Modelling of Anti-Malarial Drug Compliance

[Preliminary Report]

Sebastian Rahlf[a][b]

with

Ray Paton[b]
Ian Hastings[a]
Dyfrig Hughes[c]
Peter McBurney[b]
Guy Barnish[a]

September 2004

LIVERPOOL SCHOOL OF TROPICAL MEDICINE

THE UNIVERSITY
of LIVERPOOL

[a] Liverpool School of Tropical Medicine, Molecular & Biochemical Parasitology Group, Pembroke Place, Liverpool L3 5QA

[b] Department of Computer Science, University of Liverpool, Chadwick Building, Peach Street, Liverpool L69 7ZF

[c] Department of Pharmacology and Therapeutics, University of Liverpool, Ashton Street Medical School, Ashton Street, Liverpool L69 3GE

Ray Paton (1954-2004)

It is with great sadness that we must announce the death of Dr Ray Paton on 29 July 2004.

Ray entered academia relatively late in life, after a spell as a teacher in a Liverpool high school. He joined the Department of Computer Science in 1989, initially as a research assistant in the area of knowledge-based systems. He became a Lecturer in 1991, and was promoted to Senior Lecturer in 2001, and then Reader in January 2004.

Ray’s main research interests were at the intersection of biology and computer science. He was an original, influential, and charismatic researcher, with collaborators across the world. Many computer scientists with an interest in biology are met with scepticism by researchers in the biology community, but Ray had the rare ability to win over researchers in both computer science and biology with his vision. Those who worked with him will readily attest to his enthusiasm and willingness to listen, and his skill at making connections between people and ideas. The author of innumerable research papers and books, Ray was also involved in founding and editing several journals.

As well as being a successful acadcemic, Ray was a dedicated and loving father and husband: He is survived by his wife Christine and two sons, Daniel and Andrew.

Acknowledgement

This research was financed by the DFID-funded Malaria Knowledge Programme at the Liverpool School of Tropical Medicine.

Abstract

We concentrate on understanding compliance in the treatment of uncomplicated (i.e. non-severe) malaria in children. Generally, the people involved in any decisions regarding the treatment of children are their mothers.

Due to the high cost of medication and consultation, and the low impact of training programs, mothers in Africa often decide either not to fully comply with the doctors' prescriptions or not to obtain the proper treatment in the first place.

Attempting to simulate how they would decide under certain circumstances would help a great deal in understanding the factors involved in making health care decisions. For this purpose a network was devised that contained all parameters we thought would be significantly involved, and their relationship to each other.

Using methods from the qualitative decision theory (QDT) and an agent-based modelling approach, the outcome showed a correlation between the cost of treatment and drugs, and the level of compliance.

Keywords: Non-compliance, uncomplicated malaria, QDT, simulation

Contents

Introduction

Malaria is a major killer. Each year there are 300 million clinical cases worldwide of which 90% occur in Sub-Saharan Africa—more than 1 million end fatal. Especially young children are at risk with a high mortality rate for those under the age of five. A solution is needed badly since the damage to the African economy alone is estimated at US$12 billion each year.

One big problem are the resources which are far from good. In most places in Africa doctors don't even have microscopes to check blood samples for malaria parasites. The WHO has therefore recommended to treat every fever attack as if it was malaria.

This would require mothers to obtain proper medication from the health authority, which can be quite a distance away. However, children come down with a fever about 4 to 5 times a year caused by normal childhood diseases or infections that have nothing to do with malaria. A mother with 5 children has to get new medication roughly every two weeks for one of her children. If the nearest health clinic is 10 miles down a muddy road, she may not want to put up with it if not absolutely necessary.

A further problem is that medical treatment is expensive and not everyone can afford it. In those circumstances it seems like a perfectly reasonable decision to save costs and, for example, not to administer the complete drug regimen but to save some for later when it is needed again, buy from a street vendor where the drugs are cheaper but the risk of drug failure due to fakes or bad manufacturing, or resort to traditional healing method.

If a child with a presumed malaria attack had been cured with paracetamol (because unbeknown to the mother it was just an ordinary flue), or half the dosage work just as well, the mother may not treat her child differently next time. A malaria relapse a few weeks later, which is quite common if the dosage instructions are not followed to the letter, may not be seen in connection with the fever before.

Understanding which factors are involved in the decision making process when it comes to adequately treating malaria is of great importance. It would enable researchers and health organisations to improve the health care situation by concentrating on certain aspects like education programs and may thus help to reduce the malaria mortality rate.

In the scope of this relatively small project, we concentrate on understanding compliance in the treatment of uncomplicated (i.e. non-severe) malaria in children. Severe childhood malaria is often regarded as a separate disease, frequently attributed to curses or other occult causes, while malaria during pregnancy is usually regarded as a separate topic requiring different interventions.

We will draw up a network of factors that we think should be taken into consideration. Applying methodologies from decision theory we will attempt to predict the behaviour of mothers using an agent based approach. The outcome will tell us how likely mothers are to take a certain course of action under certain conditions.

A model for the estimating non-compliance has been attempted before (Chickering and Pearl 1997). However, it did not try to predict which specific action the patients would take.

A note for non-computer scientists

This document is aimed at a readership from both biological and computing background. Specialised knowledge of those areas is therefore not required. However, since this paper has been written by a computer scientist, some use of mathematical notation is inevitable—but is kept fairly simple. Only section 2.4.3 requires the reader to understand how a *logic statement* is expressed in first-order logic. A short introduction can be found in Appendix C.

Chapter 1

Background Research

There are essentially three big areas involved in this project: understanding of *compliance to medical treatment* with all its social and economic relations, *reasoning under uncertainty* in order to determine the weight of each influencing factor individually, and the underlying *agent architecture* used for the implementation of the simulation later on.

1.1 Compliance

Compliance is in general defined as "the extend to which the patient's dosing history corresponds to the prescribed drug regimen" (Vermeire *et al.* 2001) (in the literature also frequently referred to as *adherence*[1]).

It is important to understand that the concept of compliance is a *process* of seeking and receiving and following treatment and advice which has many stages and many opportunities for non-compliance (Vermeire *et al.* 2001). It comprises of (1) the patient's *acceptance* of the treatment, (2) the *execution* of the prescribed dosing regimen, and (3) the *discontinuation* when the treatment ends.

Non-compliance can in general be defined as "a person's informed decision not to adhere to a therapeutic regimen" (Vermeire *et al.* 2001). There are three different types of compliance: if patients do not accept the treatment (i.e. do not redeem their prescriptions), we speak of *primary non-compliance*. *Dose-taking* or *drug regimen non-compliance* refers to the patient's not executing the treatment, including all forms of behaviours such as timing errors, double dosing, dose omission etc. And lastly, *persistence* to the continuation of treatment after its official end or *premature discontinuation*.

[1] The term adherence reduces attribution of greater power to the doctor in the doctor–patient relationship (Vermeire *et al.* 2001). There is a notion among doctors to switch away from the rather authoritarian concept of compliance to the more harmonic concept of *concordance* which encourages shared decision making between doctor and patient (Jones 2003). However, for the scope of this project there is no need to distinguish between these terms which have been introduced rather for political reasons. *Compliance* will do nicely.

1.1.1 Reasons for non-compliance

Compliance is affected by a variety of factors including the frequency of regimen (Table 1.1), the patient's knowledge, perception of health and benefits of therapy or behaviour, complexity of regime, and age (Held *et al.* 1994)—but surprisingly not on the type of disease.

Frequency of regimen	**Mean dose-taking compliance**
Once daily dosing	79%
twice daily	69%
three times daily	65%
four times daily	51%

Table 1.1: Association between dosing frequency and compliance: Patients generally comply much better with less frequent regimens.

What tends to be forgotten is that it is the patient who ultimately decides whether to take medicines. They have their own beliefs about their medicines and medicine in general, their own priorities and their own rational discourse in relation to health and care, risk and benefit. These may differ from (and sometimes contradict) those of doctors.

1.1.2 Consequences of non-compliance

The *clinical* consequences of non-compliance are numerous. A considerably high number of clinical cases is due to unrecognised lapses in dosing. Better compliance with oral contraceptives, for example, may prevent up to 700 000 unwanted pregnancies a year in the US and, it is suggested, up to 80% of acute transplant rejections which are due to nun-compliance to immunosupressive medication. Others include the extended duration or even the worsening of symptoms, the increased risk of infection or the disease's relapse (Hughes 2001).

In this context the terms *efficacy* and *effectiveness* are frequently used and are worth defining.

efficacy is how well a drug works under ideal conditions i.e. in a clinical trial where patients are selected carefully and monitored extensively.[2]

effectiveness is how well a drug performs in the *real world* i.e. where patients do not comply with the labelled dosis instructions.

Drugs shown to be effective in clinical trials, for example, often under-perform when used in routine clinical settings (Hughes 2001).

It is therefore only reasonable to assume that the *economic* consequences are for grater medical expenditure, since the effects of non-compliance are to reduce the efficacy of a drug and so increase the chances of treatment failure.

[2] It can therefore be assumed that $p_{\text{drug failure}} = 1 - \textit{efficacy}$.

1.1.3 Ways to improve compliance

For short-term regimens (2 weeks), compliance with medication is readily achieved by giving clear instructions. Chronic health problems, however, require mostly complex and labour-intesive methods but are unfortunately not predictably effective.

1.2 Reasoning under Uncertainty

Ideally, when a decision is reached *rationally*, all factors and likely events involved are weighted against each other and the option with the most favourable outcome is picked (*probabilistic reasoning*). However, most of the time the decision maker will have at best only a rough idea of how to prioritise the factors since in real life applications an assignment of fixed probabilities is seldom possible.

Out of this void of clarity a special branch of mathematics evolved that dealt primarily with *reasoning about uncertainty*. Unlike *classic decision theory*, this involves reasoning about dependencies, independences, and causality. An overview of the field is provided in Pearl (1988), Almond (1995), Jensen (1997), and Parson (2001).

1.2.1 Qualitative Probabilistic Networks (QPNs)

Qualitative Probabilistic Networks (QPNs) are an abstraction of Bayesian belief networks replacing numerical relations by qualitative influences and synergies (Wellman 1990)[3]. QPNs are weaker than their quantitative counterparts, but they can provide more robust results with much less effort. An expert may express his or her uncertain knowledge of a domain directly in the form of a QPN. This requires significantly less effort than a full numerical specification of a belief network (Druzdzel and Henrion 1993).

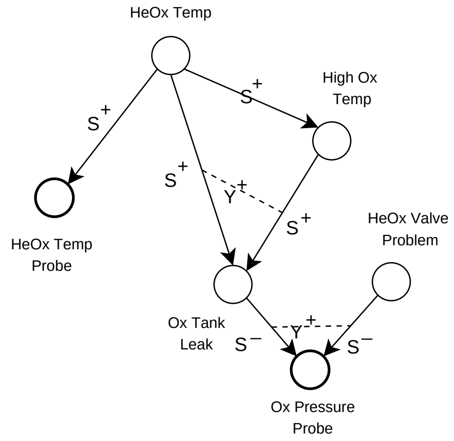


Figure 1.1: An example of a qualitative probabilistic network

[3] As cited in Druzdzel and Henrion (1993).

An example from Horvitz *et al.* (1992): Figure 1.1 shows a small fragment of an Orbital Maneuvering System (OMS) propulsion engine of the Space Shuttle.

> The OMS engine’s fragment captured by the network consists of two liquid gas tanks: an oxidizer and a helium tank. Helium is used to pressurise the oxidizer, necessary for expelling the oxidizer into the combustion subsystem. A potential temperature problem in the neighbourhood of the two tanks (*HeOx Temp*) can be discovered by a probe (*HeOx Temp Probe*) built into the valves between the tanks. An increased temperature in the neighbourhood of the two tanks can increase the temperature in the oxidizer tank (*High Ox Temp*) and this in turn can cause a leak in the oxidizer tank (*Ox Tank Leak*). A leak may lead to a decreased pressure in the tank. A problem with the valve between the two tanks (*HeOx Valve Problem*) can also be a cause of a decreased pressure in the oxidizer tank. The pressure in the oxidizer tank is measured by a pressure gauge (*Ox Pressure Probe*). Of all the variables in the network, only the values of the two probes (*HeOx Temp Probe* and *Ox Pressure Probe*) are directly observable. The others must be inferred.

Links in a QPN are labelled by signs of qualitative influences S^{δ}, each pair of links coming into a node is described by a sign of synergy between them. All in all there are four types of qualitative influence (S^{+} *(positive qualitative influence)*, S^{-} *(negative qualitative influence)*, S^{0} *(zero qualitative influence)*, and $S^{?}$ *(unknown qualitative influence)*—all of which are uncertain) and two types of synergies (Y^{δ} *(additive synergy)* and X^{δ} *(product synergy)* both of which can be *positive* or *negative* (e.g. Y^{+}), *zero* or *unknown*). An increased *XeOx Temp* will usually lead to an increased reading from the *HeOx Temp Probe*—but the probe may fail. However, the fact that an increased *HeOx Temp* makes an increased *HeOx Temp Probe* more probable is denoted by a positive influence S^{+} (Horvitz *et al.* 1992).

⊗	+	−	0	?
+	+	−	0	?
−	−	+	0	?
0	0	0	0	0
?	?	?	0	?

⊕	+	−	0	?
+	+	?	+	?
−	?	−	−	?
0	+	−	0	?
?	?	?	?	?

Table 1.2: Sign multiplication (⊗) and sign addition (⊕) operators (Wellman 1990)

Using the signs in Table 1.2, an evaluation function between the three nodes, for example, *Ox Tank Leak* (a), *HeOx Valve Problem* (b), and *Ox Pressure Problem* (c) can quite easily applied by using

$$S^{\delta}(\{a,b\},b) = S^{\delta_{ac}\oplus\delta_{bc}}$$

1.3 Agent Architecture

In the context of agent theory there are some terms that need to be explained as they will be used frequently later on and there can be much confusion to what they exactly mean.

> "An *agent* is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." (Wooldridge 2002, p. 15)

Although this definition has been much debated about, computer scientists agree that it is very important to distinguish *agents* from simple *objects*. While there are obvious similarities, there are also significant differences.

1. An object exhibits control over its *state* but not over its *behaviour*,

2. in contrast to objects, agents portray flexible (reactive, proactive, social) autonomous behaviour.

The *environment* is the world that an agent lives in. It is worth noting that there is a difference between a *static* environment that stays the same, and a *dynamic* environment that changes through influences from the outside or by the agent.

Agents are typically deployed in situations where an entity is needed that can act independently without any sort of control, like for example in the NASA mars missions where a robotic device has to react to changes in its environment and react accordingly. A mere object would not be capable of such a task.

In order to explain the principles of different agent architectures, the following simple example is useful.

1.3.1 The Blocks World

The Blocks World contains three blocks (A, B and C) of equal size on a table which can be picked up by a robot arm one at a time. The blocks may be placed on the table or on top of each other. The robot arm is now given the problem of stacking the boxes in a specific way, say A on top of B on top of C. Figure 1.2 shows a possible configuration of the problem.

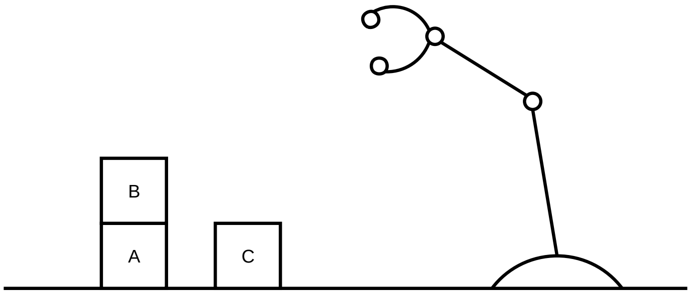


Figure 1.2: The Blocks World

This means that the robot not only needs to perceive its environment (i.e. it must have certain *beliefs* about its environment) but it must also be able to *plan* its actions in order to reach the desired goal. In the literature this is called *means-ends reasoning* (or simply *planning*), which is the process of deciding how to achieve an end using the available means. For this the agent needs the following

- A *goal* or *intention* which the agent wants to achieve, like putting A on top of B,
- the current state of the environment (the agent's *beliefs*), and
- the *actions* or *plans* available to the agent.

In the afore mentioned example the robot would believe that block A and C are on the table, block B on top of block A, and that its hand was currently empty.

In the past there have been a number of approaches as to how exactly this problem can be dealt with. All have their advantages and disadvantages, and discussing or explaining them is way beyond the scope of this project. However, there is one agent architecture which seems to be especially suited to dynamic environments.

1.3.2 The Belief–Desire–Intention (BDI) model

In contrast to the classical decision theory which explains the decision making behaviour rather mathematically, the BDI approach uses the cognitive concepts of *beliefs*, *desires* and *intentions* which are intuitive and easily accessible (Dastani and van der Torre 2004).

Beliefs are informal attitudes (how the world is expected to be), desires are the external and internal motivational attitudes, and intentions are the results of the decision making (Broersen *et al.* 2001). Figure 1.3 depicts the basic structure of a BDI architecture. A *reasoner* negotiates between these three and a number of plans to derive an intended action. These plans plans are atomic steps of instructions that will, when combined, form a description of how to solve the problem. They have certain *pre-conditions* that have to be fulfilled for the plan to be chose, and certain *post-conditions* that will be fulfilled if the plan is executed[4]. The challenge lies in designing an interpreter that is capable of devising an efficient line of thought. For the example of the *Blocks World*, the agent may use the following plans.

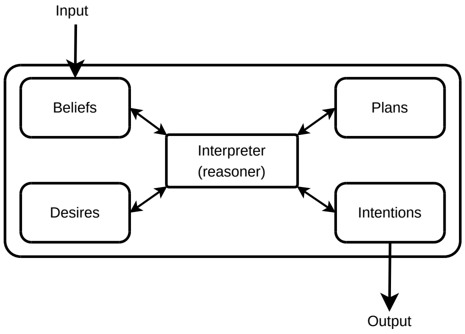


Figure 1.3: BDI architecture

[4] It is worth noting that for this relatively simple a *perfect world* is assumed where each plan will have the predicted outcome. For more complex examples this may not be the case!

stack(x, y) put box x on top of box y

pre-conditions: arm free $\wedge$ x on top $\wedge$ y on top

post-conditions: arm free $\wedge$ x on y

unstack(x) put box x on table

pre-conditions: arm free $\wedge$ x on top $\wedge$ $\neg$x on table

post-conditions: arm free $\wedge$ x on table

⋮

This approach lacks the clear formulation of decision rules like in classical decision theory, which combines the underlying qualitative concepts to decide which action to perform at any moment in time (Dastani and van der Torre 2004). On the other hand, because it first has to be worked out which goal to achieve and which actions to take, it is much more flexible when faced with unfamiliar situations and can deal with much more complex problems. It is also able to change its mind when the attempted solution turns out not to be achievable and can try settle for something else. It is therefore better suited for real-time applications in dynamic environments.

Chapter 2

System Specifications

2.1 General structure

Given the complex interactions this simulation requires, the best approach will probably be a hierarchical structure. This will also support the idea of individual-based modelling.

2.1.1 Household

A household is the smallest unit in the simulation and also used as the basis for most data collected in the field.

Its composition and structure varies greatly depending on tradition, religion and region. A household in northern Zimbabwe, for example, has the same monogamous structure as the typical western nuclear family we know. In the south, on the other hand, a polygamous family structure prevails, with the man in the household often having his own hut and his wives and their children living in huts surrounding the man.

Generally, the members of one household can be classified into 3 groups: infants (1 to 31 days), children (1 month to 10 years) and adults (older than 10 years)[1].

Although it is the men who bring home the money (if they don't spend it entirely on booze), it is assumed that the women make the decisions concerning the health of children (MacCormack 1988).

Mention the study about education of young women in Malawi!

A household in this context will therefore refer to a mother with her children, even if she lives in a polygamous household or together with other mothers in the same hut.

2.1.2 Village

A *village* can be understood as a collection of households. In some regions of Africa there are no villages in the European sense of the word, but rather a number of loosely grouped farms and settlements stretching over a larger area.

[1] At the age of 10, most children in sub-Saharan Africa are assumed to have developed sufficient (but not total!) immunity to survive malaria for the rest of their lives (assuming constant exposure to malaria parasites). For other regions such as Asia, this will look different.

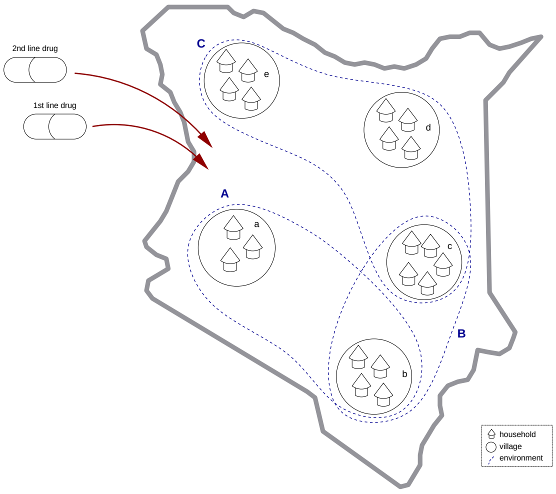


Figure 2.1: Schematic diagram of model interaction

Nevertheless, they still have a centre and some sort of village chief. The term village is intended to include both forms.

2.1.3 Environment

A environment is a set of villages for which a certain group of common input parameters apply. These input parameters can be of geographical, epidemiological or structural nature.

In this example (Figure 2.1) village b belongs to environment A and B but not to C, or mathematically speaking

$$b \in A \wedge b \in B \wedge b \notin C.$$

Therefore each and every household in village b takes the input parameters from village b, environment A and environment B.

2.1.4 Drug

Drugs are the one entity that do not fit in the proposed hierarchy. Since their values don't change, they will be dealt with as a *global entity*.

2.2 Notion of Time

It is very complex and difficult to put the simulated behaviour of mothers in relation with time. Each fever attack had to be tracked, treatment and recovery time had to be recorded, and also how long the mother needs to organise treatment had to be taken into account. Upon return, for instance, the child could already be dead. Yet another problem would be posed by overlapping fevers for mothers of two or more children that fall ill simultaneously.

However, we know the number of fevers per child per year and the proportion of malaria attacks. Therefore, for the sake of simplicity, the fevers will be simulated one at a time in discrete time steps without overlapping. Any time span can be modelled as a fraction or multiple of a year, and the data calculated accordingly.

2.3 Input parameters

Although designed to fit any sub-Saharan region in Africa, we will—as far as possible—take inputs based on studies from Kenya. Where this was not possible due to lack of sources, we have substituted, what we think, represents an average value for the missing input.

2.3.1 Hierarchy

The inputs will be derived from the *household*, the *village*, any *environment* it may be a member of, and the set of *global input parameters* (see Figure 2.2). The value of a single input will depend on how many other inputs of the same type apply to the household in question.

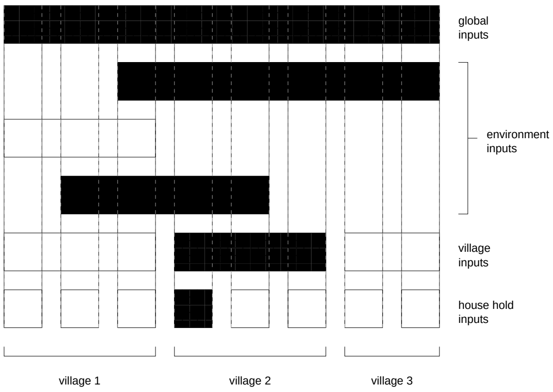


Figure 2.2: Diagram of hierarchical parameter structure

2.3.2 Relationships

All these parameters have to be seen in relation to each other in order to work out to what extend they individually influence the outcome. However, it is very difficult to express the relationship between inputs in numbers because we either don't know enough or the information available is not *certain*. A very elegant way to do just this is using *Qualitative Probabilistic Networks*.

Figure 2.3 shows how the inputs relate to each other. Only the coloured nodes will be evaluated, the node in the middle (Mother) does not belong to the network. This will be discussed in more detail below in section 2.4.1.

2.3.3 Clash of inputs

There are certain situations when inputs from different sources interfere with each other. Explicit rules will apply for each of these cases of how to deal with it individually.

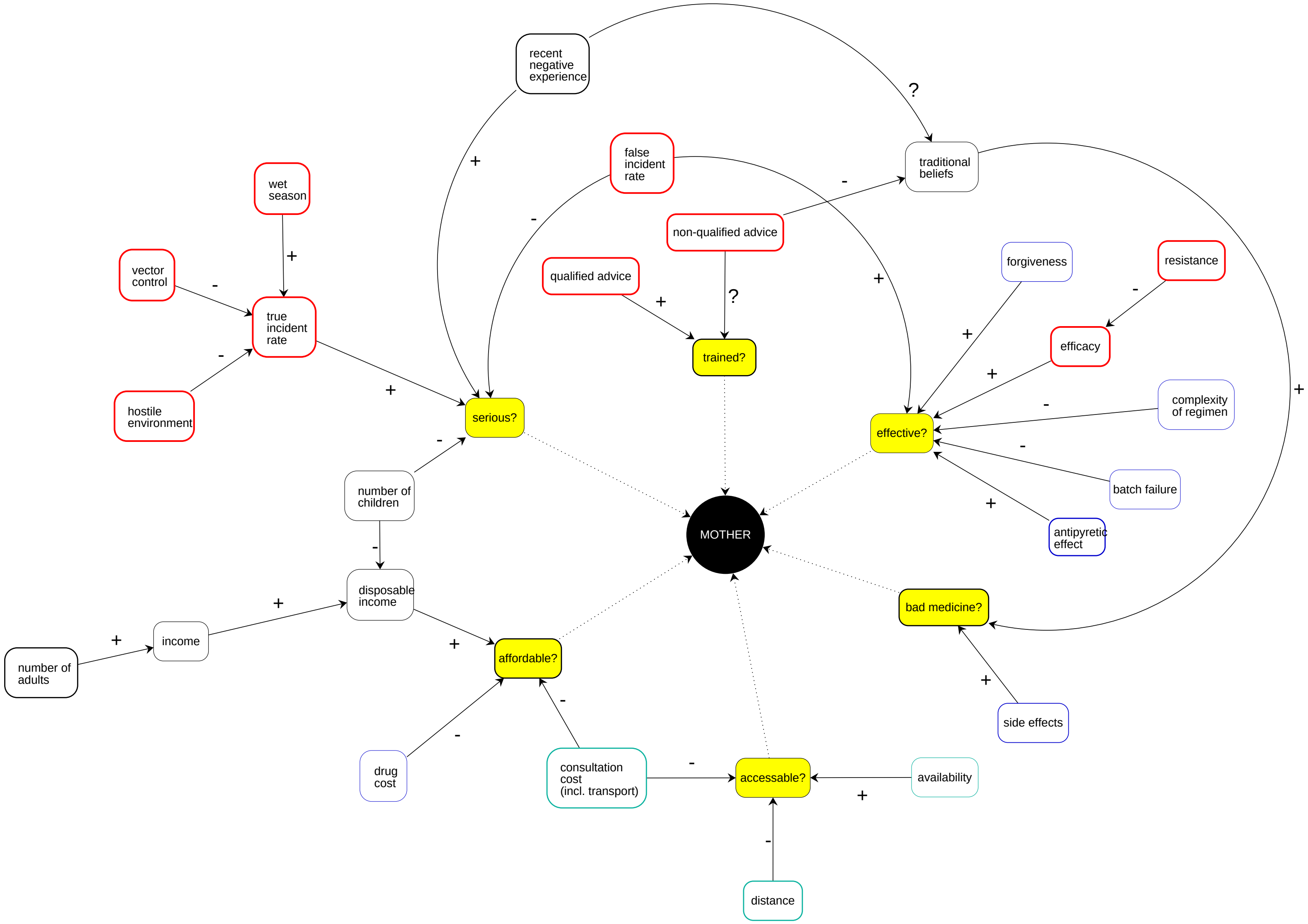


Figure 2.3: Input relationships

name	description	unit
House hold		
number of children	Members of household between 1 month and 10 years	number
number of adults	Members of household earning money	number
traditional beliefs	How likely are they to fall back on witch craft/religion rather than western medicine?	%
income	How much money is available and who decides how to spend it?	£/day
disposable income	How much money is available for health care?	% (of income)
Village		
distance	distance to nearest clinic (convenience)	km
availability	availability of health care/medication	%
consultation cost	how much does seeking advice cost?	£/consultation
recent negative experience	recent deaths of children in the village	number
Environment[a]		
true incident rate ⊡	How often does an infant/child come down with fever?	number of incidents per child per year
false incident rate ⊡	presumptive treatment of non-malarial fever	%
rain season ⊡	during rain season the number of incidents will be considerably higher	%[b]
hostile environment ⊡	climate/environment not favourable toward vector (e.g. too cold) [inhibits incident rate]	%[b]
vector control ⊡	How effectively are vector-controlling devices being used (e.g. bed nets, drainage) [inhibits incident rate]	%[b]
resistance ▽	How resistant are parasites in this environment?	%
qualified advice △	advice from health care worker/doctor/-chemist	%
non-qualified advice △	advice from neighbour etc.	%
Drug		
drug cost	cost of one treatment	£/treatment
efficacy	clinical efficacy	%
side effects	roughly probability × perceived severity	%
forgiveness (half-life)	Drug still works although compliance has not been 100%	%
complexity	How complex is regimen/packaging?	%
anti-pyretic effect	Does the drug alleviate fever?	0 or 1
batch failure/fakes	How likely are drugs to fail due to bad production/wrong ingredients?	%

[a] In case of a clash: ▽ lower one counts, △ higher one counts, ⊡ take the average

[b] units are % change in *true incident rate*

Table 2.1: Input parameters

If there are two (or more) inputs from different environments that apply to one and the same household, it has to be decided which one has the higher priority. In theory there can only be the following cases

- *lower one counts* The lowest value will be taken [$\min(i_1, i_2, \cdots, i_n)$]

- *higher one counts* The highest value will be taken [$\max(i_1, i_2, \cdots, i_n)$]
- *take the average* The average over all inputs will be taken [$\text{avg}(i_1, i_2, \cdots, i_n)$]
- *multiply* $i_1 \times i_2 \times \cdots \times i_n$
- *add* The sum of all inputs will be taken $\sum_i^n = i_i$

2.3.4 Translation

The best way to compare the different inputs effectively is when all have the same value, ideally this would be a percentage (a number between 0 and 1). Where this is not the case, the input has to be converted.

Table 2.2 provides a series of translation ranges to convert the inputs. If not stated otherwise, the weight is distributed evenly. For example, given the range {*1–2, 3–7, 7+*}, values 1 and 2 will be translated into $\frac{1}{3}$, 3 through to 7 into $\frac{2}{3}$, and anything above 7 into $\frac{3}{3} = 1$.

Some inputs, however, will be translated using a *weighted range*. The more complex a drug is to use, for example, the more influence this input will have. Given the range {*simple* ↦ 0.2*, typical* ↦ 0.4*, complex* ↦ 1.0}, a typical drug will have the value .4 (rather than $\frac{2}{3}$) and a simple one only .2 (instead of $\frac{1}{3}$).

name	unit	translates to
House hold		
number of children	number	{*1–2, 3–7, 7+*}
number of adults	number	{*1–2, 3–7, 7+*}
traditional beliefs	%	{*modern, liberal, religious*}
income	£/day	{*very poor, . . . , very rich*}
Village		
distance	km	{*<2km, 2-5km, >5km*}
consultation cost	£/consultation	{*nominal, moderate*[a]*, expensive*}
recent negative experience	number	{*0, 1–2, 3–5, 6–9, 9+*}
Environment		
true incident rate	number of incidents p.c.p.y.	{*low transmission, normal tr., high tr.*}
Drug		
drug cost	£/treatment	{*nominal, moderate*[1]*, expensive*}
complexity (regimen/packaging)	%	{*simple* ↦ 0.2*, typical* ↦ 0.4*, complex* ↦ 1.0}

[a] $< \$1/treatment$

Table 2.2: Input translation

2.4 The Mother

The mother acts as an *autonomous agent* which interacts with its environment. In order to achieve this, the behaviour has to be defined beforehand.

2.4.1 Beliefs

The input parameters (Figure 2.3) feed into 6 beliefs the mother has about her environment, and which will determine her actions.

serious? The mother believes the child to be seriously ill and fears for its life.

affordable? The mother believes that she can afford anti-malarial drugs/professional treatment for the child.

accessible? The mother believes that medical help is accessible, i.e. a health care facility is in reasonable distance, medication is available and transport within her means.

effective? The mother believes the medical treatment/the drug to be effective against the child's fever.

trained? The mother knows what to do in order to deal with her child's fever appropriately, i.e. she is more inclined to use anti-malarial medicine rather than anti-pyretic drugs or the services of a traditional healer.

bad medicine? The mother believes that (western) medicine is a bad alternative to traditional treatment and cannot cure her child's fever.

Additionally, the mother has four other beliefs about her environment.

fever? The mother believes that her child is ill. This belief will always be true since the decision process only commences if the child has a fever. It is the mother's intention to always achieve ¬fever? or at least ¬serious?.

drug? The mother believes that she has medicine available that she can give to her child. This also includes medicine which has been previously bought but not used.

enough? The mother believes that she has medicine *and* enough to treat her child properly. This includes *old medicine*, too. Naturally, if enough? is **true** then drug? has to be **true** as well.

save money? The mother believes that this particular plan of action will save her money.

Evaluation

Due to time constraints, the evaluation will be as simple as possible. Only the coloured belief nodes (Figure 2.4) are of interest and will be evaluated using the following formulas with only the percentages/translated values of the parameters are used as inputs. They are derived from the household (h_n), the village (v_n), the environment (e_n), and drug (d_n) as described in Table 2.1, where the number n refers to the position of the input in the table.

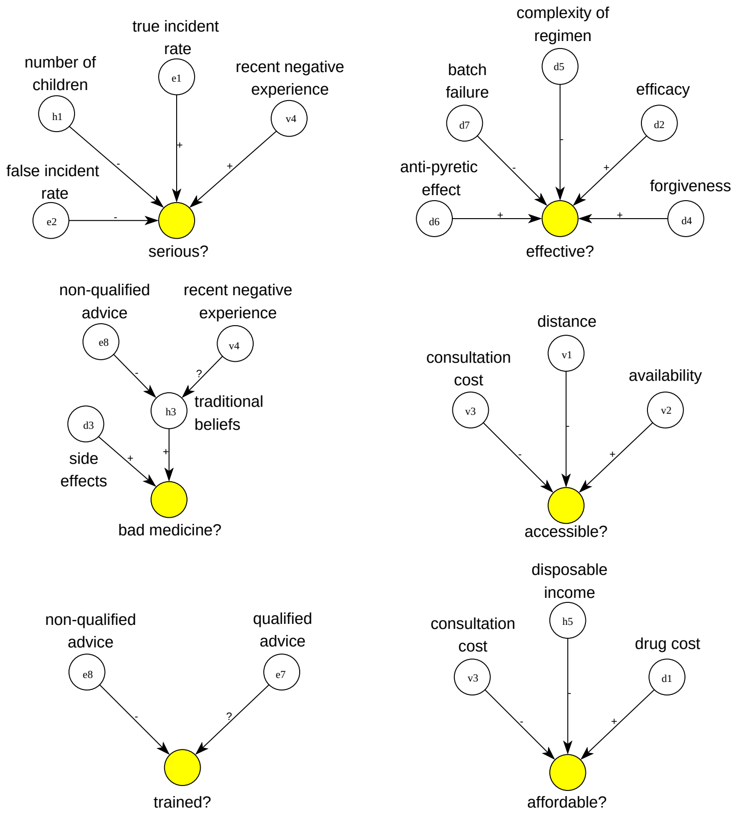


Figure 2.4: Belief evaluation

$$\text{serious?} = \frac{e_1 + v_4 - h_1 - e_2}{4}$$

$$\text{effective?} = \frac{d_6 - d_7 - d_5 + d_2 + d_4}{5}$$

$$\text{accessible?} = \frac{\frac{v}{2} - v_1 - v_3}{3}$$

$$\text{affordable?} = \frac{d1 - v_3 - h_5}{3}$$

The last two are more problematic because we are not sure how some inputs influence the outcome. For the sake of simplicity, these cases will be treated as positive influences ($S^? = S^+$).

$$\text{trained?} = \frac{e_7 - e_8}{2}$$

$$\text{bad medicine?} = \frac{d_3 + \frac{v_4 - e_8}{2}}{2} = \frac{4d_3 + v_4 - e_8}{4}$$

Plausibility

Once the beliefs are evaluated a value will be assigned to them. In classic (crisp) logic that would *only* be 0 (for **false**) or 1 (for **true**). In the problem domain an outcome of this clarity is highly unlikely, especially taken into consideration the vast amount of factors involved in deriving the result. It is better to assume that if a belief is *plausible* enough, say to 70%, it is deemed to be **true**.

Range

It is important to note that if a belief has not been explicitly set to **true**, it is assumed to be **false**—even if this belief was previously **true**.

2.4.2 Desires

A mother will essentially have two desires: to *save her child* and, where possible, *save money* (which can be spent on subsequent medical care of the child or siblings). For the sake of simplicity it is assumed that a mother will always choose her child's life over financial benefits.

2.4.3 Plans

The plans are going to be very straight forward and don't have to be complex. The notation below takes the form

{n} name : description

$$\big[\ \textit{pre-condition}\ \big]_t \rightarrow \big[\ \textit{outcome the mother hopes for}\ \big]_{t+1}$$

where the *pre-condition* has to be fulfilled at time t and the *outcome* is what the mother hopes for to become true at time $t+1$. The *instructions* are executed and the beliefs will be updated in the following round.

Only one plan can be executed by the mother at any one time. If more than one plan is triggered, the plan with the higher number (n) will be chosen.[2]

The following plans are derived from (Guyatt and Snow 2004) and (Mwenesi, Harpham, and Snow 1995).

{0} do nothing : The mother does not do anything to treat her child.

$$\big[\ \textit{none}\ \big]_t \rightarrow \big[\ \textit{none}\ \big]_{t+1}$$

This is the default action when no other plan can be chosen.

{1} buy anti-pyretic drug : The mother buys *only* an anti-pyretic drug

$$\begin{bmatrix} \textsf{fever?} \wedge \neg\textsf{serious?} \\ \wedge\neg\textsf{drug?} \wedge \neg\textsf{enough?} \\ \wedge\neg(\textsf{affordable?} \vee \textsf{accessible?}) \\ \wedge\neg\textsf{bad medicine?} \end{bmatrix}_t \rightarrow \big[\ \textsf{save money?} \vee \neg\textsf{fever?}\ \big]_{t+1}$$

[2] A complete list of all possible combinations can be found in Appendix D

{2} buy part of an anti-malarial treatment : The mother buys an anti-malarial drug

$$\begin{bmatrix} \text{fever?} \land \text{serious?} \\ \land\neg\text{drug?} \land \neg\text{enough?} \\ \land\neg\text{affordable?} \land \text{trained?} \\ \land\text{effective?} \land \neg\text{bad medicine?} \end{bmatrix}_t \to \begin{bmatrix} (\text{save money?} \\ \lor\neg\text{fever?} \lor \neg\text{serious?}) \\ \land\text{drug?} \end{bmatrix}_{t+1}$$

{3} buy complete anti-malarial treatment : The mother buys an anti-malarial drug

$$\begin{bmatrix} \text{fever?} \land \text{serious?} \\ \land\neg\text{enough?} \\ \land(\text{affordable?} \lor \text{trained?}) \\ \land\text{effective?} \land \neg\text{bad medicine?} \end{bmatrix}_t \to \begin{bmatrix} (\neg\text{fever?} \lor \neg\text{serious?}) \\ \land\text{drug?} \end{bmatrix}_{t+1}$$

{4} go to health care facility : The mother goes to a health care facility where her child is given an anti-malarial and an anti-pyretic drug.

$$\begin{bmatrix} \text{fever?} \land \text{serious?} \land \text{affordable?} \land \text{accessible?} \\ \land\text{trained?} \land \text{effective?} \land \neg\text{bad medicine?} \end{bmatrix}_t \to \begin{bmatrix} \neg\text{fever?} \lor \neg\text{serious?} \end{bmatrix}_{t+1}$$

Then the mother has to make two decisions

{5} give part course : The mother gives *only* the initial dose to the child and saves the rest for later.

$$\begin{bmatrix} \text{fever?} \land \text{drug?} \\ \land\neg\text{serious?} \lor \text{bad medicine?} \end{bmatrix}_t \to \begin{bmatrix} \text{save money?} \lor \neg\text{fever?} \end{bmatrix}_{t+1}$$

{6} give full course : She gives the drug to child as prescribed.

$$\begin{bmatrix} \text{fever?} \land \text{serious?} \\ \land\neg\text{enough?} \land \text{trained?} \end{bmatrix}_t \to \begin{bmatrix} \neg\text{fever?} \lor \neg\text{serious?} \end{bmatrix}_{t+1}$$

Please note that $[\text{save money?} \lor \neg\text{fever?}]$ is not as heartless as it sounds if you bare in mind that $\lor$ is *not* an exclusive *either . . . or*. The more hopes are fulfilled the better.

2.5 The Simulation

The simulation itself combines the in the previous sections discussed components and generated a meaningful output. Only a few questions remain before the implementation can commence.

2.5.1 dynamic vs. snap shot

A *static* simulation does not need to go into much detail. It doesn’t even need to make any speculation toward the outcome of the fever attack or/and the effectiveness of the treatment. The only thing of interest is how the mothers would decide in this moment in time when facing a particular set of inputs.

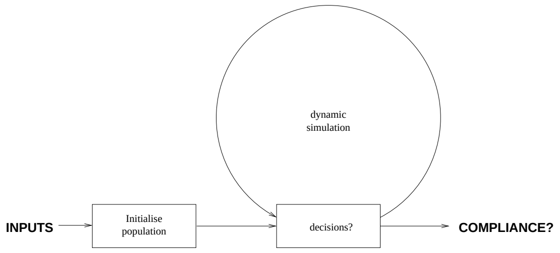


Figure 2.5: Simulation run

In a *dynamic* simulation, on the other hand, each and every fever episode has to be modelled, including its possible fatal outcome. A great deal of variable is needed to keep track of the current state of a household.

In the limited scope of this project, a dynamic implementation is very likely not feasible.

2.5.2 Main control loop

The general behaviour of the system can be defined algorithmically

```
for each village $v$ do
  for each household $h \in v$ do
    no. of incidents ← number of malaria attacks per child × no. of children
    for each incident do
      get environmental and global inputs
      while fever do
        beliefs ← $brf$(inputs)
        plan ← $decide$(beliefs)
        execute plan
      end while
    end for
  end for
end for
```

where the number of incidents directly correlates with the desired timespan (as fraction/multiple of incidents per year).

2.5.3 Variables

The following variables have to be stored for the simulation. The units used are: pc (=per child), pcpy (=per child per year)

$$\textbf{malaria incidents}\ [pcpy] = \text{rain season}^{e3} \times \text{hostile env.}^{e4} \times \text{true incident rate}^{e1}$$

$$\textbf{money}\ [pc] = \frac{\text{no. of adults}^{h2} \times \text{income}^{h4} \times \text{disposable income}^{h5}}{\text{number of children}^{h1}}$$

$$\textbf{child fevers}\ [pcpy] = \textbf{malaria incidents} + \text{false incident rate}^{e2}$$

Chapter 3

Implementation & Testing

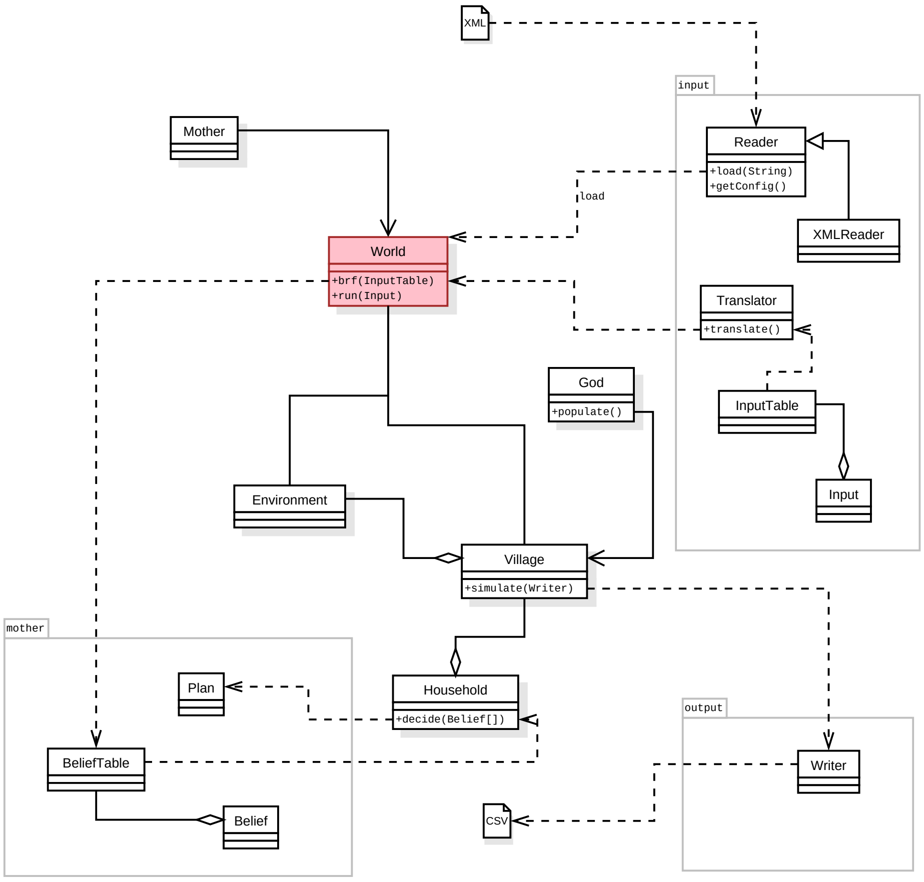


Figure 3.1: Class diagram

3.1 General structure

Given the complexity of the system, it seems almost impossible to write a simulation that includes all desired features. A main class will be implemented that contains all facilities needed (`World` in Figure 3.1). Any desired scenario can be simulated using an instance of this class.

A typical simulation run would take the following form:

```
World world = null;
Input[] config = null;

// load or generate world
...

world.setFilter(some filters);
world.setWriter("data/test");

System.out.print("calculate money");
world.newDataBlock("money");

// increase money in steps of .1 from 0.0 to 10.0
for (double money = 0.0; money < 10.0; money += .1) {

        Input input = new Input(Input.DRUG_COST, money);
        world.run(input);
        System.out.print(".");
}
System.out.println();
```

After initialisation of the world a comparison value (such as `Input.DRUG_COST`) is increased or decreased, and its effect monitored.

3.2 The Entities

Following the system specifications, there are four main entities which have to be taken care of. Apart from their inputs they do not contain any attribute of importance. In some cases attributes are added for matters of convenience.

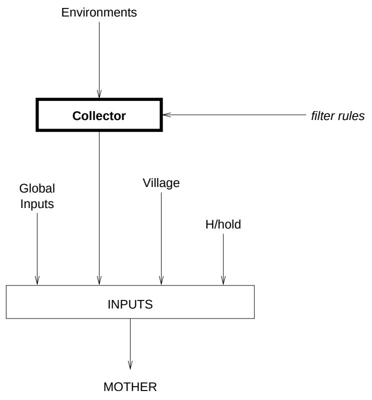
3.3 Input

An input consists of two parts: a *name* and a *value*. The name is a `String` and uniquely identifies the input[1], the value a `double`. At any one time there can only be one value for each input—by default this value is 0.

3.3.1 Input collection

In case one or more inputs clash over two or more environments, they have to be collected (`Collector` class) and filtered (`Filter` class).

[1] In order to ensure integrity only the constant `String` values in class `Input` should be used (e.g. `Input.INCOME`).



```
input table ← {}
for all village $v$ do
    for all environment $e$ such that $v \in e$ do
        inputs[] ← input($e$)
        for all input $i \in inputs[]$ do
            if $i \in$ input table then
                stack ← input table[i.name]
                stack.push(i.value)
            end if
        end for
    end for
end for
```

Figure 3.2: Input collection

3.3.2 Reader

At the start of the simulation there is the possibility of either generating the village data by hand and populating it by using the `God.populate()` method, *or* by loading all data from a file. The abstract class `Reader` was created for that purpose. Its `load()` method extracts all necessary information from a file. At the moment this only available for XML files (`XMLReader` on page 101)—but a reader for any other file format is quite easily implemented.

The following example (from `Mother.java` in Appendix F.6) shows how to load data from a XML file.

```
World world = null;
XMLReader reader = new XMLReader();

try {

       String file = "world_data.xml";

       System.out.println("open " + file + "...");
       world = reader.load(file);
}
catch (IOException io) {
       System.err.println("There was a problem reading the input " +
              "file!\n" + io.getMessage());
       System.exit(-1);
}
catch (ParseException pe) {
       System.err.println("There was a problem while parsing the input " +
              "file!\n" + pe.getMessage());
       System.exit(-2);
}
```

3.4 Output

In order to interpret the data generated by the program, an export mechanism has to be used that can cope with vast amount of information and safely store them away. Possibly the easiest solution is the use of files with Java's

PrintWriter. This way the memory is kept virtually free and data is constantly saved as the simulation goes on without compromising performance.

3.4.1 Data files

The most convenient file format is probably comma-separated values (or short CSV). Most programs including MS Excel and MatLab can read these files and interpret the data within them. The system can easily be adapted to use any other format (like XML) but in the short time available CVS is all it needs.

The test program Mother.java (see p. 127), for example, generates a main file test.csv which contains a summary of all decisions taken in all the villages, and one data file for each village (e.g. test.vil1.csv) which contain the decisions for each village separately.

These files have roughly the following format. Lines starting with # are comments and as such not part of the data. Most plotting programs ignore those lines.

```
# MOTHER 1.0
# data/test (MAIN)
#
# ENVIRONMENTS
# 0: Environment {v1, v2} [true incident rate=2.5, false incident rate=...]
# 1: Environment {v1, v2} [true incident rate=2.0, false incident rate=...]
# 2: Environment {v1, v2} [true incident rate=2.0, false incident rate=...]
#
# VILLAGES (h/holds, adults, kids)
# 0: v1 (3, 6, 19)
# 1: v2 (3, 0, 2)
# 2: v1 (1, 0, 2)
#

#
# money, do nothing, buy anti-pyretic drug, buy part anti-malarial treatment...
0.0, 0, 7, 0, 0, 0, 0, 0
0.1, 0, 7, 0, 0, 0, 0, 0
0.2, 0, 7, 0, 0, 0, 0, 0
0.3, 0, 7, 0, 0, 0, 0, 0
0.4, 0, 7, 0, 0, 0, 0, 0
       .
       .
       .
```

3.4.2 Generating graphs

Using a plotting package like Gnuplot, the CSV data from the file can be used to generate a graph like in Figure 3.3 (see Appendix E for details).

Since in theory the decisions do not change for the same inputs, the graphs will only become interesting when plotted against a changing value, like cost of drugs (red line in Figure 3.3).

3.5 Auxiliary Classes

In the course of the implementation there were a few classes that were not specifically required by the system specifications but were generated to add more convenience and clarity to the code.

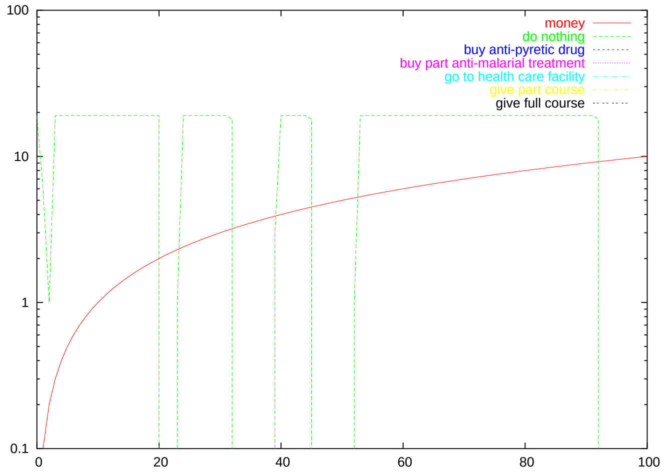


Figure 3.3: Data plot example

3.5.1 InputStack

As described above, the input collection uses a stack to store multiple occurrences of the same input. The `InputStack` works like a normal stack (LIFO) but—for convenience—uses and returns types of `Input`. This saves laborious type casting which would be inevitable when using the built-in `java.util.Stack`.

3.5.2 InputTable & BeliefTable

The most efficient data structure to save beliefs/inputs is a `Hashtable`. It works like a dynamic array but the entries are not indexed by numbers but by `String`s, which saves the laborious effort of going manually through the whole array until finally the requested entry is found.

3.6 Testing

The object-oriented design cycle allows for testing and adjusting single components without having to rewrite the whole system. All components have been tested alone and/or with other components all throughout the implementation process. In fact, most classes in Appendix F contain a `main()` method that was used only for testing purposes.

Chapter 4

Discussion

Although designed to fit any sub-Saharan region in Africa, we will—as far as possible—take inputs based on studies from Kenya. Where this was not possible due to lack of sources, we have substituted, what we think, is a representative average for the missing input. A complete description of the test run can be found in Appendix A.

name	value	references
House hold		
number of children		
number of adults		
traditional beliefs	%	Amin *et al.* (2003) Bob Snow
income	£1.5/day	WHO
disposable income	% (of income)	WHOSIS (2004)
Village		
distance	5 miles	Noor *et al.* (2003)
availability	%	
consultation cost	£2.5	Amin *et al.* (2003)
Environment		
true incident rate	2.5	Sulo *et al.* (2002)
false incident rate	4.5/7	Sulo *et al.* (2002)[a]
rain season	%	
hostile environment	%	
vector control	%	
resistance	%	
qualified advice	%	Zurovac *et al.* (2004)
non-qualified advice	%	Zurovac *et al.* (2004)

Table 4.1: Average data for the test run

[a] This study is based on Kenya. For data for other regions in Africa refer to the following papers: Malawi: Redd *et al.* (1996), Nwanyanwu *et al.* (1997); Tanzania: Shiff, Premji, and Minjas (1993), Shiff, Minjas, and Premji (1994), Rooth and Bjorkman (1992); Uganda: Guthmann *et al.* (2002), Kilian *et al.* (1997); Ethiopia: Muhe *et al.* (1999); Senegal: Gaye *et al.* (1989); Nigeria: Delfini (1973); Gambia: Olaleye *et al.* (1998), Bojang *et al.* (2000); Zimbabwe: Stein and Gelfand (1985), Basset *et al.* (1991).

4.1 The Outcome

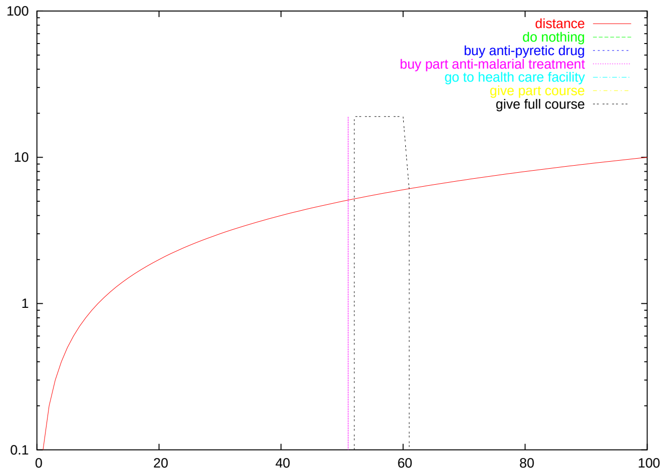


Figure 4.1: Compliance variation with changing distance

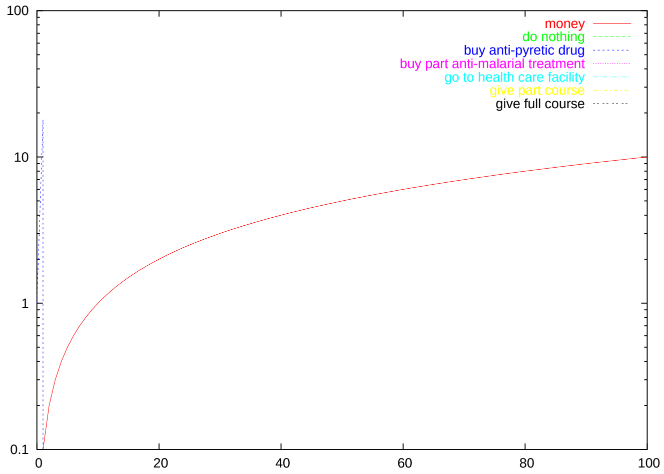


Figure 4.2: Compliance variation with changing drug cost

4.2 Fine Tuning

While the structure of the simulation is general and flexible enough to reflect the situation realistically, there are a few points which provide much more room for investigation but could not be addressed in more detail due to time constraints in this relatively short project.

Although a lot of thought and discussion went into the project, we are more than likely off at some points.

4.2.1 Clash of inputs

The probably most minor point is what happens when a village belongs to two environments with different values for one and the same input? As discussed before, a number of action can be chosen from to derive the new value for the input, which are *minimum*, *maximum*, *average*, *product*, or *sum* of all values. If we have chosen the right one for the job is still guesswork.

4.2.2 Input translation

Especially when it comes to translating the inputs into comparable values it cannot be avoided to generalise. In fact, that is what the translation step is all about.

However, it is almost certain that the translation functions used do not fit a specific region. It is also very likely that the understanding of certain factors change in the near future, all of which would require a complete revision.

4.2.3 Belief Evaluation

To cut a long story short, we do not and cannot know exactly whether the relationships that we have assumed in Figure 2.3 are accurate—or whether they even really exist. The diagram we drew up is based on the input from different members of the group with different expertise. We tried to include only the important factors and omit anything insignificant but it is—after all—still only a sophisticated guess. More field work needs to be done to gain a better insight in this issue[1].

Furthermore, due to before mentioned time constraints and to the fact that the field of QDT is relatively young, the belief evaluation was somewhat simple and not as sophisticated as it could have been. The methodology itself, however, is very powerful and proved to be a good point of start when working with experts from different fields.

4.2.4 Decision making

We generalise all the input factors down to 10 beliefs, based on which the mother is supposed to make a decision. Plausible as they may sound, there is no guarantee that they cover everything needed for the decision or do in fact accurately reflect the real life situation.

Lastly, the provided plans only cover the very basic and simple decisions. It is highly unlikely that they give our virtual mothers enough flexibility or cover all possible actions. However, using the BDI model there is no alternative. Only a restructuring of plans, beliefs and desires could help to reflect real behaviour more closely.

[1] A very interesting thought on this is elaborated in section 5.2.

Chapter 5

Future Developments

During the development of this project quite a few very interesting ideas emerged which, unfortunately, could not be taken on board. These might be relevant in the future to further enhance this work, or taken for themselves, form the basis for new research projects.

5.1 Dynamic Plausability

At the moment plans trigger when certain beliefs are set to **true**, i.e. a fixed, global plausibility level is reached, in our case 70%. In some cases, however, it might not need such a high plausibility. For instance, if the mother considers her child to be very seriously ill even a distance of 10 miles and more might seem like a stone's throw away. Whereas she probably wouldn't bother to walk the same distance for a mild flu.

Modifying the notation from before, this might look like this

$$\begin{bmatrix} ([70\%]\mathsf{serious?} \wedge [30\%]\mathsf{accessible?}) \\ \vee \\ ([30\%]\mathsf{serious?} \wedge [70\%]\mathsf{accessible?}) \end{bmatrix} \rightarrow \mathsf{plan\ x}$$

5.2 Data Mining

Another question is whether the inputs really relate to each other the way it was proposed in this document—and whether the inputs used are relevant at all. There are too many different factors involved to identify them all. However, it is possible for a computer to identify relations between inputs using techniques like *neural networks* or *Bayesian belief networks*.

If the data set is sufficiently big enough (in the region of 10 000 house holds), and enough inputs have been defined that reflect the environment accurately, and—most importantly—have been accurately recorded, then a computer can detect patterns that will probably not be obvious to a human researcher.

5.3 Advice Network

It would also be interesting to see how advice is propagated through social structures within a settlement like in Figure 5.1. The hierarchy would presumably play an important role as the chief's word is more important and influential than that of the village idiot, for instance.

If there is an advice passed on that contradicts another already existing one, which one would prevail and which one would be replaced? If you wanted to convince as many people as possible, where would be the best point of start? Who would have to be convinced first? And how long would it take for all to adopt to the new knowledge?

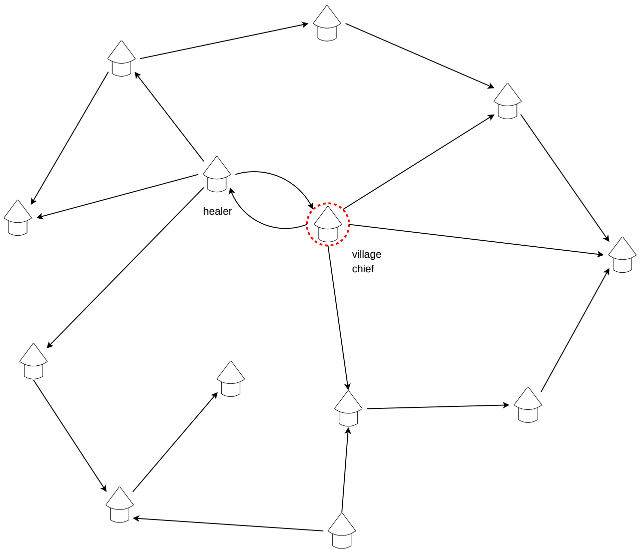


Figure 5.1: Advice network

In order to achieve this a communication interface had to be implemented for each house hold to allow outside influences to change beliefs. Even more than in this project parameters like education, social status, means of communication and behaviour had to be taken into account.

5.4 Social agents

In agent theory, it is quite common to reason about the *motivation* of agents and how this would affect their behaviour—and subsequently their environment. A *selfish* mother would first of all look after her money, whereas a *selfless* would spent all she could on her children.

The plans had to be chosen based on an attributed *utility value*, rather than on picking the highest one possible. This utility value would be governed by how many of the mother's hopes could theoretically be fulfilled.

5.5 Distributed Computing

For simulations in the region of 10 000 villages or more, the work with ordinary personal computers can become very tedious and slow. Unfortunately, so called supercomputers and clusters are highly expensive and not readily available to small research groups. However, there is a technology which allows to distribute a program quite easily over a network using so called *JavaSpaces* which are distributed over a network.

Each JavaSpace could, for example, contain a village whose decisions are computed on the machine the JavaSpace has been created. The results are sent back over the network to the main program that records them as usual. It does not matter for the main program whether the spaces are on the same machine, in the same network or thousands of miles apart on a server in China. Important is only the result of the computation.

Bibliography

Adedoyin, Michael A., and Susan J. Watts. 1989. "Child health and child care in Okelele: An indigenous area of the city of Ilorin, Nigeria." *Social Science & Medicine* 29 (12): 1333–1341.

Alloueche, A, W Bailey, S Barton, J Bwika, P Chimpeni, CO Falade, FA Fehintola, J Horton, S Jaffar, T Kanyok, PG Kremsner, JG Kublin, T Lang, MA Missinou, C Mkandala, AMJ Oduola, Z Premji, and L Robertson. 2004. "Comparison of chlorproguanil-dapsone with sulfadoxine-pyrimethamine for the treatment of uncomplicated falciparum malaria in young African children: double-blind randomised controlled trial." *Lancet* 363 (9424): 1843–8 (June).

Almond, Russel G. 1995. *Graphical Belief Modeling*. London: Chapman & Hall.

Amin, AA, V Marsh, AM Noor, SA Ochola, and RW Snow. 2003. "The use of formal and informal curative services in the management of paediatric fevers in four districts in Kenya." *Tropical Medicine And International Health* 8 (12): 1143–52 (December).

Amin, Abdinasir A., Dyfrig A. Hughes, Vicki Marsh, Timothy O. Abuya, Gilbert O. Kokwaro, Peter A. Winstanley, Sam A. Ochola, and Robert W. Snow. 2004. "The difference between effectiveness and efficacy of anti-malarial drugs in Kenya." *Tropical Medicine And International Health* 9 (9): 1–8 (September).

Basset, *et al.* 1991.

Bennett, F. J., G. A. Saxton, and V. Junod. 1968. "Family structure and health at Kasangati." *Social Science & Medicine* 2 (3): 261–282.

Bojang, KA, S Obaro, LA Morison, and BM Greenwood. 2000. "A prospective evaluation of a clinical algorithm for the diagnosis of malaria in Gambian children." *Tropical Medicine And International Health* 5 (4): 231–6 (April).

Bratman, Michael E. 1999. *Intention, Plans, and Practical Reason*. Stanford: CSLI. Originally published in 1987.

Bratman, Michael E, David J Israel, and Martha E Pollack. 1988. "Plans and Resource-bound Practical Reasoning." *Computational Intelligence* 4 (4): 349–55.

Broersen, Jan, Mahid Dastani, Joris Hulstijn, Zisheng Huang, and Leendert van der Trre. 2001, May. "The BOID Architcture." *AGENTS 01*. Montreal, Quebec, Canada.

Browner, C. H. 1989. "Women, household and health in Latin America." *Social Science & Medicine* 28 (5): 461–473.

Cassen, Robert H. 1976. "Population and development: A survey." *World Development* 4 (10-11): 785–830.

Chickering, David Maxwell, and Judea Pearl. 1997. "A Clicician's Tool for Analyzing Non-compliance." *Computing Science and Statistics* 29 (2): 424–31 (January).

Dastani, Mehdi, Joris Hulstijn, and Leendert van der Torre. 2001. "BDI and QDT: a comparison based on classical decision theory." Edited by Simon Parsons and Piotr Gmytrasiewicz, *Game Theoretic and Decision Theoretic Agents. Papers from 2001 AAAI Spring Symposium*. AAAI.

Dastani, Mehdi, and Leendert van der Torre. 2004. "Decisions, Deliberation, and Agent Types CDT – QDT – BDI – 3APL – BOID." To appear in Focus in Computer Science, Nova Science.

Defo, Barthelemy Kuate. 1996. "Areal and socioeconomic differentials in infant and child mortality in Cameroon." *Social Science & Medicine* 42 (3): 399–420.

Delfini, LF. 1973. "The relationship between body temperature and malaria parasitaemia in rural forest areas of Western Nigeria." *Journal Of Tropical Medicine And Hygiene* 76 (5): 111–4 (May).

Druzdzel, Marek J, and Max Henrion. 1993, July. "Efficient Reasoning in Qualitative Probabilistic Networks." *Proceedings of the 11th Annual Conference on Artificial Intelligence (AAAI 93).* Washington, D.C.

Gaye, O, IB Bah, S Diallo, O Faye, and D Baudon. 1989. "Malaria morbidity in rural and urban areas in Senegal." *Médecine tropicale* 49 (1): 59–62.

Georgeff, Michael, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. 1999, March. "The Belief-Desire-Intention Model of Agency." Edited by J. P. Müller, M. Singh, and A. Rao, *Intelligent Agents V Springer-Verlag Lecture Notes in AI*, Volume 1365.

Guthmann, JP, A Ruiz, G Priotto, James Kiguli, L Bonte, and D Legros. 2002. "Validity, reliability and ease of use in the field of five rapid tests for the diagnosis of Plasmodium falciparum malaria in Uganda." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 96 (3): 254–7.

Guyatt, Helen L, and Robert W Snow. 2004. "The management of fevers in Kenyan children and adults in an area of seasonal malaria transmission." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 98 (2): 111–5 (February).

Halpern, Joseph Y. 1997. "A Logical Approach to Reasoning under Uncertainty: A Tutorial." In *Discourse, Interaction, and Communication*, edited by X. Arrazola, K. Korta, and F. J. Pelletier. Kluwer.

———. 2003. *Reasoning about Uncertainty.* London: MIT Press.

Hampshire, Kate. 2002. "Networks of nomads: negotiating access to health resources among pastoralist women in Chad." *Social Science & Medicine* 54 (7): 1025–1037.

Held, TK, T Weinke, U Mansmann, M Trautmann, and HD Pohle. 1994. "Malaria prophylaxis: identifying risk groups for non-compliance." *Quarterly Journal Of Medicine* 87 (1): 17–22 (January).

Hollos, Marida, and Ulla Larsen. 2004. "Which African men promote smaller families and why? Marital relations and fertility in a Pare community in Northern Tanzania." *Social Science & Medicine* 58 (9): 1733–1749.

Horvitz, EJ, C Ruokangas, S Srinivas, and M Barry. 1992. "A decision-theoretic approach to display of information for time-critical decisions: The Vista project." *Proceedings of the SOAR-92.* NASA/Johnson Space Center, Houston, Texas.

Hughes, Dyfrig A. 2001. "Accounting for noncompliance in pharmacoeconomic evaluations." *Pharmacoeconomics* 19:1185–97.

Jensen, Finn V. 1997. *An Introduction to Bayesian Networks.* New York, NY: Springer.

Jones, Giselle. 2003. "Prescribing and taking medicine." *British Medical Journal* 327 (7419): 819 (October).

Kilian, AH, EB Mughusu, G Kabagambe, and F von Sonnenburg. 1997. "Comparison of two rapid, HRP2-based diagnostic tests for Plasmodium falciparum." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 91 (6): 666–7.

Lucas, P. 2003, October. "Bayesian Network Modelling through Qualitative Patterns." Technical Report, Nijmegen Institute for Computing and Information Sciences.

MacCormack, Carol P. 1988. "Health and the social power of women." *Social Science & Medicine* 26 (7): 677–683.

Marsh, VM, WM Mutemi, J Muturi, A Haaland, WM Watkins, G Otieno, and K Marsh. 1999. "Changing home treatment of childhood fevers by training shop keepers in rural Kenya." *Tropical Medicine And International Health* 4 (5): 383–9 (May).

McBurney, Peter. 1988. "On Transferring Statistical Techniques Accross Cultures: The Kish Grid." *Current Anthropology* 29 (2): 323–325 (April).

Muhe, L, B Oljira, H Degefu, F Enquesellassie, and MW Weber. 1999. "Clinical algorithm for malaria during low and high transmission seasons." *Archives Of Disease In Childhood* 81 (3): 216–20 (September).

Mwenesi, Halima, Trudy Harpham, and Robert W. Snow. 1995. "Child malaria treatment practices among mothers in Kenya." *Social Science & Medicine* 40 (9): 1271–7 (May).

Noor, AM, D Zurovac, SI Hay, SA Ochola, and RW Snow. 2003. "Defining equity in physical access to clinical services using geographical information systems as part of malaria planning and monitoring in Kenya." *Tropical Medicine And International Health* 8 (10): 917–26 (October).

Nwanyanwu, OC, N Kumwenda, PN Kazembe, S Jemu, C Ziba, WC Nkhoma, and SC Redd. 1997. "Malaria and human immunodeficiency virus infection among male employees of a sugar estate in Malawi." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 91 (5): 567–9.

Olaleye, BO, LA Williams, U D'Alessandro, MM Weber, K Mulholland, C Okorie, P Langerock, S Bennett, and BM Greenwood. 1998. "Clinical predictors of malaria in Gambian children with fever or a history of fever." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 92 (3): 300–4.

Parson, Simon. 2001. *Qualitative Methods for reasoning under uncertainty.* London: MIT Press.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Interference.* Revised 2nd printing. San Francisco, CA: Morgan Kaufmann.

Redd, SC, JJ Wirima, RW Steketee, JG Breman, and DL Heymann. 1996. "Transplacental transmission of Plasmodium falciparum in rural Malawi." *American journal of tropical medicine and hygiene* 55 (1 Suppl): 57–60.

Renooij, Silja. 2001, March. "Qualitative Approaches to Quantifying Probabilistic Networks." Ph.D. diss., Faculteit Wiskunde en Informatica, Universeit Utrecht.

Renooij, Silja, Linda C. van der Gaag, and Simon Parson. 2002. "Context-specific Sign-propagation in Qualitative Probabilistic Networks." *Artificial Intelligence* 140:207–230.

Rooth, I, and A Bjorkman. 1992. "Fever episodes in a holoendemic malaria area of Tanzania: parasitological and clinical findings and diagnostic aspects related to malaria." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 86 (5): 479–82.

Seale, Clive. 2000. "Changing patterns of death and dying." *Social Science & Medicine* 51 (6): 917–930.

Shiff, CJ, J Minjas, and Z Premji. 1994. "The ParaSight(R)-F test: A simple rapid manual dipstick test to detect Plasmodium falciparum infection." *Parasitol Today* 10 (12): 494–5.

Shiff, CJ, Z Premji, and JN Minjas. 1993. "The rapid manual ParaSight-F test. A new diagnostic tool for Plasmodium falciparum infection." *Transactions Of The Royal Society Of Tropical Medicine And Hygiene* 87 (6): 646–8.

Snow, Robert W, Erin Eckert, and Awash Teklehaimanot. 2003. "Estimating the needs for artesunate-based combination therapy for malaria case-management in Africa." *Trends in Parasitology* 19 (8): 363–9 (August).

Somasundaram, Daya J., Willem A. C. M. van de Put, Maurice Eisenbruch, and Joop T. V. M. de Jong. 1999. "Starting mental health services in Cambodia." *Social Science & Medicine* 48 (8): 1029–1046.

Stein, CM, and M Gelfand. 1985. "The clinical features and laboratory findings in acute Plasmodium falciparum malaria in Harare, Zimbabwe." *Cent Afr J Med* 31 (9): 166–70 (September).

Sulo, J., P. Chimpeni, J. Hatcher, J. G. Kublin, C. V. Plowe, M. E. Molyneux, K. Marsh, T. E. Taylor, W. M. Watkins, and P. A. Winstanley. 2002. "Chlorproguanil-dapsone versus sulfadoxine-pyrimethamine for sequential episodes of uncomplicated falciparum malaria in Kenya and Malawi: a randomised clinical trial." *Lancet* 360:1136–1143.

Vermeire, E, H Hearnshaw, P Van Royen, and J Denekens. 2001. "Patient adherence to treatment: three decades of research. A comprehensive review." *Journal Of Clinical Pharmacy And Therapeutics* 26 (5): 331–42 (October).

Wellman, Michael P. 1990. "Fundamental Concepts of Qualitative Probabilistic Networks." *Artificial Intelligence* 44 (3): 257–303.

WHO. 2003. "Adherence to long-trem therapies: Evidence for action." Geneva.

WHOSIS. 2004. WHO Statistical Information System. `http://www3.who.int/whosis/country/compare.cfm?lang%uage=english\&country=ken` and `http://www3.who.int/whosis/country/indicators.cfm?%country=ken`.

Winstanley, P. 2001. "Chlorproguanil-dapsone (LAPDAP) for uncomplicated falciparum malaria." *Tropical Medicine And International Health* 6 (11): 952–4 (November).

Wooldridge, Michael. 1996a, January 15-17th. "A Logic of BDI Agents with Procedural Knowledge." Edited by J. L. Fiadeiro and P.-Y. Schobbens, *Proceedings of the Second Workshop of the MODELAGE Project*. Sesimbra, Portugal.

———. 1996b, June. "Practical Reasoning with Procedural Knowledge: A Logic of BDI Agents with Know-How." Edited by D. M. Gabbay and H.-J. Ohlbach, *Proceedings of the International Conference on Formal and Applied Practical Reasoning*.

———. 2002. *An Introduction to MultiAgent Systems*. Wiley.

Zurovac, D., A. K. Rowe, S. A. Ochola, A. M. Noor, B. Midia, M. English, and R. W. Snow. 2004. "Predictors of the quality of health worker treatment practices for uncomplicated malaria at government health facilities in Kenya." *International Journal for Epidemiolgy(?)*, July.

Appendix A

Test Scenario

A.1 XML feed

```
<!--
==========================================================
  MOTHER 1.0
  TEST SCENARIO
  $Id: test_scenario.xml 55 2004-10-07 17:52:53Z Basti $
==========================================================
-->

<simulation>

    <villages>

        <!-- -->
        <village id = "v1"
                distance = "10.0"
                availability = "2.5"
                consultation = "10" >

            <household children = "7"
                      adults = "2"
                      tradition = "0"
                      income = "20"
                      disposable = ".3" />

            <household children = "5"
                      adults = "3"
                      tradition = ".4"
                      income = "10"
                      disposable = ".5" />

            <household children = "7"
                      adults = "1"
                      tradition = "1.0"
                      income = "2"
                      disposable = ".25" />

        <household children = "7"
                  adults = "2"
                  tradition = "0"
                  income = "20"
                  disposable = ".3" />

        <household children = "5"
                  adults = "3"
                  tradition = ".4"
                  income = "10"
                  disposable = ".5" />

        <household children = "7"
                  adults = "1"
                  tradition = "1.0"
                  income = "2"
                  disposable = ".25" />

    </village>

    <!-- -->
    <village id = "v2"
            distance = "5"
            availability = ".32"
            consultation = "12" >

        <household children = "7"
                  adults = "2"
                  tradition = "0"
                  income = "20"
                  disposable = ".3" />

        <household children = "5"
                  adults = "3"
                  tradition = ".4"
                  income = "10"
                  disposable = ".5" />

        <household children = "7"
                  adults = "1"
                  tradition = "1.0"
                  income = "2"
                  disposable = ".25" />

        <household children = "7"
                  adults = "2"
                  tradition = "0"
                  income = "20"
                  disposable = ".3" />

        <household children = "5"
                  adults = "3"
                  tradition = ".4"
                  income = "10"
                  disposable = ".5" />

        <household children = "7"
                  adults = "1"
                  tradition = "1.0"
                  income = "2"
                  disposable = ".25" />

        <household children = "7"
                  adults = "2"
                  tradition = "0"
                  income = "20"
                  disposable = ".3" />

        <household children = "5"
                  adults = "3"
                  tradition = ".4"
                  income = "10"
                  disposable = ".5" />

        <household children = "7"
                  adults = "1"
                  tradition = "1.0"
                  income = "2"
                  disposable = ".25" />

        <household children = "7"
                  adults = "2"
                  tradition = "0"
                  income = "20"
                  disposable = ".3" />

        <household children = "5"
                  adults = "3"
                  tradition = ".4"
                  income = "10"
                  disposable = ".5" />

        <household children = "7"
                  adults = "1"
                  tradition = "1.0"
                  income = "2"
                  disposable = ".25" />


    </village>

    <!-- -->
    <village id = "v1"
            distance = "10.0"
            availability = "2.5"
            consultation = "10">

```

```
       <household children = "2"
                 adults = ""
                 tradition = ""
                 income = ""
                 disposable = "" />

    </village>

</villages>

<environments>

    <environment id = "e0"
               true_incidents = "2.5"
               false_incidents = "3"
               rain_season = "2"
               hostile_env = "34"
               vector_control = "6"
               resistance = "4"
               good_advice = "8"
               bad_advice = "0">
       v1, v2
    </environment>

    <environment id = "e1"
               true_incidents = "12"
               false_incidents = "32"
               rain_season = "43"
               hostile_env = "54"
               vector_control = "65"
               resistance = "76"
               good_advice = "87"
               bad_advice = "98">
       v1, v2
    </environment>

    <environment id = "e2"
               true_incidents = "12"
               false_incidents = "321"
               rain_season = "43"
               hostile_env = "54"
               vector_control = "65"
               resistance = "76"
               good_advice = "xp"
               bad_advice = "98">
       v1, v2
    </environment>

</environments>

<drug name = "LAPDAP"
     cost = "1.5"
     efficacy = ".96"
     side_effects = ".46"
```

```
         forgiveness = ".66"
         complexity = ".2"
         ap_effect = "0"
         batch_failure = "0" />

</simulation>
```

Appendix B

Data Sheets

B.1 LAPDAP

We use LAPDAP, a drug developed at LSTM Winstanley (2001).

name	value	references
Drug cost	100KES (= about US$1.50)	Amin *et al.* (2003)
Efficacy	96%	Alloueche *et al.* (2004)
Side effects	46%	Alloueche *et al.* (2004)
Forgiveness	66%[a]	[guess]
Complexity	typical	
Antipyretic effect	0	
Batch failures	0[b]	Amin *et al.* (2004)

[a] i.e. miss one days' worth of doses (out of 3) and you loose clinical efficacy

[b] now, in a few years' time 50%

Appendix C

A (Very) Short Introduction to Logic

In *propositional logic* a statement (or proposition) like ‘It is raining’ can either be **true** or **false**. Using the signs below, we can combine statements which will themselves evaluate again to **true** or **false**.

$\neg$	not	e.g. $\neg\mathsf{raining} =$*‘it is* **not** *raining’*
$\wedge$	and	e.g. $\mathsf{raining} \wedge \mathsf{monday} =$*‘it is raining* **and** *it is Monday’*
$\vee$	or	e.g. $\mathsf{raining} \vee \mathsf{sun} =$*‘it is raining* **or** *the sun shines’*

This allows for quite sophisticated statements

$$(\mathsf{raining} \wedge \mathsf{umbrella} \wedge \neg\mathsf{wet}) \vee [(\mathsf{sun} \vee \neg\mathsf{raining}) \wedge \neg\mathsf{wet}]$$

It is raining and I have an umbrella and don’t get wet, **or** *either the sun shines or it is not raining and I don’t get wet.*

While combining statements the following rules apply, where T stands for **true** and F for **false**

$\neg$	
T	F
F	T

$\vee$	T	F
T	T	T
F	T	F

$\wedge$	T	F
T	T	F
F	F	F

Appendix D

The Mother's Decisions

With 10 statements there are $2^{10} = 1024$ different combinations, each triggering a plan. They are listed in D.1 and ordered by plans in D.2[1]. The bullet (•) means that the respective belief is **true**.

Please note that whenever `enough` is **true**, `drug` has to be **true** as well.

[1] The following lists were automatically created using `PlanTable.java` (see page 131).

D.1 All possiblities

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
										do nothing
									●	do nothing
								●		do nothing
								●	●	do nothing
							●			do nothing
							●		●	do nothing
							●	●		do nothing
							●	●	●	do nothing
						●				do nothing
						●			●	do nothing
						●		●		do nothing
						●		●	●	do nothing
						●	●			do nothing
						●	●		●	do nothing
						●	●	●		do nothing
						●	●	●	●	do nothing
					●					do nothing
					●				●	do nothing
					●			●		do nothing
					●			●	●	do nothing
					●		●			do nothing
					●		●		●	do nothing
					●		●	●		do nothing
					●		●	●	●	do nothing
					●	●				do nothing
					●	●			●	do nothing
					●	●		●		do nothing
					●	●		●	●	do nothing
					●	●	●			do nothing
					●	●	●		●	do nothing
					●	●	●	●		do nothing
					●	●	●	●	●	do nothing
				●						do nothing
				●					●	do nothing
				●				●		do nothing
				●				●	●	do nothing
				●			●			do nothing
				●			●		●	do nothing
				●			●	●		do nothing
				●			●	●	●	do nothing
				●		●				do nothing
				●		●			●	do nothing
				●		●		●		do nothing
				●		●		●	●	do nothing
				●		●	●			do nothing
				●		●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
				●		●	●	●		do nothing
				●		●	●	●	●	do nothing
				●	●					do nothing
				●	●				●	do nothing
				●	●			●		do nothing
				●	●			●	●	do nothing
				●	●		●			do nothing
				●	●		●		●	do nothing
				●	●		●	●		do nothing
				●	●		●	●	●	do nothing
				●	●	●				do nothing
				●	●	●			●	do nothing
				●	●	●		●		do nothing
				●	●	●		●	●	do nothing
				●	●	●	●			do nothing
				●	●	●	●		●	do nothing
				●	●	●	●	●		do nothing
				●	●	●	●	●	●	do nothing
			●							do nothing
			●						●	do nothing
			●					●		do nothing
			●					●	●	do nothing
			●				●			do nothing
			●				●		●	do nothing
			●				●	●		do nothing
			●				●	●	●	do nothing
			●			●				do nothing
			●			●			●	do nothing
			●			●		●		do nothing
			●			●		●	●	do nothing
			●			●	●			do nothing
			●			●	●		●	do nothing
			●			●	●	●		do nothing
			●			●	●	●	●	do nothing
			●		●					do nothing
			●		●				●	do nothing
			●		●			●		do nothing
			●		●			●	●	do nothing
			●		●		●			do nothing
			●		●		●		●	do nothing
			●		●		●	●		do nothing
			●		●		●	●	●	do nothing
			●		●	●				do nothing
			●		●	●			●	do nothing
			●		●	●		●		do nothing
			●		●	●		●	●	do nothing
			●		●	●	●			do nothing
			●		●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
			•		•	•	•	•		do nothing
			•		•	•	•	•	•	do nothing
			•	•						do nothing
			•	•					•	do nothing
			•	•				•		do nothing
			•	•				•	•	do nothing
			•	•			•			do nothing
			•	•			•		•	do nothing
			•	•			•	•		do nothing
			•	•			•	•	•	do nothing
			•	•		•				do nothing
			•	•		•			•	do nothing
			•	•		•		•		do nothing
			•	•		•		•	•	do nothing
			•	•		•	•			do nothing
			•	•		•	•		•	do nothing
			•	•		•	•	•		do nothing
			•	•		•	•	•	•	do nothing
			•	•	•					do nothing
			•	•	•				•	do nothing
			•	•	•			•		do nothing
			•	•	•			•	•	do nothing
			•	•	•		•			do nothing
			•	•	•		•		•	do nothing
			•	•	•		•	•		do nothing
			•	•	•		•	•	•	do nothing
			•	•	•	•				do nothing
			•	•	•	•			•	do nothing
			•	•	•	•		•		do nothing
			•	•	•	•		•	•	do nothing
			•	•	•	•	•			do nothing
			•	•	•	•	•		•	do nothing
			•	•	•	•	•	•		do nothing
			•	•	•	•	•	•	•	do nothing
	•	•								do nothing
	•	•							•	do nothing
	•	•						•		do nothing
	•	•						•	•	do nothing
	•	•					•			do nothing
	•	•					•		•	do nothing
	•	•					•	•		do nothing
	•	•					•	•	•	do nothing
	•	•				•				do nothing
	•	•				•			•	do nothing
	•	•				•		•		do nothing
	•	•				•		•	•	do nothing
	•	•				•	•			do nothing
	•	•				•	•		•	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●	●				●	●	●		do nothing
	●	●				●	●	●	●	do nothing
	●	●			●					do nothing
	●	●			●				●	do nothing
	●	●			●			●		do nothing
	●	●			●			●	●	do nothing
	●	●			●		●			do nothing
	●	●			●		●		●	do nothing
	●	●			●		●	●		do nothing
	●	●			●		●	●	●	do nothing
	●	●			●	●				do nothing
	●	●			●	●			●	do nothing
	●	●			●	●		●		do nothing
	●	●			●	●		●	●	do nothing
	●	●			●	●	●			do nothing
	●	●			●	●	●		●	do nothing
	●	●			●	●	●	●		do nothing
	●	●			●	●	●	●	●	do nothing
	●	●		●						do nothing
	●	●		●					●	do nothing
	●	●		●				●		do nothing
	●	●		●				●	●	do nothing
	●	●		●			●			do nothing
	●	●		●			●		●	do nothing
	●	●		●			●	●		do nothing
	●	●		●			●	●	●	do nothing
	●	●		●		●				do nothing
	●	●		●		●			●	do nothing
	●	●		●		●		●		do nothing
	●	●		●		●		●	●	do nothing
	●	●		●		●	●			do nothing
	●	●		●		●	●		●	do nothing
	●	●		●		●	●	●		do nothing
	●	●		●		●	●	●	●	do nothing
	●	●		●	●					do nothing
	●	●		●	●				●	do nothing
	●	●		●	●			●		do nothing
	●	●		●	●			●	●	do nothing
	●	●		●	●		●			do nothing
	●	●		●	●		●		●	do nothing
	●	●		●	●		●	●		do nothing
	●	●		●	●		●	●	●	do nothing
	●	●		●	●	●				do nothing
	●	●		●	●	●			●	do nothing
	●	●		●	●	●		●		do nothing
	●	●		●	●	●		●	●	do nothing
	●	●		●	●	●	●			do nothing
	●	●		●	●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●	●		●	●	●	●	●		do nothing
	●	●		●	●	●	●	●	●	do nothing
	●	●	●							do nothing
	●	●	●						●	do nothing
	●	●	●					●		do nothing
	●	●	●					●	●	do nothing
	●	●	●				●			do nothing
	●	●	●				●		●	do nothing
	●	●	●				●	●		do nothing
	●	●	●				●	●	●	do nothing
	●	●	●			●				do nothing
	●	●	●			●			●	do nothing
	●	●	●			●		●		do nothing
	●	●	●			●		●	●	do nothing
	●	●	●			●	●			do nothing
	●	●	●			●	●		●	do nothing
	●	●	●			●	●	●		do nothing
	●	●	●			●	●	●	●	do nothing
	●	●	●		●					do nothing
	●	●	●		●				●	do nothing
	●	●	●		●			●		do nothing
	●	●	●		●			●	●	do nothing
	●	●	●		●		●			do nothing
	●	●	●		●		●		●	do nothing
	●	●	●		●		●	●		do nothing
	●	●	●		●		●	●	●	do nothing
	●	●	●		●	●				do nothing
	●	●	●		●	●			●	do nothing
	●	●	●		●	●		●		do nothing
	●	●	●		●	●		●	●	do nothing
	●	●	●		●	●	●			do nothing
	●	●	●		●	●	●		●	do nothing
	●	●	●		●	●	●	●		do nothing
	●	●	●		●	●	●	●	●	do nothing
	●	●	●	●						do nothing
	●	●	●	●					●	do nothing
	●	●	●	●				●		do nothing
	●	●	●	●				●	●	do nothing
	●	●	●	●			●			do nothing
	●	●	●	●			●		●	do nothing
	●	●	●	●			●	●		do nothing
	●	●	●	●			●	●	●	do nothing
	●	●	●	●		●				do nothing
	●	●	●	●		●			●	do nothing
	●	●	●	●		●		●		do nothing
	●	●	●	●		●		●	●	do nothing
	●	●	●	●		●	●			do nothing
	●	●	●	●		●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●	●	●	●		●	●	●		do nothing
	●	●	●	●		●	●	●	●	do nothing
	●	●	●	●	●					do nothing
	●	●	●	●	●				●	do nothing
	●	●	●	●	●			●		do nothing
	●	●	●	●	●			●	●	do nothing
	●	●	●	●	●		●			do nothing
	●	●	●	●	●		●		●	do nothing
	●	●	●	●	●		●	●		do nothing
	●	●	●	●	●		●	●	●	do nothing
	●	●	●	●	●	●				do nothing
	●	●	●	●	●	●			●	do nothing
	●	●	●	●	●	●		●		do nothing
	●	●	●	●	●	●		●	●	do nothing
	●	●	●	●	●	●	●			do nothing
	●	●	●	●	●	●	●		●	do nothing
	●	●	●	●	●	●	●	●		do nothing
	●	●	●	●	●	●	●	●	●	do nothing
	●									do nothing
	●								●	do nothing
	●							●		do nothing
	●							●	●	do nothing
	●						●			do nothing
	●						●		●	do nothing
	●						●	●		do nothing
	●						●	●	●	do nothing
	●					●				do nothing
	●					●			●	do nothing
	●					●		●		do nothing
	●					●		●	●	do nothing
	●					●	●			do nothing
	●					●	●		●	do nothing
	●					●	●	●		do nothing
	●					●	●	●	●	do nothing
	●				●					do nothing
	●				●				●	do nothing
	●				●			●		do nothing
	●				●			●	●	do nothing
	●				●		●			do nothing
	●				●		●		●	do nothing
	●				●		●	●		do nothing
	●				●		●	●	●	do nothing
	●				●	●				do nothing
	●				●	●			●	do nothing
	●				●	●		●		do nothing
	●				●	●		●	●	do nothing
	●				●	●	●			do nothing
	●				●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●				●	●	●	●		do nothing
	●				●	●	●	●	●	do nothing
	●			●						do nothing
	●			●					●	do nothing
	●			●				●		do nothing
	●			●				●	●	do nothing
	●			●			●			do nothing
	●			●			●		●	do nothing
	●			●			●	●		do nothing
	●			●			●	●	●	do nothing
	●			●		●				do nothing
	●			●		●			●	do nothing
	●			●		●		●		do nothing
	●			●		●		●	●	do nothing
	●			●		●	●			do nothing
	●			●		●	●		●	do nothing
	●			●		●	●	●		do nothing
	●			●		●	●	●	●	do nothing
	●			●	●					do nothing
	●			●	●				●	do nothing
	●			●	●			●		do nothing
	●			●	●			●	●	do nothing
	●			●	●		●			do nothing
	●			●	●		●		●	do nothing
	●			●	●		●	●		do nothing
	●			●	●		●	●	●	do nothing
	●			●	●	●				do nothing
	●			●	●	●			●	do nothing
	●			●	●	●		●		do nothing
	●			●	●	●		●	●	do nothing
	●			●	●	●	●			do nothing
	●			●	●	●	●		●	do nothing
	●			●	●	●	●	●		do nothing
	●			●	●	●	●	●	●	do nothing
	●		●							do nothing
	●		●						●	do nothing
	●		●					●		do nothing
	●		●					●	●	do nothing
	●		●				●			do nothing
	●		●				●		●	do nothing
	●		●				●	●		do nothing
	●		●				●	●	●	do nothing
	●		●			●				do nothing
	●		●			●			●	do nothing
	●		●			●		●		do nothing
	●		●			●		●	●	do nothing
	●		●			●	●			do nothing
	●		●			●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	•		•			•	•	•		do nothing
	•		•			•	•	•	•	do nothing
	•		•		•					do nothing
	•		•		•				•	do nothing
	•		•		•			•		do nothing
	•		•		•			•	•	do nothing
	•		•		•		•			do nothing
	•		•		•		•		•	do nothing
	•		•		•		•	•		do nothing
	•		•		•		•	•	•	do nothing
	•		•		•	•				do nothing
	•		•		•	•			•	do nothing
	•		•		•	•		•		do nothing
	•		•		•	•		•	•	do nothing
	•		•		•	•	•			do nothing
	•		•		•	•	•		•	do nothing
	•		•		•	•	•	•		do nothing
	•		•		•	•	•	•	•	do nothing
	•		•	•						do nothing
	•		•	•					•	do nothing
	•		•	•				•		do nothing
	•		•	•				•	•	do nothing
	•		•	•			•			do nothing
	•		•	•			•		•	do nothing
	•		•	•			•	•		do nothing
	•		•	•			•	•	•	do nothing
	•		•	•		•				do nothing
	•		•	•		•			•	do nothing
	•		•	•		•		•		do nothing
	•		•	•		•		•	•	do nothing
	•		•	•		•	•			do nothing
	•		•	•		•	•		•	do nothing
	•		•	•		•	•	•		do nothing
	•		•	•		•	•	•	•	do nothing
	•		•	•	•					do nothing
	•		•	•	•				•	do nothing
	•		•	•	•			•		do nothing
	•		•	•	•			•	•	do nothing
	•		•	•	•		•			do nothing
	•		•	•	•		•		•	do nothing
	•		•	•	•		•	•		do nothing
	•		•	•	•		•	•	•	do nothing
	•		•	•	•	•				do nothing
	•		•	•	•	•			•	do nothing
	•		•	•	•	•		•		do nothing
	•		•	•	•	•		•	•	do nothing
	•		•	•	•	•	•			do nothing
	•		•	•	•	•	•		•	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●		●	●	●	●	●	●		do nothing
	●		●	●	●	●	●	●	●	do nothing
	●	●								do nothing
	●	●							●	do nothing
	●	●						●		do nothing
	●	●						●	●	do nothing
	●	●					●			do nothing
	●	●					●		●	do nothing
	●	●					●	●		do nothing
	●	●					●	●	●	do nothing
	●	●				●				do nothing
	●	●				●			●	do nothing
	●	●				●		●		do nothing
	●	●				●		●	●	do nothing
	●	●				●	●			do nothing
	●	●				●	●		●	do nothing
	●	●				●	●	●		do nothing
	●	●				●	●	●	●	do nothing
	●	●			●					do nothing
	●	●			●				●	do nothing
	●	●			●			●		do nothing
	●	●			●			●	●	do nothing
	●	●			●		●			do nothing
	●	●			●		●		●	do nothing
	●	●			●		●	●		do nothing
	●	●			●		●	●	●	do nothing
	●	●			●	●				do nothing
	●	●			●	●			●	do nothing
	●	●			●	●		●		do nothing
	●	●			●	●		●	●	do nothing
	●	●			●	●	●			do nothing
	●	●			●	●	●		●	do nothing
	●	●			●	●	●	●		do nothing
	●	●			●	●	●	●	●	do nothing
	●	●		●						do nothing
	●	●		●					●	do nothing
	●	●		●				●		do nothing
	●	●		●				●	●	do nothing
	●	●		●			●			do nothing
	●	●		●			●		●	do nothing
	●	●		●			●	●		do nothing
	●	●		●			●	●	●	do nothing
	●	●		●		●				do nothing
	●	●		●		●			●	do nothing
	●	●		●		●		●		do nothing
	●	●		●		●		●	●	do nothing
	●	●		●		●	●			do nothing
	●	●		●		●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●	●		●		●	●	●		do nothing
	●	●		●		●	●	●	●	do nothing
	●	●		●	●					do nothing
	●	●		●	●				●	do nothing
	●	●		●	●			●		do nothing
	●	●		●	●			●	●	do nothing
	●	●		●	●		●			do nothing
	●	●		●	●		●		●	do nothing
	●	●		●	●		●	●		do nothing
	●	●		●	●		●	●	●	do nothing
	●	●		●	●	●				do nothing
	●	●		●	●	●			●	do nothing
	●	●		●	●	●		●		do nothing
	●	●		●	●	●		●	●	do nothing
	●	●		●	●	●	●			do nothing
	●	●		●	●	●	●		●	do nothing
	●	●		●	●	●	●	●		do nothing
	●	●		●	●	●	●	●	●	do nothing
	●	●	●							do nothing
	●	●	●						●	do nothing
	●	●	●					●		do nothing
	●	●	●					●	●	do nothing
	●	●	●				●			do nothing
	●	●	●				●		●	do nothing
	●	●	●				●	●		do nothing
	●	●	●				●	●	●	do nothing
	●	●	●			●				do nothing
	●	●	●			●			●	do nothing
	●	●	●			●		●		do nothing
	●	●	●			●		●	●	do nothing
	●	●	●			●	●			do nothing
	●	●	●			●	●		●	do nothing
	●	●	●			●	●	●		do nothing
	●	●	●			●	●	●	●	do nothing
	●	●	●		●					do nothing
	●	●	●		●				●	do nothing
	●	●	●		●			●		do nothing
	●	●	●		●			●	●	do nothing
	●	●	●		●		●			do nothing
	●	●	●		●		●		●	do nothing
	●	●	●		●		●	●		do nothing
	●	●	●		●		●	●	●	do nothing
	●	●	●		●	●				do nothing
	●	●	●		●	●			●	do nothing
	●	●	●		●	●		●		do nothing
	●	●	●		●	●		●	●	do nothing
	●	●	●		●	●	●			do nothing
	●	●	●		●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
	●	●	●		●	●	●	●		do nothing
	●	●	●		●	●	●	●	●	do nothing
	●	●	●	●						do nothing
	●	●	●	●					●	do nothing
	●	●	●	●				●		do nothing
	●	●	●	●				●	●	do nothing
	●	●	●	●			●			do nothing
	●	●	●	●			●		●	do nothing
	●	●	●	●			●	●		do nothing
	●	●	●	●			●	●	●	do nothing
	●	●	●	●		●				do nothing
	●	●	●	●		●			●	do nothing
	●	●	●	●		●		●		do nothing
	●	●	●	●		●		●	●	do nothing
	●	●	●	●		●	●			do nothing
	●	●	●	●		●	●		●	do nothing
	●	●	●	●		●	●	●		do nothing
	●	●	●	●		●	●	●	●	do nothing
	●	●	●	●	●					do nothing
	●	●	●	●	●				●	do nothing
	●	●	●	●	●			●		do nothing
	●	●	●	●	●			●	●	do nothing
	●	●	●	●	●		●			do nothing
	●	●	●	●	●		●		●	do nothing
	●	●	●	●	●		●	●		do nothing
	●	●	●	●	●		●	●	●	do nothing
	●	●	●	●	●	●				do nothing
	●	●	●	●	●	●			●	do nothing
	●	●	●	●	●	●		●		do nothing
	●	●	●	●	●	●		●	●	do nothing
	●	●	●	●	●	●	●			do nothing
	●	●	●	●	●	●	●		●	do nothing
	●	●	●	●	●	●	●	●		do nothing
	●	●	●	●	●	●	●	●	●	do nothing
●										buy anti-pyretic drug
●									●	do nothing
●								●		buy anti-pyretic drug
●								●	●	do nothing
●							●			buy anti-pyretic drug
●							●		●	do nothing
●							●	●		buy anti-pyretic drug
●							●	●	●	do nothing
●						●				buy anti-pyretic drug
●						●			●	do nothing
●						●		●		buy anti-pyretic drug
●						●		●	●	do nothing
●						●	●			buy anti-pyretic drug
●						●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●						●	●	●		buy anti-pyretic drug
●						●	●	●	●	do nothing
●					●					buy anti-pyretic drug
●					●				●	do nothing
●					●			●		buy anti-pyretic drug
●					●			●	●	do nothing
●					●		●			buy anti-pyretic drug
●					●		●		●	do nothing
●					●		●	●		buy anti-pyretic drug
●					●		●	●	●	do nothing
●					●	●				do nothing
●					●	●			●	do nothing
●					●	●		●		do nothing
●					●	●		●	●	do nothing
●					●	●	●			do nothing
●					●	●	●		●	do nothing
●					●	●	●	●		do nothing
●					●	●	●	●	●	do nothing
●				●						do nothing
●				●					●	do nothing
●				●				●		buy part anti-malarial treatment
●				●				●	●	do nothing
●				●			●			do nothing
●				●			●		●	do nothing
●				●			●	●		buy full anti-malarial treatment
●				●			●	●	●	do nothing
●				●		●				do nothing
●				●		●			●	do nothing
●				●		●		●		buy part anti-malarial treatment
●				●		●		●	●	do nothing
●				●		●	●			do nothing
●				●		●	●		●	do nothing
●				●		●	●	●		buy full anti-malarial treatment
●				●		●	●	●	●	do nothing
●				●	●					do nothing
●				●	●				●	do nothing
●				●	●			●		do nothing
●				●	●			●	●	do nothing
●				●	●		●			buy full anti-malarial treatment
●				●	●		●		●	do nothing
●				●	●		●	●		buy full anti-malarial treatment
●				●	●		●	●	●	do nothing
●				●	●	●				do nothing
●				●	●	●			●	do nothing
●				●	●	●		●		do nothing
●				●	●	●		●	●	do nothing
●				●	●	●	●			buy full anti-malarial treatment
●				●	●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●				●	●	●	●	●		go to health care facility
●				●	●	●	●	●	●	do nothing
●			●							buy anti-pyretic drug
●			●						●	do nothing
●			●					●		buy anti-pyretic drug
●			●					●	●	do nothing
●			●				●			buy anti-pyretic drug
●			●				●		●	do nothing
●			●				●	●		buy anti-pyretic drug
●			●				●	●	●	do nothing
●			●			●				buy anti-pyretic drug
●			●			●			●	do nothing
●			●			●		●		buy anti-pyretic drug
●			●			●		●	●	do nothing
●			●			●	●			buy anti-pyretic drug
●			●			●	●		●	do nothing
●			●			●	●	●		buy anti-pyretic drug
●			●			●	●	●	●	do nothing
●			●		●					buy anti-pyretic drug
●			●		●				●	do nothing
●			●		●			●		buy anti-pyretic drug
●			●		●			●	●	do nothing
●			●		●		●			buy anti-pyretic drug
●			●		●		●		●	do nothing
●			●		●		●	●		buy anti-pyretic drug
●			●		●		●	●	●	do nothing
●			●		●	●				do nothing
●			●		●	●			●	do nothing
●			●		●	●		●		do nothing
●			●		●	●		●	●	do nothing
●			●		●	●	●			do nothing
●			●		●	●	●		●	do nothing
●			●		●	●	●	●		do nothing
●			●		●	●	●	●	●	do nothing
●			●	●						do nothing
●			●	●					●	do nothing
●			●	●				●		buy part anti-malarial treatment
●			●	●				●	●	do nothing
●			●	●			●			do nothing
●			●	●			●		●	do nothing
●			●	●			●	●		buy full anti-malarial treatment
●			●	●			●	●	●	do nothing
●			●	●		●				do nothing
●			●	●		●			●	do nothing
●			●	●		●		●		buy part anti-malarial treatment
●			●	●		●		●	●	do nothing
●			●	●		●	●			do nothing
●			●	●		●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●			●	●		●	●	●		buy full anti-malarial treatment
●			●	●		●	●	●	●	do nothing
●			●	●	●					do nothing
●			●	●	●				●	do nothing
●			●	●	●			●		do nothing
●			●	●	●			●	●	do nothing
●			●	●	●		●			buy full anti-malarial treatment
●			●	●	●		●		●	do nothing
●			●	●	●		●	●		buy full anti-malarial treatment
●			●	●	●		●	●	●	do nothing
●			●	●	●	●				do nothing
●			●	●	●	●			●	do nothing
●			●	●	●	●		●		do nothing
●			●	●	●	●		●	●	do nothing
●			●	●	●	●	●			buy full anti-malarial treatment
●			●	●	●	●	●		●	do nothing
●			●	●	●	●	●	●		go to health care facility
●			●	●	●	●	●	●	●	do nothing
●	●	●								give part course
●	●	●							●	do nothing
●	●	●						●		give full course
●	●	●						●	●	do nothing
●	●	●					●			give part course
●	●	●					●		●	do nothing
●	●	●					●	●		give full course
●	●	●					●	●	●	do nothing
●	●	●				●				give part course
●	●	●				●			●	do nothing
●	●	●				●		●		give full course
●	●	●				●		●	●	do nothing
●	●	●				●	●			give part course
●	●	●				●	●		●	do nothing
●	●	●				●	●	●		give full course
●	●	●				●	●	●	●	do nothing
●	●	●			●					give part course
●	●	●			●				●	do nothing
●	●	●			●			●		give full course
●	●	●			●			●	●	do nothing
●	●	●			●		●			give part course
●	●	●			●		●		●	do nothing
●	●	●			●		●	●		give full course
●	●	●			●		●	●	●	do nothing
●	●	●			●	●				give part course
●	●	●			●	●			●	do nothing
●	●	●			●	●		●		give full course
●	●	●			●	●		●	●	do nothing
●	●	●			●	●	●			give part course
●	●	●			●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●	●	●			●	●	●	●		give full course
●	●	●			●	●	●	●	●	do nothing
●	●	●		●						do nothing
●	●	●		●					●	do nothing
●	●	●		●				●		give full course
●	●	●		●				●	●	do nothing
●	●	●		●			●			do nothing
●	●	●		●			●		●	do nothing
●	●	●		●			●	●		give full course
●	●	●		●			●	●	●	do nothing
●	●	●		●		●				do nothing
●	●	●		●		●			●	do nothing
●	●	●		●		●		●		give full course
●	●	●		●		●		●	●	do nothing
●	●	●		●		●	●			do nothing
●	●	●		●		●	●		●	do nothing
●	●	●		●		●	●	●		give full course
●	●	●		●		●	●	●	●	do nothing
●	●	●		●	●					do nothing
●	●	●		●	●				●	do nothing
●	●	●		●	●			●		give full course
●	●	●		●	●			●	●	do nothing
●	●	●		●	●		●			do nothing
●	●	●		●	●		●		●	do nothing
●	●	●		●	●		●	●		give full course
●	●	●		●	●		●	●	●	do nothing
●	●	●		●	●	●				do nothing
●	●	●		●	●	●			●	do nothing
●	●	●		●	●	●		●		give full course
●	●	●		●	●	●		●	●	do nothing
●	●	●		●	●	●	●			do nothing
●	●	●		●	●	●	●		●	do nothing
●	●	●		●	●	●	●	●		give full course
●	●	●		●	●	●	●	●	●	do nothing
●	●	●	●							give part course
●	●	●	●						●	do nothing
●	●	●	●					●		give full course
●	●	●	●					●	●	do nothing
●	●	●	●				●			give part course
●	●	●	●				●		●	do nothing
●	●	●	●				●	●		give full course
●	●	●	●				●	●	●	do nothing
●	●	●	●			●				give part course
●	●	●	●			●			●	do nothing
●	●	●	●			●		●		give full course
●	●	●	●			●		●	●	do nothing
●	●	●	●			●	●			give part course
●	●	●	●			●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
•	•	•	•			•	•	•		give full course
•	•	•	•			•	•	•	•	do nothing
•	•	•	•		•					give part course
•	•	•	•		•				•	do nothing
•	•	•	•		•			•		give full course
•	•	•	•		•			•	•	do nothing
•	•	•	•		•		•			give part course
•	•	•	•		•		•		•	do nothing
•	•	•	•		•		•	•		give full course
•	•	•	•		•		•	•	•	do nothing
•	•	•	•		•	•				give part course
•	•	•	•		•	•			•	do nothing
•	•	•	•		•	•		•		give full course
•	•	•	•		•	•		•	•	do nothing
•	•	•	•		•	•	•			give part course
•	•	•	•		•	•	•		•	do nothing
•	•	•	•		•	•	•	•		give full course
•	•	•	•		•	•	•	•	•	do nothing
•	•	•	•	•						do nothing
•	•	•	•	•					•	do nothing
•	•	•	•	•				•		give full course
•	•	•	•	•				•	•	do nothing
•	•	•	•	•			•			do nothing
•	•	•	•	•			•		•	do nothing
•	•	•	•	•			•	•		give full course
•	•	•	•	•			•	•	•	do nothing
•	•	•	•	•		•				do nothing
•	•	•	•	•		•			•	do nothing
•	•	•	•	•		•		•		give full course
•	•	•	•	•		•		•	•	do nothing
•	•	•	•	•		•	•			do nothing
•	•	•	•	•		•	•		•	do nothing
•	•	•	•	•		•	•	•		give full course
•	•	•	•	•		•	•	•	•	do nothing
•	•	•	•	•	•					do nothing
•	•	•	•	•	•				•	do nothing
•	•	•	•	•	•			•		give full course
•	•	•	•	•	•			•	•	do nothing
•	•	•	•	•	•		•			do nothing
•	•	•	•	•	•		•		•	do nothing
•	•	•	•	•	•		•	•		give full course
•	•	•	•	•	•		•	•	•	do nothing
•	•	•	•	•	•	•				do nothing
•	•	•	•	•	•	•			•	do nothing
•	•	•	•	•	•	•		•		give full course
•	•	•	•	•	•	•		•	•	do nothing
•	•	•	•	•	•	•	•			do nothing
•	•	•	•	•	•	•	•		•	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
•	•	•	•	•	•	•	•	•		give full course
•	•	•	•	•	•	•	•	•	•	do nothing
•	•									give part course
•	•								•	do nothing
•	•							•		give part course
•	•							•	•	do nothing
•	•						•			give part course
•	•						•		•	do nothing
•	•						•	•		give part course
•	•						•	•	•	do nothing
•	•					•				give part course
•	•					•			•	do nothing
•	•					•		•		give part course
•	•					•		•	•	do nothing
•	•					•	•			give part course
•	•					•	•		•	do nothing
•	•					•	•	•		give part course
•	•					•	•	•	•	do nothing
•	•				•					give part course
•	•				•				•	do nothing
•	•				•			•		give part course
•	•				•			•	•	do nothing
•	•				•		•			give part course
•	•				•		•		•	do nothing
•	•				•		•	•		give part course
•	•				•		•	•	•	do nothing
•	•				•	•				give part course
•	•				•	•			•	do nothing
•	•				•	•		•		give part course
•	•				•	•		•	•	do nothing
•	•				•	•	•			give part course
•	•				•	•	•		•	do nothing
•	•				•	•	•	•		give part course
•	•				•	•	•	•	•	do nothing
•	•			•						do nothing
•	•			•					•	do nothing
•	•			•				•		do nothing
•	•			•				•	•	do nothing
•	•			•			•			do nothing
•	•			•			•		•	do nothing
•	•			•			•	•		buy full anti-malarial treatment
•	•			•			•	•	•	do nothing
•	•			•		•				do nothing
•	•			•		•			•	do nothing
•	•			•		•		•		do nothing
•	•			•		•		•	•	do nothing
•	•			•		•	•			do nothing
•	•			•		•	•		•	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●	●			●		●	●	●		buy full anti-malarial treatment
●	●			●		●	●	●	●	do nothing
●	●			●	●					do nothing
●	●			●	●				●	do nothing
●	●			●	●			●		do nothing
●	●			●	●			●	●	do nothing
●	●			●	●		●			buy full anti-malarial treatment
●	●			●	●		●		●	do nothing
●	●			●	●		●	●		buy full anti-malarial treatment
●	●			●	●		●	●	●	do nothing
●	●			●	●	●				do nothing
●	●			●	●	●			●	do nothing
●	●			●	●	●		●		do nothing
●	●			●	●	●		●	●	do nothing
●	●			●	●	●	●			buy full anti-malarial treatment
●	●			●	●	●	●		●	do nothing
●	●			●	●	●	●	●		go to health care facility
●	●			●	●	●	●	●	●	do nothing
●	●		●							give part course
●	●		●						●	do nothing
●	●		●					●		give part course
●	●		●					●	●	do nothing
●	●		●				●			give part course
●	●		●				●		●	do nothing
●	●		●				●	●		give part course
●	●		●				●	●	●	do nothing
●	●		●			●				give part course
●	●		●			●			●	do nothing
●	●		●			●		●		give part course
●	●		●			●		●	●	do nothing
●	●		●			●	●			give part course
●	●		●			●	●		●	do nothing
●	●		●			●	●	●		give part course
●	●		●			●	●	●	●	do nothing
●	●		●		●					give part course
●	●		●		●				●	do nothing
●	●		●		●			●		give part course
●	●		●		●			●	●	do nothing
●	●		●		●		●			give part course
●	●		●		●		●		●	do nothing
●	●		●		●		●	●		give part course
●	●		●		●		●	●	●	do nothing
●	●		●		●	●				give part course
●	●		●		●	●			●	do nothing
●	●		●		●	●		●		give part course
●	●		●		●	●		●	●	do nothing
●	●		●		●	●	●			give part course
●	●		●		●	●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●	●		●		●	●	●	●		give part course
●	●		●		●	●	●	●	●	do nothing
●	●		●	●						do nothing
●	●		●	●					●	do nothing
●	●		●	●				●		do nothing
●	●		●	●				●	●	do nothing
●	●		●	●			●			do nothing
●	●		●	●			●		●	do nothing
●	●		●	●			●	●		buy full anti-malarial treatment
●	●		●	●			●	●	●	do nothing
●	●		●	●		●				do nothing
●	●		●	●		●			●	do nothing
●	●		●	●		●		●		do nothing
●	●		●	●		●		●	●	do nothing
●	●		●	●		●	●			do nothing
●	●		●	●		●	●		●	do nothing
●	●		●	●		●	●	●		buy full anti-malarial treatment
●	●		●	●		●	●	●	●	do nothing
●	●		●	●	●					do nothing
●	●		●	●	●				●	do nothing
●	●		●	●	●			●		do nothing
●	●		●	●	●			●	●	do nothing
●	●		●	●	●		●			buy full anti-malarial treatment
●	●		●	●	●		●		●	do nothing
●	●		●	●	●		●	●		buy full anti-malarial treatment
●	●		●	●	●		●	●	●	do nothing
●	●		●	●	●	●				do nothing
●	●		●	●	●	●			●	do nothing
●	●		●	●	●	●		●		do nothing
●	●		●	●	●	●		●	●	do nothing
●	●		●	●	●	●	●			buy full anti-malarial treatment
●	●		●	●	●	●	●		●	do nothing
●	●		●	●	●	●	●	●		go to health care facility
●	●		●	●	●	●	●	●	●	do nothing
●	●	●								give part course
●	●	●							●	do nothing
●	●	●						●		give full course
●	●	●						●	●	do nothing
●	●	●					●			give part course
●	●	●					●		●	do nothing
●	●	●					●	●		give full course
●	●	●					●	●	●	do nothing
●	●	●				●				give part course
●	●	●				●			●	do nothing
●	●	●				●		●		give full course
●	●	●				●		●	●	do nothing
●	●	●				●	●			give part course
●	●	●				●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
•	•	•				•	•	•		give full course
•	•	•				•	•	•	•	do nothing
•	•	•			•					give part course
•	•	•			•				•	do nothing
•	•	•			•			•		give full course
•	•	•			•			•	•	do nothing
•	•	•			•		•			give part course
•	•	•			•		•		•	do nothing
•	•	•			•		•	•		give full course
•	•	•			•		•	•	•	do nothing
•	•	•			•	•				give part course
•	•	•			•	•			•	do nothing
•	•	•			•	•		•		give full course
•	•	•			•	•		•	•	do nothing
•	•	•			•	•	•			give part course
•	•	•			•	•	•		•	do nothing
•	•	•			•	•	•	•		give full course
•	•	•			•	•	•	•	•	do nothing
•	•	•		•						do nothing
•	•	•		•					•	do nothing
•	•	•		•				•		give full course
•	•	•		•				•	•	do nothing
•	•	•		•			•			do nothing
•	•	•		•			•		•	do nothing
•	•	•		•			•	•		give full course
•	•	•		•			•	•	•	do nothing
•	•	•		•		•				do nothing
•	•	•		•		•			•	do nothing
•	•	•		•		•		•		give full course
•	•	•		•		•		•	•	do nothing
•	•	•		•		•	•			do nothing
•	•	•		•		•	•		•	do nothing
•	•	•		•		•	•	•		give full course
•	•	•		•		•	•	•	•	do nothing
•	•	•		•	•					do nothing
•	•	•		•	•				•	do nothing
•	•	•		•	•			•		give full course
•	•	•		•	•			•	•	do nothing
•	•	•		•	•		•			do nothing
•	•	•		•	•		•		•	do nothing
•	•	•		•	•		•	•		give full course
•	•	•		•	•		•	•	•	do nothing
•	•	•		•	•	•				do nothing
•	•	•		•	•	•			•	do nothing
•	•	•		•	•	•		•		give full course
•	•	•		•	•	•		•	•	do nothing
•	•	•		•	•	•	•			do nothing
•	•	•		•	•	•	•		•	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●	●	●		●	●	●	●	●		give full course
●	●	●		●	●	●	●	●	●	do nothing
●	●	●	●							give part course
●	●	●	●						●	do nothing
●	●	●	●					●		give full course
●	●	●	●					●	●	do nothing
●	●	●	●				●			give part course
●	●	●	●				●		●	do nothing
●	●	●	●				●	●		give full course
●	●	●	●				●	●	●	do nothing
●	●	●	●			●				give part course
●	●	●	●			●			●	do nothing
●	●	●	●			●		●		give full course
●	●	●	●			●		●	●	do nothing
●	●	●	●			●	●			give part course
●	●	●	●			●	●		●	do nothing
●	●	●	●			●	●	●		give full course
●	●	●	●			●	●	●	●	do nothing
●	●	●	●		●					give part course
●	●	●	●		●				●	do nothing
●	●	●	●		●			●		give full course
●	●	●	●		●			●	●	do nothing
●	●	●	●		●		●			give part course
●	●	●	●		●		●		●	do nothing
●	●	●	●		●		●	●		give full course
●	●	●	●		●		●	●	●	do nothing
●	●	●	●		●	●				give part course
●	●	●	●		●	●			●	do nothing
●	●	●	●		●	●		●		give full course
●	●	●	●		●	●		●	●	do nothing
●	●	●	●		●	●	●			give part course
●	●	●	●		●	●	●		●	do nothing
●	●	●	●		●	●	●	●		give full course
●	●	●	●		●	●	●	●	●	do nothing
●	●	●	●	●						do nothing
●	●	●	●	●					●	do nothing
●	●	●	●	●				●		give full course
●	●	●	●	●				●	●	do nothing
●	●	●	●	●			●			do nothing
●	●	●	●	●			●		●	do nothing
●	●	●	●	●			●	●		give full course
●	●	●	●	●			●	●	●	do nothing
●	●	●	●	●		●				do nothing
●	●	●	●	●		●			●	do nothing
●	●	●	●	●		●		●		give full course
●	●	●	●	●		●		●	●	do nothing
●	●	●	●	●		●	●			do nothing
●	●	●	●	●		●	●		●	do nothing

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?	
●	●	●	●	●		●	●	●		give full course
●	●	●	●	●		●	●	●	●	do nothing
●	●	●	●	●	●					do nothing
●	●	●	●	●	●				●	do nothing
●	●	●	●	●	●			●		give full course
●	●	●	●	●	●			●	●	do nothing
●	●	●	●	●	●		●			do nothing
●	●	●	●	●	●		●		●	do nothing
●	●	●	●	●	●		●	●		give full course
●	●	●	●	●	●		●	●	●	do nothing
●	●	●	●	●	●	●				do nothing
●	●	●	●	●	●	●			●	do nothing
●	●	●	●	●	●	●		●		give full course
●	●	●	●	●	●	●		●	●	do nothing
●	●	●	●	●	●	●	●			do nothing
●	●	●	●	●	●	●	●		●	do nothing
●	●	●	●	●	●	●	●	●		give full course
●	●	●	●	●	●	●	●	●	●	do nothing

D.2 Ordered by plans

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
do nothing									
									●
								●	
								●	●
							●		
							●		●
							●	●	
							●	●	●
						●			
						●			●
						●		●	
						●		●	●
						●	●		
						●	●		●
						●	●	●	
						●	●	●	●
					●				
					●				●
					●			●	
					●			●	●
					●		●		
					●		●		●
					●		●	●	
					●		●	●	●
					●	●			
					●	●			●
					●	●		●	
					●	●		●	●
					●	●	●		
					●	●	●		●
					●	●	●	●	
					●	●	●	●	●
				●					
				●					●
				●				●	
				●				●	●
				●			●		
				●			●		●
				●			●	●	
				●			●	●	●
				●		●			
				●		●			●
				●		●		●	
				●		●		●	●
				●		●	●		
				●		●	●		●
				●		●	●	●	
				●		●	●	●	●
				●	●				
				●	●				●
				●	●			●	
				●	●			●	●
				●	●		●		
				●	●		●		●
				●	●		●	●	
				●	●		●	●	●
				●	●	●			
				●	●	●			●
				●	●	●		●	
				●	●	●		●	●
				●	●	●	●		
				●	●	●	●		●
				●	●	●	●	●	
				●	●	●	●	●	●
			●						
			●						●
			●					●	
			●					●	●
			●				●		
			●				●		●
			●				●	●	
			●				●	●	●
			●			●			
			●			●			●
			●			●		●	
			●			●		●	●
			●			●	●		
			●			●	●		●
			●			●	●	●	
			●			●	●	●	●
			●		●				
			●		●				●
			●		●			●	
			●		●			●	●
			●		●		●		
			●		●		●		●
			●		●		●	●	
			●		●		●	●	●
			●		●	●			
			●		●	●			●
			●		●	●		●	
			●		●	●		●	●
			●		●	●	●		
			●		●	●	●		●
			●		●	●	●	●	
			●		●	●	●	●	●
			●	●					
			●	●					●
			●	●				●	
			●	●				●	●
			●	●			●		
			●	●			●		●
			●	●			●	●	
			●	●			●	●	●
			●	●		●			
			●	●		●			●
			●	●		●		●	
			●	●		●		●	●
			●	●		●	●		
			●	●		●	●		●
			●	●		●	●	●	
			●	●		●	●	●	●
			●	●	●				
			●	●	●				●
			●	●	●			●	
			●	●	●			●	●
			●	●	●		●		
			●	●	●		●		●
			●	●	●		●	●	
			●	●	●		●	●	●
			●	●	●	●			
			●	●	●	●			●
			●	●	●	●		●	
			●	●	●	●		●	●
			●	●	●	●	●		
			●	●	●	●	●		●
			●	●	●	●	●	●	
			●	●	●	●	●	●	●
	●								
	●								●
	●							●	
	●							●	●
	●						●		
	●						●		●
	●						●	●	
	●						●	●	●
	●					●			
	●					●			●
	●					●		●	
	●					●		●	●
	●					●	●		
	●					●	●		●
	●					●	●	●	
	●					●	●	●	●
	●				●				
	●				●				●
	●				●			●	
	●				●			●	●
	●				●		●		
	●				●		●		●
	●				●		●	●	
	●				●		●	●	●
	●				●	●			
	●				●	●			●
	●				●	●		●	
	●				●	●		●	●
	●				●	●	●		

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
	●				●	●	●		●
	●				●	●	●	●	
	●				●	●	●	●	●
	●			●					
	●			●					●
	●			●				●	
	●			●				●	●
	●			●			●		
	●			●			●		●
	●			●			●	●	
	●			●			●	●	●
	●			●		●			
	●			●		●			●
	●			●		●		●	
	●			●		●		●	●
	●			●		●	●		
	●			●		●	●		●
	●			●		●	●	●	
	●			●		●	●	●	●
	●			●	●				
	●			●	●				●
	●			●	●			●	
	●			●	●			●	●
	●			●	●		●		
	●			●	●		●		●
	●			●	●		●	●	
	●			●	●		●	●	●
	●			●	●	●			
	●			●	●	●			●
	●			●	●	●		●	
	●			●	●	●		●	●
	●			●	●	●	●		
	●			●	●	●	●		●
	●			●	●	●	●	●	
	●			●	●	●	●	●	●
	●		●						
	●		●						●
	●		●					●	
	●		●					●	●
	●		●				●		
	●		●				●		●
	●		●				●	●	
	●		●				●	●	●
	●		●			●			
	●		●			●			●
	●		●			●		●	
	●		●			●		●	●
	●		●			●	●		
	●		●			●	●		●
	●		●			●	●	●	
	●		●			●	●	●	●
	●		●		●				
	●		●		●				●
	●		●		●			●	
	●		●		●			●	●
	●		●		●		●		
	●		●		●		●		●
	●		●		●		●	●	
	●		●		●		●	●	●
	●		●		●	●			
	●		●		●	●			●
	●		●		●	●		●	
	●		●		●	●		●	●
	●		●		●	●	●		
	●		●		●	●	●		●
	●		●		●	●	●	●	
	●		●		●	●	●	●	●
	●		●	●					
	●		●	●					●
	●		●	●				●	
	●		●	●				●	●
	●		●	●			●		
	●		●	●			●		●
	●		●	●			●	●	
	●		●	●			●	●	●
	●		●	●		●			
	●		●	●		●			●
	●		●	●		●		●	
	●		●	●		●		●	●
	●		●	●		●	●		
	●		●	●		●	●		●
	●		●	●		●	●	●	
	●		●	●		●	●	●	●
	●		●	●	●				

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
	●		●	●	●				●
	●		●	●	●			●	
	●		●	●	●			●	●
	●		●	●	●		●		
	●		●	●	●		●		●
	●		●	●	●		●	●	
	●		●	●	●		●	●	●
	●		●	●	●	●			
	●		●	●	●	●			●
	●		●	●	●	●		●	
	●		●	●	●	●		●	●
	●		●	●	●	●	●		
	●		●	●	●	●	●		●
	●		●	●	●	●	●	●	
	●		●	●	●	●	●	●	●
	●	●							
	●	●							●
	●	●						●	
	●	●						●	●
	●	●					●		
	●	●					●		●
	●	●					●	●	
	●	●					●	●	●
	●	●				●			
	●	●				●			●
	●	●				●		●	
	●	●				●		●	●
	●	●				●	●		
	●	●				●	●		●
	●	●				●	●	●	
	●	●				●	●	●	●
	●	●			●				
	●	●			●				●
	●	●			●			●	
	●	●			●			●	●
	●	●			●		●		
	●	●			●		●		●
	●	●			●		●	●	
	●	●			●		●	●	●
	●	●			●	●			
	●	●			●	●			●
	●	●			●	●		●	
	●	●			●	●		●	●
	●	●			●	●	●		
	●	●			●	●	●		●
	●	●			●	●	●	●	
	●	●			●	●	●	●	●
	●	●		●					
	●	●		●					●
	●	●		●				●	
	●	●		●				●	●
	●	●		●			●		
	●	●		●			●		●
	●	●		●			●	●	
	●	●		●			●	●	●
	●	●		●		●			
	●	●		●		●			●
	●	●		●		●		●	
	●	●		●		●		●	●
	●	●		●		●	●		
	●	●		●		●	●		●
	●	●		●		●	●	●	
	●	●		●		●	●	●	●
	●	●		●	●				
	●	●		●	●				●
	●	●		●	●			●	
	●	●		●	●			●	●
	●	●		●	●		●		
	●	●		●	●		●		●
	●	●		●	●		●	●	
	●	●		●	●		●	●	●
	●	●		●	●	●			
	●	●		●	●	●			●
	●	●		●	●	●		●	
	●	●		●	●	●		●	●
	●	●		●	●	●	●		
	●	●		●	●	●	●		●
	●	●		●	●	●	●	●	
	●	●		●	●	●	●	●	●
	●	●	●						
	●	●	●						●
	●	●	●					●	
	●	●	●					●	●
	●	●	●				●		

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
	•	•	•				•		•
	•	•	•				•	•	
	•	•	•				•	•	•
	•	•	•			•			
	•	•	•			•			•
	•	•	•			•		•	
	•	•	•			•		•	•
	•	•	•			•	•		
	•	•	•			•	•		•
	•	•	•			•	•	•	
	•	•	•			•	•	•	•
	•	•	•		•				
	•	•	•		•				•
	•	•	•		•			•	
	•	•	•		•			•	•
	•	•	•		•		•		
	•	•	•		•		•		•
	•	•	•		•		•	•	
	•	•	•		•		•	•	•
	•	•	•		•	•			
	•	•	•		•	•			•
	•	•	•		•	•		•	
	•	•	•		•	•		•	•
	•	•	•		•	•	•		
	•	•	•		•	•	•		•
	•	•	•		•	•	•	•	
	•	•	•		•	•	•	•	•
	•	•	•	•					
	•	•	•	•					•
	•	•	•	•				•	
	•	•	•	•				•	•
	•	•	•	•			•		
	•	•	•	•			•		•
	•	•	•	•			•	•	
	•	•	•	•			•	•	•
	•	•	•	•		•			
	•	•	•	•		•			•
	•	•	•	•		•		•	
	•	•	•	•		•		•	•
	•	•	•	•		•	•		
	•	•	•	•		•	•		•
	•	•	•	•		•	•	•	
	•	•	•	•		•	•	•	•
	•	•	•	•	•				
	•	•	•	•	•				•
	•	•	•	•	•			•	
	•	•	•	•	•			•	•
	•	•	•	•	•		•		
	•	•	•	•	•		•		•
	•	•	•	•	•		•	•	
	•	•	•	•	•		•	•	•
	•	•	•	•	•	•			
	•	•	•	•	•	•			•
	•	•	•	•	•	•		•	
	•	•	•	•	•	•		•	•
	•	•	•	•	•	•	•		
	•	•	•	•	•	•	•		•
	•	•	•	•	•	•	•	•	
	•	•	•	•	•	•	•	•	•
•									•
•								•	•
•							•		•
•							•	•	•
•						•			•
•						•		•	•
•						•	•		•
•						•	•	•	•
•					•				•
•					•			•	•
•					•		•		•
•					•		•	•	•
•					•	•			
•					•	•			•
•					•	•		•	
•					•	•		•	•
•					•	•	•		
•					•	•	•		•
•					•	•	•	•	
•					•	•	•	•	•
•				•					
•				•					•
•				•				•	•
•				•			•		
•				•			•		•

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
•				•			•	•	•
•				•		•			
•				•		•			•
•				•		•		•	•
•				•		•	•		
•				•		•	•		•
•				•		•	•	•	•
•				•	•				
•				•	•				•
•				•	•			•	
•				•	•			•	•
•				•	•		•		•
•				•	•		•	•	•
•				•	•	•			
•				•	•	•			•
•				•	•	•		•	
•				•	•	•		•	•
•				•	•	•	•		•
•				•	•	•	•	•	•
•			•						•
•			•					•	•
•			•				•		•
•			•				•	•	•
•			•			•			•
•			•			•		•	•
•			•			•	•		•
•			•			•	•	•	•
•			•		•				•
•			•		•			•	•
•			•		•		•		•
•			•		•		•	•	•
•			•		•	•			
•			•		•	•			•
•			•		•	•		•	
•			•		•	•		•	•
•			•		•	•	•		
•			•		•	•	•		•
•			•		•	•	•	•	
•			•		•	•	•	•	•
•			•	•					
•			•	•					•
•			•	•				•	•
•			•	•			•		
•			•	•			•		•
•			•	•			•	•	•
•			•	•		•			
•			•	•		•			•
•			•	•		•		•	•
•			•	•		•	•		
•			•	•		•	•		•
•			•	•		•	•	•	•
•			•	•	•				
•			•	•	•				•
•			•	•	•			•	
•			•	•	•			•	•
•			•	•	•		•		•
•			•	•	•		•	•	•
•			•	•	•	•			
•			•	•	•	•			•
•			•	•	•	•		•	
•			•	•	•	•		•	•
•			•	•	•	•	•		•
•			•	•	•	•	•	•	•
•	•								•
•	•							•	•
•	•						•		•
•	•						•	•	•
•	•					•			•
•	•					•		•	•
•	•					•	•		•
•	•					•	•	•	•
•	•				•				•
•	•				•			•	•
•	•				•		•		•
•	•				•		•	•	•
•	•				•	•			•
•	•				•	•		•	•
•	•				•	•	•		•
•	•				•	•	•	•	•
•	•			•					
•	•			•					•
•	•			•				•	
•	•			•				•	•
•	•			•			•		

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
●	●			●			●		●
●	●			●			●	●	●
●	●			●		●			
●	●			●		●			●
●	●			●		●		●	
●	●			●		●		●	●
●	●			●		●	●		
●	●			●		●	●		●
●	●			●		●	●	●	●
●	●			●	●				
●	●			●	●				●
●	●			●	●			●	
●	●			●	●			●	●
●	●			●	●		●		●
●	●			●	●		●	●	●
●	●			●	●	●			
●	●			●	●	●			●
●	●			●	●	●		●	
●	●			●	●	●		●	●
●	●			●	●	●	●		●
●	●			●	●	●	●	●	●
●	●		●						●
●	●		●					●	●
●	●		●				●		●
●	●		●				●	●	●
●	●		●			●			●
●	●		●			●		●	●
●	●		●			●	●		●
●	●		●			●	●	●	●
●	●		●		●				●
●	●		●		●			●	●
●	●		●		●		●		●
●	●		●		●		●	●	●
●	●		●		●	●			●
●	●		●		●	●		●	●
●	●		●		●	●	●		●
●	●		●		●	●	●	●	●
●	●		●	●					
●	●		●	●					●
●	●		●	●				●	
●	●		●	●				●	●
●	●		●	●			●		
●	●		●	●			●		●
●	●		●	●			●	●	●
●	●		●	●		●			
●	●		●	●		●			●
●	●		●	●		●		●	
●	●		●	●		●		●	●
●	●		●	●		●	●		
●	●		●	●		●	●		●
●	●		●	●		●	●	●	●
●	●		●	●	●				
●	●		●	●	●				●
●	●		●	●	●			●	
●	●		●	●	●			●	●
●	●		●	●	●		●		●
●	●		●	●	●		●	●	●
●	●		●	●	●	●			
●	●		●	●	●	●			●
●	●		●	●	●	●		●	
●	●		●	●	●	●		●	●
●	●		●	●	●	●	●		●
●	●		●	●	●	●	●	●	●
●	●	●							●
●	●	●						●	●
●	●	●					●		●
●	●	●					●	●	●
●	●	●				●			●
●	●	●				●		●	●
●	●	●				●	●		●
●	●	●				●	●	●	●
●	●	●			●				●
●	●	●			●			●	●
●	●	●			●		●		●
●	●	●			●		●	●	●
●	●	●			●	●			●
●	●	●			●	●		●	●
●	●	●			●	●	●		●
●	●	●			●	●	●	●	●
●	●	●		●					
●	●	●		●					●
●	●	●		●				●	●
●	●	●		●			●		
●	●	●		●			●		●

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
●	●	●		●			●	●	●
●	●	●		●		●			
●	●	●		●		●			●
●	●	●		●		●		●	●
●	●	●		●		●	●		
●	●	●		●		●	●		●
●	●	●		●		●	●	●	●
●	●	●		●	●				
●	●	●		●	●				●
●	●	●		●	●			●	●
●	●	●		●	●		●		
●	●	●		●	●		●		●
●	●	●		●	●		●	●	●
●	●	●		●	●	●			
●	●	●		●	●	●			●
●	●	●		●	●	●		●	●
●	●	●		●	●	●	●		
●	●	●		●	●	●	●		●
●	●	●		●	●	●	●	●	●
●	●	●	●						●
●	●	●	●					●	●
●	●	●	●				●		●
●	●	●	●				●	●	●
●	●	●	●			●			●
●	●	●	●			●		●	●
●	●	●	●			●	●		●
●	●	●	●			●	●	●	●
●	●	●	●		●				●
●	●	●	●		●			●	●
●	●	●	●		●		●		●
●	●	●	●		●		●	●	●
●	●	●	●		●	●			●
●	●	●	●		●	●		●	●
●	●	●	●		●	●	●		●
●	●	●	●		●	●	●	●	●
●	●	●	●	●					
●	●	●	●	●					●
●	●	●	●	●				●	●
●	●	●	●	●			●		
●	●	●	●	●			●		●
●	●	●	●	●			●	●	●
●	●	●	●	●		●			
●	●	●	●	●		●			●
●	●	●	●	●		●		●	●
●	●	●	●	●		●	●		
●	●	●	●	●		●	●		●
●	●	●	●	●		●	●	●	●
●	●	●	●	●	●				
●	●	●	●	●	●				●
●	●	●	●	●	●			●	●
●	●	●	●	●	●		●		
●	●	●	●	●	●		●		●
●	●	●	●	●	●		●	●	●
●	●	●	●	●	●	●			
●	●	●	●	●	●	●			●
●	●	●	●	●	●	●		●	●
●	●	●	●	●	●	●	●		
●	●	●	●	●	●	●	●		●
●	●	●	●	●	●	●	●	●	●
buy anti-pyretic drug									
●									
●								●	
●							●		
●							●	●	
●						●			
●						●		●	
●						●	●		
●						●	●	●	
●					●				
●					●			●	
●					●		●		
●					●		●	●	
●			●						
●			●					●	
●			●				●		
●			●				●	●	
●			●			●			
●			●			●		●	
●			●			●	●		
●			●			●	●	●	
●			●		●				
●			●		●			●	
●			●		●		●		
●			●		●		●	●	

fever?	drug?	enough?	save money?	serious?	affordable?	accessible?	effective?	trained?	bad medicine?
buy part anti-malarial treatment									
●				●				●	
●				●		●		●	
●			●	●				●	
●			●	●		●		●	
buy full anti-malarial treatment									
●				●			●	●	
●				●		●	●	●	
●				●	●		●		
●				●	●		●	●	
●				●	●	●	●		
●			●	●			●	●	
●			●	●		●	●	●	
●			●	●	●		●		
●			●	●	●		●	●	
●			●	●	●	●	●		
●	●			●			●	●	
●	●			●		●	●	●	
●	●			●	●		●		
●	●			●	●		●	●	
●	●			●	●	●	●		
●	●		●	●			●	●	
●	●		●	●		●	●	●	
●	●		●	●	●		●		
●	●		●	●	●		●	●	
●	●		●	●	●	●	●		
go to health care facility									
●				●	●	●	●	●	
●			●	●	●	●	●	●	
●	●			●	●	●	●	●	
●	●		●	●	●	●	●	●	
give part course									
●	●								
●	●							●	
●	●						●		
●	●						●	●	
●	●					●			
●	●					●		●	
●	●					●	●		
●	●					●	●	●	
●	●				●				
●	●				●			●	
●	●				●		●		
●	●				●		●	●	
●	●				●	●			
●	●				●	●		●	
●	●				●	●	●		
●	●				●	●	●	●	
●	●		●						
●	●		●					●	
●	●		●				●		
●	●		●				●	●	
●	●		●			●			
●	●		●			●		●	
●	●		●			●	●		
●	●		●			●	●	●	
●	●		●		●				
●	●		●		●			●	
●	●		●		●		●		
●	●		●		●		●	●	
●	●		●		●	●			
●	●		●		●	●		●	
●	●		●		●	●	●		
●	●		●		●	●	●	●	
●	●	●							
●	●	●					●		
●	●	●				●			
●	●	●				●	●		
●	●	●			●				
●	●	●			●		●		
●	●	●			●	●			
●	●	●			●	●	●		
●	●	●	●						
●	●	●	●				●		
●	●	●	●			●			
●	●	●	●			●	●		
●	●	●	●		●				
●	●	●	●		●		●		
●	●	●	●		●	●			
●	●	●	●		●	●	●		
give full course									
●	●	●						●	
●	●	●					●	●	
●	●	●				●		●	
●	●	●				●	●	●	
●	●	●			●			●	
●	●	●			●		●	●	
●	●	●			●	●		●	
●	●	●			●	●	●	●	
●	●	●		●				●	
●	●	●		●			●	●	
●	●	●		●		●		●	
●	●	●		●		●	●	●	
●	●	●		●	●			●	
●	●	●		●	●		●	●	
●	●	●		●	●	●		●	
●	●	●		●	●	●	●	●	
●	●	●	●					●	
●	●	●	●				●	●	
●	●	●	●			●		●	
●	●	●	●			●	●	●	
●	●	●	●		●			●	
●	●	●	●		●		●	●	
●	●	●	●		●	●		●	
●	●	●	●		●	●	●	●	
●	●	●	●	●				●	
●	●	●	●	●			●	●	
●	●	●	●	●		●		●	
●	●	●	●	●		●	●	●	
●	●	●	●	●	●			●	
●	●	●	●	●	●		●	●	
●	●	●	●	●	●	●		●	
●	●	●	●	●	●	●	●	●	

Appendix E

Gnuplot script

The following script has been used to generate graphs from the CSV files. It can be easily modified to suit other needs. A good point of reference is gnuplot’s homepage `http://www.gnuplot.info` where the software can also be downloaded for various platforms.

```
# plot plans against comparison value
# $Id$
#
# columns in 'test.csv'
# [ 1] comparison value
# [ 2] do nothing
# [ 3] buy anti-pyretic drug
# [ 4] buy part anti-malarial treatment
# [ 5] buy full anti-malarial treatment
# [ 6] go to health care facility
# [ 7] give part course
# [ 8] give full course
#

set terminal postscript eps enhanced color #png
set output 'test_money.eps'

set nogrid
set data style lines #points

set key title ""

set log y

# set xlabel "generations"
# set ylabel "cells [log_{10}]" 1

plot \
    'test.csv' index 0 using 1 title "money", \
    'test.csv' index 0 using 3 title "do nothing", \
    'test.csv' index 0 using 4 title "buy anti-pyretic drug", \
    'test.csv' index 0 using 5 title "buy part anti-malarial treatment", \
    'test.csv' index 0 using 6 title "go to health care facility", \
    ’test.csv’ index 0 using 7 title "give part course", \
    ’test.csv’ index 0 using 8 title "give full course"

set output


set terminal postscript eps enhanced color #png
set output ’test_distance.eps’

set nogrid
set data style lines #points

set key title ""

set log y

plot \
    ’test.csv’ index 1 using 1 title "distance", \
    ’test.csv’ index 1 using 3 title "do nothing", \
    ’test.csv’ index 1 using 4 title "buy anti-pyretic drug", \
    ’test.csv’ index 1 using 5 title "buy part anti-malarial treatment", \
    ’test.csv’ index 1 using 6 title "go to health care facility", \
    ’test.csv’ index 1 using 7 title "give part course", \
    ’test.csv’ index 1 using 8 title "give full course"

set output
```

Appendix F

Source Code

F.1 Collector.java

```java
/*
 * Created on 21.08.2004
 * $Id: Collector.java 45 2004-10-05 15:41:27Z Basti $
 */

package project;

import project.input.*;
import project.input.Stack;

import java.lang.reflect.*;
import java.util.*;

/**
 * The Collector collects all input parameters for a specific village
 * from all environments it is a member of by sorting them into a
 * hash table. In case of dublicate entries, certain <i>filter
 * rules</i> will be applied.
 */
public class Collector {

// private static Class[] stackClass = new Class[] {Stack.class.getClass()};
// public static Method AVG = Collector.class.getMethod("avg", stackClass);

    public static String AVERAGE = "avg";
    public static String MINIMUM = "min";
    public static String MAXIMUM = "max";
    public static String SUM = "sum";

    /**
     * gets the lowest value in the stack
     * @param stack stack to be worked on
     * @return lowest value in the stack
     */
    public static double min(Stack stack) {

        double array[] = stack.getValues();
        if (array.length < 1) {
//  throw new Exception();
        }

        double min = array[0];
        for (int i = 1; i < array.length; i++)
            min = (array[i] < min) ? array[i] : min;

        return min;
    }

    /**
     * gets the highest value in the stack
     * @param stack stack to be worked on
     * @return highest value in the stack
     */
    public static double max(Stack stack) {

        double array[] = stack.getValues();
        if (array.length < 1) {
//   throw new Exception();
        }

        double max = array[0];
        for (int i = 1; i < array.length; i++)
            max = (array[i] > max) ? array[i] : max;

        return max;
    }

    /**
     * gets the average value in the stack
     * @param stack stack to be worked on
     * @return average value in the stack
     */
    public static double avg(Stack stack) {

        double array[] = stack.getValues();
        if (array.length < 1) {
//   throw new Exception();
        }

        double total = 0;
        for (int i = 0; i < array.length; i++)
            total += array[i];

        return total / stack.size();
    }

    /**
     * gets the product of the stack
     * @param stack stack to be worked on
     * @return product of the stack
     */
    public static double mul(Stack stack) {

        double array[] = stack.getValues();
        if (array.length < 1) {
//   throw new Exception();
        }

        double total = array[0];
        for (int i = 1; i < array.length; i++)
            total *= array[i];

        return total;
    }

    /**
     * gets the sum of the stack
     * @param stack stack to be worked on
     * @return sum of the stack
     */
    public static double sum(Stack stack) {

        double array[] = stack.getValues();
        if (array.length < 1) {
//   throw new Exception();
        }

        double total = 0;
        for (int i = 0; i < array.length; i++)
            total += array[i];

        return total;
    }

    /**
     *
     * @param vil village
     * @param envs array of environmenst
     * @param filter filter methods to be used
     * @return an array of inputs that apply to village vil
     */
    public static Input[] collect(Village vil,
            Environment[] envs,
            Filter filter) {

        Hashtable table = new Hashtable();

        String name;
        Stack stack;
        Input[] inputs;

        // for all environments
        for (int e = 0; e < envs.length; e++) {

// System.err.println(”check environment ” + e + ” ...”);

            // where environment e contains village vil
            if (envs[e].contains(vil)) {

//  System.err.println(”contains ’” + vil.getName() + ”’”);

                // get inputs of e
                inputs = envs[e].getInputs();

                // for all inputs of e
                for (int i = 0; i < inputs.length; i++) {

                    name = inputs[i].getName();
//   System.out.print(name);

                    // if already there
                    if (table.containsKey(name)) {

                        // add value to stack
                        stack = (Stack) table.get(name);
                        stack.push(inputs[i].getValue());

//      System.out.println(” : exists → ” + stack);

                    }
                    // otherwise create
                    else {
                        table.put(name, new Stack(inputs[i].getValue()));
//      System.out.println(” = ” + inputs[i].getValue());
                    }
                } // end−for
            } // end−if
        } // end−for

// System.out.println(”\ncheck inputs...”);

        inputs = new Input[table.size()];
        int nr = −1;

        for (Enumeration keys = table.keys(); keys.hasMoreElements() ;) {
             name = (String) keys.nextElement();
// System.out.print(name);

             stack = (Stack) table.get(name);

             try {

                 if (stack.size() == 1) {
                        inputs[++nr] = new Input(name, stack.pop());
                        System.out.println(" = " + inputs[nr].getValue());
                 }
                 else if (stack.size() > 1) {

//  new Class[] {Stack.class.getClass()

                    String method = filter.get(name);
                    if (method == null)
                        System.err.println("No method declared for clash "+
                                "of ’" + name + "’!");

                    try {
                        Class cl = Class.forName("project.Collector");
                        Method me = cl.getMethod(method,
                                new Class[] {Class.forName("project.input.Stack")});

                        Double value = (Double) me.invoke(
                                null, new Object[] {stack});

                        inputs[++nr] = new Input(name, value.doubleValue());
//   System.e.print(” = ” + stack);
//   System.out.println(” → ” + inputs[nr].getValue());

                    }
                    catch (ClassNotFoundException ce) {
                        System.err.println("Class ’" + ce.getMessage() + "’ not found!");
                    }
                    catch (NoSuchMethodException ne) {
                        System.err.println("Method ’" + ne.getMessage() + "’ not found!");
                    }

                 }
                 else System.err.println("Funny stack size of " + stack.size());
            }
            catch (Exception e) {
                System.err.println(e.getMessage());
            }
         }

        return inputs;
    }

    public static void main(String[] args) {

        Village vil = new Village("Deuringen", new Input[] {
                new Input("distance", 2.2),
                new Input("consultation cost", 4.5)});
        Village vil2 = new Village("Steppach");

        Environment[] envs = new Environment[] {
                new Environment(vil, new Input[] {
                        new Input("true incident rate", 4.5),
                        new Input("false incident rate", 4.5/7),
                        new Input("resistance", .75)}),
                new Environment(new Village[] {vil, vil2}, new Input[] {
                        new Input("true incident rate", 2.5),
                        new Input("resistance", .5)}),
        };

        Filter filter = new Filter(new String[][] {
                new String[] {"true incident rate", Collector.AVERAGE},
                new String[] {"false incident rate", Collector.AVERAGE},
                new String[] {"rain season", Collector.AVERAGE},
                new String[] {"hostile environment", Collector.AVERAGE},
                new String[] {"vector control", Collector.AVERAGE},
                new String[] {"resistance", Collector.MINIMUM},
                new String[] {"qualified advice", Collector.MAXIMUM},
                new String[] {"non-qualified advice", Collector.MAXIMUM},
        });

        Input[] inputs = Collector.collect(vil, envs, filter);

        System.out.println("\ninput list...");
        for (int i = 0; i < inputs.length; i++)
            System.out.println(inputs[i].getName() + " = " +
                    inputs[i].getValue());

    }

}
```

F.2 Drug.java

```
/*
 * Created on 28.09.2004
 * $Id: Drug.java 57 2004-10-07 19:22:11Z Basti $
 */
package project;

import project.input.Input;

/**
 */
public class Drug {

    private String name;
    private Input[] inputs;

    /**
     * default constructor
     */
    public Drug() {}

    public Drug(String name, Input[] inputs) {
        this.name = name;
        this.inputs = inputs;
    }

    /**
     * @param inputs intputs to be set
     */
    public void setInputs(Input[] inputs) {
        this.inputs = inputs;
    }

    /**
     * string representation
     */
    public String toString() {
        String str = "";
        for (int i = 0; i < inputs.length; i++)
            str += inputs[i] + ", ";
        return name + " [" + str.substring(0, str.length() - 2) + "]";
    }

}
```

F.3 Environment.java

```
/*
 * Created on 21.08.2004
 * $Id: Environment.java 41 2004-09-28 11:59:54Z Basti $
 */

package project;

import project.input.*;
import java.util.ArrayList;

/**
 */
public class Environment extends ArrayList {

    private Input[] inputs;

    /**
     * default constructor
     */
    public Environment() {

        super();
    }

    public Environment(Village[] vil) {

        this();

        for (int i = 0; i < vil.length; i++)
            this.addVillage(vil[i]);
    }

    public Environment(Village vil, Input[] inputs) {

        this();
        this.addVillage(vil);
        this.inputs = inputs;
    }

    public Environment(Village[] vils, Input[] inputs) {

        this(vils);
        this.inputs = inputs;
    }

    public void addVillage(Village vil) {
        super.add(vil);
    }

    public Village getVillage(int nr) {
        return (Village) super.get(nr);
    }

    /**
     * Checks whether village vil belongs to the
     * environment
     * @param vil village to be checked
     * @return true</true> if vil belongs to the
     * environment
     */
    public boolean contains(Village vil) {
        return super.contains(vil);
    }

    public Input[] getInputs() {
        return inputs;
    }

    /**
     * get string representation
     */
    public String toString() {

        String istr = "";
        for (int i = 0; i < inputs.length; i++)
            istr += inputs[i] + ", ";

        String vstr = "";
        for (int i = 0; i < this.size(); i++)
            vstr += this.getVillage(i).getName() + ", ";

        return "Environment" +
            " {" + vstr.substring(0, vstr.length() - 2) + "}" +
            " [" + istr.substring(0, istr.length() - 2) + "]";
    }

    /**
     * test method
     * @param args
     */
    public static void main(String[] args) {

        Village a, b, c, d, e;
        Environment A, B, C;

        a = new Village("a");
        b = new Village("b");
        c = new Village("c");
        d = new Village("d");
        e = new Village("e");

        A = new Environment(new Village[] {a, b});
        B = new Environment(new Village[] {b, c});
        C = new Environment(new Village[] {c, d, e});

        System.out.println(A);
        System.out.println("\tA contains:");
        System.out.println("\ta? " + A.contains(a));
        System.out.println("\tb? " + A.contains(b));
        System.out.println("\tc? " + A.contains(c));

        System.out.println(B);
        System.out.println("\tB contains:");
        System.out.println("\ta? " + B.contains(a));
        System.out.println("\tb? " + B.contains(b));
        System.out.println("\tc? " + B.contains(c));

        System.out.println(C);
        System.out.println("\tC contains:");
        System.out.println("\ta? " + C.contains(a));
        System.out.println("\tb? " + C.contains(b));
        System.out.println("\tc? " + C.contains(c));

    }

}
```

F.4 God.java

```
/*
 * Created on 30-Aug-2004
 * $Id: God.java 50 2004-10-07 17:47:30Z Basti $
 */
package project;

import java.util.Random;

/**
 * This class is only used to initialise the population.
 */
public class God {

    /**
     * populate villages with households
     * @param vils vilages to be populated
     * @param households average number of households
     * @param children average number of children per household
     * @param adults average number of adults per household
     * @param income
     * @param disp disposable income (% of income)
     * @return populated villages
     */
    public static Village[] populate(Village[] vils,
            double households,
            double children,
            double adults,
            double income,
            double disp) {

        Random random = new Random();

        for (int i = 0; i < vils.length; i++) {

            Household[] hholds = new Household[(int) (random.nextDouble() * households)];
            for (int j = 0; j < hholds.length; j++) {
                hholds[j] = new Household();
                hholds[j].setKids((int) (random.nextDouble() * children));
                hholds[j].setAdults((int) (random.nextDouble() * adults));
            }

            vils[i].setHouseholds(hholds);
        }

        return vils;
    }
}
```

F.5 Household.java

```
/*
 * Created on 23.08.2004
 * $Id: Household.java 57 2004-10-07 19:22:11Z Basti $
 */

package project;

import project.mother.*;
import project.input.*;

/**
 */
public class Household {

    private Input[] inputs;

    private int kids = 0;
    private int adults = 0;
    private int income = 0;
    private double dispIncome = 0;


    /**
     * Has the mother drugs available?
     * 0 = no drugs
     * 1 = few drugs (part-treatment)
     * 2 = enough drugs (full-treatment)
     */
    private int drug = 0;

    /**
     * standard plans
     */
    public static Plan[] plans = new Plan[] {
            Plan.DO_NOTHING,
            Plan.BUY_AP,
            Plan.BUY_PART,
            Plan.BUY_FULL,
            Plan.HEALTH_CARE,
            Plan.GIVE_PART,
            Plan.GIVE_FULL
    };

    /**
     * what the mother hopes for (we assume that she wants to achieve as many
     * of her hopes as possible → all hopes joined with OR)
     */
    public static Belief[] hopes = new Belief[] {
            new Belief(Belief.FEVER, Belief.FALSE),
            new Belief(Belief.SERIOUS, Belief.FALSE),
            new Belief(Belief.SAVE_MONEY),
    };

    /**
     * default constructor
     */
    public Household() {
        inputs = new Input[0];
    }

    /**
     * constructor
     * @param kids number of children
     * @param adults number of adults
     * @param income income (in )
     * @param dispIncome disposable income (in %)
     */
    public Household(Input[] inputs) {
        //(int kids, int adults, int income, double dispIncome) {
        this();

//  this.kids = kids;
//  this.adults = adults;
//  this.income = income;
//  this.dispIncome = dispIncome;

        this.inputs = inputs;
    }

    /**
     * Has mother bought anti−malarial drug/has left−overs from last treatment?
     */
    public boolean hasDrug() {
        return drug > 0;
    }

    /**
     * Has mother enough drugs available?
     */
    public boolean hasEnough() {
        return drug > 1;
    }

    /**
     * buy additional drug
     * @param quantity quantity of drugs bought
     */
```

```
    public void buyDrug(int quantity) {
        drug += quantity;
    }

    /**
     * decide which plan to choose
     * @param beliefs belies about the world
     * @return plan to be executed
     * @todo more than one plan possible?
     */
    public int decide(BeliefTable beliefs) {

// Plan[] options = plan(beliefs);
        return (int) Collector.max(plan(beliefs));
    }

    /**
     * get available plans
     * @param beliefs beliefs about the world
     * @return list of available plans
     */
    public Stack plan(BeliefTable beliefs) {

// Plan[] subset = new Plan[plans.length];
        Stack subset = new Stack();
        int counter = 0;

// System.err.println(”The following plans trigger:”);
        for (int i = 0; i < plans.length; i++)
            if (plans[i].triggers(beliefs)) {
// System.err.println(plans[i].getName() +
// ” (utility=” + plans[i].getUtility(hopes) + ”)”);
// subset[counter++] = plans[i];
                subset.push(i);
            }

        // reduce array
// Plan[] dummy = new Plan[counter];
// for (int i = 0; i < counter; i++)
// dummy[i] = subset[i];

        return subset;
    }

    /**
     * @return available money per child
     */
    public double getMoney() {
        return (adults * income * dispIncome) / kids;
    }

    /**
     * @return returns inputs
     */
    public Input[] getInputs() {
        return inputs;
    }
    /**
     * @return number of adults

     */
    public int getAdults() {
        return adults;
    }

    /**
     * @return number of adults
     */
    public int getKids() {
        return kids;
    }

    /**
     * @param nr number of adults
     */
    public void setAdults(int nr) {
        adults = nr;
    }

    /**
     * @param nr number of children
     */
    public void setKids(int nr) {
        kids = nr;
    }

    /**
     * @return disposable income
     */
    public double getDispIncome() {
        return dispIncome;
    }

    /**
     * @return income
     */
    public int getIncome() {
        return income;
    }

    /**
     * @param d disposable income
     */
    public void setDispIncome(double d) {
        dispIncome = d;
    }

    /**
     * @param i income
     */
    public void setIncome(int i) {
        income = i;
    }

    /**
     * string representation
     */
    public String toString() {

        String istr = "";
        for (int i = 0; i < inputs.length; i++)
            istr += inputs[i] + ", ";

        return "Household" +
                " [" + istr.substring(0, istr.length() - 2) + "]";
    }
}
```

F.6 Mother.java

```
/*
 * Created on 28.09.2004
 * $Id: Mother.java 51 2004-10-07 17:48:20Z Basti $
 */
package project;

import java.io.IOException;

import project.input.*;

/**
 * Test simulation.
 *
 *
 */
public class Mother {

    public static void main(String[] args) {

        if (args.length != 1) {
            System.err.println("No configuration file specified!");
            System.err.println("Usage: java Mother <world.xml>");
            System.exit(-1);
        }

        World world = null;
        Input[] config = null;

        // load from file
        //

        XMLReader reader = new XMLReader();

        try {

            String file = args[0];

            System.out.println("open " + file + "...");
            world = reader.load(file);

        }
        catch (IOException io) {
            System.err.println("There was a problem reading the input " +
                "file!\n" + io.getMessage());
            System.exit(-1);
        }
        catch (ParseException pe) {
            System.err.println("There was a problem while parsing the input " +
                "file!\n" + pe.getMessage());
            System.exit(-2);
        }

//  System.out.println(world);

        // collect inputs
        Filter filter = new Filter(new String[][] {
                new String[] {Input.TRUE_INCIDENTS, Collector.AVERAGE},
                new String[] {Input.FALSE_INCIDENTS, Collector.AVERAGE},
                new String[] {Input.RAIN_SEASON, Collector.AVERAGE},
                new String[] {Input.HOSTILE_ENV, Collector.AVERAGE},
                new String[] {Input.VECTOR_CONTROL, Collector.AVERAGE},
                new String[] {Input.RESISTANCE, Collector.MINIMUM},
                new String[] {Input.GOOD_ADVICE, Collector.MAXIMUM},
                new String[] {Input.BAD_ADVICE, Collector.MAXIMUM},
        });
        world.setFilter(filter);

        world.setWriter("data/test");

        System.out.print("calculate money");
        world.newDataBlock("money");
        for (double r = 0.0; r < 10.0; r += .1) {

            Input input = new Input(Input.DRUG_COST, r);
            world.run(input);
            System.out.print(".");
        }
        System.out.println();

        System.out.print("calculate accessibility");
        world.newDataBlock("distance");
        for (double r = 0.0; r < 10.0; r += .1) {

            Input input = new Input(Input.DISTANCE, r);
            world.run(input);
            System.out.print(".");
        }
        System.out.println();

        System.out.println("done.");
    }

}
```

F.7 Village.java

```
/*
 * Created on 21.08.2004
 * $Id: Village.java 57 2004-10-07 19:22:11Z Basti $
 */

package project;
import project.input.*;

/**
 */
public class Village {

    private String name;
    private Input[] inputs;
    private Household[] households;

    // recent deaths
    private int deaths = 0;

    /**
     * constructor
     * @param name village name
     */
    public Village(String name) {
        this.name = name;
        inputs = new Input[0];
    }

    /**
     * constructor
     * @param name village name
     * @param inputs village inputs
     */
    public Village(String name, Input[] inputs) {
        this.name = name;
        this.inputs = inputs;
    }

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }

    /**
     * @return Returns the inputs.
     */
    public Input[] getInputs() {
        return inputs;
    }

    /**
     * @return Returns the hose holds.
     */
    public Household[] getHouseholds() {
        return households;
    }

    /**
     * @param households The households to set.
     */
    public void setHouseholds(Household[] households) {
        this.households = households;
    }


    /**
     * @return recent deaths in village.
     */
    public int getDeaths() {
        return deaths;
    }

    /**
     * set recent deaths tozero again.
     */
    public void resetDeaths() {
        deaths = 0;
    }

    /**
     * string representation
     */
    public String toString() {

        String hstr = "";
        for (int i = 0; i < households.length; i++)
            hstr += households[i] + ", ";

        String istr = "";
        for (int i = 0; i < inputs.length; i++)
            istr += inputs[i] + ", ";

        return name +
        " {" + hstr.substring(0, hstr.length() - 2) + "}" +
        " [" + istr.substring(0, istr.length() - 2) + "]";
    }

}
```

F.8 World.java

```
/*
 * Created on 30-Aug-2004
 * $Id: World.java 52 2004-10-07 17:48:57Z Basti $
 */

package project;

import java.io.*;
import java.util.*;

import project.mother.*;
import project.input.*;

/**
 */
public class World {

    private Village[] villages;
    private Environment[] environments;
    private Drug drug;

    private Input[] global;
    private Filter filter;

    // file writers
// private Writer mainWriter; // for the summary
// private Writer[] vilWriter; // for each village

    // initialise writer
    PrintWriter main;
    PrintWriter[] writer;

    public World(Village[] villages,
            Environment[] environments,
            Drug drug,
            Input[] global) {

        this.villages = villages;
        this.environments = environments;
        this.drug = drug;
        this.global = global;

    }

    /**
     * initialise log file writers
     * @warning This method has to be called BEFORE any simulation
     * can be run!
     */
    public void setWriter(String basename) { // Writer writer

        // initialise writer
        main = null;
        writer = new PrintWriter[villages.length];

        try {

            // main file
            main = new PrintWriter(new FileWriter(basename + ".csv"));
            main.println("# MOTHER 1.0");
            main.println("# " + basename + " (MAIN)");

            // village files
            for (int i = 0; i < writer.length; i++)
                writer[i] = new PrintWriter(new FileWriter(basename + ".vil" + i + ".csv"));

        }
        catch (IOException io) {
            System.err.println(io.getMessage());
            System.exit(−1);
        }

        // main file
        main.println("# ");
        main.println("# ENVIRONMENTS");

        for (int i = 0; i < environments.length; i++)
            main.println("# " + i + ": " + environments[i]);

        main.println("# ");
        main.println("# VILLAGES (h/holds, adults, kids)");

        Input[] inputs;
        InputTable inputTbl;

        // initialise files
        for (int v = 0; v < villages.length; v++) {

            writer[v].println("# MOTHER 1.0");
            writer[v].println("# " + basename + v);

            // collect inputs for village v
            inputTbl = new InputTable();
            inputTbl.add(global);
            inputTbl.add(villages[v].getInputs());
            inputTbl.add(Collector.collect(villages[v], environments, filter));

            // record inputs
            writer[v].println("# ");
            writer[v].println("# INPUT LIST");

            Iterator iter = inputTbl.keySet().iterator();
            while (iter.hasNext()) {
                String input = (String) iter.next();
                writer[v].println("# " + input +
                    "=" + inputTbl.get(input));
            }

            // record h/holds
            writer[v].println("# ");

            Household[] hholds = villages[v].getHouseholds();
            writer[v].println("# " + hholds.length + " HOUSE HOLDS");

            int adults = 0;
            int kids = 0;
            for (int h = 0; h < hholds.length; h++) {
                adults += hholds[h].getAdults();
                kids += hholds[h].getKids();

//    System.err.println(”H/hold ” + h + ” (adults/kids): ” +
//     hholds[h].getAdults() + ”/” + hholds[h].getKids() +
//      ” [total ” + adults + ”/” + kids + ”]”);
            }

            writer[v].println("# adults " + adults);
            writer[v].println("# children " + kids);

            // main file
            main.println("# " + v + ": " +
                villages[v].getName() + " (" +
                hholds.length + ", " +
                adults + ", " +
                kids + ")");

            // we need children to run simulation
            if (kids == 0) {
                System.err.println("No children in village " + v + "!");

                for (int i = 0; i < writer.length; i++)
                    writer[i].flush();

                System.exit(-1);
            }

        }
    }

    /**
     * starts a new data block in the output file
     * @param title title of the data block (e.g. money)
     */
    public void newDataBlock(String title) {

        // initialise CSV header
        String strPlans = title;
        for (int i = 0; i < Household.plans.length; i++)
            strPlans += ", " + Household.plans[i].getName();

        main.println("\n#\n# " + strPlans);
        for (int i = 0; i < writer.length; i++)
            writer[i].println("\n#\n# " + strPlans);

    }

    /**
     * simulate one cycle
     * @param change
     */
    public void run(Input change) { // Input[] global, Village[] vils, Environment[] envs

        Input[] inputs;
        InputTable inputTbl;

        // comparison value
//  double compValue = .1 * r;

        // storage arrays
        int[] vstore = null; // village store
        int[] mstore = new int[Household.plans.length]; // main store

        // for each village
        for (int v = 0; v < villages.length; v++) {

            // initialise village store new every time
            vstore = new int[Household.plans.length];

// System.out.println(”village ” + v + ”-------------------------- - - -”);

            // collect inputs for village v
            inputs = Input.merge_inputs(new Input[][] {
//      global,
                    villages[v].getInputs(),
                    Collector.collect(villages[v], environments, filter)}
                );

            Household[] hholds = villages[v].getHouseholds();
            for (int h = 0; h < hholds.length; h++) {

                inputTbl = new InputTable(inputs);
                inputTbl.clear();
                inputTbl.add(hholds[h].getInputs());

//    inputTbl.add(new Input(Input.DRUG_COST, compValue)); // <−−−−−−−−−−−−−−−−−−−−−−

                inputTbl.add(new Input(Input.EXPERIENCE,
                    villages[v].getDeaths()));

                // ======================================
                // add changed inputs
                // (overwrites existing inputs!)
                // ======================================
                if (change != null) inputTbl.add(change);

                // ======================================
                // review beliefs
                // ======================================
                BeliefTable beliefs = brf(inputTbl);

                beliefs.add(new Belief(Belief.FEVER, true));
                beliefs.add(new Belief(Belief.DRUG, hholds[h].hasDrug()));
                beliefs.add(new Belief(Belief.ENOUGH, hholds[h].hasEnough()));
                // beliefs.add(new Belief(Belief.SAVE_MONEY, hholds[h].hasDrug()));

//    System.err.println(”beliefs = {” + beliefs + ”}”);

//    System.err.println(inputTbl);
//    System.err.println(beliefs);

```

```
                // =====================================
                // decide
                // =====================================
                int decision = hholds[h].decide(beliefs);
                Plan plan = Household.plans[decision];

// System.out.println(”h/hold ” + h + ”: (” + decision + ”) ” +
//  Household.plans[decision].getName());

                // =====================================
                // save decision
                // =====================================
                vstore[decision]++;

                // =====================================
                // execute plan
                // =====================================
                if (plan == Plan.DO_NOTHING) {
                    // do nothing
                }
                else if (plan == Plan.BUY_FULL
                    || plan == Plan.BUY_PART) {

                    // buy drugs
                    hholds[h].buyDrug(plan == Plan.BUY_FULL ? 2 : 1);
                }
                else if (plan == Plan.GIVE_FULL
                    || plan == Plan.GIVE_PART) {
                }

//    System.out.println(”[” + plan + ”] ” +
//       Household.plans[plan].getName());

            } // end−for (h/holds)

            // store in file
            String strData = "" + change.getValue();
            for (int i = 0; i < vstore.length; i++)
                strData += ", " + vstore[i];
            writer[v].println(strData);
            writer[v].flush();

            // increase main store values
            for (int i = 0; i < vstore.length; i++)
                mstore[i] += vstore[i];

        } // end−for (villages)

        // store in main file
        String strData = "" + change.getValue();
        for (int i = 0; i < mstore.length; i++)
            strData += ", " + mstore[i];
        main.println(strData);
        main.flush();

    }

    /**
     * Belief revision function. This is the most important function. Here is
     * the evaluation of the belief network.
     * @param inputs all relevant inputs from h/hold, village, environment etc.
     * @return beliefs
     */
    public BeliefTable brf(InputTable inputs) {

        // translator mapping
        HashMap connector = new HashMap(12);
        connector.put(Input.CHILDREN, Translator.CHILDREN);
        connector.put(Input.ADULTS, Translator.ADULTS);
        connector.put(Input.INCOME, Translator.INCOME);
        connector.put(Input.DISTANCE, Translator.DISTANCE);
        connector.put(Input.CONSULTATION, Translator.CONSULTATION);
        connector.put(Input.TRUE_INCIDENTS, Translator.TRUE_INCIDENTS);
        connector.put(Input.DRUG_COST, Translator.DRUG);
        connector.put(Input.COMPLEXITY, Translator.COMPLEXITY);

        connector.put(Input.EXPERIENCE, Translator.EXPERIENCE);


        InputTable translated = new InputTable();

        Iterator iter = inputs.keySet().iterator();
        while (iter.hasNext()) {
            Object obj = iter.next();
//   System.err.println(obj.getClass() + ”: ’” + obj + ”’”);

            String input = (String) obj;
            double value = inputs.get(input);

            // translator exists?
            if (connector.containsKey(input)) {

                // get corresponding translator
                Translator trans = (Translator) connector.get(input);

                translated.add(new Input(
                    input, trans.translate(value)));

            }
            // otherwise simply copy over
            else {

                translated.add(new Input(
                    input, value));

            }
        }

        BeliefTable beliefs = new BeliefTable();

        beliefs.add(new Belief(Belief.ACCESSIBLE,
            (translated.get(Input.AVAILABILITY) -
            translated.get(Input.CONSULTATION) -
            translated.get(Input.DISTANCE)) / 3
        ));

        // is evaluated outside this method
        beliefs.add(new Belief(Belief.AFFORDABLE,
            translated.get(Input.DRUG_COST) -
            translated.get(Input.CONSULTATION) -
            translated.get(Input.DISPOSABLE)
        ));

        beliefs.add(new Belief(Belief.BAD_MEDICINE,
            (4 * translated.get(Input.SIDE_EFFECTS) +
            translated.get(Input.EXPERIENCE) -
//   translated.get(Input.TRADITION) // TODO this belong to it somehow, too!
            translated.get(Input.BAD_ADVICE)) / 4
        ));

        beliefs.add(new Belief(Belief.EFFECTIVE,
            (translated.get(Input.ANTI_PYRETIC) -
            translated.get(Input.BATCH_FAILURE) -
            translated.get(Input.COMPLEXITY) +
            translated.get(Input.EFFICACY) +
            translated.get(Input.FORGIVENESS)) / 5
        ));

        beliefs.add(new Belief(Belief.SERIOUS,
            (translated.get(Input.TRUE_INCIDENTS) +
            translated.get(Input.EXPERIENCE) -
            translated.get(Input.CHILDREN) -
            translated.get(Input.TRUE_INCIDENTS)) / 4
        ));

        beliefs.add(new Belief(Belief.TRAINED,
            (translated.get(Input.GOOD_ADVICE) -
            translated.get(Input.BAD_ADVICE))/2
        ));

        Random random = new Random();

        beliefs = new BeliefTable(new Belief[] {
                new Belief(Belief.SERIOUS, random.nextDouble()),
                new Belief(Belief.AFFORDABLE, random.nextDouble()),
                new Belief(Belief.ACCESSIBLE, random.nextDouble()),
                new Belief(Belief.EFFECTIVE, random.nextDouble()),
                new Belief(Belief.TRAINED, random.nextDouble()),
                new Belief(Belief.BAD_MEDICINE, Belief.FALSE),

                new Belief(Belief.FEVER, Belief.TRUE),
                new Belief(Belief.ENOUGH, random.nextDouble()),
                new Belief(Belief.DRUG, random.nextBoolean()),
                new Belief(Belief.SAVE_MONEY, random.nextDouble()),
        });

        return beliefs;
    }

    /**
     * @param filter The filter to set.
     */
    public void setFilter(Filter filter) {
        this.filter = filter;
    }

    /**
     * string representation
     */
    public String toString() {

        String str = "";

        str += "Villages (" + villages.length + "):";
        for (int i = 0; i < villages.length; i++)
            str += "\n" + villages[i];
```

```java

        str += "\nEnvironments (" + environments.length + "):";
        for (int i = 0; i < environments.length; i++)
            str += "\n" + environments[i];

        str += "\nDrug:";
        str += "\n" + drug;

//  str += ”\nGlobal inputs (” + global.length + ”):”;
//  for (int i = 0; i < global.length; i++)
//    str += ”\n” + global[i];

        return str;
    }

}
```

F.9 XMLReader.java

```java
/*
 * Created on 16.09.2004
 * $Id: XMLReader.java 56 2004-10-07 19:17:58Z Basti $
 */
package project;

import project.input.*;
import project.input.Reader;

import java.io.*;
import java.util.HashMap;

import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

/**
 */
public class XMLReader extends Reader {

    // h/hold inputs
    public static final String CHILDREN = "children";
    public static final String ADULTS = "adults";
    public static final String TRADITION = "tradition";
    public static final String INCOME = "income";
    public static final String DISPOSABLE = "disposable";

    // village inputs
    public static final String DISTANCE = "distance";
    public static final String AVAILABILITY = "availability";
    public static final String CONSULTATION = "consultation";
    public static final String EXPERIENCE = "experience";

    // env inputs
    public static final String TRUE_INCIDENTS = "true_incidents";
    public static final String FALSE_INCIDENTS = "false_incidents";
    public static final String RAIN_SEASON = "rain_season";
    public static final String HOSTILE_ENV = "hostile_env";
    public static final String VECTOR_CONTROL = "vector_control";
    public static final String RESISTANCE = "resistance";
    public static final String GOOD_ADVICE = "good_advice";
    public static final String BAD_ADVICE = "bad_advice";

    // drug inputs
    public static final String DRUG_COST = "cost";
    public static final String EFFICACY = "efficacy";
    public static final String SIDE_EFFECTS = "side_effects";
    public static final String FORGIVENESS = "forgiveness";
    public static final String COMPLEXITY = "complexity";
    public static final String ANTI_PYRETIC = "ap_effect";
    public static final String BATCH_FAILURE = "batch_failure";

    private DocumentBuilderFactory factory;
    private DocumentBuilder builder;
    private Document document;

    private Input[] config = null;

    /**
     * default constructor
     */
    public XMLReader() {

        try {
            factory = DocumentBuilderFactory.newInstance();
            builder = factory.newDocumentBuilder();
        }
        catch (ParserConfigurationException pce) {
            System.err.println("Couldn’t initialise XML parser!\n" +
                pce.getMessage());
            System.exit(-1);
        }
    }

    /* (non-Javadoc)
     * @see input.Reader#open(java.lang.String)
     */
    public World load(String filename) throws IOException, ParseException {

        try {
            // parse XML file
            document = builder.parse(new File(filename));
        }
        catch (SAXException se) {
            throw new ParseException(se.getMessage());
        }

        NodeList list;
        int total;

        Environment[] envs = null;
        Village[] vils = null;
        Drug drug = null;

        // maps village id → village object
        HashMap map = new HashMap();

        try {

            // get villages
            //
//   System.err.println(”Villages:”);

            list = document.getElementsByTagName("village");

            total = list.getLength();
            if (total <= 0) throw new NotFoundException("village");

            // initialise array
            vils = new Village[total];

            for(int s = 0; s < list.getLength(); s++) {

                Element element = (Element) list.item(s);
                Village vil = parseVillage(element);

                String id = element.getAttribute("id");

//    System.err.println(”\t” + id + ”: ’” + vil.getName() + ”’”);
                Input[] inputs = vil.getInputs();
//    for (int i = 0; i < inputs.length; i++)
//     System.err.println(”\t\t[” + inputs[i] + ”]”);

                if (id == null || id.equals(""))
                    throw new ParseException("Village " + s + " has no ID!");

                vils[s] = vil;
                map.put(id, vil); // save to map
            }


            // get environments
            //

            list = document.getElementsByTagName("environment");

            total = list.getLength();
            if (total <= 0) throw new NotFoundException("environment");

            // initialise array
            envs = new Environment[total];

            for(int s = 0; s < list.getLength(); s++) {

                Element element = (Element) list.item(s);
                Environment env = parseEnvironment(element);

                String id = element.getAttribute("id");
                String content = element.getFirstChild().getNodeValue();

//    System.err.println(id + ”: ” + env);
//    System.err.println(”contains :” + content.trim());

                String[] parts = content.trim().split(",");
                for (int i = 0; i < parts.length; i++) {
                    String part = parts[i].trim();
                    if (map.containsKey(part))
                        env.addVillage((Village) map.get(part));
                    else throw new ParseException(part +
                        " not found while parsing environment " + id);
                }

                envs[s] = env;
            }

            // get drug
            //

            list = document.getElementsByTagName("drug");

            total = list.getLength();
            if (total <= 0)
                throw new NotFoundException("drug");
            else if (total > 1)
                throw new ParseException("drug");

            // initialise
            drug = parseDrug((Element) list.item(0));

        }
        catch (NotFoundException nf) {
            System.err.println("Didn't find tag '" + nf.getMessage() + "'!\n");
        }

        // otherwise
        return new World(vils, envs, drug, config);
    }

    /* (non-Javadoc)
     * @see input.Reader#getConfig()
     */
    public Input[] getConfig() throws NotFoundException, IOException {
        // @TODO implement configuration fetch method
        return null;
    }

    /**
     * parse node to get drug data
     * @param node element node
     * @return drug generated from node
     * @throw s ParseException
     */
    private Drug parseDrug(Element node) throws ParseException {

        if(node.getNodeType() == Node.ELEMENT_NODE) {

            String[] attributes = new String[] {
                DRUG_COST,
                EFFICACY,
                SIDE_EFFECTS,
                FORGIVENESS,
                COMPLEXITY,
                ANTI_PYRETIC,
                BATCH_FAILURE,
            };

            String[] inputs = new String[] {
                Input.DRUG_COST,
                Input.EFFICACY,
                Input.SIDE_EFFECTS,
                Input.FORGIVENESS,
                Input.COMPLEXITY,
                Input.ANTI_PYRETIC,
                Input.BATCH_FAILURE,
            };

            String name = node.getAttribute("name");
            Input[] props = getAttributes(node, attributes, inputs);

            return new Drug(name,
                props); // inputs

        }

        return null;
    }

    /**
     * parse node to get environment data
     * @param node element node
     * @return environment generated from node
     * @throw s ParseException
     */
    private Environment parseEnvironment(Element node) throws ParseException {

        if(node.getNodeType() == Node.ELEMENT_NODE) {

//   System.err.println("Environment " + node.getAttribute("id") +
//     " (" + node.getAttributes().getLength() + " attrs.)");

            String[] attributes = new String[] {
                TRUE_INCIDENTS,
                FALSE_INCIDENTS,
                RAIN_SEASON,
                HOSTILE_ENV,
                VECTOR_CONTROL,
                RESISTANCE,
                GOOD_ADVICE,
                BAD_ADVICE,
            };

            String[] inputs = new String[] {
                Input.TRUE_INCIDENTS,
                Input.FALSE_INCIDENTS,
                Input.RAIN_SEASON,
                Input.HOSTILE_ENV,
                Input.VECTOR_CONTROL,
                Input.RESISTANCE,
                Input.GOOD_ADVICE,
                Input.BAD_ADVICE,
            };

            Input[] props = getAttributes(node, attributes, inputs);

            return new Environment(
                new Village[] {}, // no villages for now
                props); // inputs

        }

        return null;
    }

    /**
     * parse node to get village data
     * @param node element node
     * @return village generated from node
     * @throw s ParseException
     */
    private Village parseVillage(Element node) throws ParseException {

        if (node.getNodeType() == Node.ELEMENT_NODE) {

//   System.err.println("Village " + node.getAttribute("id") +
//     " (" + node.getAttributes().getLength() + " attrs.)");

            String[] attributes = new String[] {
                DISTANCE,
                AVAILABILITY,
                CONSULTATION,
                EXPERIENCE,
            };

            String[] inputs = new String[] {
                Input.DISTANCE,
                Input.AVAILABILITY,
                Input.CONSULTATION,
                Input.EXPERIENCE,
            };

            String name = node.getAttribute("id");
            Input[] props = getAttributes(node, attributes, inputs);
            Village vil = new Village(name,
                props); // inputs

            int badEntries = 0;
            Household[] hholds = new Household[node.getChildNodes().getLength()];
            for (int i = 0; i < node.getChildNodes().getLength(); i++) {
                if (node.getChildNodes().item(i).getNodeType() == Node.ELEMENT_NODE) {
//      System.err.println(node.getChildNodes().item(i).getNodeName());
                    hholds[i] = parseHousehold((Element) node.getChildNodes().item(i));
                }
                else badEntries++;
            }

            // are there any null entries?
            if (badEntries > 0) {

//    System.err.println("+++" + badEntries + " bad entries!");

                Household[] dummy = new Household[hholds.length − badEntries];
                int count = 0;

                for (int i = 0; i < hholds.length; i++)
                    if (hholds[i] != null)
                        dummy[count++] = hholds[i];

                hholds = dummy;
            }

            vil.setHouseholds(hholds);

            return vil;

        }

        return null;
    }

    /**
     * parse node to get house hold data
     * @param node element node
     * @return house hold generated from node
     * @throw s ParseException
     */
    private Household parseHousehold(Element node) throws ParseException {

        if (node.getNodeType() == Node.ELEMENT_NODE) {

//  System.err.println("Village " + node.getAttribute("id") +
//    " (" + node.getAttributes().getLength() + " attrs.)");

            String[] attributes = new String[] {
                CHILDREN,
                ADULTS,
                TRADITION,
                INCOME,
                DISPOSABLE,
            };

            String[] inputs = new String[] {
                Input.CHILDREN,
                Input.ADULTS,
                Input.TRADITION,
                Input.INCOME,
                Input.DISPOSABLE,
            };

            Input[] props = getAttributes(node, attributes, inputs);
            Household hhold = new Household(props);

            double kids;
            double adults;
            double income;
            double dispIncome;

            kids = parseValue(node.getAttribute(CHILDREN));
            adults = parseValue(node.getAttribute(ADULTS));
            income = parseValue(node.getAttribute(INCOME));
            dispIncome = parseValue(node.getAttribute(DISPOSABLE));

            hhold.setKids((int) kids);
            hhold.setAdults((int) adults);
            hhold.setIncome((int) income);
            hhold.setDispIncome((int) dispIncome);

            return hhold;

        }

        return null;
    }

    /**
     * get all attributes from element node
     * @param node element node
     * @param tags XML tags to look for
     * @param inputs corresponding inputs
     * @return list of inputs from attributes
     * @throw s ParseException
     * @attention tags and inputs must have the
     * same length and the entries must correspond to each other!
     */
    private Input[] getAttributes(Element node,
        String[] tags, String[] inputs) throws ParseException {
```

```

// NamedNodeMap attrList = node.getAttributes();
// Input[] props = new Input[attrList.getLength()];
        Input[] props = new Input[tags.length];

        for (int i = 0; i < tags.length; i++) {

            try {

                Attr attr = node.getAttributeNode(tags[i]);
                if (attr == null) throw new NullPointerException();

//    System.err.println(tags[i] + ”=” +
//     attr.getValue());

                props[i] = new Input(
                    inputs[i],
                    Double.parseDouble(attr.getValue().trim()));

            }
            catch (NullPointerException npe) {
                System.err.println("Warning: missing attribute " + tags[i] +
                    " in " + node.getNodeName() +
                    " ’" + node.getAttribute("id") + "’");

                props[i] = new Input(inputs[i], 0.0);
            }
            catch (NumberFormatException nfe) {
                System.err.println("Warning: wrong format for attribute " +
                    tags[i] + " in " + node.getNodeName() +
                    " ’" + node.getAttribute("id") + "’! " +
                    "(0 substituted for ’" +
                    node.getAttributeNode(tags[i]).getValue().trim() + "’)");

                props[i] = new Input(inputs[i], 0.0);

//    throw new ParseException(”attribute ” + tags[i] +
//    ” has the wrong format” +
//    ” in ” + node.getNodeName() +
//    ” ” + node.getAttribute(”id”));
            }
        }

        return props;
    }

    /**
     * parse attribute value
     * @param value
     * @return double value if parsable, 0.0 otherwise
     */
    private double parseValue(String value) {

        try {
            return Double.parseDouble(value.trim());
        }
        catch (NumberFormatException nfe) {
            return 0.0;
        }

    }

    /**
     * test method
     * @param args command line arguments
     */
    public static void main(String[] args) {

        XMLReader reader = new XMLReader();

        World world = null;
        Input[] config = null;

        try {

            String file = "data/test_scenario.xml";

            System.out.println("open " + file + "...");
            world = reader.load(file);

            config = reader.getConfig();

        }
        catch (IOException io) {
            System.err.println("There was a problem reading the input " +
                "file!\n" + io.getMessage());
            System.exit(-1);
        }
        catch (ParseException pe) {
            System.err.println("There was a problem while parsing the input " +
                "file!\n" + pe.getMessage());
            System.exit(-2);
        }
        catch (NotFoundException nf) {
            System.err.println("Didn’t find tag ’" + nf.getMessage() + "’!\n");
        }
    }
}
```

F.10 input/Filter.java

```
/*
 * Created on 31-Aug-2004
 * $Id: Filter.java 10 2004-09-27 09:31:20Z Basti $
 */
package project.input;

import java.util.Hashtable;

/**
 * ”Filter rule for environmental input clash
 * e.g. <pre>
 *  ”true incident rate” → ”avg”
 *  ”resistance” → ”low”
 *  ...
 * </pre>
 */
public class Filter extends Hashtable {

    public Filter() {
        super();
    }

    public Filter(String input, String method) {
        super();
        put(input, method);
    }

    public Filter(String[][] rules) {
        super();
        for (int i = 0; i < rules.length; i++)
            put(rules[i][0], rules[i][1]);
    }

    public String get(String key) {
        return (String) super.get(key);
    }
}
```

F.11 input/Input.java

```
/*
 * Created on 21.08.2004
 * $Id: Input.java 47 2004-10-05 15:43:33Z Basti $
 */
package project.input;

/**
 */
public class Input {

    private String name;
    private double value;

    // Inputs with *) are saved locally!
    //

    // h/hold inputs
    public static final String CHILDREN = "number of children";
    public static final String ADULTS = "number of adults";
    public static final String TRADITION = "traditional beliefs";
    public static final String INCOME = "income";
    public static final String DISPOSABLE = "disposable income";

    // village inputs
    public static final String DISTANCE = "distance";
    public static final String AVAILABILITY = "availability";
    public static final String CONSULTATION = "consultation cost";
    public static final String EXPERIENCE = "recent negative experiece"; // *)

    // env inputs
    public static final String TRUE_INCIDENTS = "true incident rate";
    public static final String FALSE_INCIDENTS = "false incident rate";
    public static final String RAIN_SEASON = "rain season";
    public static final String HOSTILE_ENV = "hostile environment";
    public static final String VECTOR_CONTROL = "vector control";
    public static final String RESISTANCE = "resistance";
    public static final String GOOD_ADVICE = "qualified advice";
    public static final String BAD_ADVICE = "non-qualified advice";

    // drug inputs
    public static final String DRUG_COST = "drug cost";
    public static final String EFFICACY = "efficacy";
    public static final String SIDE_EFFECTS = "side effects";
    public static final String FORGIVENESS = "forgiveness";
    public static final String COMPLEXITY = "complexity";
    public static final String ANTI_PYRETIC = "anti-pyretic effect";
    public static final String BATCH_FAILURE = "batch failure";

    /**
     * default constructor
     * @param name
     * @param value
     */
    public Input(String name, double value) {

        this.name = name;
        this.value = value;
    }

    /**
     * @return returns the name.
     */
    public String getName() {
        return name;
    }
    /**
     * @param name The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * @return Returns the value.
     */
    public double getValue() {
        return value;
    }
    /**
     * @param value The value to set.
     */
    public void setValue(double value) {
        this.value = value;
    }

    /**
     * merge two or more input arrays
     * @warning Does not check for duplicates!
     * @param inputs array of input arrays
     * @return one array
     * @todo What happens if one or more arrays = null?
     */
    public static Input[] merge_inputs(
            Input[][] inputs) {

        // get length of target array
        int length = 0;
        for (int i =0; i < inputs.length; i++)
            length += (inputs[i] != null) ? inputs[i].length : 0;
        Input[] list = new Input[length];

        // merge
        int count = 0;
        for (int i = 0; i < inputs.length; i++) {
            for (int j = 0; j < inputs[i].length; j++)
                list[count + j] = inputs[i][j];
            count += inputs[i].length;
        }

        return list;
    }

    public String toString() {
        return name + "=" + value;
    }

}
```

F.12 input/InputStack.java

```
/*
 * Created on 21.08.2004
 * $Id: InputStack.java 10 2004−09−27 09:31:20Z Basti $
 */
package project.input;

import java.util.ArrayList;

/**
 */
public class InputStack extends ArrayList {

    private int count;

    public InputStack () {
        super();
        count = 0;
    }

    public void push(Input input) {

        add(input);
        count++;
    }

    public Input pop() {
        return (Input) get(count−−);
    }

    public int size() {
        return count;
    }

    public static void main(String[] args) {
    }
}
```

F.13 input/InputTable.java

```
/*
 * Created on 25.09.2004
 * $Id: InputTable.java 48 2004−10−05 15:43:59Z Basti $
 */
package project.input;

import java.util.Hashtable;

/**
```

```
 */
public class InputTable extends Hashtable {

    /**
     * default constructor
     */
    public InputTable() {
        super();
    }

    /**
     * constructor from array of Inputs
     * @param inputs array of inputs
     */
    public InputTable(Input[] inputs) {
        super();

        for (int i = 0; i < inputs.length; i++)
            add(inputs[i]);
    }

    /**
     * add input to list
     * @param input input to be added
     */
    public void add(Input input) {

        String key = input.getName();
        Double val = new Double(input.getValue());

        this.put(key, val);
    }

    /**
     * add list of input
     * @param inputs inputs to be added
     */
    public void add(Input[] inputs) {

        if (inputs == null) return;

        for (int i = 0; i < inputs.length; i++)
            add(inputs[i]);
    }

    /**
     * get an input from the table
     * @param name input name
     * @return corresponding input for name iff in the table,
     * null otherwise
     */
    public Input getInput(String name) {

        if (!this.contains(name)) return null;

        Double value = (Double) super.get(name);
        return new Input(name, value.doubleValue());
    }

    /**
     * get an input value from the table
     * @param name input name
     * @return corresponding value for name iff in the table
     */
    public double get(String name) {

        if (!this.containsKey(name))
            return 0.0;

//  System.err.println(name + ”: ” + super.isEmpty());
//  System.err.println(name + ”: ” + super.size());
//  System.err.println(name + ”: ” + super.get(name).getClass());

        Double value = (Double) super.get(name);
        return value.doubleValue();
    }

    /**
     *
     * @param name
     * @return
     */
    public boolean contains(String name) {
        return (!this.containsKey(name));
    }

    /**
     *
     * @param input
     * @return
     */
    public boolean contains(Input input) {
        return contains(input.getName());
    }

}
```

F.14 input/NotFoundException.java

```
/*
 * Created on 16.09.2004
 * $Id: NotFoundException.java 15 2004−09−27 10:04:09Z Basti $
 */
package project.input;

/**
 * Standard exception thrown if a requested field is not found by the
 * Reader class.
 */
public class NotFoundException extends Exception {

    /**
     * @param missing missing entry
     */
    public NotFoundException(String missing) {
        super(missing);
        // TODO Auto−generated constructor stub
    }

    public NotFoundException() {
        super("?");
    }

}
```

F.15 input/ParseException.java

```
/*
 * Created on 16.09.2004
 * $Id: ParseException.java 15 2004−09−27 10:04:09Z Basti $
 */
package project.input;

/**
 * Standard exception thrown if a requested field is not found by the
 * Reader class.
 */
public class ParseException extends Exception {

    /**
     * @param missing missing entry
     */
    public ParseException(String missing) {
        super(missing);
        // TODO Auto−generated constructor stub
    }

    public ParseException() {
        super("?");
    }

}
```

F.16 input/Reader.java

```
/*
 * Created on 16.09.2004
 * $Id: Reader.java 30 2004−09−27 21:24:12Z Basti $
 */
package project.input;

import java.io.IOException;

import project.∗;

/**
 */
public abstract class Reader {

// public abstract void open(String filename)
//  throws IOException, ParseException;
// public abstract void close() throws IOException;

    public abstract World load(String filename)
        throws IOException, ParseException;

    public abstract Input[] getConfig()
        throws NotFoundException, IOException;

// public abstract Environment[] getEnvironments()
//  throws NotFoundException, IOException;
// public abstract Village[] getVillages()
//  throws NotFoundException, IOException;

}
```

F.17 input/Stack.java

```java
/*
 * Created on 23.08.2004
 * $Id: Stack.java 10 2004-09-27 09:31:20Z Basti $
 */

package project.input;
import java.util.ArrayList;

/**
 */
public class Stack extends ArrayList {

    private int counter;

    /**
     * default constructor
     */
    public Stack() {
        super();
        counter = 0;
    }

    public Stack(double item) {
        this();
        this.push(item);
    }

    public void push(double val) {
        this.add(new Double(val));
        counter++;
    }

    public double pop() throws Exception {

        if (counter <= 0) {
            throw new Exception("Stack access violation! Stack is empty!");
        }

        Double item = (Double) this.get(--counter);
        return item.doubleValue();
    }

    public String toString() {

        if (counter == 0)
            return "[empty]";

        String str = "[";
        for (int i = 0; i < counter; i++)
            str += this.get(i) + ", ";

        return str.substring(0, str.length() - 2) + "]";
    }

    public double[] getValues() {

        double[] values = new double[counter];
        for (int i = 0; i < counter; i++) {
            Double item = (Double) this.get(i);
            values[i] = item.doubleValue();
        }
        return values;
    }

    /**
     * test method
     * @param args
     */
    public static void main(String args[]) {

        Stack stack = new Stack();

        for (double i = 0; i < 10; i++) {
            System.out.print("adding " + i + ": ");
            stack.push(i);
            System.out.println(stack);
        }

        try {
            for (double i = 0; i < 11; i++) {
                System.out.print("removing ");
                System.out.print(stack.pop() + ": ");
                System.out.println(stack);
            }
        }
        catch (Exception e) {
            System.err.println("ERROR! " + e.getMessage());
        }

    }

}
```

F.18 input/Translator.java

```
/*
 * Created on 31-Aug-2004
 * $Id: Translator.java 10 2004-09-27 09:31:20Z Basti $
 */
package project.input;

import java.util.Random;

/**
 * Translator for inputs. The Traslator class provides a method that translates
 * each input into a value between 0 and 1.
 *
 * Example:
 *
 *   number of children → {1-2, 2-7, 7+}
 *   1 = .3
 *   5 = .6
 *   9 = 1.0
 *
 * <pre>
 * Translator children = new Translator() {
 *   public double translate(double value) {
 *    if (value <= 0) return 0.0;
 *    if (value >= 1 && value <= 2) return 1/3;
 *    if (value >= 2 && value <= 7) return 2/3;
 *    return 1.0; // otherwise
 *   }
 * };
 * </pre>
 */
public abstract class Translator {

    /**
     * default translator for
     * number of children → {1−2, 2−7, 7+}
     */
    public final static Translator CHILDREN = new Translator() {
        public double translate(double value) {
            if (value <= 0.0) return 0.0;
            if (value >= 1.0 && value <= 2.0) return .3333;
            if (value >= 2.0 && value <= 7.0) return .6666;
            return 1.0; // otherwise
        }
    };

    /**
     * default translator for
     * number of adults → {1−2, 2−7, 7+}
     */
    public final static Translator ADULTS = new Translator() {
        public double translate(double value) {
            if (value <= 0.0) return 0.0;
            if (value >= 1.0 && value <= 2.0) return .3333;
            if (value >= 2.0 && value <= 7.0) return .6666;
            return 1.0; // otherwise
        }
    };

    /**
     * default translator for
     * income → {poor, ..., rich}
     * TODO what are the boundaries?
     */
    public final static Translator INCOME = new Translator() {
        public double translate(double value) {
            if (value <= 0.0) return 0.0;
            if (value >= 1.0 && value <= 2.0) return .3333;
            if (value >= 2.0 && value <= 7.0) return .6666;
            return 1.0; // otherwise
        }
    };

    /**
     * default translator for
     * distance → {<2km, 2−5km, >5km}
     */
    public final static Translator DISTANCE = new Translator() {
        public double translate(double value) {
            if (value < 2.0) return 0.0;
            if (value >= 2 && value <= 5.0) return .5;
            return 1.0; // otherwise
        }
    };

    /**
     * default translator for
     * consultation cost → {nominal (<1), moderate (1−2), expensive (>2)}
     */
    public final static Translator CONSULTATION = new Translator() {
```

```
    public double translate(double value) {
        if (value < 1.0) return 0.0;
        if (value >= 1.0 && value <= 2.0) return .5;
        return 1.0; // otherwise
    }
};

/**
 * default translator for
 * true incident rate → {low transmission, normal trans., high trans.}
 * TODO what are the boundaries?
 */
public final static Translator TRUE_INCIDENTS = new Translator() {
    public double translate(double value) {
        if (value <= 0.0) return 0.0;
        if (value >= 1.0 && value <= 2.0) return .3333;
        if (value >= 2.0 && value <= 7.0) return .6666;
        return 1.0; // otherwise
    }
};

/**
 * default translator for
 * drug cost → {nominal (<1), moderate (1−2), expensive (>2)}
 */
public final static Translator DRUG = new Translator() {
    public double translate(double value) {
        if (value < 1.0) return 0.0;
        if (value >= 1.0 && value <= 2.0) return .5;
        return 1.0; // otherwise
    }
};

/**
 * default translator for
 * complexity of regimen →
 * {simple ↦ 0.2, typical ↦ 0.4, complex ↦ 1.0}
 */
public final static Translator COMPLEXITY = new Translator() {
    public double translate(double value) {
        if (value <= 0.3333) return .2;
        if (value <= 0.6666) return .4;
        return 1.0; // otherwise
    }
};

/**
 * default translator for
 * recent negative experience (=recent deaths in village) →
 * {0, 1−2, 3−5, 5−9, 9+}
 */
public final static Translator EXPERIENCE = new Translator() {
    public double translate(double value) {
        if (value == 0) return .0;
        if (value <= 2.0) return .25;
        if (value <= 5.0) return .5;
        if (value <= 9.0) return .75;
        return 1.0; // otherwise
    }
};
```

```
    /**
     * default constructor
     */
    public Translator() {}

    public abstract double translate(double value);

    /**
     * main methods
     * @param args command line arguments
     */
    public static void main(String[] args) {

//  Translator children = new Translator() {
//   public double translate(double value) {
//    if (value <= 0.0) return 0.0;
//    if (value >= 1.0 && value <= 2.0) return .3;
//    if (value >= 2.0 && value <= 7.0) return .6;
//    return 1.0; // otherwise
//   }
//  };
//
        Random random = new Random();
        for (int i = 0; i < 50; i++) {
            int kids = random.nextInt(12);
            System.out.println(kids + " -→  " + CHILDREN.translate(kids));
        }
    }

}
```

F.19 output/StorageException.java

```
/*
 * Created on 19.09.2004
 * $Id: StorageException.java 15 2004-09-27 10:04:09Z Basti $
 */
package project.output;

/**
 */
public class StorageException extends Exception {

    public StorageException() {
        super();
    }

    public StorageException(String msg) {
        super(msg);
    }

}
```

F.20 output/Writer.java

```
/*
 * Created on 31-Aug-2004
 * $Id: Writer.java 10 2004-09-27 09:31:20Z Basti $
 */
package project.output;
```

```java

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Date;

/**
 */
public class Writer {

    public static final int PLAN_DO_NOTHING = 0;
    public static final int PLAN_BUY_AP = 1;
    public static final int PLAN_BUY_PART = 2;
    public static final int PLAN_BUY_FULL = 3;
    public static final int PLAN_HEALTH_CARE = 4;
    public static final int PLAN_GIVE_PART = 5;
    public static final int PLAN_GIVE_FULL = 6;

    private double[] store;
    private String basename;
    private PrintWriter[] writer;

    /**
     * default constructor
     */
    public Writer(String name) {
        basename = name;
        store = new double[7];
    }

    public String getBasename() {
        return basename;
    }

// public void put(int identifier, double value) {
//  // increase value
//  store[identifier] += value;
// }
//
// public double get(int identifier) {
//  return store[identifier];
// }
//
// public int length() {
//  return store.length;
// }
//
// public void clear() {
//  for (int i = 0; i < store.length; i++)
//   store[i] = 0;
// }
//
// public abstract void open(int nr) throws StorageException;
// public abstract void close() throws StorageException;
//
// public abstract void flushLine() throws StorageException;
// public abstract void flush() throws StorageException;
    public void open(int nr) throws StorageException {

        writer = new PrintWriter[nr + 1];

        try {

            writer[0] = new PrintWriter(
                new FileWriter(getBasename(), true));

            for (int i = 0; i < nr; i++) {
                writer[i + 1] = new PrintWriter(
                    new FileWriter(getBasename() + i, true));
            }

//   writer.print("# Generation");
//   for (int i = 0; i < Census.DATA_ITEMS; i++)
//    writer.print(", " + Census.label[i]);
//   writer.println();

        }
        catch (IOException e) {
            throw(new StorageException(
                "Couldn't create '" + getBasename() + "'!"));
        }

    }

    public void init() {

        writer[0].println("# Mother v1.0");
        writer[0].println("# -------------------------------------------");
        writer[0].println("# ");
        writer[0].println("# PARAMETERS USED");
        writer[0].println("# -------------------------------------------");
        writer[0].println("# ");

        Date start = new Date(System.currentTimeMillis());
        writer[0].println("# SIMULATION STARTED AT");
        writer[0].println("# " + start);
        writer[0].println("# -------------------------------------------");
        writer[0].println("# ");

    }

    public void close() {

//  flushLine();
//
//  Date end = new Date(System.currentTimeMillis());
//  writer.println("# -------------------------------------------");
//  writer.println("# SIMULATION TERMINATED AT");
//  writer.println("# " + end);
//  writer.println("# ");
//
//  // flush buffer
//  writer.flush();
    }
}
```

F.21 mother/Belief.java

```
/*
 * Created on 29-Aug-2004
 * $Id: Belief.java 10 2004-09-27 09:31:20Z Basti $
 */
package project.mother;

```

```
/**
 */
public class Belief {

    private String name;
    private double plausibility;

    public final static double TRUE = 1.0;
    public final static double FALSE = 0.0;

    /**
     * extend to which the mother thinks a belief to be true (TODO again!)
     */
    public final static double PLAUSIBLE = .7;

    //
    // beliefs
    //
    public final static String FEVER = "fever?";
    public final static String ENOUGH = "enough?";
    public final static String DRUG = "drug?";
    public final static String SAVE_MONEY = "save money?";

    public final static String SERIOUS = "serious?";
    public final static String AFFORDABLE = "affordable?";
    public final static String ACCESSIBLE = "accessible?";
    public final static String EFFECTIVE = "effective?";
    public final static String TRAINED = "trained?";
    public final static String BAD_MEDICINE = "bad medicine?";

    /**
     * default constructor
     * @param name name of belief
     */
    public Belief(String name) {
        this.name = name;
        this.plausibility = TRUE;
    }

    /**
     * default constructor
     * @param name name of belief
     * @param pl plausibility
     */
    public Belief(String name, double pl) {
        this.name = name;
        this.plausibility = pl;
    }

    /**
     * default constructor
     * @param name name of belief
     * @param pl plausibility
     */
    public Belief(String name, boolean pl) {
        this.name = name;
        this.plausibility = pl ? TRUE : FALSE;
    }

    /**
     * @return true if the belief is plausible enough,
     * false otherwise.
     */
    public boolean isPlausible() {
        return isPlausible(PLAUSIBLE);
    }

    /**
     * @param pl plausibility rate
     * @return true if the belief is plausible enough (i.e.
     * more than pl), false otherwise.
     */
    public boolean isPlausible(double pl) {
        return (plausibility >= pl);
    }

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }
    /**
     * @param name The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * @return Returns the plausibility.
     */
    public double getPlausibility() {
        return plausibility;
    }
    /**
     * @param plausibility The plausibility to set.
     */
    public void setPlausibility(double plausibility) {
        this.plausibility = plausibility;
    }

    /**
     * string representation
     */
    public String toString() {
        return (isPlausible() ? "" : "") + name + " (" + plausibility + ")";
    }
}
```

F.22 mother/BeliefTable.java

```
/*
 * Created on 30-Aug-2004
 * $Id: BeliefTable.java 10 2004-09-27 09:31:20Z Basti $
 */
package project.mother;

import java.util.*;

/**
 */
public class BeliefTable extends Hashtable {

    /**
```

```
     * default constructor
     */
    public BeliefTable() {
        super();
    }

    /**
     *
     * @param beliefs
     */
    public BeliefTable(Belief[] beliefs) {
        super();

        for (int i = 0; i < beliefs.length; i++)
            add(beliefs[i]);
    }

    public BeliefTable(String[] beliefs) {
        super();

        for (int i = 0; i < beliefs.length; i++)
            add(new Belief(beliefs[i], Belief.TRUE));
    }

    /**
     *
     * @param belief
     */
    public void add(Belief belief) {

        String key = belief.getName();
        Double val = new Double(belief.getPlausibility());

// if (!this.containsKey(key)) {
// }
// else
        this.put(key, val);
    }

    /**
     *
     * @param name
     * @return
     */
    public Belief get(String name) {

        Double value = (Double) super.get(name);
        return new Belief(name, value.doubleValue());
    }

    /**
     * XXX complete descr.
     * @param name
     * @return
     */
    public boolean contains(String name) {

        if (!this.containsKey(name))
            return false;

        return this.get(name).isPlausible();
    }

    /**
     * @param belief belief to be checked
     * @return true if belief matches a belief in
     * the table (i.e. even belief), false otherwise.
     */
    public boolean contains(Belief belief) {
        return belief.isPlausible() == contains(belief.getName());
    }

    /**
     *
     * @param beliefs
     * @return returns the proportion of matched beliefs in the table
     */
    public double matchRate(Belief[] beliefs) {

// System.err.println(”beliefs:”);
// for (int i = 0; i < beliefs.length; i++)
// System.err.println(beliefs[i]);
// System.err.println(”\nbelief table [” + this.size() + ” items]:”);
// System.err.println(this);

        if (this.size() == 0) return 0.0;

        int matches = 0;
        for (int i = 0; i < beliefs.length; i++)
            if (contains(beliefs[i])) ++matches;

// System.err.println(matches + ” / ” + this.size());

        return matches / (double) this.size();
    }

    /**
     * string representation
     */
    public String toString() {

        String key, str = "";
        Belief belief;

        for (Enumeration keys = keys(); keys.hasMoreElements() ;) {
            key = (String) keys.nextElement();
            belief = get(key);

// str += (belief.isPlausible() ? ”” : ””) + belief.getName() + ”\n”;
            str += (belief.isPlausible() ? "" : "") + belief.getName() + " ^ ";
        }

        return str.substring(0, str.length() - 3);
    }

    /**
     * test method
     */
    public static void main(String[] args) {

        BeliefTable table = new BeliefTable(new Belief[] {
                new Belief("ill", Belief.FALSE),
                new Belief("serious", Belief.FALSE),
                new Belief("save money"),
        });

        Belief[] beliefs = new Belief[] {
                new Belief("ill", Belief.FALSE),
                new Belief("serious", Belief.TRUE),
                new Belief("trained", Belief.FALSE),
                new Belief("effective"),
                new Belief("effective", Belief.FALSE),
        };


        System.out.println("belief table:\n" + table);

        for (int i = 0; i < beliefs.length; i++)
            System.out.println("table contains " +
                    beliefs[i] + "? " + table.contains(beliefs[i]));

    }

}
```

F.23 mother/Plan.java

```
/*
 * Created on 23.08.2004
 * $Id: Plan.java 54 2004−10−07 17:50:35Z Basti $
 */
package project.mother;

/**
 */
public abstract class Plan {

    private String name;
    private BeliefTable postcondition;

    /**
     * default plan
     */
    public static Plan DO_NOTHING = new Plan("do nothing",
            new BeliefTable(new Belief[] {})) {
        public boolean triggers(BeliefTable beliefs) {
            // always triggers
            return true;
        }
    };

    /**
     * mother buys only an anti−pyretic drug
     */
    public static Plan BUY_AP = new Plan("buy anti-pyretic drug",
            new BeliefTable(new Belief[] {
                    new Belief(Belief.FEVER, Belief.FALSE),
                    new Belief(Belief.SAVE_MONEY),
            })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && !beliefs.contains(Belief.SERIOUS)
                && !beliefs.contains(Belief.DRUG)
                && !beliefs.contains(Belief.ENOUGH)
                && (!beliefs.contains(Belief.AFFORDABLE)
                        || !beliefs.contains(Belief.ACCESSIBLE))
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };

    /**
     * mother buys part of an anti-malaria treatment
     */
    public static Plan BUY_PART = new Plan("buy part anti-malarial treatment",
            new BeliefTable(new Belief[] {
                new Belief(Belief.SERIOUS, Belief.FALSE),
                new Belief(Belief.FEVER, Belief.FALSE),
        })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && beliefs.contains(Belief.SERIOUS)
                && !beliefs.contains(Belief.DRUG)
                && !beliefs.contains(Belief.ENOUGH)
                && !beliefs.contains(Belief.AFFORDABLE)
                && beliefs.contains(Belief.TRAINED)
//  && beliefs.contains(Belief.EFFECTIVE)
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };

    /**
     * mother buys a complete anti-malaria treatment
     */
    public static Plan BUY_FULL = new Plan("buy full anti-malarial treatment",
            new BeliefTable(new Belief[] {
                new Belief(Belief.SERIOUS, Belief.FALSE),
                new Belief(Belief.FEVER, Belief.FALSE),
        })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && beliefs.contains(Belief.SERIOUS)
                && (!beliefs.contains(Belief.DRUG)
                    || !beliefs.contains(Belief.ENOUGH))
                && (beliefs.contains(Belief.AFFORDABLE)
                        || beliefs.contains(Belief.TRAINED))
                && beliefs.contains(Belief.EFFECTIVE)
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };

    /**
     * mother takes her child to a health care facility
     */
    public static Plan HEALTH_CARE = new Plan("go to health care facility",
            new BeliefTable(new Belief[] {
                    new Belief(Belief.SERIOUS, Belief.FALSE),
                    new Belief(Belief.FEVER, Belief.FALSE),
            })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && beliefs.contains(Belief.SERIOUS)
                && beliefs.contains(Belief.AFFORDABLE)
                && beliefs.contains(Belief.ACCESSIBLE)
                && beliefs.contains(Belief.TRAINED)
                && beliefs.contains(Belief.EFFECTIVE)
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };

    /**
     * mother gives her child <i>only part</i> of the treatment
     */
    public static Plan GIVE_PART = new Plan("give part course",
            new BeliefTable(new Belief[] {
                    new Belief(Belief.FEVER, Belief.FALSE),
                    new Belief(Belief.SERIOUS, Belief.FALSE),
                    new Belief(Belief.SAVE_MONEY),
            })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && (beliefs.contains(Belief.ENOUGH)
                    || beliefs.contains(Belief.DRUG))
                && !beliefs.contains(Belief.SERIOUS)
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };

    /**
     * mother gives her child the treatment as prescribed
     */
    public static Plan GIVE_FULL = new Plan("give full course",
            new BeliefTable(new Belief[] {
                    new Belief(Belief.FEVER, Belief.FALSE),
                    new Belief(Belief.SERIOUS, Belief.FALSE),
            })) {
        public boolean triggers(BeliefTable beliefs) {
            return beliefs.contains(Belief.FEVER)
                && beliefs.contains(Belief.ENOUGH)
                && beliefs.contains(Belief.TRAINED)
                && !beliefs.contains(Belief.BAD_MEDICINE);
        }
    };


    /**
     * default constructor
     * @param name short description of plan
     * @param postcondition what the mother hopes for
     */
    public Plan(String name, BeliefTable postcondition) {

        this.name = name;
        this.postcondition = postcondition;
    }

    public boolean triggers(Belief[] beliefs) {
        BeliefTable table = new BeliefTable(beliefs);
        return triggers(table);
    }

    /**
     * checks whether a plan can be chosen (because its precondition
     * is fulfilled.
     * @return true if the precodition evaluates to
     *  true.
     */
    public abstract boolean triggers(BeliefTable beliefs);

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }
    /**
     * @return Returns the postcondition.
     */
    public BeliefTable getPostcondition() {
        return postcondition;
    }
    /**
     * @param name The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * @param postcondition The postcondition to set.
     */
    public void setPostcondition(BeliefTable postcondition) {
        this.postcondition = postcondition;
    }

    /**
     * @param beliefs
     * @return the utility of the plan (as compared to the outcome hoped for)
     */
    public double getUtility(Belief[] beliefs) {
        return postcondition.matchRate(beliefs);
    }

    /**
     * test method
     */
    public static void main(String args[]) {

        Plan[] plans = new Plan[] {
                DO_NOTHING,
                BUY_AP,
                BUY_PART,
                BUY_FULL,
                HEALTH_CARE,
                GIVE_PART,
                GIVE_FULL
        };

        BeliefTable beliefs = new BeliefTable(
                new String[] {
// Belief.FEVER, ”trained”, ”affordable”
                        Belief.SERIOUS, Belief.AFFORDABLE, Belief.ACCESSIBLE,
                        Belief.TRAINED, Belief.EFFECTIVE
                });
        Belief[] hopes = new Belief[] {
                new Belief(Belief.FEVER, Belief.FALSE),
                new Belief(Belief.SERIOUS, Belief.FALSE),
                new Belief(Belief.SAVE_MONEY),
        };

        System.out.println("The following plans trigger:");
        for (int i = 0; i < plans.length; i++)
            if (plans[i].triggers(beliefs))
                System.out.println(plans[i].getName() +
                        " (utility=" + plans[i].getUtility(hopes) + ")");

    }
}
```

F.24 test/PlanTable.java

```
/*
 * Created on 09.09.2004
 * $Id: PlanTable.java 10 2004-09-27 09:31:20Z Basti $
 */

package test;

import project.*;
import project.mother.*;

import java.io.*;
import java.util.ArrayList;

/**
 * This class generates the outcome of every possible combination of beliefs
 * and saves the result in LaTeX format into 2 files, which are specified as
 * arguments (args[0] and args[1]). The first file
 * contains a table with all beliefs and the plans they trigger, the second
 * lists all possible combination of beliefs for each plan in order.
 *
 * Example: java PlanTable plans-unsorted.tex plans-sorted.tex
 *
 * @attention The following LaTeX packages are needed to generate the output:
 * - longtable
 * - rotating.
 */
public class PlanTable {

    /**
     * Sort array in ascending order using binary conversion.
     * Double entries will be thrown out.
     * @param list array list with belief cominations
     * @attention The changes are made in the object straight away!
     */
    private static void sort(ArrayList list) {

        Entry[] old = new Entry[list.size()];
        for (int i = 0; i < list.size(); i++)
            old[i] = (Entry) list.get(i);

        int changes = 1;
        int same = 0;

        Entry dummy;
        while (changes > 0) {

            changes = 0;

            for (int i = 0; i < old.length - 1; i++) {

                if (old[i] != null && old[i + 1] != null)
                    if (old[i].binary > old[i + 1].binary) {

                        dummy = old[i];
                        old[i] = old[i + 1];
                    old[i + 1] = dummy;

                    changes++;
                }
                else if (old[i].binary == old[i + 1].binary) {
                    old[i] = null;
                    same++;
                }
            }
        }

// System.err.println(”same=” + same);

        list.clear();

        for (int i = 0; i < old.length; i++)
            if (old[i] != null) list.add(old[i]);

    }

    /**
     * @param b string representation of belief array
     * @return corresponding LaTeX string
     */
    private static String translate(String b) {

        String logic = "";
        for (int i = 0; i < b.length(); i++) {
            logic += (b.charAt(i) == ’1’) ? "$\\bullet$" : "";
            logic += " & ";
        }
        return logic.substring(0, logic.length() − 3) + " \\\\";
    }

    /**
     * main method
     * @param args program parameters. args[0] and
     *  args[0] both contain file names.
     * @warning Exits if args.length != 2!
     */
    public static void main(String[] args) {

        if (args.length != 2) {
            System.err.println("usage: java PlanTable <unsorted.tex> <sorted.tex>");
            System.exit(−1);
        }

        String sorted, unsorted;

        Plan[] plans = new Plan[] {
                Plan.DO_NOTHING,
                Plan.BUY_AP,
                Plan.BUY_PART,
                Plan.BUY_FULL,
                Plan.HEALTH_CARE,
                Plan.GIVE_PART,
                Plan.GIVE_FULL
        };

        Household mother = new Household();

        String[] b = new String[] {
```

```
    Belief.FEVER,
    Belief.DRUG,
    Belief.ENOUGH,
    Belief.SAVE_MONEY,
    Belief.SERIOUS,
    Belief.AFFORDABLE,
    Belief.ACCESSIBLE,
    Belief.EFFECTIVE,
    Belief.TRAINED,
    Belief.BAD_MEDICINE,
};

//
// table headers
//
String head = "\\tablehead{%";

for (int i = 0; i < b.length - 1; i++) {
    head += "\n\\belief{" + b[i] + "} & ";
}
head += "\n\\belief{" + b[b.length - 1] + "} \\\\" +
    "\\hline}" +
    "\n\\tablelasttail{%"+
    "\n \\hline}" +
    "\n\\tabletail{%" +
    "\n\\hline" +
    "\n%\\multicolumn{10}{r}{continues on the next page\\ldots}"+
    "\n}";

// plan storage (unsorted)
ArrayList list = new ArrayList();

// plan storage (sorted)
ArrayList[] plist = new ArrayList[plans.length];
for (int i = 0; i < plist.length; i++)
    plist[i] = new ArrayList();

System.out.print("generating maternal response");

double total = Math.pow(2, b.length); // 2^10 (with 10 beliefs)
int mod = Math.round((float) total / 10);

for (int c = 0; c < total; c++) {

    // progress bar
    if (c % mod == 0) System.out.print('.');

    String binary = Integer.toBinaryString(c);
    // bring all strings to same length
    int diff = b.length - binary.length();
    if (diff > 0)
        for (int i = 0; i < diff; i++)
            binary = '0' + binary;

    // if |enough?| == true then |drug?| = true
    if (binary.charAt(2) == '1')
        binary = binary.substring(0, 1) + '1' + binary.substring(2);

    BeliefTable beliefs = new BeliefTable();

    for (int i = 0; i < binary.length(); i++)
        beliefs.add(new Belief(b[i], binary.charAt(i) == '1'));
```

```

            int plan = mother.decide(beliefs);

            list.add(new Entry(binary, plan));
            plist[plan].add(new Entry(binary, plan));
        }

        sorted = head + "\n\n\\begin{supertabular}{|c|c|c|c|c|c|c|c|c|c|}";
        unsorted = head + "\n\n\\begin{supertabular}{|c|c|c|c|c|c|c|c|c|c|l}";

//  sort(list);
        for (int i = 0; i < list.size(); i++) {
            Entry entry = (Entry) list.get(i);
            String trans = translate(entry.beliefs);
            unsorted += "\n" + trans.substring(0, trans.length()− 3) +
                " & " + plans[entry.plan].getName() + " \\\\";
        }

        // counter
        int ucount = list.size();

        unsorted += "\n\\end{supertabular}";

        System.out.print("\nsorting results");

        int scount = 0;
        for (int i = 0; i < plist.length; i++) {

            System.out.print(’.’);

            sorted += "\n\\hline\\multicolumn{10}{|l|}{" + plans[i].getName() +
                "} \\\\\\hline";

            sort(plist[i]);
            scount += plist[i].size();

            for (int j = 0; j < plist[i].size(); j++) {
                Entry entry = (Entry) plist[i].get(j);
                if (j % 2 != 0) sorted += "\\LR";
                sorted += "\n" + translate(entry.beliefs);
            }

        }

        sorted += "\n\\end{supertabular}";

        System.out.println("\nwriting to files...");

        try {

            System.out.println(ucount + " combinations [" + args[0] + "]");
            FileWriter file = new FileWriter(args[0]);
            file.write(unsorted);
            file.flush();
            file.close();

            System.out.println(scount + " different combinations [" + args[1] + "]");
            file = new FileWriter(args[1]);
            file.write(sorted);
            file.flush();
            file.close();
        }
        catch (IOException e) {

            e.printStackTrace();
            System.exit(-2);
        }

    }
}

class Entry {

    public String beliefs;
    public int plan;
    public int binary; // integer value of binary belief represetation

    public Entry(String b, int p) {
        beliefs = b;
        plan = p;

        binary = Integer.parseInt(beliefs, 2);
    }

    public String toString() {
        return beliefs + " → " + plan;
    }
}

```

F.25 test/Ray.java

```
/*
 * Created on 23.08.2004
 * $Id: Ray.java 55 2004-10-07 17:52:53Z Basti $
 */

package test;

import project.*;
import project.input.*;

/**
 */
public class Ray {

    public static void main(String[] args) {

        Village vil = new Village("Deuringen", new Input[] {
                new Input(Input.DISTANCE, 2.2),
                new Input(Input.CONSULTATION, 4.5)});
        Village vil2 = new Village("Steppach");

        Environment[] envs = new Environment[] {
                new Environment(vil, new Input[] {
                        new Input(Input.TRUE_INCIDENTS, 4.5),
                        new Input(Input.FALSE_INCIDENTS, 4.5/7),
                        new Input(Input.RESISTANCE, .75)}),
                new Environment(new Village[] {vil, vil2}, new Input[] {
                        new Input(Input.TRUE_INCIDENTS, 2.5),
                        new Input(Input.RESISTANCE, .5)}),
        };

        // collect inputs
        Filter filter = new Filter(new String[][] {
                new String[] {Input.TRUE_INCIDENTS, Collector.AVERAGE},
                new String[] {Input.FALSE_INCIDENTS, Collector.AVERAGE},
                new String[] {Input.RAIN_SEASON, Collector.AVERAGE},
                new String[] {Input.HOSTILE_ENV, Collector.AVERAGE},
                new String[] {Input.VECTOR_CONTROL, Collector.AVERAGE},
                new String[] {Input.RESISTANCE, Collector.MINIMUM},
                new String[] {Input.GOOD_ADVICE, Collector.MAXIMUM},
                new String[] {Input.BAD_ADVICE, Collector.MAXIMUM},
        });

        Drug drug = new Drug();

        Input[] inputs = Input.merge_inputs(new Input[][] {
                new Input[] {},
                vil.getInputs(),
                Collector.collect(vil, envs, filter)});

        World world = new World(
                God.populate(
                  new Village[] {vil, vil2},
                  20, // households
                  4.5, // kids
                  2.5, // adults
                  100, // income
                  .10), // disp. income
                envs, drug, inputs);
        world.setFilter(filter)
        ;

        world.setWriter("data/ray");

        // run the simulation
        while (true) { // until the end of time
          world.run(new Input(Input.ADULTS, 2.0));
        }

    }
}
```

Appendix G

CD-ROM Content

```
|
+- mother/
+- doc/
    +- html/
    +- latex/
+- report/
    +- index.html
    +- report.pdf
```