toString();
        if (!value.contains("Avatar")) {
            value = value.split("\(\)\)[1].split("\)\)\]0);
            final long objectId = Long.parseLong(value);
            ObjectSelectionController.this.aw.setSelection(objectId, true);
        } else {
            // Selects avatar
            ObjectSelectionController.this.aw.setCurrentAvatarSelection();
        }
    }
}
```
APPENDIX M. MODULE CONTROLLER ANTICIPATION
Appendix N

Module Controller Simulation

N.1 Agent Selection Controller

```java
package fr.n7.saceca.u3du.controller.simulation;

import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import fr.n7.saceca.u3du.view.SimulationWindow;

/**
 * The Class AgentSelectionController.
 */
public class AgentSelectionController implements ListSelectionListener {

    private SimulationWindow sw;

    /**
     * Instantiates a new agent selection controller.
     *
     * @param sw the sw
     */
    public AgentSelectionController(SimulationWindow sw) {
        this.sw = sw;
    }
}
```
/**
 * Value changed.
 *
 * @param arg0
 * the arg0
 */

@Override
public void valueChanged(ListSelectionEvent arg0) {
    // We get the selected item
    Object selectedValue = this.sw.agentsList.getSelectedValue();
    if (selectedValue != null) {
        String value = selectedValue.toString();
        value = value.split("# ")[1].split(" ");
        final long agentId = Long.parseLong(value);
        AgentSelectionController.this.sw.setSelection(agentId, true);
    }
}

N.2 DisplayAnticipationGraphController

package fr.n7.saceca.u3du.controller.simulation;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import fr.n7.saceca.u3du.model.Model;
import fr.n7.saceca.u3du.model.ai.agent.Agent;

/**
 * show the anticipation memory graph.
 *
 * @author Albert Jornet Puig
 *
 */
public class DisplayAnticipationGraphController implements ActionListener {

    /**
     * the Agent
     */
N.3 DisplayAnticipationWindowController

```java
package fr.n7.saceca.u3du.controller.simulation;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;

import fr.n7.saceca.u3du.view.AnticipationWindow;

/**
 * Calls the anticipation window
 * @author Albert Jornet Puig
 */
public class DisplayAnticipationWindowController implements ActionListener {

    /**
     * Object to show the information
     * @uml.property name="id"
     */
    private long id;
}
```
```java
public DisplayAnticipationWindowController(long id) {
    super();
    this.id = id;
}

@Override
public void actionPerformed(ActionEvent e) {
    // Show window
    JFrame frame = new JFrame();
    AnticipationWindow anticipationWindow = new AnticipationWindow(this.id, frame);
    frame.getContentPane(anticipationWindow);
    frame.setSize(800, 600);
    frame.setLocation(100, 100);
    frame.setVisible(true);
}
```
Appendix O

Prolog

0.1 main

% kate: hl Prolog;

%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% MAIN CALL
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% getplan(integer, Output, int, int) – succeeds if a plan is found.
% It starts from 1 to MaxDepth until it finds an answer.
%
% MyIdent: avatar identification
% Plan – output –
% Hour: hour to start planning
% MaxDepth: max number of actions in a plan
getplan(MyIdent, Plan, Hour, MaxDepth):

    concat_atom([‘data/ai/tmp/avatars/’, MyIdent, ‘/’], PathAvatar).

    loader(PathAvatar, ‘goal.pl’),
    loader(PathAvatar, ‘memory.pl’),

    concat(‘avatar’, MyIdent, Id),
    between(1,MaxDepth, Depth).
Minutes is Hour * 60,
algorithm(Id, Plan, Minutes, Depth).

% LOAD ONCE
%
% load common files to all calls
% only need to be load once
%
% preload(Path):
% Path contains the path to other basic files
 preload(Path):-
   concat_atom([Path, 'util.pl'], PathUtil),
   concat_atom([Path, 'actions.pl'], PathActions),
   concat_atom([Path, 'body.pl'], PathBody),

% Load common data
consult(PathUtil),
consult(PathActions), % available actions
consult(PathBody). % main algorithm

O.2 body

% kate: hl Prolog:
/*-----------------------------------------------*/
/* PLANIFICATION */
/*-----------------------------------------------*/

/*
Minutes: sets to what time avatar schedules start
Depth: sets how many actions will be realized
ToStop: it indicates when to finish
because to continue iterating will be useless.
*/
algorithm(Avatar, P, Minutes, Depth) :-
   initialise(I, B, Minutes, Avatar),
% add to data current start minute
ccurrent(Avatar,
VCash,
VAccount,
VTiredness,
VThirsty,
VHunger,
VHappiness),

CAIIData = [VCash,
VAccount,
VTiredness,
VThirsty,
VHunger,
VHappiness,
Minutes],
nl,
write('Profondeur limite : '),
writeln(Depth),
writeln('**************************'),nl,

get_time(Xin),
mplan(I,B,P,Depth,F,CAIIData,Avatar),
write('Plan = '),
affiche(P),
nl,
write('Situation finale = '),
affiche(F),
nl,
get_time(Xout),
Y is Xout - Xin,
write('Temps calcul plan = '), writeln(Y),nl,
writeln('**************************'),nl.

initialise(I,B,Minutes, Avatar) :-
init(I, Minutes, Avatar),
% settime(I,Minutes),
nl,write('Situation initiale = '), affiche(I),
but(B),
nl,write('Situation visée = '), affiche(B).
/* plan is already accomplished, no need to execute algorithm */

mplan(_,_,_,P,_,_,_,Avatar) :-
  writeln('Conditions already accomplish, no need to exec algorithm'), nl,
  P = [],
  F = CData,
  ccurrent(Avatar,
    CCash ,
    CAccount ,
    CTiredness ,
    CThristy ,
    CHunger ,
    CHappiness ),
  caim(Avatar,
    CashAim ,
    BankAim ,
    TirednessAim ,
    ThristyAim ,
    HungerAim ,
    HappinessAim ),
  % check defined aim vs output
  CCash >= CashAim ,
  CAccount >= BankAim ,
  CTiredness >= TirednessAim ,
  CThristy >= ThristyAim ,
  CHunger >= HungerAim ,
  CHappiness >= HappinessAim ,
  nl.

/* we need to find a plan */
mplan(I,B,P,N,F,_,_,_,Avatar) :-
  M is N,
  writeln(CData),
  generf(I,F,P,M,CData,Avatar),
  inclus(B,F),
  % extracts information which must be satisfied
  caim(Avatar,
    CashAim ,
    ...
O.2. BODY

\[ \text{BankAim,} \]
\[ \text{TirednessAim,} \]
\[ \text{ThristyAim,} \]
\[ \text{HungerAim,} \]
\[ \text{HappinessAim),} \]

% algorithm data output
\[ \text{member(cash(robo,FinalCash),F),} \]
\[ \text{member(account(robo,FinalBank),F),} \]
\[ \text{member(tiredness(robo,FinalTiredness),F),} \]
\[ \text{member(thristy(robo,FinalThristy),F),} \]
\[ \text{member(hunger(robo,FinalHunger),F),} \]
\[ \text{member(happiness(robo,FinalHappiness),F),} \]

% check defined aim vs output
\[ \text{FinalCash} \geq \text{CashAim,} \]
\[ \text{FinalBank} \geq \text{BankAim,} \]
\[ \text{FinalTiredness} \geq \text{TirednessAim,} \]
\[ \text{FinalThristy} \geq \text{ThristyAim,} \]
\[ \text{FinalHunger} \geq \text{HungerAim,} \]
\[ \text{FinalHappiness} \geq \text{HappinessAim,} \]
\[ nI. \]

generf(E,F,[] ,0,_,_ ) :- \text{egal}(E,F),!.

generf(EI,EF,[\text{ACT} | \text{PLAN}].M,CData,Avatar) :-
\[ \text{P is M-1,} \]
\[ \text{P } \geq 0, \]
\[ \text{\% writeln(CData),} \]
\[ \text{trans(ACT, EI, E, CData, CNewData, Avatar),} \]
\[ \text{\% writeln(CNewData),} \]
\[ \text{generf(E,EF,PLAN,P,CNewData,Avatar).} \]

trans(ACT, EI, EF, CData, CNew, Avatar) :-
\[ \text{action(ACT,COND,SUP,AJ,CData,CNew,Avatar),} \]
\[ \text{inclus(COND, EI),} \]
\[ \text{deliste(SUP,EI,E),} \]
\[ \text{union(AJ,E,EF).} \]

% exemple de situations initiales et finales
i nit(El, Min, Avatar) :-
   % load current avatar
   c current(Avatar,
      InitCash,
      InitAccount,
      InitTiredness,
      InitThirsty,
      InitHunger,
      InitHappiness),
   El=[ heurecourante(Min),
      cash(robo, InitCash),
      account(robo, InitAccount),
      tiredness(robo, InitTiredness),
      thristy(robo, InitThristy),
      hunger(robo, InitHunger),
      happiness(robo, InitHappiness)].

but(EF):- EF = [ cash(robo, _),
                 account(robo, _),
                 tiredness(robo, _),
                 thristy(robo, _),
                 hunger(robo, _),
                 happiness(robo, _),
                 heurecourante(_)].

O.3 actions

% k ate: hl Prolog:

/*
  * Cash: +X to bank account, it depends on every avatar
  */
action(work(robo, CurrTime, NewTime, I dObject),
   [ heurecourante(CurrTime), account(robo, CBank) ],
   [ heurecourante(CurrTime), account(robo, CBank) ],
   [ heurecourante(NewTime), account(robo, PlusBank) ],
   [ CurrCash, CBank, CTired, CThrist, CHunger, CHappy, CurrTime],
   [ CurrCash, PlusBank, CTired, CThrist, CHunger, CHappy, NewTime],
   Avatar) :-
g etOffices(Avatar, I dObject),
getProperty(Avatar, c_Worker_workingTime, Duree),
NewTime is CurrTime + Duree,
getProperty(Avatar, c_Worker_salary, Salary), % avatar salary
PlusBank is CBank + Salary.

% +50 in cash at ATM
action(withdrawmoney(robo, Duree, IdObject),
 [heurecourante(CurrTime), cash(robo, CurrCash), account(robo, CurrBank)],
 [heurecourante(CurrTime), cash(robo, CurrCash), account(robo, CurrBank)],
 [heurecourante(NewTime), cash(robo, NewCash), account(robo, LessBank)],
 [CurrCash, CurrBank, CTired, CThrist, CHunger, CHappy, CurrTime],
 [NewCash, LessBank, CTired, CThrist, CHunger, CHappy, NewTime],
Avatar) :-
 % getObjectsId(Avatar, withdrawmoney, IdObject),
 getObjectsByPriorityId(Avatar, IdObject, withdrawmoney),
 Duree is 10,
 NewTime is CurrTime + Duree,
 NewCash is CurrCash + 50,
 LessBank is CurrBank - 50,
 LessBank >= 0.

% +500 in cash at ATM
action(withdrawmoneyatbank(robo, Duree, IdObject),
 [heurecourante(CurrTime), cash(robo, CurrCash), account(robo, CurrBank)],
 [heurecourante(CurrTime), cash(robo, CurrCash), account(robo, CurrBank)],
 [heurecourante(NewTime), cash(robo, NewCash), account(robo, LessBank)],
 [CurrCash, CurrBank, CTired, CThrist, CHunger, CHappy, CurrTime],
 [NewCash, LessBank, CTired, CThrist, CHunger, CHappy, NewTime],
Avatar) :-
 % getObjectsId(Avatar, withdrawmoneyatbank, IdObject),
 getObjectsByPriorityId(Avatar, IdObject, withdrawmoneyatbank),
 Duree is 10,
 NewTime is CurrTime + Duree,
 NewCash is CurrCash + 500,
 LessBank is CurrBank - 500,
 LessBank >= 0.

% +100 in tiredness
action(restathome(robo, CurrTime, NewTime, IdObject),
    [heurecourante(CurrTime), tiredness(robo, CurrTired)],
    [heurecourante(CurrTime), tiredness(robo, CurrTired)],
    [heurecourante(NewTime), tiredness(robo, PlusTired)],
    [CurrCash, CBank, CurrTired, CThrist, CHunger, CHappy, CurrTime],
    [CurrCash, CBank, PlusTired, CThrist, CHunger, CHappy, NewTime],
    Avatar) :-
    getHome(Avatar, IdObject),
    getProperty(Avatar, c_Human_restDuration, Duree),
    NewTime is CurrTime + Duree,
    PlusTired is CurrTired + 100.

% meal duration, it depends on each restaurant
% +100 in hunger
% -120 in cash
action(eatat(robo, Duree, IdObject),
    [heurecourante(CurrTime), cash(robo, CurrCash), hunger(robo, CHunger)],
    [heurecourante(CurrTime), cash(robo, CurrCash), hunger(robo, CHunger)],
    [heurecourante(NewTime), cash(robo, LessCash), hunger(robo, PlusHunger)],
    [CurrCash, CBank, CurrTired, CThrist, CHunger, CHappy, CurrTime],
    [LessCash, CBank, CurrTired, CThrist, PlusHunger, CHappy, NewTime],
    Avatar) :-
    % getObjectId(Avatar, eatat, IdObject),
    % getObjectsByPriorityId(Avatar, IdObject, eatat),
    % getPropertyObj(Avatar, IdObject, c_Restaurant_mealDuration, Duree),
    % Duree is 1,
    NewTime is CurrTime + Duree,
    PlusHunger is CHunger + 100,
    LessCash is CurrCash - 120,
    % NewTime <= 24,
    LessCash >= 0.

% film duration: it depends on each cinema
% cash: -30
% happiness: +100
action(watchmovie(robo, Duree, IdObject),
    [heurecourante(CurrTime), cash(robo, CurrCash), happiness(robo, CHappy)],
    [heurecourante(CurrTime), cash(robo, CurrCash), happiness(robo, CHappy)],
    [heurecourante(NewTime), cash(robo, LessCash), happiness(robo, PlusHappy)],
    [CurrCash, CBank, CurrTired, CThrist, CHunger, CHappy, CurrTime],
    [LessCash, CBank, CurrTired, CThrist, CHunger, PlusHappy, NewTime],
Avatar :-
  % getObjectsId(Avatar, watchmovie, IdObject),
  getObjectsByPriorityId(Avatar, IdObject, watchmovie),
  getPropertyValue(Avatar, IdObject, c_Cinema_filmDuration, Duree),
  NewTime is CurrTime + Duree,
  PlusHappy is CHappy + 100,
  LessCash is CurrCash - 30,
  LessCash >= 0.

% default time is 10, but in Saceca there isn't
% cash: -120
% thirsty: +20

action(drinkacan(robo, Duree, IdObject),
  [heurecourante(CurrTime), cash(robo, CurrCash), thirsty(robo, CThirst)],
  [heurecourante(CurrTime), cash(robo, CurrCash), thirsty(robo, CThirst)],
  [heurecourante(NewTime), cash(robo, LessCash), thirsty(robo, PlusThirst)],
  [CurrCash, CBank, CurrTired, CThirst, CHunger, CHappy, CurrTime],
  [LessCash, CBank, CurrTired, PlusThirst, CHunger, CHappy, NewTime],
  Avatar) :-
  % getObjectsId(Avatar, drinkacan, IdObject),
  getObjectsByPriorityId(Avatar, IdObject, drinkacan),
  Duree is 10,
  NewTime is CurrTime + Duree,
  PlusThirst is CThirst + 20,
  LessCash is CurrCash - 120,
  LessCash >= 0.

O.4 util

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Let the same predicates to be
% load from different files. IMPORTANT
% % This data is accessed from multiple files
% because each agent has its own info
% % Prolog Documentation:
%  http://www.cse.unsw.edu.au/~billw/prologdict.html
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
APPENDIX O. PROLOG

:- multifile object_properties/4.
:- multifile position/4.
:- multifile service/2.
:- multifile worldobject/4.
:- multifile current/7.
:- multifile caim/7.

% used in actions.pl

% It tells which building is the office
getOffices(Av, IdObject):-
    % determine which object is the avatar
    object_properties(Av, ObjAvatar, c_AccountOwner_account, _),

    % gets address of office
    object_properties(Av, ObjAvatar, c_Worker_companyName, Address),

    % search the object with this address
    object_properties(Av, IdObject, c_Company_name, Address),
    ObjAvatar = IdObject.

getHome(avatar<Id>, +obj<Id>) retrieves id avatars home
getHome(Av, IdObjects):-
    object_properties(Av, Object, c_BuildingOwner_address, Address),
    object_properties(Av, IdObjects, c_Building_address, Address),
    Object = IdObjects.

% get objects id from a given service
% First chooses with highest priority
% among those one, Random.
getObjectsId(Av, Service, IdObject):-
    object_properties(Av, _, c_AccountOwner_account, _),
    worldobject(Av, IdObject, _, Service).

% gets the value property from avatar
O.4. UTIL

getProperty(Av, Property, Value):-
  object_properties(Av, _, Property, Value).

getPropertyObj(Av, Obj, Property, Value):-
  object_properties(Av, Obj, Property, Value).

% Finds the highest number in PRIORITY
% predicate with the service Service
%
% more_points(+Avatar, −X, −Y, +Service)
more_points(Avatar, X, Y, Service):-
  worldobject(Avatar, IdObject, _, Service),
  worldobject(Avatar, IdOther, _, Service),
  priority(Avatar, X, P1),
  priority(Avatar, Y, P2),
  P1 > P2,
  IdObject == X,
  IdOther == Y.

% Returns all those objects with highest priority
% as well as they also use the service “Service”.
%
% getObjectsByPriorityId(+Avatar, −IdObject, +Service)
getObjectsByPriorityId(Avatar, IdObject, Service):-
  aggregate_all(count, worldobject(Avatar, IdObject, _, Service), N),
  N == 1 -> worldobject(Avatar, IdObject, _, Service) ;
  getObjectsByPriorityId(Avatar, _, IdObject, Service).

getObjectsByPriorityId(Avatar, Old, New, Service):-
  more_points(Avatar, Current, Old, Service),
  getObjectsByPriorityId(Avatar, Current, New, Service).

getObjectsByPriorityId(Avatar, Old, New, Service):-
  \+ more_points(Avatar, _, Old, Service),
  New = Old.

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % used in body.pl
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
egal(E, F) :-
    ((var(E); var(F)) -> E = F;
     inclus(E, F),
     inclus(F, E)).

inclus(E, F) :-
    forall (member(X, E), member(X, F)).

deliste([], L, L) :-
    !.
deliste([X|Y], Z, T) :-
    delete(Z, X, U),
    deliste(Y, U, T).

settime(I, M) :-
    getheure(M),
    member(heurecourante(M), I).

getheure(H) :-
    nl,
    write('Quelle heure est-il (0-18) ? : '),
    write(H),
    nl.

affiche(L) :-
    forall (member(S, L), (nl, tab(5), write(S))), nl.

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% used in main.pl
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

% load files, should go to utils
loader(Path, Name) :-
    concat_atom([[Path, Name], Out],
                consult(Out).
# List of Figures

4.1 Example’s tree expansion. ........................................... 36
4.2 Algorithm schematic representation. ............................... 37
4.3 Schematic Prolog Diagram. ......................................... 38
4.4 Temporal layout. .................................................... 40
4.5 UML of PrologWrapper.java ....................................... 43
4.6 UML of PrologInit class ........................................... 45
4.7 UML of PrologResult class ........................................ 51
4.8 New Prolog Integration Packages ................................. 51

5.1 Architecture of Anticipation. ..................................... 57
5.2 Architecture of Ui Anticipation. ................................... 59
5.3 UML of analyze package and its classes. ....................... 60
5.4 UML of Anticipation Memory package and its classes. ....... 61
5.5 Graph example. ..................................................... 62
5.6 UML of Statistics package and its class. ....................... 62
5.7 Displayed statistics data. ........................................... 63
5.8 UML of drawer package and its classes. ........................ 63
5.9 Example Circle Layout. ............................................ 64
5.10 UML Simulation. .................................................... 64
5.11 Avatar Anticipation Tab. .......................................... 66
5.12 Avatar Anticipation Window. .................................... 67

6.1 Anticipation Window Fail. ......................................... 70
6.2 Anticipation Window Success ...................................... 71

7.1 Prevision of dedication in Project ............................... 73
7.2 Final dedication in Project . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
Bibliography


[7]


   http://www.springerlink.com/content/978-3-540-87701-1/?MUD=MP&sort=p_OnlineDate&sortorder=desc