Machine learning in Intelligent Buildings

Andreas Møller s042809, David Emil Lemvigh s042899

Kongens Lyngby 2012

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary (English)

The purpose of this thesis is to explore the possibilities of developing a low cost intelligent home control system, capable of reducing the power consumption in normal households. This control system will be based on the core concepts of smart environments. The thesis will serve as a research paper on the possibilities of using machine learning algorithms to develop an advanced artificial intelligence, capable of controlling a house hold, and reducing power consumption. We have created a prototype of a smart environment, that serves as a proof of concept, and can be used as the basis for further development. The final product shows the power of ubiquity computing, as a means of reducing energy consumption in the normal household.

Summary (Danish)

Formålet med denne afhandling er at udforske mulighederne, for at udvikle et billigt intelligent lys styrings system, der er i stand til at reducere energi forbruget i normale hjem. Dette system er baseret på de centrale ideer bag smart environments. Afhandlingen vil undersøge mulighederne, for at anvende machine learning algoritmer, til at udvikle en avanceret kunstig intelligens, der er i stand til at kontrollere lyset i et almindeligt hjem, og samtidigt reducere energi forbruget. Vi har udviklet en prototype der kan se som et "proof of concept", og som kan anvendes som udgangspunkt for videre udvikling. Det endelige produkt demonstrerer fordelene ved ubiquitous computing, som middel til at reducere energi forbruger i normale hjem.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Informatics.

The purpose of the thesis is to examine the possibilities of incorporation machine learning in home control systems.

The thesis consists of a comprehensive analysis of the problems involved in the integration between these two technologies. This is followed by a description of the design and implementation process of developing the experimental system. Finally the report is concluded by an evaluation of the results optained.

Lyngby, 24-Febuary-2012

Andreas Møller s042809, David Emil Lemvigh s042899

Acknowledgements

We would like to thank Sune Keller and Martin Skytte Khristensen for developing the smart house simulator we have used to evaluate our project. We would also like to thank Mads Ingwar, Elisa Wangsgaard Andreasen and Rasmus Møller for providing feedback and great support. Most of all we would like to thank our advisor Christian Damsgaard Jensen for providing great support and advice throughout the project.

Contents

Chapter 1

Introduction

In the recent years we have seen an increase in climate awareness. Environmental issues such as global warming, are widely debated both on a political and personal level, and the subject is gaining increased media coverage. A survey conducted from 2007 to 2008 by the international research organization Galllup[1] shows that 82% of americans and 88% of europeans are very aware of the current climate issues we are facing (1, gallup–2009). In the same survey Gallup also concludes, that 67% of americans and 59% europeans view global warming as a serious threat to them selves and their families. With the rise of concern with the general public, the demand for sustainable solutions increases. We are already seeing a large number of companies, spending a considerable amount of money to be classified as environmentally conscious. Companies such as Amazon are spending millions of dollars on sustainable buildings, in order to maintain an image as an environmentally conscious company.

In the residential sector the environmental awareness id equally present, but the so called "green wave"[^green wave] has not had nearly the same commercial impact. This is however not due to lack of potential. According to the United States Energy Information Administration[2], the residential sector constituted 22% of the total energy consumption in the US (2, eia–2011). The main problem

[1] International research organization famous for their large scale international polls. http://ww.gallup.com

[2] http://www.eia.gov/

in this sector is financial. Improving your residence to be more environmentally friendly is costly, and though most improvements generally pay for them selves over time, the return of investment will often take several years. This problem is not nearly as big in the business sector, where the gain in public image can be very valuable, and may even be worth the investment in it self. In the residential sector, however, the financial benefits of installing environmentally friendly technology solely come from the reduction in energy consumption.

There is a lot of focus on saving energy by changing habits, such as remembering to turn off the light on the bathroom, or not using the standby feature on many appliances. All these initiatives certainly help, but if we want to make a significant reduction in our energy consumption we need smart environments[^smart-envirionments], that are capable of micro managing our energy use.

The idea of smart environments is a product of the concept ubiquitous computing[^ubiquitous computing], a term invented by the late computer scientist Mark Weiser[^weiser]. Weisier coined the term while working as chief technologist at the Xerox Palo Alto Research Center (PARC)[^parc]. Ubiquitous computing proposes a new paradigm in human-computer interaction, where the role of the computer is to serve the users, rather than act as a tool that requires direct interaction. Smart environments fulfill this role, by monitoring its users, and acting on their behalf, without the need of their active participance. It is described by Mark Weiser as:

"a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network" -Mark Weiser

The concept of smart environments originated in 1988, but in the recent years we have seen a large development in this field. This in mainly due to the development in computing power, and availability of embedded systems, which lies at the heart of the smart environments.

The purpose of this thesis is to explore the possibilities of developing a low cost intelligent home control system, capable of reducing the power consumption in normal households. This control system will be based on the core concepts of smart environments. The thesis will serve as a research paper on the possibilities of using machine learning[^machine-learning] algorithms to develop an advanced artificial intelligence, capable of controlling a house hold, and reducing power consumption. We have created a prototype of a smart environment, that serves as a proof of concept, and can be used as the basis for further development. The final product shows the power of ubiquity computing, as a means of reducing energy consumption in the normal household.

The thesis will focus solely on lighting control in the smart environment. This allows us to focus on the integration of machine learning, rather than adding a large array of functionality. The advantage of focussing on lighting compared to other aspects, is that we are provided instant visual feedback when manipulating the environment.This will prove very useful for development and testing purposes. The core concepts of controlling the light will be similar to many of the other task that can be handled by a smart environment, therefore the solutions developed as a result of the work done in this thesis, will be transferable to other areas, such as heating regulation, air-conditioning, etc.

The thesis is structured as follows:

In the chapter "Analysis" we will identify and analyze the problems and issues, related to developing an intelligent home control system. This involves analyzing existing solutions and technologies related to the technological field of smart environments.

In the chapter "Design" we will discuss our solutions to the problems identified in the analysis. We will also briefly present the development process, and how this have affected the final product. This chapter will also hold a theory section, where we will discuss the most important technologies we have used, along with the mathematical theory that forms the basis for our solution.

The "Implementation" chapter examines the transition from a software blueprint to working code. In the chapter, we will in detail describe the problems we had to solve when coding the system.

Finally we will evaluate the results of our research in the chapter "Evaluation". The chapter will both evaluate our solution and contain a description of the software tests we have performed on the system.

Chapter 2

Analysis

"If I have seen further it is by standing on the shoulders of giants" – Isaac Newton

The first task in any project is to analyze the problem at hand. Before attempting to design an intelligent home control system, we must first identify which problems that may arise when developing a system like this. These problems may be related to the general field of home control systems, or they may be arise with the introduction of machine learning. In this chapter we will also clearly define what features we want in the system, and what features we do not want. Some features will also be excluded to avoid spreading the focus of the project too thin. Features ignored with the purpose of limiting the project scope will be discussed in the section "Future work" in the "Conclusion" chapter.

The project contains a large element of unpredictability, as a result of incorporating machine learning based on real life data. As a result the development process will be a repeating cycle where one iteration will look as follows:

$$\textit{Development} \rightarrow \textit{Training} \rightarrow \textit{Evaluation}$$

First the system is developed, then the system must be trained based on collected data, and finally we can evaluate on our solution. This cycle will then continue through out the development phase.

In this chapter we will only discuss the problems that we have been able to identify in the initial stages of the project. Some problems have arisen in the evaluation stage of the first development cycle. These problems will be discussed in the “Design” chapter, since analyzing them requires some knowledge of how the system functions. The general concept of development cycles will also be discussed further in the “Design” chapter.

With each problem discussed in this chapter we will briefly present our solution strategy, and discuss relevant alternatives.

We will start out with a small representative survey of existing systems, both available on the commercial market, and in development.

We will then, in relation to the findings in the survey, discuss both the problems we have found with the existing solutions, and those related to developing a smart environment based on machine learning.

2.1 Smart House Survey

The beginning of any good project starts with a survey of what already exists.

In the following section we present a short survey of what already exists in the field of home control systems and smart environments. We evaluate the existing solutions and their capabilities, and review the industry standards. This section is intended as a representative selection of smart environments, and thus will not contain an exhaustive survey of all existing solutions on the market. First we will establish some basic classifications of smart houses, to better compare the different systems. All systems can contain switches, sensors and remote controls, the difference is the functionally they provide, and how they operate. We classify the smart environments into three categories, Controllable, Programmable and Intelligent. These categories are based on the taxonomy presented in Boguslaw Pilich’s Master Thesis and we refer interested readers to (3, Boguslaw–2004)

2.1.1 Controllable houses

These are the simplest of the home control solutions. Input devices such as switches, remotes and sensors, can be setup to control output devices such as appliances, dimmer switches and HVAC (Heating, Ventilation and Air Conditioning), etc. These solutions may also include macros, e.g. where a single button may turn off all the lights in the home.

2.1.2 Programmable houses

These solutions incorporate some degree of logical operations, like having motion sensors only turn on the lights, if lux[1] sensors are below above a certain threshold. They may be able to have scheduled, tasks e.g adjusting the thermostats during standard work-hours. The behavior of these systems have to be programmed by the manufacturer or the users. Consequently, changes in user needs require the system to be reprogrammed.

2.1.3 Intelligent houses

In these solutions some form of artificial intelligence is able to control the home. In computer science the term artificial intelligence is often used loosely. For the purpose of this thesis we will define an intelligent house, as a system that is capable of machine learning. That means that the system is capable of evolving behavioral patterns based on empirical data (4). Consequently, the system will over time adept itself to changes in user needs.

The solutions presented, are some of the most widespread smart house solutions, and represents the three different types of systems: Controllable, Programmable and Intelligent houses.

INSTEON

INSTEON

Figure 2.1: INSTEON logo

INSTEON is a controllable home control system, targeted at private homes.

[1] A device for measuring the amount of light in a room.

Nodes in the network can communicate using either RF signals or home's existing electrical wiring. A standard array of devices are supported:

- Dimmers & switches
- HVAC
- sprinklers
- motion sensors
- assorted bridge devices

INSTEON supports external applications to be run on a PC connected through a bridge device to the network. By extension it is technically possible to extend the system with a programmable or even intelligent component. However no commercial products providing these features currently exists. (5)

INSTEON's solution is fairly widespread in the US. It represents what a commercial controllable house is capable of. The systems functionality is very limited, but being able to communicate using the home electrical wiring, makes it a very non-intrusive system to install in an existing home. It enables the user increased control of his home compared to the regular wall switches, by allowing him to control his home with remote controls and motion sensors. The INSTEON system is limited by its lack of intelligence or programmable logic, and can only perform simple actions based on user inputs.

Clipsal (C-Bus)

Figure 2.2: Clipsal logo

Clipsal is a large scale control system, targeted at the industrial sector. The system is installed in such prominent buildings as the Sydney Opera house, Wembly Stadium and many more. Nodes communicate over its own separate wired network, using the C-Bus protocol. Each node has its own microprocessor, allowing for distributed logic. This means each node can be individually programmed, allowing a wide array of different devices to be added to a Clipsal system without having to modify the c-bus protocol. This allows unconventional devices like motors for stadium roofs and many others, to be part of the network. Nodes can also be programmed to autonomously control the system, e.g.

in a hotel a control unit in each apartment could monitor temperature sensors, control ventilation and heating, while also logging power to a central database.

Clipsal represents the flexibility and scalability, programmable solutions on the market are able to achieve. A very unique feature of Clipsal is the distributed logic. Most programmable systems are essentially controllable systems witn a central logic, where every other node in the system acts as slave nodes[ˆslave-nodes]. With the microprocessors in each node, logic can be distributed over a multitude of nodes, allowing nodes to be in charge of subsections of the system. The distributed logic can also remove the problem of a single point of failure, where a single faulty node can prevent the entire system from working. This results in a more robust and fault tolerant system.

All of the features of the Clipsal system comes at a price. The system requires a wired communication network, and programming nodes to individual needs requires professional expertise. This is a negligible price to pay for a larger corporation, compared to the features it provides, but makes the system very expensive for a private user. (6)

LK IHC

Lauritz Knudsen LK
by Schneider Electric

Figure 2.3: LK logo

LK IHC is targeted at private homes. It can be installed with a wired network, or using wireless communication. This solution tends to be build around simple wall switches, but with programmable scenarios. An example of this could be having a switch near the front door and the master bedroom that turns off all lights. The IHC is a modular system, where modules like wireless communication or alarms, can be added to the base installation.

The IHC modules includes a programmable logic controller[ˆplc] which allows the system to be programmed. An example of this taken from their own presentation of the product is that motion sensors that normally are set to control the lights could, if the alarm is activated, be programmed to dial 911. LK IHC was per 2008 installed in nearly 30% of newly constructed building in denmark. (7, Ingwar-jensen–2008) (8).

While the programmable logic controller provides an extended list of possibilities, programming the PLC requires a great deal of technical expertise and is not a feasible task for the average end user.

Zensus Z-Wave

Figure 2.4: Z-Wave logo

Zensus develops a communication standard for wireless communication between nodes in smart environments. Z-Wave is a protocol, like C-Bus is for the Clipsal system. Zensus only produce the communication hardware and the protocol, and leave it to other companies to produce the actual smart house products. Companies who produce devices for the system, have to get them certified by Zensus to ensure all Z-Wave certified can communicate with each other. This means there is no single supplier of Z-Wave products, and products from different suppliers are freely interchangeable. Like the Clipsal system, Z-Wave devices can have distributed logic. Depending on the products added to a Z-Wave system, the system can be controllable or programmable.

Some companies sell complete smarthouse solutions based on the Z-Wave, with switches, sensors, remotes, et cetera. Other companies instead focus on their normal market segment, producing Z-Wave cerfied versions of their products. The Danish company Danfoss, produces thermostats which can control the temperature based a schedule, and turn off the heating if windows are opened.

The hardware we've had available for this thesis was Z-Wave switches, sensors and USB-dongles. This allowed us to make a setup where the a PC could send and receive messages to a Z-Wave network. This allows us to create an Intelligent smart environment, based on Z-Wave hardware.

The devices in a Z-Wave system are freely interchangeable, allowing a user to tailor the system to specific needs. Based on commercially available products, a Z-Wave system can be build to be controllable or programmable. There aren't any commercially available products to make Z-Wave an intelligent system.

Aware Home Research Inititive

Figure 2.5: AHRI logo

AHRI[ˆahri] differs from the previous systems, as it is not a finished implementation, but a framework for a research projects. There are not any widespread commercially available intelligent smart house solution on the market, or at least that satisfies our classification of intelligent.

AHRI represent one of many smart environment, build by universities around the world. The smart environments usually houses one or more inhabitants, and are part of a living laboratory. Part of AHRI is a living laboratory, a thre story house, inhabited by volunteers for varying lengths of time. These homes are designed for multi-disciplinary studies, of people and their pattens and interactions with new technology and smart home environments. Being university run smart homes, the work coming out of these facilities tends to be proof of concepts. This means there are no complete product based on these projects. (9)

Like the Clipsal system, the nodes of AHRI have distributed logic, but are also able to learn from user behavior. The system is also able to relay data gathered by the system, to PDA's or smartphones carried by the inhabitants of the house.

The intelligence of AHRI comes from the work of each team of master students working on the project. Each project explores different aspects of machine learning. The exact intelligence implementation of House_n is dependant on the currently ongoing projects (10)

The projects shown in this survey represent the solutions currently available or in development. There are many different controllable and programmable solutions commercially available, with INSTEON, Clipsal C-bus and LK IHC being some of the more widespread solutions. INSTEON represents the domain of controllable houses. Clipsal C-bus and LK IHK are both programmable home control solutions, but where LK IHC is designed for private homes, the Clipsal C-bus system is better suited for larger buildings.

AHRI in this survey represents that truly intelligent smart houses only exists in demonstration environments and as proofs of concept, and are not yet available on the commercial market.

One of the general problems with current home control solutions is that purchasing such a system is rather costly and requires both installation and configuration, which is rarely trivial. Some of the more advanced systems on the market, such as the LK IHC, incorporate motion sensors and timers that automatically turn on and off lights or various appliances. These systems will save money over time, but they require extensive configuration or programming in order to function properly.

As mentioned in the introduction, the main focus of our project is reducing energy consumption. This is an area where most modern home control systems falls short . Most systems are capable of providing only a modest reduction in power consumption, and some even increase the net consumption by adding the cost of running the control system. We want our system to differ from others on this specific aspect. In our system, reducing power consumption is the number one priority.

2.2 Our solution

Based on the result of our survey, and the vision for our project highlighted in the introduction, we can now begin to identify and analyze the the problems that our solution must address.

Our main objective is reducing energy consumption. This is one area where many of the control systems in the survey falls short. Only AHRI have made a viable attempt, at addressing the issue of energy sustainability. We want to go even further, by developing a smart environment capable of micro managing the energy consumption of the house.

We will accomplish this by creating a system that focuses on turning off all light where it is not needed. There are several advantages to this approach, compared to attempting to reduce the power consumption of active appliances. The main advantage is that it provides the largest potential for reduction in energy consumption. Most people remember to turn off the light in the bathroom, when they leave it, but this is far less common for the kitchen, or dining room, and only the most environmentally conscious people would ever turn off the light in the living room when they got to the bathroom. This means that there is a lot of wasted energy in the normal household, and therefore a large potential for optimization.

An other advantage is that it incorporates perfectly with most other power reducing technologies. Buying lamps and appliances that use less energy will still give you the same percentage of power reduction as in a normal house. This makes it a very sustainable approach to energy reduction, that does not run the risk of being outdated, by new technology.

Though we focus on controlling the lights in the house, the system must also be scalable so that it, in the future, can incorporate other aspects, such as heating, ventilation, and electrical appliances. This system will also eliminate the common problem of standby mode on many appliances such as TVs or stereos by having the appliance only in standby mode, when the user is likely to turn it on. The rest of the time the appliance is simply turned off.

In the spirit of ubiquitous computing, we want the users interactions with the system, to be as simple and familiar as possible. The user should only interact with the system through the wall mounted switches, that are already present in all normal households.

Our approach is inspired by AHRI, and similarly we will develop an intelligent system, capable of predicting what the user wants it to do. The system will accomplish this by learning from what the user does and mimic these actions at the right times. To accomplice this, the system must be do three things:

- The system must gather data on the user and his behavior in the house
- The system must analyze the data in order to build a decision scheme[^decision-scheme] on which it will base its actions
- The system must be able control the house in real time, based on the decision scheme.

2.3 Gathering data on the user

To mimic user actions, the system must first gather information on how the user interacts with the house. Therefore the first question we must answer is: What data should we collect on the user? In order for the system to effectively take over the users direct interactions with the house, we need to know two things.

- What action needs to be done?
- When shall the action be done?

The first question can be answered by monitoring the users direct interactions with the house. Since we have limited our system to handle lighting, this means, monitoring the users interactions with the light switches.

The second question is a lot more complex. We need to collect data that can help us determine, if the conditions are right for performing a specific action. We could quite literally look at the time the action is performed, and then use that as a trigger, but this requires that the user follow a very specific schedule.

To get a more detailed picture of when an action should be performed, we must analyze it relative to what the user is doing at the time. Since we are focussing on lighting, this can be done simply by tracking the users movements. Thereby we will determine when an action shall be done based on where the user is, and where he is heading.

Perhaps the most obvious way of accomplishing this is by using cctv cameras. Using visual analysis is the most effective way of monitoring the user, as it will provide us with vast amounts of data on what the user is doing. By, for example, installing a fisheye camera[2] in every room, and use motion tracking on the video data stream, we can determine exactly where the user is, and what he is doing. While this is probably the solution that provides us with the most precise and detailed data, it does pose one problem. Installing cameras in every room of the users house is, in our opinion, an unnecessary invasion of the users privacy. Even if the video data is not stored in the system, the presence of cameras will give many people the feeling of being watched in their own homes.

An other approach would be to use a token worn by the user that sends out a digital signal. The system could then use multilateration[3] to pinpoint the exact location of the user. The token could be attached to the users keychain or built into his cellphone. Like the camera approach this solution also has very high precision, in tracking the user through the house. However, besides the point that the user might not always carry his keys or cellphone around, the main issue with this solution is scalability of users. Even though we limit the system to one user for now, we want a system that can be scaled to accommodate multiple users. Having to attach a token to every visitor coming into the house is gonna be an annoyance, and without it the house would not react to the visitor at all.

The solution we chose is to use motion sensors. While this solution does not provide nearly the same precision in determining the users location as using fish eye cameras or multilateration, motion sensors does come with a range of other advantages. Motion sensors are very cheap, compared to installing cctv cameras, and will be far less invasive on the user's privacy. The motion sensor solution will also work for any user in the house, and does not require the user to carry any beacon device like in the multilateration system.

The system could easily be expanded by several other types of sensors as well. E.g. pressure sensors in the furniture, so the system can determine if there is someone present, even when motion sensors do not register them. There are several other examples of sensor technologies that could be incorporated in the system. Some of these will be discussed in the section 'Future work' in the

[2]A ceiling mounted camera with a distorted lens that allows for a wider viewing angle.

[3]Multilateration is a navigation technique based on the measurement of the difference in distance to two or more stations at known locations that broadcast signals at known times.

“Conclusion” chapter.

For the moment we want to use as few hardware components as possible. There are two reasons for this:

- We want to keep the system as simple as possible from the consumers perspective. That means a system with as few components as possible.
- Creating a system that analyses and mimics user behavior will have a lot of unknown variables that, are hard to predict no matter how it is implemented. It will therefore be preferable, to start out with a system that is stripped down to the bare necessaries, and then add components as the need for them arises.

Because we want a system that is easy to install and configure, we have chosen not to inquire any information on the position of the motion sensors in the house. This means that the system does not know where each sensor is located, nor which other sensors are in the same room as it. This does make analyzing the data a lot more complicated, but we want to stick with the idea of minimizing the installation and configuration. This way the installation process can be boiled down to putting up the sensors, plugging in the system, and pressing “Start”. This also simplifies the maintenance of the system, when for example the user needs to replace a faulty sensor. This is again subscribes to the idea of smart environments, that are created with the purpose of simplifying the users life.

Choosing to only monitor the light switches and using motion sensors to track the user, greatly simplifies the data collection. Both the motion sensors and the switches generate events when they are triggered, and the system should simply store these events in a database.

An alternative to this is to have the system analyze the data live, which would eliminate the need to store the event data. With this approach we do not have to store the events in the system, which over time could accumulate to a considerable amount of data. The problem is that if we should choose to modify the algorithms that analyze the data, we would effectively loose everything the system has learned so far. By storing the raw event data we can always recalculate a new decision scheme based on the collected data. This solution leaves us with a lot more options later on. The collection of data must still happen in real time. Since it is very important that the events are recorded exactly when they happen, the system must not stall in this process.

Since the project serves as a proof of concept for the idea of an intelligent house, we will need to collect real user data in order to properly evaluate our system. This is a necessary step in order to draw any meaningful conclusions on the system. There are two reasons for this:

- If we use generated data the house is not actually intelligent, it is merely acting on data created by the developers. The data we could supply the house would be based on how we think the user would behave. As developers it would be almost impossible not to be bias towards a behavioral pattern that is easy for the house to interpret, rather than how an actual user would interact with the house.
- The project had a very large unknown element when we started out. No system quite like it, have ever been created before, and it is almost impossible to predict how the system will react to different inputs. Though we are creatures of habit, our movement patterns do not run like clockworks. No matter how well we would generate training data using simulators, algorithms or any other artificial method, there would always be a doubt on how close to actual human behavior it actually is.

We have chosen create a fully functional physical installation, since this would take away too much focus developing the actual software system. Instead we opted to install a “placebo” system[4] of wireless switches and sensors, to collect training data. This gives us the best quality training data for the system, without the expenses of installing operational wireless switches. With this training data, we can then use a simulator to evaluate that the system is learning properly. The simulator cannot replace the physical installation, collecting real life data collection, but it is an excellent tool when used for evaluation purposes.

The data collection system must be able to capture events in real time, since the system must know the exact time an event occurs, as well as the relative time between events. This part of the system must therefore be optimized towards a fast runtime.

2.4 Analyzing the collected data

Now that we have collected a lot of data on our users interactions with the house, we need to analyze the data in order for our systems AI to act on the

[4] A system where the sensors and switches have no actual effect on the house, but are merely there to collect data.

collected data. To be more specific: We need to create a decision scheme, that the AI can use as a base for its decision making.

This is the critical part of the system. Collecting data, and acting based on an existing scheme are both relatively simple tasks, however, designing the decision scheme, based on collected data, is far more complicated.

The purpose of analyzing the data is to find which specific situations that require the system to perform an action. Since the system does not know which sensors are located near which switches, the system will have to learn these relations based on the data collected. The simplest solution would be to have the system learn which switches and which sensors are located in the same room, and then create a "link" between them so the motion sensors control the light. This would result in what we have named the silvan[5] system.

The silvan system is basically having a motion sensor turn on the light when triggered, and then to have a timer turn off the light if the sensor is not triggered for a set amount of time. The main problem with this kind of system is, that if the user does not trigger a motion sensor regularly, the light will turn off even if the user is still in the room. This is commonly a problem in a room like the living room, where the user is likely spend an extended amount of time sitting still. This problem can be addressed by extending the light's timeout time.

However, this brings us to the second problem. If the user is merely passing by a sensor, the light will still be turned on for its full duration. This greatly reduces the effectiveness of the system from a power saving point of view. When extending the timeout time of the system, this problem escalates.

A better solution is to attempt to identify the users behavior leading up to a switch event[6]. Since the system only use motions sensors to track the users movements, these sensor events will form the basis for the data analysis. The system could simply look at what sensor was triggered right before a switch was activated, and then create a link between that sensor and the switch. This, however, would result in a system much like the silvan system described above.

If we instead look at a series of sensor events leading up to a switch event, we will get a much more complex picture of what the user is doing. Since the switches in the house are located in fixed positions around the house, these movement patterns should repeat themselves relatively often. The movement patterns that lead up to a switch being turned off, will most likely also differ from a pattern leading up to a switch being turned on, since the user will be either entering or

[5] Danish building material retail-chain.

[6] An event generated in the system, by the user turning a switch on or off.

exiting the room. Once we have analyzed the data and identified the movement patterns related to a switch event, we need to create a decision scheme that the system can base its decision making on. That means we have to organize the analyzed data in a way so we easily can look up a specific pattern, and see whether it should perform an action.

This concept of event patterns is also the main reason we have chosen to only use data from a single user house. Having multiple users interacting with the house simultaneously will break the patterns, and make it much harder for the system to identify which situations should lead to an action. There are several ways of handling this issue, some of which will be discussed in the section "Future work" in the "Conclusion" chapter.

Unlike data collection, analyzing the data does not have strict time constraints. Since the decision scheme will be based on data collected over an extended period of time, the system will not benefit from having the decision scheme updated in real time. As a result the time constraints on analyzing the data will be quite loose, and should not pose as a restriction on the system.

2.5 Controlling the house

After we have collected and analyzed data, the final task is to have the system control the house in real time, using the decision scheme created from the analyzed data. This is done by having the system constantly monitor the user, and attempt to match his movement pattern to those present in the decision scheme. As with data collection this has to happen in real time so the patterns are not corrupted.

Chapter 3

Design

The best computer is a quiet, invisible servant. -Mark Weiser

In this chapter we will describe the design process, and discuss the major decisions we have made in regard to the system design. Since the system is research minded, and since the purpose of the project is to analyze the possibilities of developing an intelligent home control system, using machine learning technology, we had to make some adjustments to the development process. The traditional waterfall model[ˆwaterfallmodel] for software development dictates that after finishing the project analysis, we would start designing the systems architecture, and how the system should handle the problems found in the analysis. Finally we would then implement the designed solution. With this project we were however faced with an additional challenge. When using machine learning you generally end up with a system that does not have an intuitive execution flow. This means that it can be almost impossible to predict the execution outcome because of the vast amounts of data that form basis for the systems decision making. This subject was briefly discussed in the previous chapter, where we discussed the cyclic development structure of the project. Because of this structure we have no way of verifying the validity of our proposed solution before implementing the system, or at least parts of it. Therefore we decided to approach the project by using incremental development instead[ˆincremental-development].

In order to successfully apply this development model, we must first divide the project into smaller parts, that can be implemented with each cycle. This design approach also inspired our final system design. Just like the development had several phases, where each phase had to be concluded in order to activate the next, the system will have have different stages of operation. These stages are determined by the amount of data the system have collected on the user.

The system will have two different stages of operation.

- In **The passive learning stage**, the system is running, but it has not yet collected enough data to make intelligent decisions. This stage is called the passive learning stage because the system is training it self solely by monitoring the user.
- The system enters **the active learning stage** when there's enough data to attempt to manipulate the switches in the house. We call this the active learning stage, because the system now actively attempts to interact with the house's switches . If the system makes a mistake and the user corrects it, e.g., the system turns off the lights and the user turns it back on, we can use that interaction to train our system further. In this case we can see it as the user punishing the system for making a mistake. The system will then adjust its decision scheme. This way the system will actively initiate a learning sequence. The system will remain in this stage indefinitely, and will continue to train it self using both passive and active learning.

By using incremental development we are able to design and implement the system one stage at a time, and evaluate the passive part of the system before designing the active part.

In this chapter we will discuss the different stages of the system, the problems that are present in each stage, and the solutions designed to solve these problems.

In the section "Theory" we will present the mathematical and statistical theory, that forms the basis for our algorithms. This section will also provide a brief rudimentary introduction to the concept of machine learning.

In the section Configuration, we will briefly discuss how the system is designed to be flexible, and allow us to quickly manipulate the different variables, that impact how the data is analyzed.

The process of collecting data is very simple, and will not be discussed in this chapter. In the chapter "Implementation" this process will be described in detail.

The section “The passive learning stage” consists of three subsections. In the sections “Event pattern” and “Decision table” we will discuss how the system analyses the passively collected data. As discussed in the chapter “Analysis”, using motion sensors can reduce the precision, and reliability of the collected data. In the subsection “Zones” we will discuss our approach to solve these problems. We will also provide a brief evaluation of the system in this stage, which will form the basis for the design of the active learning stage.

In the section “The active learning stage” we will discuss the additional processes that are present in this stage. These processes are made in response to the problems we have identified in the evaluation of the passive learning stage. Some of the problems we will address in this section has not been discussed in the analysis, since they have arisen, in the evaluation stage of the first development cycle.

3.1 Theory

“Stand back! I’m going to try science!” -Randal Munroe

In the core of our system lies a series of machine learning algorithms. In this section we will explain some of the basic concepts of machine learning, along with the statistical theory that our system is based on.

3.1.1 Machine learning

The purpose of machine learning is to have the system evolve behaviors based on empirical data, rather than programming a specific behavioral pattern. By using the supplied data as examples of relationships between data events, the system can recognize complex patterns, and make intelligent decisions based on the data analyzed (4).

With **supervised learning**[ˆsupervised-learning] the system is give labeled data consisting of examples of correct behavior. This is opposed to unsupervised learning, where the input data in unstructured, and un labeled. Because of both the human factor, and the imperfection of the motion sensors, the system will generate a certain amount of invalid data called noise. The algorithm will have to distinguish between what is proper training examples and what is noise.

Active learning is a form of supervised learning where the learner (the computer) prompts the user for information. In this form of learning the system initiates the interaction with the user, and trains it self based on the users response. This is especially useful if the system is generally well trained, but lacks training in specific areas. The system can focus on improving its training, in areas where its weak.

3.1.2 Markov chains

A Markov chain is a mathematical system that under goes transitions from one stage to an other (11). In a Markov system each step taken in a Markov chain is represented by a certain probability, based on the current state that the system is in. Formally:

$$P(X_{n+1}|X_n)$$

Here X_{n+1} represents the next state, and X_n represents the current state. And the entire notion is defined as the probability of the event X_{n+1} occurring, given that event X_n has just occurred.

By arranging these values in a matrix you can create a lookup table for future reference.

	X_1	X_2	X_3	X_4	X_5
X_1	P($X_1\|X_1$)	P($X_1\|X_2$)	P($X_1\|X_3$)	P($X_1\|X_4$)	P($X_1\|X_5$)
X_2	P($X_2\|X_1$)	P($X_2\|X_2$)	P($X_2\|X_3$)	P($X_2\|X_4$)	P($X_2\|X_5$)
X_3	P($X_3\|X_1$)	P($X_3\|X_2$)	P($X_3\|X_3$)	P($X_3\|X_4$)	P($X_3\|X_5$)
X_4	P($X_4\|X_1$)	P($X_4\|X_2$)	P($X_4\|X_3$)	P($X_4\|X_4$)	P($X_4\|X_5$)
X_5	P($X_5\|X_1$)	P($X_5\|X_2$)	P($X_5\|X_3$)	P($X_5\|X_4$)	P($X_5\|X_5$)

Each cell in the table represents the probability of entering the state represented by the cells row, assuming the system is currently in the state represented by the cells column.

3.1.3 Markov chains with memory

One of the most iconic features of Markov chains is the fact that they are memoryless. The probability of entering a new state is only based on the current state of the system. The states prior to the current have no effect on this probability. With "Markov chains of order m", or "Markov chains with memory", the system has memory of the last m steps in the chain, and these affect the probability of entering future states. This probability can be written as:

$$P(X_{n+1}|X_n, X_{n-1}, ..., X_{n-m})$$

Now the probabilities are calculated based on the pattern of steps made through the system rather than just the current state.

Since our probabilities are calculated based on collected data, we will not have to perform any complex statistical calculations, but will simply estimate the probability values based on the collected data.

3.2 The passive learning stage

In the passive learning stage the system monitors the user and trains it self based on his actions. In this stage the system does not interact actively with the house, but merely collects data, in order to develop a decision scheme.

3.2.1 Event patterns

As discussed in the "Analysis" Chapter, we want to base our decision making on more than just where the user is right now. We want to be able to look at where the user is coming from, and try to predict where the light needs to be turned on or off. In order to do this we look at a series of sensor events leading up to a switch even. Thereby we are incorporating the users movement pattern in to the decision making. We define this series of events, as an event pattern. An event pattern can consist of a number of sensor event, and may end in a singe switch event. An event pattern consisting only of sensor events is also defined as a "sensor pattern".Since we are interested in the users behavior leading up to a switch event, an event pattern can only contain one switch event, and this will always be the last event in the pattern.

The event pattern is an implementation of a Markov chain. Each event in the pattern is a separate step in the chain, and the probability to performing the next step in a chain, can be calculated using the theory of Markov chains of order m, where m denotes the length of the pattern.

Events are grouped in a pattern based of the time the event was triggered. If a series of sensor events, are less than some time interval apart, we consider them to be part of an event pattern. We define this time interval as the “pattern interval”. The pattern interval needs to be long enough, that a user moving around normally is seen as a continuous event pattern, and not broken into fragments. The time interval also needs to be short enough, that different user action, is seen as separate event patterns. For instance, a user going the kitchen to get a snack, and then returns to the living room, should ideally be seen as two separate event patterns.

With the idea of an event pattern, we can look at what patterns lead up to a switch event. And by extension of that analysis, when we observe an event pattern, we can determine the probability that it would lead to a switch event.

3.2.2 Configuration

We have designed the system in a way that allows us to rapidly change the various variables that affect the calculation of our decision scheme. This is done by using a config file, that sets the list of variable, and can quickly be modified between each execution of the software. The variables we wish to manipulate includes the length of an event pattern, and the maximum interval between two events in the same pattern.

3.2.3 Decision Table

In the core of the intelligent system lies the decision table. This is the product of the machine learning algorithm. The decision table is designed to be an efficient lookup table that the system can use as part of its decision scheme.

The algorithm for training the system in this stage is based on the concepts of passive supervised learning, since the user generates concrete examples for the system to follow. The data are labeled by type of event (sensor, switch), and the switch events are further divided into “on” and “off” events. These labels help the system determine how to analyze each pattern of events.

The decision table is designed as a Markov matrix, but we need the system to be able to handle Markov chains with memory, since we are tracking patterns, instead of single events. This effects the design of the Markov matrix.

Lets start by looking at the simple system with a pattern of length 1. Here we can simply use the Markov matrix described in the theory section.

switches extbackslash sensors	sensor 1	sensor 2	sensor 3
switch 1	$P(switch1\|sensor1)$	$P(switch1\|sensor2)$	$P(switch1\|sensor3)$
switch 2	$P(switch2\|sensor1)$	$P(switch2\|sensor2)$	$P(switch2\|sensor3)$
switch 3	$P(switch3\|sensor1)$	$P(switch3\|sensor2)$	$P(switch3\|sensor3)$

For each set of sensor and switch events, the table above holds the probability of the switch event occurring, given that the sensor event has just occurred. This table acts as a relation table between the sensors and switches, in a system based on traditional Markov chains. In our system this is the result of the decision table, if the pattern length is set to 1.

When we expand the Markov matrix to handle chains with memory, the matrix becomes more complicated. In the table above, the number of cells is given by the number of sensors in the system multiplied by the number of switches in the system:

$$\#switches \cdot \#sensors$$

When we add a sensor event to the eventlist the number of cells in the matrix is multiplied by the number of switches again. This results in the general formula:

$$\#switches \cdot \#sensors^{patternlength}$$

As a result of this we see that for each event we add to the event pattern the matrix must be expanded by a new dimension. Thus a pattern length of n results in an n-dimensional matrix.

The optimal pattern length will be determined based on experimentation and evaluation of the implemented system, therefore we must develop a system design that is flexible enough so that we can change the pattern length. This means that the decision table must be of n dimensions.

![Illustration of the n dimensional matrix required to contain the Markov table. Each colum in the left side represents a new dimension][n-dimension] [n-dimension]: figures/n-dimension.png

One advantage is that, since we are only interested in the users behavior related to his interaction with the wall switches, we only need to handle the patterns where the last event is a switch event. We must now go though our database, and for each switch event we must extract an event pattern consisting of that event, and the n–1 sensor events preceding it. The decision matrix will consist of the number of times a pattern has occurred in the collected data. This value is then divided by the number of occurrences of the event pattern without the final switch event.

This value can also be interpreted as an estimate of the probability of the final switch event of the pattern occurring, given that the preceding sensor pattern has been observed.

The system must also be able to handle patterns that are shorter than the maximum length, in case the pattern leading up to a switch event is smaller than the maximum pattern length. This could for example occur if the interval between two events have been too long.

The algorithm that handles the table generation looks as follows:

```
GenerateDecisionTable(events[]);
lastevent = 0
map decision_table
map denominator
queue eventpattern

for event in events
do
    if event is sensorevent
    do
        if event.time <= lastevent + pattern_interval
        do
            push event to eventpattern
            if eventpattern.length > pattern_length
                remove tail from eventpattern
        else
            clear eventpattern
            push event to eventpattern
        done
        insert event into denominator
        lastevent = event.time
    else if event is switchevent
    do
        if event.time <= lastevent + pattern_interval
        do
            insert event into decision_table
        else
            clear eventpattern
            add event to eventpattern
```

```
        done
    done
done

for entry in decision_table
do
    extract eventpattern
    divide by matching denominator
done
```

First the algorithm creates two maps: decision_table and denominator. The decison_table will, as the name suggests, hold the decision table. The denominator maps is used to keep track of the number of times each pattern of sensor events occur. This is used as the denominator in the fraction for calculating the probability in the decision table. The event pattern always contains the last n events in the system, unless the time between events exceeds the value stored in pattern interval. The algorithm now runs through the collected data in chronological order.

If the current event is a sensor event, this is added to the event pattern, assuming that the time since the last event has occurred has not exceeded the pattern interval. The event pattern is now used to navigate through the n dimensional matrix denominator, and increase the occurrence of the pattern by 1.

If the current event is a switch event, this is added to decision table in the same fashion as with the denominator matrix. Since we are not interested in patterns that contains more than one switch even, the event pattern is now emptied.

Finally each value in the decision table is divided by the corresponding value in the denominator tables. This is done by extracting the event pattern from the decision table and using it to navigate the denominator matrix.

The entire algorithm is run both for "on" and "off" switch event. This results in two separate tables, one for turning the lights on, and on for turning them off.

3.2.4 Zones

On of the problems that arose when evaluating the first implementation of the system, was that the motion sensor were not always able to deliver reliable event patterns. The main issue is that the sensors will aways have a certain amount of overlap on the areas they cover. If the user enters an area covered by more than one sensor, it is impossible to predict in which order the sensors will trigger. For an over lap between two sensors this effectively means twice the amount of sensor patterns for the same movement. As a result the system will require twice the amount of data to train these patterns. When the overlap occurs between more sensors the number of patterns increase by a factor of $n!$.

In order avoid this undermining of the collected data, we have designed a solution called sensor zones. When using sensor zones we define each area in the house as a zone that can be covered by one or more sensors. If a zone is only covered by a single sensor the system will handle it the same way it handles sensors. If a zone is covered by more than one sensor the system creates a virtual sensor, that acts as the combination of these two sensors. For a series of sensor events to be classified as a zone event, they will have to trigger with a very short time interval between them. The effect of this system is that if multiple sensors trigger within a very short time interval, the system will see them as a single event, no matter the order they triggered in.

Take (Figure 3.1) as an example, three sensors (1, 2 and 3) with overlap, and three paths the user could take (A, B and C). The paths B and C should only be observed as zone events by the system. Path A should be detected as the event pattern [1, zone 1 & 2, 2, zone 2 & 3, 3].

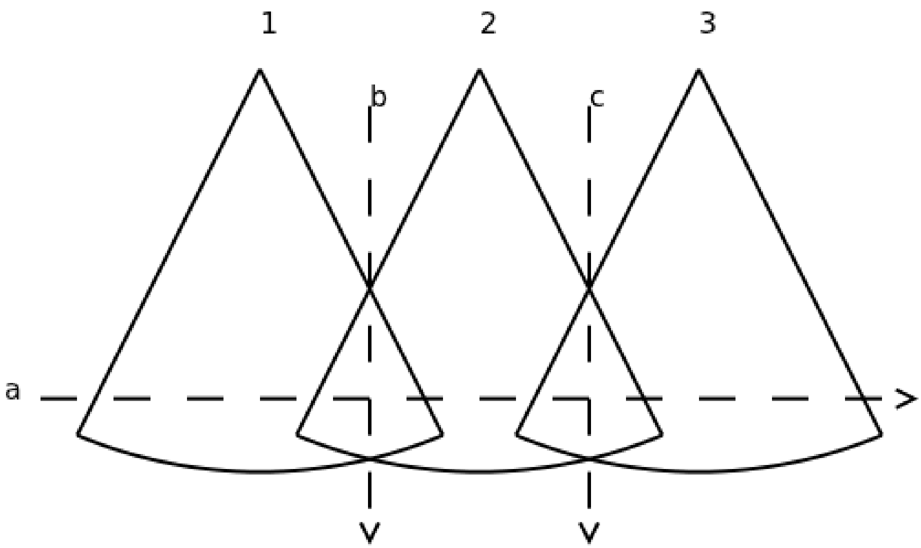


Figure 3.1: Sensors with overlapping zones

For path c without zone events, it's uncertain if sensor 2 or 3 would detect the user first, and these would be considered distinct event patterns by the system.

With zone detection, the pattern will look the same to the system no matter which sensor fired first, and as a result the system would be able to learn the intended behavior for path c faster.

Zones allow the system to determine the user's position more precisely, and to learn faster by removing ambiguity in some cases.

An other benefit of having sensor zones, is that it effectively increases the amount of sensors in the house. This results in a much better precision in tracking the user through the house.

Sensor zones is designed to be a supplement to the basic system. This means that the actual event pattern of events generated by the motion sensors, will always be trained. When the system is running it will check against both the raw sensor pattern, and the pattern with zones activated. This way the incorporation of sensor zones will not corrupt the collected data.

3.2.5 Evaluating the passive learning stage

After the first development cycle we were able to evaluate on the performance of the system. In the chapter "Evaluation" we will describe our findings in detail, so for now we will simply analyze the problems we were able to identify as a result of this evaluation.

We collected data over a two week period, and ran our data analysis on this collected data. This resulted in the decision table, which we then used as decision scheme while running the system in the simulator. The initial results were as expected somewhat hard to navigate, but by adjusting the system variables described in the config section, we were able to stabilize the system to a point where we could evaluate its performance. The realization we made was that the amount of data we had collected was far from enough. When the pattern length were set above 3, the number of times each pattern was observed was drastically decreased. We also realized that the system had a high probability of learning incorrect behavior. This problem will become negligible as the amount of data increases, but at the same time it is not practical to have a training period that is too long. Because of this we decided to implement active learning, as the next step for the system.

An other problem we encountered was that the system would not necessarily identify all of the event patterns that were supposed to result in the system turning off the light. Because of this we decided to implement a timer function, that can act as a backup, if the system does not recognize an "off" event pattern.

These two initiatives will be discussed in the next section.

3.3 The active learning stage

A key element of the system, is the transition from the passive learning stage to the active learning stage.

The system should start attempting to control the home, once it is confident enough, to act upon the decision schemes it has learned. But the system needs to have some quantifiable metric to determine its confidence, before it start to take over control of the home. There are two main metrics, we believe should determine when the system is confident enough:

1. The probability in the decision scheme must be above a certain threshold. $P(switch_i|pattern_j) > \varphi$
2. The specific $pattern_j$ must have occurred at least a certain number of times.

Exactly what the threshold should be, must be determined through experimentation, once the system is fully implemented in a physical environment. The second rule is to make sure, the system does not start acting based on patterns only observed a few times.

3.3.1 Switch Timeout

We want to create a system where, no matter what happens, the light is eventually turned off. We accomplish this by creating a timer for each switch that will turn it off after a set amount of time. This acts as a backup system, if the main system fails to recognize an off pattern. The idea of the timer is an extension of the timer used in the silvan system, described in the "Analysis" chapter. When a switch is turned on, a timer starts, that will eventually turn the switch off. We want the user to be able to extend the timeout period by activating motion sensors, in order to prevent the system from turning off the light, when he is still in the room. Since the system does not have any previous knowledge of which sensors and switches are in the same room, we need to establish these connections.

This is the purpose of the correlation table. With this table, the system attempts to identify links between sensors and switches. When a user turns a switch on, it we can safely assume that light is turned off where the user intends to be in the immediate future. So it is possible to get an idea of which sensors are near a switch, by looking at what sensor events occur shortly after a switch is turned on.

When flicking a switch off, the user may be leaving the room, or just have entered the room to turn the switch off. Each of the two cases are just as likely as the other, but the sensor events in the interval leaving up to the off event is completely opposite. Therefore this pattern is less suited for training the correlation system.

Based on the statistical data it is possible to generate a table, containing the probability that a specific sensor is triggered shortly after a switch is turned on. This gives us an idea of which sensors are in the same room as a switch. This is based on the same idea as the decision table, but we examine the sensor events that follows a switch event instead of those leading up to it. Also this table does not use sensor patterns, but only a single sensor event is correlated to a switch event.

$$P(sensor_i|switch_j, \Delta t) = \frac{\sum 1_{sensor_i}(switch_i, \Delta t)}{\sum switch_j\ events}$$

The identity function $1_{sensor_i}(switch_i, \Delta t)$ is 1 if the sensor is triggered within Δt after $switch_j$ is triggered, and i therefor not counted twice, if the sensor triggers multiple times after the same switch event.

So to reiterate $P(sensor_i|switch_j, \Delta t)$ is the probability that $sensor_i$) fires within Δt after $switch_j$ fires.

Table 3.1: Correlation table

	sensor 1 (se_1)	sensor 2 (se_1)	...	sensor n (se_n)
switch 1 (sw_1)	$P(se_1\|sw_1, \Delta t)$	$P(se_2\|sw_1, \Delta t)$	...	$P(se_n\|sw_1, \Delta t)$
switch 2 (sw_2)	$P(se_1\|sw_2, \Delta t)$	$P(se_2\|sw_2, \Delta t)$	...	$P(se_n\|sw_2, \Delta t)$
⋮	⋮	⋮	⋱	⋮
switch m (sw_m)	$P(se_1\|sw_m, \Delta t)$	$P(se_2\|sw_m, \Delta t)$	...	$P(se_n\|sw_m, \Delta t)$

Using this table we can then identify sensors and switches as being in the same room, if they are above a certain threshold.

So far the correlation table is till based on passive learning. By incorporating active learning methods, we can greatly increase the precision of the probabilities the the correlation table.

The active learning of the system is done by the system performing a switch action, and the user reacting to this action. There are two criteria that must be met for this interaction to occur.

First of all the action performed by the system must be incorrect, in order for the user to react to it. If the system does what the user wants the user will not interact with the switch, and the system will not receive feedback.

The second condition is that this will only work when the system turns the light off. If the system turns the light off at an incorrect time, it means that the user will be present in the room where the light is turned off. It is reasonable to assume the user will react by turning the lights back on, thus providing the system with the needed feedback. If the system turns on the lights at an incorrect time, it means that the user is not present in the room where the light is turned on. The system will there for not receive any feedback from this action, whether its correct or not. While it is possible to imagine situations, where the user will want the lights turned on in a room, where he is not present, and vice versa, the system is based on probabilities, and these situations will not occur often enough to have a noticeable affect on these values.

In the correlation table the active learning starts when the system turns off a switch based on a timeout event. If the user reacts by turning the lights back on, the system will use this as training data. In this scenario the system will train the correlation between the first sensor triggered after the system turned off the light, since this will be the sensor closest to the user location when the light was turned off.

The concept of timers was presented in the "Analysis" chapter when describing what we call the silvan system. As described then there are some problems that arise when choosing this solution.In order to address these problems, we want to be able to adjust the individual timers, based on which motion sensors are triggered. By using the correlation table, we can calculate timeout periods based on the correlation value between a switch and a motion sensor. When the user triggers a motion sensor, the system performs a lookup in the correlation table, to see if there are any switches that meets the criteria of being correlated to the sensor. If any are found, and if these are on, the timer of this switch is then set to a value calculated based on the sensor. The default value is set as

$$15 minutes * probability of sensor switch correlation$$

The system checks the correlation value between the sensor and switch, and multiplies by 15 minutes. The optimal value of this constant should be the result of experimentation on the finished system. We have arbitrarily chosen to use 15 minutes, until such experiments can be made.

By using a combination of passive and active learning to train the systems correlation table, we are able to create an enhanced timeout system, that calculates its timeout intervals based on input generated by the user. This ensures a system where we can minimize the timeout period, to reduces power consumption, and at the same time the system will automatically adjust this interval in areas where the user is likely to remain stationary for extended periods of time.

3.4 Controlling the house

Now that we have created a decision scheme based on the decision table, and correlation table, the final task is to control the house, based on this decision scheme. The amount of work put into the data analysis, greatly simplifies controlling the system.

The system constantly keeps track of the current event pattern. When a new event occurs it is added to the event pattern, assuming that the time since the last event in the pattern is not greater than the pattern interval.

Every time the system receives a sensor event,it will check the decision table to see if the current event patterns requires an action to be made. This is done iterating through all the switches in the system. One at a time the switches are added to the event list, and the system performs a lookup in the decision table. If the pattern exists in the table, and the value returned is above the probability threshold a switch action is made. These lookups are performed based on the current state of the switch the system is examining. If the switch is turned on, the lookup is performed in the “off” table, and if the switch is off, the lookup is performed in the “on” table. If the system receives a switch event, it resets the event pattern.

The system is set to re analyze its collected data on a daily basis. This is a scheduled event set to happen when the user would normally be a sleep. At this point the decision table, and correlation table is recalculated, taking in to account the data collected the previous day.

Chapter 4

Implementation

"If it compiles, it is good; if it boots up, it is perfect." – Linus Torvalds

In this section we will discuss transition from the software blueprint described in the "Design" chapter, to functional code. We will discuss the product in its current state, and will in this chapter not elaborate on the development process that lead to its current state, since this has been thoroughly documented in the previous chapters.

As stated in the introduction, the purpose of this thesis is to research the possibilities of incorporating machine learning in smart environments, and is designed to be a proof of concept study. Because the focus of the thesis is not on developing a fully functional home control system, this chapter will not include every aspect of the implementation process but will instead highlight some of the major choices we were facing, when implementing the system.

First we will describe the physical system we used to collect real life data. This includes a brief introduction to the hardware, as well as the database setup we used for storing the collected data.

We will then discuss the overall structure of the software system, and describe how each subject discussed in the "Design" chapter are represented in the code.

After presenting the general structure of the system, we will introduce the simulator we used to test our system, and describe the integration between the two.

Finally we will discuss the implementation of each of the elements discussed in the "Design" Chapter.

4.1 The physical setup

Since we needed real life data to train the system, the first task of the project was to create a physical setup to start collecting data. We installed wireless switches and PIR sensors[1] in David's appartment (Figure 5.2). The placebo switches were placed next to the normal switches controlling the light for each room, in all cases being the switch closest to the entrance. We installed a total of 10 motion sensors and 5 switches throughout the apartment, that collected data non stop for a period of two weeks.

The sensor setup consisted of three sensors in the living room, two sensors in the hallway, kitchen and bedroom, and one in the bathroom. When placing the sensors, we tried to provide as close to full coverage as possible, with special emphasis on making sure all the doorways were covered.

The wireless nodes we have available communicate using the Zensys Z-Wave protocol. This protocol was chosen because we already prior to this project had designed and implemented a z-wave API[^api] in java. This greatly reduced the time and effort needed to setup and implement the data collection system.

[1] Passive infrared sensors. [^api]: Application programmers interface

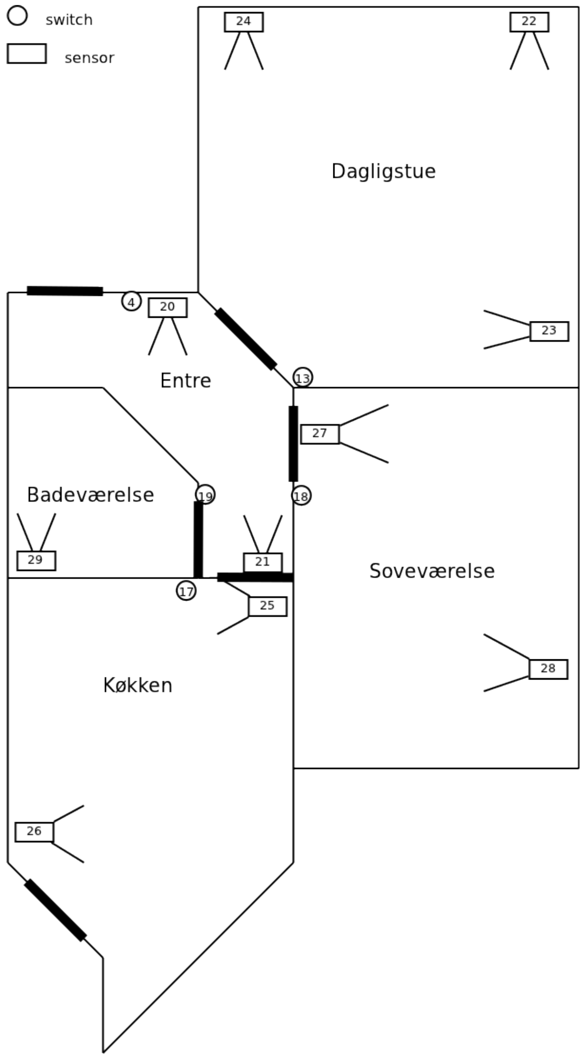
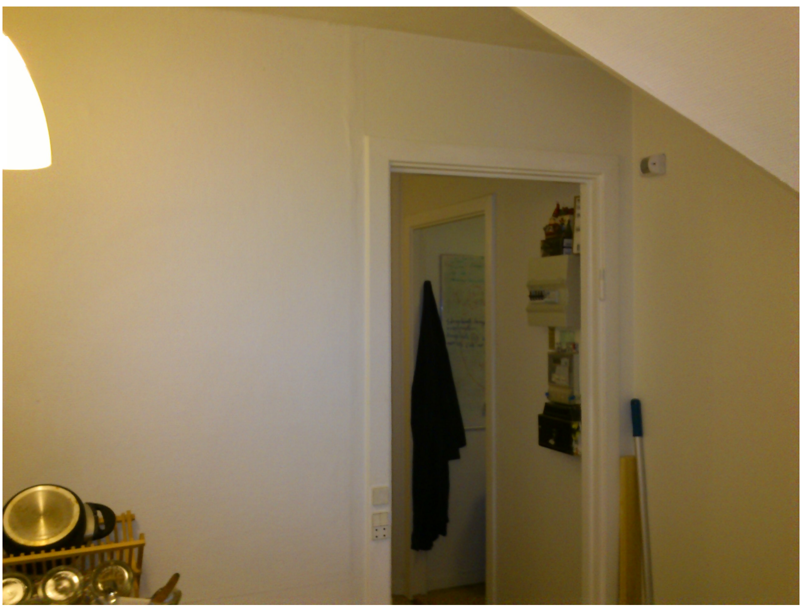


Figure 4.1: Map of the testing environment with sensor and switch locations



We setup a mini PC with a Z-Wave serial device, and configured all PIR sensor and switches to send notifications to the PC, when they where triggered. The PC ran a Z-Wave API, which we added a listener to, so that sensor and switch event was logged to an SQL database.

We kept the database vary simple, and only logged the type of event, along with the time the event occurred. Below is a representation of the database setup.

Table 4.1: Database table for sensor events

sensor_events	
id	Integer
timestamp	Timestamp

Table 4.2: Database table for switch events

switch_events	
id	Integer
timestamp	Timestamp
status	Boolean

4.2 General system structure

The system is divided into 5 packages.

- The “smarthouse” contains SmartHouse.java which is the class in charge of controlling the house. This is the central class of the system.
- The “config” contains Config.java which loads the system configurations from a "“.settings” file.
- The “core” package contains DecisionMatrix.java which is the class in charge of generating the decision tables “on” and “off”. This package also contains the class Correlation.java which generates the correlation table.
- The “event” package contains the classes representing the various types of events in the system, along with the class EventList.java, which is the implementation of the event pattern.
- The “timer” package contains the classes dedicated to handling timeout events, and running timers for the individual switches.

We have divided the class diagram in to three separate diagrams for simplicity:

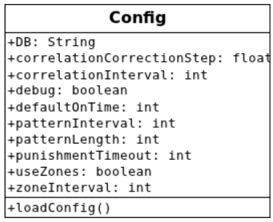


Figure 4.2: The class diagram of the config class

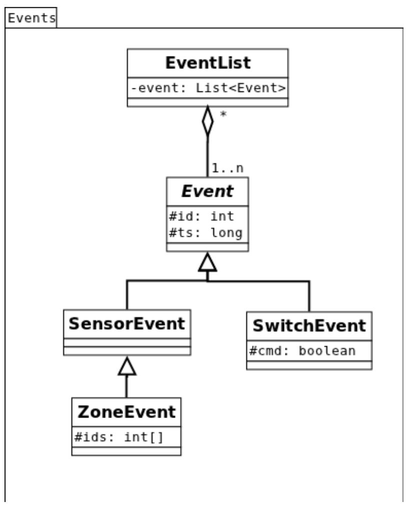


Figure 4.3: Class diagram for the events package

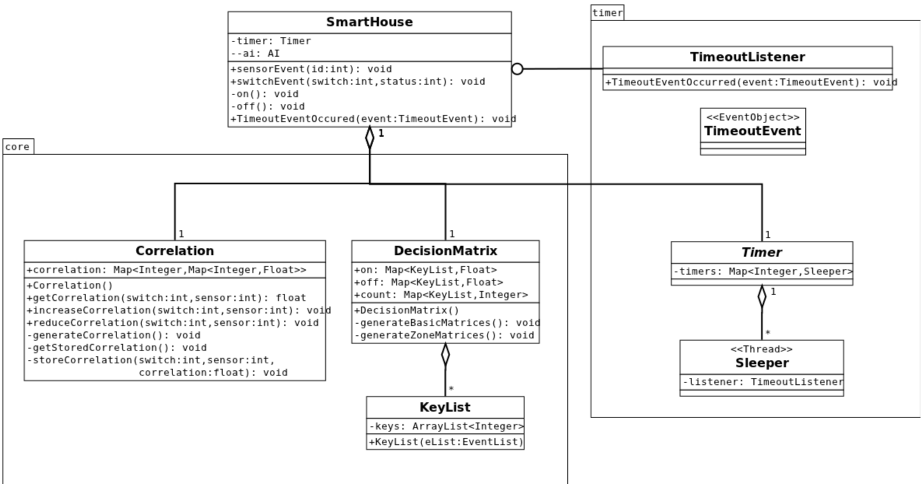


Figure 4.4: Class diagram for the smarthouse, core and timer packages

4.3 Simulator /AI interface

In order to effectively evaluate the system we use a smart house simulator, which was developed by a team of DTU students as part of a (12, bachelor thesis). We extended the simulator with an AI module, implementing the features discussed in this report. The simulator is implemented in scala, but we chose to implement the AI module in Java. Since both languages compiles to Java byte code Scala and Java interfaces very seamlessly, and Scala code can directly invoke Java methods and vice versa. We chose to implement the AI in Java, to work in a language we're well-versed in, to increase our productivity and quality of the code.

With the simulator we are able the test, and evaluate, the system in the different stages of development. The system has all the data gathered from the passive learning stage, and we are able to see how the system would behave in the beginning of the active learning stage. As stated in the analysis, simulated user data will never be as good as actual user data, and we have therefore not trained the system based on data generated in the simulator. The advantage of the simulator is that we can see if the system is acting and reacting as expected in the active learning stage.

We can also compare the output of the simulator to the probabilities calculated in the decision table and correlation table.

4.4 Configuration

The Config.java class in created as a simple static class, that uses a file reader to load a config file stored on the hard drive. The config class initially holds the default values for the system, which are overwritten with the values from the config file. If no config file is present on the system, the config class generates a file based on the default values. After loading the config file, the other classes in the system, can then access the static fields of the class. These values remain constant after initially loading the config file.

A typical config file could look like this:

```
#automatically generated preferences file
#delete to return to default settings
pattern_interval 3000
pattern_length 2
probability_threshold 0.01
use_zones true
zone_interval 500
correlation_interval 7000
```

4.5 Event patterns

To make lookups based on the observed event pattern, each new sensor event is matched to see if it is part of a pattern. As each sensor and switch event is received by the system, a list of the most recent event pattern is maintained in an EventList. EventList is basically a queue of sensor events. It is implemented as a FIFO list with a max length matching the pattern length property. If the list is at max capacity when a new event is added, the first item in the list is subsequently dequeued. The pattern interval rule is maintained by looking that last event in the queue, when a new event is added. If the last event is more than pattern interval old compared to the new event, the queue is cleared before the new event is queued.

```
EventList add(event):
queue events
if events.tail.time + pattern_interval < event.time
do
    events.clear
```

```
fi
events.add(event)
```

If zone detection is enabled, EventList first checks if the difference in the timestamp between the last event and current event is smaller than the zone interval. If a zone is detected, the last event in the list is replaced with a zone event.

The EventList is used to make lookups in the decision matrix, which takes a fixed length array of sensor IDs as key. When looking up patterns shorter than the configured pattern length, the pattern is prefixed with the id *−1*, to maintain the fixed length.

4.5.1 Zone events

Zone events are represented as en extension of sensor events, with a list off all the sensors that are part of the zone event. In order to look up zone events in the decision matrix, each zone also has a single integer id representation. The id is calculated from the sorted list sensor.

```
getID()
sum = 0
for sensor in zone
    sum = sum*256 + sensor.id
return sum
```

For zone events based on at most 4 sensors, with id values less than 256, this function generates unique, comparable ids.

4.6 Decision Matrix and KeyList

The Decision Matrix is the class that holds the decision table. The class consists of the two matrices "on" and "off" which together forms the decision table. Instead of implementing the matrices as multidimensional arrays, we have chosen to use hash-maps were the key is an array with a size equal to the pattern length. There are two main advantages to using hash maps instead of multidimensional arrays. The first advantage of this is that the lookup time is much faster in a hash map, than an n-dimensional array. This is especially true when the amount of data in the system increases, and when increasing the number of dimension, i.e. increasing the pattern length. Secondly the multidimensional array would be much larger, since it would have to allocate space for every possible pattern instead of just the ones extrapolated from the collected data.

Using an array as a key for the hash map does however create a few problems. The main issue is that the hash function for arrays is inherited from the object class. This means that two arrays containing the same elements will produce different hash codes. In order for our map to function properly, identical arrays must produce identical hash codes. The same problem occurs when comparing arrays using the equals() method.

We addressed this problem by implementing a KeyList class with a custom designed hashCode() and equals() method. The equals method was done by individually comparing each element in the list, and returning true, if the pairwise comparisons all succeeded. The hasCode() method is based on the hashCode method used in the String object in java. The method iterated through each element in the list, and for each value the sum of the previous values are multiplied by 31, and the current value is added. This ensures a very low collision rate with the amount of sensor and switches that are likely to be used in a private home.

Besides the increased speed when performing lookup operations, the main advantage of using Hash maps is that it greatly simplifies extracting the keylist from a specific value. This is necessary when we divide the values in the decision maps “on” and“off” with the values in the denominator map. This is done by iterating through the decision maps, and for each value we extract the key, remove the last element, the switch event, and converts the resulting EventList into a KeyList to be used in the denominator map. When using Hash Maps this process is simply done using the keySet() method. If instead we had used multidimensional arrays, we would have to iterate through all possible key combinations in an array of n dimensions.

The Hash maps are generated in the method generateBasicMatrices(). This function first sends a query to the database returning all existing events. As the system scales, this will be have to be changed since collecting all the data using a single query could be a problem especially on a system with limited memory. During the course of the project the size of the database never exceeded 1.3 MB, so it will require a substantial amount of data to cause problems for an average laptop.

Once the data is returned from the database the system iterates through the resultset, and inserts the data into the hash maps as described in the design chapter. Finally the values in the maps “on” and “off” is divided by the corresponding values in the denominator map.

If the use_zones option is enabled in the config file, the Decision matrix will repeat the process above using an EventList with zones enabled. This is done in the method generateZoneMatrices(). This time how ever any pattern not containing a zone event will not be added to the decision maps. This method uses temporary decision maps called “zoneOn” and “zoneOff”. After the probability values in these maps have been properly calculated, the content of these maps are appended to the original decision maps “on” and “off”.

4.7 Correlation table

The correlation table is based on both statistical data from the passive learning stage, as well as corrections and punishments from the active learning stage. First the statistical correlation is calculated, and then the corrections are added on top of that.

4.7.1 Correlation statistical generation

Correlation is the system's estimate of the probability of a switch and sensor being in the same room. The system looks at the time interval after a switch is triggered. The sensors triggered in this interval, are defined as having a correlation to the switch. Each sensor is counted only once per switch event.

The correlation is the probability that $sensor_i$ is triggered at most Δt after $switch_j$ was turned on.

4.7.2 Correlation correction

The system is able to adjust correlations, based on active learning. When a switch is turned on, a timer is started for that switch. If a correlated sensor is triggered, the timeout is extended. The duration is determined by the correlation between the sensor and the switch, higher correlation gives longer timeouts.

If the switch is turned off before the timeout is reached, the timer is stopped and nothing further happens. If the timer runs out a timeout event is triggered, and the light is turned off. A new timer is started when a switch is autonomously turned off, to verify that no manual overrides occur. If a manual override occurs (e.g. the user turns the switch on again, while the timer is running), the system is "punished". The system increases the timeout time, by increasing the correlation between the switch and the first sensor triggered after the switch was turned off. If no manual override occurs, the system was correct in turning off the light, and lowers the timeout time, by reducing the correlation between switch and the last sensor triggered, before the switch was turned off.

These correlation corrections are stored in a database. The correlations used for the timeout is based on both the statistical correlation, and the correlation corrections. The correlation for each switch-sensor pair is the statistical probability plus any correlation corrections.

The correlation corrections increase or reduce the correlation by 10 percent points, each time is punished or correct. The system doesn't have a limit to correlation corrections, so correlations can be higher than 100%. This gives the system the ability to get timeouts longer than the default timeout.

Table 4.3: Database table for correlation corrections

correlation_confirmation	
switch	Integer
sensor	Integer
correlation	Float

4.8 Timers and timeout

Timers are implemented in the Timer and Sleeper class. Sleeper is a fairly simple class. It starts a new thread, sleeps for a given time, then fires a timeout event to a given timeout listener. Timer simply holds a map, where each switch can set a timeout. Timer creates a sleeper object, and puts in the map. The sleepers can then easily be monitored and interrupted if needed.

To received the timeout events the SmartHouse class implements TimeoutListener.

CHAPTER 5

Evaluation

If you torture data long enough, it will tell you what you want -Ronald Coase

In this chapter we will evaluate the system that we have developed, and the data the system has been able to produce. As defined in the introduction the purpose of the thesis is to investigate the possibilities of incorporating machine learning in home control systems. In this chapter we will discuss the results of this investigation, and and examine the results we have produced during the project. Finally we will discuss what conclusion that can be made based on these results.

Before any evaluation is made, we must however first describe our strategy for testing our software. Since this is what allows us to argue for the correctness of the produced results.

The main focus in our software testing strategy have been to ensure that the parts of the software that is responsible for producing that data we evaluate on, are functioning as intended. For this purpose we have used a combination of unit testing, and integration testing.

5.1 Software testing

As mentioned above the software testing has been divided into two separate types of tests.

5.1.1 Unit testing

We have used JUnit tests to test the implementation of the relative simple classes Event, EventList, and KeyList. The testing files have been included in the appendix. All these files produced the expected output.

5.1.2 Integration testing

The more complex classes DecisionMatrix and Correlation are tested using integration testing, since they are very tightly coupled to EventList and KeyList. The integration testing, is based on simulated data, instead of the collected data, in order to have verifiable outputs. The simulated setup consists of 6 sensors (1,2,3,7,8,9) and 3 switches (4,5,6). A simulated user takes various paths to generate a representative sample of event patterns. These test are made using the smart house simulator.

Table 5.1: Event patterns used for black box testing

Test case	Description	Event sequence
1	Path A, switch 4 on, sensor 9	[1, 1&2, 2&3, 3, 4 on, 9]
2	Path B then turns switch 5 off	[1&2, 5 off]
3	Path B without using any switches	[1&2]
4	Path C, switch 6 on, sensor 7	[2&3, 6 on, 7]
5	Path C without using any switches	[2&3]
6	Path C, switch 6 on, sensor 8	[2&3, 6 on, 8]

Based on these simple event patterns, an expected output can be determined for both the DecisionMatrix and Correlation. The expected output for the DecisionMatrix is based on the number of times each event pattern has been seen, and the number of times they have led to a switch event.

Testing of the DecisionMatrix revealed what at first looked like an error. The probability for Path C without zones had a probability of 100%, but with zones had the expected probability of 67%. Investigation revealed the cause was test

Table 5.2: DecisionMatrix's expected output

Description	Sensor pattern	Switch	State	Probability
	without zone events			
Path A	[1, 1, 2, 2, 3]	4	on	1
Path B	[1, 2]	5	off	0.5
Path C	[2, 3]	6	on	0.67
	with zone events			
Path A	[1, 1&2, 2&3]	4	on	1
Path B	[1, 2]	5	off	0.5
Path C	[2, 3]	6	on	0.67

case 5, where sensor 2 and 3 was triggered in the opposite order as test case 4 and 6. So while this error at first glance looked like a bug, is actually a feature, and one of the very reasons zone events were implemented. All other probabilities in the DecisionMatrix was as expected.

For the correlation table, the output is determined only by test cases where switches are turned on (test case 1, 4 and 6). The expected output is seen in the table below.

Table 5.3: Correlation table's expected output

switches	sensors		
	7	8	9
4	0	0	1
6	0.5	0.5	0

The correlation table produced the expected results.

5.2 Evaluation based on passive learning data

In this section we are going to evaluate how much the system have been able to learn, based on the data collected from the passive learning stage. In total 45.628 sensor events and 346 switch events was recorded. This is a very high sensor event to switch event ration, slightly above 130 sensor events per switch event.

Of the 346 switch events, 194 was ON events and 152 was OFF events. If all switch event in a continuous period was recorded, the discrepancy between ON

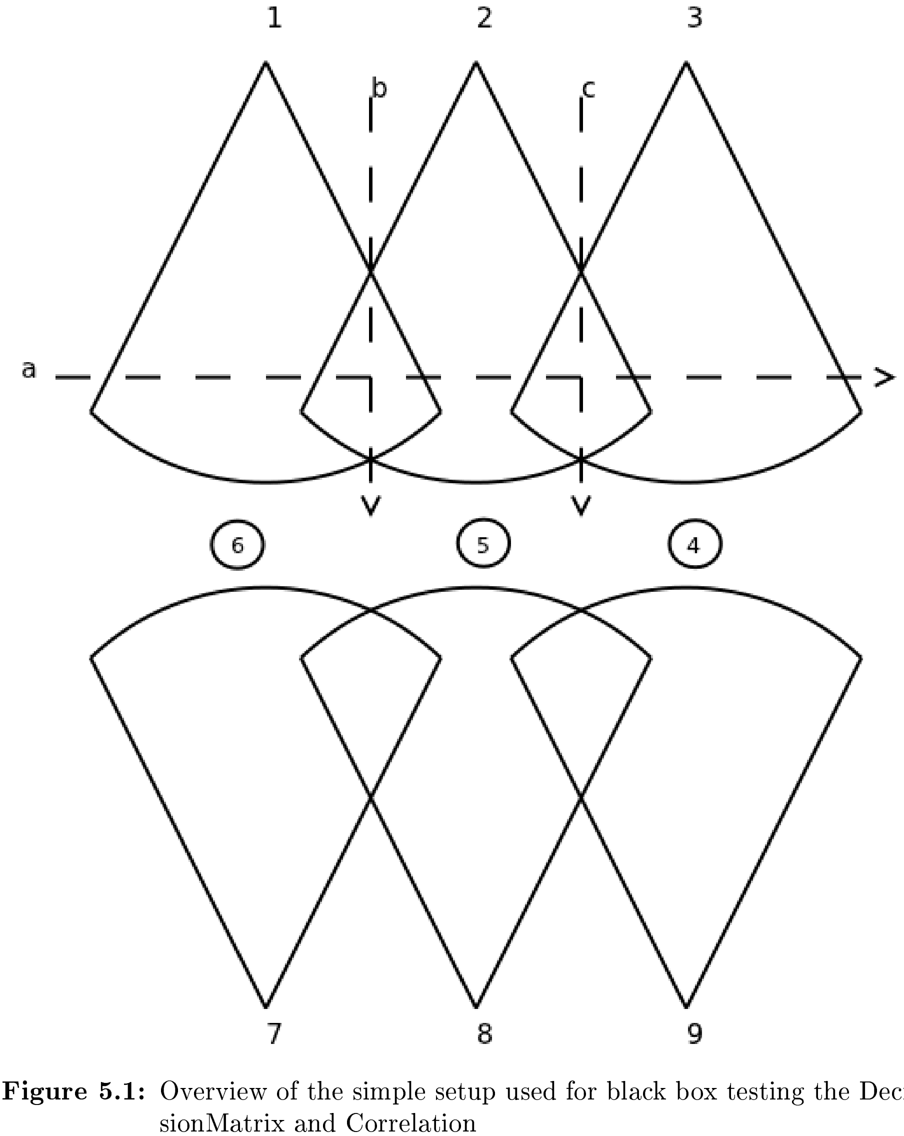


Figure 5.1: Overview of the simple setup used for black box testing the DecisionMatrix and Correlation

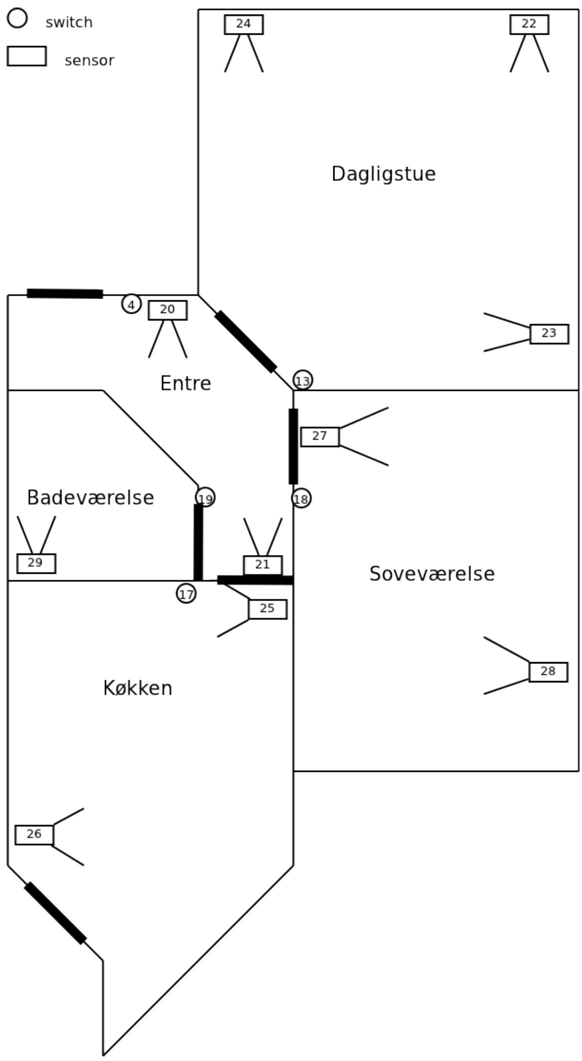


Figure 5.2: Map of the testing environment with sensor and switch locations

and OFF events would be at most the number of actual switches. This could be due to lost Z-Wave messages or users forgetting to press the placebo switches. The system is not dependent on the correct ordering of switch events, i.e. that ON events are eventually always followed by an OFF event, and vice versa.

The discrepancy between ON and OFF events, are an indicator that data have been lost, the system should still be able to learn based on the user data. Assuming only switch events are lost, this will impact the system by having an increased sensor to switch event ratio, thus lowering the estimated probabilities in the decision matrix.

The Correlation table is not based on the entire data set of sensor events, but merely the interval after each On event. Therefor the sensor to switch event ratio for the Correlation table, is not widely affected by missing switch events.

5.2.1 Decision matrix

In order to better evaluate the Decision Matrix, it has been run on the training data several times, with different pattern lengths, with and without zone detection. The evaluation will look upon the advantages and disadvantages of the different configurations, and evaluate on how much the system is able to learn from the collected data.

Table 5.4: Statistics about the Decision Matrix, using different configurations

Settings		Unique observed patterns		
Pattern length	Zones enabled	Movement patterns	On patterns	Off patterns
2	No	111	90	78
2	Yes	1.168	149	121
3	No	910	142	116
3	Yes	3.870	227	173
4	No	3.614	169	121
7	Yes	12.967	322	215

With zones enabled, the system looks at the event patterns leading up to each switch event, with and without zone detection. Detecting up to two switch patterns for every switch event, in some configurations there are more total switch patterns detected than actual switch events. A complete dump of all patterns detected by the Decision Matrix for each configuration is included in the appendix.

With a 130 to 1 sensor to switch event ratio, the probabilities for each event pattern leading to a switch event is very low. This is not necessarily a problem, it may just mean the probability threshold, for the system, needs to be equally low.

A lot of the ON and OFF patterns detected by the Decision Matrix have only been observed once. We're going to set the confidence threshold so that a pattern must have lead to an On or Off event at least 5 times, and then analyze the correctness of the patterns observed

With the expectancy that the probabilities are going to be relatively low, for each switch pattern, the evaluation of Decision Matrix will look at plausibility of the patterns detected, more than how high or low the probability should be. Does the detected patterns make sense from a user point of view? The expected result is to detect plausible user patterns, when users press switches as they're entering or leaving each room. The reverse of that expectancy is the system shouldn't detect implausible patterns, where motion events lead to switch events in non-adjacent rooms.

Table 5.5: Decision matrix, patterns detected at least 5 times, pattern length 2, without zone detection

Pattern	Probability	Description
20 21 13 on	0.57%	Moving in the hallway, and turning on the light in the Living room
27 28 18 on	0.75%	Moving in the bedroom, and turning on the light
20 20 19 on	2.38%	Moving in the hallway and turning on the light in the restroom
20 21 19 on	2.17%	
21 20 19 on	1.70%	
21 25 17 on	3.26%	Moving from the hallway into the kitchen and turning on the light
20 25 17 on	5.76%	
20 20 19 off	1.49%	Moving in the hallway turning off the light in the restroom
21 20 19 off	1.2%	
20 21 19 off	1.14%	

With pattern length two, most of the patterns above the confidence limit, only contain sensor event from a single room (from here on referred to as single room patterns). In some cases there are identical patterns for turning a switch on and off; [20, 20],[20,21],[21,20] all both turn the switch to the rest room on and off. This is partially because the switch for the restroom is outside the restroom, so the light is turned on before the user opens the door and is detected by the motion sensor inside. With the probabilities being as low as they are, the system cannot meaningfully determine if the light should be turned on or off. If the system were to act based on these conflicting patterns, it would mostly likely turn the lights on and off constantly, without there being need for it. Since the conflicting pattern are for an adjacent room, where the door is likely closed, the user wouldn't necessarily be aware of it.

There are two pattern where sensor events are from different room (from here on referred to as multi room patterns): [20, 25 -> 17 on] and [21, 25 -> 17 on]. These two patterns occur when the user moves from the hallway and into the kitchen, and then turns on the light in the kitchen. These two multi room patterns, not only sound reasonable, but also have the highest probabilities of all the patterns above the confidence limit.

With pattern length two, and zone detection enabled, no event patterns with zones (from here on referred to as zone patterns) are seen leading to switch events 5 times or more. So for pattern length two, adding zones detection doesn't give any patterns above the confidence limit, for our data set. While zone events can reduce the ambiguity and allow the system to learn faster, physical motion sensors tends to have a cooldown. Cooldown means it takes some time, after the sensor has detected motion, before it will detect motion again. A result of this is that zone events are less likely to be detected. Two sensors might overlap, but if time between the two sensors are triggered are longer than the zone detection interval. The cooldown will cause the two sensors to keep firing sensor events too far apart to be detected as zone events. This problem should be solved by choosing motion sensors with a more adjustable cooldown.

Table 5.6: Decision matrix, patterns detected at least 5 times, pattern length 3, without zone detection

Pattern	Probability	Description
27 27 28 18 on	1.86%	Moving in the bedroom, and turning on the light
20 21 20 19 on	2.35%	Moving in the hallway, and turning on the light in the restroom
21 20 21 19 on	2.03%	
29 21 20 19 off	10.2%	Moving from the restroom to the hallway, turning off the light in the restroom
21 20 21 19 off	2.36%	Moving in the hallway, turning off the light in the restroom

When the pattern length is increased to three, fewer distinct switch patterns above the confidence limit are detected. Just as when the pattern length was two, the majority of the patterns are single room patterns. There is one multi room pattern: [29, 21, 10 -> 19 off] where the user leaves the restroom, enters the hallway and turns off the light to the restroom. Like the other two multi room patterns, this pattern sounds reasonable, and have a relatively high probability of just over 10%.

Again adding zone detection doesn't produce any zone patterns above the confidence threshold.

Table 5.7: Decision matrix, patterns detected at least 3 times, pattern length 4, without zone detection

Pattern	Probability	Description
28 27 21 20 19 on	8.33%	Moving from the bedroom to the hallway, turning on the light in the restroom
29 29 21 20 19 off	11.11%	Moving from the restroom the the hallway, turning off the light in the restroom
–1 21 20 21 19 off	9.38%	Moving in the hallway, turning off the light in the restroom

When increasing the pattern length to 4, no patterns were above the confidence limit of 5, so these patterns have only been see 3 or more times. This matrix has an interesting multi room pattern [28, 27, 21, 20 -> 19 on], where the user moves from the bedroom to the hallway, and then turn on the light to the restroom. While a plausible pattern, it isn't a pattern that can be guaranteed to always happen. This is because the switch for the restroom is located outside the restroom, so users tend to activate the switch before being detected by the sensor on the other side of the door.

The multi room patterns detected by the system, all seem like plausible behavior, and these pattens have some of the highest probabilities, of the patterns seen at least 5 times. Although three confirmed multi room patterns aren't a lot. With more learning data, more patterns would be above the confidence limit. The data suggests, that the plausible patterns that system should learn to act on stand out with high probabilities. So with more data, more plausible patterns should appear which stand out by having high probabilities.

The Decision Matrix learned different patterns, when the pattern length was changed. With pattern length 2, the most distinct patterns above the confidence limit was detected. The number of confidently detected patterns decreased as the pattern length increased. This mean the system is able to learn faster with a lower pattern length. The patterns learned from pattern length 2 and 3 both had merit, so while a lower pattern length cause the system to learn faster, longer patterns enables the system to better take into account where the user is coming from. For instance the pattern [29, 21, 20, 19 off] where the system turns off the light in the restroom, when the user leaves is too long to be detected with a pattern length of 2.

5.2.2 Correlation

In this section we are going to evaluate how well correlation, based on the generated user data, matches to the actual setup. The system's ability to get accurate estimates of which sensors and switches are in the same room. We are also going to evaluate how well the correlation based timeout would work, with or without correlation corrections. Prior to looking at the actual data, we want to state some reasonable goals we want the system to achieve for the correlation probabilities:

1. A sensor should have the highest correlation to the switch in the room it is in.
2. Some correlation threshold should exist, so that sensors and switches in the same room are above the threshold, and those not in the same room are below the threshold.

Table 5.8: Correlation table, based on statistical data. > 40% in bold, 40–20% in italic.

Switches		Sensors									
		20	21	22	23	24	25	26	27	28	29
		Hallway		Living room			Kitchen		Bedroom		WC
4	Hallway	**0.4**	**0.67**	0	0.2	0.13	0.07	0	0	0.07	0
13	Living room	*0.35*	*0.23*	0.12	*0.27*	**0.42**	0.04	0.04	0.08	0.08	0
17	Kitchen	*0.22*	*0.28*	0	0.03	0.17	*0.39*	**0.58**	0.14	0.03	0.03
18	Bedroom	0.1	0.13	0	0	0.03	0.03	0	**0.57**	**0.6**	0.03
19	WC	*0.29*	*0.29*	0.06	0.09	0.08	0.06	0	0.07	0.03	**0.75**

The correlation table (Table 5.8) is based on collected data from the testing environment. The first criteria holds, that all sensors have the highest correlation with the switch in the room they are in.

The second criteria does not hold for all correlations. Most correlation probability for sensors and switches in the same room are above 40%. All correlations for switches and sensors not in the same room are below 40%. Although three sensors have correlations lower than 40% to the switch in the room they are in, and one of them as low as 12%. In the living room, two sensors not only have correlations below 40%, but correlations below those of sensors in the adjacent hallway.

As can be seen in the overview of the apartment (Figure 5.2), the sensors 22 and 25 are located in the far end of the rooms from the switch and doorway. The calculated correlations are based on the time interval after a switch is turned on, so it makes sense that sensors being relatively far away from the switches ends up with a lower correlation.

Sensor 23 is positioned to monitor the sofa in front of the TV, and the data suggest that it only detect motion if the user goes to the sofa immediately after entering the room. So not all sensors necessarily trigger in a room, depending on what the user decides to do in the room.

So in this case, the correlation still gives an excellent estimate of which switches and sensors are in the same room, by looking switch each sensor has the highest correlation probability too.

One thing to note is, these are the probabilities based solely on the statistical data, and that correlation corretions would be added onto this schema. So it is not a perfect reflect of which sensors are in the same room each switch, on it is own. But it does gives a good approximation.

5.2.3 Correlation based timeout

The implemented functionality of the correlation table, is to determine the timeout for each switch. How well is the correlation table able to keep the light on where it's needed. Different areas should have different timeouts, but most important aspect, is for the system to have long timeouts in areas where the user is likely to be still for extended periods of time, while still wanting the light to remain on. The most obvious area would be the sofa, where a user is likely to be for hours. Based on passive learning data, the system would have one of the lowest timeouts when the user is detected in the sofa, where it should be the highest.

However with active learning the correlation correction comes into effect. Every time the system incorrectly turns off the light, and the user turns it on again, the system is punished and increases the correlation, and by extension the timeout. As a result of this, the system will gradually increase the timeout until it no longer turns off the light, while the user is watching TV.

Power savings

The proposition for this project, was its ability to reduce energy consumption. One thing is learning power reducing behavior, another how well this learned behavior is able to reduce power consumption. This section is going to evaluate how well the system is able to turn the light off when it is not needed.

If the system was fully functional and installed, the ideal way to measure the energy savings would be to simply look at how much power the home consumes when the system is running, and how much it consumed before the system was installed. Since the system have not been installed in a home, where it is able to control the switches, this is not a possibility.

Since we are not able to evaluate the system based on its actual performance, due to the lack of a complete installation, we will have to analyze the collected data, and create estimates of how much energy the system is capable of saving.

A suitable room to analyze for energy savings is the living room. This is a room where most people do not turn off the light until they go to bed at night. This a place where our system would be much more vigorous about controlling the light. Therefore we are going to analyze how well the system would be able to learn, when to turn the light in the living room on and off, and how much power is saved by comparing automated light switching to the actual switch data. Unfortunately not all switch events have been logged, as there is a discrepancy of 40 more ON events than OFF events. So care has to be taken when analyzing how long the light have been on based on the switch events. We have looked at the data and chosen periods where the switch patterns look plausible.

First we examine who well the system have learned the patterns related to turning on and off the light, when the user enters or leaves the living room. (4/716) key: 20 23 13 on value: 0.005586592 (3/530) key: 20 24 13 on value: 0.0056603774 (2/593) key: 24 20 13 off value: 0.0033726813 (2/577) key: 23 20 13 off value: 0.0034662045

These are the patterns the system have detected. None of the patterns are above the confidence limit, but they have been detected more than once, so it is plausible that more user data would get them above the confidence limit. The probabilities are very low, but this is natural, for a room such as the living room, where lights are kept on most of the time. For the purpose of evaluating the potential energy saving, lets assume that the system has learned the four decision patterns listed above.

First attempt of analyzing the data this way revealed that the system was some times not able to detect the user reentering the living room. This would result in the system leaving the user in darkness for hours. This was due to the pattern being interrupted by other sensors. To fix this problem, the simulated user will turn on the light upon entering the living room, even if the ON pattern is not registered by the system.

Table 5.9: Power saving when running the system. The duration is how long the light was on for without the system, and the power saving how much time light was off with the system running

Date	Duration	Power saving
Dec 18th	11 hours	1.5 hours
Dec 23rd	15 hours	1 hour

For most other days the switch data was to unreliable to be used for evaluation. The data does not have any off events for the living room from December 26th to the 29th. Based on December 18th and 23rd, the system is able to reduce living room energy consumption by 10%.

This is a very low estimate of the systems energy reduction. Since we have identified that the system often did not recognize the correct ON pattern when the user returned to the living room, this is most likely also the case for detecting OFF patterns. This problem is a result of the system not being properly trained because of the relatively small amount of data available. The living room was chosen because it had the best available sensor data. It is however the room with the smallest potential for energy reductions, since it is the room where the user spends most of his time.

Chapter 6

Conclusion

The goal of the thesis was to examine the possibilities of incorporating machine learning technology in home control systems, and thus creating a smart environment, capable of micromanaging the homes energy consumption. The purpose of the thesis was to act as a proof of concept for this idea.

During the project we have successfully designed and implemented a system that uses state of the art machine learning algorithms, to evolve a behavioral pattern based on empirical data. By processing sensor data, the system has been able to identify key movement patterns that can be used to predict the users intentions. The software implementation can act as a solid base for future development, and potentially lead to a commercial implementation.

While we regard all of the achievements above as criteria for success, the most important evaluation of the project must be based on the results our system was able to produce. In a very early stage of analyzing the collected data, we realized that the amount of data we were able to collect would not be ideal. Due to changes in the living conditions in the apartment the system was installed in, this was however the most we could collect. While the amount of data does not allow us to draw any definitive conclusions, we were still able to identify some clear patterns that can be used to predict user behavior. As described in the "Evaluation" chapter the system was able to find and extensive list of patterns that have been seen at least five time, which is considerable compared to the amount of data collected.

As a result of the evaluation, we can certainly conclude that there exists a potential in integrating machine learning in home control systems, and by incorporating the technologies developed during the project, it is possible to guide the learning process of the system.

The system has also shown its potential for reducing energy consumption. While the estimated reduction for the living room was only 10 percent, this is a very modest estimate, based on the room with the lowest potential. It is very likely that a properly trained system will be able to produce an energy reduction that is several times higher.

6.1 Future work

Since the project is intended as a proof of concept study, this section could potentially be quite comprehensive. We have chosen only to discuss some of the most relevant additions that should be made to the project.

6.1.1 Active learning of switch patterns

The next phase of development for the project would be to get ready for the active learning stage. It would be necessary to create a fully functional installation of sensors and switches in a home, so the system is able to manipulate the light, and monitor the system's interaction with the user. The next incremental development stage should focus on allowing the system learn switch patterns based on active learning. This would allow the system to try and guess which switches should be turned on or off, and learn from the user's reactions. This step in the development would probably take several month, in order for the system to accumulate enough data, to create informed decisions.

6.1.2 Multiple users

The system is very sensitive to the noise from having multiple users in the same environment. Multiple users moving around, will break the movement patterns detected by the system, making it unrealistic for the system to learn anything meaningful.

A way to solve this problem could be to have a thread for each user moving around. The challenge then becomes matching each motion event to the right user. The correlation table gives a good estimate of which sensors are in the same room. By assuming the patterns of each user is made up of adjacent sensor events, the system would be able to track each pattern separately, as long as the users doesn't get near each other.

6.1.3 Switch and sensor correlation

We base our statistical correlation table on the assumption, that a user will most likely turn on the light where he is, and look at the interval just after a switch is turned on. A way to augment that analysis, is by flipping the assumption on its head, that the user will most likely turn off the light where he is not. The user is most likely not going to be where the lights are off, so any sensors activated when the lights are off, are most likely not in the same room as the switch.

6.1.4 Decision matrix persistency

The longer back in time the system looks for user data, the more likely it is too see each pattern multiple times. The more times the system sees a given pattern, the more precise estimates the system can calculate of the probabilities for that pattern. However the system should also be able to react to changes in user behavior, so there is a limit to how long back in time the system should look.

To be able to best react to changes, the system should only keep the most recent data. But this would drastically reduce the systems confidence in the decision matrix. A static way to solve the problem would be to always look a fixed period of time back, attempting to strike a balance between the systems confidence and ability to react.

A dynamic way to solve the problem would be to compare the most recent patterns to the old patterns. As long as there is a reasonably low discrepancy, the system can keep using old data. And if the discrepancy gets too big, the system base it decisions purely on recent data, to better react to the changes in user behavior.

6.1.5 Additional hardware

There are many types of hardware it would be interesting to add to the system. Lux sensors would allow the system to not waste power if there is enough natural light present. Pressure sensors in chairs and furniture, would allow the system to detect users when they are sitting still. However these hardware additions does not simplify the system, but instead adds complexity and means the system needs consider more variables. While some of these addition will certainly be necessary for the system to reach a commercial level, the potential of the system in its current state should first be fully explored. This will require a fully implemented system.

Bibliography

[1] Awareness of Climate Change and Threat Vary by Region, Anita Pugliese and Julie Ray, http://www.gallup.com/poll/124652/awareness-climate-change-threat-vary-region.aspx, December 11, 2009

[2] Annual Energy Review, US Energy Information Administraion, http://205.254.135.24/totalenergy/data/annual/showtext.cfm?t=ptb0201a, October 19, 2011

[3] Boguslaw Pilich. Engineering Smart Houses, DTU IMM MSc Thesis Nr. 49/2004

[4] Wikipedia article on machine learning. http://en.wikipedia.org/wiki/Machine_learning, February 2012

[5] INSTEON. http://www.insteon.net

[6] Wipedia article on the Clipsal C-Bus protocol. http://en.wikipedia.org/wiki/C-Bus_(protocol), February 2012

[7] Mads Ingwar and Soeren Kristian Jensen. IMM Smart House Project: a state of the art survey. 2008.

[8] Lauritz Knudsens. http://www.lk.dk

[9] Aware Home Research Initiative. http://awarehome.imtc.gatech.edu

[10] http://awarehome.imtc.gatech.edu/publications

[11] Wikipedia article on markov chains. http://en.wikipedia.org/wiki/Markov_chain, February 2012

[12] Sune Keller and Martin Skytte Kristensen. Simulation and visualization of intellight light control system. Bachelor thesis 2010.

.1 Source Listings

Appendix A

Source Listings

A.1 Package: smarthouse

A.1.1 SmartHouse.java

```java
package smarthouse;

import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import timer.TimeoutEvent;
import timer.TimeoutListener;
import timer.Timer;

import events.*;
import config.Config;
import core.*;

/**
 * @author Andreas & David
 */
public class SmartHouse implements TimeoutListener {

  private static boolean debug = true;
```

```java
Connection conn = null;
Statement stmt;
AI ai;
EventList eventlist,zoneeventlist;
Correlation correlation;
Timer timer;
List<Integer> timeout;
int onTime;
int punishmentTimeout;
Map<Integer, Boolean> switchStatus;
Map<Integer, Integer> firstSensorAfterTimeout;
DecisionMatrix decisionMatrix;
public static void main(String[] args){
  SmartHouse sh = new SmartHouse();
}

/*
 * Constructor for the class SmartHouse
 * Handles the input and output for the ai
 */
public SmartHouse(){
  Config.loadConfig();
  try {
    debug = Config.debug;
    Class.forName("com.mysql.jdbc.Driver");//load the mysql driver
    conn = DriverManager.getConnection(Config.DB);//connect to the
        database
    stmt = conn.createStatement();
    decisionMatrix = new DecisionMatrix();
    correlation = new Correlation();

    eventlist = new EventList();
    zoneeventlist = new EventList(true);
    timer = new Timer();
    timeout = new ArrayList<Integer>(10);
    onTime = Config.defaultOnTime;
    punishmentTimeout = Config.punishmentTimeout;
    firstSensorAfterTimeout = new HashMap<Integer, Integer>();
    switchStatus = new HashMap<Integer, Boolean>();
    for (int sw : decisionMatrix.switches){
      switchStatus.put(sw,false);
    }
  }
  catch (SQLException se){
    System.out.println("SQLException: " + se.getMessage());
    System.out.println("SQLState: " + se.getSQLState());
    System.out.println("VendorError: " + se.getErrorCode());

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

public SmartHouse(AI ai) {
  this();
  this.ai = ai;
}

/*
 * Method called when a sensorevent occurs in the simulator
 * @author Andreas & David
 */
public void sensorEvent(int sensorId){
  try{
```

```java
      System.out.println("Sensor "+sensorId+" fired!");
      eventlist.sensorEvent(sensorId);
      zoneeventlist.sensorEvent(sensorId);

      if (!debug)
        stmt.executeUpdate("INSERT INTO sensor_events VALUES("+sensorId+",NOW())");

      for (int sw : timeout) {
        if (!firstSensorAfterTimeout.containsKey(sw))
          firstSensorAfterTimeout.put(sw, sensorId);
      }
      for (int sw : correlation.getSwitches(sensorId, 0.5f)) {
        if (isOn(sw) && !timeout.contains(sw)) {
          float t = onTime * correlation.getCorrelation(sw, sensorId);
          System.out.printf("keep switch %d on (%d ms)\n", sw, (long) t);
          timer.updateTimeout(sw, (long) t, this);
        }
      }
    }
    catch(SQLException se){
      System.out.println("SQLException: " + se.getMessage());
      System.out.println("SQLState: " + se.getSQLState());
      System.out.println("VendorError: " + se.getErrorCode());
    }
    matrixLookUp();
  }
  /*
   * Method called when a switch event occurs in the simulator
   * @author Andreas & David
   */
  public void switchEvent(int switchId, int status){
    try{
      System.out.println("Switch "+switchId+" turned "+status);
  //            System.out.println(eventlist);
      boolean cmd = (status == 1) ? true : false;

      if (cmd) {
        if (timeout.contains(switchId)) {
          timeout.remove((Object) switchId);
          timer.stop(switchId);
          if (firstSensorAfterTimeout.containsKey(switchId))
            correlation.increaseCorrelation(switchId, firstSensorAfterTimeout.get(switchId));
        }
        on(switchId);
        timer.setTimeout(switchId, onTime, this);

      } else {
        off(switchId);
      }
      if (!debug)
        stmt.executeUpdate("INSERT INTO switch_events VALUES("+switchId+","+status+",NOW())");
    }
    catch(SQLException se){
      System.out.println("SQLException: " + se.getMessage());
      System.out.println("SQLState: " + se.getSQLState());
      System.out.println("VendorError: " + se.getErrorCode());
    }
  }

  private Map<Integer, Boolean> testMap = new HashMap<Integer, Boolean>();
```

```
     public void TimeoutEventOccurred(TimeoutEvent event) {
    System.out.println("I should probably turn off the light now");
    int id = (Integer) event.getSource();
    if (timeout.contains(id) && eventlist.getLastEvent() != null) {
      correlation.reduceCorrelation(id, eventlist.getLastEvent().getID()
          );// adjust for zoneeventlist
      timeout.remove(event.getSource());
    } else {
      off(id);
      timeout.add(id);
      timer.setTimeout(id, punishmentTimeout, this);
    }
  }
  private void matrixLookUp(){
    try{
      KeyList keylist;
      int P;
      float value = 0;
      for (int sw : decisionMatrix.switches){
        keylist = new KeyList(eventlist);
        keylist.add(sw);
        if(switchStatus.get(sw)){
          if(decisionMatrix.off.containsKey(keylist)){
            value = decisionMatrix.off.get(keylist);
          }
          System.out.println("probability value : "+value);
          if(value>Config.probabilityThreshold){
            off(sw);
          }
          if(Config.useZones){
            if(decisionMatrix.off.containsKey(keylist)){
              keylist = new KeyList(zoneeventlist);
              keylist.add(sw);
              value = decisionMatrix.off.get(keylist);
            }

          }

        }
        else{
          if(decisionMatrix.on.containsKey(keylist)){

            value = decisionMatrix.on.get(keylist);
          }
          if(Config.useZones){
            if(decisionMatrix.on.containsKey(keylist)){
              keylist = new KeyList(zoneeventlist);
              keylist.add(sw);
              value = decisionMatrix.on.get(keylist);
            }

          }
          System.out.println("probability value for switch  "+sw+" : "+
              value);
          if(value>Config.probabilityThreshold){
            on(sw);
          }

        }
      }
    }
    catch(Exception e){
      e.printStackTrace();
    }
```

```
  }

  private void on(int id) {
    System.out.println("Turning switch "+id+" on");
    ai.on(id);
    switchStatus.put(id, true);
  }

  private void off(int id) {
    System.out.println("Turning switch "+id+" off");
    ai.off(id);
    switchStatus.put(id, false);
  }

  private boolean isOn(int id) {
    if (switchStatus.containsKey(id))
      return switchStatus.get(id);

    return false;
  }
}
```

Listing A.1: SmartHouse.java

A.1.2 AI.java

```
package smarthouse;

public interface AI {

    public void on(int id);
    public void off(int id);

}
```

Listing A.2: AI.java

A.2 Package: timer

A.2.1 Sleeper.java

```
package timer;

import javax.swing.event.EventListenerList;

/**
 * @author David
 */
public class Sleeper extends Thread {

    private int id;
    private long time;
```

```java
    private long end;
    private TimeoutListener listener;

    public static void main(String args[]) throws InterruptedException {
        System.out.println("here we go...");
        new Sleeper(1, 1000);
        new Sleeper(2, 2000);
        new Sleeper(2, 2000);
        new Sleeper(3, 3000).join();
        System.out.println("all done");
    }

    public Sleeper(int id, long time) {
        this.id = id;
        this.time = time;
        this.end = System.currentTimeMillis() + time;
        this.start();
    }

    public Sleeper(int id, long time, TimeoutListener l) {
        this(id, time);
        this.listener = l;
    }

    public long getEnd() {
        return end;
    }

    public void run() {
        try {
            sleep(time);
            System.out.println(id + ": done");

            if (listener != null) {
                listener.TimeoutEventOccurred(new TimeoutEvent(id));
                System.out.println(id + ": event fired");
            }
        } catch (InterruptedException ex) {
            return;
        }
    }
}
```

Listing A.3: Sleeper.java

A.2.2 Timer.java

```java
package timer;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.swing.event.EventListenerList;

/**
 * @author David
 */
public class Timer implements TimeoutListener {

```

```java

    private Map<Integer, Sleeper> timers;
    private TimeoutListener listener;

    public static void main(String[] args) throws Exception{
        Timer t = new Timer();
        t.setTimeout(1, 1000, t);
        t.setTimeout(2, 2000, t);
        t.setTimeout(3, 2000, t);
        Thread.sleep(1000);
        t.setTimeout(3, 2000, t);
    }

    public Timer() {
        timers = new HashMap<Integer, Sleeper>();
    }

    public Timer(TimeoutListener l) {
        this.listener = l;
    }

    public void setTimeout(int id, long time) {
        setTimeout(id, time, listener);
    }

    public void setTimeout(int id, long time, TimeoutListener l) {
        if (timers.containsKey(id))
            timers.get(id).interrupt();

        timers.put(id, new Sleeper(id, time, l));
    }

    /**
     * set the timeout, only if a timer is already is set for the id,
     * and the new timeout will end later than the old timeout
     * @param id
     * @param time
     */
    public void updateTimeout(int id, long time, TimeoutListener l) {
       if (!timers.containsKey(id) || !timers.get(id).isAlive())
           return;

       if (timers.get(id).getEnd() < System.currentTimeMillis() + time)
           setTimeout(id, time, l);
    }

    public void updateTimeout(int id, long time) {
        updateTimeout(id, time, listener);
    }

    public void stop(int id) {
        timers.get(id).interrupt();
    }

    @Override
    public void TimeoutEventOccurred(TimeoutEvent event) {
        // TODO Auto-generated method stub
        System.out.println(event.getSource() + ": event detected");
    }

}
```

Listing A.4: Timer.java

A.2.3 TimeoutListener.java

```
package timer;

import java.util.EventListener;

public interface TimeoutListener extends EventListener {

    public void TimeoutEventOccurred(TimeoutEvent event);

}
```

Listing A.5: TimeoutListener.java

A.2.4 TimeoutEvent.java

```
package timer;

import java.util.EventObject;

public class TimeoutEvent extends EventObject {

    public TimeoutEvent(int id) {
        super(id);
    }

}
```

Listing A.6: TimeoutEvent.java

A.3 Package: events

A.3.1 EventList.java

```
package events;

import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;

import config.Config;

/**
 * @author David
 */
public class EventList {

    private LinkedList<Event> events;
//    private LinkedList<Event> zone;

    /**
```

```java
     * Maximum interval between sensor events, for the event to be considered a zone event.
     *  Default value 1 sec.
     */
    private int zone_interval;

    /**
     * Time interval stored in the event list.
     */
    private int pattern_interval;
    private int pattern_length;
    private boolean useZones;

    public EventList() {
        events = new LinkedList<Event>();
        this.pattern_interval = Config.patternInterval;
        this.pattern_length = Config.patternLength;
        this.zone_interval = Config.zoneInterval;
        this.useZones = Config.useZones;
    }

    public EventList(boolean useZones) {
        this();
        this.useZones = useZones;
    }

    public EventList(int zone_interval, int pattern_interval, int pattern_length) {
        this();
        if (zone_interval <= 0) {
            useZones = false;
        } else {
            useZones = true;
        }
        this.zone_interval = zone_interval;
        this.pattern_interval = pattern_interval;
        this.pattern_length = pattern_length;
    }



    /**
     * Add event
     * @param e
     */
    public void add(Event e) {
        removeOld(e.getTS());

        if(useZones && e instanceof SensorEvent)
            determineZone(e);
        else
            events.add(e);

        while (events.size() > pattern_length)
            events.removeFirst();
    }

    /**
     * removes all events if more than pattern interval has passed since the last event
     * also mantains a maximum pattern depth
     */
    private void removeOld(long time) {
        if(events.size() > 0 && time - events.getLast().getTS() > pattern_interval)
```

```java
            events.clear();

    }

    private int currentPatternLength() {
        int count = 0;
        for (Event e : events)
            if (e instanceof SensorEvent || e instanceof ZoneEvent)
                count++;
        return count;
    }

    private void determineZone(Event e) {
        if (events.size() > 0 && events.getLast().getTS() +
            zone_interval > e.getTS()) {

            Event last = events.getLast();
            if (last instanceof ZoneEvent) {
                boolean contains = false;
                ZoneEvent z = (ZoneEvent) last;
                for (int id : z.getIDs()) {
                    if (id == e.getID()) {
                        contains = true;
                        break;
                    }
                }
                if (!contains) {
                    z.addID(e.getID());
                    return;
                }

            } else if (last instanceof SensorEvent){
                if (last.getID() != e.getID()) {
                    events.removeLast();
                    events.addLast(new ZoneEvent(last.getTS(), last.
                        getID(), e.getID()));
                    return;
                }
            }
        }
        events.add(e);
    }

    public String toString() {
        StringBuffer sb = new StringBuffer("=== Event list ===\n");
        for (Event e : events) {
            sb.append(e.toString() + "\n");
        }
        return sb.toString();
    }

    public void sensorEvent(int id) {
        add(new SensorEvent(id));
    }

    public void switchEvent(int id, int status) {
        boolean cmd = (status == 0) ? false : true;
        add(new SwitchEvent(id, cmd));
    }

    /**
     * get events in event list, including detected zone events
     * @return
     */
    public Event[] getEvents() {
```

```java
        Event[] array = new Event[events.size()];
        events.toArray(array);
        return array;
    }

    public Event[] getDistinctEvents() {
        HashSet<Event> set = new HashSet<Event>(events);
        Event[] array = new Event[set.size()];
        set.toArray(array);
        return array;
    }

    /**
     * get only sensor and zone events
     * @return
     */
    public Event[] getPattern() {
        Event[] pattern = new Event[pattern_length];
        //if current pattern depth is less than pattern depth, fill missing with -1
        for (int i = 0; i < pattern_length - currentPatternLength(); i++) {
            pattern[i] = new SensorEvent(-1);
        }

        Iterator<Event> it = events.iterator();
        for (int i = pattern_length - currentPatternLength(); i < pattern_length; i++) {
            pattern[i] = it.next();
        }
        return pattern;
    }

    public Event getLastEvent() {
        if (events.size() > 0)
            return events.getLast();

        return null;
    }
  public boolean containsZoneEvent(){
    if(useZones){
      for(Event e : events){
        if (e instanceof ZoneEvent)
            return true;
      }
    }
    return false;
  }
}
```

Listing A.7: EventList.java

A.3.2 Event.java

```java
package events;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
```

```
 * @author David
 */
public abstract class Event {

    private static SimpleDateFormat sdm = new SimpleDateFormat("[HH:mm:
        ss]");

    protected int id;
    protected long ts;

    public Event(int id, long ts) {
        this.id = id;
        this.ts = ts;
    }

    public Event(int id) {
        this(id, System.currentTimeMillis());
    }

    public int getID() {
        return id;
    }

    public long getTS() {
        return ts;
    }

    public boolean compareID(int id) {
        return this.id == id;
    }
    public boolean equals(Object o) {
        if (!(o instanceof Event)) {
            return false;
        }
        Event e = (Event) o;
        if (e.id != this.id)
            return false;
        if (e.ts != this.ts)
            return false;

        return true;
    }

    public int hashCode() {
        return id ^ (int) ts;
    }

    /**
     * return timestamp as human readable string
     * @return
     */
    public String tsString(){
        return sdm.format(new Date(ts));
    }

}
```

Listing A.8: Event.java

A.3.3 SensorEvent.java

```java
package events;

/**
 * @author David
 */
public class SensorEvent extends Event {

    public SensorEvent(int id, long ts) {
        super(id, ts);
    }

    public SensorEvent(int id) {
        super(id);
    }

    public String toString() {
        return tsString() + " Sensor event " + this.id;
    }

    public boolean equals(Object o) {
        if (!super.equals(o))
            return false;

        if (!(o instanceof SensorEvent))
            return false;

        return true;
    }

}
```

Listing A.9: SensorEvent.java

A.3.4 ZoneEvent.java

```java
package events;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

/**
 * @author David
 */
public class ZoneEvent extends Event {

    protected int[] ids;

    public ZoneEvent(int ... ids) {
        super(0);
        Arrays.sort(ids);
        this.ids = ids;

        this.id = getID(ids);
    }

    public ZoneEvent(long ts, int ... ids) {
        this(ids);
        this.ts = ts;
        this.id = getID(ids);
```

```
    }

    public ZoneEvent(List<Event> zone) {
        this(zone, System.currentTimeMillis());
    }

    public ZoneEvent(List<Event> zone, long ts) {
        super(0);

        ids = new int[zone.size()];
        for(int i = 0; i < zone.size(); i++)
            ids[i] = zone.get(i).getID();

        Arrays.sort(ids);

        this.id = getID(ids);
        this.ts = zone.get(zone.size()-1).getTS();
    }

    private static int getID(int ...ids) {
        int sum = 0;
        for (int i : ids)
            sum = sum*256 + i;

        return sum;
    }

    public int[] getIDs() {
        return ids;
    }

    public void addID(int id) {
        int[] tmp = new int[ids.length + 1];
        tmp[0] = id;
        System.arraycopy(ids, 0, tmp, 1, ids.length);
        ids = tmp;
        Arrays.sort(ids);
        this.id = getID(ids);
    }


    /**
     * overrides the super class method compareID, to compare idx to all
         the ids in the zone event
     */
    @Override
    public boolean compareID(int idx) {
        for(int id : ids) {
            if (id == idx)
                return true;
        }
        return false;
    }

    public String toString() {
        return tsString() + " Zone event " + Arrays.toString(ids);
    }

    public boolean equals(Object o) {
        if (!super.equals(o))
            return false;

        if (!(o instanceof ZoneEvent))
            return false;

```

```java
        ZoneEvent e = (ZoneEvent) o;
        if (e.ids.length != this.ids.length)
            return false;

        for (int i = 0; i < e.ids.length; i++) {
            if (e.ids[i] != this.ids[i])
                return false;
        }
        return true;
    }

    /**
     *
     * @param id
     * @return
     */
    public static List<Integer> getIDs (int id) {
        LinkedList<Integer> ids = new LinkedList<Integer>();
        while(id > 0) {
            ids.addFirst(id % 256);
            id /= 256;
        }

        return ids;
    }

    public static String getIDString(int id) {
        if (id < 256)
            return Integer.toString(id);

        StringBuffer sb = new StringBuffer("[");
        for (int i : getIDs(id))
            sb.append(i + ",");
        sb.setCharAt(sb.length()-1, ']');

        return sb.toString();
    }
}
```

Listing A.10: ZoneEvent.java

A.3.5 SwitchEvent.java

```java
package events;

/**
 * @author David
 */
public class SwitchEvent extends Event {

    protected boolean cmd;

    public SwitchEvent(int id, long ts, boolean cmd) {
        super(id, ts);
        this.cmd = cmd;
    }

    public SwitchEvent(int id, boolean cmd) {
        super(id);
        this.cmd = cmd;
```

```
    }

    public boolean getCmd() {
        return cmd;
    }

    public String toString() {
        return tsString() + " Switch event " + this.id +
                ((cmd) ? " on" : " off");
    }

    public boolean equals(Object o) {
        if (!super.equals(o))
            return false;

        if (!(o instanceof SwitchEvent))
            return false;

        SwitchEvent e = (SwitchEvent) o;
        if (e.cmd != this.cmd)
            return false;

        return true;
    }
}
```

Listing A.11: SwitchEvent.java

A.4 Package: config

A.4.1 Config.java

```
package config;
import java.io.*;
import java.util.Scanner;

/**
 * @author Andreas
 */
public class Config{

    /**
     * database
     */
    public static String DB = "jdbc:mysql://localhost/kiiib_dev?user=
        KIIIB&password=42";
    /**
     * pattern length for markov chains
     */
    public static int patternLength = 2;
    /**
     * maximum time interval in ms, for events to count as a pattern
     */
    public static int patternInterval = 10*1000;
    /**
     * maximum time interval in ms, for events to count as a zone event
     */
```

```
    public static int zoneInterval = 500;
    /**
     * the interval after an on event, that sensor events is considered
         to be correlated to the switch
     */
    public static int correlationInterval = 7*1000;
    /**
     * minimum correlation probability for a sensor to extend the
         timeout of a switch
     */
    public static float probabilityThreshold = .5f;
    /**
     * should the system detect zone events
     */
    public static boolean useZones = true;
    /**
     * base timeout for all switches in ms
     */
    public static int defaultOnTime = 5000;
    /**
     * the interval after a switch is turned off based on timeout, that
         the system considers a on event a punishment
     */
    public static int punishmentTimeout = 10*1000;
    /**
     * the correlation correction when the system is punished
     */
    public static float correlationCorrectionStep = .1f;
    /**
     * flag for when the system is in debug mode
     * used toggle debug output
     * also toggles simulator logging motion and switch event to
         database (doesn't log in debug mode)
     */
    public static boolean debug = false;

    public static void main(String[] args) {
        Config.loadConfig();
    }

    public static void loadConfig(){
        System.out.println("Loading Configurations");
        try{
            File f = new File("kiiib.settings");
            if(!f.exists()){
                System.out.println("could not find preferences file,
                    generating a new one");
                f.createNewFile();
                FileWriter fstream = new FileWriter(f);
                BufferedWriter out = new BufferedWriter(fstream);
                out.write("#automatically generated preferences file\n#
                    delete to return to default settings\n");
                out.write("DB " + DB +"\n");

                out.write("pattern_interval " + patternInterval + "\n");
                out.write("pattern_length " + patternLength + "\n");

                out.write("use_zones " + useZones + "\n");
                out.write("zone_interval " + zoneInterval + "\n");

                out.write("probability_threshold " +
                    probabilityThreshold + "\n");
                out.write("correlation_interval " + correlationInterval+
                    "\n");
```

```
                out.write("correlation_correction " + correlationCorrectionStep +"\n");
                out.write("default_on_time " + defaultOnTime +"\n");
                out.write("punishment_timeout " + punishmentTimeout +"\n");

                out.write("debug " + debug +"\n");
                out.close();

            }
            else{
                Scanner scan = new Scanner(f);
                String token;
                while(scan.hasNextLine()){
                    token = scan.next();
                    if(token.equals("pattern_length")){
                        patternLength = Integer.parseInt(scan.next());
                        System.out.println("pattern_length = "+patternLength);
                    }
                    else if(token.equals("pattern_interval")){
                        patternInterval = Integer.parseInt(scan.next());
                        System.out.println("pattern_interval = "+patternInterval);
                    }
                    else if(token.equals("use_zones")){
                        useZones = Boolean.parseBoolean(scan.next());
                        System.out.println("use_zones = "+useZones);
                    }
                    else if(token.equals("zone_interval")){
                        zoneInterval = Integer.parseInt(scan.next());
                        System.out.println("zone_interval = "+zoneInterval);
                    }
                    else if(token.equals("probability_threshold")){
                        probabilityThreshold = Float.parseFloat(scan.next());
                        System.out.println("probablility_threshold = "+probabilityThreshold);
                    }
                    else if(token.equals("correlation_interval")){
                        correlationInterval = Integer.parseInt(scan.next());
                        System.out.println("correlation_interval = " + correlationInterval);
                    }
                    else if(token.equals("correlation_correction")){
                        correlationCorrectionStep = Float.parseFloat(scan.next());
                        System.out.println("correlation_correction = " + correlationCorrectionStep);
                    }
                    else if(token.equals("default_on_time")){
                        defaultOnTime = Integer.parseInt(scan.next());
                        System.out.println("default_on_time = " + defaultOnTime);
                    }
                    else if(token.equals("punishment_timeout")){
                        punishmentTimeout = Integer.parseInt(scan.next());
                        System.out.println("punishment_timeout = " + punishmentTimeout);
                    }
                    else if(token.equals("DB")){
                        DB = scan.next();
```

```
                        System.out.println("Database = " + DB);
                    }
                    else if(token.equals("debug")){
                        debug = Boolean.parseBoolean(scan.next());
                        System.out.println("debug = " + debug);
                    }
                    scan.nextLine();

                }
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }
        catch(Exception e){
            System.out.println("could not read preferences file... using
                default settings");
        }
    }
}
```

Listing A.12: Config.java

A.5 Package: core

A.5.1 Correlation.java

```
package core;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import timer.TimeoutEvent;
import timer.TimeoutListener;

import config.Config;

import events.*;

/**
 * @author David
 */
public class Correlation implements TimeoutListener {

    private Statement stmt;
```

```
    private Connection conn;
    private ResultSet result;
    private long correlation_interval = 7*1000;
    private float correction;
    private Map<Integer, Map<Integer, Float>> correlation;

    public static void main(String[] args) throws IOException {
        System.out.println(new Correlation());
    }

    public Correlation () {
        correlation = new HashMap<Integer, Map<Integer, Float>>();
        try {
            Class.forName("com.mysql.jdbc.Driver");//load the mysql
                driver
            conn = DriverManager.getConnection(Config.DB);//connect to
                the database
            stmt = conn.createStatement();
        }
        catch (SQLException se){
            System.out.println("SQLException: " + se.getMessage());
            System.out.println("SQLState: " + se.getSQLState());
            System.out.println("VendorError: " + se.getErrorCode());

        }
        catch (Exception e){
             e.printStackTrace();
        }
        correction = Config.correlationCorrectionStep;
        generateCorrelation();
//          getStoredCorrelations();
    }

    public float getCorrelation(int switchId, int sensorId) {
        if (! correlation.containsKey(switchId))
            return 0;

        if (! correlation.get(switchId).containsKey(sensorId))
            return 0;

        return correlation.get(switchId).get(sensorId);
    }

    public static void incrementSwitchCount(Map<Integer, Integer>
        switch_count, int id) {
        if (!switch_count.containsKey(id))
            switch_count.put(id, 1);
        else
            switch_count.put(id, switch_count.get(id) + 1);
    }

    public static void incrementSensorCount(Map<Integer, Map<Integer,
        Integer>> sensor_count, int switchId, int sensorId) {
        if (!sensor_count.containsKey(switchId)) {
            sensor_count.put(switchId, new HashMap<Integer, Integer>());
        }

        Map<Integer, Integer> map = sensor_count.get(switchId);
        if (!map.containsKey(sensorId)) {
            map.put(sensorId, 1);
        } else {
            map.put(sensorId, map.get(sensorId) + 1);
        }
    }

```

```java
    private void updateCorrelation(int sw, int se, float corr) {
        if (correlation.containsKey(sw)) {
            Map<Integer, Float> map = correlation.get(sw);
            if (map.containsKey(se)) {
                map.put(se, Math.max(0, map.get(se) + corr));
            }
        }
    }

    public void generateCorrelation() {

        try {
            Map<SwitchEvent, EventList> switch_eventlist = new HashMap<
                SwitchEvent, EventList>();
            Map<Integer, Integer> switch_count = new HashMap<Integer,
                Integer>();
            Map<Integer, Map<Integer, Integer>> sensor_count = new
                HashMap<Integer, Map<Integer, Integer>>();
            LinkedList<SwitchEvent> gc = new LinkedList<SwitchEvent>();

            result = stmt.executeQuery("(select id,timestamp,'sensor' AS
                type, '0' AS status from sensor_events) union (select
                id,timestamp,'switch' AS type,status from switch_events)
                order by timestamp;");
            while(result.next()) {
                int id = result.getInt("id");
                long ts = result.getTimestamp("timestamp").getTime();
                if (result.getString("type").equals("switch")) {
                    boolean cmd = (result.getInt("status") == 1) ? true
                        : false;
                    if (cmd) {
                        SwitchEvent s = new SwitchEvent(id, ts, cmd);
                        switch_eventlist.put(s, new EventList(Config.
                            zoneInterval, Config.correlationInterval,
                            Integer.MAX_VALUE));
                        gc.addLast(s);
                    }
                } else if (result.getString("type").equals("sensor")) {
                    for (SwitchEvent e : switch_eventlist.keySet()) {
                        if (e.getTS() + correlation_interval > ts) {
                            switch_eventlist.get(e).add(new SensorEvent(
                                id, ts));
                        }
                    }
                }

                while(gc.size() > 0 && gc.getFirst().getTS() +
                    correlation_interval < ts) {
                    SwitchEvent se = gc.getFirst();
                    incrementSwitchCount(switch_count, se.getID());

                    for (Event e : new HashSet<Event>(Arrays.asList(
                        switch_eventlist.get(se).getEvents()))) {
                        incrementSensorCount(sensor_count, se.getID(), e
                            .getID());
                    }
                    gc.removeFirst();
                    switch_eventlist.remove(se);
                }

                for(int sw : sensor_count.keySet()) {
                    Map<Integer, Float> map = new HashMap<Integer, Float
                        >();
                    for (int se : sensor_count.get(sw).keySet()) {
```

```java
                        map.put(se, (float) sensor_count.get(sw).get(se) / switch_count.get(sw));
                    }
                    correlation.put(sw, map);
                }
            }
            int i = 0;
            while(gc.size() > 0) {
                SwitchEvent se = gc.getFirst();
                incrementSwitchCount(switch_count, se.getID());

                for (Event e : new HashSet<Event>(Arrays.asList(switch_eventlist.get(se).getEvents()))) {
                    incrementSensorCount(sensor_count, se.getID(), e.getID());
                }
                gc.removeFirst();
                switch_eventlist.remove(se);
            }
            for(int sw : sensor_count.keySet()) {
                Map<Integer, Float> map = new HashMap<Integer, Float>();
                for (int se : sensor_count.get(sw).keySet()) {
                    map.put(se, (float) sensor_count.get(sw).get(se) / switch_count.get(sw));
                }
                correlation.put(sw, map);
            }
        } catch (SQLException se){
            se.printStackTrace();
            System.out.println("SQLException: " + se.getMessage());
            System.out.println("SQLState: " + se.getSQLState());
            System.out.println("VendorError: " + se.getErrorCode());
        }
    }

    public Set<Integer> getSwitches() {
        return new TreeSet<Integer>(correlation.keySet());
    }

    public Set<Integer> getSensors() {
        Set<Integer> sensors = new TreeSet<Integer>();
        for(int sw : correlation.keySet()) {
            sensors.addAll(correlation.get(sw).keySet());
        }
        return sensors;
    }

    /**
     * get a list of switches, that have a correlation with a sensor above the threshold
     * @param sensor
     * @param threshold 0 <= x <= 1
     * @return
     */
    public List<Integer> getSwitches(int sensor, float threshold) {
        List<Integer> list = new LinkedList<Integer>();
        for (int sw : correlation.keySet()) {
            Map<Integer, Float> map = correlation.get(sw);
            if (!map.containsKey(sensor))
                continue;

            if (map.get(sensor) > threshold)
                list.add(sw);
        }
        return list;
```

```
    }

    public String toString() {
        StringBuilder sb = new StringBuilder(1024);
        sb.append("Corr.\t");
        for (int s : getSensors())
            sb.append(ZoneEvent.getIDString(s) + "\t");
        sb.append("\n");

        for (int sw : getSwitches()) {
            sb.append(sw + "\t");
            for (int se : getSensors()) {
                if (correlation.get(sw).containsKey(se)) {
                    float f = correlation.get(sw).get(se);
                    if (f >= 0.5)
                        sb.append("*");
                    if (f > 0)
                        sb.append(String.format("%.2f\t", f));
                    else
                        sb.append("\t");
                } else {
                    sb.append("0\t");
                }
            }
            sb.append("\n");
        }
        return sb.toString();
    }

    @Override
    public void TimeoutEventOccurred(TimeoutEvent event) {
        // TODO Auto-generated method stub

    }

    public void increaseCorrelation(int sw, int se) {
        System.out.println("Increase correlation " + sw + "~" +se);
        storeCorrelation(sw, se, Config.correlationCorrectionStep);
        updateCorrelation(sw, se, correction);
        storeCorrelation(sw, se, correction);
    }

    public void reduceCorrelation(int sw, int se) {
        System.out.println("Reduce correlation " + sw + "~" +se);
        storeCorrelation(sw, se, -Config.correlationCorrectionStep);
        updateCorrelation(sw, se, -correction);
        storeCorrelation(sw, se, -correction);
    }

    public void getStoredCorrelations() {
        String query = "SELECT switch, sensor, correlation FROM
            correlation_confirmation";
        try {
            result = stmt.executeQuery(query);
            while(result.next()) {
                int sw = result.getInt("switch");
                int se = result.getInt("sensor");
                float corr = result.getFloat("correlation");
                updateCorrelation(sw, se, corr);
            }
        } catch (SQLException ex){
            ex.printStackTrace();
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
```

```
            }
        }

        /**
         * insert correlation correction into sql table
         * @param sw switch id
         * @param se sensor id
         * @param corr correlation change
         */
        public void storeCorrelation(int sw, int se, float corr) {
            String query = String.format("INSERT INTO
                correlation_confirmation " +
                "(switch, sensor, correlation) VALUES (%d, %d, %f) " +
                "ON DUPLICATE KEY UPDATE correlation = correlation + %f; ",
                    sw, se, corr, corr);
            try {
                stmt.executeUpdate(query);
            } catch (SQLException ex){
                ex.printStackTrace();
                System.out.println("SQLException: " + ex.getMessage());
                System.out.println("SQLState: " + ex.getSQLState());
                System.out.println("VendorError: " + ex.getErrorCode());
            }

        }
}
```

Listing A.13: Correlation.java

A.5.2 DecisionMatrix.java

```
package core;

import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.util.HashMap;
import config.Config;
import core.KeyList;

import java.util.Date;
import java.util.LinkedList;
import java.util.ArrayList;
import events.*;

/**
 * @author Andreas
 */
public class DecisionMatrix{
    public HashMap<KeyList, Float> on, off;
    private HashMap<KeyList, Integer> count;
    private Statement stmt;
    private Connection conn;
    private LinkedList<Integer> eventBuffer; // holds the last n
        sensorevents, n = memoryDepth
    public ArrayList<Integer> switches,sensors;

    /**
```

```
     * temporary main method for testing puposes
     * @author Andreas
     **/
    public static void main(String[] args){
        Config.loadConfig();
        DecisionMatrix dm = new DecisionMatrix();
    }

    public DecisionMatrix(){
        connect2DB();
        generateBasicMatrices();
        if(Config.useZones)
            generateZoneMatrices();
      // printTables();
        System.out.println("switches");
        for(int i : switches){
           System.out.println(i);
        }
        System.out.println("sensors");
        for(int i : sensors){
           System.out.println(i);
        }
        printMatrices();
    }
    /**
     * Connects to the database, and initiates the statement object to
         be used later
     * @author Andreas
     **/
    public void connect2DB(){
         try {
            System.out.println("Trying to connect to the database");
            Class.forName("com.mysql.jdbc.Driver");//load the mysql
                driver
            conn = DriverManager.getConnection(Config.DB);//connect to
                the database
            stmt = conn.createStatement();
            System.out.println("connection established");


        }
        catch (SQLException se){
            System.out.println("SQLException: " + se.getMessage());
            System.out.println("SQLState: " + se.getSQLState());
            System.out.println("VendorError: " + se.getErrorCode());

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


    }
    /**
     * generates the basic tables on / off
     * @author Andreas
     * */
    public void generateBasicMatrices(){
        System.out.println("generating basic matrices");
        try {
      HashMap<KeyList,Float> temp;


            switches = new ArrayList<Integer>();
            sensors = new ArrayList<Integer>();
```

```java

        ResultSet result = stmt.executeQuery("SELECT DISTINCT id FROM sensor_events");
        while(result.next()){
            sensors.add(result.getInt("id"));
        }
        result = stmt.executeQuery("SELECT DISTINCT id FROM switch_events");
        while(result.next()){
            switches.add(result.getInt("id"));
        }


        long lastevent = 0;
        int val,id;
        int i = 0;
        EventList eventlist = new EventList(false);
        long time;
        long start = System.currentTimeMillis();
        String type;
        KeyList keylist;
        on = new HashMap<KeyList, Float>();
        off = new HashMap<KeyList, Float>();
        count = new HashMap<KeyList, Integer>();
        HashMap<KeyList,Integer> denominator = new HashMap<KeyList,Integer>();
        System.out.println("fetching data from db");
        result = stmt.executeQuery("(SELECT id, timestamp, 'sensor' AS type, '0' AS status FROM sensor_events) UNION " +
            "(SELECT id, timestamp,'switch' AS type, status FROM switch_events) ORDER BY timestamp;");
        System.out.println(" iterating resultset");
        while(result.next()){
            i++;
            id = result.getInt("id");
            time = result.getTimestamp("timestamp").getTime();
            type = result.getString("type");
            //System.out.println("event : "+id+" type: "+type+" time : "+time);
            if(type.equals("sensor")){
                eventlist.add(new SensorEvent(id, time));
                keylist = new KeyList(eventlist);
                if (denominator.containsKey(keylist)){
                    denominator.put(keylist, denominator.get(keylist) + 1);
                } else{
                    denominator.put(keylist, 1);
                }
                lastevent = time;
            }
            else if(type.equals("switch")){
                temp = (result.getBoolean("status")) ? on : off;

                if(time > lastevent + Config.patternInterval){
                    eventlist = new EventList(false);
                    keylist = new KeyList(eventlist);
                    if (denominator.containsKey(keylist)){
                        denominator.put(keylist,denominator.get(keylist)+1);
                    }else{
                        denominator.put(keylist, 1);
                    }
                }
                keylist = new KeyList(eventlist);
                keylist.add(id);
```

```java

                    if(temp.containsKey(keylist)){
                        temp.put(keylist,temp.get(keylist)+1);
                    }
                    else{
                        temp.put(keylist,1f);
                    }

                }
            }
            KeyList ksub;
            long end = System.currentTimeMillis();
            long runtime = end-start;
            System.out.println("rows : "+i);
            System.out.println("runtime = "+runtime);
            for(KeyList k : on.keySet()){
                ksub = k.subList(0,k.size()-2);
                on.put(k,on.get(k) / denominator.get(ksub));
                count.put(ksub, denominator.get(ksub));
            }
            for(KeyList k : off.keySet()){
                ksub = k.subList(0,k.size()-2);
                off.put(k, off.get(k) / denominator.get(ksub));
                count.put(ksub, denominator.get(ksub));
            }
            System.out.printf("basic %d/%d (%d)\n", on.size(), off.size(), denominator.size());

        } catch (SQLException se){
            System.out.println("SQLException: " + se.getMessage());
            System.out.println("SQLState: " + se.getSQLState());
            System.out.println("VendorError: " + se.getErrorCode());

        }
    }

    public void generateZoneMatrices(){
        System.out.println("generating zone matrices");

    HashMap<KeyList,Float> temp,zoneOn,zoneOff;
    zoneOn = new HashMap<KeyList,Float>();
    zoneOff = new HashMap<KeyList,Float>();
    long lastevent = 0;
    int val,id;
    int i = 0;
    EventList eventlist = new EventList(true);
    long time;
    long start = System.currentTimeMillis();
    String type;
    KeyList keylist;
    HashMap<KeyList,Integer> denominator = new HashMap<KeyList,Integer>();
        try {
      System.out.println("fetching data from db");
            ResultSet result = stmt.executeQuery("(select id,timestamp,'sensor' AS type, '0' AS status from sensor_events) union (select id,timestamp,'switch' AS type,status from switch_events) order by timestamp;");
            System.out.println(" iterating resultset");
            while(result.next()){
                i++;
                id = result.getInt("id");
                time = result.getTimestamp("timestamp").getTime();
                type = result.getString("type");
```

```
                    //System.out.println("event : "+id+" type: "+type+" time : "+time);
                    if(type.equals("sensor")){
                        eventlist.add(new SensorEvent(id,time));
                        lastevent = time;
                        if (!eventlist.containsZoneEvent())
                            continue;

                        keylist = new KeyList(eventlist);
                        if (denominator.containsKey(keylist)) {
                            denominator.put(keylist,denominator.get(keylist)+1);
                        } else{
                            denominator.put(keylist,1);
                        }
                    }
                    else if (type.equals("switch")) {
                        temp = (result.getBoolean("status")) ? zoneOn : zoneOff;
//                      if(time > lastevent+Config.patternInterval){
//                          eventlist = new EventList(true);
//                          keylist = new KeyList(eventlist);
//                          if (denominator.containsKey(keylist)){
//                              denominator.put(keylist,denominator.get(keylist)+1);
//                          } else {
//                              denominator.put(keylist,1);
//                          }
//                      }
                if(eventlist.containsZoneEvent()){
                    keylist = new KeyList(eventlist);
                    keylist.add(id);
//                  System.out.println("keylist : "+keylist.toString());
                    if(temp.containsKey(keylist)){
                        temp.put(keylist,temp.get(keylist)+1);
                    }
                    else{
                        temp.put(keylist,1f);
                    }
                }
                    }
                }
                KeyList ksub;
                long end = System.currentTimeMillis();
                long runtime = end-start;
                System.out.println("rows : "+i);
                System.out.println("runtime = "+runtime);
                for(KeyList k : zoneOn.keySet()){
                    ksub = k.subList(0,k.size()-2);
                    zoneOn.put(k,zoneOn.get(k)/denominator.get(ksub));
                    count.put(ksub, denominator.get(ksub));
                }
                 for(KeyList k : zoneOff.keySet()){
                    ksub = k.subList(0,k.size()-2);
                    zoneOff.put(k,zoneOff.get(k)/denominator.get(ksub));
                    count.put(ksub, denominator.get(ksub));
                }

            }
            catch (SQLException se){
                System.out.println("SQLException: " + se.getMessage());
                System.out.println("SQLState: " + se.getSQLState());
                System.out.println("VendorError: " + se.getErrorCode());

            }
```

```
    for(KeyList k: zoneOn.keySet()){
      on.put(k,zoneOn.get(k));
    }
    for(KeyList k: zoneOff.keySet()){
      off.put(k,zoneOff.get(k));
    }
        System.out.printf("zone %d/%d (%d)\n", on.size(), off.size(),
            denominator.size());
    }
    public void printMatrices(){
        KeyList ksub;
        System.out.println();
    System.out.println("*********************************************");
    System.out.println("printing matrix on");
    System.out.println("*********************************************");
    for(KeyList k : on.keySet()){
            ksub = k.subList(0,k.size()-2);
            long seen = Math.round(count.get(ksub) * on.get(k));
            if (seen <= 1)
                continue;
            if (k.get(Config.patternLength) != 13)
                continue;

            System.out.printf("(%d/%d) ", seen, count.get(ksub));

            System.out.print("key: ");
      k.printValues();

      System.out.println("value: "+on.get(k));
    }
    System.out.println();
    System.out.println();
    System.out.println("*********************************************");
    System.out.println("printing matrix off");
    System.out.println("*********************************************");

    for(KeyList k : off.keySet()){
            ksub = k.subList(0,k.size()-2);
            long seen = Math.round(count.get(ksub) * off.get(k));
            if (seen <= 1)
                continue;
            if (k.get(Config.patternLength) != 13)
                continue;

            System.out.printf("(%d/%d) ", seen, count.get(ksub));

            System.out.print("key: ");
      k.printValues();

      System.out.println("value: "+off.get(k));
    }
    System.out.println();
    }

}
```

Listing A.14: DecisionMatrix.java

A.5.3 KeyList.java

```java
package core;
import java.util.ArrayList;
import events.*;
/**
 * @author Andreas
 * */
public class KeyList{
    private ArrayList<Integer> keys;
    public KeyList(){
        keys = new ArrayList<Integer>();
    }
    public KeyList(EventList elist){
        keys = new ArrayList<Integer>();
        for(Event e : elist.getPattern()){
            keys.add(e.getID());
        }
    }
    public int hashCode() {
        int hashcode=0;
       for(int i : keys){
            hashcode = hashcode*31 +i;
       }
       return hashcode;
    }
    public boolean equals(Object o) {
        try{
            KeyList a = (KeyList)o;
            if(this.size() != a.size()){
                return false;
            }
                for(int i=0;i<keys.size();i++){
                    if(this.get(i)!=a.get(i)){
                        return false;
                    }
                }
                return true;
        }
        catch(Exception e){
            return false;
        }
    }
    public void add(int i){
        keys.add(i);
    }
    public void add(int k, int i){
        keys.add(k,i);
    }
    public int get(int k){
        return keys.get(k);
    }
    public int size(){
        return keys.size();
    }
    public KeyList subList(int x, int y){
        KeyList k = new KeyList();
        for (int i = x;i<=y;i++){
            k.add(keys.get(i));
        }
        return k;
    }
    public void printValues(){
        for (int i : keys){
            System.out.print(ZoneEvent.getIDString(i) + " ");
        }
    }
```

```
    public ArrayList<Integer> getKeys(){
        return keys;
    }

    public boolean hasZoneEvent() {
        for (int i : keys){
            if (i >= 256)
                return true;
        }
        return false;
    }
  public String toString(){
    String returnstr = "";
    for (int i : keys){
      returnstr = returnstr+ZoneEvent.getIDString(i)+" ";
    }
    return returnstr;
  }
}
```

Listing A.15: KeyList.java

Appendix B

Testing

B.1 Source Listings

B.1.1 UnitTests.java

```java
package events;

import static org.junit.Assert.*;

import java.util.Arrays;

import org.junit.Before;
import org.junit.Test;

import config.Config;

public class UnitTests {

    EventList events;
    SensorEvent[] se;
    SwitchEvent[] sw;
    ZoneEvent z1;

    @Before
    public void setUp() throws Exception {
        events = new EventList(500, 10000, 7);
        se = new SensorEvent[]{new SensorEvent(1), new SensorEvent(2),
            new SensorEvent(3)};
        sw = new SwitchEvent[]{new SwitchEvent(11, true), new
            SwitchEvent(12, false)};
```

```java
            z1 = new ZoneEvent(0L, 20, 21);
    }

    /**
     * test that the single integer id for zone events are the no matter
          , no matter the order the ids are added to the zone event.
     */
    @Test
    public void zoneIdConsistency() {
        int actual, expected = new ZoneEvent(0L, 1, 2, 3).getID();

        ZoneEvent z = new ZoneEvent();
        z.addID(1);
        z.addID(2);
        z.addID(3);
        actual = z.getID();
        assertEquals(expected, actual);

        z = new ZoneEvent();
        z.addID(2);
        z.addID(3);
        z.addID(1);
        actual = z.getID();
        assertEquals(expected, actual);

        z = new ZoneEvent();
        z.addID(3);
        z.addID(1);
        z.addID(2);
        actual = z.getID();
        assertEquals(expected, actual);
    }

    /**
     * test the equals method for sensor events
     */
    @Test
    public void testEquals() {
        SensorEvent s1 = new SensorEvent(1, 123456789);
        SensorEvent s2 = new SensorEvent(1, 123456789);
        assertEquals(s1, s2);
        SensorEvent s3 = new SensorEvent(3, 123456789);
        assertTrue(!s1.equals(s3));
    }

    /**
     * basic get events test
     * the same sensor event 3 times, then one switch event
     */
    @Test
    public void testGetEvents() {
        events.add(se[0]);
        events.add(se[0]);
        events.add(se[0]);
        events.add(sw[0]);

        Event[] expected = {se[0], se[0], se[0], sw[0]};
        Event[] actual = events.getEvents();
        for (int i = 0; i < expected.length; i++) {
            assertEquals(expected[i], actual[i]);
        }
    }

    /**
     * tests the ordering of sensor events going into an eventlist
```

```java
     * adds 7 sensor events to event list,
     * ids are sequencial,
     * and timestamps are 1000ms appart.
     * verifies the ordering of the entire list, after each event is added.
     * also tests the getLastEvent method
     */
    @Test
    public void testEventOrdering() {
        Event expected, actual;
        Event[] e = new Event[7];
        for (int i = 0; i < 7; i++) {
            e[i] = new SensorEvent(i, 1000*i);
            events.add(e[i]);

            expected = e[i];
            actual = events.getLastEvent();
            assertEquals(expected, actual);

            for (int j = 0; j <= i; j++) {
                expected = e[j];
                actual = events.getEvents()[j];
                assertEquals(expected, actual);
            }
        }


    }

    /**
     * test getPattern, to make sure the array has fixed length,
     * independant of events in eventlist,
     * and that the array is properly prefixed with -1
     */
    @Test
    public void testGetPattern() {
        assertEquals(7, events.getPattern().length);
        for (Event actual : events.getPattern()) {
            assertEquals(-1, actual.getID());
        }
        events.add(se[0]);
        events.add(se[0]);
        events.add(se[0]);

        Event[] actuals = events.getPattern();
        for (int i = 0; i < Config.patternLength; i++) {
            if (i < 4)
                assertEquals(-1, actuals[i].getID());
            else
                assertEquals(se[0], actuals[i]);
        }

        //adds 5 more events, for a total of 8
        events.add(se[0]);
        events.add(se[0]);
        events.add(se[0]);
        events.add(se[0]);
        events.add(se[0]);

        for (Event actual : events.getPattern()) {
            assertEquals(se[0], actual);
        }
        assertEquals(7, events.getPattern().length);
    }

```

```
    /**
     * test of zone events:
     * 1 - eventlist is able to detect zone events, if zones are enabled
         .
     * 2 -  zone events are not produced, if zones are disabled.
     */
    @Test
    public void testZoneDetection() {

        se[0] = new SensorEvent(1, 123456781000L);
        se[1] = new SensorEvent(2, 123456781000L);
        se[2] = new SensorEvent(1, 123456789000L);

        events.add(se[0]);
        events.add(se[1]);
        events.add(se[2]);

        Event[] actuals = events.getEvents();

        assertTrue(actuals[0] instanceof ZoneEvent);
        assertTrue(actuals[0].compareID(se[0].getID()));
        assertTrue(actuals[0].compareID(se[1].getID()));
        assertEquals(se[2], actuals[1]);

        //repeats test without zone detection
        events = new EventList(0, 10000, 7);
        events.add(se[0]);
        events.add(se[1]);
        events.add(se[2]);

        actuals = events.getEvents();

        for (int i = 0; i < 3; i++) {
            assertEquals(se[i], actuals[i]);
        }

    }


    /**
     * tests the removal of events "pattern interval" older than the
         last event
     */
    @Test
    public void testPurgeOld() {
        se[0] = new SensorEvent(1, 0L);
        se[1] = new SensorEvent(2, 123456781000L);

        events.add(se[0]);
        events.add(se[1]);

        assertEquals(1, events.getEvents().length);

        SensorEvent expected = se[1];
        Event actual = events.getEvents()[0];
        assertEquals(expected, actual);
    }

    /**
     * tests that event list maintains the correct number of events,
     * using various pattern length configurations (2, 3 and 7)
     */
    @Test
    public void testPatternLength() {
        EventList e2 = new EventList(500, 10000, 2);
```

```java
        EventList e3 = new EventList(500, 10000, 3);
        EventList e7 = new EventList(500, 10000, 7);
        EventList[] es = new EventList[]{e2, e3, e7};

        int actual, expected = 0;

        //makes sure the length is initially zero
        for (EventList e : es) {
            actual = e.getEvents().length;
            assertEquals(expected, actual);
        }

        //adds an event to each list, and verifies the length to be 1
        expected = 1;
        for (EventList e : es) {
            e.add(se[0]);
            actual = e.getEvents().length;
            assertEquals(expected, actual);
        }

        //adds the event a 2nd time, and verifies the length to be 2
        expected = 2;
        for (EventList e : es) {
            e.add(se[0]);
            actual = e.getEvents().length;
            assertEquals(expected, actual);
        }

        //adds 8 more sensor events, so all lists are full
        for (EventList e : es) {
            for (int i = 0; i < 8; i++)
                e.add(se[0]);
        }

        //verifies that all lists are at their max capacity
        assertEquals(2, e2.getEvents().length);
        assertEquals(3, e3.getEvents().length);
        assertEquals(7, e7.getEvents().length);

}

}
```

Listing B.1: UnitTests.java

B.2 DecisionMatrix dumps

B.2.1 Pattern length 2, without zones

```
Loading Configurations
Database = jdbc:mysql://localhost/kiiib?user=KIIIB&password=42
pattern_interval = 10000
pattern_length = 2
use_zones = false
zone_interval = 500
probablility_threshold = 0.5
correlation_interval = 7000
```

```
correlation_correction = 0.1
default_on_time = 5000
punishment_timeout = 10000
debug = false
Trying to connect to the database
connection established
generating basic matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 1854
basic 88/75 (114)
switches
18
13
19
17
4
sensors
24
23
20
21
27
28
22
25
26
29
30

********************************************
printing matrix on
********************************************
(1/1150) key: 28 28 18 value: 8.6956524E-4
(1/935) key: 27 28 19 value: 0.0010695187
(9/935) key: 27 28 18 value: 0.009625669
(1/161) key: 20 27 19 value: 0.0062111802
(4/666) key: 29 29 19 value: 0.006006006
(1/161) key: 20 27 17 value: 0.0062111802
(1/1627) key: 22 23 13 value: 6.1462814E-4
(1/161) key: 20 27 18 value: 0.0062111802
(1/289) key: 21 23 13 value: 0.0034602077
(1/105) key: 25 20 19 value: 0.00952381
(3/1209) key: 24 23 13 value: 0.0024813896
(1/42) key: 23 26 17 value: 0.023809524
(1/53) key: 27 25 17 value: 0.018867925
(1/170) key: 21 27 13 value: 0.005882353
(2/170) key: 21 27 17 value: 0.011764706
(2/720) key: 20 23 13 value: 0.0027777778
(1/126) key: -1 21 19 value: 0.007936508
(2/170) key: 21 27 19 value: 0.011764706
(1/41) key: 20 26 19 value: 0.024390243
(3/231) key: 25 21 17 value: 0.012987013
(2/106) key: 26 21 19 value: 0.018867925
(1/666) key: 26 25 17 value: 0.0015015015
(2/231) key: 25 21 19 value: 0.008658009
(1/41) key: 20 26 17 value: 0.024390243
(1/231) key: 25 21 18 value: 0.0043290043
(3/811) key: 25 25 17 value: 0.003699137
(1/811) key: 25 25 19 value: 0.0012330456
(3/106) key: 26 21 17 value: 0.028301887
(2/126) key: -1 21 4 value: 0.015873017
(1/187) key: 28 21 4 value: 0.0053475937
(1/230) key: 24 21 19 value: 0.004347826
```

```
(1/371) key: 21 21 17 value: 0.0026954177
(1/334) key: 20 20 4 value: 0.002994012
(1/371) key: 21 21 19 value: 0.0026954177
(2/722) key: 25 26 17 value: 0.002770083
(1/722) key: 25 26 19 value: 0.0013850415
(1/180) key: -1 20 19 value: 0.0055555557
(1/146) key: 21 28 19 value: 0.006849315
(1/146) key: 21 28 18 value: 0.006849315
(4/363) key: 23 21 19 value: 0.011019284
(1/58) key: 23 25 17 value: 0.01724138
(1/134) key: 27 21 19 value: 0.0074626864
(1/363) key: 23 21 18 value: 0.002754821
(1/1161) key: -1 28 19 value: 8.6132647E-4
(3/107) key: 29 21 19 value: 0.028037382
(5/334) key: 20 20 19 value: 0.0149700595
(4/215) key: 21 25 17 value: 0.018604651
(1/363) key: 23 21 13 value: 0.002754821
(6/136) key: 20 25 17 value: 0.04411765
(1/180) key: -1 20 4 value: 0.0055555557
(1/73) key: 20 29 19 value: 0.01369863
(1/81) key: 24 25 17 value: 0.012345679
(3/991) key: 21 20 17 value: 0.0030272452
(2/1230) key: 23 24 4 value: 0.0016260162
(1/991) key: 21 20 18 value: 0.0010090817
(2/584) key: 24 20 19 value: 0.0034246575
(2/584) key: 24 20 13 value: 0.0034246575
(4/875) key: 20 21 4 value: 0.0045714285
(20/991) key: 21 20 19 value: 0.020181635
(1/28) key: 22 29 19 value: 0.035714287
(5/19) key: -1 -1 18 value: 0.2631579
(5/593) key: 23 20 13 value: 0.008431703
(4/875) key: 20 21 13 value: 0.0045714285
(3/870) key: 28 27 18 value: 0.0034482758
(1/88) key: 22 20 19 value: 0.011363637
(1/296) key: 21 24 19 value: 0.0033783785
(3/100) key: 21 29 19 value: 0.03
(1/593) key: 23 20 18 value: 0.0016863407
(1/1044) key: 27 27 17 value: 9.578544E-4
(2/103) key: 27 20 19 value: 0.019417476
(1/1044) key: 27 27 18 value: 9.578544E-4
(2/593) key: 23 20 19 value: 0.0033726813
(1/1044) key: 27 27 13 value: 9.578544E-4
(1/88) key: 22 20 13 value: 0.011363637
(1/571) key: -1 27 18 value: 0.0017513135
(1/529) key: 20 24 13 value: 0.0018903592
(1/1230) key: 23 24 18 value: 8.130081E-4
(1/149) key: 28 20 19 value: 0.0067114094
(1/875) key: 20 21 17 value: 0.0011428571
(1/875) key: 20 21 18 value: 0.0011428571
(17/875) key: 20 21 19 value: 0.019428572
(2/991) key: 21 20 4 value: 0.0020181634
(1/529) key: 20 24 19 value: 0.0018903592
(1/19) key: -1 -1 13 value: 0.05263158
(1/529) key: 20 24 4 value: 0.0018903592
(2/116) key: 20 28 18 value: 0.01724138
(1/584) key: 24 20 4 value: 0.0017123288
(2/991) key: 21 20 13 value: 0.0020181634


*******************************************
printing matrix off
*******************************************
(1/1209) key: 24 23 18 value: 8.271299E-4
(1/1150) key: 28 28 18 value: 8.6956524E-4
(2/68) key: 24 27 18 value: 0.029411765
```

```
(1/935) key: 27 28 19 value: 0.0010695187
(2/935) key: 27 28 18 value: 0.0021390375
(1/58) key: 28 24 13 value: 0.01724138
(3/161) key: 20 27 19 value: 0.01863354
(2/666) key: 29 29 19 value: 0.003003003
(1/65) key: 27 24 4 value: 0.015384615
(1/105) key: 25 20 17 value: 0.00952381
(3/1209) key: 24 23 13 value: 0.0024813896
(1/197) key: -1 26 17 value: 0.005076142
(1/720) key: 20 23 4 value: 0.0013888889
(3/720) key: 20 23 13 value: 0.004166667
(1/720) key: 20 23 18 value: 0.0013888889
(1/28) key: 28 22 18 value: 0.035714287
(1/1842) key: 23 22 17 value: 5.428882E-4
(1/666) key: 26 25 17 value: 0.0015015015
(1/811) key: 25 25 17 value: 0.0012330456
(1/811) key: 25 25 19 value: 0.0012330456
(1/106) key: 26 21 17 value: 0.009433962
(1/246) key: -1 25 17 value: 0.0040650405
(1/371) key: 21 21 18 value: 0.0026954177
(1/230) key: 24 21 18 value: 0.004347826
(1/371) key: 21 21 19 value: 0.0026954177
(2/334) key: 20 20 4 value: 0.005988024
(1/28) key: 22 28 18 value: 0.035714287
(2/722) key: 25 26 17 value: 0.002770083
(1/180) key: -1 20 19 value: 0.0055555557
(1/363) key: 23 21 19 value: 0.002754821
(1/58) key: 23 25 17 value: 0.01724138
(1/134) key: 27 21 19 value: 0.0074626864
(1/230) key: 24 21 4 value: 0.004347826
(3/1161) key: -1 28 18 value: 0.0025839794
(2/821) key: 26 26 17 value: 0.0024360537
(3/107) key: 29 21 19 value: 0.028037382
(1/363) key: 23 21 13 value: 0.002754821
(1/371) key: 21 21 4 value: 0.0026954177
(4/334) key: 20 20 19 value: 0.011976048
(2/136) key: 20 25 17 value: 0.014705882
(2/5968) key: -1 24 13 value: 3.3512065E-4
(3/363) key: 23 21 4 value: 0.008264462
(1/991) key: 21 20 17 value: 0.0010090817
(3/991) key: 21 20 18 value: 0.0030272452
(1/62) key: 27 23 19 value: 0.016129032
(2/584) key: 24 20 13 value: 0.0034246575
(12/991) key: 21 20 19 value: 0.012108981
(3/875) key: 20 21 4 value: 0.0034285714
(3/19) key: -1 -1 19 value: 0.15789473
(2/593) key: 23 20 13 value: 0.0033726813
(9/19) key: -1 -1 18 value: 0.47368422
(2/875) key: 20 21 13 value: 0.0022857143
(4/870) key: 28 27 18 value: 0.004597701
(1/296) key: 21 24 19 value: 0.0033783785
(1/103) key: 27 20 19 value: 0.009708738
(2/1044) key: 27 27 18 value: 0.0019157088
(1/1044) key: 27 27 13 value: 9.578544E-4
(1/571) key: -1 27 18 value: 0.0017513135
(1/41) key: 29 20 19 value: 0.024390243
(2/1230) key: 23 24 17 value: 0.0016260162
(1/149) key: 28 20 19 value: 0.0067114094
(1/1230) key: 23 24 18 value: 8.130081E-4
(1/875) key: 20 21 17 value: 0.0011428571
(4/2575) key: -1 23 18 value: 0.0015533981
(2/870) key: 28 27 13 value: 0.0022988506
(13/875) key: 20 21 19 value: 0.014857143
(3/991) key: 21 20 4 value: 0.0030272452
(1/529) key: 20 24 19 value: 0.0018903592
```

```
(1/19) key: -1 -1 13 value: 0.05263158
(1/116) key: 20 28 17 value: 0.00862069
(1/116) key: 20 28 19 value: 0.00862069
(1/296) key: 21 24 4 value: 0.0033783785
(2/116) key: 20 28 18 value: 0.01724138
(2/4375) key: 24 24 18 value: 4.5714286E-4
(2/584) key: 24 20 4 value: 0.0034246575
```

Listing B.2: EventList.java

B.2.2 Pattern length 2, with zones

```
Loading Configurations
Database = jdbc:mysql://localhost/kiiib?user=KIIIB&password=42
pattern_interval = 10000
pattern_length = 2
use_zones = true
zone_interval = 500
probablility_threshold = 0.5
correlation_interval = 7000
correlation_correction = 0.1
default_on_time = 5000
punishment_timeout = 10000
debug = false
Trying to connect to the database
connection established
generating basic matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 1662
basic 88/75 (114)
generating zone matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 680
zone 144/119 (1173)
switches
18
13
19
17
4
sensors
24
23
20
21
27
28
22
25
26
29
30

******************************************
printing matrix on
******************************************
```

```
(1/4) key: [20,21] [21,29] 19 value: 0.25
(2/2) key: [21,24] [23,24] 4 value: 1.0
(1/9) key: 28 [20,21,27] 19 value: 0.11111111
(1/36) key: 23 [20,21] 4 value: 0.027777778
(1/161) key: 20 27 19 value: 0.0062111802
(1/169) key: [20,23] 21 19 value: 0.00591716
(1/26) key: [20,21] 28 18 value: 0.03846154
(1/1) key: 28 [20,27,28] 19 value: 1.0
(1/20) key: [27,28] [20,21] 19 value: 0.05
(1/161) key: 20 27 17 value: 0.0062111802
(1/161) key: 20 27 18 value: 0.0062111802
(1/289) key: 21 23 13 value: 0.0034602077
(1/2) key: 25 [21,27] 17 value: 0.5
(2/36) key: 23 [20,21] 19 value: 0.055555556
(2/97) key: [20,21] 21 19 value: 0.020618556
(1/20) key: [25,26] 21 19 value: 0.05
(1/53) key: 27 25 17 value: 0.018867925
(1/43) key: 21 [20,23] 13 value: 0.023255814
(1/4) key: [20,25] 21 18 value: 0.25
(1/126) key: -1 21 19 value: 0.007936508
(1/1) key: 21 [24,25] 17 value: 1.0
(1/40) key: [20,23] 20 18 value: 0.025
(1/41) key: 20 26 19 value: 0.024390243
(1/666) key: 26 25 17 value: 0.0015015015
(1/41) key: 20 26 17 value: 0.024390243
(3/811) key: 25 25 17 value: 0.003699137
(1/73) key: [20,21] 20 19 value: 0.01369863
(1/811) key: 25 25 19 value: 0.0012330456
(1/73) key: [20,21] 20 17 value: 0.01369863
(2/126) key: -1 21 4 value: 0.015873017
(1/10) key: [20,27] 21 19 value: 0.1
(3/111) key: 28 [20,21] 19 value: 0.027027028
(1/26) key: [20,21] [20,21] 13 value: 0.03846154
(1/180) key: -1 20 19 value: 0.0055555557
(1/1) key: [22,29] 21 19 value: 1.0
(1/44) key: 27 [20,21] 19 value: 0.022727273
(1/58) key: 23 25 17 value: 0.01724138
(2/111) key: 28 [20,21] 13 value: 0.018018018
(1/134) key: 27 21 19 value: 0.0074626864
(1/8) key: [20,21] 26 19 value: 0.125
(1/1161) key: -1 28 19 value: 8.6132647E-4
(3/107) key: 29 21 19 value: 0.028037382
(4/215) key: 21 25 17 value: 0.018604651
(1/57) key: -1 [20,23] 13 value: 0.01754386
(6/136) key: 20 25 17 value: 0.04411765
(1/95) key: [27,28] 28 18 value: 0.010526316
(1/180) key: -1 20 4 value: 0.0055555557
(1/81) key: 24 25 17 value: 0.012345679
(1/5) key: 27 [21,27] 13 value: 0.2
(1/28) key: 21 [20,24] 19 value: 0.035714287
(2/1230) key: 23 24 4 value: 0.0016260162
(1/39) key: 20 [20,21] 4 value: 0.025641026
(1/3) key: [20,21] [20,21,24] 19 value: 0.33333334
(1/4) key: [21,23] 25 17 value: 0.25
(3/870) key: 28 27 18 value: 0.0034482758
(1/296) key: 21 24 19 value: 0.0033783785
(1/6) key: 21 [21,27] 17 value: 0.16666667
(1/96) key: 27 [27,28] 18 value: 0.010416667
(2/103) key: 27 20 19 value: 0.019417476
(1/571) key: -1 27 18 value: 0.0017513135
(1/529) key: 20 24 13 value: 0.0018903592
(1/2) key: 25 [20,21,25] 19 value: 0.5
(1/39) key: 20 [20,21] 19 value: 0.025641026
(1/1230) key: 23 24 18 value: 8.130081E-4
(1/529) key: 20 24 19 value: 0.0018903592
```

```
(1/529) key: 20 24 4 value: 0.0018903592
(1/33) key: -1 [20,21] 19 value: 0.030303031
(1/1150) key: 28 28 18 value: 8.6956524E-4
(1/33) key: -1 [20,21] 17 value: 0.030303031
(1/935) key: 27 28 19 value: 0.0010695187
(1/34) key: [20,21] 25 17 value: 0.029411765
(9/935) key: 27 28 18 value: 0.009625669
(1/5) key: 22 [20,21] 19 value: 0.2
(4/666) key: 29 29 19 value: 0.006006006
(1/1627) key: 22 23 13 value: 6.1462814E-4
(1/26) key: 21 [20,27] 19 value: 0.03846154
(1/105) key: 25 20 19 value: 0.00952381
(1/26) key: 21 [20,27] 17 value: 0.03846154
(3/1209) key: 24 23 13 value: 0.0024813896
(1/2) key: [25,27] 27 17 value: 0.5
(1/8) key: [20,21] [21,25] 17 value: 0.125
(1/42) key: 23 26 17 value: 0.023809524
(1/8) key: [21,28] 20 18 value: 0.125
(1/170) key: 21 27 13 value: 0.005882353
(2/170) key: 21 27 17 value: 0.011764706
(2/720) key: 20 23 13 value: 0.0027777778
(2/170) key: 21 27 19 value: 0.011764706
(1/26) key: [20,23,24] 21 18 value: 0.03846154
(1/26) key: [20,23,24] 21 19 value: 0.03846154
(3/231) key: 25 21 17 value: 0.012987013
(1/16) key: 21 [21,25] 17 value: 0.0625
(2/106) key: 26 21 19 value: 0.018867925
(2/231) key: 25 21 19 value: 0.008658009
(1/231) key: 25 21 18 value: 0.0043290043
(3/106) key: 26 21 17 value: 0.028301887
(1/56) key: 23 [20,24] 13 value: 0.017857144
(1/14) key: 28 [21,27] 19 value: 0.071428575
(1/187) key: 28 21 4 value: 0.0053475937
(1/371) key: 21 21 17 value: 0.0026954177
(1/230) key: 24 21 19 value: 0.004347826
(1/371) key: 21 21 19 value: 0.0026954177
(1/334) key: 20 20 4 value: 0.002994012
(1/17) key: [21,27] 28 19 value: 0.05882353
(2/722) key: 25 26 17 value: 0.002770083
(1/7) key: [21,24] 25 17 value: 0.14285715
(1/722) key: 25 26 19 value: 0.0013850415
(1/146) key: 21 28 19 value: 0.006849315
(1/92) key: [20,21] 23 13 value: 0.010869565
(4/363) key: 23 21 19 value: 0.011019284
(1/146) key: 21 28 18 value: 0.006849315
(1/363) key: 23 21 18 value: 0.002754821
(1/363) key: 23 21 13 value: 0.002754821
(5/334) key: 20 20 19 value: 0.0149700595
(1/3) key: 21 [20,29] 19 value: 0.33333334
(1/73) key: 20 29 19 value: 0.01369863
(1/16) key: [21,27] 20 19 value: 0.0625
(3/991) key: 21 20 17 value: 0.0030272452
(1/62) key: 21 [20,21] 4 value: 0.016129032
(1/991) key: 21 20 18 value: 0.0010090817
(2/584) key: 24 20 19 value: 0.0034246575
(2/584) key: 24 20 13 value: 0.0034246575
(20/991) key: 21 20 19 value: 0.020181635
(4/875) key: 20 21 4 value: 0.0045714285
(1/28) key: 22 29 19 value: 0.035714287
(4/875) key: 20 21 13 value: 0.0045714285
(5/593) key: 23 20 13 value: 0.008431703
(5/19) key: -1 -1 18 value: 0.2631579
(1/88) key: 22 20 19 value: 0.011363637
(1/593) key: 23 20 18 value: 0.0016863407
(3/100) key: 21 29 19 value: 0.03
```

```
(1/1044) key: 27 27 17 value: 9.578544E-4
(1/1044) key: 27 27 18 value: 9.578544E-4
(2/593) key: 23 20 19 value: 0.0033726813
(1/88) key: 22 20 13 value: 0.011363637
(1/1044) key: 27 27 13 value: 9.578544E-4
(1/149) key: 28 20 19 value: 0.0067114094
(1/875) key: 20 21 17 value: 0.0011428571
(1/875) key: 20 21 18 value: 0.0011428571
(2/991) key: 21 20 4 value: 0.0020181634
(17/875) key: 20 21 19 value: 0.019428572
(1/19) key: -1 -1 13 value: 0.05263158
(2/116) key: 20 28 18 value: 0.01724138
(1/584) key: 24 20 4 value: 0.0017123288
(2/991) key: 21 20 13 value: 0.0020181634


******************************************
printing matrix off
******************************************
(1/169) key: [20,23] 21 4 value: 0.00591716
(2/68) key: 24 27 18 value: 0.029411765
(1/131) key: [23,24] 24 18 value: 0.007633588
(1/58) key: 28 24 13 value: 0.01724138
(3/161) key: 20 27 19 value: 0.01863354
(1/20) key: [20,27] 28 18 value: 0.05
(1/65) key: 27 24 4 value: 0.015384615
(1/20) key: [27,28] [20,21] 13 value: 0.05
(2/97) key: [20,21] 21 19 value: 0.020618556
(1/3) key: 26 [23,24] 17 value: 0.33333334
(1/8) key: [23,24] 25 17 value: 0.125
(1/2) key: 21 [20,21,23] 19 value: 0.5
(1/1) key: [20,21,28] [24,27] 18 value: 1.0
(2/5) key: [20,24] [21,23] 4 value: 0.4
(1/1842) key: 23 22 17 value: 5.428882E-4
(1/666) key: 26 25 17 value: 0.0015015015
(1/226) key: 23 [22,23] 17 value: 0.0044247787
(1/811) key: 25 25 17 value: 0.0012330456
(1/73) key: [20,21] 20 19 value: 0.01369863
(1/811) key: 25 25 19 value: 0.0012330456
(1/111) key: 28 [20,21] 19 value: 0.009009009
(1/111) key: 28 [20,21] 18 value: 0.009009009
(1/180) key: -1 20 19 value: 0.0055555557
(1/58) key: 23 25 17 value: 0.01724138
(2/44) key: 27 [20,21] 19 value: 0.045454547
(1/134) key: 27 21 19 value: 0.0074626864
(3/1161) key: -1 28 18 value: 0.0025839794
(2/821) key: 26 26 17 value: 0.0024360537
(3/107) key: 29 21 19 value: 0.028037382
(2/8) key: 29 [20,21] 19 value: 0.25
(2/136) key: 20 25 17 value: 0.014705882
(2/57) key: -1 [20,23] 13 value: 0.03508772
(2/95) key: [27,28] 28 18 value: 0.021052632
(1/57) key: -1 [20,23] 18 value: 0.01754386
(1/1) key: 21 [22,28] 18 value: 1.0
(1/62) key: 27 23 19 value: 0.016129032
(1/29) key: [20,23,24] 24 18 value: 0.03448276
(1/1) key: [20,21,26] [20,23,24] 19 value: 1.0
(1/19) key: [25,26] [25,26] 17 value: 0.05263158
(4/870) key: 28 27 18 value: 0.004597701
(1/296) key: 21 24 19 value: 0.0033783785
(2/96) key: 27 [27,28] 18 value: 0.020833334
(1/103) key: 27 20 19 value: 0.009708738
(1/571) key: -1 27 18 value: 0.0017513135
(1/41) key: 29 20 19 value: 0.024390243
(2/1230) key: 23 24 17 value: 0.0016260162
```

```
(1/39) key: 20 [20,21] 19 value: 0.025641026
(1/1230) key: 23 24 18 value: 8.130081E-4
(2/870) key: 28 27 13 value: 0.0022988506
(1/529) key: 20 24 19 value: 0.0018903592
(1/296) key: 21 24 4 value: 0.0033783785
(1/1) key: [20,23] [20,28] 19 value: 1.0
(2/4375) key: 24 24 18 value: 4.5714286E-4
(1/1209) key: 24 23 18 value: 8.271299E-4
(2/33) key: -1 [20,21] 19 value: 0.060606062
(1/27) key: 24 [20,21] 19 value: 0.037037037
(1/1150) key: 28 28 18 value: 8.6956524E-4
(1/935) key: 27 28 19 value: 0.0010695187
(1/34) key: [20,21] 25 17 value: 0.029411765
(2/935) key: 27 28 18 value: 0.0021390375
(2/666) key: 29 29 19 value: 0.003003003
(1/1) key: [23,28] 24 13 value: 1.0
(1/105) key: 25 20 17 value: 0.00952381
(1/6) key: [25,26] [20,21] 19 value: 0.16666667
(1/26) key: 21 [20,27] 19 value: 0.03846154
(3/1209) key: 24 23 13 value: 0.0024813896
(1/197) key: -1 26 17 value: 0.005076142
(1/720) key: 20 23 4 value: 0.0013888889
(1/125) key: [27,28] 27 13 value: 0.008
(1/79) key: [20,21] 24 4 value: 0.012658228
(3/720) key: 20 23 13 value: 0.004166667
(1/720) key: 20 23 18 value: 0.0013888889
(1/1) key: [23,26] [25,26] 17 value: 1.0
(1/28) key: 28 22 18 value: 0.035714287
(1/2) key: [20,24] [21,24] 19 value: 0.5
(1/106) key: 26 21 17 value: 0.009433962
(1/246) key: -1 25 17 value: 0.0040650405
(1/230) key: 24 21 18 value: 0.004347826
(1/371) key: 21 21 18 value: 0.0026954177
(2/334) key: 20 20 4 value: 0.005988024
(1/371) key: 21 21 19 value: 0.0026954177
(1/84) key: [25,26] 26 17 value: 0.011904762
(2/722) key: 25 26 17 value: 0.002770083
(1/28) key: 22 28 18 value: 0.035714287
(1/24) key: [23,24] 21 4 value: 0.041666668
(1/2) key: [23,28] [20,21] 19 value: 0.5
(1/363) key: 23 21 19 value: 0.002754821
(1/230) key: 24 21 4 value: 0.004347826
(4/334) key: 20 20 19 value: 0.011976048
(1/371) key: 21 21 4 value: 0.0026954177
(1/363) key: 23 21 13 value: 0.002754821
(2/5968) key: -1 24 13 value: 3.3512065E-4
(3/363) key: 23 21 4 value: 0.008264462
(1/181) key: 24 [20,23] 13 value: 0.005524862
(1/991) key: 21 20 17 value: 0.0010090817
(3/991) key: 21 20 18 value: 0.0030272452
(2/584) key: 24 20 13 value: 0.0034246575
(3/875) key: 20 21 4 value: 0.0034285714
(12/991) key: 21 20 19 value: 0.012108981
(3/19) key: -1 -1 19 value: 0.15789473
(2/875) key: 20 21 13 value: 0.0022857143
(9/19) key: -1 -1 18 value: 0.47368422
(2/593) key: 23 20 13 value: 0.0033726813
(1/55) key: [20,24] 23 18 value: 0.018181818
(2/1044) key: 27 27 18 value: 0.0019157088
(1/1044) key: 27 27 13 value: 9.578544E-4
(1/149) key: 28 20 19 value: 0.0067114094
(1/875) key: 20 21 17 value: 0.0011428571
(4/2575) key: -1 23 18 value: 0.0015533981
(3/991) key: 21 20 4 value: 0.0030272452
(13/875) key: 20 21 19 value: 0.014857143
```

```
(4/62) key: 21 [20,21] 19 value: 0.06451613
(1/19) key: −1 −1 13 value: 0.05263158
(1/116) key: 20 28 17 value: 0.00862069
(1/5) key: 21 [20,28] 18 value: 0.2
(1/116) key: 20 28 19 value: 0.00862069
(2/116) key: 20 28 18 value: 0.01724138
(2/584) key: 24 20 4 value: 0.0034246575
(1/1) key: 26 [21,29] 19 value: 1.0
```

Listing B.3: EventList.java

B.2.3 Pattern length 3, without zones

```
Loading Configurations
Database = jdbc:mysql://localhost/kiiib?user=KIIIB&password=42
pattern_interval = 10000
pattern_length = 3
use_zones = false
zone_interval = 500
probablility_threshold = 0.5
correlation_interval = 7000
correlation_correction = 0.1
default_on_time = 5000
punishment_timeout = 10000
debug = false
Trying to connect to the database
connection established
generating basic matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 1666
basic 142/112 (914)
switches
18
13
19
17
4
sensors
24
23
20
21
27
28
22
25
26
29
30

********************************************
printing matrix on
********************************************
(1/10) key: 21 27 25 17 value: 0.1
(6/255) key: 20 21 20 19 value: 0.023529412
(1/180) key: −1 −1 20 4 value: 0.0055555557
(1/30) key: 20 21 29 19 value: 0.033333335
(1/10) key: 21 21 27 17 value: 0.1
```

```
(1/180) key: -1 -1 20 19 value: 0.0055555557
(1/47) key: 23 20 20 4 value: 0.021276595
(1/72) key: 27 28 21 4 value: 0.013888889
(1/12) key: 20 22 20 19 value: 0.083333336
(1/150) key: 23 24 20 13 value: 0.006666667
(1/1161) key: -1 -1 28 19 value: 8.6132647E-4
(1/204) key: 26 25 25 17 value: 0.004901961
(1/397) key: 25 25 25 17 value: 0.0025188916
(1/78) key: 20 24 21 19 value: 0.012820513
(2/202) key: 20 23 21 19 value: 0.00990099
(1/56) key: -1 23 20 19 value: 0.017857144
(3/68) key: 20 20 21 19 value: 0.04411765
(1/14) key: 26 23 26 17 value: 0.071428575
(1/56) key: -1 23 20 13 value: 0.017857144
(1/72) key: 27 28 20 19 value: 0.013888889
(1/382) key: 24 23 24 18 value: 0.002617801
(1/202) key: 20 23 21 13 value: 0.004950495
(1/93) key: 24 20 21 18 value: 0.010752688
(2/23) key: 26 26 21 17 value: 0.08695652
(1/167) key: 28 28 27 18 value: 0.005988024
(1/126) key: -1 -1 21 19 value: 0.007936508
(1/180) key: 23 20 21 4 value: 0.0055555557
(1/1) key: 22 29 21 19 value: 1.0
(1/438) key: 29 29 29 19 value: 0.002283105
(2/382) key: 24 23 24 4 value: 0.005235602
(1/6) key: 21 24 25 17 value: 0.16666667
(1/58) key: 29 29 21 19 value: 0.01724138
(1/81) key: 20 24 20 19 value: 0.012345679
(1/23) key: 23 22 20 13 value: 0.04347826
(1/6) key: 21 23 25 17 value: 0.16666667
(1/160) key: -1 24 23 13 value: 0.00625
(1/139) key: 20 23 20 19 value: 0.007194245
(1/31) key: 21 25 25 19 value: 0.032258064
(1/139) key: 20 23 20 18 value: 0.007194245
(2/126) key: -1 -1 21 4 value: 0.015873017
(2/180) key: 23 20 21 19 value: 0.011111111
(2/22) key: 20 29 29 19 value: 0.09090909
(5/19) key: -1 -1 -1 18 value: 0.2631579
(2/30) key: 21 21 25 17 value: 0.06666667
(1/68) key: 20 21 27 19 value: 0.014705882
(4/73) key: 21 20 25 17 value: 0.05479452
(1/7) key: 22 20 20 19 value: 0.14285715
(1/17) key: 20 28 28 18 value: 0.05882353
(2/91) key: 21 25 26 17 value: 0.021978023
(1/9) key: 27 21 27 13 value: 0.11111111
(1/19) key: -1 -1 -1 13 value: 0.05263158
(2/83) key: 28 20 21 19 value: 0.024096385
(2/47) key: 25 26 21 19 value: 0.04255319
(2/83) key: 28 20 21 13 value: 0.024096385
(1/47) key: 25 26 21 17 value: 0.021276595
(1/114) key: 21 20 24 19 value: 0.00877193
(3/61) key: -1 21 20 19 value: 0.04918033
(1/15) key: 22 21 20 19 value: 0.06666667
(1/2) key: 29 28 27 18 value: 0.5
(1/61) key: -1 21 20 17 value: 0.016393442
(1/222) key: 24 20 23 13 value: 0.0045045046
(1/6) key: 25 21 27 17 value: 0.16666667
(1/46) key: 20 20 24 4 value: 0.02173913
(1/37) key: 27 20 21 19 value: 0.027027028
(1/111) key: 21 24 23 13 value: 0.009009009
(1/571) key: -1 -1 27 18 value: 0.0017513135
(1/255) key: 20 21 20 4 value: 0.003921569
(1/46) key: 20 20 24 13 value: 0.02173913
(1/110) key: 23 20 23 13 value: 0.009090909
(2/255) key: 20 21 20 17 value: 0.007843138
```

```
(1/14) key: 29 22 29 19 value: 0.071428575
(1/255) key: 20 21 20 13 value: 0.003921569
(1/54) key: -1 20 21 4 value: 0.018518519
(1/62) key: 20 21 25 17 value: 0.016129032
(1/54) key: -1 20 21 13 value: 0.018518519
(1/59) key: 23 21 21 17 value: 0.016949153
(2/68) key: 21 23 20 13 value: 0.029411765
(1/59) key: 23 21 21 19 value: 0.016949153
(1/5) key: 29 20 21 19 value: 0.2
(1/35) key: 23 21 28 19 value: 0.028571429
(1/35) key: 23 21 28 18 value: 0.028571429
(1/21) key: 21 29 21 19 value: 0.04761905
(2/15) key: 23 23 21 19 value: 0.13333334
(1/54) key: -1 20 21 19 value: 0.018518519
(1/69) key: 26 25 21 19 value: 0.014492754
(1/60) key: 20 25 26 19 value: 0.016666668
(1/18) key: 21 20 26 19 value: 0.055555556
(1/11) key: 25 27 27 17 value: 0.09090909
(1/13) key: 28 21 27 19 value: 0.07692308
(2/235) key: 24 23 20 13 value: 0.008510638
(1/327) key: 28 27 28 18 value: 0.003058104
(1/6) key: 27 20 25 17 value: 0.16666667
(1/255) key: 26 26 25 17 value: 0.003921569
(1/117) key: 25 21 20 19 value: 0.008547009
(5/269) key: 27 27 28 18 value: 0.01858736
(1/65) key: 21 24 20 19 value: 0.015384615
(1/112) key: 23 21 20 13 value: 0.008928572
(1/81) key: 21 27 28 19 value: 0.012345679
(2/81) key: 21 27 28 18 value: 0.024691358
(2/112) key: 23 21 20 19 value: 0.017857144
(1/10) key: 28 20 28 18 value: 0.1
(2/52) key: 25 25 21 17 value: 0.03846154
(1/73) key: 20 27 28 18 value: 0.01369863
(1/30) key: 25 20 21 19 value: 0.033333335
(1/81) key: 20 24 20 4 value: 0.012345679
(1/12) key: 24 21 25 17 value: 0.083333336
(1/92) key: 21 20 27 19 value: 0.010869565
(1/7) key: 27 29 29 19 value: 0.14285715
(1/92) key: 21 20 27 17 value: 0.010869565
(1/22) key: 20 25 25 17 value: 0.045454547
(1/81) key: 20 24 20 13 value: 0.012345679
(1/18) key: 23 20 25 17 value: 0.055555556
(1/261) key: 24 24 23 13 value: 0.0038314175
(1/289) key: 28 27 27 18 value: 0.0034602077
(2/49) key: 29 21 20 19 value: 0.040816326
(1/555) key: 27 27 27 13 value: 0.0018018018
(3/68) key: 27 21 20 19 value: 0.04411765
(1/104) key: 28 21 20 18 value: 0.009615385
(1/296) key: 21 20 21 13 value: 0.0033783785
(1/11) key: 20 25 20 19 value: 0.09090909
(2/104) key: 28 21 20 19 value: 0.01923077
(1/34) key: 21 25 21 17 value: 0.029411765
(1/225) key: 22 22 23 13 value: 0.0044444446
(1/128) key: 20 21 23 13 value: 0.0078125
(1/296) key: 21 20 21 17 value: 0.0033783785
(6/296) key: 21 20 21 19 value: 0.02027027
(1/54) key: 21 20 28 18 value: 0.018518519
(1/56) key: 21 28 27 18 value: 0.017857144
(1/3) key: 23 20 26 17 value: 0.33333334
(2/296) key: 21 20 21 4 value: 0.006756757
(1/48) key: 28 27 21 19 value: 0.020833334
(1/129) key: 20 21 24 19 value: 0.007751938
(1/79) key: 20 20 20 19 value: 0.012658228
(1/27) key: 20 25 21 18 value: 0.037037037
(3/97) key: 21 20 20 19 value: 0.030927835
```

```
(1/27) key: 20 25 21 19 value: 0.037037037
(1/9) key: 25 20 27 18 value: 0.11111111
(1/90) key: 21 21 20 4 value: 0.011111111
(2/28) key: 21 27 20 19 value: 0.071428575
(1/34) key: 21 20 29 19 value: 0.029411765
(2/21) key: 21 21 29 19 value: 0.0952381
(1/62) key: 24 23 21 18 value: 0.016129032


*******************************************
printing matrix off
*******************************************
(1/255) key: 20 21 20 19 value: 0.003921569
(1/6) key: 23 20 28 19 value: 0.16666667
(1/180) key: -1 -1 20 19 value: 0.0055555557
(1/249) key: 25 26 26 17 value: 0.004016064
(1/47) key: 23 20 20 4 value: 0.021276595
(1/176) key: 21 20 23 4 value: 0.0056818184
(3/1161) key: -1 -1 28 18 value: 0.0025839794
(1/397) key: 25 25 25 19 value: 0.0025188916
(1/14) key: 29 29 20 19 value: 0.071428575
(1/202) key: 20 23 21 19 value: 0.004950495
(2/68) key: 20 20 21 19 value: 0.029411765
(1/207) key: 27 28 28 18 value: 0.004830918
(1/124) key: -1 27 27 18 value: 0.008064516
(2/56) key: -1 23 20 13 value: 0.035714287
(1/8) key: 24 27 24 4 value: 0.125
(1/382) key: 24 23 24 18 value: 0.002617801
(1/382) key: 24 23 24 17 value: 0.002617801
(1/93) key: 24 20 21 17 value: 0.010752688
(1/66) key: 23 24 21 4 value: 0.015151516
(1/202) key: 20 23 21 4 value: 0.004950495
(1/167) key: 28 28 27 18 value: 0.005988024
(1/17) key: 27 20 28 18 value: 0.05882353
(2/180) key: 23 20 21 4 value: 0.011111111
(2/58) key: 29 29 21 19 value: 0.03448276
(1/26) key: 28 28 20 19 value: 0.03846154
(1/180) key: 23 20 21 13 value: 0.0055555557
(1/284) key: 26 25 26 17 value: 0.0035211267
(1/180) key: 23 20 21 19 value: 0.0055555557
(3/19) key: -1 -1 -1 19 value: 0.15789473
(1/58) key: -1 20 23 13 value: 0.01724138
(2/69) key: 24 21 20 4 value: 0.028985508
(1/7) key: 27 20 20 19 value: 0.14285715
(1/14) key: 27 27 23 19 value: 0.071428575
(9/19) key: -1 -1 -1 18 value: 0.47368422
(1/12) key: 24 23 25 17 value: 0.083333336
(2/73) key: 21 20 25 17 value: 0.02739726
(1/45) key: 21 24 21 18 value: 0.022222223
(1/69) key: 24 21 20 19 value: 0.014492754
(1/197) key: -1 -1 26 17 value: 0.005076142
(1/19) key: -1 -1 -1 13 value: 0.05263158
(1/10) key: 28 24 27 18 value: 0.1
(1/83) key: 28 20 21 13 value: 0.012048192
(1/47) key: 25 26 21 17 value: 0.021276595
(1/13) key: 26 20 21 19 value: 0.07692308
(1/61) key: -1 21 20 19 value: 0.016393442
(1/7) key: 22 25 25 17 value: 0.14285715
(1/51) key: 22 24 20 13 value: 0.019607844
(1/168) key: 20 24 23 13 value: 0.005952381
(1/222) key: 24 20 23 13 value: 0.0045045046
(1/222) key: 24 20 23 18 value: 0.0045045046
(1/37) key: 27 20 21 19 value: 0.027027028
(1/54) key: 20 28 27 13 value: 0.018518519
(1/571) key: -1 -1 27 18 value: 0.0017513135
```

```
(1/110) key: 23 20 23 13 value: 0.009090909
(1/2) key: 26 29 21 19 value: 0.5
(1/255) key: 20 21 20 18 value: 0.003921569
(1/485) key: 23 23 22 17 value: 0.0020618557
(1/175) key: 25 25 26 17 value: 0.0057142857
(1/130) key: -1 28 27 18 value: 0.0076923077
(2/321) key: 23 24 24 18 value: 0.0062305294
(1/27) key: 28 27 20 19 value: 0.037037037
(2/5968) key: -1 -1 24 13 value: 3.3512065E-4
(1/43) key: 21 29 29 19 value: 0.023255814
(1/156) key: 23 20 24 19 value: 0.0064102565
(1/14) key: 22 23 21 13 value: 0.071428575
(1/54) key: -1 20 21 19 value: 0.018518519
(1/117) key: 25 21 20 17 value: 0.008547009
(1/327) key: 28 27 28 18 value: 0.003058104
(1/65) key: 21 24 20 13 value: 0.015384615
(1/255) key: 26 26 25 17 value: 0.003921569
(1/3) key: 21 22 28 18 value: 0.33333334
(1/269) key: 27 27 28 19 value: 0.003717472
(1/6) key: 29 20 20 4 value: 0.16666667
(1/269) key: 27 27 28 18 value: 0.003717472
(1/246) key: -1 -1 25 17 value: 0.0040650405
(1/10) key: 28 20 28 17 value: 0.1
(1/413) key: 26 26 26 17 value: 0.0024213076
(2/81) key: 20 24 20 4 value: 0.024691358
(1/261) key: 24 24 23 18 value: 0.0038314175
(3/92) key: 21 20 27 19 value: 0.032608695
(1/7) key: 27 29 29 19 value: 0.14285715
(1/261) key: 24 24 23 13 value: 0.0038314175
(1/93) key: 21 21 21 18 value: 0.010752688
(5/49) key: 29 21 20 19 value: 0.10204082
(1/93) key: 21 21 21 19 value: 0.010752688
(1/555) key: 27 27 27 13 value: 0.0018018018
(1/6) key: 26 23 24 17 value: 0.16666667
(1/68) key: 27 21 20 19 value: 0.014705882
(1/555) key: 27 27 27 18 value: 0.0018018018
(1/104) key: 28 21 20 18 value: 0.009615385
(1/68) key: 27 21 20 18 value: 0.014705882
(1/104) key: 28 21 20 19 value: 0.009615385
(7/296) key: 21 20 21 19 value: 0.02364865
(1/11) key: 20 25 20 17 value: 0.09090909
(1/54) key: 21 20 28 18 value: 0.018518519
(1/93) key: 21 21 21 4 value: 0.010752688
(1/7) key: 28 24 23 13 value: 0.14285715
(1/296) key: 21 20 21 4 value: 0.0033783785
(1/27) key: 24 21 24 19 value: 0.037037037
(1/7) key: 20 24 27 18 value: 0.14285715
(2/90) key: 21 21 20 19 value: 0.022222223
(4/2575) key: -1 -1 23 18 value: 0.0015533981
(1/1) key: 20 28 22 18 value: 1.0
(1/79) key: 20 20 20 19 value: 0.012658228
(2/97) key: 21 20 20 19 value: 0.020618556
(1/90) key: 21 21 20 4 value: 0.011111111
(2/62) key: 24 23 21 4 value: 0.032258064
(1/31) key: 20 27 21 19 value: 0.032258064
(1/129) key: 20 21 24 4 value: 0.007751938
(1/414) key: 27 28 27 13 value: 0.002415459
(1/3) key: 23 28 24 13 value: 0.33333334
(2/414) key: 27 28 27 18 value: 0.004830918
```

Listing B.4: EventList.java

B.2.4 Pattern length 4, without zones

```
Loading Configurations
Database = jdbc:mysql://localhost/kiiib?user=KIIIB&password=42
pattern_interval = 10000
pattern_length = 3
use_zones = true
zone_interval = 500
probablility_threshold = 0.5
correlation_interval = 7000
correlation_correction = 0.1
default_on_time = 5000
punishment_timeout = 10000
debug = false
Trying to connect to the database
connection established
generating basic matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 1613
basic 137/113 (914)
generating zone matrices
fetching data from db
 iterating resultset
rows : 45797
runtime = 754
zone 225/168 (3872)
switches
18
13
19
17
4
sensors
24
23
20
21
27
28
22
25
26
29
30

******************************************
printing matrix on
******************************************
(1/9) key: 21 27 25 17 value: 0.11111111
(4/255) key: 20 21 20 19 value: 0.015686275
(1/2) key: 28 [20,21] [20,21,24] 19 value: 0.5
(1/1) key: 28 28 [20,21,27] 19 value: 1.0
(1/11) key: 20 22 20 19 value: 0.09090909
(1/23) key: 27 [27,28] 28 18 value: 0.04347826
(1/1165) key: -1 -1 28 19 value: 8.583691E-4
(1/396) key: 25 25 25 17 value: 0.0025252525
(1/104) key: 21 23 24 13 value: 0.009615385
(1/1) key: 27 [21,27] 20 19 value: 1.0
(1/57) key: [20,23] 21 20 19 value: 0.01754386
(1/112) key: 20 21 21 19 value: 0.008928572
(1/19) key: 27 21 28 19 value: 0.05263158
(1/389) key: 24 23 24 18 value: 0.002570694
```

```
(1/19) key: 27 21 28 18 value: 0.05263158
(2/23) key: 26 26 21 17 value: 0.08695652
(1/1) key: 20 21 [24,25] 17 value: 1.0
(1/164) key: 28 28 27 18 value: 0.0060975607
(1/1) key: [21,22] 20 [20,21] 19 value: 1.0
(1/1) key: 20 22 [20,21] 19 value: 1.0
(1/57) key: -1 -1 [20,23] 13 value: 0.01754386
(2/389) key: 24 23 24 4 value: 0.005141388
(2/58) key: 29 29 21 19 value: 0.03448276
(1/25) key: 23 22 20 13 value: 0.04
(1/3) key: [21,28] 20 21 19 value: 0.33333334
(2/24) key: 20 29 29 19 value: 0.083333336
(1/3) key: [20,23] [20,21] 28 18 value: 0.33333334
(2/1) key: 21 [21,24] [23,24] 4 value: 2.0
(1/1) key: [20,23,24] 20 21 13 value: 1.0
(5/20) key: -1 -1 -1 18 value: 0.25
(1/6) key: [20,21] [20,21] 21 19 value: 0.16666667
(1/4) key: 25 27 21 17 value: 0.25
(1/20) key: -1 -1 -1 4 value: 0.05
(1/14) key: 20 28 28 18 value: 0.071428575
(1/20) key: -1 -1 -1 13 value: 0.05
(1/7) key: 27 21 27 13 value: 0.14285715
(1/80) key: 28 20 21 19 value: 0.0125
(2/48) key: 25 26 21 19 value: 0.041666668
(2/80) key: 28 20 21 13 value: 0.025
(1/48) key: 25 26 21 17 value: 0.020833334
(2/66) key: -1 21 20 19 value: 0.030303031
(1/2) key: 21 21 [21,27] 17 value: 0.5
(1/16) key: 22 21 20 19 value: 0.0625
(1/4) key: [21,27] 25 26 17 value: 0.25
(1/3) key: [21,27] 20 20 19 value: 0.33333334
(1/2) key: [20,21] 27 25 17 value: 0.5
(1/255) key: 20 21 20 4 value: 0.003921569
(1/566) key: -1 -1 27 18 value: 0.0017667845
(1/1) key: 26 25 [20,21,25] 19 value: 1.0
(1/2) key: [20,23] 20 24 4 value: 0.5
(1/54) key: 23 21 21 17 value: 0.018518519
(1/7) key: [20,21] 21 29 19 value: 0.14285715
(2/9) key: 24 23 [20,21] 19 value: 0.22222222
(1/54) key: 23 21 21 19 value: 0.018518519
(1/1) key: [22,23,29] 20 20 19 value: 1.0
(2/17) key: 23 23 21 19 value: 0.11764706
(1/11) key: 21 [20,21] 23 13 value: 0.09090909
(1/15) key: 21 20 26 19 value: 0.06666667
(1/10) key: 25 27 27 17 value: 0.1
(1/11) key: 28 21 27 19 value: 0.09090909
(1/7) key: 27 20 25 17 value: 0.14285715
(1/3) key: 26 20 27 18 value: 0.33333334
(1/9) key: 24 23 [20,21] 4 value: 0.11111111
(1/51) key: 27 28 [20,21] 19 value: 0.019607844
(1/114) key: 23 21 20 13 value: 0.00877193
(2/51) key: 27 28 [20,21] 13 value: 0.039215688
(1/81) key: 21 27 28 18 value: 0.012345679
(1/114) key: 23 21 20 19 value: 0.00877193
(1/11) key: 28 20 28 18 value: 0.09090909
(1/13) key: 27 27 [20,21] 19 value: 0.07692308
(1/13) key: 27 27 [20,21] 17 value: 0.07692308
(1/79) key: 20 24 20 4 value: 0.012658228
(1/7) key: 27 29 29 19 value: 0.14285715
(1/93) key: 21 20 27 19 value: 0.010752688
(1/93) key: 21 20 27 17 value: 0.010752688
(1/1) key: 21 27 [21,27] 13 value: 1.0
(1/3) key: [20,23] 20 25 17 value: 0.33333334
(1/1) key: [20,24] [20,21] [21,25] 17 value: 1.0
(1/2) key: [20,27] 21 28 18 value: 0.5
```

```
(1/14) key: 20 25 20 19 value: 0.071428575
(1/1) key: [20,27] 21 [23,29] 19 value: 1.0
(1/226) key: 22 22 23 13 value: 0.0044247787
(1/2) key: 21 28 [21,27] 19 value: 0.5
(1/53) key: 21 20 28 18 value: 0.018867925
(1/12) key: 21 [20,21] 21 19 value: 0.083333336
(1/1) key: [22,29] 21 20 19 value: 1.0
(1/2) key: 20 [21,23] 25 17 value: 0.5
(1/44) key: 28 27 21 19 value: 0.022727273
(1/3) key: 20 [21,28] 20 18 value: 0.33333334
(1/80) key: 20 20 20 19 value: 0.0125
(1/34) key: -1 -1 [20,21] 19 value: 0.029411765
(1/28) key: 28 28 [20,21] 19 value: 0.035714287
(1/6) key: 27 23 21 19 value: 0.16666667
(1/27) key: 20 25 21 19 value: 0.037037037
(1/9) key: 26 [25,26] 21 19 value: 0.11111111
(1/2) key: [20,22] 25 26 19 value: 0.5
(1/4) key: [21,24] 25 25 19 value: 0.25
(1/1) key: [21,24] 27 [27,28] 18 value: 1.0
(1/4) key: 20 [20,21] 24 13 value: 0.25
(1/1) key: [20,21,27] 28 [20,27,28] 19 value: 1.0
(1/177) key: -1 -1 20 4 value: 0.0056497175
(1/14) key: 21 21 27 18 value: 0.071428575
(3/36) key: 20 21 29 19 value: 0.083333336
(1/188) key: 21 20 23 13 value: 0.005319149
(1/177) key: -1 -1 20 19 value: 0.0056497175
(1/71) key: 27 28 21 4 value: 0.014084507
(1/51) key: 23 20 20 4 value: 0.019607844
(1/162) key: 23 24 20 13 value: 0.0061728396
(1/202) key: 26 25 25 17 value: 0.004950495
(1/83) key: 20 24 21 19 value: 0.012048192
(2/89) key: 24 [20,23] 21 19 value: 0.02247191
(1/193) key: 20 23 21 19 value: 0.005181347
(1/53) key: -1 23 20 19 value: 0.018867925
(1/3) key: [20,21] 23 [20,24] 13 value: 0.33333334
(3/72) key: 20 20 21 19 value: 0.041666668
(1/14) key: 26 23 26 17 value: 0.071428575
(1/74) key: 27 28 20 19 value: 0.013513514
(1/193) key: 20 23 21 13 value: 0.005181347
(1/95) key: 24 20 21 18 value: 0.010526316
(1/7) key: 28 [27,28] [20,21] 19 value: 0.14285715
(1/95) key: 24 20 21 13 value: 0.010526316
(1/188) key: 23 20 21 4 value: 0.005319149
(1/438) key: 29 29 29 19 value: 0.002283105
(1/1) key: [21,25] 20 20 19 value: 1.0
(1/12) key: -1 21 [20,21] 19 value: 0.083333336
(1/6) key: 21 24 25 17 value: 0.16666667
(1/79) key: 20 24 20 19 value: 0.012658228
(1/126) key: 20 23 20 19 value: 0.007936508
(1/133) key: -1 -1 21 4 value: 0.007518797
(1/126) key: 20 23 20 18 value: 0.007936508
(1/188) key: 23 20 21 18 value: 0.005319149
(1/32) key: 23 21 25 17 value: 0.03125
(1/33) key: 21 25 25 19 value: 0.030303031
(4/188) key: 23 20 21 19 value: 0.021276595
(1/77) key: 21 20 25 18 value: 0.012987013
(1/58) key: -1 20 23 13 value: 0.01724138
(1/1) key: [27,28] 27 21 19 value: 1.0
(1/13) key: [20,23] 21 27 19 value: 0.07692308
(1/69) key: 20 21 27 19 value: 0.014492754
(3/28) key: 21 21 25 17 value: 0.10714286
(6/77) key: 21 20 25 17 value: 0.077922076
(1/7) key: 22 20 20 19 value: 0.14285715
(1/17) key: 28 27 [20,21] 19 value: 0.05882353
(1/1) key: 20 [20,21,23] 24 13 value: 1.0
```

```
(2/93) key: 21 25 26 17 value: 0.021505376
(1/16) key: 20 21 [20,23] 13 value: 0.0625
(1/5) key: 21 29 23 19 value: 0.2
(1/20) key: 24 23 [20,24] 13 value: 0.05
(1/124) key: 21 20 24 19 value: 0.008064516
(1/5) key: 23 [20,21] 25 17 value: 0.2
(1/3) key: 29 28 27 18 value: 0.33333334
(1/1) key: 24 [21,24] 25 17 value: 1.0
(1/218) key: 24 20 23 13 value: 0.0045871558
(1/37) key: 27 20 21 17 value: 0.027027028
(1/41) key: 20 20 24 4 value: 0.024390243
(1/124) key: 21 20 24 13 value: 0.008064516
(1/9) key: 20 21 [20,24] 19 value: 0.11111111
(1/1) key: [23,24] 22 20 13 value: 1.0
(1/41) key: 20 20 24 13 value: 0.024390243
(1/1) key: [20,28] 25 [21,27] 17 value: 1.0
(1/101) key: 23 20 23 13 value: 0.00990099
(1/4) key: 21 [20,21] [20,21] 13 value: 0.25
(1/14) key: 29 22 29 19 value: 0.071428575
(2/63) key: 20 21 25 17 value: 0.031746034
(1/51) key: -1 20 21 4 value: 0.019607844
(1/1) key: [25,26] 20 27 18 value: 1.0
(1/51) key: -1 20 21 13 value: 0.019607844
(1/67) key: 21 23 20 13 value: 0.014925373
(1/17) key: 21 27 21 17 value: 0.05882353
(1/9) key: 29 20 21 19 value: 0.11111111
(1/32) key: 23 21 28 18 value: 0.03125
(1/57) key: 20 25 26 19 value: 0.01754386
(1/70) key: 26 25 21 19 value: 0.014285714
(1/51) key: -1 20 21 19 value: 0.019607844
(1/14) key: [20,21] 21 20 19 value: 0.071428575
(1/6) key: 27 [20,21] 20 19 value: 0.16666667
(1/137) key: 23 20 24 13 value: 0.00729927
(1/2) key: -1 [20,21] 25 17 value: 0.5
(1/322) key: 28 27 28 18 value: 0.0031055901
(1/236) key: 24 23 20 13 value: 0.004237288
(1/3) key: 27 [20,21] [21,29] 19 value: 0.33333334
(1/113) key: 25 21 20 19 value: 0.0088495575
(1/252) key: 26 26 25 17 value: 0.003968254
(4/275) key: 27 27 28 18 value: 0.014545455
(2/64) key: 21 24 20 19 value: 0.03125
(1/11) key: [20,24] 21 20 19 value: 0.09090909
(1/1) key: [20,23,24] 21 [21,25] 17 value: 1.0
(2/50) key: 25 25 21 17 value: 0.04
(1/76) key: 20 27 28 18 value: 0.013157895
(1/32) key: 25 20 21 19 value: 0.03125
(1/9) key: 24 21 25 17 value: 0.11111111
(1/23) key: 20 25 25 17 value: 0.04347826
(1/18) key: 23 20 25 17 value: 0.055555556
(1/4) key: -1 20 [20,21] 4 value: 0.25
(2/269) key: 24 24 23 13 value: 0.007434944
(1/1) key: -1 [20,21] 26 19 value: 1.0
(2/298) key: 28 27 27 18 value: 0.0067114094
(2/49) key: 29 21 20 19 value: 0.040816326
(1/19) key: 28 [20,21] 21 19 value: 0.05263158
(1/1) key: 26 [21,25] [20,25] 18 value: 1.0
(1/547) key: 27 27 27 13 value: 0.0018281536
(1/3) key: [21,27] 20 25 17 value: 0.33333334
(4/68) key: 27 21 20 19 value: 0.05882353
(1/285) key: 21 20 21 13 value: 0.003508772
(1/104) key: 28 21 20 18 value: 0.009615385
(2/104) key: 28 21 20 19 value: 0.01923077
(1/122) key: 20 21 23 13 value: 0.008196721
(8/285) key: 21 20 21 19 value: 0.028070176
(1/5) key: 21 [20,23] 20 18 value: 0.2
```

```
(1/8) key: 20 21 [20,27] 17 value: 0.125
(1/2) key: 23 [21,27] 28 19 value: 0.5
(1/6) key: [27,28] [27,28] 27 18 value: 0.16666667
(1/1) key: [21,28] [25,27] 27 17 value: 1.0
(1/17) key: 24 [20,23,24] 21 19 value: 0.05882353
(1/1) key: [20,29] 21 20 19 value: 1.0
(1/3) key: 23 20 26 17 value: 0.33333334
(1/8) key: 20 21 [20,27] 19 value: 0.125
(1/3) key: [25,26] 23 26 17 value: 0.33333334
(1/1) key: [20,24] [20,23,24] 21 18 value: 1.0
(2/285) key: 21 20 21 4 value: 0.007017544
(1/7) key: 21 21 [20,21] 4 value: 0.14285715
(6/93) key: 21 20 20 19 value: 0.06451613
(1/91) key: 21 21 20 4 value: 0.010989011
(2/27) key: 21 27 20 19 value: 0.074074075
(1/6) key: [20,21] 25 21 19 value: 0.16666667


*******************************************
printing matrix off
*******************************************
(1/1) key: [27,28] 20 20 19 value: 1.0
(1/1) key: 29 21 [20,21,23] 19 value: 1.0
(2/255) key: 20 21 20 19 value: 0.007843138
(1/5) key: 23 20 28 19 value: 0.2
(1/89) key: 24 [20,23] 21 4 value: 0.011235955
(1/1) key: [24,27,28] 21 20 18 value: 1.0
(1/177) key: -1 -1 20 19 value: 0.0056497175
(1/247) key: 25 26 26 17 value: 0.004048583
(1/23) key: 27 [27,28] 28 18 value: 0.04347826
(1/51) key: 23 20 20 4 value: 0.019607844
(1/12) key: -1 [20,21] 21 19 value: 0.083333336
(1/1) key: [21,23,24] 26 [23,24] 17 value: 1.0
(1/188) key: 21 20 23 4 value: 0.005319149
(3/1165) key: -1 -1 28 18 value: 0.0025751074
(1/396) key: 25 25 25 19 value: 0.0025252525
(1/15) key: 29 29 20 19 value: 0.06666667
(1/193) key: 20 23 21 19 value: 0.005181347
(1/72) key: 20 20 21 17 value: 0.013888889
(2/72) key: 20 20 21 19 value: 0.027777778
(1/205) key: 27 28 28 18 value: 0.004878049
(1/53) key: -1 23 20 13 value: 0.018867925
(1/8) key: 24 27 24 4 value: 0.125
(1/389) key: 24 23 24 18 value: 0.002570694
(1/389) key: 24 23 24 17 value: 0.002570694
(1/57) key: -1 -1 [20,23] 18 value: 0.01754386
(1/23) key: 26 26 21 17 value: 0.04347826
(1/1) key: 27 [23,28] [20,21] 19 value: 1.0
(1/3) key: [27,28] 28 20 19 value: 0.33333334
(1/58) key: 23 24 21 4 value: 0.01724138
(1/193) key: 20 23 21 4 value: 0.005181347
(1/164) key: 28 28 27 18 value: 0.0060975607
(2/188) key: 23 20 21 4 value: 0.010638298
(1/1) key: 26 [23,26] [25,26] 17 value: 1.0
(2/57) key: -1 -1 [20,23] 13 value: 0.03508772
(1/2) key: [20,21] [23,24] 21 4 value: 0.5
(1/12) key: -1 21 [20,21] 19 value: 0.083333336
(1/1) key: 29 26 [21,29] 19 value: 1.0
(2/58) key: 29 29 21 19 value: 0.03448276
(1/30) key: 28 28 20 19 value: 0.033333335
(1/188) key: 23 20 21 13 value: 0.005319149
(1/283) key: 26 25 26 17 value: 0.003533569
(1/3) key: 27 [20,21] 25 17 value: 0.33333334
(1/188) key: 23 20 21 19 value: 0.005319149
(1/3) key: 24 [27,28] 27 13 value: 0.33333334
```

```
(3/20) key: -1 -1 -1 19 value: 0.15
(1/1) key: 21 21 [22,28] 18 value: 1.0
(2/73) key: 24 21 20 4 value: 0.02739726
(2/58) key: -1 20 23 13 value: 0.03448276
(1/13) key: 27 27 23 19 value: 0.07692308
(9/20) key: -1 -1 -1 18 value: 0.45
(1/9) key: 24 23 25 17 value: 0.11111111
(1/77) key: 21 20 25 17 value: 0.012987013
(1/44) key: 21 24 21 18 value: 0.022727273
(1/17) key: 28 27 [20,21] 19 value: 0.05882353
(1/73) key: 24 21 20 19 value: 0.01369863
(1/3) key: 25 23 [22,23] 17 value: 0.33333334
(1/195) key: -1 -1 26 17 value: 0.0051282053
(1/20) key: -1 -1 -1 13 value: 0.05
(1/7) key: 28 20 20 19 value: 0.14285715
(1/14) key: 20 27 24 18 value: 0.071428575
(2/1) key: [20,21] [20,24] [21,23] 4 value: 2.0
(1/48) key: 25 26 21 17 value: 0.020833334
(1/1) key: [21,24] [20,24] [21,24] 19 value: 1.0
(1/13) key: 26 20 21 19 value: 0.07692308
(3/66) key: -1 21 20 19 value: 0.045454547
(1/7) key: 22 25 25 17 value: 0.14285715
(1/52) key: 22 24 20 13 value: 0.01923077
(1/166) key: 20 24 23 13 value: 0.006024096
(1/218) key: 24 20 23 18 value: 0.0045871558
(1/1) key: 24 [25,26] 26 17 value: 1.0
(1/37) key: 27 20 21 19 value: 0.027027028
(1/52) key: 20 28 27 13 value: 0.01923077
(1/34) key: 28 27 [27,28] 18 value: 0.029411765
(2/566) key: -1 -1 27 18 value: 0.003533569
(1/101) key: 23 20 23 13 value: 0.00990099
(1/1) key: 26 29 21 19 value: 1.0
(1/255) key: 20 21 20 18 value: 0.003921569
(1/10) key: -1 [20,21] 24 4 value: 0.1
(1/483) key: 23 23 22 17 value: 0.0020703934
(1/68) key: 24 24 [20,23] 13 value: 0.014705882
(1/174) key: 25 25 26 17 value: 0.0057471264
(1/132) key: -1 28 27 18 value: 0.007575758
(1/315) key: 23 24 24 18 value: 0.0031746032
(1/28) key: 28 27 20 19 value: 0.035714287
(2/5966) key: -1 -1 24 13 value: 3.35233E-4
(1/70) key: 26 25 21 17 value: 0.014285714
(1/1906) key: 24 24 24 18 value: 5.2465894E-4
(1/43) key: 21 29 29 19 value: 0.023255814
(1/137) key: 23 20 24 19 value: 0.00729927
(1/17) key: 22 23 21 13 value: 0.05882353
(1/3) key: 28 29 28 19 value: 0.33333334
(1/4) key: 29 29 [20,21] 19 value: 0.25
(1/1) key: 27 [20,21,28] [24,27] 18 value: 1.0
(1/236) key: 24 23 20 13 value: 0.004237288
(1/322) key: 28 27 28 18 value: 0.0031055901
(1/64) key: 21 24 20 13 value: 0.015625
(1/252) key: 26 26 25 17 value: 0.003968254
(1/7) key: 29 20 20 4 value: 0.14285715
(1/275) key: 27 27 28 19 value: 0.0036363637
(2/3) key: 29 [20,21] 20 19 value: 0.6666667
(1/275) key: 27 27 28 18 value: 0.0036363637
(1/12) key: 28 21 21 19 value: 0.083333336
(1/1) key: 21 28 22 18 value: 1.0
(1/2) key: [20,21,24] 20 21 17 value: 0.5
(1/248) key: -1 -1 25 17 value: 0.004032258
(1/9) key: -1 27 [27,28] 18 value: 0.11111111
(1/13) key: 27 27 [20,21] 19 value: 0.07692308
(1/2) key: [20,21] 20 [20,21] 19 value: 0.5
(1/11) key: 28 20 28 17 value: 0.09090909
```

```
(1/1) key: [20,21] [23,24] 25 17 value: 1.0
(1/1) key: 25 [25,26] [20,21] 19 value: 1.0
(2/79) key: 20 24 20 4 value: 0.025316456
(1/269) key: 24 24 23 18 value: 0.003717472
(2/11) key: [20,24] 21 20 4 value: 0.18181819
(2/93) key: 21 20 27 19 value: 0.021505376
(1/7) key: 27 29 29 19 value: 0.14285715
(1/3) key: 24 [20,23,24] 24 18 value: 0.33333334
(1/8) key: 21 [20,27] 21 19 value: 0.125
(1/269) key: 24 24 23 13 value: 0.003717472
(1/91) key: 21 21 21 18 value: 0.010989011
(1/1) key: 24 [20,23] [20,28] 19 value: 1.0
(1/19) key: 28 [20,21] 21 19 value: 0.05263158
(1/1) key: 20 [23,28] 24 13 value: 1.0
(4/49) key: 29 21 20 19 value: 0.08163265
(1/104) key: 28 21 20 13 value: 0.009615385
(1/91) key: 21 21 21 19 value: 0.010989011
(1/547) key: 27 27 27 13 value: 0.0018281536
(1/7) key: 26 23 24 17 value: 0.14285715
(1/68) key: 27 21 20 19 value: 0.014705882
(1/547) key: 27 27 27 18 value: 0.0018281536
(1/104) key: 28 21 20 18 value: 0.009615385
(1/68) key: 27 21 20 18 value: 0.014705882
(5/285) key: 21 20 21 19 value: 0.01754386
(1/14) key: 20 25 20 17 value: 0.071428575
(1/3) key: 25 [25,26] [25,26] 17 value: 0.33333334
(1/53) key: 21 20 28 18 value: 0.018867925
(1/91) key: 21 21 21 4 value: 0.010989011
(1/7) key: 28 24 23 13 value: 0.14285715
(1/1) key: -1 [27,28] [20,21] 13 value: 1.0
(1/2) key: 28 24 [20,21] 19 value: 0.5
(1/12) key: 27 20 27 18 value: 0.083333336
(1/4) key: 29 21 [20,21] 19 value: 0.25
(1/285) key: 21 20 21 4 value: 0.003508772
(1/1) key: 21 [20,27] 27 18 value: 1.0
(1/5) key: 20 24 27 18 value: 0.2
(1/91) key: 21 21 20 19 value: 0.010989011
(4/2572) key: -1 -1 23 18 value: 0.00155521
(1/56) key: [27,28] 27 27 18 value: 0.017857144
(1/9) key: [21,25] 20 23 4 value: 0.11111111
(1/17) key: 23 [20,24] 23 18 value: 0.05882353
(1/1) key: 20 28 22 18 value: 1.0
(1/80) key: 20 20 20 19 value: 0.0125
(1/42) key: [23,24] 24 24 18 value: 0.023809524
(2/34) key: -1 -1 [20,21] 19 value: 0.05882353
(3/93) key: 21 20 20 19 value: 0.032258064
(1/91) key: 21 21 20 4 value: 0.010989011
(2/39) key: 24 21 23 4 value: 0.051282052
(1/28) key: 28 28 [20,21] 18 value: 0.035714287
(1/27) key: 20 25 21 17 value: 0.037037037
(1/4) key: 20 21 [20,28] 18 value: 0.25
(1/1) key: 26 [20,21,26] [20,23,24] 19 value: 1.0
(2/29) key: 20 27 21 19 value: 0.06896552
(1/20) key: 24 24 21 19 value: 0.05
(1/113) key: 20 21 24 4 value: 0.0088495575
(1/6) key: [27,28] [27,28] 28 18 value: 0.16666667
(1/419) key: 27 28 27 13 value: 0.002386635
(1/5) key: 23 28 24 13 value: 0.2
(2/419) key: 27 28 27 18 value: 0.00477327
```

Listing B.5: EventList.java