AN EARLY WARNING SYSTEM FOR AMBIENT ASSISTED LIVING

June 2012

Andrea Monacchi

Master of Science in Computer Science

AN EARLY WARNING SYSTEM FOR AMBIENT ASSISTED LIVING

Andrea Monacchi
Master of Science
Computer Science
June 2012
School of Computer Science
Reykjavík University

M.Sc. PROJECT REPORT

ISSN 1670-8539

An Early Warning System for Ambient Assisted Living

by

Andrea Monacchi

Project report submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

June 2012

Project Report Committee:

Hannes Högni Vilhjálmsson, Supervisor
Associate Professor, Reykjavik University

Emanuela Merelli, Supervisor
Professor, University of Camerino

Stephan Schiffel
Postdoctoral Research Associate, Reykjavik University

The undersigned hereby certify that they recommend to the School of Computer Science at Reykjavík University for acceptance this project report entitled **An Early Warning System for Ambient Assisted Living** submitted by **Andrea Monacchi** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

_____

Date

_____

Hannes Högni Vilhjálmsson, Supervisor
Associate Professor, Reykjavik University

_____

Emanuela Merelli, Supervisor
Professor, University of Camerino

_____

Stephan Schiffel
Postdoctoral Research Associate, Reykjavik University

_____

Date

_____

Andrea Monacchi
Master of Science

An Early Warning System for Ambient Assisted Living

Andrea Monacchi

June 2012

Abstract

Daily-life activities at home can generate dangers that may lead to accidents. Risky situations may be difficult to notice by people with a cognitive or physical impairment. Therefore, recognizing dangers is very important so as to assist users in preventing accidents, and ensure their health, safety and well-being.

The present thesis aims to design a system that, given a representation of the environment as input, learns how to evaluate states according to their danger level, and is able to alert and prevent users from getting too close to a potential danger. We explore the search space for disclosing dangers and finding a safe path leading to the goal. The project led to the implementation of a working prototype, which is able to suggest the best action to perform, and reports the level of danger and an evaluation of the last performed action. Also, it is able to warn the user when the level of danger exceeds a given threshold. We offer a *general* solution, as the system is able to play arbitrary games described with the Game Description Language, and perform on-line planning by means of the Q(λ) algorithm. For this purpose, we implemented a Java library for implementing TD-learning agents. In addition, we defined the concept of sphere of protection and we disclose dangers by using a variant of breadth-first search. Finally, we exploited virtual environments as a general testbed for simulating effects of warning notifications and we conducted informal user testing for evaluating the effects of warning notifications on actual users.

To my family and all people living away from home

Acknowledgements

Every work is the result of time and effort. This is the place where I can finally give thanks to all people who supported me somehow.

First of all, I would not be here without my parents, I will always be grateful for the open-minded setting in which I was raised and for the love they feel for me. My mother helped me during my first approach to a computer, while my father tought me more than every professor about practical mechanics and electronics. They developed my way of thinking and always do the best for my future. Moreover, I want to thank my syster for sharing with me dreams and hopes. Indeed, my family supported me in every way when I decided to come to Iceland for a year.

Likewise, I would like to thank my close relatives: my grandmothers and my aunts.

I spent an amazing time in Iceland. My mates David, Lillo and Alfredo deserve my mention here, as they made my stay much more enjoyable. In particular, I want to thank David for our everlasting arguments about Computer Science, we shared our passion for two years of our life and I hope I will have the chance to work with you in the future, as one of the most qualified persons I ever met.

Regarding this work, It would not have been possible for me to come here without the double degree programme. Therefore, I want to thank both University of Camerino and Reykjavik for this invaluable possibility. In particular, I am grateful to my supervisors Emanuela Merelli and Hannes Högni for offering to me the possibility to work at CADIA for a project related to my personal interests.

I would also like to thank Stephan Schiffel for his invaluable tutoring and for showing so much passion for Artificial Intelligence and General Game Playing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As life expectancy has increased significantly producing a change in the percentage of elderly people, there is an increasing concern about active ageing. Ageing denotes the natural process of physical, psychological and social change of individuals. In 2002, The World Health Organisation adopted the term "Active" to express the concept of ageing with safety and social participation. The United Nations identified specific dimensions behind this concept: dignity, independence, participation, equity, safety, appeasement, recognition [TGM11].
Indeed, nowadays many elderly people live on their own. Therefore, it may be necessary to assist them due to a cognitive or physical impairment, which means monitoring their activities and ensuring their health, safety and well-being.
Pervasive and ubiquitous technologies can be useful tools to entertain, monitor, assist and automate people's tasks in smart environments. Thus, assistive technologies allow in-place ageing, and consequently improve the quality of life and help reducing costs of dedicated care-givers for the institutions. In this project, we focus on domestic scenarios to help elderly people in their everyday life towards the above presented dimensions of active ageing.

1.1 Overall project

This work is conceived as part of a larger project in Assisted Living[1]. The main project idea is a real-time tracking and simulation of a home environment to identify and avert

[1] *Assisted Living* provides supervision or assistance with activities of daily living; coordination of services by outside health care providers; and monitoring of resident activities to help to ensure their health, safety, and well-being. http://wikipedia.org

potentially dangerous situations. The general approach is to embed a range of sensors in

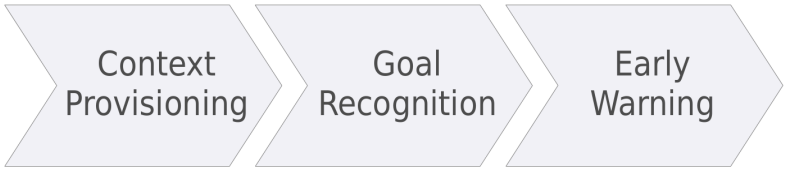

Figure 1.1: Project overview

objects and appliances, in order to allow context awareness. By exploiting the context, a state of the real world is described in an abstract language.
Eventually, a simulation, that simulates both the physics of objects and the behaviour of the human, runs a number of steps into the future, and watches for possible dangerous states (as defined by given rules). If the current behaviour of the human seems to be leading towards a disaster (according to the simulation), then the human is alerted through voice, sounds or projected visuals.
For this purpose, the goal of the user needs to be defined; conjoint use of activity recognition and prediction is exploited to achieve this.

1.2 Research statement

Daily-life activities at home can generate dangers that may lead to accidents. Risky situations are even more difficult to notice by elderly people and people with a disease. Therefore, recognizing dangers and alerting users is very important so as to assist them in preventing accidents. This requires to keep track of environment changes in order to predict intentions that are leading to risky situations. For taking a snapshot of the setting, we need to observe users and monitor the physical environment by means of sensors. We assume we already have a way to recognize the user's current goal (e.g. he wants to cook and this requires the pot to be on the stove) and the environment state in terms of physical environment (e.g. temperature), as well as user high-level information (e.g. identity, position). In this way, we can reduce the problem of preventing dangerous situations to a search problem where we explore the search space for disclosing dangers and finding a safe path leading to the goal.
The problem of finding a safe path leading to a goal configuration is a complex decision problem that requires a way to estimate the danger level of states. Such a complex decision problem can be modeled as a Markov Decision Process. Accordingly, the system may alert the user as soon as he gets too close to a risky state. Indeed, MDPs are a powerful tool for finding an optimal policy that maximizes a certain utility or performance

measure (i.e. the danger level).
Furthermore, HCI researchers have been working on the problem of notification alerts in dangerous situations and they have used multimodal interfaces for interacting with the user in a more natural way. This work can be a useful reference to understand how users react to alert cues and what research methods can be used for testing user's satisfaction, as well as effectiveness and efficiency of systems.

The area of early warning systems received considerable attention from the Ambient Assisted Living research community. However, there still is a lack of approaches that aim to solve this problem. Therefore, we propose the following research question:

How is it possible to design a system that, given a representation of the environment as input, learns how to evaluate states according to their danger level, and is able to alert and prevent users from getting too close to a potential danger? For addressing this question we exploit knowledge representation techniques in order to represent the environment in terms of satisfied properties. This is useful both for keeping track of changes and for simulating effects of actions (e.g. when using search algorithms). Moreover, we implement a decision maker that is able to learn an evaluation function for representing the desirability of a certain situation. Thus, we specify this problem as a Markov Decision Process and we use reinforcement learning to compute a policy. This policy describes the behaviour of the decision maker and is able to take the danger level into account for warning the user beforehand and aiding the user to achieve his goal. We are implementing an early warning system as a decision maker that guides the user during his everyday life activities. Since we want to build a useful system that meets all requirements, we show how the consistence of warning messages and the degree of intrusiveness can be evaluated. Therefore, we propose a complete methodology for the evaluating systems in terms of effectiveness and users' satisfaction.

1.3 Organization

In chapter 2, we start by reporting some background information about context-aware computing.

In chapter 3, we answer part of the research question by reporting some information about knowledge representation.

In chapter 4, we show how to model complex decision-making problems by means of Markov Decision Processes and reinforcement learning algorithms.

In chapter 5, we rely on some related work for showing projects that are using similar approaches and technologies.

In chapter 6, we design a complete early warning system and we discuss a design methodology within a complete framework for smart applications.

In chapter 7, we report the steps that led to the implementation of a prototype of an early warning system.

In chapter 8, we present a methodology for evaluating the presented solution.

In chapter 9, we discuss our results and we suggest further developments to this work.

Chapter 2

Context-aware computing

According to Mark Weiser's vision, "the most profound technologies are those that disappear" [Wei99]. Computing systems are becoming pervasive in our daily life and services are present ubiquitously, as they can be accessed everywhere. Indeed, traditional interaction paradigms such as the WIMP (windows, icons, menus and pointing devices) are too obtrusive and demanding of human attention, as they tend to divert us from the task. In this chapter we report a short survey about Context-aware computing, that is, the use of situational information for automating tasks and minimizing the interaction with users.

2.1 Context-awareness

Context-awareness is the main way to produce unobtrusive systems towards the concept of calm and disappearing computer coined by Weiser [WB97]. Context-aware systems aim to build an approximate representation of the human intent in order to act properly. When humans speak to each other, they are able to use situational information. Unfortunately, this is not possible in human-computer interaction as computers are not able to understand and take advantage of this information. Thus, we need to explicitly specify it during the interaction. The first solution is to improve the interaction by using different modalities, though this does not solve the need to explicitly interact with the system. Therefore, the solution is to use situational information in order to automate the interaction [Kru09]. This is called implicit human-computer interaction (iHCI) and can occur with different degrees of autonomy. [Pos09] mentions active context-awareness when the system acts on behalf of the user, automatically adapting the environment based on the context. On

the contrary, passive context-aware systems report the current context to the user without any adaptation (e.g. after a deviation from the original setting).

The first to define the concept of context were Schilit *et al.* in [SAW94]. They refer to context as computing environment (i.e. computing resources), user environment (e.g. identity and preferences) and physical environment (e.g. location and temperature). A discussion of six different types of representation for the context is proposed in [SP04]. The authors concluded that ontology is the best representation for the context, though low-resource computing systems might be unable to process them.

2.2 Context adaptation

Context-aware applications dynamically adapt their behaviour to changing situations. Those systems are able to autonomously perform a sequence of actions that lead the environment to the desired context. Autonomous systems are self-governing[1] systems that are capable of their own independent decisions and actions, that is, they are designed to adhere to a policy or to achieve a goal. Therefore, users only need to specify high level tasks or goals, while the system will plan the set of low-level tasks needed, thus with a reduced complexity for the interaction [Pos09]. Planning and acting in nondeterministic domains may occur on the belief state space, or use online replanning for taking changes to the environment model into account. However, a single-path solution from a deterministic planner may be weak under strong uncertainty. Therefore, decision making often is addressed by means of Markov Decision Processes (MDPs) where the solution is a policy that describes the behaviour in every possible state [RN10].
As a matter of fact, designing decision makers involves different approaches: rule-based methods (i.e. reactive agents based on event-condition-action rules), model-based methods (i.e. autonomous planning agents that search over a model of the environment for computing the best path of actions) and machine-learning techniques [Kru09]. Since different users have very different expectations and preferences, planning approaches can not provide optimal control strategies, unless the designer constructs customized models. Consequently, machine-learning techniques offer the best means for tailoring a service to end-users and autonomously adapt to the inhabitants [CD04]. In this context, many learning algorithms have been used. Supervised-learning techniques such as neural networks [Moz98] were used to learn a model of the user by means of a training set that describes his behaviour and preferences. However, models often are not available a priori and defin-

[1] IBM refers to *Autonomic Computing* [KC03] as self-managing software architectures (e.g. MAPE-k) implementing a closed control loop by means of policies [Dar09].

ing a good and complete policy by means of a training set may not be possible. Moreover, adaptive methods that learn user's preferences may prefer performing continuous learning rather than off-line learning, as the user may change his behaviour over time.
Reinforcement learning algorithms [SB98] can be used for learning a model from the interaction with users, where a delayed reward function is used for defining the utility or cost of reaching certain outcomes. Reinforcement learning algorithms have been used in [MM98, KWA10, CYH+03] for learning an optimal policy to control resources such as lights or heating systems. In the related work, we propose a more detailed survey of using reinforcement learning and Markov Decision Processes for Smart Environments.

2.3 Context prediction

Context-aware systems can be enhanced by temporal models and the use of prediction, that is, the computation of properties (or features) of a future state given the evidence (i.e. all observations) to date. For instance, user's next activity could be inferred by using a dynamic Bayesian network[2], and the environment could be proactively prepared for that (e.g. the user is likely going to be in his bedroom for sleeping, therefore we switch the heating system on for that room).
Indeed, the predicted information can be exploited for enabling proactive adaptation in services and applications, so as to reduce the interaction with users by anticipating potential and future requests. Predicted context information has been used for [Boy11, NMF05]:

- **Preventive reconfiguration.** Configuration task such as loading libraries or applications can be done right before the user is about to need of them.
- **Device power management.** Appliances are managed taking user's habits into account in order to save energy.
- **Early warning of possible problems.** When a system is about to enter in a dangerous state (e.g. network overload), an early adaptation could avoid the problem.
- **Aid the user to achieve the desired goal.** A room could be prepared in advance to the activity the user is going to perform there.
- **Early coordination of individuals.** If the needs of several users of a group can be predicted, the system can satisfy the interest of the group.

[2] A *dynamic Bayesian network* is a graphical model that represents a set of random variables and their probabilistic relationships over the time. A complete survey of probabilistic reasoning methods over the time is presented in [RN10].

An architecture for performing practical experiments with context-prediction is described in [May04]. It provides to client applications a set of loadable modules that implements

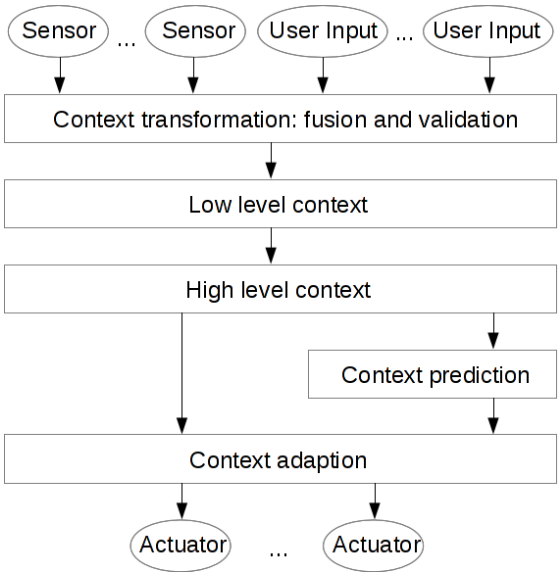

Figure 2.1: An architecture for a context-aware system

several predictor models, as the literature survey presented did not result in one algorithm showing clear advantages over other alternatives. An overview of context prediction approaches and applications is reported in [Boy11]. However, this kind of systems has been criticized as they easily distract the user, thus offering a worse user experience [NMF05].

Chapter 3

The Game Description Language

The Game Description Language (GDL) is a language used for describing the rules of games in the General Game Playing[1] context [GLP05] in a compact and high-level declarative way. It is a variant of first order logic and it is purely axiomatic, as no arithmetics is included in the language and it has to be defined in the game description. In its first version it allows the modeling of deterministic and fully observable games, whereas imperfect information can be directly expressed in the second version. According to [Thi11], GDL allows the concise and high-level specification of arbitrary finite games, and thus it can be considered complete for the purpose of General Game Playing. Moreover, as games are used for modeling multiagent systems, it can be used as a general description language that allows reasoning about the intentions of the other players (or agents). An axiomatization for a multiagent environment using the Game Description Language is proposed in [ST09]. The authors show how the Game Description Language can be seen as a declarative language for producing compact specifications of multiagent environments, as well as how autonomous agents can automatically learn how to participate in a multiagent society from the game rules and without the need of being re-programmed.

Games are modeled as state machines where a state is a set of true facts or properties called *fluents*. Playing a match consists of selecting a move for each role so as to apply the joint action to the state-machine and move to a different state. Therefore, a game starts in a predefined initial state and players select moves until the game reaches a terminal state.
Game rules (i.e. the transition function) are described using logical rules that define the next state as a modification of the current one. Accordingly, the *next* rule can be used for describing effects of an action performed by a certain player. Therefore, *does* can

[1] The General Game Playing aims at implementing intelligent agents able to learn to play previously unknown games given their rules.

role(?r)	?r is a player
init(?f)	?f holds in the initial position
true(?f)	?f holds in the current position
legal(?r,?m)	the role ?r can perform the move ?m
does(?r,?m)	player ?r does move ?m
next(?f)	?f holds in the next position
terminal	the state is terminal
goal(?r,?v)	the role ?r gets the reward ?v
sees(?r,?p)	the role ?r perceives ?p in the next turn
random	the random player

Table 3.1: GDL keywords
[Thi11]

be used for narrowing consequences to the actual action performed. Obviously, applicability of actions depends on the context, that is, the structure of the state in terms of holding properties constrains the applicability of actions. GDL provides the *legal* rule for specifying what a certain player can perform in a given context. Indeed, designers may specify multi Agent environments by means of the *role* rule, and use *goal* for defining the reward that each player would get in a certain terminal state. The game usually starts in a initial state. For this purpose, we can use *init* to specify the facts that hold in the initial configuration. Likewise, we can use *terminal* for specifying absorbing states where the game ends and the players should receive their reward [LHH+08]. The table 3.1 reports the GDL keywords for the Game Description Language. The *sees* relation can be used for specifying partially observable environments where a player can explicitly specify the amount of information to disclose to other agents.

There are some requirements that GDL descriptions have to satisfy in order to be well formed [LHH+08]:

Definition 1 (Termination)
A game description in GDL terminates if all infinite sequences of legal moves from the initial state of the game reach a terminal state after a finite number of steps.

Definition 2 (Playability)
A game description in GDL is playable if and only if every role has at least one legal move in every non-terminal state reachable from the initial state.

Definition 3 (Monotonicity)
A game description in GDL is monotonic if and only if every role has exactly one goal value in every state reachable from the initial state, and goal values never decrease.

Definition 4 (Winnability)
A game description in GDL is strongly winnable if and only if, for some role, there is a sequence of individual moves of that role that leads to a terminal state of the game where that role's goal value is maximal. A game description in GDL is weakly winnable if and only if, for every role, there is a sequence of joint moves of all roles that leads to a terminal state where that role's goal value is maximal.

Definition 5 (Well-formed games)
A game description in GDL is well-formed if it terminates, is monotonic, and is both playable and weakly winnable.

Chapter 4

Learning to make complex decisions

Intelligent Agents perceive the state of the environment through sensors and act on it by means of actuators (fig. 4.1) [RN10]. Decision making is the cognitive process that

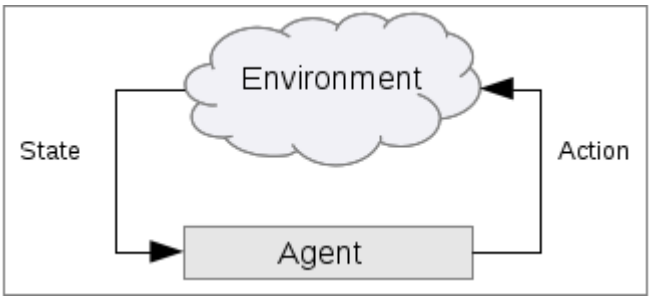

Figure 4.1: The classic perception-action loop

consists of making a choice among several alternatives. A rational agent acts on the environment in order to achieve the highest possible score or utility. It therefore selects actions that maximize its performance measure according to the expected outcome of actions.
Simple decision makers deal with episodic environments, where action selection is based on the immediate utility of actions. Therefore, preferences are expressed by a utility function which maps each state to a real value so as to produce an ordering over the set of applicable actions for a certain state.
Reactive and planning agents may not be enough to deal with complex environments, as the limited number of sensors offers a restricted and potentially noisy view of the environment. Indeed, the real world is a stochastic environment, and actions may have nondeterministic outcomes due to unexpected effects. Therefore, taking the same action in the same state in different occasions may lead to different states and different scores. For the sake of simplicity, we usually consider the environment as stationary (or slowly-varying non-stationary), where transition probabilities do not change over time.

Furthermore, utility may also depend on a sequence of decisions, in other words, the decision maker interacts with the environment over a sequence of time steps. Therefore, it selects an action according to the environment state and gets a numerical reward from the environment as a consequence of its outcome. Thus, the behaviour of the agent is defined as the mapping from states to probabilities of selecting each possible action (fig. 4.2). This behaviour, called policy, stricly depends on the agent's goal and can be computed in

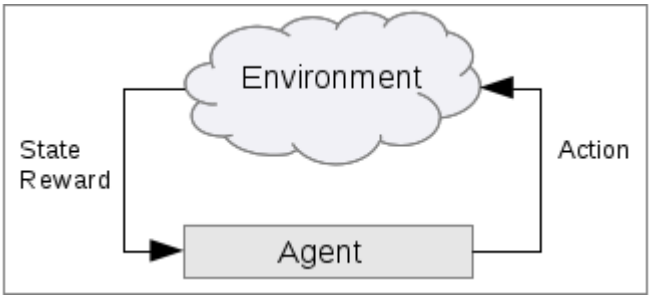

Figure 4.2: The Markov-decision-process agent

order to maximize the total reward. Similarly, a reward function produces a numeric value that expresses desirability of states and thus the agent's goal. The reward is the way to express what the agent should achieve, that is which conditions are associated to a reward or a penalty.

However, the reward expresses the immediate desirability of a state while a value function specifies the long-term desirability of states, that is, the expected utility that we would possibly and eventually get when passing through a certain state. Thus, the reward signal is handled by the task environment, whereas the value function is estimated by the agent over time as a result of its experience.

Action-value estimation has been achieved through different approaches such as optimization methods (e.g. genetic algorithms and genetic programming), dynamic programming [Bel57] and reinforcement learning. However, optimization methods only work for small state spaces as they completely ignore the problem and they can not take advantage from the interaction with the environment in order to drive the search over the space of policies. A classical alternative consists of changing the value of states while interacting with the environment. This is a form of learning called reinforcement learning. Indeed, each action has a value representing the expected reward given that the action is selected. The so called n-armed bandit problem describes the action selection problem as a n-levers slot machine. Accordingly, the player can maintain his estimates of the action values by greedily selecting the best action (i.e. exploitation) so as to obtain reward, or he can improve the estimate by exploring non-greedy actions, as they may produce a greater reward in the long run. The problem of balancing exploration and exploitation has been addressed by several exploration strategies such as ϵ-greedy and Gibbs sampling. In ϵ-greedy an

action is randomly selected with a fixed probability ϵ, whereas in the so called Gibbs or Boltzmann exploration strategy, the action is selected according to its value [SB98].

In this chapter we report a short survey on decision-making with Markov Decision Processes and reinforcement learning.

4.1 Markov Decision Processes

Sequential decision-making problems for stochastic environments can be modeled as Markov Decision Processes. The decision maker, gets the current state of the environment in order to make its decision. Therefore, we assume that the state has the Markov property, that is, the state is function only of the current view of the environment and independent of the path that has led to it. A Markov Decision Process consists of a set of states, a set of applicable actions for each state, a transition model and a reward function. A solution to the problem is a policy Π, and $\Pi(s)$ is the action selected by the policy for state s. An optimal policy Π^* is the one that yields the highest expected utility.
According to [KLM96], a Markov Decision Process consists of:

- a set of states $\mathcal{S}$
- a set of actions $\mathcal{A}$
- a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ that specifies the expected instantaneous reward for the state that results from the application of the action $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$.
- a state transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, where $\Pi(\mathcal{S})$ is a probability distribution over the set $\mathcal{S}$. Therefore $T(s, a, s')$ returns the probability of moving to the state s' using the action a.

Similarly, [Sze10] defines a Markov Decision Process as a triplet $M = (Q, A, P_0)$ where Q is the countable non-empty set of states and A is the countable non-empty set of actions. A MDP is finite if both Q and A are finite. P_0 is the transition probability kernel that assigns to each state-action pair $(q, a) \in Q \times A$ a probability measure $P_0(U|q, a)$, that is, the probability that the next state and the associated reward belongs to the set U, when performing an action a in the state q. Therefore, it returns the probability $P(q_1, a, q_2) = P_0(q_2 \times \Re|q_1, a)$ of performing an action a for moving from a state q_1 to a state q_2, and the expected immediate reward for choosing the action a in the state q_1. The goal of the decision maker is to maximize the espected total discounted reward. Indeed, the result of executing a behaviour is the total discounted sum of the rewards incurred during the state sequence: $\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_{t+1}$, where γ^t is the discount factor.

The finite-horizon model forces the agent to optimize its expected reward for a limited number of interactions. Therefore, this is appropriate when the length of interaction is well known as in episodic environments. On the contrary, the infinite horizon model uses a discount factor γ for discounting rewards received in the future and limit a potentially unbounded sum. We can thus distinguish in two different kinds of interaction with the environment. Whereas continuing tasks have no end, in episodic tasks the interaction is divided in episodes where the agent ends in a set of terminal states in order to be reset to a fixed initial state. The reward signal should be delayed to terminal states so that the agent doesn't get any positive reward until it reaches the goal position, otherwise the agent might find a way to get a reward without achieving the goal.

Partially Observable Markov Decision Processes

In a real setting, the environment is not fully observable. This means that the agent does not completely know in which state it is. To handle those scenarios, MDPs are extended with a sensor model that specifies the probability of perceiving evidence e in state s. In partially observable MDPs (POMDPs) the sensor model provides a probability distribution over possible states. Thus, a belief state can be computed by means of filtering and the original problem can be considered as a MDP, where the policy determines the action selection and the execution leads to a new belief state.

4.2 Computing a policy

The simplest way to compute a policy is to use dynamic programming, a collection of algorithms, such as value- and policy-iteration, that can be used to compute optimal policies given a complete transition model of the environment.
A model of the environment is something that the agent can use to predict the effects of certain actions, it is thus a way to reason about its actions and deciding how to act. A model of a deterministic environment, returns the next state and the next reward, given the current state and an action. When the environment is described as stochastic, there are several possible next states and rewards for a state-action pair.
In fact, models can be classified in *distribution models* and *sample models*. Dynamic programming needs a complete distribution model, which means, all applicable actions for a certain state and their probabilities should be returned by the environment model. Dynamic programming methods generate for each state the probability distribution of all

possible transitions. Each distribution is used to compute a backed-up value and update the estimated utility value. This value is computed using the Bellman equation, where the utility of a state is the immediate reward of a state plus the expected discounted reward for the next state. Therefore, dynamic programming produces a full backup of values, and thus a better estimate, though this requires more computation (i.e. the branching factor determines the complexity of the problem). Instead, a sample model returns an action based on the transition model and this process may be corrupted by the so called sampling error. However, if the time to complete a full backup of values is insufficient and the branching factor is too high, sample backups are the preferred solution.

For this reason, dynamic programming is rather considered as a probabilistic planning method, where the policy represents a path of actions leading to a goal state. Another way to compute a policy is to use optimization methods (e.g. genetic algorithms) for searching through the space of policies. [SB98] emphasizes differences and relationships between planning and learning methods. In fact, whereas planning uses simulated experience generated by a model, learning methods use real experience gathered during the interaction with the environment. In fact, in real world problems transition probabilities or rewards may be unknown, thus it is not possible to take advantage of a complete model of the environment. In this case, Monte-Carlo and TD-learning methods are the preferred solution.

In Monte-Carlo methods, a complete environment model is not required as the estimation of the state value is based on the average of multiple independent executions (i.e. episodes) starting from the given state. Monte-Carlo methods backup each state value based on the entire sequence of rewards gathered until the end of the episode. Therefore, policy improvement occurs after the end of the episode, which means that in order to backup the estimation, all episodes have to eventually terminate. The advantage is that the method is general and it is possible to estimate only a subset of states by starting episodes from those states and ignoring the others. However, the variance of the results can be very high and thus the quality of the estimates very poor. In fact, by the law of large numbers the standard deviation is inversely proportional to the number of simulations performed. An alternative solution is to user Temporal-Difference learning algorithms. These require to get a reward after one step, rather than waiting for the end of the episode. In the one-step version of TD, the backup is based on the reward of the next state plus the discounted estimated value of the next state (i.e. $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$). This can easily extended to the general n-step TD. However, n-steps methods requires waiting n steps to observe modification on the policy, and this may be problematic for large n. This problem is solved by the TD(λ) algorithms by means of elegibility traces.

Therefore, TD(λ) allows a more efficient learning even with long-delayed rewards and non perfectly Markovian states.

TD learning algorithms can be classified in on-policy and off-policy approaches. In on-policy methods, the policy that is evaluated and improved is the same that is used for the selection (or control). On the contrary, in off-policy methods the policy that is evaluated and improved is separated by the policy used to sample actions. Off-policy approaches such as Q-learning usually are more flexible than their on-policy counterpart (e.g. SARSA algorithm) as they can learn an optimal policy even when guided by a random exploration policy. However, on-policy approaches are more realistic as they take an actual policy into account, and they tend to converge much quicklier to the optimal behaviour. Moreover, as soon as the environment becomes more complex, the advantages of on-policy methods becomes more apparent, as problem-specific knowledge can guide the exploration of big state spaces and make the learning process converge faster [RN10].

The simplest way to store the evaluation function is to use a tabular data structure such as hashmap. This works well for small state spaces but is infeasible for more realistic worlds (i.e. big state spaces with big branching factors). Indeed, the problem is both the memory required for the tabular structure and the time needed to fill it, as a bigger state space requires the learner to run more episodes in order to accurately estimate the value function. This can be solved by using a function approximator. A function approximator allows the learner to generalize from its experience and infer values for states that it has not even visited. Therefore, supervised-learning methods can be used for representing the value function. Accordingly, each backup can be considered as a training example of a desired input-output behaviour of the value function. An example is the use of a linear approximator such as a weighted sum of features. Features are problem-specific properties and determine the power of the approximation. Linear methods include also tile coding, where binary features called tiles are grouped into partitions called tilings that are representative and easier to handle. Artificial neural networks are the most common example of nonlinear function approximation. This approach was used by [Tes95] for a backgammon player agent. In new states, moves are selected by the agent according to the information collected for similar states visited in the past.

4.3 Markov Games

Markov games allow the modeling of multiple adaptive agents. In fact, a Markov decision process specifies a single adaptive agent that interacts with a stochastic environment. This

means that any other agent should be defined as a part of the environment. However, we assume a stationary probabilistic transition function, which means that agents described in the environment model cannot modify their behaviour over time. For this reason, Markov games apply game theory to Markov decision processes and allow the modeling of both competitive and cooperative multiagent environments. Markov games can be considered as a generalization of MDPs. For this purpose, the state evaluation and the concept of state-action value (i.e. Q-value) have been redefined for dealing with opponent strategies. Similarly, variants of traditional reinforcement learning algorithms were proposed, among these we can mention minimax-Q learning [Lit94]. In this work, the author shows how Q-learning is designed for finding deterministic policies, whereas the extension minimax-Q can find optimal probabilistic policies. In conclusion, Markov games are a mathematical framework for reasoning in multiagent environments.

Chapter 5

Related Work

5.1 Integrating GGP and Reinforcement Learning

The project RL-GGP[1] addresses the problem of integrating Reinforcement Learning libraries into the General Game Playing context. In particular, the author used the Jocular[2] player for handling GDL game descriptions, and the RL-Glue[3] interface for providing a general stub to all reinforcement learning algorithms offered by the RL-Library[4]. Therefore, the RL-GGP connects RL-Glue with Jocular, thus providing a connetion to the GGP-Server[5] and play games.

In [BS07], a TD(λ) learning agent is used to automatically discover features in a game tree, in order to use them for initializing the value function of other players of the same genre. The authors implemented a complete GGP learner that uses the GGP protocol for playing matches against other opponents and learn features. Those features do not incorporate any game-specific information and thus can be trasferred as a knowledge for speeding up the learning process for other games to play.

[1] http://users.dsic.upv.es/~flip/RLGGP/, accessed April 2012
[2] http://games.stanford.edu/resources/reference/jocular/jocular.html
[3] http://glue.rl-community.org/wiki/Main_Page
[4] A library implementing reinforcement learning algorithms to use with the RL-Glue interface.
[5] It is a Java tool developed by Stanford University that allows players to connect and play games described in GDL.

5.2 Assisted Living with Markov Decision Processes

Elderly and people with impairments may find difficult to perform daily life activities and require an aid to complete those tasks.

A planning system for the completion of handwashing has been implemented in [BHP+06]. The planner uses a Markov Decision Process for modeling the stochasticity of the outcomes of actions taken by the system. The state is represented by two environment variables: hand position (tap, towel, soap, water, away) and water flow (on or off). The transition model of the planning system was manually defined according to the observation of a professional caregiver guiding ten subjects with dementia through handwashing. A large reward is given to a completed handwashing, while a small cost proportional to the level of detail is associated to the prompt. In this way, the system begins with a minimal level of prompting and increases the level only when the user is not responding, thus encouraging user independence. The optimal policy for the decision maker was computed using a value iteration algorithm. Furthermore, the study used a questionnaire for evaluating the effectiveness of the system. The authors asked professional caregivers to evaluate the difference between the prompting given by the system and the one given by a caregiver in simulated handwashing scenarios. However, both the input and the output were simulated in this experiment. An improved version of the system is presented in [HvBPM07]. This time, the authors used computer vision for recognizing the current state and speakers for notifying audio cues. The decision maker was designed as a partially observable Markov decision process (POMDP) able to prompt, call human assistance or do nothing. Moreover, the user attitude was described by three different factors: the level of dementia (low, medium and high), the awareness of the task (never, no and yes) and the the type of prompts the user is more responsive to (none, maximum, medium, minimum). However, the reward function was still specified by hand and based on prior information from caregivers and users.

POMDPs are used for decision making in [LBP+09]. The belief state is computed by collecting information from different modalities such as a computer vision module for the human posture, and a speech recognizer component for interacting with the system. In effect, multimodal observations allow collecting evidence in different domains, thus reducing the amount of data needed for representing the same state in a unique modality.

Reinforcement learning has been used for a smart light control in [KWA10]. The system uses hierarchical reinforcement learning for learning user's preferences and providing a comfortable light setting. A context module infers user's state and mantains a model of

his preferences, while the adaptation module balances the tradeoff between user comfort and energy usage. Settings are represented by a utility function similar to the Q-value. Therefore, each user in the smart home may have his profile corresponding with his own Q-table, and the table may be loaded for tailoring the service to a particular user. Moreover, activities are classified in a two-level hierarchy and services are automatically chosen based on the user's preference up to that point in time. A change to the settings is used as a feedback for updating the decision policy. The hierarchical abstraction over the set of states allows the reduction of the state space as decisions and their effects affect only certain portions of the search tree. This speeds the convergence of the learning process up, which is crucial for on-line applications.

The goal of the MavHome project [CD04, CYH+03] is to automate basic functions in order to maximize the comfort, adapt to inhabitants and minimize the cost of operating the home. The ALZ[6] prediction algorithm [GC03] is used for predicting inhabitants' behaviour in terms of interactions with devices, while reinforcement learning is used for computing a control policy. For this purpose, the authors defined the following reward function:

- -1 for every manual interaction the user has to perform
- -0.2 for each action the decision maker performs
- -0.3 for each time interval in which a lamp is turned on

The reinforcement learning agent uses tile coding for constructing a compact representation of the Q-value function and dealing with the state space. In fact, we remind the reader that the size of the state space grows exponentially in the number of devices in the home, and therefore, using tabular approaches is infeasible even for small scenarios.

5.3 Notifying dangerous situations

Visual and audio cues are used in [KTNK08] for enhancing risk perception and help people to realize dangers beforehand. In this work, the system tracks state and position of persons and objects in order to assess the level of risk based on a ontology (where relationships describe dangerous situations). Furthermore, a notification unit uses audio and visual cues (i.e. speakers and illuminators) for alerting users. The work reports a user study where 20 subjects simulated the behaviour of an elderly person during a fall due to obstacles placed on the floor. Subjects answered questions about the visual noti-

[6] Active LeZi is a prediction algorithm based on the LZ78 text compression algorithm.

fications in order to assess the effectiveness of this component. In particular the authors were concerned in understanding whether the subject could perceive the notification and whether it assisted the subjects in avoiding the obstacles. The results showed that the visual cues were effective under different conditions for alerting users and preventing them from falling.

Chapter 6

Approach

We propose a solution to the problem of designing an early warning system for Ambient Assisted Living. Indeed, our research question states the following requirements:

- The world model must embed an abstract representation that can be used by search algorithms.
- The warning system evaluates the danger level of states and is allowed to use warning notifications for guiding the user towards his goal in order to avoid that he gets too close to a danger.
- The user is aware of the current state of the world and is able to act on the environment and receive any warning notification.

Therefore, we start by modelling a domestic setting as a single-player game and we design the warning agent as player. Secondly, we discuss different ways to implement an early warning system and we report some considerations about the undertaken design. In conclusion, we design the interaction with the user by means of a virtual environment and we report some related work.

6.1 Modelling a domestic environment

We are interested in simulating the behaviour of users at home. In a domestic environment, a user may change position and is able to manipulate both active and passive objects. Passive objects (e.g. an apple) can be held and moved by the agent, whereas active objects (e.g. an appliance) cannot be moved by the user, though he can interact with them. The table 6.1 shows some examples of this classification. We decided to represent the envi-

Action	Examples
Position changes	Movements: left, right, forward and backward
Manipulation of passive objects	• Take an apple • Hold a mobile phone • Release the hand-held content
Interaction with active objects	• Switch a stove on/off • Open/Close a cupboard

Table 6.1: Actions in a domestic context

ronment as a grid, and the user can only manipulate objects in his cell. However, in case we want to define the behaviour of movable devices such as hairdriers, we should define another category of objects.

Describing environment dynamics in GDL

Regarding the representation of the environment, we decided to model the setting as a game description by means of the Game Description Language. It is a declarative language which allows the concise and high-level specification of arbitrary finite games. This is very important, as games arise in every multiagent environment, it can be seen as a complete specification language. Moreover, all tools (e.g. players, game servers) used in the General Game Playing context can constitute a complete framework for building autonomous agents that can automatically learn how to participate in a multiagent society from the game rules and without the need of being re-programmed.
Therefore, the reason behind the choice of GDL is the possibility to exploit available tools and techniques from the GGP context. In addition, we can exploit the leading expertise of Reykjavik University in this field, as the CADIAPlayer agent representing the university in the GGP competition earned the world title for two years in row (i.e. 2007 and 2008), by breaking the boundaries of adversarial search in big state spaces with an innovative technique based on Monte-Carlo tree search [FB08, BF09].

Modeling a warning agent in GDL

Suppose a scenario with a user in a 3×3 grid world. A state is represented by the position of the user in terms of x and y coordinates. Therefore, the size of the state space is 9 (i.e.

state is $(x, y) \in X \times Y$ and $|X| = |Y| = 3$) and the state diagram is the one in figure 6.1. For the sake of simplicity, we can assume user's initial position at $(1, 1)$ and his goal

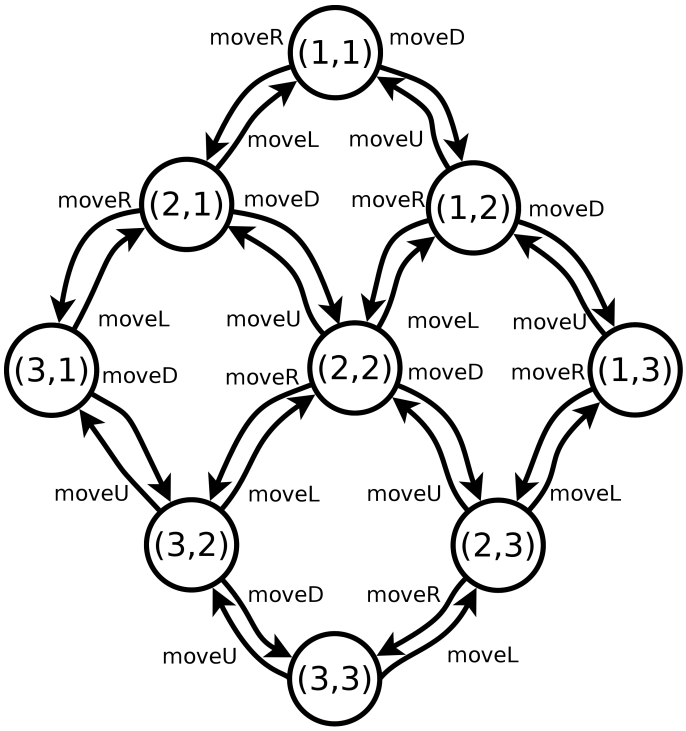

Figure 6.1: The State Machine for the 3x3 grid scenario

position at $(3, 3)$, and we can mark the positions $(3, 1)$ as dangerous.

```
(role user)

(size 3 3)
(init (at user 1 1))
```

Listing 6.1: A single-player game

User's behaviour is defined in terms of rules that can be applied to a certain state to get a new position. The user can perform actions according to his context, which means properties holding in certain states make certain actions applicable. Moreover, each action affects the state of the system, thus defining effects of actions is important in order to be able to simulate environment dynamics. For instance, it should not be possible to get out of the grid, as well as knowing that a move right makes the user position increase (listing 6.2).

```
;; User
(<= (legal user mover)
  (true (at user ?x ?y))
  (size ?xmax ?ymax)
  (smaller ?x ?xmax))

(<= (next (at ?what ?x ?y))
  (true (at ?what ?x ?y))
  (not moves))

(<= moves
  (does user mover))

;; moving users
(<= (next (at user ?x1 ?y))
  (true (at user ?x ?y))
  (succ ?x ?x1)
  (does user mover))
```

Listing 6.2: Legal actions and their effects on the environment

Goals and dangers can be specified as properties that hold in certain conditions (listing 6.3).

```
(<= terminal
  usergoal)

;; Definition of dangers and goals
(<= danger
  (true (at user 1 3)))

(<= usergoal
  (true (at user 3 3)))
```

Listing 6.3: Specifying dangers and goals

The game ends when the user achieves his goal or when a potential danger occurs. This makes perfect sense as we want the system to warn and guide the user to his personal objective and getting into a risky configuration would mean failing this task. For this purpose, we distinguish terminal states by means of a *danger* relation that we added to the GDL language. Similarly, we defined a *reward* relation in order to assign reward to non-terminal states. The listing 6.4 shows a small example.

```
(<= terminal
    danger)

(<= terminal
  usergoal)

(<= danger
  (true (at user 3 1)))

(<= usergoal
  (true (at user 3 3)))

(<= (reward user 1)
    usergoal
    (not danger))

(<= (reward user 0)
    (not usergoal)
    (not danger))

(<= (reward user -1)
    (not usergoal)
    danger)
```

Listing 6.4: Assigning rewards to the user

This means that the warner must warn the user in the cell (3,1) and decrease the level of intrusiveness in the other cells, based on the danger value. By intuition, the farther we are from the danger, the lower the danger value is (fig. 6.2).

6.2 Designing an early warning system

An early warning system is an intelligent agent that monitors the state of the environment in order to estimate its level of danger. Consequently, it can issue a warning message as soon as it realizes that the monitored user is likely going to get in a dangerous situation. Moreover, the system may suggest to the user a sequence of actions to achieve the goal in a safe way, and give an estimation of user's behaviour in order to give him a complete feedback of the environment dangerousness.

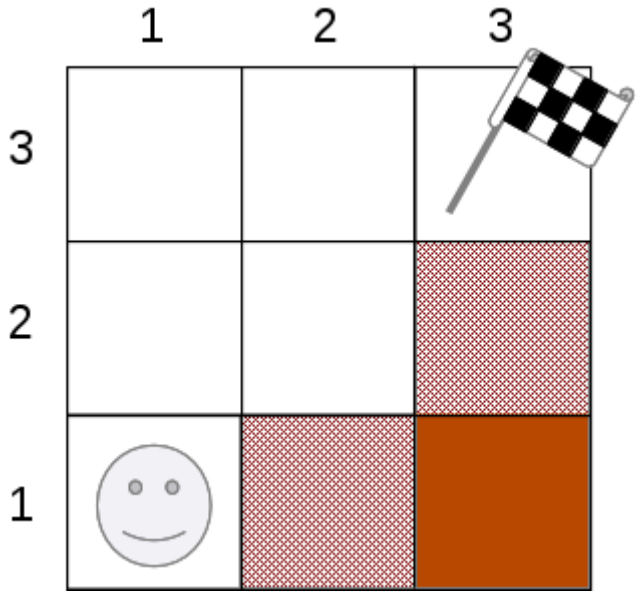

Figure 6.2: The 3x3 grid scenario with a danger in (3,1) and goal in (3,3)

6.2.1 Guiding the user

Conducting a user to his goals is a planning problem. This requires the application of search algorithms to a model of the environment, for producing a path of actions connecting the current state to the goal. However, the real world is stochastic. Therefore, deterministic planners can be extended to work with partially observable or nondeterministic environments, by means of online planning and replanning. Unfortunately, deterministic planners do not always work for this purpose. Indeed, in some environments the uncertainty is too high and the result would be suboptimal [RN10]. For this reason, decision problems are often modeled as Markov Decision Processes, where the solution is a policy describing the behaviour of the agent for every possible state. Thus, uncertainty can be modeled as probability distribution that can be applied both to state transitions and to the observability of the environment state, which is modeled as a belief state. Indeed, as we already introduced, uncertainty may be specified by a Markovian (i.e. the probability assigned to the transition only depends on the current state and not on previous history) transition model that describes the behaviour of the user in certain situations. This could be done by performing case studies where observed individuals are described by customized profiles and the player strategy (i.e. the warner's policy) may be computed off-line (see section 4.2) and loaded when required. However, we want the system to be general, that is, no prior knowledge about the user should be provided and the system should learn how to behave from the specification of the scenario. Therefore, we decided to use TD-learning methods to make the system autonomously compute a pol-

icy while interacting with the environment. Indeed, "TD-learning can learn directly from raw experience without a model of the environment's dynamics" [SB98]. Consequently, TD-learning methods are called model-free methods [RN10]. Moreover, as TD-learning grows out of Monte-Carlo methods, we need to balance the trade-off between exploration and exploitation, thus distinguishing in on-policy and off-policy methods. Two classical examples of TD-learning algorithms falling into these categories are Q-learning and SARSA[1]. In fact, Q-learning allows us to use an arbitrary sampling strategy[2] during the learning process (e.g. ϵ-greedy or Boltzmann exploration.), while backing up the best Q-value and thus without taking the actual policy into account. On the contrary, SARSA is an on-policy algorithm, as it is guided by the selection strategy. Despite both of them converging to the optimal policy when the number of explorations tends to infinite[3], Q-learning has been proven to converge slower than SARSA [SB98].
Therefore, Q-learning provides a *general* (i.e. independent from the scenario) way to perform on-line planning in a stochastic environment. The pseudocode for the Q-learning algorithm is shown in the listing 6.5 [SB98].

```
Initialize Q(s,a)
Repeat (for each episode):
    Initialize s
    Repeat (for each step of the episode):
        Select a from s using the policy derived from Q
        Perform a and perceive the next state s' and the reward r
        δ ← [r + γ · max_{a'∈A} Q(s', a') − Q(s, a)]
        Q(s, a) ← Q(s, a) + α · δ
        s ← s'
    until s is terminal
```

Listing 6.5: Pseudocode of the Q-learning algorithm

To exploit Q-learning properties, the warning agent is trained by exploring the environment by means of a pseudo-random selection strategy (i.e. ϵ-greedy). Thus, the warning agent discloses dangers by visiting dangerous states and getting the associated reward. The listing 6.6 shows the use of *random-sample one-step tabular* Q-learning as planning method [SB98].

[1] SARSA stands for current State, current Action, next Reward, next State and next Action.

[2] A sampling, selection or exploration strategy is the approach used for selecting actions and facing the exploration-exploitation trade-off of the bandit problem.

[3] Most of the proofs apply to the tabular version, whereas classic Q-learning with non-linear function approximation may fail to converge.

```
Do forever:
    1. Select a state s ∈ S, and an action a ∈ A at random
    2. Send s and a to the sample model
       and obtain a sample next state s' and a sample next reward r
    3. Apply one-step tabular Q-learning for the update
        δ ← [r + γ · max_{a'∈A} Q(s', a') − Q(s, a)]
        Q(s, a) ← Q(s, a) + α · δ
```

Listing 6.6: Random-sample one-step tabular Q-planning

However, one-step Q-learning (i.e. Q(0)) produces a policy which fits very well with Markovian states, while we may want to assign a value to actions according to their desirability and contribution on achieving the actual goal. Moreover, long-delayed rewards cause an actual distribution of values only close to terminal states and this means that we cannot rely on non-terminal state evaluations for estimating the desirability of actions given that they are selected in certain states. For this purpose, we decided to use the multi-step version of TD algorithms, the so called TD(λ), and in particular, we used Q(λ) (listing 6.7) for computing state-action values and taking advantage of elegibility traces.

```
Initialize Q(s, a) arbitrarily and e(s, a) = 0, for all s, a
Repeat (for each episode):
    Initialize s, a
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g. ε-greedy)
        a* ← argmax_b Q(s', b) (if a' ties for the max, then a* = a')
        δ ← r + γQ(s', a*) − Q(s, a)
        e(s, a) ← e(s, a) + 1
        for all s,a:
            Q(s, a) = Q(s, a) + αδe(s, a)
            if a' = a*, then e(s, a) ← γλe(s, a)
                        else e(s, a) ← 0
        s ← s'; a ← a'
    until s is terminal
```

Listing 6.7: Pseudocode for the tabular Q(lambda)

6.2.2 Warning the user

Warning the user consists of finding dangerous states in order to alert him when he gets too close to them. Therefore, we need to search for terminal and dangerous states, and

compute the distance from the closest danger, expressed as the number of actions leading to that risky situation. For this purpose, we define a sphere of protection around the user (figure 6.3), where we monitor the presence of dangers. This means that we need to repeat this process as soon as the user modifies the environment state, though it can be interrupted as soon as we complete the visit to the level of the closest danger. Indeed, as we may be in between different dangers, we should list all paths leading to potential dangers. For exploring the environment and finding dangerous states, we use a variant

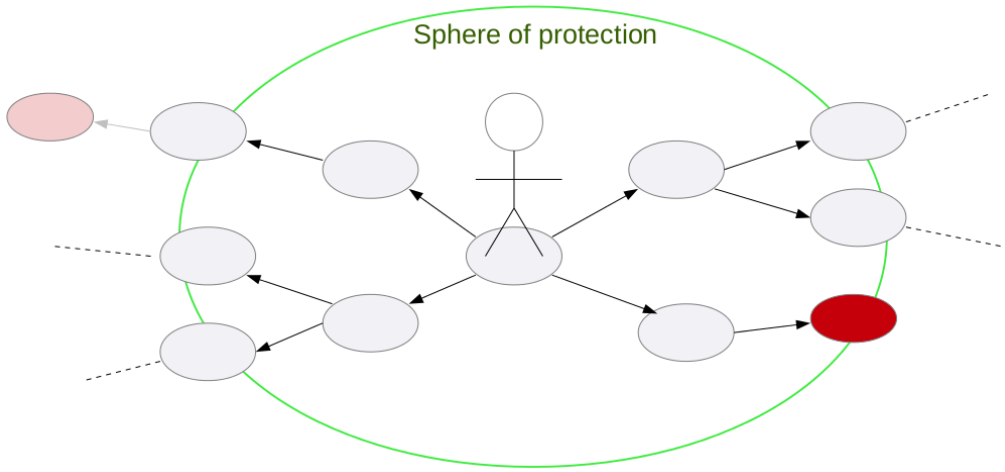

Figure 6.3: The sphere of protection concept

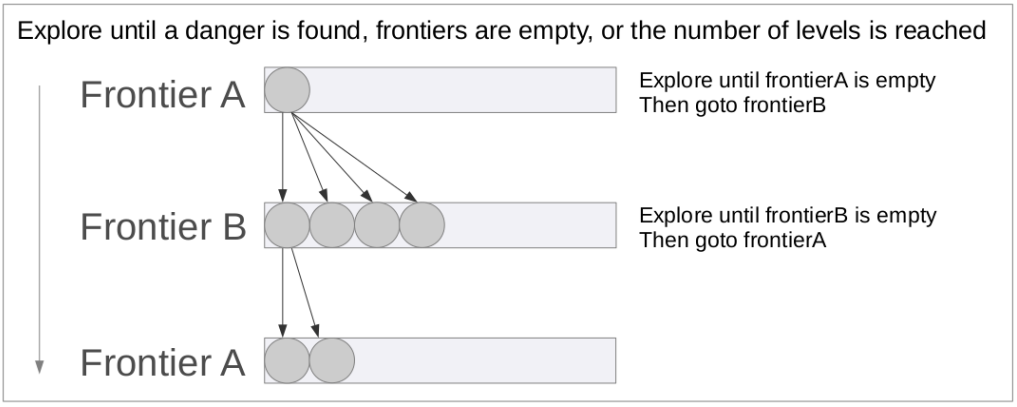

Figure 6.4: The depth-limited breadth first algorithm

of breadth-first search which limits the number of levels. We use two different frontiers and we count the number of levels explored up to a given moment, in order to block the exploration (figure 6.4). Obviously, we may want to speed this process up by using informed search methods. Designing a good heuristic function would require choosing problem-specific properties of the state, whereas we want the system to be general and able to warn the user, given a description of the scenario as input. Moreover, this may

be a good point to deepen during further studies, as we may want learn user's habits and guide the search of dangers according to the actions that he is likely going to perform in certain situations.

Learning to intervene

The warning system uses a static threshold to decide whether to display a warning message or not. Indeed, as soon as we find a danger, we may want to report it to the user by showing the first action of a sequence leading to the risk. However, people may have different preferences for the intervention before the potential danger, which means, we need to define a threshold to decide whether to intervene or not. For this purpose, we give the possibility to define a threshold before the beginning of the simulation. Whenever the distance from the closest danger falls under the threshold, we display a warning message with the action that the user should avoid.

In fact, the system should adapt to users' preferences and different awareness faculties, in order to maximize system effectiveness and users' satisfaction. A straightforward representation of this problem is the Markov decision process presented in figure 6.5.
The environment state is fully observable while the outcome of actions is stochastic. In-

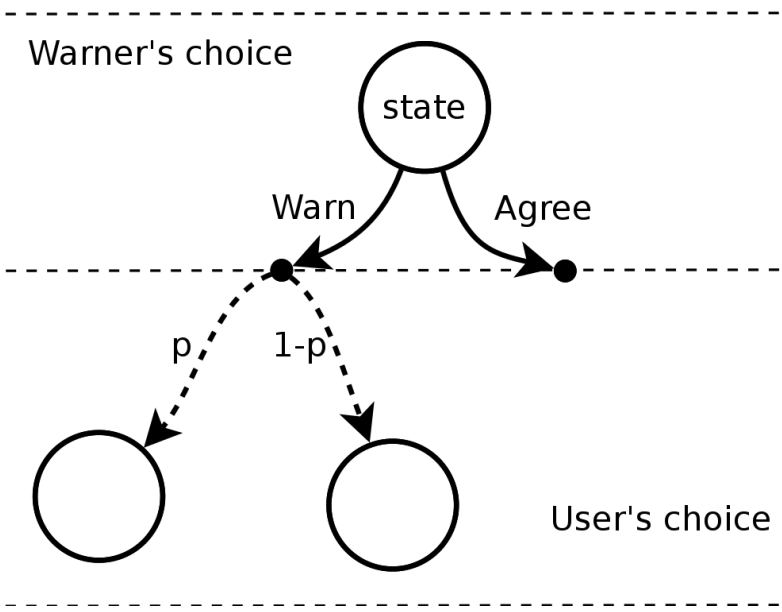

Figure 6.5: The Markov Decision Process for the intervention

deed, the outcome of the intervention is stochastic, as the user may decide to accept the notification or reject the intervention by returning a penalty. Regarding the state representation, we may use the level of danger, though this does not allow us to distinguish

dangers according to their gravity. For this reason, we can exploit the same state representation that we are using in the planner, so as to distinguish different states with the same danger level and be able to use function approximators, such as neural networks, for generalizing from the experience gathered during the interaction. Thus, a solution is to use a Q(λ) reinforcement-learning agent for training a neural network like in [Tes95]. A similar approach is the one followed by [CD04, CYH$^+$03] where an MDP and reinforcement learning are used to learn a policy for controlling lights. A discomfort factor is given to the system whenever the user needs to manually modify the light setting.
Unfortunately, training such a system requires the interaction with an actual user, in order to get feedback. Therefore, both for its lack of generality and for the need for a specific user, we decided not to implement this approach, and we refer to future developments for any improvement.

Chapter 7

Implementation

In this chapter, we report the steps that led to the implementation of an early warning system.

7.1 Implementing an early warning system

The early warning system prototype consists of the following layers (figure 7.1): a GDL parser and reasoning tool that we use for getting a GDL description as input and build a model of the environment that can be used by planning algorithms, a warning agent that explores the state space and learns a state evaluation function based on the level of danger, as well as a user interface that shows the current state of the environment and allows the user to select actions for acting in a simulated setting.

7.1.1 Practical reasoning with GDL

Since a game description is a logic program, we need to use an automatic reasoning tool for inferring legal moves and successor states. Several basic players can be found on line, as well as parsing and reasoning tools for handling GDL game descriptions. A list is available on the German website of the General Game Playing project[1].
The General Game Playing Base package[2] is a set of Java libraries and applications designed for writing, validating and playing game descriptions written in GDL. It is released under a BSD[3] license and implements a complete player. We used the parser and the

[1] http://www.general-game-playing.de. Accessed may 2012.
[2] http://code.google.com/p/ggp-base/. Accessed may 2012.
[3] http://www.opensource.org/licenses/bsd-license.php. Accessed may 2012

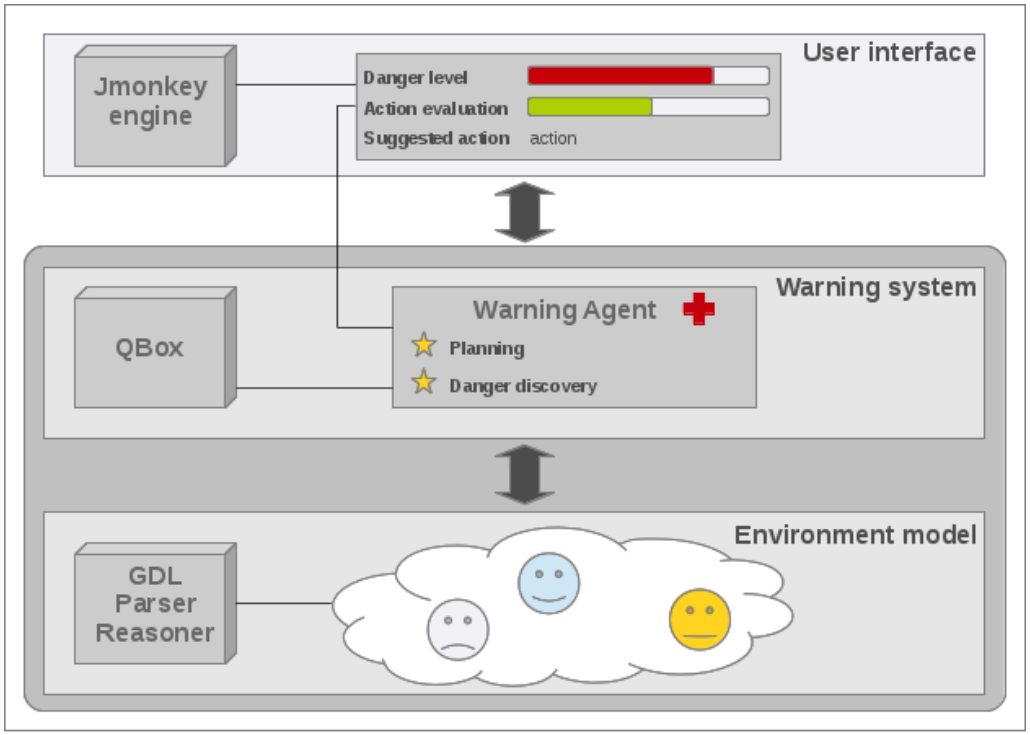

Figure 7.1: An overview of the system

prover of this project for handling the game as a state machine. In particular, we implemented a general task environment class which embeds the state machine and provides to applications an omogeneous interface for managing the environment model. It returns information about the setting (e.g. size, roles, devices), and provides functionalities to handle the game dynamics such as the initial state, the transition function and applicable actions, as well as the reward function and terminal states.

As the semantics of the GDL *goal* relation was to assign a reward only to terminal states, we modified the tool so that it is possible to specify rewards like we usually do with the *goal* relation, and use them for non-terminal states. In this way, we can handle dangerous states as non-terminal states where the agent gets a negative reward (i.e. a penalty).
Similarly, we added a *danger* relation that can be used to mark states as dangerous, in order to be found by the system. Moreover, we added the possibility to define appliances as roles so that it is possible to model non-deterministic behaviours such as a telephone ringing because of an exogenous and unpredictable event. This would not be feasible with standard GDL as environment stochasticity usually is modeled with the random role, while we may want to control those events for simulating certain conditions.

7.1.2 Implementing a warning agent

The warning agent is implemented in Java, as a single tabular Q-learning agent. Since a state is a set of holding properties called fluents, the hash code of a state results from the hash codes of its individual fluents. In particular, fluents are converted to strings and hash-codes are easily computed by means of a JDK[4] provided function.

QBox

At the beginning, we were convinced that using an existing reinforcement learning library was the best choice to take advantage of plenty of implementations and frameworks (e.g. Weka for machine learning) for our solution. However, due to the simplicity of the task and to the educational purposes of this project, we decided to implement the basic TD-algorithms (i.e. SARSA, TD(0), Q(0) and Watkins Q(λ)) as a Java library that we called QBox (figure 7.2). The name refers to the objective of providing a basic out-of-the-box

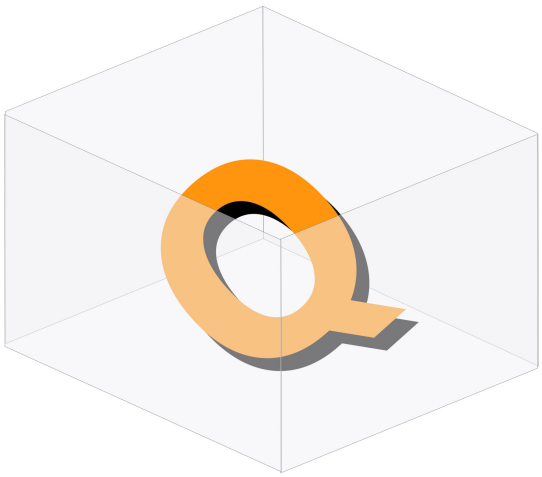

Figure 7.2: Our QBox logo

framework for implementing TD-learning agents. In particular we distinguish between (figure 7.3):

- **Environment** A task environment describes the environment dynamics in terms of a transition function, as well as goals by means of the reward function. Therefore, it wraps the task environment that we defined in 7.1.1. The interaction between agent and environment takes place by means of states and actions. A state describes the properties of the current situation. Similarly, an action is the result of the decision

[4] Java Development Kit. It is the set of API for the Java programming language.

making process and is used by the agent to modify the state of the environment. Both are defined as Java interfaces.

- **Brain** A brain defines the decision making element. It implements the selection strategy and is responsible for storing Q-values. In particular, we provide ϵ-greedy and Boltzmann selection, as well as the best policy. We use hash tables for storing state values (i.e. *VTable* brain) and state-action values (i.e. *QTable* brain). Consequently, a brain may be serialised to a file and loaded when required.
- **Agent** An Agent defines the agent behaviour as perception-action loop. The agent can be used for running a certain number of episodes or for evaluating its policy. Indeed, an episode consists in selecting an action, performing it on the environment in order to get the associated reward, as well as a policy improvement step. We implemented several learning agents based on the following reinforcement learning algorithms: Sarsa, TD(0), Q(0), Q(λ).

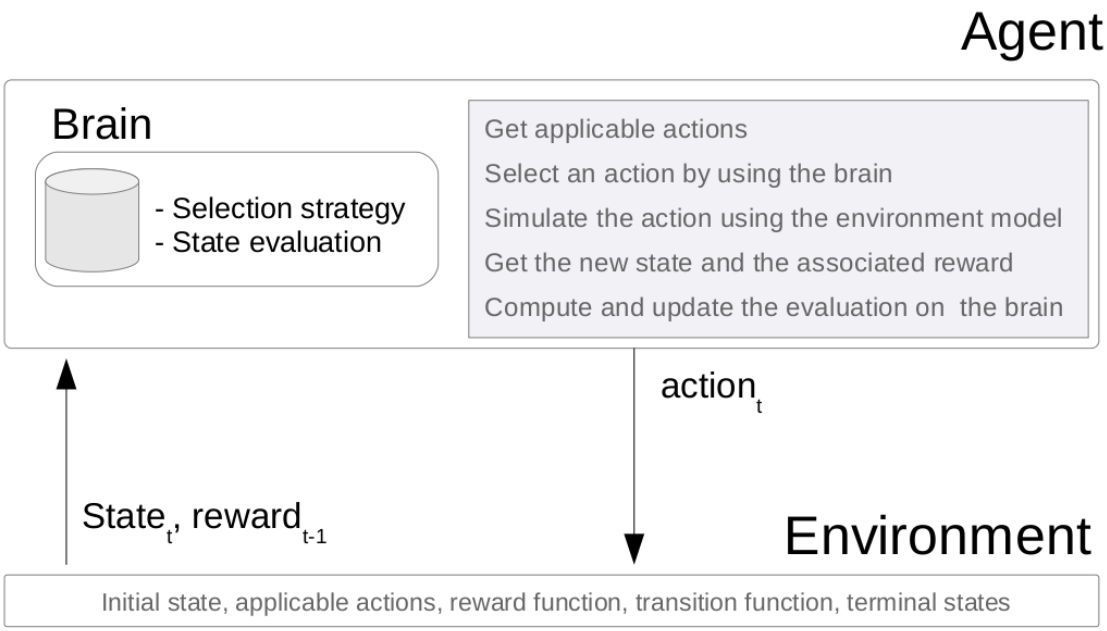

Figure 7.3: The QBox organization

The warning agent

The warning agent is implemented as a Q(λ) agent that exploits its experience for returning suggestions and warning notifications. For the purpose, we built a system (figure 7.4) that can load, train and save agents, as well as monitor and interact with users within a simulated setting.

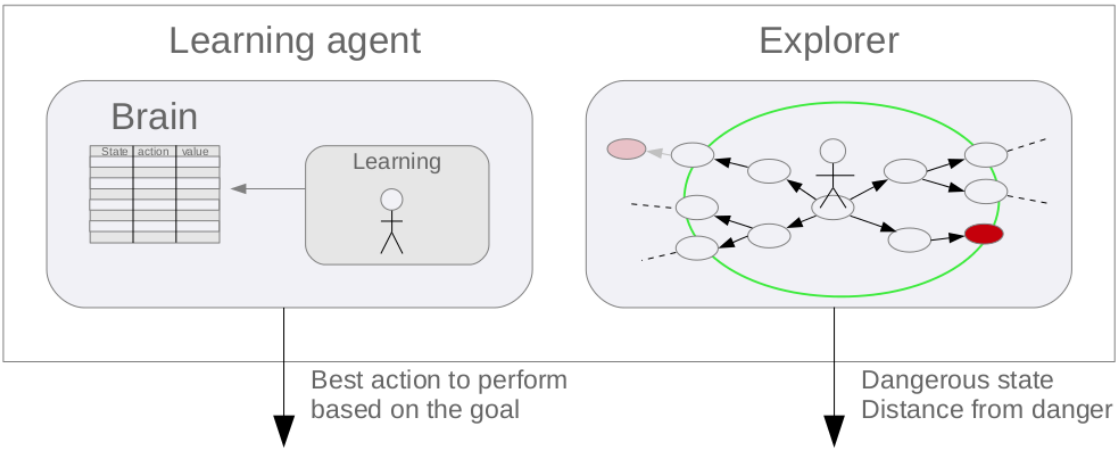

Figure 7.4: The early warning system

At the beginning, a role to warn is chosen and the size of the sphere of protection is defined. The system uses a Q(λ) agent with a *QTable* as Brain, as well as an explorer module that handles the distance from dangers.

The training process consists of simulating the chosen role by running several episodes using the ϵ-greedy selection strategy. On the other hand, any other role defined in the environment uses a random selector. This means that the environment queries all players for selecting an action and the joint action is used for moving the state machine to the next state. During the learning, the Q-value of state-action pairs is updated according to the reward the agent gets from the problem description. The brain of the agent can be saved and loaded when required. This means that the learning process can be done off-line and the complexity of this task can be mitigated.

The warning process consists of running an episode from the initial state to a terminal state (i.e. a goal or a dangerous state). During the episode, the system uses the experience gathered by the Q-learning agent for suggesting to the user the best action to perform, based on its Q-value. Then, the explorer module is used to start a depth-limited breadth-first search from the current state. The exploration process is interrupted whenever the depth-limit is exceeded or a danger is found (i.e. the closest). Accordingly, all nodes on the level of the closest danger are explored in order to return all paths leading to dangers with the same distance. Therefore, the danger level for the current state is proportional to the number of actions leading to closest dangers. In particular, we use the following formula:

$$DL = (S - N)/S$$

where S is the size of the sphere of protection in terms of depth limit, and N is the number of actions leading to the closest danger. The danger level is set to 0 when a dangerous state is not found within the sphere of protection.
At this moment, the user can select an action among the applicable ones for the current state. The action is evaluated by the brain module of the Q-learning agent, according to its experience. The evaluation is based on the Q-value of the state-action pair and is normalized to the minimum and maximum value of the table. Accordingly, we use the formula:

$$(Q(s,a) - Q_{min})/(Q_{max} - Q_{min})$$

where $Q(s,a)$ is the Q-value for the given state-action pair, whereas Q_{min} and Q_{max} are the minimum and maximum value of the table. Moreover, the action is used to update the environment model and start a new monitoring process from the beginning.

7.2 Interaction design with virtual environments

An early warning system alerts the user as soon as a potential danger is detected. This means that the user should be aware of the current state of the world in order to be able to act on it and perceive any warning notification. In this sense, we exploit virtual environments as a solution for simulating smart environments.

In MavHome [CD04], a graphical model of the environment is used for showing the state of sensors and providing to a remote visitor the possibility to change the state of devices in the physical environment.
Using virtual environments as testbeds for assessing prototypes is a typical technique in the human-computer interaction community. In fact, it is very difficult to predict how users will perceive an interactive service and virtual environments can be used as a rapid user-centered prototyping technique which offers a flexible, fast and cheap alternative to expensive evaluations performed in real environments. A complete survey of prototyping toolkits for assessing ubiquitous applications is offered by [TYZ+11]. However, most of projects are focused on simulating particular technologies such as mobile hand-held devices or wireless-domain issues. A specific prototyping methodology for Assisted Living is discussed in [NFS+09].

7.2.1 jMonkeyEngine

In order to implement a virtual environment, we exploit the jMonkeyEngine[5] game engine. It is written in Java and released under the BSD license by a open-source project community. The architecture is GLSL[6] compliant and includes the jBullet physics library[7], the Nifty GUI library[8] and the SpiderMonkey networking engine. It comes as a collection of libraries and an IDE based on the NetBeans[9] platform, which allows graphical editing of the scene.

7.2.2 The user interface

Figure 7.5: The user interface

The user interface consists of four different layers (figure 7.5):

- **The main menu.** A Java Swing menu is used for managing the main controls of the application. In particular, we use a *Warning Agent* menu for disposing controls related to the warning agent and the creation of a setting. Similarly, simulation-

[5] http://jmonkeyengine.com/. Accessed may 2012.
[6] OpenGL Shading Language. http://www.opengl.org/documentation/glsl/. Accessed may 2012.
[7] http://bulletphysics.org/wordpress/. Accessed may 2012.
[8] http://nifty-gui.lessvoid.com/. Accessed may 2012.
[9] An open-source IDE. http://netbeans.org/. Accessed may 2012.

specific commands are listed in the *Simulation* menu, while the *help* menu reports basic instructions.

- **The presentation layer.** A JMonkey virtual environment is used for showing the current scene.
- **The simulation commands.** A combo box and a button are used by the user for selecting moves during a simulation.
- **The log.** A text area is used for recording the interaction with the user and reporting errors.

The virtual environment is managed by means of application states, that is, a portion of the application logics (e.g. GUI elements, scene graph composition) is organized in a centralised position and the main application can determine states to run. The interface can be in four different states: *start*, *intro*, *training* and *simulation* (figure 7.6). At the

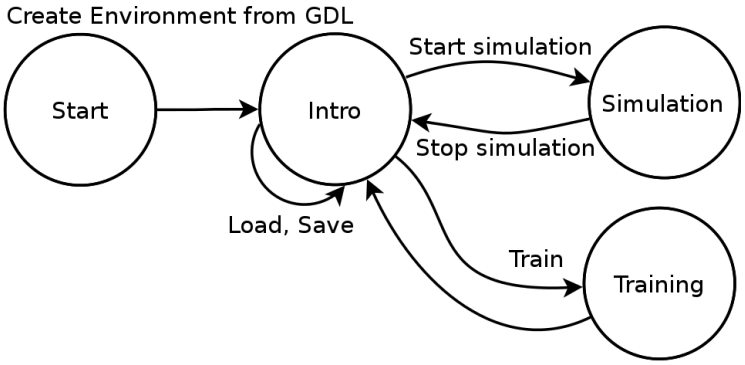

Figure 7.6: States of the interface

beginning the system is in the *start* state. As soon as the user loads a GDL description the environment is created and a grid is shown in the virtual environment (figure 7.7). At this moment, the user can use the settings to select a role, as well as the radius of the sphere of protection. Subsequently, the user may load an existing warning agent or use the one just created with the environment for training it by means of random roles. Accordingly, a dialog window is displayed to let the user specify the number of episodes to run. Subsequently, a loading bar tracks the complete training process (figure 7.8).Finally, the user may also decide to save the trained agent in order to use it again in the future.
As soon as the simulation starts, the virtual environment shows the state of objects and players and the simulation commands are unlocked in order to let the user select and perform a move. We decided to show only 9 cells in the world, as we are mainly interested in dangers in the user's cell, as well as in the closest cells. Therefore, whenever the user changes position, the camera moves and updates the view of the environment.

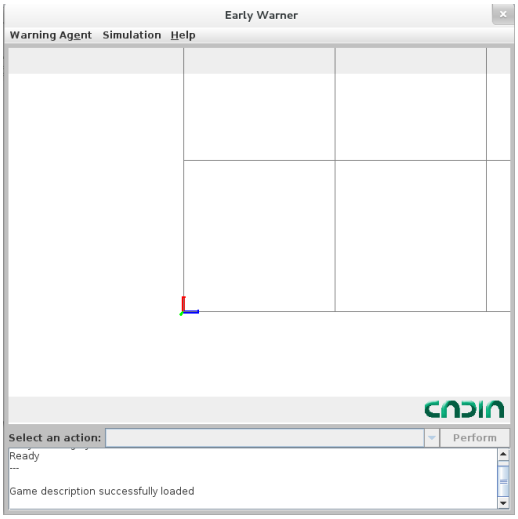

Figure 7.7: The *intro* state

Figure 7.8: The *training* process

In order to keep the interface general, we define the appearance of entities by means of billboard panels[10] and we use GDL ground terms (e.g. a role name, a object name) for loading graphic textures from the same directory of the GDL file. In this way, we avoid complex definition files and problem-specific 3D models. Regarding the size of panels, we decided to use a fixed width, while we keep the ratio aspect of pictures by computing the height accordingly. Similarly, we use a random positioning of objects so as to avoid that they are stacked on the same position.

Regarding the head-up display, we exploited the Nifty library for describing the 2-D interface as XML file. As we can notice in figure 7.9, during the *simulation* state we use a red progress bar to indicate to the user the current level of danger, while a green bar

[10] A billboard panel is a graphic component that rotates in order to always face the camera.

reports the estimation of the last performed action and a label shows the suggested next move. Similarly, we use text labels to show the content of user's hands and the state of

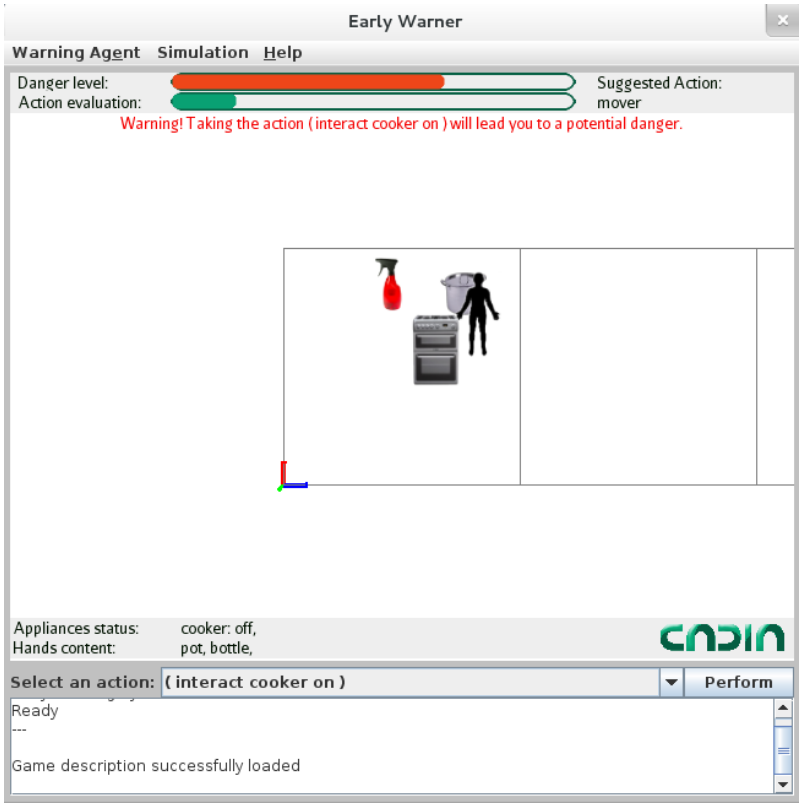

Figure 7.9: The user interface during a simulation

appliances. In this way, the user should be aware of the objects around him and the potential dangers related to them. In the example, the user picked up a bottle containing a flammable product and got close to the first cell, where there is a stove on. Therefore, the system notifies a potential danger through the red bar and an informative message. The text alert is shown according to the user preferences and the actual distance from the potential danger. Whenever the distance from the closest danger falls under the threshold, we basically display a warning message with the action that the user should avoid. The simulation is interrupted when the user reaches his goal or a dangerous configuration. Consequently, a dialogue window is used for notifying the termination of the simulation and, then, the interface shows the *intro* state to let the user load a different scenario or start another simulation.

Chapter 8

Evaluating the solution

8.1 Evaluating the system

Reinforcement learning algorithms require the tuning of parameters and present a sensitive behaviour to modifications of those values. In particular, we wanted to figure out how parameters affect the quality of the service provided by the system, in terms of:

- exploration of the state space (i.e. the knowledge of the system),
- best suggested path to achieve the goal in a safe way.

For this purpose, we built a test environment where it is possible to define an optimal policy and compare it to the ones computed. The optimal policy is defined by going through all states of the state space and assigning a value to actions. Once defined, it is saved and loaded when required for future experiments. Regarding the tests, they consist in training 20 different policies for 200 episodes. Policies are created using the same settings and are compared to the given optimal solution. The deviation expresses the number of differences with respect to the optimal policy. For each state, we compute the ordered sequence of actions expressed by the policy (i.e. the Q-table), and we create a data structure called slice that groups actions according to their preference (figure 8.1). As the reader may notice, the slice is used to handle the comparison and allow the specification of actions with the same value. In the example, the action C increases the deviation as we are expecting an action in the first set (i.e. the A action). The action C is removed from the slice and the expected action is one of the first set. Consequently, the actions A and D are marked correct, since the ordering between the first and the third set is respected. Moreover, we decided to increase the deviation when the state is not explored (i.e. values of actions for the state are equal to the initial value 0.0), as the decision making is affected

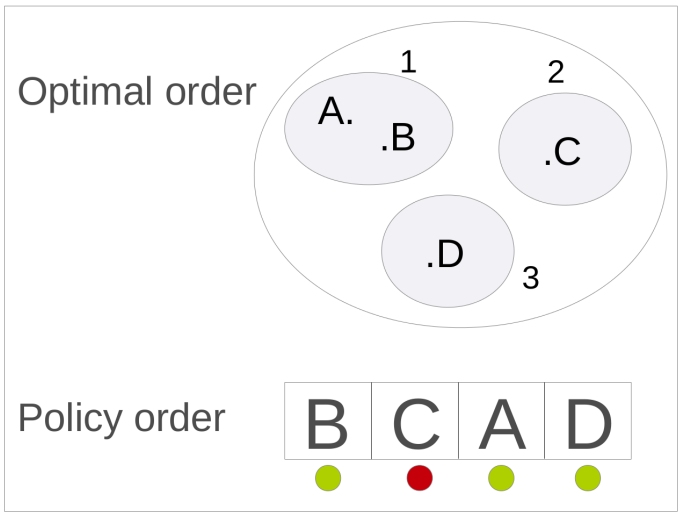

Figure 8.1: The action deviation comparison

by those actions. Consequently, the average deviation is used as a quality measure for the settings. However, the average deviation depends on the number of actions involved in the comparison. For this reason, we compute the percentage of deviation for the experiment as follows:

$$ExpDev(\%) = (AvgDEV/AN) * 100$$

where $AvgDEV$ is the average deviation (i.e. $AvgDEV = \sum_{k=1}^{N} dev_k/N$) for policies and AN is the total number of actions for the policy. A straightforward measure for the accuracy of the family of policies is:

$$Acc(\%) = 100 - ExpDev$$

As previously mentioned, we are interested in observing how variations to certain parameters affect the quality of the policy. For this reason, an array of values is given as input in order to define several classes of policies and repeat the experiments. All classes are presented in output as a histogram chart by means of the JFreeChart[1] library.

The game definition used for the test is reported in appendix A.1.

8.1.1 Exploration of the state space

Since TD-learning algorithms grow out Monte Carlo algorithms, it is required to balance the trade-off between exploration and exploitation. In particular, we used a ϵ-greedy selection strategy with an exponential decay 0.9999, which means that the ϵ determines

[1] http://www.jfree.org/jfreechart/

the balance and tends to converge to the policy faster due to the decay factor. We therefore decided to test different ϵ values and the presence of the decay factor. In particular, we repeated the experiments for $\epsilon = 0.1, 0.3, 0.5, 0.7, 0.9$. We report the setting in table 8.1 and the results in the charts 8.2 and 8.3.

Parameter	Value
α (learning rate)	0.2
α-decay	0.8
α-decay type	exponential (ensures convergence)
γ (discount factor)	0.95
λ (decay rate)	0.9

Table 8.1: Parameters for the tests

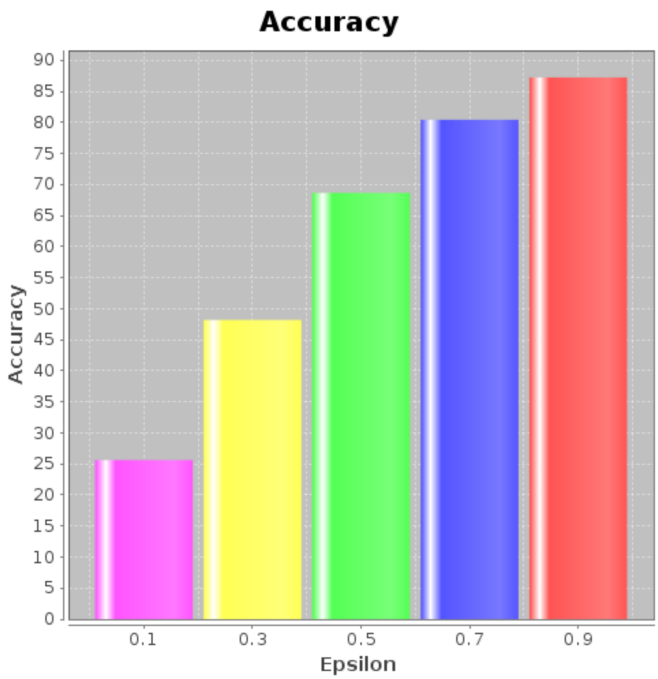

Figure 8.2: The test for the epsilon

Moreover, we repeated the experiments without using the epsilon decay. Results show the accuracy increases for higher epsilon values.

8.1.2 Defining the rewards

The reward function determines the trade-off between goal and danger. For the specific scenario presented in appendix A.1, we can distinguish two main behaviours. When the danger has a penalty much bigger than the reward associated to the goal, the system may

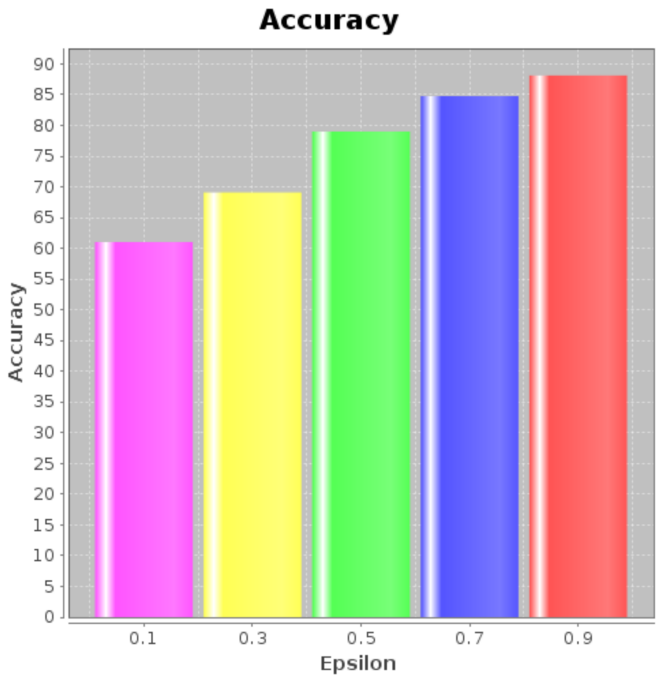

Figure 8.3: The test without the epsilon decay

suggest to the user to take the bottle and move it away from the danger, even when the stove is off. On the contrary, when the reward associated to the goal is too high, the system tends to give more priority to the pot and to suggest to the user to switch the cooker on, even when the pot is not in the same cell. Therefore, assigning rewards is a difficult task and may produce cycles in the policy that will mislead the end user. It therefore is important to understand the best setting. The test did consist in computing 20 policies for each experiment, so as to find the deviation and return a chart showing the quality of the reward function. The results are showed in the table 8.2.

No-danger/Goal	Danger/No-Goal	No-danger/No-goal	Danger/Goal	Accuracy
-1.0	1.0	-0.01	0.0	44.09%
0.0	1.0	-0.01	0.0	39.63%
1.0	0.0	-0.01	0.0	71.01%
1.0	-1.0	-0.01	0.0	84.35%
0.0	-1.0	-0.01	0.0	75.88%

Table 8.2: Results for different reward functions

As the reader may notice, the table shows the accuracy for different reward values. In particular, we used a penalty of 0.01 for non terminal states in order to make the algorithm compute a short path to achieve the goal. The best policy is the one with the highest gap between danger and goal (i.e. -1.0 and +1.0). In fact, we are considering unexplored

states for the deviation measure. Consequently, the results can be improved by increasing the ϵ parameter or omitting unexplored states for the measure (see 8.1.1).

8.2 Assessing the interaction with users

Our approach was to design and conduct a small experiment in which participants performed a usability test on a single interface. We were interested in observing whether the designed system is effective[2], which means whether the user is aware of the level of danger for the current state and is able to avoid it. In addition, we measured the user satisfaction[3] towards the danger indicators.

We ran a total of 10 participants, all between 22 and 28 years. They consisted of 3 female and 7 male students from different disciplines from Reykjavik University. Participants were asked to interact with the graphical interface presented in section 7.2.2. Such a small number of users usually is enough to redesign the system during an iterative user-centred design process [LFH10]. Thus, this is fine for discovering flaws in the interface and understanding whether the system is working according to the specific nature of the problem and users' expectations.

Figure 8.4: A user testing session

We described a typical scenario, such as a small cooking task involving a pot and a stove, as well as a flammable cleaning product. The environment consists of 3 cells. At the beginning, the user is in the first cell with a stove off. His goal is to pick up the pot in the

[2] The accuracy and completeness with which users achieve specified goals. ISO 9241-11:1998.
[3] Freedom from discomfort, and positive attitudes towards the use of the product. ISO 9241-11:1998.

third cell and avoid the bottle in the second cell as this may set up a fire whenever is put too close to the stove. For the purpose, we used a 3-step big sphere of protection and we warned the user a step before the potential danger. A complete summary of the setting is reported in the appendix A.1.

The virtual environment was used as low fidelity prototype of the scene, and we were interested in collecting user's perception of indicators and actual danger level. Indeed, informal user testing provides an inexpensive way to refine a design and enable the exploration of several alternatives that would otherwise be unaffordable.

Since we were interested in figuring out whether the user is aware of the level of danger and is able to avoid it, we defined a list of tasks to guide him during the simulation and be sure that he would get close to dangers and goals. For each task, we asked the following questions:

- Current status of indicators
  - What can you draw from the status of indicators?
  - Is the system displaying a danger?
- Perception of the current state
  - If so, what is the closest danger?
  - Do you feel in a potential danger?
- Expectations for the next state
  - What would you do now?
  - If you perform the next action, how do you expect the indicators to be?

After completing all the tasks, each participant filled out a questionnaire to rate the design and the usability of the danger indicators. For the purpose, we used a 5-point Likert scale, with "strongly agree" as left anchor and "strongly disagree" as right anchor. In addition, they were asked to express positive and negative aspects of the system, as well as suggestions for further improvements.

8.2.1 Results

Regarding the effectiveness of the system interface, the tests showed that all users relied on indicators for understanding the level of danger and the quality of the last performed action.

Users showed trust in the system and its indicators, justifying it as a need for guidance in the virtual and unknown setting. Consequently, they tended to perceive a danger where the system was displaying a danger, though most of them would not link certain situations to actual dangers. This means that dangers should be grouped in different categories, so as to handle them according to different levels of importance. Moreover, users tended to consider the difference between values of the same type to understand changes over the time, and they assigned more importance to the bar with the higher value. In addition, they tended to evaluate the quality of their behaviour by using the values of both bars during the time, that is, according to the formula:

$$f(t) = (A_t - B_t) - (A_{t-1} - B_{t-1})$$

f describes the user perception of the indicators, A is the bar with the highest value, and t is the time.

Regarding the alert message, all users noticed and used the hint to avoid the potential danger, though someone suggested improving the notification with an audio modality. However, some users found the early notification a bit intrusive for the kind of danger that the system was preventing. Indeed, despite all users agreeing with the need of an alert right before a potential danger, they suggested taking their habits into account, as they would never perform certain risky actions and, consequently, the system should not intervene for a behaviour that they would not take.

In addition, due to the simplicity of the task, most users considered the suggested action only for finding the pot and getting out of a dangerous situation, as well as a proof that their plan to achieve the goal was correct. Therefore, the higher the level of mental impairment (of the monitored user), the more useful we expect this functionality to be.

Results for the users' satisfaction are reported in the table 8.3 and figure 8.5. To get the mean values, we enumerated the answers from 1 (strongly disagree) to 5 (strongly agree). As the reader may notice, users found the action to avoid useful, though they claimed the intervention was intrusive when the dangerous situation that the system is trying to prevent is unlikely to happen. Users tended to rely on bars for their orientation, whereas they tended to reject the best suggested action by claiming this functionality was something they would not rely on during an actual task.

In conclusion, users suggested improving the warning system by taking into account their habits and preferences, as well as to limit its intervention to actual dangers. In general, users gave good comments about the interface and the system in the whole, rating their experience as not frustrating and the system as a useful aid to everyday life activities. However, most of them would not use it every day. Indeed, users tended to remark that the need of such a system in the presence of mental impairments was great. Accordingly,

Question	Mean	SD
It takes too long to learn the meaning of indicators	2.3	0.82
The organisation of the indicators seems quite logical	3.8	0.63
The colours used in the bars are helpful to understand their meaning	4.8	0.42
Suggesting the action to avoid is intrusive	1.5	0.71
Suggesting the action to avoid is useless	1.1	0.32
The action to avoid is suggested too late	1.3	0.67
The danger level bar is misleading	1.5	0.85
I do not understand when I am getting closer to dangers	1.1	0.32
The danger level bar follows modifications on actual dangers	4.2	0.63
Looking at the bars is not enough to understand the effects of my actions	2.8	0.79
I do not understand when I get closer to the goal	2.2	0.63
Suggesting the best action to perform is ineffective to achieve the goal	2.4	1.17
Suggesting the best action to perform is intrusive	2.7	0.95

Table 8.3: Results for the satisfaction with a 1-5 enumeration

the system could be seen as a notification tool and used to alert relatives or rescuers.

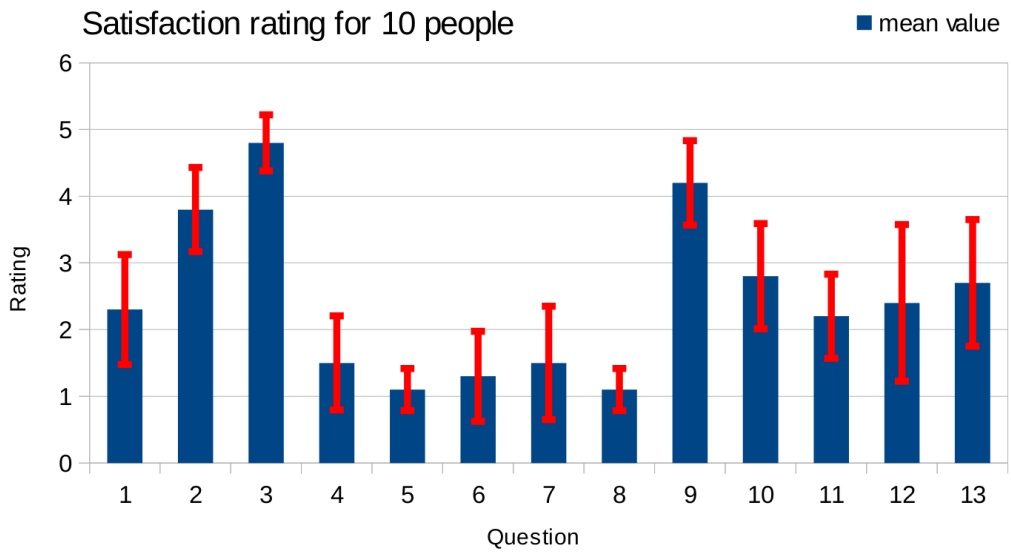

Figure 8.5: Users's satisfaction

Chapter 9

Conclusions

In this chapter we propose a short summary of results and achieved goals. We start by reporting our conclusions and then we suggest some further developments and work.

9.1 Conclusions

Daily-life activities at home can generate dangers that may lead to accidents. Risky situations may be difficult to notice by people with a cognitive or physical impairment. Therefore, recognizing dangers and alerting users is very important so as to assist them in preventing accidents, and ensure their health, safety and well-being.
Pervasive and ubiquitous technologies can be useful tools to entertain, monitor, assist and automate people's tasks in smart environments. Indeed, assistive technologies allow in-place ageing, and consequently improve the quality of life and help reducing costs of dedicated care-givers for institutions.

The present thesis aimed to design a system that, given a representation of the environment as input, learns how to evaluate states according to their danger level, and is able to alert and prevent users from getting too close to a potential danger. Therefore, the purpose of this project was to implement an early warning system as a decision maker that guides the user during his everyday life activities. We reduced the problem of preventing dangerous situations to a search problem where we explore the search space for disclosing dangers and finding a safe path leading to the goal.

The project led to the implementation of a working prototype of an early warning system. The system can be trained to suggest to the user the best action to perform for the current state, and it can report the level of danger given the distance from the dangerous state in

terms of actions. Moreover, the system returns a feedback about the last performed action by using the policy as indicator of the quality of state-action pairs, and it is able to warn the user when the number of actions to get to a danger exceeds a given threshold. Thus, the actual user can get inside one of the roles in order to simulate certain behaviours and observe system reaction. Indeed, we designed and implemented a complete platform for simulating everyday life activities described as game descriptions.

We presented an intelligent agent which is able to evaluate the danger level of states and act on the environment for notifying warning messages to users. This is a general solution, as the system is able to play arbitrary games described with the Game Description Language. For this purpose, we modified an existing GDL reasoner to handle descriptions oriented to the assisted living context. Moreover, the system learns how to behave from the scenario specification and no prior knowledge about the actual user is provided. Indeed, we used TD-learning methods to make the system autonomously compute a policy from the interaction with the environment model. We therefore applied Q-learning for providing a *general* (i.e. independent from the scenario) way to perform on-line planning in a stochastic environment. In particular, the warning agent is implemented as a tabular Q(λ) agent that exploits its experience for returning suggestions and warning notifications. For this purpose, we implemented a Java library called QBox, which is out-of-the-box framework for implementing TD-learning agents. Moreover, we defined the concept of sphere of protection and we implemented a variant of breadth-first search to search for dangerous states around the user and return the number of actions to get to the danger.
This functionality can still be improved. Indeed, we may want to speed this process up by using informed search methods, though this may require taking problem-specific properties into account for the state evaluation. Moreover, we used a static threshold for deciding when to intervene. A straightforward improvement to this approach may be to learn the intervention policy for deciding whether agreeing with or warning the user.

Regarding the user interface, we exploited virtual environments as a general testbed for simulating effects of warning notifications. The system may be implemented on mobile platforms such as pads, mobile phones and smart watches, so as to provide a direct feedback to a user acting in an actual environment. To this purpose, we showed how the system can be used to perform informal user testing, so that effects of warning notifications can be evaluated on actual users.

9.2 Future work

A straightforward improvement to this work is the implementation of a decision maker for learning when to intervene (see 6.2.2). This is required for limiting the system intrusiveness, as otherwise the user may be alerted for behaviours that the user would not take.

Another direct improvement is the use of a General Game Playing server for coordinating a distributed simulation of agents. In fact, we are already providing the possibility to define scenarios with multiple agents, though it is possible to get inside one role at a time. This means that the other roles will behave randomly and we are not able to simulate particular configurations such as a child getting close to his mother during a cooking session.

We decided to focus on the universality of the system, and this affected the quality of the simulation in terms of detail of objects in the scene. Therefore, we should find an alternative way to load models, perhaps in a 3D format, and we should take advantage of the grid for placing objects according to specific criteria rather than randomly. In this way, we can provide a navigable environment and reproduce certain interiors in a more realistic way, so as to use virtual environments as prototyping testbeds for assessing applications for smart environments.

9.2.1 Scaling the decision making

The use of a tabular Q-learning (i.e. hash-table) makes the solution infeasible for big state spaces. Thus, we should refer to further improvements for implementing a function approximator and tile coding. However, these approaches may require the designer to make problem-specific decisions such as features for representing a state or creating tilings. This would make the system no longer general and a solution could be to specify those features in the game definition. Our system can be considered as working prototype of an early warning system, though many improvements can still be discussed.

A problem with smart environments is the state explosion, as the complexity of the decision making problem increases exponentially with the number of devices in the environment. One solution is to aggregate data to reduce the amount of information to be considered. The problem can also be scaled by decomposing it into smaller problems that are assigned to cooperating agents in a multi-Agent environment. Agents cover different roles and may share resources (e.g goals and knowledge) in order to control different aspects of the environment in a distributed way. An example may be dividing the

environment-monitoring task into different kinds of danger and associate these categories to different agents. An important protocol used in the so called *Cooperative Distributed Problem Solving* (CDPS) is Contract Net [Woo09].

Another way to scale the decision making process is to directly decompose the problem space in a hierarchy of tasks and use hierarchical reinforcement learning. [BM03]. This allows to model problems in different abstraction levels so as to speed up the learning process and make the solution scale to big state spaces.

Moreover, the task of defining a reward signal is difficult, as reinforcement learning algorithms are very sensitive to variations and designers usually rely on their own experience. Apprenticeship learning [NR00, AN04] can be considered as a potential solution to this problem. The task to learn is demonstrated by an expert and the reward function is defined as a combination of features and learnt during the demonstration.

9.2.2 Embedded Systems

Reinforcement Learning has already been used for multiagent environments in previous work. However, the literature presents a lack of systems that are capable to provide specific reinforcement learning functionalities to engineers and designers of smart environments. In fact, embedded systems are the usual target platform of assistive applications in smart environments. Therefore, it is important to satisfy any resource constraint that these platforms may present such as computing and memory resources, as well as network availability and power. Therefore, a future work should consider the need of a middleware that implements context provisioning functionalities over a distributed network of resource-constrained devices, and enables developers to create intelligent applications by means of reinforcement learning agents.

Scaling the reasoning process

A game specification written in the Game Description Language (GDL) can be expressed by a graph representation of propositions and their relationships. *Propositional networks* are directed bipartite graphs that consist of nodes representing propositions connected to either boolean gates or transitions [CSMG09]. The dynamics of a multi-agent system can be represented by a *propositional automaton*, which consists of a propositional network, a truth assignment for the set of propositions (i.e. the initial state) and a legality function defining applicable actions for a given state. This representation allows a straightforward

discovery of indepent sub-games, thus reducing the game state space to a set of independent sub-games, that can be handled in a more convenient way. This is a clear advantage of this representation, and in addition, we can take advantage of programmable logic devices for implementing those networks on a physical device and speed the reasoning process up. Reconfigurable computing claims to fill the gap between software flexibility and hardware performance by means of computer architectures that can dynamically adapt the hardware to specific needs [HD07]. Therefore, we can rely on advances in this field for implementing propositional networks as logic circuits on programmable logic devices such as FPGAs[1]. A faster reasoner would allow to visit much more states and in consequence we would get a more accurate policy.

[1] Field Programmable Gate Array. It is an integrated circuit containing programmable logic components that can be connected for implementing complex functions.

Appendix A

Evaluating the solution

A.1 The game description

In this section, we report the setting that we used for performing informal user studies.

A.1.1 The parameters for the learning agent

The table A.1 reports the settings used for the learning agent.

Parameter	Value
α (learning rate)	0.2
α-decay	0.8
α-decay type	exponential (ensures convergence)
γ (discount factor)	0.95
λ (decay rate)	0.9
ϵ	0.5
ϵ-decay	0.9999
ϵ-decay type	exponential

Table A.1: The Q-learning agent parameters

A.1.2 The game description for a dangerous kitchen

The listing A.1 reports the description of a cooking task in a 3-cells kitchen.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Dangerous Kitchen
;;
;; Andrea Monacchi
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; --- Roles ---
(role user)

;; reactive agents
(device cooker)

;; objects
(object bottle)
(object pot)

;; --- Initial state ---
;; world size
(size 3 1)

;; position of objects and appliances
(init (at user 1 1))
(init (at cooker 1 1))
(init (at pot 3 1))
(init (at bottle 2 1))

;; state of appliances
(init (is cooker off))

;; User
(<= (legal user mover)
    (true (at user ?x ?y))
    (size ?xmax ?ymax)
    (smaller ?x ?xmax)
)

(<= (legal user movel)
    (not user_in_column_one))

(<= (legal user moveu)
    (not user_in_row_one))

(<= user_in_column_one
    (true (at user 1 ?y)))
```

```
(<= user_in_row_one
    (true (at user ?x 1)))

(<= (legal user moved)
    (true (at user ?x ?y))
    (size ?xmax ?ymax)
    (smaller ?y ?ymax))

(<= (legal user (take ?what))
    (true (at user ?x ?y))
    (true (at ?what ?x ?y))
    (object ?what)
    (not (holding ?what)))

(<= (holding ?what)
    (true (hold ?what)))

(<= (legal user (release ?what))
    (true (hold ?what)))

(<= (legal user (interact ?what ?action))
    (true (at user ?x ?y))
    (true (at ?what ?x ?y))
    (device ?what)
    (applicable ?what ?action))

;; -- Appliance state --
(<= (applicable cooker on)
    (true (is cooker off)))

(<= (applicable cooker off)
    (true (is cooker on)))

;; --- action effects ---

;; frame axioms for environment parameters
(<= (next (at ?what ?x ?y))
    (true (at ?what ?x ?y))
    (device ?what))

(<= (next (at ?what ?x ?y))
    (true (at ?what ?x ?y))
    (object ?what)
    (not (holding ?what)))
```

```
(<= (holding ?what)
    (true (hold ?what)))

(<= (next (at ?what ?x ?y))
    (true (at ?what ?x ?y))
    (not moves))

(<= moves
    (does user movel))

(<= moves
    (does user mover))

(<= moves
    (does user moveu))

(<= moves
    (does user moved))

(<= (next (hold ?what))
    (true (hold ?what))
    (not (released ?what)))

(<= (released ?what)
    (true (hold ?what))
    (does user (release ?what)))

(<= (next (hold ?what))
    (does user (take ?what)))

(<= (next (is ?what ?state))
    (true (is ?what ?state))
    (not (interact_with ?what)))

(<= (interact_with ?what)
    (does user (interact ?what ?action)))

;; state transitions
(transition cooker off on on)
(transition cooker on off off)

;; moving users
(<= (next (at user ?x1 ?y))
    (true (at user ?x ?y))
    (succ ?x ?x1)
```

```
    (does user mover))

(<= (next (at user ?x1 ?y))
    (true (at user ?x ?y))
    (succ ?x1 ?x)
    (does user movel))

(<= (next (at user ?x ?y1))
    (true (at user ?x ?y))
    (succ ?y ?y1)
    (does user moved))

(<= (next (at user ?x ?y1))
    (true (at user ?x ?y))
    (succ ?y1 ?y)
    (does user moveu))

;; moving objects
(<= (next (at ?what ?x1 ?y))
    (true (at user ?x ?y))
    (succ ?x ?x1)
    (true (hold ?what))
    (does user mover))

(<= (next (at ?what ?x1 ?y))
    (true (at user ?x ?y))
    (succ ?x1 ?x)
    (true (hold ?what))
    (does user movel))

(<= (next (at ?what ?x ?y1))
    (true (at user ?x ?y))
    (succ ?y ?y1)
    (true (hold ?what))
    (does user moved))

(<= (next (at ?what ?x ?y1))
    (true (at user ?x ?y))
    (succ ?y1 ?y)
    (true (hold ?what))
    (does user moveu))

(<= (next (is ?what ?newstate))
    (does user (interact ?what ?action))
    (true (is ?what ?oldstate))
```

```
    (transition ?what ?oldstate ?action ?newstate)
)

;; --- game definition ---
(<= terminal
    danger)

(<= terminal
    usergoal)

(<= (reward user 1)
    (not danger)
    usergoal)

(<= (reward user -0.8)
    (not usergoal)
    danger)

(<= (reward user -0.01)
    (not danger)
    (not usergoal))

(<= (reward user 0)
    danger
    usergoal)

;; definition of dangers and goals
(<= danger
    (true (is cooker on))
    (true (at bottle 1 1))
)

(<= usergoal
    (true (at pot 1 1))
    (true (is cooker on))
    (not (holding pot)))

;; --- auxiliary ---
(<= (smaller ?x ?y)
    (succ ?x ?y))

(<= (smaller ?x ?y)
    (succ ?z ?y)
    (smaller ?x ?z))
```

```
(succ 0 1)
(succ 1 2)
(succ 2 3)
(succ 3 4)
(succ 4 5)
(succ 5 6)
(succ 6 7)
(succ 7 8)
(succ 8 9)
(succ 9 10)
```

Listing A.1: The game description for a dangerous kitchen

Bibliography

[AN04] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[Bel57] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[BF09] Y. Bjornsson and H. Finnsson. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, March 2009.

[BHP+06] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. A planning system based on markov decision processes to guide people with dementia through activities of daily living. *Information Technology in Biomedicine, IEEE Transactions on*, 10(2):323–333, April 2006.

[BM03] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, October 2003.

[Boy11] Andrey Boytsov. *Context Reasoning, Context Prediction and Proactive Adaption in Pervasive Computing Systems*. PhD thesis, Luleå University of technology, 2011.

[BS07] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 672–677, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[CD04] Diane Cook and Sajal Das. *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[CSMG09] Evan Cox, Eric Schkufza, Ryan Madsen, and Michael R Genesereth. Factoring general games using propositional automata. *Network*, 2009.

[CYH+03] Diane J. Cook, Michael Youngblood, Edwin O. Heierman, III, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. Mavhome: An agent-based smart home. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, PERCOM '03, pages 521–, Washington, DC, USA, 2003. IEEE Computer Society.

[Dar09] Waltenegus Dargie. *Context-Aware Computing and Self-Managing Systems*. Chapman & Hall/CRC, 1 edition, 2009.

[FB08] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1*, AAAI'08, pages 259–264. AAAI Press, 2008.

[GC03] Karthik Gopalratnam and Diane J. Cook. Active lezi: An incremental parsing algorithm for sequential prediction. In *In Sixteenth International Florida Artificial Intelligence Research Society Conference*, pages 38–42, 2003.

[GLP05] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[HD07] Scott Hauck and Andre DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[HvBPM07] J. Hoey, A. von Bertoldi, P. Poupart, and A. Mihailidis. Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *International Conference on Vision Systems (ICVS)*, 2007.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[KLM96] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kru09] J. Krumm. *Ubiquitous computing fundamentals*. Chapman & Hall/CRC Press, 2009.

[KTNK08] Hideaki Kanai, Goushi Tsuruma, Toyohisa Nakada, and Susumu Kunifuji. Notification of dangerous situation for elderly people using visual cues. In

Proceedings of the 13th international conference on Intelligent user interfaces, IUI '08, pages 345–348, New York, NY, USA, 2008. ACM.

[KWA10] A. H. Khalili, Chen Wu, and H. Aghajan. Hierarchical preference learning for light control from user feedback. pages 56–62, June 2010.

[LBP+09] Yong Lin, Eric Becker, Kyungseo Park, Zhengyi Le, and Fillia Makedon. Decision making in assistive environments using multimodal observations. In *Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '09, pages 6:1–6:8, New York, NY, USA, 2009. ACM.

[LFH10] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research Methods in Human-Computer Interaction*. Wiley Publishing, 2010.

[LHH+08] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical report, Stanford Logic Group, March 2008.

[Lit94] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[May04] Rene Mayrhofer. *An Architecture for Context Prediction*. PhD thesis, Johannes Kepler University of Linz, Austria, October 2004.

[MM98] Michael Mozer and Debra Miller. Parsing the stream of time: The value of event-based segmentation in a complex real-world control problem. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello"-Tutorial Lectures*, pages 370–388, London, UK, 1998. Springer-Verlag.

[Moz98] Michael C. Mozer. The neural network house: An environment that adapts to its inhabitants. pages 110–114, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence Spring Symposium on Intelligent Environments.

[NFS+09] Juan-Carlos Naranjo, Carlos Fernandez, Pilar Sala, Michael Hellenschmidt, and Franco Mercalli. A modelling framework for ambient assisted living validation. In *Proceedings of the 5th International on ConferenceUniversal Access in Human-Computer Interaction. Part II: Intelligent and Ubiquitous Interaction Environments*, UAHCI '09, pages 228–237, Berlin, Heidelberg, 2009. Springer-Verlag.

[NMF05] Petteri Nurmi, Miquel Martin, and John A. Flanagan. Enabling proactiviness through context prediction. In *Workshop on Context Awareness for Proactive Systems, 2005 (CAPS 2005)*, pages 159–168, Helsinki, Finland, June 2005. Helsinki University Press.

[NR00] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Pos09] Stefan Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley Publishing, 1st edition, 2009.

[RN10] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[SP04] Thomas Strang and Claudia L. Popien. A Context Modeling Survey, September 2004.

[ST09] Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In *In: ICAART*. Springer, 2009.

[Sze10] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[Tes95] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, 1995.

[TGM11] Alessandra Talamo, Sabina Giorgi, and Barbara Mellini. Designing technologies for ageing: is simplicity always a leading criterion? In *Proceedings of the 9th ACM SIGCHI Italian Chapter International Conference on Computer-Human Interaction: Facing Complexity*, CHItaly, pages 33–36, New York, NY, USA, 2011. ACM.

[Thi11] Michael Thielscher. The general game playing description language is universal. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1107–1112, Barcelona, 2011. AAAI Press.

[TYZ+11] Lei Tang, Zhiwen Yu, Xingshe Zhou, Hanbo Wang, and Christian Becker. Supporting rapid design and evaluation of pervasive applications: challenges and solutions. *Personal Ubiquitous Comput.*, 15:253–269, March 2011.

[WB97] Mark Weiser and John Seely Brown. *The coming age of calm technology*, pages 75–85. Copernicus, New York, NY, USA, 1997.

[Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3:3–11, July 1999.

[Woo09] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539