NTNU
Innovation and Creativity

Marvin - Intelligent Corridor Guide

Ole Kristian Hartvigsen

Master of Science in Computer Science
Submission date: June 2006
Supervisor: Tore Amble, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

One can imagine a Guiding Robot that is able to interact verbally with its users in order to guide them in a hallway environment. Such a system would use speech recognition, data vision, speech generation and robot movement, in addition to an intelligent system for natural language understanding, reasoning and planning.

The task is to implement an Intelligent Corridor Guide, that can guide a person in a hallway environment in order to find the office of a named person. The system will partly build on already existing modules and a prototype simulator. An important task is to integrate the modules tightly and come up with good solutions for robot movement, planning and identification of locations.

Assignment given: 20. January 2006
Supervisor: Tore Amble, IDI

Preface

This master thesis project was carried out by Ole Kristian Hartvigsen. It is part of the Master of Science and Technology degree at Norwegian University of Science and Technology in Trondheim, Norway. In addition to this paper, an implementation was delivered.

I would like to thank my supervisor Tore Amble for creating this challenging task and for all the assistance during the project. I would also like to thank Magne Syrstad for providing the WSU Khepera Robot Simulator and Arild Wærnes for helpful information from the LDAP Database.

Trondheim June 23 2006

Ole Kristian Hartvigsen

Summary

Intelligent helpers are becoming increasingly popular as computer systems are being used in new areas and by new users every day. Programs and robots that communicate with users in a human-like way offer friendlier and easier use, especially for systems that are used by a random selection of people who shouldn't need prior knowledge of the interface.

This project considers an intelligent helping system that performs a specific human-like task in a real world environment. The system is named Marvin and is going to be a guide for people who are unfamiliar with a building. Imagine entering a building full of hallways and doors, not knowing where to go, and having a robot greet you. You can speak to the robot just as if it was a human being and it will give you the information that you need or even lead you to the place where you want to go.

In this project, a prototype simulator of Marvin is implemented to work in the third floor of the building of The Department of Computer and Information Science at Norwegian University of Science and Technology. Questions and requests to Marvin can be made through written natural language. The program answers questions with natural language sentences, additional map presentations, and simulated robot movement.

Contents

1 Introduction

The design of a corridor guide robot that can be controlled with human language through speech, and that can guide the user to a location within a building, requires integration of many different technologies. On the hardware side, microphones, speakers, sensors/cameras and motors are required for the robot to be able to interact with its surroundings, gather information about location and to physically move around. On the software side; speech recognition, databases and planners are some of the pieces needed in the puzzle to make the robot be able to extract the actual meanings of sentences, gather information about people, offices and hallways, and plan for its own movements in the real world.

In this chapter, an introduction is made to some of the technologies needed for Marvin – The Intelligent Corridor Guide. A more in-depth presentation of the different technologies is made in Chapter 2. A presentation of this specific project is given in Chapter 3, while the design of the system is explained in Chapter 4. Chapter 5 gives a detailed description of the implementation of Marvin. There is an evaluation of the project in Chapter 6.

All source code made by the author is printed in the Appendices.

1.1 History of robots

The history of robots, or rather artificial people, goes as far back as to ancient legends and myths. Typically, these stories are about mechanical servants and warriors, or clay monsters controlled by magic. Even in old Norse mythology, there is the story about the troll Hrugnir who created a clay giant called Mökkurkálfi to aid him in a duel with the God of Thunder, Thor.

Even though these stories differ from what we today know as «robots», they still have a connection to the word that originated the concept, *robotovát*, which is a Czech slang verb meaning «to work» or «to slave». Robots, or artificial beings, are primarily created to perform some task. In the 1927 classic film «Metropolis», a robotic woman is created by the scientist Rotwang, not only to imitate a real woman, but to create riots among the working men.

Robots that simulate human beings, also known as androids, have always fascinated the human race. However, the real success of robots has come through automation in industrial factories, where robots can be designed to effectively perform repeating tasks over and over again. When it comes to «guiding» robots, a number of attempts have been made with Museum Guides [Burgard 1998] [Thrun 1998]. The results from such projects are interesting, but they have not lead to a break-through on the commercial market. Other similar robot projects have been focusing on transportation of medicines and equipment in hospitals.

1.2 History of artificial intelligence

Artificial intelligence (AI) has always been a treat for the human mind, and has been the theme for several works in literature and films, like «The Terminator» and «The Matrix», where AI becomes so powerful that it overthrows the human society. In reality, AI has proven to be harder to develop than expected, or maybe, hoped for. Early developments of programs that could perform elementary reasoning tasks and play chess set high hopes for the future, but proved to be mainly automated mathematics within very specific boundaries. Later, two concepts emerged: «Strong AI», which deals with machines becoming human-like and self-aware, and «Weak AI», which deals with specific problem solving and reasoning that does not encompass the full range of human-like thinking. Today, weak AI is successfully being used in areas like economics, medicine, engineering and the military, performing various tasks of control, planning, scheduling, diagnostics, speech-, face- and handwriting recognition, etc.

The subject of this specific project is an intelligent «helping device» for people. Such systems may not necessarily be called AI systems. «Intelligent helpers» is a more convenient description.

1.3 Conversational user interface

The task of a conversational user interface (CUI) is to make it possible for a computer program (or in this project, a robot) to interact with the user through natural language. Figure 1.3.1 shows how a CUI works on a high level. The transformations between airwaves and digital audio (microphones, speakers and D/A converters) have been left out.

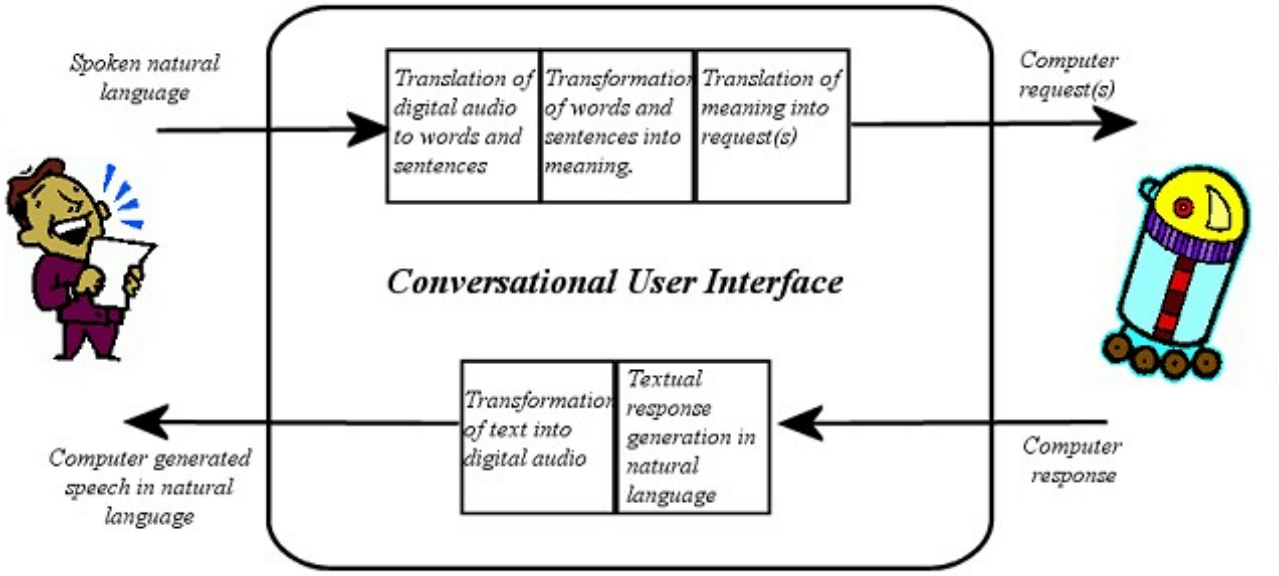

Fig. 1.3.1: A conversational user interface.

An interface like this has two major advantages compared to other, traditional interfaces:

- The user can interact with the program without using hands.
- There is no «technological barrier» between the user and the system.

The first of these advantages should be intuitive and is important for tasks that require physical attention of the user. A car with a voice controlled CUI, for instance, could make it possible for the driver to turn on the air-conditioning or change radio channels without removing the hands from the steering wheel. It is also an advantage for people with certain disabilities. For the robot guide system of this project, freedom of hands isn't a major issue. Still, it makes the use of the robot easier and friendlier, which leads to the second advantage of a voice controlled CUI: No technological barrier between the user and the system. This may seem somewhat unimportant at first, but is actually a very important feature for a system that is supposed to be used by a wide selection of random users. Even a computer keyboard can be enough to scare users away. The fact that the CUI works with natural language also implies that the user does not need any prior knowledge of the system to use it, or at least to interact with it.

1.4 World knowledge

The “guide robot” must have access to information that the users of the system may ask for, and information that the robot will need to perform its tasks. For instance, if the user asks for the room- or phone number of a specific person at NTNU, the robot must be able to access a database of people who work at NTNU and find the requested information. If a user asks for guidance to a specific room, the robot needs to know the location of this room and any other information that is required for planning how to get there, or at least to make a description of how the *user* can get there.

1.5 Mobile robots

[Kelly 1996] defines mobile robots as any system which is mobile, somewhat autonomous and intelligent. There are many possible solutions to designing a mobile robot given specific tasks. Things like terrain, type of locomotion and steering, «grabbing» functions, size and shape all determine how a robot will function. For instance, a robot can be designed for indoor or outdoor use, be wheeled or legged, look like a Man or a dog, or a small car, etc.

Mobile robots usually have one of the following four purposes: They either provide access to areas that are hard or impossible for humans to reach, they reduce costs of some specific work, they increase productivity, or they improve the quality of a system. In this project, the role of the robot is the latter, to improve the quality of the system by adding a new and unique feature to it: guiding in the real world. Figure 1.5.1 shows a guide robot, called RoboX, which has been used as a tour-guide in a crowded environment [Philippsen and Siegwart 2002]. There are some key features that are absolutely needed for such a robot:

- It should be fast enough for a person to walk comfortably with it.
- Collision risk must be as low as possible and collisions must be harmless when they occur.
- Motion must be smooth and obstacle avoidance must be fast and reliable.

Fig. 1.5.1: Tour-guide robot

1.6 Goal description

The project description that was defined before the project started reads (translated from Norwegian):

One can imagine a Guiding Robot that is able to interact verbally with its users in order to guide them in a hallway environment. Such a system would use speech recognition, data vision, speech generation and robot movement, in addition to an intelligent system for natural language understanding, reasoning and planning.

The task is to implement an Intelligent Corridor Guide, that can guide a person in a hallway environment in order to find the office of a named person. The system will partly build on already existing modules and a prototype simulator. An important task is to integrate the modules tightly and come up with good solutions for robot movement, planning and identification of locations.

The ultimate vision behind this project is to make a robot that can interact with users through natural language, answer questions about people and locations, and provide other useful information for getting around. The robot should also be able to offer guiding to nearby locations. This requires a very complex and expensive system, and goes far beyond the goals of this project.

Specific goals for this project are to explore the possibilities within the different parts that are required for a complete system, and make a prototype or simulator. Focus is mainly on the interaction and information parts of the system and less on the robotics part.

Interaction between user and program shall be in natural language with, at least, a textual interface, or, more preferably, with a speech/voice interface. The program shall be able to answer questions and requests in an intelligent way, and be able to provide information about people and locations within the scope of the project (see chapter 1.7). The program must also be able to control dialogue in a way that makes conversation realistic.

Since a useful robot is not available for the project, a simulator should be used to show how the robot would move around in the world. The simulator should have some sort of planning and scheduling algorithm.

1.7 Scope of the work

The project focuses mainly on implementation of a prototype/simulator. Some research should also be done to determine possible solutions to different parts of the system. The program will be imagining a robot placed in the 3rd floor of the IDI building at NTNU Gløshaugen. A map of this floor is shown in figure 1.7.1. The simulated robot should be able to move around on this floor. In addition, the program should have knowledge about people at NTNU, buildings and offices, so that it can provide information about places outside the robot's «reach». Other useful information that helps the user getting to a wanted location is also desirable.

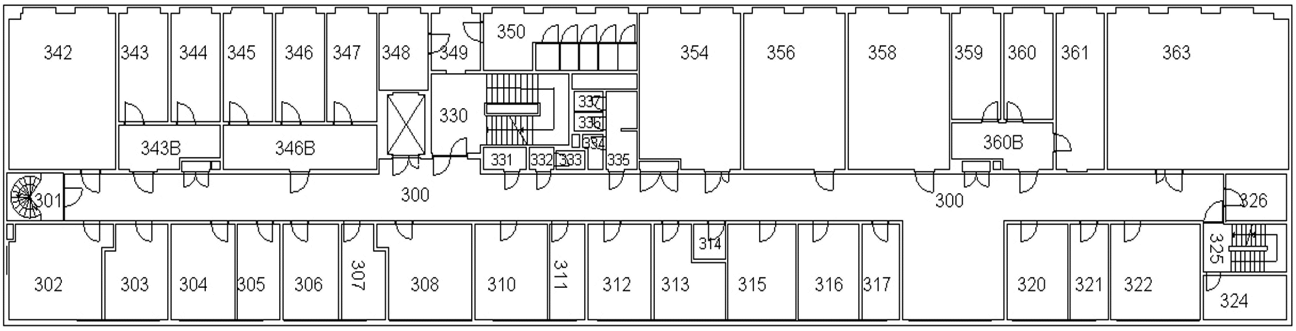

Fig. 1.7.1: The third floor of the IDI building at NTNU Gløshaugen.

1.8 Related projects

This project builds on work already done at NTNU. A key part of this project is Buster. Buster is a natural language dialogue system developed at NTNU, which is built on The Understanding Computer (TUC) [Amble 2004]. TUC has been the foundation for other projects as well, including BusTUC [Bratseth 1997] which is a system that answers questions about bus routes.

1.8.1 TUC

The Understanding Computer is a natural language processor developed by Tore Amble. It is implemented in SICStus Prolog. TUC translates natural language sentences into logical expressions in a language called The Query Logic (TQL). Here is an example of how this works:

Input sentence:

Where is the office of Magne Johnsen?

TQL expression:

which(A)

magne isa firstname

johnsen isa lastname

B isa office

A isa place

has/person/office/(magne,johnsen)/B

event/real/C

lie1/B/D

srel/in/place/A/D

event/real/D

The TQL expression is used to express the semantics of the input sentence to an external application. The first line of the TQL expression determines the sentence type, and the rest of the expression gives the actual meaning of the sentence. TQL expressions will be further discussed in Chapter 2.

1.8.2 Buster

Buster [Fledsberg & Bjerkevoll 1999] is a natural language dialogue system based on BusTUC, which in turn is based on TUC. There also exists a version of Buster that aims on answering questions about phone numbers for employees at NTNU. Buster works with the concept of frames, which is a set of slots that are updated throughout the dialogue. Such slots may be information about desired departure or desired arrival (for BusTUC), and each slot may contain several slots with information like «place», «time», etc. This makes it possible for Buster to keep track of the information that the user has provided and enables Buster to ask questions back to the user for additional information.

The BusTUC system has successfully been used by the bus company Team Trafikk AS as an «oracle» that answers questions about bus routes to users on the internet (Figure 1.8.2.1) and via SMS.

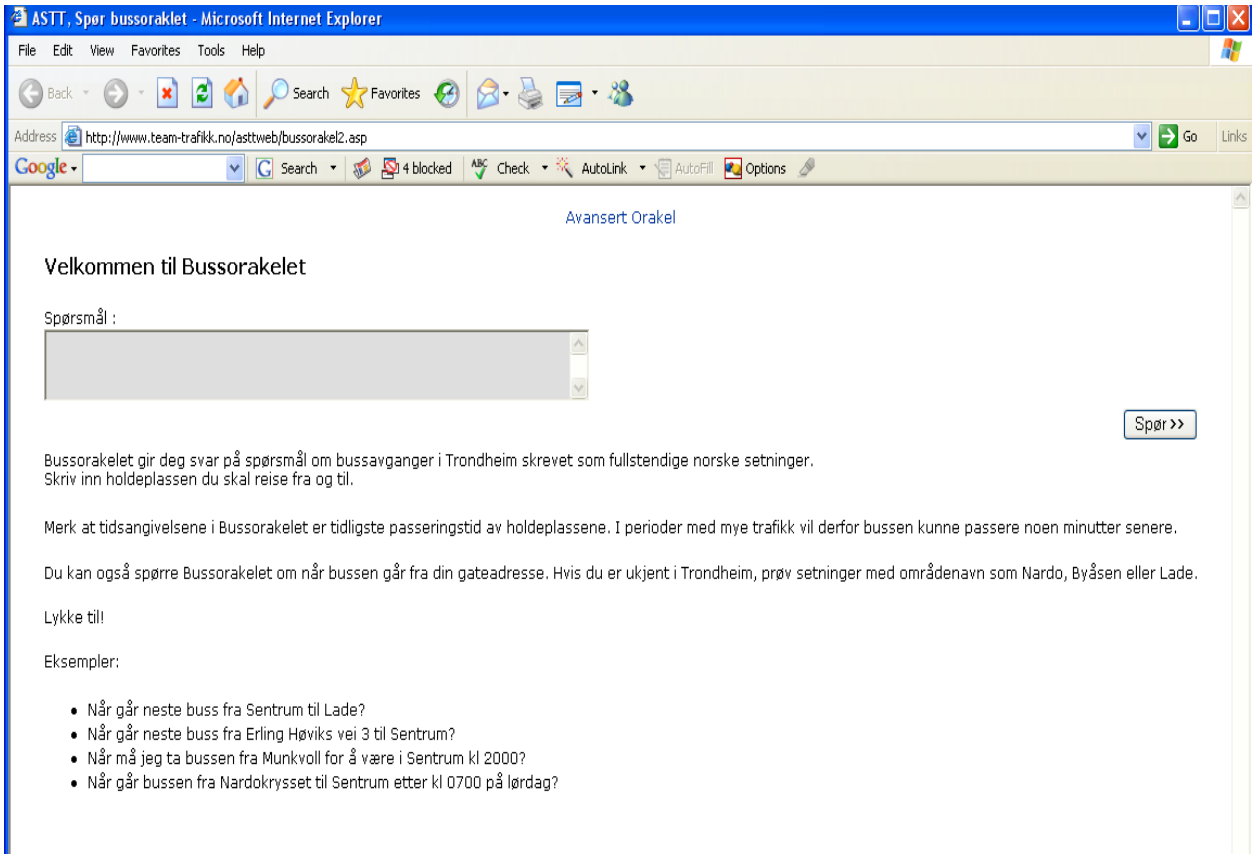

Fig. 1.8.2.1: BusTUC serving as a «bus oracle».

2 Theoretical Framework

In this chapter some of the basic theories that make out the foundation of this project are presented. Some of the material presented here may not be directly connected with the actual implementation that is later presented in making a prototype of Marvin, but is still important in understanding how the system works.

2.1 Conversational User Interface

The advantages of a conversational user interface (CUI) should be quite intuitive. One of them is that a user won't need any instructions in order to use it, since it aims at letting the user «speak» directly to the system in a «human» way. When a CUI is implemented to handle spoken sentences and to generate audible answers, it removes the technological barrier completely. This section gives an overview of different kinds of CUIs, based on [McTear 2002] who defines three different kinds of CUIs: *Finite state-based systems*, *frame-based systems* and *agent-based* systems.

2.1.1 Finite state-based CUI

The most simple kind of CUI is the finite-state CUI. These systems take full control of the dialogue and lets the user answer questions along the way. Staff at fast food restaurants often utilize this kind of dialogue, as they ask questions like «What kind of menu do you want?», «What kind of soda do you want with that?», «Take-away or eat here?» and let the customer answer the questions one by one. An example of how such a dialogue can look like in a computer program is given below:

Program: Please enter the name of the person you want to visit.

User: I don't know his name.

Program: You have not entered a name. Please enter the number of the room you want to go to.

User: 397.

Program: The room number you entered is incorrect. Program terminated.

Program: Please enter the name of the person you want to visit.

User: Tore Amble.

Program: Tore Amble's office is room number 312. Do you want me to take you there?

User: Yes.

Program: Okay. Please follow me.

This example shows how a corridor guide robot could work with a simple finite state CUI. The

state-map of this program would be similar to the one shown in Figure 2.1.1.1. A finite state CUI is not very flexible when it comes to dialogue and allowed input. In this example, if the user first answers with «I don't know the name of the person, but his office is 312», then the program would still just answer «You have not entered a name», and continue by asking for the already given room number. The state-machine could be made more complex by letting the program listen for both names and room numbers after it's initial question, and then choosing it's next state, but it still wouldn't be able to handle «realistic» dialogues very convincingly. If the user answers with just a first name, the system would need another state to ask for the sir name. If the user wants to ask the program something, for instance what time it is, the state-machine would need another layer just to uncover what the user wants to know. Should the user find it appropriate to ask for the time while in some other state than the first, the program would be completely lost unless every state had this event as a possibility.

The advantage of the finite state-based CUI lies in its simplicity. For a program with very specific tasks, this can be a very good solution. Grammar and vocabulary is also very simple, as the program knows what information it wants from the user at any time.

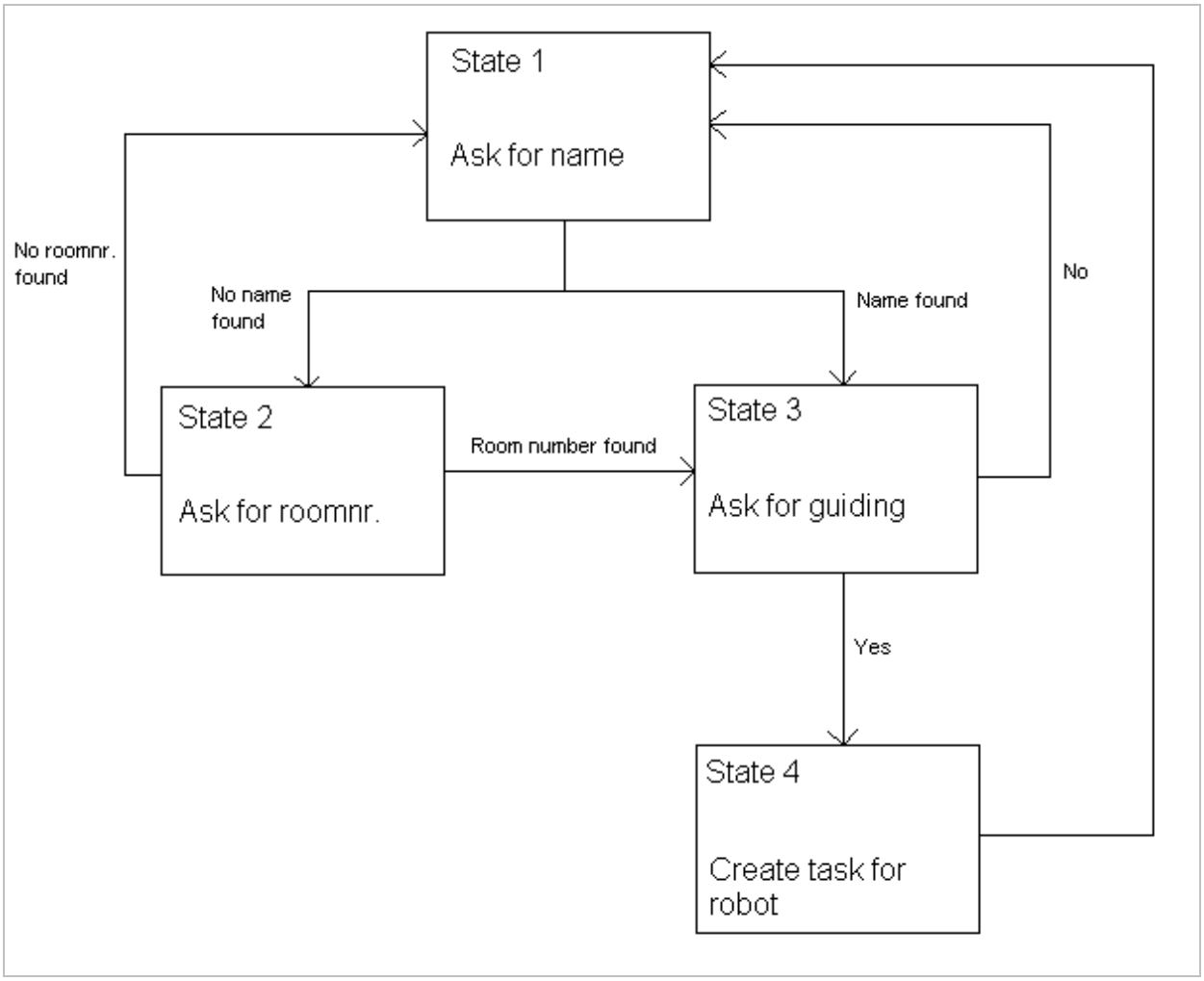

Fig. 2.1.1.1: Example of a state-based CUI.

2.1.2 Frame-based CUI

Frame-based systems are similar to finite state-based systems in many ways, but offer more flexible dialogues. A frame-based system has an information frame which is updated as the user provides information, which makes it possible for the system to gather different information at any time and ask for whatever information is missing in the frame. The Buster program has a frame based CUI. An example of how a dialogue with a frame-based CUI may look like follows:

Program: Hello, I am a Corridor Guide Robot. How may I help you?

User: I'm here to visit Tore.

Program: Do you know his last name?

User: No.

Program: Do you know his department?

User: He works at idi.

Program: Tore Amble's office is room 312. Do you want me to take you there?

User: No thanks.

Program: Okay.

User: What's his phone number?

Program: Tore Amble's phone number is 94451.

This example assumes that there is only one person named Tore at the idi department, which makes it possible for the program to track down the desired person. When the user asks for the phone number of «him», the program looks up the name of the person from its frame and uses the name to search for the correct phone number. Notice that the program asks for the missing last name after the user has provided the first name. What makes the frame based system similar to the state-based system is that the program still takes control of the dialogue, but since the goal of the program is to fill in slots in the frame, the user can give more than one information at any time and the dialogue becomes more flexible.

A problem with this model can be to identify when a conversation starts and ends. For example, the

dialogue above isn't necessarily finished, but it could be. If the next user input is «Does Arvid work here?», the program needs to understand that the user has ended the previous conversation and is asking about a different person. If not, the program would update its frame with a new first name and start searching for a person called «Arvid Amble» at the idi department.

2.1.3 Agent-based CUI

Agent-based systems are the most advanced dialogue systems. They allow complex dialogue between the user and the program, with both parties allowed to take initiative. An example of a conversation with an agent-based CUI follows:

Program: Hello, I'm a Corridor Guide Robot. How may I help you?

User: I'm supposed to meet Tore Amble.

Program: Tore Amble's office is room 312. Do you want me to take you there?

User: I'm supposed to meet him at a meeting room.

Program: There are two meeting rooms on this floor, room 356 and room 354. There is a meeting scheduled at room 356 at 11:00 and at 354 at 11:15.

User: The meeting I'm going to be at is supposed to be just about now.

Program: The current time is 10:56, so I'm assuming you want to go to the meeting at room 356 at 11:00. Do you want me to take you to room 356?

User: Yes, please.

In this example, the user takes control of the dialogue by first stating that he or she wants to go to a meeting room instead of the office of the person in question, and by giving information about the time he or she is supposed to be there. The program tries to take control of the dialogue by asking the question «Do you want me to take you there?», but allows the user to disregard this question and continue the dialogue in another direction.

2.1.4 Framework for a CUI

Figure 2.1.4.1 shows a basic structure of a CUI. Conversion from audio to text and vice versa is ignored in the figure. User input is sent to the *Dialogue Manager* (DM), which is responsible for

handling the dialogue with the user. The *Language Understanding* module gets the user input as a string and tries to extract the meaning of the text. Based on this information, the DM decides what to do next. If external information is needed, for example the room number of a given person, the DM sends this request to the *External Communications* module which is responsible for communicating with some external module, for example a database. If any information is found, it is sent back to the DM. Based on the language understanding and the external information, the DM can decide how to respond to the user in the best possible way. In the previous dialogue examples, the DM could decide to answer with the name of a person and his or her room number, in addition to ask the user if he or she would like to be taken there by the robot. This «idea» of an answer is then passed to the *Answer Generator*, which formulates a natural language response.

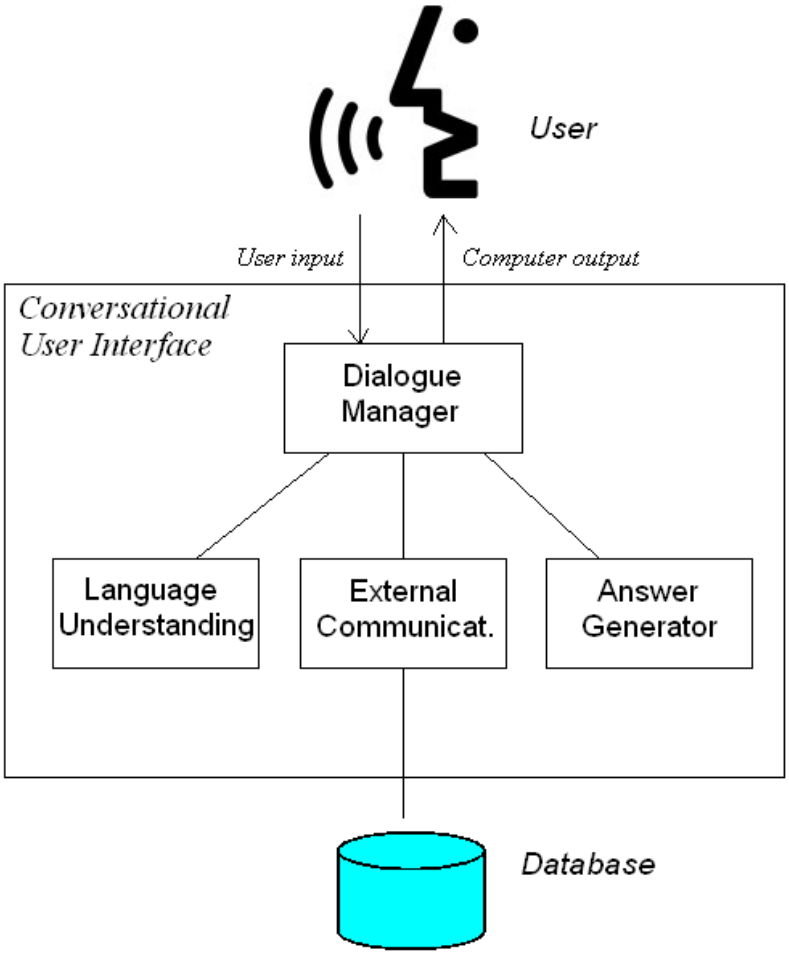

Fig. 2.1.4.1: Framework for a Conversational User Interface.

2.2 Mobile robots

Robots are generally divided into two different classes, industrial robots and mobile robots. The difference between them is mainly that a mobile robot can move around in its environment, while an industrial robot is set at a fixed location. Industrial robots are widely used today, for example on assembly lines in the car industry. Mobile robots are the focus of much research in different universities world-wide, and are also used in space programs, by the military and in industry.

Mobile robots are also found as entertainment customer products, like the Sony AIBO (Figure 2.2.1) and the Lego Mindstorm family. Some of the key challenges for a mobile robot are planning, positioning, obstacle avoidance and verification of locations and mechanical issues regarding motors, wheels, belts, power supply and more.

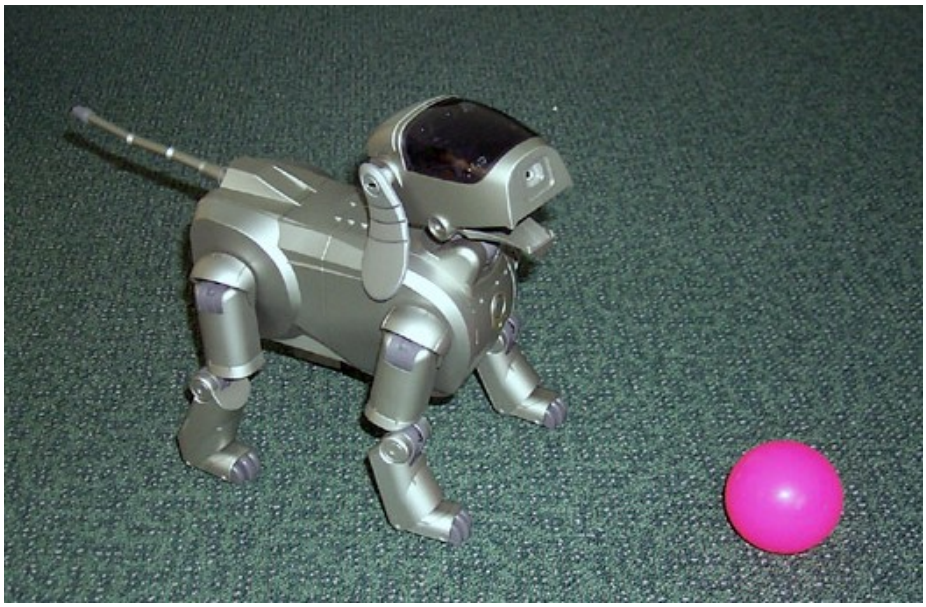

Fig. 2.2.1: Sony AIBO.

2.2.1 Scheduling and planning

To perform various tasks, a mobile robot needs some sort of planning module. For an office delivery robot [Simmons 1997] suggests dividing the planning into two separate modules. The *task scheduling module* determines in which order tasks are to be performed. This is done in order to perform several tasks with the smallest possible travel time. The *path planning module* then decides how to make an efficient journey from one place to another.

A major challenge in path planning is that the environment may be changing and the shortest route may not be the fastest or safest. For an autonomous robot, a good path should be both fast and easy to follow, so that the robot doesn't get lost. Some variation of the **A* search algorithm** is a popular choice for path planning in autonomous robots. Estimated recovery time from navigation error at different places should be taken into consideration when using this search algorithm, as well as expected time used at closed doors and other time consuming events.

2.2.2 Navigation

In order to follow the path given by the *path planning module*, the robot needs to be able to know where it is located at certain times. One way to do this is by identifying certain «landmarks» along the path, so called **landmark-based navigation**. According to [Dixon & Heinlich 1997], the best

way to identify landmarks is by using data vision. The robot needs to have one or more cameras mounted on itself and data vision is used for verifying landmarks. This requires some sort of identification mark placed at certain locations, so called «artificial landmarks».

Another type of landmark navigation is **line navigation**, which can be thought of as a continuous landmark that the robot follows. A straight-forward implementation of this technique is to have white or black lines on the floor that the robot can follow by using a camera. The main drawback of this method is that the robot can not move freely in its environment.

Yet another way of navigation with cameras is to have the cameras placed externally. This way, the cameras can «monitor» the robot and send information back to it about its location.

Readings from light- and distance sensors can also be used to determine the location of a mobile robot, in addition to assumed travel distance from the wheel motors. With this information, the navigation module can create a probability distribution over the current position and direction of the robot. With this kind of model, the robot will never be completely lost, because there will always be a position and direction with higher probability than any other. This method also works well with noisy sensor readings. Such navigation models are often called **map-based navigation**.

2.2.3 Obstacle avoidance

A mobile robot that is supposed to move around in an office environment needs to be able to avoid obstacles. These obstacles can be people walking around, cardboard boxes, furniture, and more. This is a vital challenge for a mobile robot, since collision risk must be at the lowest possible level and collisions must be harmless when they occur (both for the robot and the surroundings!).

A popular way of solving this task is by using sonars, infrareds or laser rangefinders to provide **distance measurements** in the immediate vicinity of the robot. This helps the robot to steer away from close objects as they occur, and the sensors are usually cheap. There are, however, some major drawbacks with these kinds of sensors, as pointed out by [Nourbakhsh 1997]. Sonars have limited range resolution and are sensitive to false echoes by specular reflections, and they can suffer from interference problems with other sonar systems. The same problems occur with infrareds and lasers, in addition to problems with strong radiation and sunlight.

Depth from focus is another approach to obstacle avoidance. The idea is to use several cameras that are grouped closely together, but with different focus adjustments. The scene is divided into regions and the best distance for each region is obtained by the image that provides the best focus. Depth from focus is computationally simple and has proven to be very robust [Nourbakhsh 1997]. Cameras are passive systems and offer many advantages over active ranging systems like sonars, infrareds and lasers. Passive systems have no interference problems and do not suffer from anomalies based on reflection from different textures and colours, which can be problematic for active sensors.

2.2.4 Hardware and reality issues

A major challenge when working with mobile robots is the hardware. As pointed out by [Simmons 1997], a number of failures can be caused by this. Motors need maintenance, sensors and cameras are likely to break down, give noisy information and go out of its intended positions, circuit boards shake loose, etc. When designing software for a robot it is also important to understand the uncertainties of a real robot and create a system that *works* with these uncertainties rather than in a «perfect world» situation.

3 This Project

The project description from Chapter 1.6 reads:

One can imagine a Guiding Robot that is able to interact verbally with its users in order to guide them in a hallway environment. Such a system would use speech recognition, data vision, speech generation and robot movement, in addition to an intelligent system for natural language understanding, reasoning and planning.

The task is to implement an Intelligent Corridor Guide, that can guide a person in a hallway environment in order to find the office of a named person. The system will partly build on already existing modules and a prototype simulator. An important task is to integrate the modules tightly and come up with good solutions for robot movement, planning and identification of locations.

This chapter describes the decisions made during the project with regards to design and implementation of the Corridor Guide Robot demonstrator. These decisions have been made together with the project supervisor Tore Amble, and are based on the information in Chapter 2 – Theoretical Framework.

3.1 Telebuster

It was decided to use Telebuster as a part of the Conversational User Interface in this project. Telebuster is a version of Buster [Fledsberg & Bjerkevoll 1999] that focuses on questions about phone numbers at NTNU and is therefore already connected with a database of personnel at NTNU. This database includes information about offices, which provides what is needed for the Corridor Guide Robot. Telebuster is built on TUC [Amble 2004] (see Chapter 1.8.1), which gives adaptable natural language processing in the TQL language.

Some changes were needed in Telebuster to make it compatible with the domain of corridor guiding. The meaning of words like *«room»*, *«floor»*, *«elevator»*, *«stairs»*, *«entrance», «exit»* and more have been added to the module, and sentences like *«where is that?»* and *«can you take me there?»* are no longer ignored just because they don't fill any slots in Telebuster's information frame.

3.2 Textual interface

Due to lack of time and resources it was decided to use a textual interface for this project. Extending the prototype in this project with a speech interface is the goal of later projects. A speech-based version of Telebuster already exists and should be the basis for such a project. [Johnsen 2003] is a paper about a system-driven spoken dialogue system connected with BusTUC for answering questions about bus routes in Trondheim.

For more information about ongoing work and research with speech interfaces on NTNU, see [Johnsen & Kvale 2005].

3.3 Robot simulation

Because there was no prior real-world robot solution available for the project it was decided to use some sort of software simulation of the mobile robot. Building a simple prototype robot with cheap hardware, like Lego Mindstorm, was considered, but since the project had limited time and resources, and the participants have little or no experience with robotics, it was decided to bias the work more towards other parts of the system.

The model which is going to be simulated is a system with a stationary interface, placed at the entrance of a hall, and an independent robot which can go to specific locations and return back again. A different model would be to have the whole interface mounted on the moving robot, but this would require a much more complex and expensive robot if it is to be physically realised.

The simulator should be able to get path plans from the main system, execute them, and give a

graphical representation of the robot's movement and other important information. The simulated robot should not be implemented in a way that gives it unrealistic information, like it's own position coordinates. A more appropriate way for the simulated robot to figure out it's location would be simulated sensor readings or identification of landmarks.

3.3.1 WSU Khepera Robot Simulator

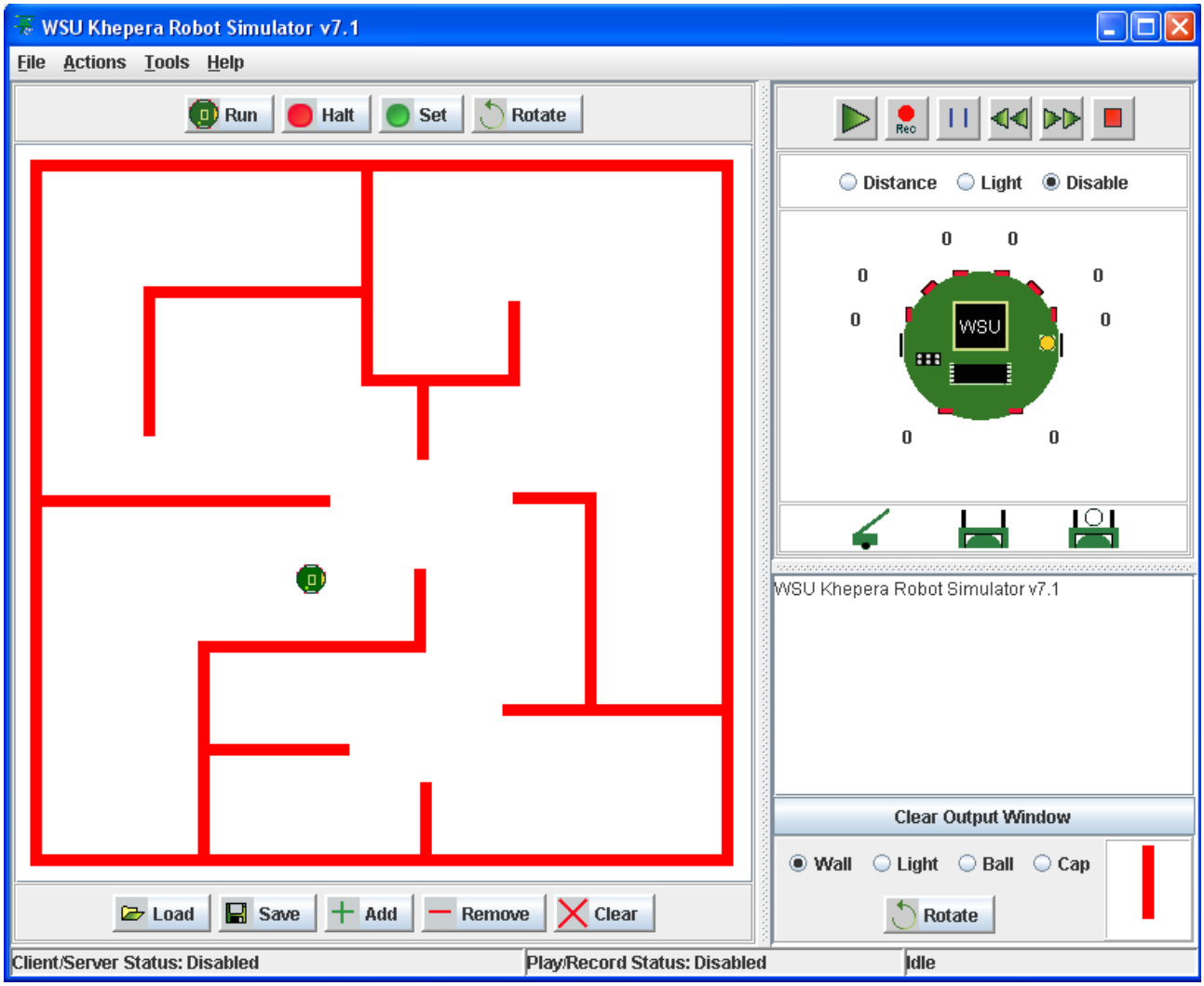

Fig. 3.3.1.1: WSU Khepera Robot Simulator 7.1.

The WSU Khepera Robot Simulator (Figure 3.3.1.1) is a robot simulator software, developed at Wright State University, and written in Java. It aims at simulating the Khepera Robot [Perretta & Gallagher 2002] (Figure 3.3.1.2) which is made by K-TEAM Corporation. The Khepera is a small and compact autonomous mobile robot which is used for experimentation in fields such as navigation, artificial intelligence [Pfeifer & Scheier 1999], evolutionary algorithms [Nolfi & Floreano 2000], multi-agent systems, control, collective behaviour and real-time programming. It is designed to have the same functionality as larger robots. As a Corridor Guide Robot, the Khepera would be to small, with a size of only 70 mm in diameter, and too slow, with an absolute maximum speed of 1 m/s. But the Khepera robot, and the simulator, is suitable for research and demonstration.

At the time of writing, a Khepera II robot without any extensions costs about 1500 Euro.

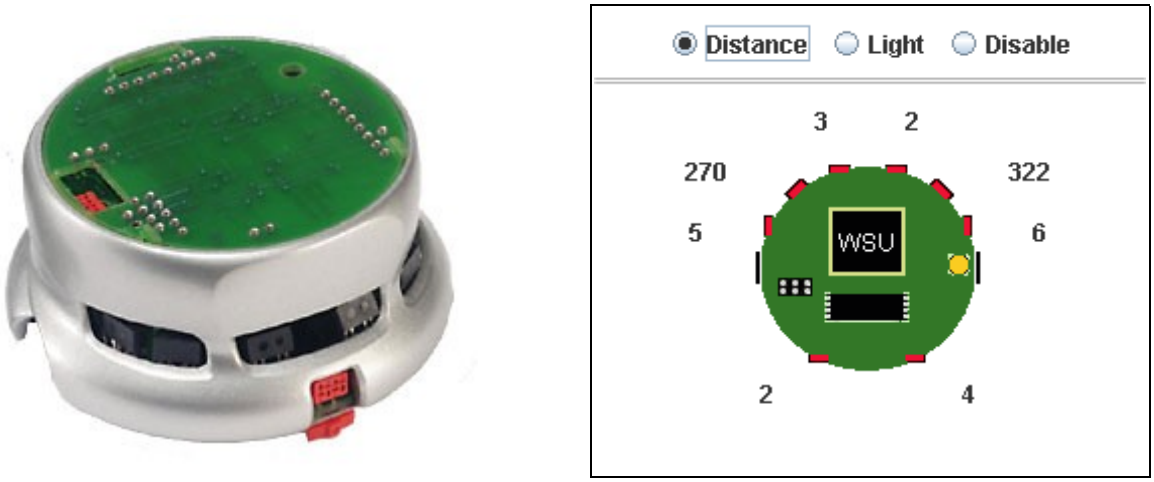

Fig. 3.3.1.2: On the left, a picture of a real Robot Khepera II. On the right, the sensors in the simulator marked with red squares (the front of the robot is up).

The simulated robot has eight distance- and light sensors, which even in the simulator give somewhat noisy readings (Figure 3.3.1.2). It also has travel sensors on the two wheels. Implementation and running of custom robot controllers are straight-forward, and the robot's environment can be created with walls, light sources and small objects.

The main reasons for using WSU Khepera Robot Simulator in this project are:

Building a simulator from scratch is time-consuming

To build a new robot simulator from scratch would be time-consuming and isn't a project goal in the first place. Utilizing an already implemented module that can provide the simulation is a better choice.

Realism

The simulator is fairly realistic. It is built to simulate a real robot with real sensors. Within the simulator, the robot only knows about it's sensors, and has to move around based on only that. The sensors are also noisy, like in a real-world situation.

Customizing is simple

WSU Khepera Robot Simulator is designed in a way that makes design of unique controllers and environments easy. In addition, the simulator is open source, so modifications of the simulator itself is also possible.

3.4 Path planning

Based on the successful results of [Simmons 1997] it was decided to use an A* search algorithm for path planning. Even though the domain of this project (the 3rd floor at the idi building) is relatively small, a naive search would be much too time-consuming. Doors are used as locations in the hallway and are therefore the nodes in the search graph. Distance between adjacent locations are the weighted arcs in the graph. Other possibly time-consuming events are ignored, like estimation of correcting navigation errors, opening closed doors, etc.

The output of the planner should be a series instructions that will take the robot from door to door in order to reach it's goal.

3.5 Extended scope

The scope of the work originally intended for this project states that the prototype should work on the 3rd floor of the idi building at NTNU Gløshaugen. During the project, however, it was decided that the program should also have some knowledge of people and places outside this distinct area. Since Telebuster is used in the CUI, information about all people working at NTNU is already available to the system. It was therefore decided that the program implemented in this project should be able to give a graphical and textual guidance to other buildings and locations within the university. This may also involve suggesting busroutes, which is a part of the Buster system.

3.6 Requirements specification

This section states the requirements specification for the prototype/demonstrator. Requirements are based on the project goals (Chapter 1.6) and the decisions made earlier in this chapter. Requirements are divided into three sub-groups, *Functional Requirements* (FR), *Non-functional Requirements* (NFR) and *User Interface Requirements* (UIR).

3.6.1 Functional Requirements

Requirement no.	***Description***
Conversation	
FR 1	The program shall be able to understand written natural language input, and answer them in an intelligent way using natural language.
FR 2	The program shall be able to answer questions about people working at NTNU regarding offices, phone numbers, addresses and other important information.

Requirement no.	*Description*
FR 3	The program shall be able to describe directions to offices within NTNU, using both text and graphics. For offices in the 3rd floor of the idi building at NTNU Gløshaugen, the directions shall be detailed and simulated robot guiding to the desired locations shall be possible.
FR 4	Dialogue shall be realistic and flexible. The program shall be able to let both the user and the program take control of the dialogue when it is needed.
FR 5	The program shall be able to answer questions about bus routes in Trondheim.
Robot simulation	
FR 6	When the user wants the robot to perform guiding, the program shall be able to make a path plan that the simulated robot can use.
FR 7	The simulated robot shall be able to perform the tasks given by the system within the simulator.
FR 8	Based on sensor readings, the simulated robot shall be able to verify certain «landmarks» along its path.

3.6.2 Non-functional Requirements

Requirement no.	*Description*
NFR 1	The individual modules of the system shall be independent of each other, so that modifications and improvements can easily be made to the different parts of the system.
NFR 2	The program shall be named 'Marvin'.

3.6.3 User Interface Requirements

Requirement no.	*Description*
UIR 1	The program shall have a written natural language interface.
UIR 2	Answers and directions shall be intuitive, and graphics shall be used actively to help the user.
UIR 3	There shall be a graphical representation of the simulated robot's movements.
UIR 4	Useful information from the robot shall be presented to the user.

4 Design

This chapter presents the design of the prototype/demonstrator. The actual implementation is described in Chapter 5. The design is based on the decisions made in Chapter 3, and aims to fulfil the requirements specification in Chapter 3.6.

4.1 Technological Platform

This section describes the technological platform for the design of the program. Some of the modules have already been presented in Chapter 3, but are repeated here. Based on the requirement specification (Chapter 3.6), the technological platform has to support implementation of:

- A Conversational User Interface
- Graphical representations of maps and places
- Database queries about people at NTNU and bus routes
- Path planning
- Communication with a simulated robot
- Visualization of simulated robot movement
- Simulated robot control

The following programming languages and already implemented modules were chosen as the technological platform for this project.

Telebuster

The decision to use Telebuster as part of the Conversational User Interface was explained in Chapter 3.1. Telebuster is a version of Buster [Fledsberg & Bjerkevoll 1999], that focuses on telephone numbers for employees at NTNU. It is implemented in SICStus Prolog, with a Java searcher for the LDAP Database that holds information about the people at NTNU. In this project, Telebuster serves as a natural language processor, providing the system with TQL expressions of the input sentences, and personnel information from the database. Telebuster is under development and needed some adjustments to be used in this project.

Java 2 Platform, Standard Edition (J2SE) version 5.0

Java was chosen as the main programming language for this project. Prior experience with Java and SICStus Prolog connected via the Jasper package has been good. Java is suitable for implementing a Windows-like interface, the Dialogue Manager, graphical representations, path planning algorithm and to communicate with the robot simulator. The WSU Khepera Robot Simulator is itself written in Java, and the custom robot controller to be used by the simulator must be programmed in Java as well.

Jasper and SICStus Prolog 3.12.2 for Windows

SICStus Prolog is used in order to run Telebuster. The Jasper package is used as an interface between Java and SICStus Prolog.

WSU Khepera Robot Simulator 7.1

This Java program is used to simulate a robot on the 3rd floor of the idi building at NTNU Gløshaugen. A modification is needed to get appropriate size of the robot's environment and a texture of the hallway. The simulator will work independently of the rest of the system, but has to have some communication routine with the path planner in order to receive tasks. A complete robot controller class must be made.

4.2 System overview

The system has two main parts: The Conversational User Interface, and the Robot Simulation. The components of each part, and the connection between the components, are shown in Figure 4.2.1. A detailed description of the Conversational User Interface is given in Chapter 4.3, and for the Robot Simulation in Chapter 4.4.

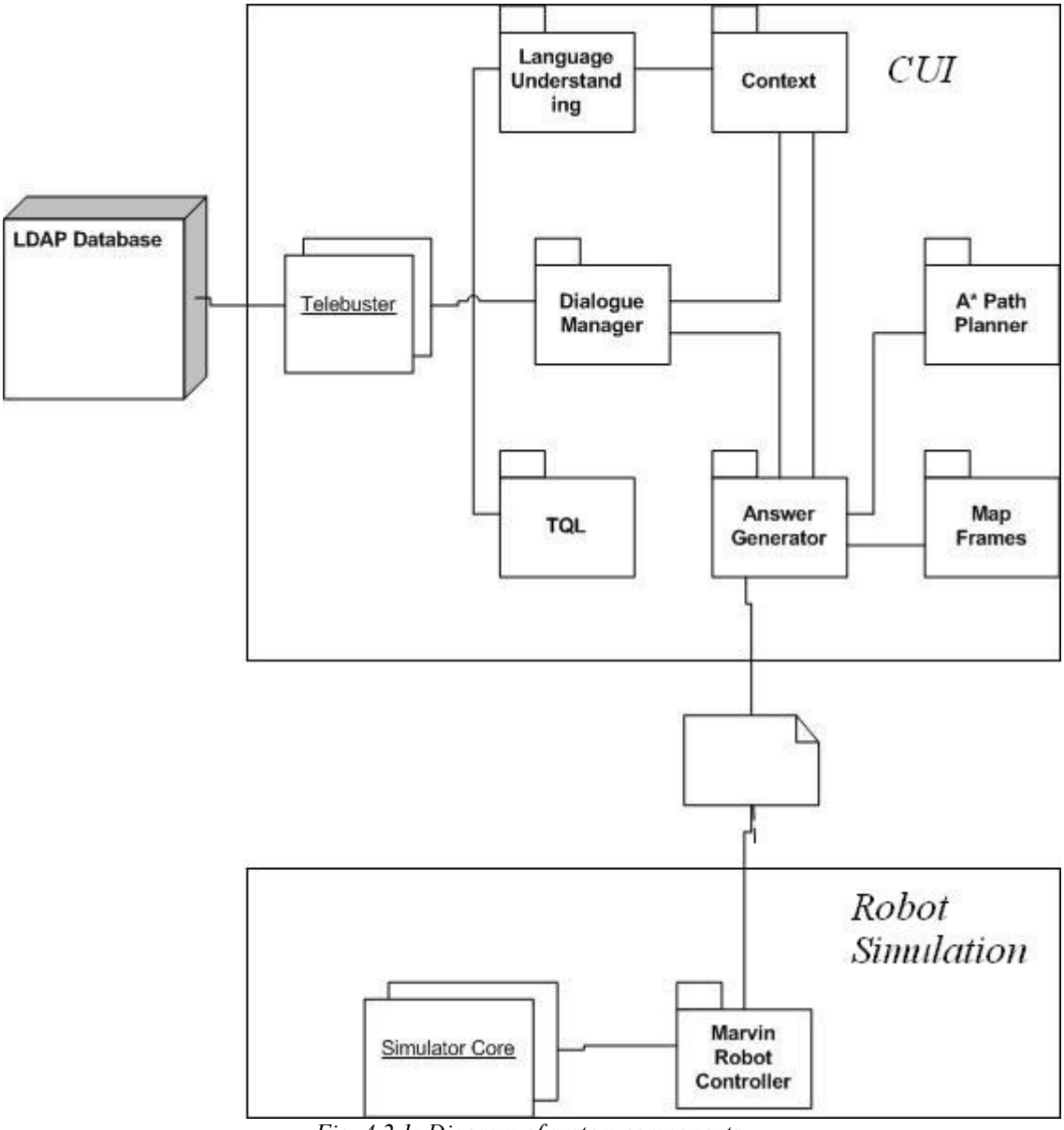

Fig. 4.2.1: Diagram of system components.

4.3 Conversational User Interface

The Conversational User Interface is where the user can input natural language sentences, and receive output as natural language text and graphical explanations. It consists of eigth modules; *Dialogue Manager*, *Telebuster*, *Context*, *TQL*, *Language Understanding*, *Answer Generator*, *A-star Path Planner* and *Map Frame*, as shown in Figure 4.2.1. Some of these modules may consist of more than one class. All the modules are described in this chapter.

4.3.1 Dialogue Manager

The Dialogue Manager is the heart of the Conversational User Interface. A vital task for the Dialogue Manager is to keep track of the state of the dialogue, in order to give the appropriate answers and to ask the user for any missing information. The dialogue flow is shown in Figure 4.3.1.1. The Dialogue Manager also works as a «data flow controller» between the different modules in the CUI.

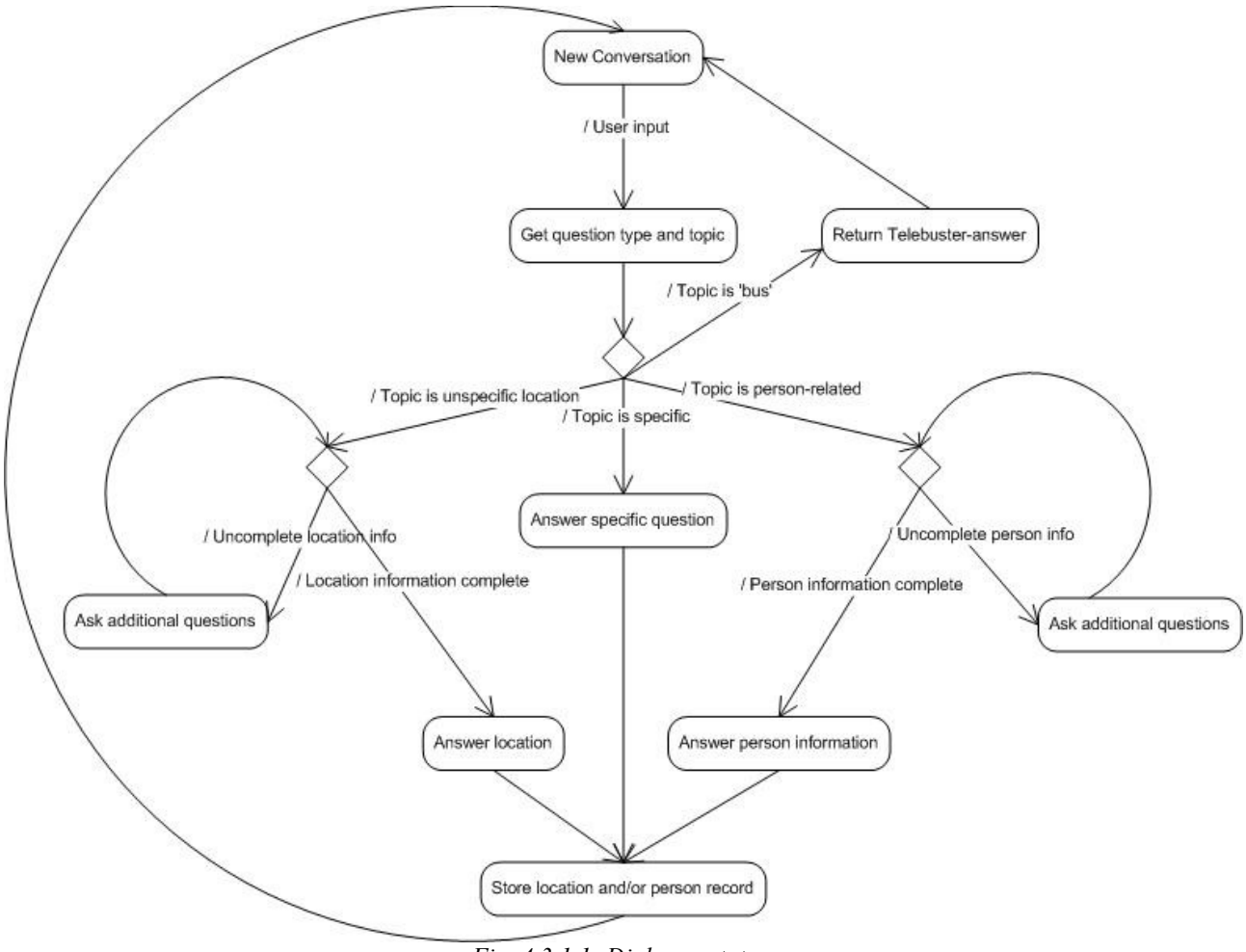

Fig. 4.3.1.1: Dialogue states.

Through ASCII-files, the Dialogue Manager can write input to *Telebuster* and read it's answers. The

Jasper package is used to perform the actual queries. The Dialogue Manager has to formulate the queries and translate the user's input into something that *Telebuster* can understand. Since *Telebuster* is already using a natural language interface, there aren't any adjustments needed to the user's input, except that a sentence must end with a terminating character. Output from Telebuster is read from file and passed along to *Context* and *TQL* for parsing.

Based on the flow of the dialogue, the Dialogue Manager asks the *Answer Generator* to create certain output. It also has to provide the *Answer Generator* with any required information to create these answers. For instance, when the Dialogue Manager reaches the state “Answer person information”, it must provide the answer generator with a record of the person.

To serve as an interface between the system and the user, the *Dialogue Manager* needs to be able to read keyboard inputs and write output to screen.

4.3.2 Telebuster

Telebuster is an already implemented module that is described in Chapter 3.1. Some adjustments have been made to Telebuster in order to make it suitable for this project. Direct queries to Telebuster can be made by using the command *direct_run('inputfile', 'outputfile')*. The input text string in this example is found in the file *inputfile*, and Telebuster writes it's output to the file *outputfile*. A typical direct output from Telebuster is show in Figure 4.3.2.1. Output from Telebuster is divided into three parts.

```
*** TQL  ***


test,yngve isa firstname,tuc isa program,free(3)isa
place,know/id/whether/tuc/free(4)/free(5),event/real/free(5),work/yngv
e/free(6),event/free(4)/free(6),srel/in/place/free(3)/free(6)



*** Context 123 ***


focus:  attributes::sn
frame:
   where ?
   when ?
   day ?
   date ?
   bus ?
   attributes
      spacename ?
      givenname yngve
      sn ?
      telephonenumber ?
      cn ?
      department ?
      title ?
   itemsfound
      items ?
      itemcount 14
   return ?
   language norsk
   topic tele
referents: [[firstname,[yngve]],[program,[tuc]]]
dialognodes: askfor item(sqt)
```

Fig. 4.3.2.1: Example of output from Telebuster's direct_run.

The first part, **TQL,** is Telebuster's translation of the input into a TQL-expression (see Chapter 1.8.1). The TQL-expression is used by the *Language Understanding* module to extract the meaning of the input.

The second part, **Answer**, is the natural language answer from Telebuster. This part is not always present (as seen in the example figure). Since Telebuster is not designed to answer as a Corridor Guide Robot, these answers are mostly ignored by the system. However, if the user asks a bus-related question, the answer from Telebuster is what we want to give the user as output.

The last part of Telebuster's output, **Context**, is the current context in Telebuster. Here we can read what the different information slots in Telebuster's frame currently contain, what information Telebuster would like to get from the user, and also what records have been fetched from the LDAP Database. The LDAP Database is where Telebuster gets information about employees at NTNU. The Context is used by the *Context* module to get focus (what to ask for next) and personnel information. Context also includes topic information, which is used to diverse between person-related answers and bus-related answers.

4.3.3 Context

The Context module gets Telebuster output-strings from the *Dialogue Manager*. It's task is to parse the Context-part of the answer-string, in order to extract information slots, focus, topic, records of people and, when needed, the natural language answer from *Telebuster*. All this information must be stored in a convenient way, so that for example specific person information like room number, department or address is easily available.

4.3.4 TQL

Just like the *Context* module, the TQL module gets Telebuster output-strings from the *Dialogue Manager*. It then parses the contents of the TQL expression into a convenient format in Java. The TQL module also provides the *Language Understanding* module with a set of methods to be used for reasoning over the TQL expression.

The TQL module needs to recognize different parts of a TQL expression. A typical TQL expression is shown in Figure 4.3.4.1. TQL expressions start with a *marker* which describes the class of the sentence. There are five different classes; *Which(x)*, *Test*, *New*, *Do* and *Explain*. Next follows the *body*, which describes the contents of the sentence. The body may have one or more 'is'-relations, which states what some atom stands for, and a set of relations between words and meanings. *Fig.*

```
test
yngve isa firstname
tuc isa program
free(3)isa place
know/id/whether/tuc/free(4)/free(5)
event/real/free(5)
work/yngve/free(6)
event/free(4)/free(6)
srel/in/place/free(3)/free(6)
```

4.3.4.1: Example of a TQL-expression.

4.3.5 Language Understanding

The Language Understanding module is responsible for identifying the meaning of input sentences. Using methods and data structures from the *TQL* module, the Language Understanding module needs to extract the topic that the user is interested in, and the type of question or request the user has given. This information will later be used by the *Dialogue Manager* to control the dialogue flow, and by the Answer Generator for generating answers. An example of identified topic and type from a TQL expression is given in Figure 4.3.5.1. Reducing sentences to a topic and a type is a major simplification, but should be sufficient to answer questions and request in the domain of the robot in a satisfactory way.

Input sentence: *«vet du hvor richard blake sitter?»* (*«do you know where richard blake is sitting?»*)

TQL-expression:

```
test
(richard,blake)isa person
tuc isa program
free(1)isa place
knowthing/tuc/free(1)/free(2)
event/real/free(2)
sit/(richard,blake)/free(3)
srel/in/place/free(1)/free(3)
event/real/free(3)
```

Identified topic and type by LanguageUnderstanding:

```
topicAndType = {«person», «place»}
```

Fig. 4.3.5.1: Input sentence, TQL expression and topic/type identification.

The Language Understanding module must also provide the Dialogue Manager with methods for determining whether a topic is person-related, specific or is about an unspecific location.

4.3.6 Answer Generator

The Answer Generator creates text answers to the user, and additional maps and guidance if required. The *Dialogue Manager* provides the Answer Generator with topic and question type information from the *Language Understanding* module, as well as person records and locations from previous conversations if needed. Based on this, the generator creates a string from different hard-coded answers.

For route plan descriptions, the Answer Generator can use a set of methods provided by the *A* Path Planner*. These methods makes it possible to generate answers like «To get to room X, go down the hall, through door Y on the right, and then through door Z on the left.» Information about buildings are given by the different *Map Frame* modules. By using these, the Answer Generator can provide answers like «The person you asked for has office in building X, which is located at place Y. The map shows you where you currently are and where the building is located.».

The Answer Generator must also decide if the robot is to be given guiding tasks. If the question type and topic suggests that the user wants to be guided to a place on the 3rd floor of the idi building, the generator must call the A* Path Planner to get a path plan and generate instructions that the robot can understand.

4.3.7 A Path Planner*

There are two main tasks for the A* Path Planner module. The first is to create plans for the guiding robot, that is, some set of instructions that the robot can perform in order to go safely from one location to another. The second task is to provide the *Answer Generator* with «important events» in a path that a user would walk along in order to reach a destination. Such events could be «the user must turn right at the next corner», or «the user must go through the door marked X». Both these tasks require that the planner has a precise map representation of the environment.

A convenient way of representing a hallway with a lot of doors is to use these doors as nodes in the search tree and the distance between adjacent doors as arcs. These arcs should not represent distances that are so long, or so «difficult», that the robot may get lost between the two locations. Arcs should be small x- and y-distances that the robot must travel in the 2-dimensional world in order to go from one landmark to another. The planner should also have some kind of landmark information about each location, so that the robot knows how to identify them. By using sensor readings, the robot can detect walls and door openings, so each location should be stored with a direction of the door and whether there are walls in other direction.

4.3.8 Map Frames

Map Frames assist the Answer Generator in creating graphical representations of locations, identifying whether rooms exist on certain floors in the idi building, and where buildings are located. At least three different maps are needed: One for the 3rd floor of the idi building, one for the campus of Gløshaugen, and one for the city of Trondheim. Additional maps could be used to increase details on certain areas, for instance with a map of the campus at Dragvoll, or to display some other type of information, like a map of bus stops for bus information.

4.4 Robot Simulation

The robot simulation aims at simulating a stand-alone autonomous robot that can perform tasks of guiding in the 3rd floor of the idi building. That means that the simulated robot will be able to get path plans to go from a starting location to a goal location, and back again. The simulator is to show the movements of the «robot» graphically, and the WSU Khepera Robot Simulator is chosen to be the core of the robot simulation module, as described in Chapter 3.3.1. The connection between the robot simulator and the Conversational User Interface is shown in Chapter 4.2.

4.4.1 Simulator core

WSU Khepera Robot Simulator is written in Java and is the core of the simulation. The graphical user interface of the original simulator is shown in Figure 3.3.1.1. It contains a menu, a set of buttons for controller execution, an information frame for the sensors on the robot, a set of buttons for replaying simulations, an output text frame, buttons for drawing the world map, and a frame for displaying the simulated world.

Some changes are needed in the core code to make it work smoothly in the system of this project. First of all, a texture is to be displayed in the world frame to give a better graphical representation of the hallway of the 3rd floor of the idi building. The world frame itself needs to be larger in order to make this possible. A map of walls must also be drawn within the program so that the robot will get correct sensor readings when approaching them and stop when bumping into them. The simulator should also load and run Marvin's robot controller automatically when the program starts. In order for the controller to display text messages in the output text frame, a reference from the core system to the controller is needed, which isn't present in the original system.

4.4.2 Robot controller

To control the robot in the simulator, a robot controller must be programmed from scratch. Based on sensor readings, the controller must set the robot's two wheel speeds at all times. Methods must be made for turning and moving the robot, as well as detecting locations and improve accuracy of position on locations. The controller must also communicate with the Conversational User Interface, in order to receive tasks and to give back information about it's whereabouts and current state.

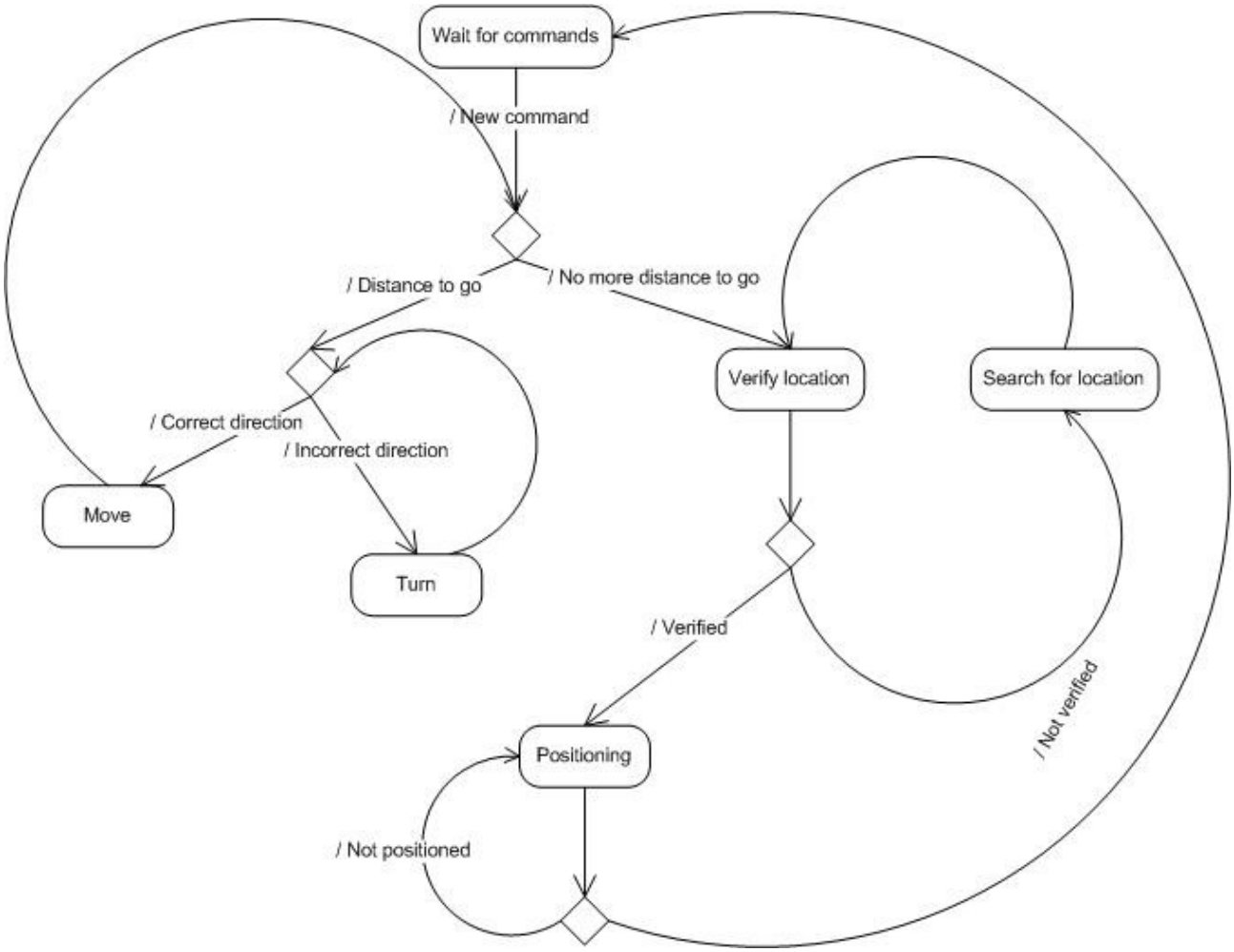

Fig. 4.4.2.1: Robot states.

The control of the robot is state-based. The state-map is shown in Figure 4.4.2.1. These states also have sub-states, which are usually *start*-, *do*-, and *stop* states. The start state typically stores wheel sensor readings and sets goal readings. The do-state is usually looped to continue some kind of behaviour, while the stop state is typically triggered when some goal sensor readings have been reached. An important thing at every sensor-reading-event is to compensate for sensor noise. A good way of doing this is to make several readings over a small period of time and calculate the average value for each sensor. This is more reliable than taking one «snapshot» of the readings, which are likely to be giving noisy values.

Moving in a 2-dimensional environment

Since the world in the simulator is 2-dimensional, the travel distance of a task will be a set of x- and y-distances to go. By knowing it's original direction, the robot can keep track of it's current direction by doing 90-degree-turns and updating the direction between north, south, east and west. In order to perform the task of going some x-distance, the robot needs to turn to either east or west direction (depending on whether the distance is positive or negative), and travel the distance. If obstacles occur, the robot should try to do some avoiding movement. For example, if the robot is supposed to go west but runs into an obstacle, it should try to move a bit in north or south direction before trying to continue west.

Turning

Turning in the WSU Khepera Robot Simulator is done by setting a certain speed to one wheel of the robot and the equivalent negative speed to the other. By monitoring the wheels' travel sensors, a measure can be made of the degrees of turning.

Verifying locations

Since the simulated robot only relies on sensor readings only, the location verification must be based on door openings and surrounding walls. Each location is stored with a set of nearby walls and the direction of the door's opening. A certain location is shown in Figure 4.4.2.2. This particular location will be recognized for having no walls to the north and west, a wall to the east, and the door to the south. By turning towards the door and measuring the front- and corner sensors of the robot (see Figure 4.4.2.3), the robot can sense if there is a small opening ahead, indicating a door. The side- and back sensors indicate if walls are present or not in the other directions.

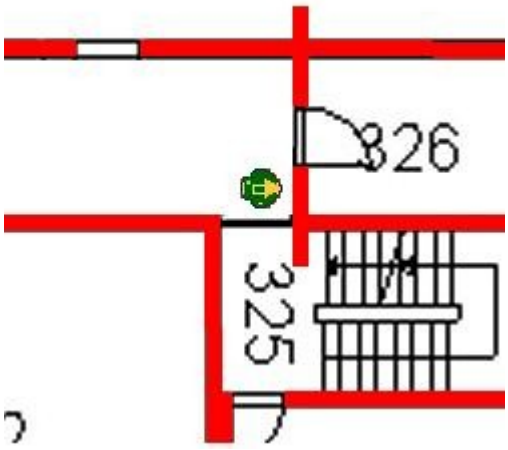

Fig. 4.4.2.2: Robot at location '25' facing east. No wall to the west and north, wall at east and door at south.

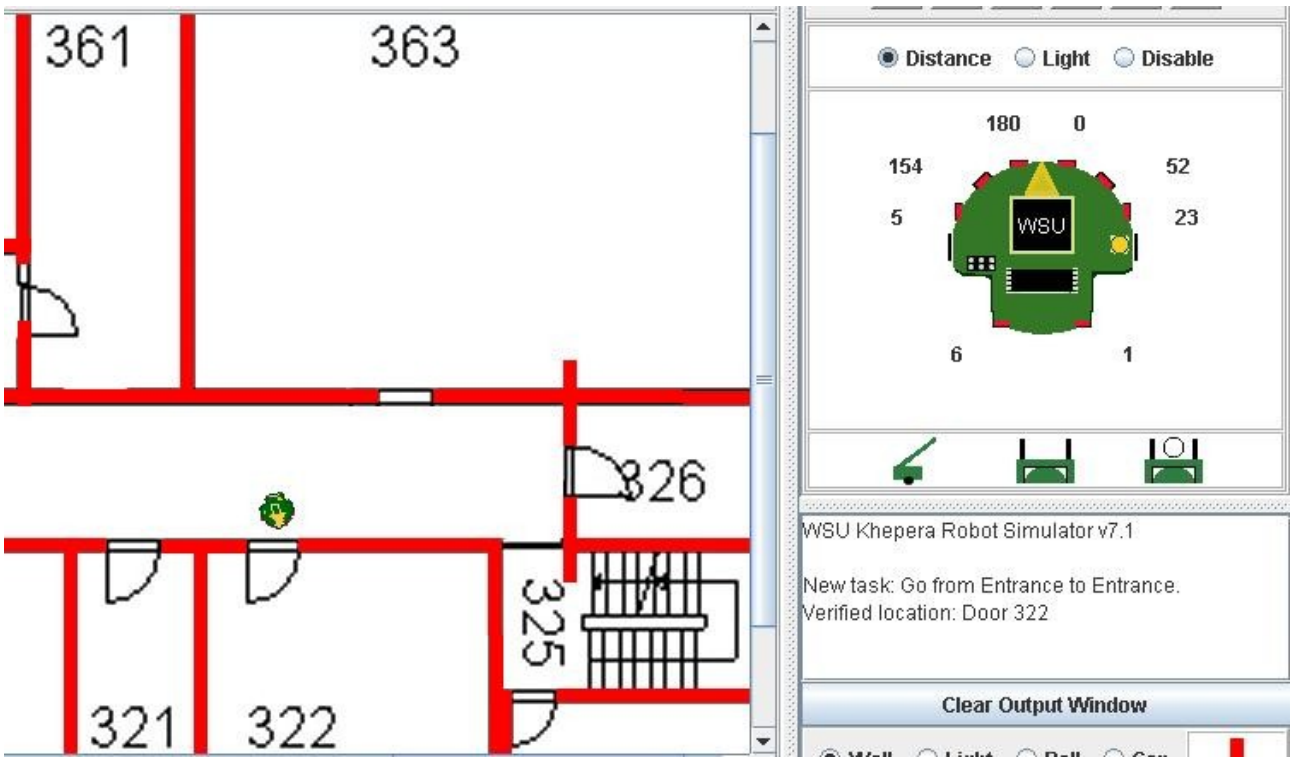

Fig. 4.4.2.3: The robot facing a door (south). Sensor readings on the right. Corner sensors read 154 and 52.

Positioning at locations

In order to prevent accumulation of positioning error from location to location, the robot needs to be as precisely positioned at each location as possible. By doing this, the robot won't have increasing risk of failure based on the length of the journeys, since every journey consists of a set of well-known tasks with a well-known starting position. The main objective for the robot at a specific location is to be placed in the absolute middle of a door opening. This can be done by measuring the difference between the two corner-sensors, and adjusting the robot's position in accordance to this.

5 Implementation

In this chapter the actual implementation of the program is described in detail. The source code for the Conversational User Interface is given in Appendix A, while the code for the Robot Controller is given in Appendix B. Source code for the WSU Khepera Robot Controller is not given in the appendices since it is an already available module. Only small changes had to be made to the code of the simulator. These are explained in Chapter 5.2.1. Every part of the program is programmed in Java, except for the Telebuster module which is programmed in Sicstus Prolog. Code changes in Telebuster to make it suitable for Marvin have been made by Tore Amble.

5.1 Conversational User Interface

Class diagram for the Conversational User Interface is shown in Figure 5.1.1. Note that not all methods are shown in the class diagram. Source code for the complete CUI is given in Appendix A. The most important methods in each class are explained in this Chapter. Not all methods are being described, and the detail of the descriptions vary depending on the importance they play in the program. Some pseudocode is given for important algorithms.

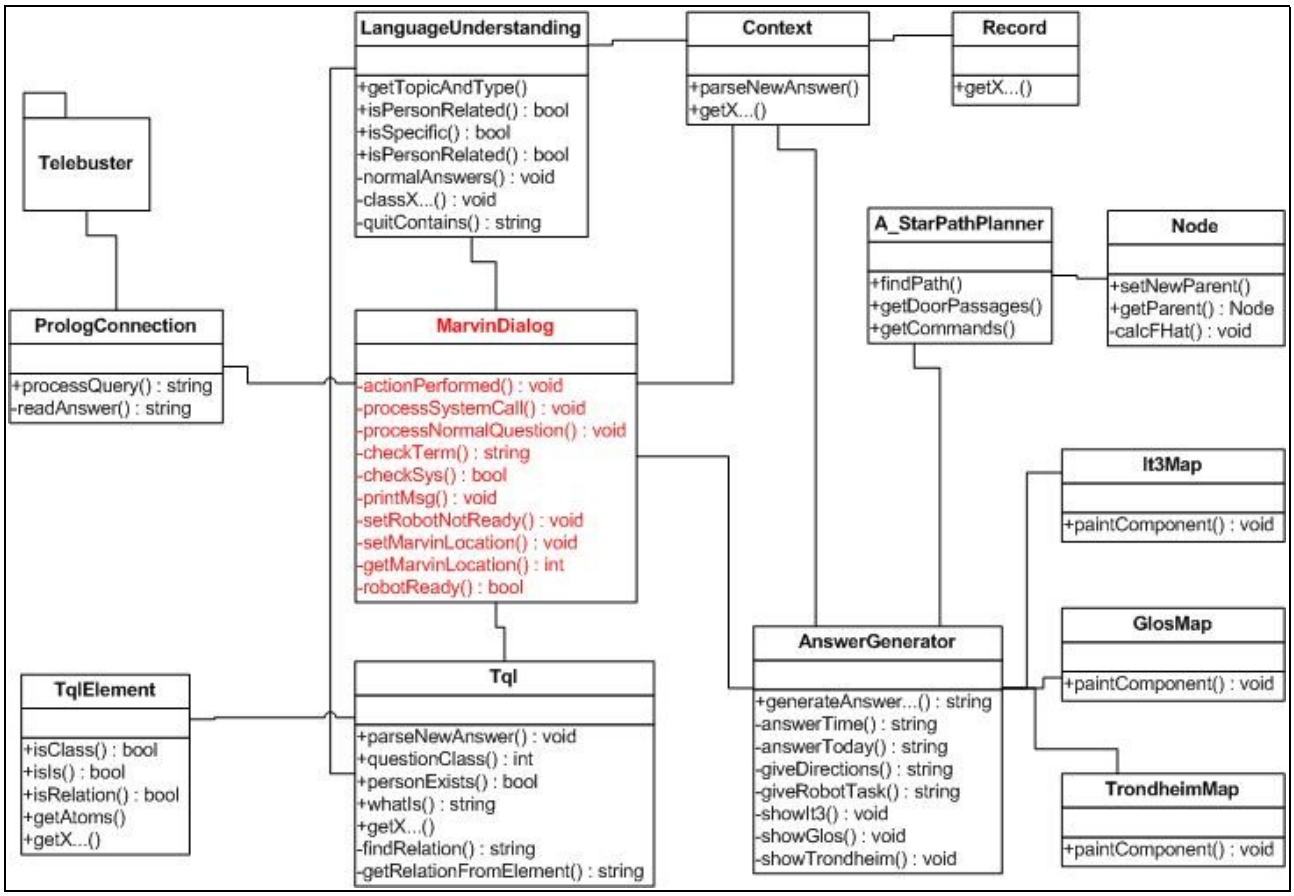

Fig. 5.1.1: Class diagram for the CUI.

5.1.1 MarvinDialog

The MarvinDialog class serves as both a dialog manager and an input/output class for the user. It extends JFrame, in order to create a window that can display text output and receive keyboard input. It also implements ActionListener, to get actions from the input text field. MarvinDialog contains the main method of the program, which simply runs the constructor in MarvinDialog.

Constants

MarvinDialog contains constant Strings for the path to files used by the system. These files are:

- `path`: Used by the CUI to give the robot instructions.
- `marvin`: Used to update information about the robot's location and state.
- `directinput1`: Used to store questions given to Telebuster.
- `directoutput2`: Used to store Telebuster output.
- `telebuster.sav`: The pre-compiled Telebuster Prolog module.

MarvinDialog also contains constant Strings for the different question types. These are *show*, *place*, *go*, *accompany*, *visit*, *routeplan*, *telephone*, *address*, *title*, *department*, *youarewelcome*, *bye* and *default*. Constant integers are also defined for the different TQL sentence classes; *which*, *test*, *new*, *explain* and *do*. Two more sentence classes are also defined for error handling. These are called *dunno* and *no_question*.

Finally, MarvinDialog defines constant integers for the different dialogue states where user input is required (see state diagram in Figure 4.3.1.1): *new_conversation*, *more_information* and *unknown_location*.

Constructor

The constructor in MarvinDialog calls the super-constructor in JFrame to create a window with the title «Marvin». Window size is determined by screen resolution. A JTextArea component for text output is added to the window and a JTextField component for text input is also added. Instances of *Context*, *TQL*, *AnswerGenerator* and *LanguageUnderstanding* are also created and stored in variables. An instance of *PrologConnection* is also created, with a reference to a Prolog object created by using the newProlog() method in the Jasper package.

After these initializations, the location of the Robot is set to the entrance (defined as the integer 25) and the Robot Simulator is started using the Runtime class in Java. MarvinDialog will be ready to take user inputs as soon as the simulator is ready. This is checked by reading the `marvin` file and looking for the contents of «SimulatorReady» which can be set to yes or no.

actionPerformed(ActionEvent e)

This method is called when the user has given an input. If the first character of the input String is a '|'-character, processSystemCall() is called, since this character denotes that the user wants to make a system call. Otherwise, processNormalQuestion() is called in order to start processing a normal user input sentence.

processSystemCall()

This method gets the system call input String from the input JTextField using getText(). The String is parsed using StreamTokenizer from the Java API, and extracts the first sub-string after the initial '|'-character. There are three different system calls available:

- «location» followed by a three-digit number is used to tell the program where the robot is currently located. This is useful if the robot simulation has to be restarted, for instance if the robot gets lost or malfunctions in some way.
- «clear» removes any robot tasks that may be present in the `path` file. Again, this is useful if the robot simulator has malfunctioned in any way.
- «details» turns on and off detailed answers. Detailed answers can be useful for debugging.

If the method does not recognize the system call, it prints "System call not understood" to the output JTextArea.

processNormalQuestion()

When a user enters an input sentence, this method is called, and this is where the dialogue control is implemented. The state-map of the dialogue was shown in Figure 4.3.1.1. Pseudo-code for the method can be written as follows:

```
newText <- getInputFromUser();
telebusterAnswer <- getAnswerFromTelebuster(newText);

if(no answer from telebuster)
      print: «No answer from Telebuster.»;
else
      if(dialogueState == new_conversation)
       getNewTopicAndType();
      else if(dialogueState == more_information)
       getPreviousTopicAndType();
      else if(dialogueState == unknown_location)
       getPreviousType();
       getNewTopic();
      else
       print: «Error.»;

      if(topic is unknown)
       print: «Couldn't understand user question.»;
      else if(topic is bus)
       generateBusAnswer();
       dialogueState <- new_conversation;
      else if(topic is about person)
       if(person record in context)
             generatePersonAnswerFromRecord();
             storeRecordAndLocation();
```

```
        dialogueState = new_conversation;
    else if(context has missing person information)
        generateAnswerBasedOnFocus();
        storeTopicAndType();
        dialogueState = more_information;
    else if(previous record exists)
        generateAnswerBasedOnPreviousRecord();
        storeLocation();
        dialogueState = new_conversation;
    else
        print: «Error.»;
  else if(topic is about unspecific location)
    if(previous location exists)
        generateAnswerFromPreviousLocation();
        dialogueState = new_conversation();
    else
        print: «Where do you mean?»;
        storeTopicAndType();
        dialogueState = unknown_location();
  else if(topic is about something specific)
    generateSpecificAnswer();
    if(location in answer)
        storeLocation();
    dialogueState = new_conversation;
```

A small example will clarify how the code works.

The user inputs the following sentence:

«***Sitter Richard Blake i denne etasjen?***» («Does Richard Blake sit on this floor?»)

First of all, the input has to be made into a query string for Telebuster. This example input would be turned into the string *query('nor 123 «Sitter Richard Blake i denne etasjen?»)*. After getting the Telebuster answer the topic and type will be fetched from *LanguageUnderstanding*, since the original dialogue state is new_conversation. The topic is person-related (in fact, the topic in this example is *person*) and there is a person record in context (Richard Blake's record), so the Answer Generator will be called to generate an answer based on topic, type and the record in context. The record is then stored. Since the answer will be including the location of Richard Blake's office, this location will also be stored.

Next, the user enters this sentence:

«***Hva er telefonnummeret hans?***» («What's his phone number?»)

The dialogue is still in new_conversation state, so the sentence is handled in the same way as the previous sentence until topic is checked. This time, we also have a person-related topic (only people can have phone numbers), but we don't have a person record in context. We also don't have missing person information (in fact, we don't have person information at all), but we do have a stored person record from previous conversation. Answer Generator is then called to generate an answer based on topic, type, and the previously stored record, which will result in answering Richard Blake's phone number.

Now, let's have the user input this:

«***Hva er adressen til Tore?***» («What is the address of Tore?»)

This sentence behaves much like the previous, only this time it will get a positive check on «context has missing person information», since the context will contain a known first name, an unknown last name and no person record. The context will also contain a focus (information Telebuster wants), so the Answer Generator will be asked to generate an answer based on this focus, which in this case will be the person's last name. It will be something like «Do you know the last name?». Next, the topic and type will be stored and the dialogue state is set to more_information.

Following Marvin's wish, the user enters a last name:

«***Amble***»

Since we are in more_information state, the stored topic and type is loaded. We then move on to find that the topic (which is still «address») is person-related. Now, we have a person record in context, so we'll ask Answer Generator to use this to answer the topic. Then, we store the record and set the state back to new_conversation.

These examples don't cover the whole algorithm, but shows in thick lines how it works. The unknown_location state is entered when a topic is reached that refers to a location and there is no specific location information either in the user's sentence or in previously stored answers. Specific topics are topics that only have specific answers, like «clock». When a user ask about «clock», it is presumed that the user wants to know what time it is, and not the location of some clock!

checkTerm(String s)

This method is needed to ensure that sentences given to Telebuster have a termination character, that is «.», «!» or «?». If a termination character is not present, the method simply adds a «.» to the end of the string before returning it.

checkSys(String s)

This method checks if the string is a system call, that is, if the string starts with a '|' character. It returns a boolean value.

printMsg(String s, boolean b)

This method prints the string to the output JTextArea. It uses the Calender class in Java to create a time stamp at the beginning of the string. The boolean value is used to decide whether a «Marvin>» or «User>» stamp should be added as well. There is also an equivalent printMsg method without the boolean value, which prints the string without any stamps.

setRobotNotReady()

This method sets the «SimulatorReady» tag in the `marvin` file to «no». It is called in the contructor before the simulator has been started (since the simulator cannot be ready when it's not been started yet!). The robot simulator will change this value to “yes” after it has been initialized.

setMarvinLocation(int location)

Updates the «Location» tag in the `marvin` file according to the given integer.

getMarvinLocation()

Reads the value of the «Location» tag in the `marvin` file and returns it as an integer.

robotReady()

This method returns a boolean value based on the value in the «SimulatorReady» tag in the `marvin` file.

5.1.2 PrologConnection

This class is used to perform queries on Telebuster, and read the answers. It uses the Jasper package to communicate with Telebuster, which is a SICStus Prolog program. Jasper package documentation is available at [http://www.sics.se/sicstus/docs/3.12.4/html/jasper/se/sics/jasper/SICStus.html]. For more about Telebuster, see Chapter 3.1 and 4.3.2.

Constructor

The constructor gets four inputs: A reference to the Prolog object created in MarvinDialog, a reference back to MarvinDialog, a string with the path to the input question file which is going to be used by Telebuster, and a string to the answer file where Telebuster should write it's answers. It uses the File class in Java to get the canonical paths to the files.

processQuery(String s)

This is the method that is used to input sentences to Telebuster, and return the answer given by

Telebuster. The pseudocode for this method is given below.

```
Write String s to questionFile;
Write default answer to answerFile;
Perform query on Telebuster;
Read answer from answerFile;
return answer;
```

Writing and reading from files is straight-forward. The query()-method is called in the Prolog-object to perform the query on Telebuster. The query is always the same: *direct_run('inputfile', 'outputfile')* , where input- and output-file were given when the PrologConnection object was created. After this query has been performed, the answer can be read from the answer file using the method *readAnswer()*. This method simply returns the contents of the *outputfile* as a String.

5.1.3 Context

This class is responsible for parsing the context field in a Telebuster answer. A typical context field is shown in Figure 5.1.3.1. The fields that the Context class must extract are *firstname*, *lastname* and *focus*, as well as a boolean value indicating if the topic is «bus» and, in that case, Telebuster's natural language answer as a String. In addition, a Record object must be created in order to parse the personnel record field in the context.

```
*** Context 123 ***


focus:  []
frame:
    where ?
    when ?
    day ?
    date ?
    bus ?
    attributes
        spacename ?
        givenname yngve
        sn dahl
        telephonenumber ?
        cn ?
        department ?
        title ?
    itemsfound
        items
                record([ou::NTNU,ou::Fakultet for informasjonsteknologi\,
matem.    og    elektroteknikk,ou::Institutt    for    datateknikk    og
informasjonsvitenskap,cn::Yngve
Dahl,telephonenumber::91453,mail::yngveda@idi.ntnu.no,roomnumber::IT-
bygget*057,givenname::Yngve,sn::Dahl,title::Stipendiat,street::Sem
Sælands v 7-9])
        itemcount 1
    return
        roomnumber
    language norsk
    topic tele
referents:                                                  [[room,[IT-
bygget*057]],[firstname,[yngve]],[lastname,[dahl]]]
dialognodes: uiqRepl2 item(sat)
```

Fig. 5.1.3.1: Context field.

parseNewAnswer(String s)

The input string to this method is the complete answer from Telebuster, so the first thing the parser must do is to locate the start of the context field. This is marked with «*** Context ***» in Telebuster's answers. Pseudocode for the parser follows.

```
Find context area;
token <- getNextToken();
while(in context field)
      if(in topic field)
       if(topic is bus)
```

```
            set bus = true;
            answerString <- getTelebusterAnswerString();
      if(in attributes field)
       find and store givenname;
       find and store sn;
      token <- getNextToken();
create new Record object;
```

In Telebuster's context, `givenname` is the notation for first name and `sn` is the notation for last name. Parsing is straight-forward in Java, using the StreamTokenizer and StringTokenizer classes.

There are also a number of getter and setter methods in the Context class. In addition, there is a boolean method called missingInformation(), which returns true if *either* first name or last name is `null`, and the focus field contains information. This method is used by the Dialogue Manager to see if the program should ask the user for some missing information.

5.1.4 Record

This class is used to parse the person record field in a Telebuster answer. The information that is stored is *full name*, *department*, *building*, *room number*, *title* and *street* (address).

Constructor

The parsing is done from the constructor, which takes a Telebuster answer as a String and calls different extract-methods for the different information fields that it is interested in. Each extract-method stores an information in the appropriate variable. Parsing is straight-forward using Java's StreamTokenizer class.

The Record class also implements *cloneable*, in order for the Dialogue Manager to store copies of old records.

5.1.5 Tql

The Tql class is one of the most important and complex classes in this program. It parses the TQL expression given by Telebuster and provides a set of methods for the Language Understanding module, to be used when reasoning over the expression.

The TqlElement class

The Tql class uses this class to store a TQL expression as an ArrayList of elements. For instance,

one element may be *free(1) isa place*, and another may be *srel/in/place/free(1)/free(2)*. Each TqlElement object stores it's contents in an ArrayList of «atoms». In the previous example, the atoms would be *srel*, *in*, *place*, *free(1)* and *free(2)*. This creates a hiearchical structure, with a Tql object on top, containing a list of TqlElements, each of which contains one or more atoms.

The TqlElement object provides methods for getting specific atoms or the whole list of atoms. In addition, the TqlElement holds an integer value based on what kind of element it is. There are three different types of elements, *class (or "marker")*, *is* and *relation*.

parseNewAnswer(String s)

The parser gets the Telebuster answer as a String. After locating itself at the beginning of the TQL field, marked with «*** TQL ***», it stores each TQL element as TqlElement objects and stores them in an ArrayList. Pseudocode for the parser is given below.

```
Find TQL field;
Find first TQL element;
for(all elements)
      create new TqlElement;
      elementList.add(TqlElement);
```

When TqlElement classes are created, they are given the element as a String, and it is up to the TqlElement object to parse it into atoms and store the atoms in a list (see Figure 5.1.4.1).

questionClass()

This method returns the class of the TQL expression as a constant integer defined in MarvinDialog, representing the classes *which*, *test*, *new*, *do* and *explain*. This is done simply by reading the first element of the TQL expression, since it always denotes the sentence class.

personExists()

Checks if there is information about some person in the TQL expression. This is done by looking at the last atom of all the 'is'-relations and see if any of them equals *person*, *firstname* or *lastname*. There is also an equivalent roomExists()-method, that sees if there is room-information in the TQL expression.

whatIs(String s)

This method is commonly used by the LanguageUnderstanding class to track relations from some term to another. For instance, if the class of the TQL expression is *which(free(1))*, LanguageUnderstanding can call *whatIs(«free(1)»)* to figure out what the term free(1) stands for. The whatIs(String s)-method uses another method, findRelations(), to find relations between terms. findRelations() can also call another helping method, getRelationFromElement(), which has a set of rules for relations between the different atoms in relation-elements like *be1*, *be2*, *sit*, *live*, *has*, *srel*, *nrel*, and also some general rules for other relations. There are a lot of special cases when tracking

relations, so no pseudocode will be given in this section. The Java code itself can be found in Appendix A.

«get» methods

There are a number of «get» methods in the TQL class, including *getWhichContent()*, *getFirstRelationType()*, *getRoom()*, *getThirdAtom()*, etc. These methods are used by both LanguageUnderstanding and AnswerGenerator to extract various information from the TQL expression.

5.1.6 LanguageUnderstanding

The LanguageUnderstanding class has one main task: To find a TQL expression's topic and type. To do this, LanguageUnderstanding is provided with a set of methods from the Tql class, as described in Chapter 5.1.5. It also needs to know if the topic is person-related.

Constructor

The LanguageUnderstanding class gets references to the Context and the Tql when it is constructed. These are stored in variables `c` and `tql` and are used in almost every method in the class.

getTopicAndType()

This is the method that is called from the Dialogue Manager to get the topic and type of an input sentence. It returns the topic and type as an array of two Strings. The method itself is very simple. It checks the sentence class and calls the appropriate private method depending on the class. For instance, a *which(x)* sentence will lead to a call to *classWhich()*. The reason for doing this is that each sentence class requires somewhat different methods to find topic and type.

Which-sentences are the most simple to analyse. A which-sentence always has a content, for instance *which(free(1))*, where *free(1)* is the content. All we need to do is to identify what the content of the which-sentence is. A sentence like *«Hvor er Tore Amble?»* (*«Where is Tore Amble?»*) yields the TQL expression:

which(free(1))

(tore,amble)isa person

free(1)isa place,

be1/(tore,amble)/free(2)

srel/in/place/free(1)/free(2)

event/real/free(2)

By using the whatIs()-method in the Tql class, LanguageUnderstanding will be told that the content

of which, free(1), is a *place*. It will then call the recursive method **normalAnswers(String w)**. This method will be explained in detail later in this chapter, but for now it's sufficient to know that the purpose of the method is to continue tracking down relations in the TQL expression, and store type and topic as we move along. The example above will yield *person* as topic and *place* as type. First, *free(1)* is identified as *place*, then *place* is related to *free(2)* and finally *free(2)* is related to *(tore,amble)* which is a *person*.

The other sentence classes work in somewhat the same was as which-sentences. First, we identify what the question or request is about and then we track down the type and topic throughout the TQL expression by recursive calls to *normalAnswers(String w)*. In **test, new, do** and **explain**-sentences, we start with the unknown atom from the first relation-element. There are some special cases where we have to skip the first relation-element in order to find the element that provides useful information. There is also a special case of do-sentences, where the only other element is *quit(x)*. In this case, «quit» is stored are topic and the content of quit is stored as type.

normalAnswers(String w)

This is a recursive method that tracks an atom through a TQL expression until either a definite topic has been found or a dead end has been reached. The pseudocode is given below:

```
normalAnswers(String w)
       if(w is null or «empty»)
       if(person information in TQL)
             topic <- «person»;
       else if(room(number) information in TQL)
             topic <- «roomnumber»;
       else if(room(free(x)) information in TQL)
             topic <- «roomfree»;
       else
             topic <- «location»;
      else if(w is «place»)
       type <- «place»;
       call recursive: normalAnswers(whatIs(w));
      else if(w is «routeplan» or «street»)
       type <- «routeplan»;
       call recursive: normalAnswers(whatIs(w));
      else if(w is «room»)
       if(room has a number)
             topic <- «roomnumber»;
       else if(room has a free(x))
             topic <- «roomfree»;
      else if(w is «telephone» or «address» or «title» or «department»)
       topic <- w;
```

```
type <- w;
else if(w is specific topic or person-related topic)
 topic <- w;
else
 if(w not in is-relation)
       topic <- «unknown»;
 else
       call recursive: normalAnswers(isRelation(w));
```

At the end, the method returns the the topic and type variables. The pseudocode can be somewhat confusing to follow, but in general it continues tracking down relations until it reaches a topic that is definite, that is, a topic that indisputably decides what the essence of the sentence is. On the way to this topic, different types may be stored in order to find out what the user wants to know about the topic. We have already seen this in an example, where the user wanted to know about a *person*'s (topic) *place* (type).

The different question types are defined as constant Strings in MarvinDialog. They are: *show*, *place*, *go*, *accompany*, *routeplan*, *telephone*, *address*, *title*, *department*, *youarewelcome*, *bye* and *default,* and more. Other types could easily be added, for instance if we wanted to differ between questions about meetings and visits, we could add *meet* and *visit* as types and make LanguageUnderstanding set these types when it tracks through a meet- or visit relation. Then, the Answer Generator must be programmed to differ between meet, visit and other question types. For instance, we could program the generator to answer meeting questions with «You can meet that person at room xxx» and visit questions with «You can visit that person at room xxx». However, for the prototype Marvin program, the types mentioned in the beginning of the paragraph are sufficient to fulfil the requirements. For questions about locations we are mainly interested in whether robot guidance is wanted or not. Typical question types where guiding is wanted are *go* and *accompany.*

Recognized question topics are defined in three lists in LanguageUnderstanding: *personRelatedTopics*, *specificTopics* and *unspecificLocation*. Currently, the person related topics are *person*, *roomfree*, *firstname*, *lastname*, *agent*, *man*, *woman*, *telephone*, *address*, *title* and *department*. Specific topics are *roomnumber*, *marvin*, *robot*, *tuc*, *I*, *self*, *exit*, *entry*, *stair*, *lift*, *toilet*, *clock*, *day*, *date*, *time*, *quit*. These topics are defined as «specific» because they are independent of context and person records. The final group of topics, unspecificLocation has only one topic, called *location*. This topic is set for every sentence that asks about a location with no reference, like «Hvor er det?» («Where is that?»). Each group has a corresponding boolean method that can be used by MarvinDialog to see what kind of topic we are dealing with. These methods are **isPersonRelated()**, **isSpecific()** and **isUnspecificLocation()**.

5.1.7 AnswerGenerator

The Answer Generator is called by MarvinDialog to generate answers based on the question topic and question type. Not only must the Answer Generator create a text output (which may consist of

one or more sentences), but it should also call Map Frames if needed, or give the Robot guiding tasks.

Constructor

The constructor in AnswerGenerator takes references to the MarvinDialog-, Context- and Tql objects and stores them in variables `md`, `c` and `tql`. It then creates an A_StarPathPlanner object and initializes a map of location descriptions.

generateAnswer...()

The generator has five general methods that can create any answer that the Dialogue Manger needs. These are: **generatePersonAnswer()**, which creates answers about people, their locations or other attributes. **generateBusAnswer()**, which just returns Telebuster's natural language answer of a bus question. **generateFocusAnswer()**, which generates a question to the user asking for an additional information given in the focus field of Telebuster's context. **generateAnswerFromLocation()**, which creates an answer based on a previously asked location. And finally, **generateSpecificAnswer()**, which returns answers about specific topics, for example *clock* or *toilet*. All these methods work in the same way (except the bus answer generator, which just returns an already generated answer). They take topic and type as input, and additional information like previously stored location or person record, and choose answers based on topic and type. If the topic and type requires showing maps, the generator can create these with methods **showIt3()**, **showGlos()** and **showTrondheim()**. Robot guiding and textual description of path to a location is achieved by calling **giveDirections(String type, String location)**.

All the generateAnswer-methods return a textual answer as a String back to MarvinDialog.

answerTime()

This is a helping method for the answer generating methods, returning a String about the current time.

answerToday()

This is another helping method, returning a String about the current date.

giveDirections(String type, String location)

This method is called by the generateAnswer-methods in order to answer questions about locations on the 3rd floor of the idi building. Pseudocode for *giveDirections()* is given below:

```
if(question about entry)
      call showEntry();
else
      if(type requires explaining the user how to get to location)
       roomCoords <- get room coordinates from a_star;
```

```
if(roomCoords.y > 180)
      print: «The room is down the hall to the right»;
else
      print: «The room is down the hall to the left»;
doorsList <- get doors needed to pass from a_star;
for(each door)
      print: «, through door » + door.number;
call showIt3();
else if(type requires robot guiding)
call giveRobotTask();
```

In the pseudocode above, *a_star* denotes the A_StarPathPlanner, which has detailed information about the 3rd floor of the idi building. As seen in the code above, all path descriptions will be of the form «Go down to the left/right, through door X, through door Y, ...». This is sufficient for the 3rd floor of the idi building, but more sophisticated generators would be needed for more complex building structures.

giveRobotTask(String s)

This is actually the only method that sends information from the Conversational User Interface to the Robot Simulation. It calls the A_StarPathPlanner to get the path plan and writes it to `path` file in a format that the Robot Controller can understand. A typical path plan written to file is shown in Figure 5.1.7.1.

```
From=25,To=22,{-2183,0,0,0,0,2}
From=22,To=21,{-1119,0,0,0,0,2}
From=21,To=20,{-1567,0,0,0,0,2}
From=20,To=601,{0,-699,0,2,0,0}
From=601,To=60,{-363,-1203,0,2,0,0}
From=60,To=601,{363,1203,0,2,0,0}
From=601,To=20,{0,699,0,0,0,2}
From=20,To=21,{1567,0,0,0,0,2}
From=21,To=22,{1119,0,0,0,0,2}
From=22,To=25,{2183,0,0,0,1,2}
```

Fig. 5.1.7.1: A typical path plan for the robot. This is a plan for going from the entrance to room 360 and back again.

5.1.8 A_StarPathPlanner

As the name of the class indicates, the A_StarPathPlanner uses an A* search algorithm to find paths in the 3rd floor of the idi building. Rooms are stored in one array with number, location and surroundings (walls and door direction), and arcs (the «path» between doors) are stored in another

array. The weighting of the arcs is done in the simplest possible way, with «1» denoting a normal path and «99» denoting a path that requires going through a door.

The Node class

Node is a helping class used in the A* search. It contains a pointer to the parent node and the coordinates of the room it represents. The A* algorithm is a heuristic search, so the Node class contains a method for calculating the f-hat value, called **calcFHat()**. It calculates the air-distance between itself and the parent-node, which is a simple but good estimation for path planning. The Node class also contains a number of get- and set methods.

findPath(int start, int goal)

This is the most important method in the A_StarPathPlanner class. It returns a list of location numbers that goes from *start* to *goal*. To find this path, it uses an A* search as described in [Nilsson 1998 – page 141-145]

getDoorPassages(ArrayList l, int start, int end)

This is the method used by AnswerGenerator to make textual descriptions of paths. It returns a list of doors that needs to be passed to get from *start* to *end*. The ArrayList *l* is a path plan, as made by *findPath()*. The method compares every part of the path with the array of arcs, and adds to the list if an arc has «99» as cost.

getCommands(ArrayList l, int start, int end)

This method also takes a path as input, in the format that *findPath()* returns. It creates an ArrayList of commands for the Robot. Each command is stored as an array of eight integers, representing *x-distance*, *y-distance*, *from-room*, *to-room*, *wall(west)*, *wall(north)*, *wall(east)*, *wall(south)*. These values are gathered one-by-one from the array of rooms. The x-distance and y-distance are the distances that the robot is supposed to travel for each command, and must therefore be multiplied by a constant to approximate the equivalent wheel sensor's travel. This constant has been set by trial-and-error testing, and the value of 27.99 times pixel map values is working well. This constant doesn't need to be 100% accurate, since the robot will always try independently to position itself exactly at locations when they are reached. Walls are represented as *0* (no wall), *1* (wall) and *2* (door).

5.1.9 Map Frames

The map frames are used to give a graphical representation of locations. There are three different map frame classes. **It3Map** is used to show locations on the 3rd floor of the idi building. **GlosMap** is used to show buildings on the Gløshaugen campus. **TrondheimMap** is used to show locations in the city of Trondheim. Map frames are called from the Answer Generator. They all share the same basic structure.

Constructor

The map frames all extend JLabel and uses the super constructor. The super constructor is given an ImageIcon object as input, created with a jpeg image of the map. The rest of the constructor draws start- and goal locations. They take a location as input and use an arrays of names and coordinates to get the coordinates of the location to be shown. The «you are here» location is described with a dot (Ellipse2D), a line (Line2D) and a text. Goal location is shown with an ellipse, a line and a text. A TrondheimMap map frame is shown in Figure 5.1.9.1.

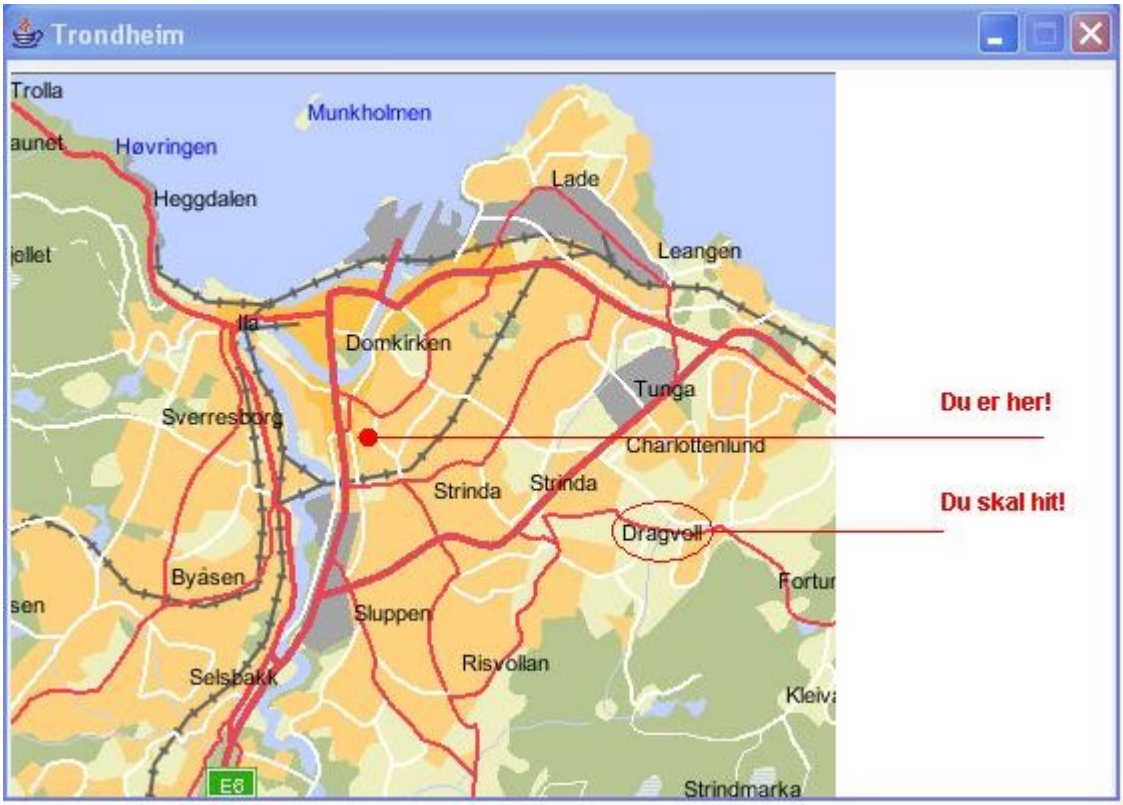

Fig. 5.1.9.1: A map frame showing the city of Trondheim. Goal location is at Dragvoll.

5.2 Robot Simulation

The «simulator core» is the WSU Khepera Robot Simulator 7.1. This simulator is programmed in Java and consists of 33 classes. Small, but important, changes have been made to the core code. These changes are described in Chapter 5.2.1. The rest of the simulator core is not described in this paper, as it is an already developed module. Detailed information about the simulator can be found in [Perretta & Gallagher 2002].

The Robot Controller is an independent part of the simulator, and has been coded from scratch for this project. It is described in detail in Chapter 5.2.2, and the Java code is given in Appendix B. The Robot Controller is responsible for all robot movement and decisions based on sensor readings.

5.2.1 Simulator Core

A screen shot from the original WSU Khepera Robot Simulator 7.1 was shown in Figure 3.3.1.1. Figure 5.2.1.1 shows a similar screen shot from the simulator after modifications were made for this project. The most notable change is the size and look of the «world», which has been increased from 500x500 pixels to 2560x600 pixels, and given an image to show the map of the 3rd floor of the idi building. The following code lines were added in a class called WorldPanel.java to load and show the map (some changes were also needed in the constructor of KSFrame.java):

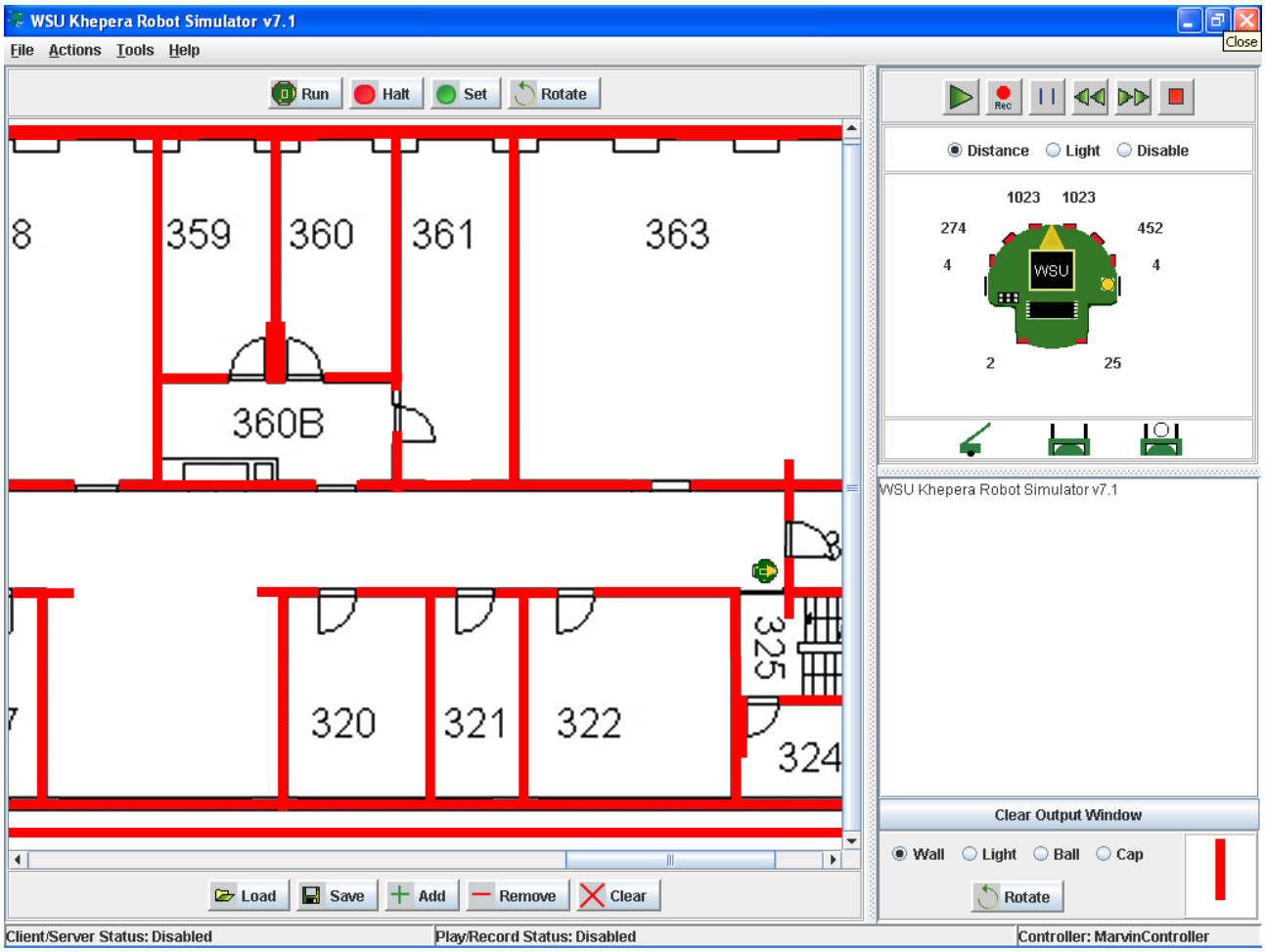

Fig. 5.2.1.1: Screenshot from the modified WSU Khepera Robot Simulator.

(In initialize():)

```
it3 = Toolkit.getDefaultToolkit().getImage("images/it3.gif");
```

(In createWorldImage():)

```
tempg2.drawImage(it3, 0,0,null);
```

The graphics for the robot has also been modified a bit, to more clearly show the direction of the robot. Some changes were also made to the code in order to automatically load the Marvin Robot

Controller:

(In KSFrame.java constructor:)

```
if (rcd.startController("MarvinController")) {
     StatusBar.setRightStatus("Controller: " + "MarvinController");
     running = true;
     server.startTransmission("MarvinController", serverTXRate);
     if (writer.isReady()) {
      writer.startRecording();
     }
}
drawManager.startSimulator();
```

The final change that was needed to the code, was to give the robot controller a reference to KSFrame in order to be able to print messages to the text output window in the simulator's GUI.

5.2.2 MarvinController

MarvinController is an extension of the RobotController class in the WSU Khepera Robot Simulator. The following instructions for implementing controllers are copied from the manual of the simulator [Perretta 2003]:

- *The controller file must import the simulator library. This can be done by adding the following import statement: "import edu.wsu.KepheraSimulator.RobotController;"*
- *The controller name (class name and file name) can be anything.*
- *The controller class must extend the RobotController class.*
- *The controller class must define and implement the methods:*
    - *doWork()*
    - *close()*
- *The constructor can contain anything. This is also a good place to call "setWaitTime()" which sets the time to wait before the next call to doWork().*
- *Never call the wait() method to insert your own delays. Instead use the RobotController's sleep() method if a delay is needed.*
- *The JavaDoc documentation for the RobotController class will provide all the information needed to check the robots sensors and to control the robot.*

To control the movement of the robot, the method *setMotorSpeeds(int a, int b)* is used to set the motor speeds of the left and right wheel. To make the robot move forward, the same speed is set to both wheels, for example *setMotorSpeeds(3, 3)*. Stopping the robot is done by setting both motor speeds to 0. By reversing one motor compared to the other, the robot can turn on a spot, for example by calling setMotorSpeeds(3, -3), which will make the robot turn to the right.

Using *getLeftWheelPosition()*, we can know approximately how much the robot has travelled or turned. The distance sensors' readings are returned by the method *getDistanceValue(int sensorNumber)*, where the integer denotes the sensor number. It's important to be aware of noisy sensors, so when reading sensors one should always use a fixed number of readings and calculate the average value.

doWork()

This is the most important method in the controller, and is called repeatedly as long as the controller runs. The controller is implemented in a way that makes it shift between four different states; *moving*, *turning*, *verifying* and *positioning*. The two first are self-explanatory. The *verifying* state occurs when the robot has performed a movement from one location to another and needs to try to verify that it is at the correct location. The *positioning* state occurs after a verification is complete and has the job of positioning the robot as exactly as possible at the location. High-level pseudocode for the doWork()-method is given below:

```
int speed <- the desired robot speed;
if(there is x- or y-distance to go AND robot is not in any state)
      if(y-distance to go < 0 AND robot not forced to move in x direction)
       if(robot direction is not NORTH): turn(NORTH);
       else: go to moving state;
      else if(y-distance to go > 0 AND robot not forced to move in x direction)
       if(robot direction is not SOUTH): turn(SOUTH);
       else: go to moving state;
      else if(x-distance to go < 0 AND robot not forced to move in y direction)
       if(robot direction is not WEST): turn(WEST);
       else: go to moving state;
      else if(x-distance to go > 0 AND robot not forced to move in y direction)
       if(robot direction is not EAST): turn(EAST);
       else: go to moving state;
else if(robot in turning state)
      if(turning is to the right): setMotorSpeeds(speed, -speed);
      else if(turning is to the left): setMotorSpeeds(-speed, speed);
      if(turning is complete)
       setMotorSpeeds(0,0);
       go out of turning state;
else if(robot in moving state)
      setMotorSpeeds(speed,speed);
      if(wall ahead)
       setMotorSpeeds(0,0);
       force movement in another direction;
```

```
      if(no more distance to go in the current direction)
       setMotorSpeeds(0,0);
       go out of moving state;
else if(robot in verifying state)
      turn towards door;
      if(walls and door opening are correct)
       go to positioning state;
      else
       perform small movement to try and find the location;
else if(robot in positioning state)
      if(sensors indicate adjustment needed)
       perform small movement based on sensor readings;
      else
       update location information;
       set command complete;
else if(command is not complete but x- and y-distances to go is 0)
      go to verifying state;
else
      if(more commands in list)
       get next command;
      else
       wait for instructions;
```

This bit of pseudocode may be difficult to understand at first and will be explained further. It is important to remember that the code is being called over and over again in a loop.

When the robot gets a new command, it will update it's values for x-distance to go and y-distance to go, and the robot is originally not in any state. This will trigger the first if-sentence, which will turn the robot in the right direction before making it move forward. While turning, the robot will be in *turning* state, which simply makes the robot continue turning until it has turned enough. It checks this by continually comparing *getLeftWheelPosition()* with the value that it had when it entered the state.

When the robot moves forward, it will count down the x- and y-distance to go variables based on *getLeftWheelPosition()*. When, at some point, the robot has no more distances to go, but the command is not set as «complete», the robot will go to *verifying* state. As mentioned in Chapter 5.1.7 and 5.1.8, each location is described with either *no wall*, *wall* or *door* in each direction NORTH, SOUTH, EAST and WEST. Each distance sensor reads a value between 0 and 1023, where bigger value means closer to object. The robot assumes that there is a wall on the left if the left sensor has a value bigger than 250. The same goes for right sensor. It assumes that there is a wall in front if both front sensors has a bigger value than 250, and a wall at the back if the two rear sensors have a combined value bigger than 600. Doors are treated as «no wall» in the verification state. If the robot cannot verify it's location, it will perform small movements to «search» for the

door. This searching part of the controller hasn't been much developed. If the location was verified correctly, the robot goes to positioning state.

The positioning state uses the door opening to position itself as accurately at the door as possible. By measuring the difference between the values of the corner sensors, the robot can adjust itself to the right or left by giving itself small commands. The combined value of the corner sensors is used to determine if the robot should move a bit forward or backward. Error tolerance is set in the constant integer DIFF_TOLERANCE, and the size of adjustment commands are calculated from the constant double ADJUSTMENT_REFINEMENT.

Path plans are read from the `path` file. By continually reading this file, the robot knows when new commands are issued. It parses the file and stores the path as a series of commands (arrays of integers) in an ArrayList. Commands are then performed until the list is finished, and the robot starts reading the `path` file again.

Other methods

There are a lot of methods in MarvinController that are called from doWork(). For instance, *turn(int direction)* will turn the robot in a specific direction, *getAvgReadings()* will return sensor readings as an average of ten readings, *findWalls()* will check every direction for walls, etc. None of these will be described in detail here, since they are not a vital scientific part of this project. The interested reader can look up the complete code for MarvinController.java in Appendix B.

5.3 Example run of the program

Here follows an example step-by-step run of the program. The steps are described in more detail in the following sub-chapters.

Start-up and initialization

- Main-method in MarvinDialog is executed to start the program.
- Connection to Telebuster is set up with Jasper.
- The Robot Simulator is initialized by launching startrobot.bat from MarvinDialog's constructor.
- When the Robot Simulator is ready, Marvin can take user inputs.

User inputs text with missing person information

- User inputs a sentence (“Jeg lurer på om du vet hvor magne jobber”).
- Input is forwarded to Telebuster, and Telebuster's answer is read from file.
- TQL and Context is parsed.

- LanguageUnderstanding returns *person* as topic and *place* as type.
- Context returns *true* for “missing person information”.
- The topic and type is stored for later use.
- MarvinDialog tells AnswerGenerator to generate answer based on *focus*.
- The generated answer is printed to the output panel.

User provides the required information

- User inputs the missing information (“ja, det er johnsen”).
- Input is forwarded to Telebuster, and Telebuster's answer is read from file.
- Because MarvinDialog is in *missing_information* state, the previous topic and type are used.
- This time, Context returns *true* for “person found”.
- MarvinDialog tells AnswerGenerator to generate person answer based on type and person record.
- AnswerGenerator recognizes the building name and creates a TrondheimMap object to show the location.
- Person record and answered location is stored for later use.
- The generated text is printed to the output panel.

User asks about a place on this floor

- User inputs a new sentence (“jeg skal møte ham her på rom 356”).
- Input is forwarded to Telebuster, and Telebuster's answer is read from file.
- TQL and Context is parsed.
- LanguageUnderstanding returns *roomnumber* as topic and *default* as type.
- *Roomnumber* is defined as specific, so MarvinDialog calls AnswerGenerator to generate specific answer based on topic and type.
- AnswerGenerator calls Tql to get the room number and answers with text and graphics. It also stores the location for later use.

User asks for guiding

- User input a new sentence (“kan du følge meg dit?”).
- After Telebuster communication and parsing, LanguageUnderstanding returns *location* as topic and *accompany* as type.
- *Location* is defined as a location with no reference, so MarvinDialog looks for previous answers, and finds the location '356'. It calls AnswerGenerator to answer this specific location.
- Based on the type *accompany*, AnswerGenerator calls the A_StarPathPlanner to create a new task for the robot. It also returns a text answer.

- When the robot sees a new task in the path file, it immediately starts executing it.

This is just an example of four inputs that shows how the program works. It will react differently to different user input. The steps are now explained in more detail.

5.3.1 Start-up and initialization

The program is started with the batch-file `marvin.bat`. It includes the Jasper package and executes the main method in the MarvinDialog class, which immediately runs it's own constructor. The constructor sets up the input/output window before initializing the connection with Telebuster through Jasper. If the connection was successful, it prints “Prolog connection established”. It then tries to initialize the Robot Simulator by launching the batch-file `startrobot.bat` and waiting for the robot to tell that it is ready. When MarvinDialog gets the confirmation that the robot is ready, it prints “Robot simulator ready!” and waits for user input (see Figure 5.3.1.1).

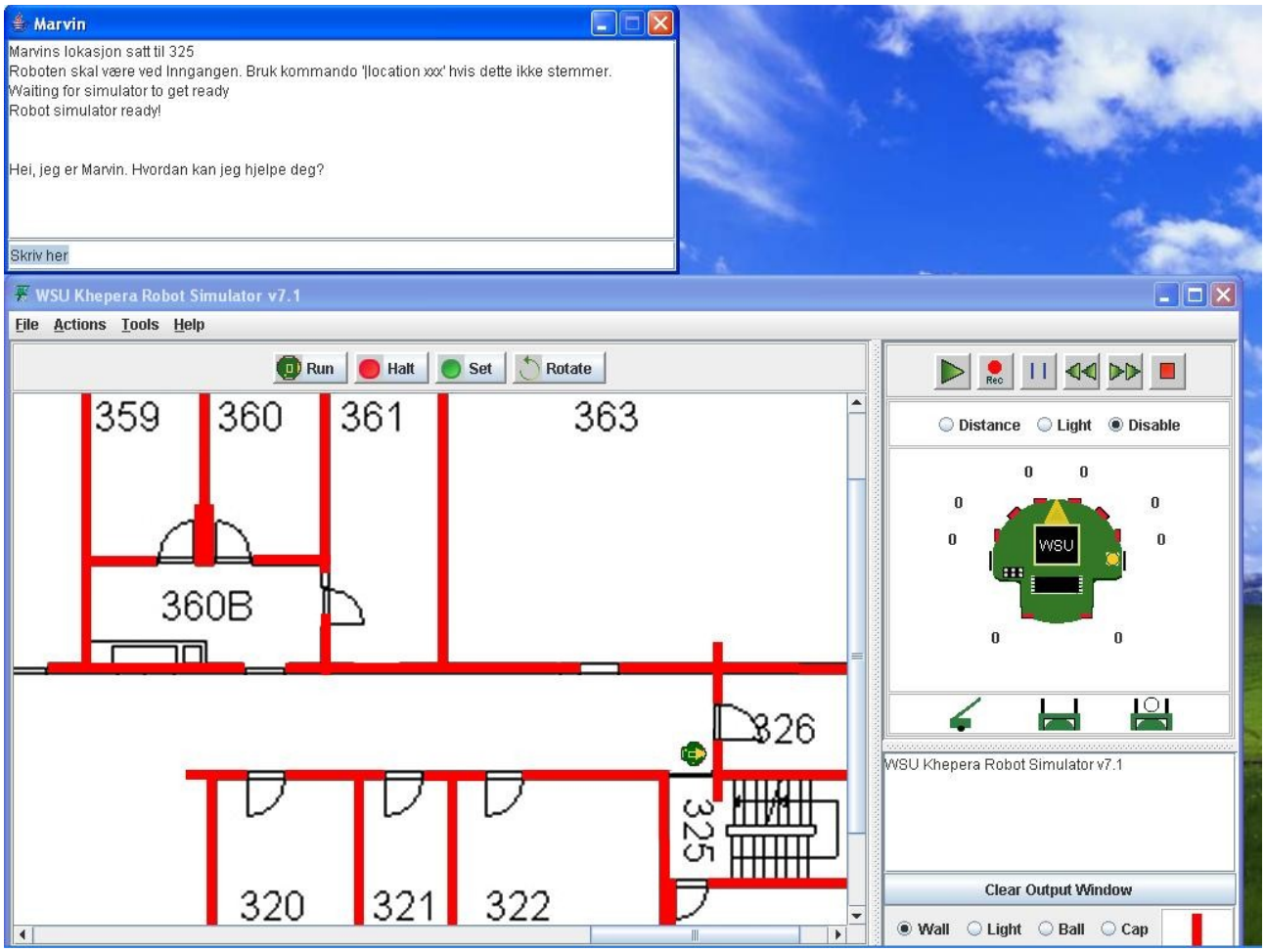

Fig. 5.3.1.1: Marvin is ready to take user input.

5.3.2 User inputs text with missing person information

Now, let's assume that the user inputs the following sentence:

Jeg lurer på om du vet hvor Magne jobber

(*I'm wondering if you know where Magne is working*)

The input is first formatted to a string that Telebuster understands: *"query('nor 123 Jeg lurer på om du vet hvor Magne jobber.')."* This sentence is then written to the question file. Query is performed on Telebuster and the output from Telebuster is read from the answer file. This all happens in the PrologConnection object. The returned answer from Telebuster is shown in Figure 5.3.2.1.

```
*** TQL  ***


new,'I'  isa  self,magne  isa  firstname,tuc  isa  program,free(1)isa
place,wonder/id/whether/'I'/free(2)/free(3),event/real/free(3),knowthing/t
uc/free(1)/free(4),event/free(2)/free(4),work/magne/free(5),srel/in/place/
free(1)/free(5),event/real/free(5)



*** Context 123 ***


focus:  attributes::sn
frame:
    where ?
    when ?
    day ?
    date ?
    bus ?
    attributes
        spacename ?
        givenname magne
        sn ?
        telephonenumber ?
        cn ?
        department ?
        title ?
    itemsfound
        items ?
        itemcount 29
    return ?
    language norsk
    topic tele
referents: [[self,[I]],[firstname,[magne]],[program,[tuc]]]
dialognodes: askfor item(sqt)
```

Fig. 5.3.2.1: Telebuster answer.

After getting the answer from Telebuster, MarvinDialog starts using it's main Dialogue Managing algorithm, as described in Chapter 5.1.1 (see also Figure 4.3.1.1). If first checks to see if Telebuster returned “no answer”, which it didn't. It then calls parsing of the Telebuster answer in the Context and Tql classes. Since the dialogue state is *new_conversation*, the LanguageUnderstanding object is called to extract topic and type from the TQL expression. To explain the tracking of terms, we repeat the TQL expression in a readable format:

new

'I' isa self

magne isa firstname

```
tuc isa program
free(1)isa place
wonder/id/whether/'I'/free(2)/free(3)
event/real/free(3)
knowthing/tuc/free(1)/free(4)
event/free(2)/free(4)
work/magne/free(5)
srel/in/place/free(1)/free(5)
event/real/free(5)
```

The first thing the LanguageUnderstand class does is to look at the first relation element, `wonder/id/whether/'I'/free(2)/free(3)`, which tells that the user is wondering about the term `free(2)`. It then goes through the rest of the relations looking for '`free(2)`' and finds it in `event/free(2)/free(4)`. It then goes through the relations once again, this time looking for `free(4)`, which is found in `knowthing/tuc/free(1)/free(4)`, further on to `free(1)`, which is found to be a place – so type is set to *place*. The term *place* is found to be connected to `free(5)` in `srel/in/place/free(1)/free(5)`, and `free(5)` is connected to `magne` in `work/magne/free(5)`. Finally, `magne` is identified as a `firstname`, and *person* is set at topic.

To sum it all up, LanguageUnderstanding returns that the topic is *person* and the type is *place*, which means that the user wants to know about **the location of some person**. The Dialogue Manager now knows that it has to handle the dialogue as a person-related question.

The next step in the Dialogue Manager is to call Context to see if person information is available, which it isn't. It then calls Context again to see if we are missing some specific part of the person information. Since the firstname slot is filled, but the lastname (sn) slot is not containing anything, and the focus is set to "sn", Context returns *true*. MarvinDialog stores the question type and topic for later use and calls generateFocusAnswer() in the AnswerGenerator object.

AnswerGenerator returns the string *"Vet du etternavnet?"* (*"Do you know the last name?"*) and MarvinDialog prints it in the output window (see Figure 5.3.2.2).

Fig. 5.3.2.2: Marvin answers a user-question.

5.3.3 User provides the required information

Next, we let the user enter the following sentence:

ja, det er johnsen

(yes, it's johnsen)

The first part of the data flow is the same as the examples above, only that MarvinDialog loads the previously stored topic and type instead of new ones. The Context parser also stores person record now, since the answer from Telebuster is containing information about Magne Johnsen. This time, the Context returns *true* when MarvinDialog asks if person information is complete.

The AnswerGenerator will this time be called to answer a person question, and since location information is available in the context record, it will look for the building in the GlosMap class. It finds the building name “Elektro C” and shows it on the map (see Figure 5.3.3.1). The text output string is returned to MarvinDialog. It reads:

Du har spurt om hvor kontoret til Magne Hallstein Johnsen er.

Magne Hallstein Johnsen har kontor på rom 331 i Elektro C.

Elektro C er her på Gløshaugen. På kartet ser du hvor det er.

(You have asked where the office of Magne Hallstein Johnsen is.)

(Magne Hallstein Johnsen has office in room 331 in Elektro C.)

(Elektro C is here on Gløshaugen. On the map you see where it is.)

MarvinDialog will now set the state back to *new_conversation*. Person record and the answered location were stored for possibly later use.

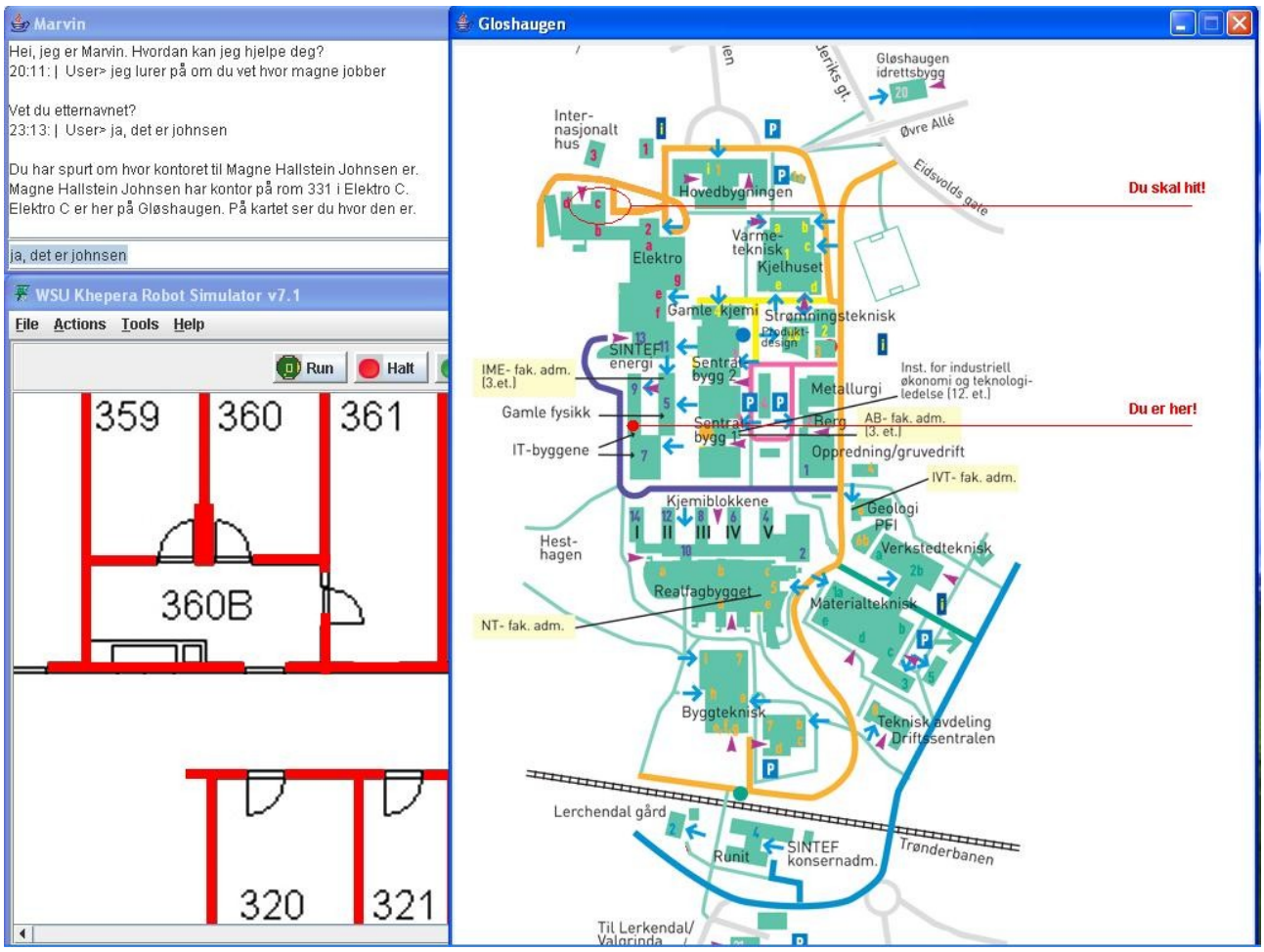

Fig. 5.3.3.1: The program outputs a text answer and a map showing goal location.

5.3.4 User asks about a place on this floor

For the next part of the example, we let the user enter the following sentence:

Jeg skal møte ham her på rom 356

(I'm meeting him here at room 356)

In the previous parts of the example, we have seen how the basic data flow goes so we'll skip that part here. For this sentence, the LanguageUnderstanding class will return *roomnumber* as topic and *default* as type. The reason why the type is *default* and not, for instance, *meeting*, is that not all kinds of TQL relations will change the question type in the system. The program doesn't differ between questions about “meeting”, “visit”, “work here”, etc. All such questions should just yield an answer about the location, and that's exactly what the *default* type does.

Since *roomnumber* is defined as a specific topic, MarvinDialog calls generateSpecificAnswer() in AnswerGenerator, and gives it the topic and type. With the topic *roomnumber*, AnswerGenerator knows that there is a room number in the TQL expression, so it asks Tql to return it. After finding

that the room exists on the 3rd floor of the idi building, it answers with text and graphics (by creating a It3Map object). The result is shown in Figure 5.3.4.1.

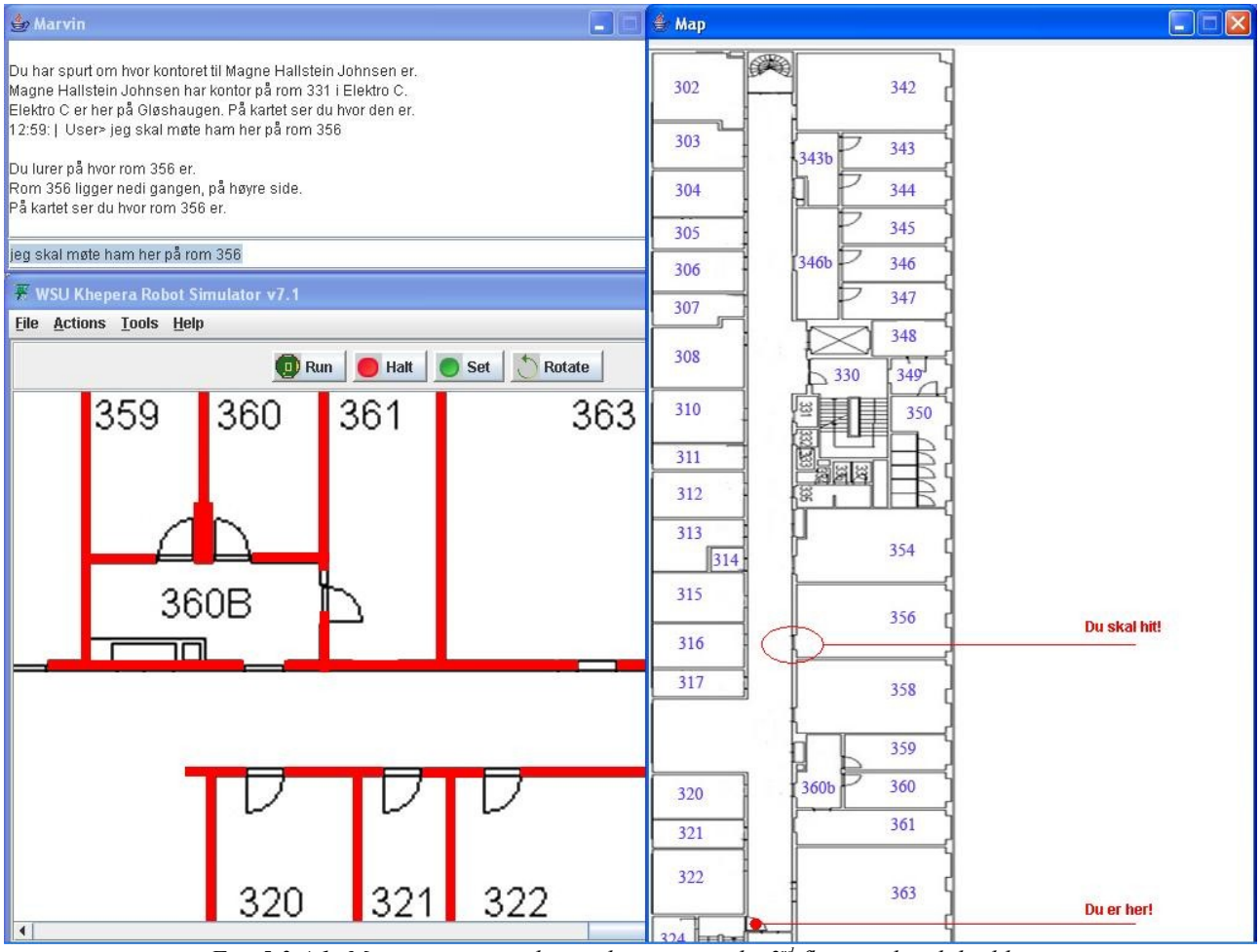

Fig. 5.3.4.1: Marvin answers about a location on the 3rd floor in the idi building.

AnswerGenerator stores the location for later use. Also, MarvinDialog returns to *new_conversation* state.

The example above shows an interesting behaviour. The question is about meeting a person. This would normally yield an answer about the person's office. But, since the meeting with the person is related to a specific room, Marvin answers about the room.

5.3.5 User asks for guiding

The final user input in the example is:

kan du følge meg dit?

(can you lead me there?)

As earlier, the sentence is sent to Telebuster, and answer is parsed by Contex and Tql. This time, LanguageUnderstanding will find that the topic is about a place with no reference (*there*). It therefor sets the topic to *location*, which will be recognized by MarvinDialog as an unspecific location. The question type returned by LanguageUnderstanding is *accompany.*

MarvinDialog will get a positive check from LanguageUnderstanding that the topic is an unspecific location. It will then check for previously stored locations. Remember that during the last answer generation room 356 was stored, so MarvinDialog will send this location to AnswerGenerator and ask for an answer based on this location.

AnswerGenerator will then check the type, which is *accompany*. This question type indicates that the user wants to be guided by the robot to the location. AnswerGenerator therefore calls giveRobotTask("356"), which in turn calls the A_StarPathPlanner for a path plan. It then prints the path plan to the path file and when the robots reads it, it will immediately start executing the plan. The result is shown in Figure 5.3.5.1.

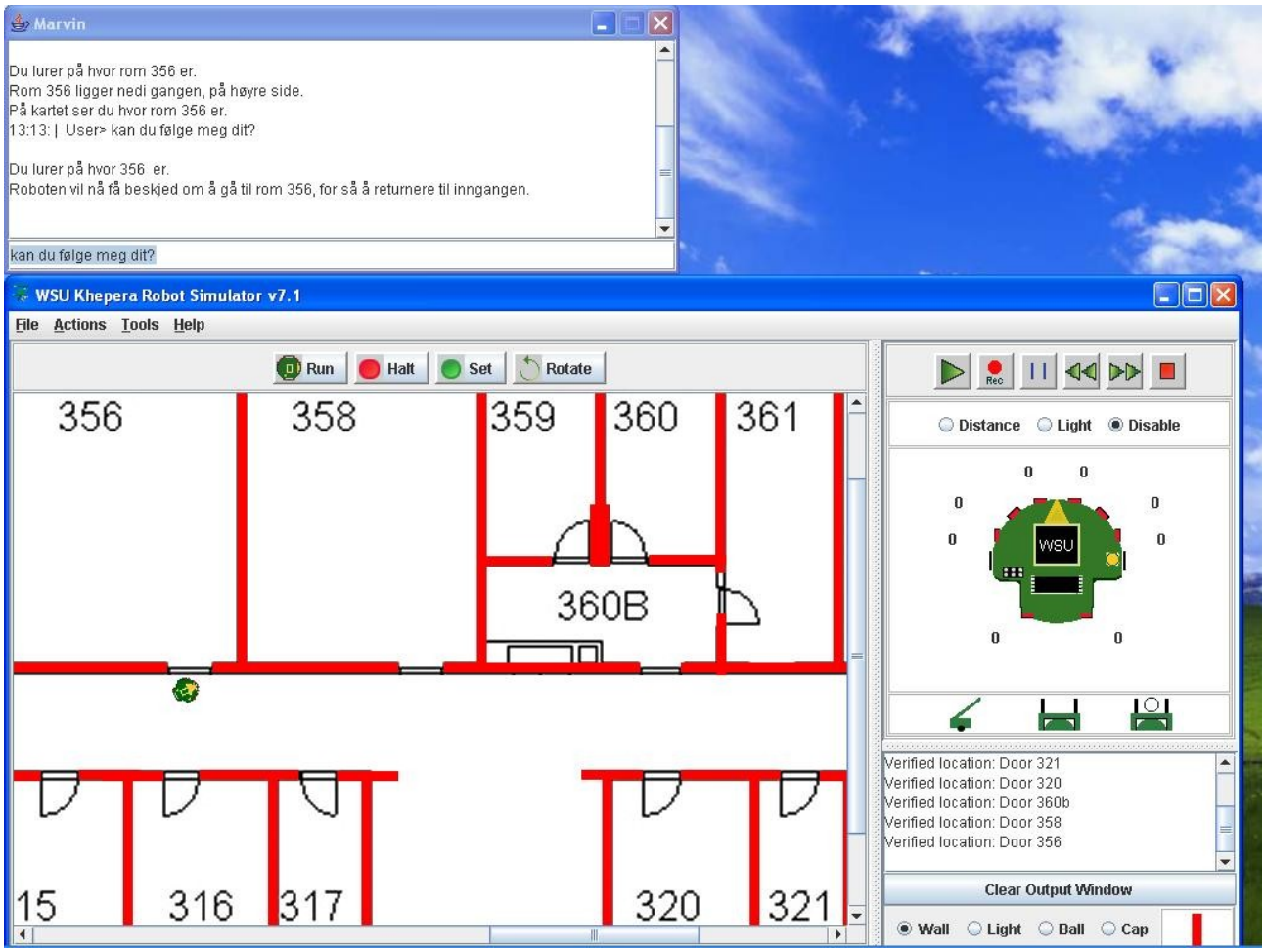

Fig. 5.3.5.1: The simulated robot has reached it's goal at room 356 and heads back to the entrance.

6 Evaluation

This chapter gives a short evaluation of the project.

The project had great freedom of choices from the start. It was difficult to determine which parts of the system to focus on and a lot of time was consumed on research and evaluation of different solutions. The majority of work with the demonstration program didn't start until two vital decisions were made:

- The choice of using a robot simulator instead of a prototype “real” robot.
- The choice of using Telebuster as a basis for the conversational user interface.

After these choices were made, the project became even more encouraging and interesting. The project description involves a very interesting theme regarding intelligent interaction and behaviour in “helping systems”, with multi medial output.

The specific task of the project was to implement a demonstration Intelligent Corridor Guide for the 3rd floor of the idi building at NTNU Gløshaugen. A prototype for such a system has now been made, and the possibilities of extending the program is very good. The natural next step in the process will be to implement speech input and generated voice output. It is also possible to extend the system's knowledge of the world further, with more detailed descriptions of places and means of travel. Integration with future bus systems, working with live-updated bus locations, is one possibility. As for robotics, the project has shown that it is possible to have an intelligent guide system that gives a specialized robot tasks for guiding. Robotics is, however, a major area of study in itself.

References

[Amble 2004] Amble, T. *The Understanding Computer.* Department of Computer and Information Science, Norwegian University of Science and Technology, Norway.

[Fledsberg & Bjerkevoll 1999] Fledsberg, Bjerkevoll. *Buster – Robust dialogue management.* Department of Computer and Information Science, Norwegian University of Science and Technology, Norway.

[McTear 2002] McTear, M.F. *Spoken dialogue technology: enabling the conversational user interface. ACM Computing Surveys, Vol. 34, Issue 1* (Mar 2002). ACM Press, New York, NY, USA.

[Simmons 1997] Simmons, Goodwin, Haigh, Koenig, O'Sullivan. *A Layered Architecture for Office Delivery Robots.* School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

[Dixon & Heinlich 1997] Dixon, Heinlich. *Mobile robot navigation*. Imperial College, London, UK.

[Nourbakhsh 1997] Nourbakhsh, Andre, Tomasi, Genesereth. *Mobile robot obstacle avoidance via depth from focus.* Carnegie Mellon University, University of Berkeley, Stanford University, USA.

[Burgard 1998] Burgard, Cremers, Fox, Hähnel, Lakemeyer, Schulz, Steiner, Thrun. *The Interactive Museum Tour-Guide Robot.* Computer Science Department III, University of Bonn, Germany.

[Thrun 1998] Thrun, Bennewitz, Burgard, Cremers, Dellaert, Fox, Hähnel, Rosenberg, Roy, Schulte, Schulz. *MINERVA: A Second-Generation Museum Tour-Guide Robot.* Computer Science Department III, University of Bonn, Germany. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

[Nolfi & Floreano 2000] Nolfi, Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press 2000.

[Pfeifer & Scheier 1999] Pfeifer, Scheier. *Understanding Intelligence.* MIT Press 1999.

[Perretta & Gallagher 2002] Perretta, Gallagher. *A general purpose Java mobile robot simulator for artificial intelligence research and education.* In Proceedings of the 13th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS-2002), 2002.

[Johnsen 2003] Johnsen, Amble, Harborg. *A Norwegian Spoken Dialogue System for Bus Travel Information. Alternative dialogue structures and evaluation of a system driven version.* Norwegian University of Science and Technology (NTNU), Norway, 2003.

[Johnsen & Kvale 2005] Johnsen, Kvale. *Improving speech centric dialogue systems – The BRAGE project.* Norwegian University of Science and Technology (NTNU), Norway, 2005.

[Perretta 2003] Perretta, S. *WSU Khepera Robot Simulator Manual for Version 7.1.* Wright State University College of Engineering and Computer Science.

[Nilsson 1998] Nilsson, N. *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann Publishers, Inc. San Francisco, CA, USA.

Appendix A – Conversational User Interface source code

A_StarPathPlanner.java

```
import java.util.ArrayList;

public class A_StarPathPlanner{


  /*
  * These are the rooms in the world that the robot knows.
  * Rooms are represented on the form
  * {room_number, x_position, y_position, wall_west, wall_north, wall_east, wall_south}
  * where walls are represented as 0=no wall, 1=wall, 2=door
  */
  private int[][] rooms = {
           {25, 1170, 185,0,0,1,2},
           {22, 1092, 185,0,0,0,2},
           {21, 1052, 185,0,0,0,2},
           {20,  996, 185,0,0,0,2},
           {63, 1130, 160,0,2,0,0},
           {26, 1170, 175,0,0,2,0},
           {24, 1170, 233,1,0,0,2},
           {17,  857, 185,0,0,0,2},
           {16,  801, 185,0,0,0,2},
           {15,  751, 185,0,0,0,2},
           {14,  704, 185,0,0,0,2},
           {13,  660, 185,0,0,0,2},
           {12,  606, 185,0,0,0,2},
           {11,  557, 185,0,0,0,2},
           {10,  509, 185,0,0,0,2},
           {8,  411, 185,0,0,0,2},
           {7,  363, 185,0,0,0,2},
           {6,  314, 185,0,0,0,2},
           {5,  280, 185,0,0,0,2},
           {4,  238, 185,0,0,0,2},
           {3,  167, 185,0,0,0,2},
           {2,  119, 185,0,0,0,2},
           {1,   97, 174,2,0,0,0},
           {42,  119, 160,0,2,0,0},
           {431, 167, 160,0,2,0,0},  //343b
           {43,  157, 119,1,2,0,0},
           {44,  207, 119,0,2,0,0},
           {461, 314, 160,0,2,0,0},  //346b
           {45,  256, 119,1,2,0,0},
           {46,  305, 119,0,2,0,0},
           {47,  353, 119,0,2,0,0},
           {90,  408, 147,0,2,0,0},  //Elevator
           {30,  463, 148,0,2,0,0},
           {49,  460,  75,0,2,0,0},
           {48,  445,  40,2,0,0,0},
           {31,  509, 160,0,2,0,0},
           {32,  541, 160,0,2,0,0},
           {33,  546, 143,0,1,2,1},
           {35,  609, 160,0,2,0,0},
           {34,  607, 129,2,0,0,0},
           {36,  607, 111,2,0,0,0},
           {37,  607,  94,2,1,0,0},
           {54,  700, 160,0,2,0,0},
           {56,  801, 160,0,2,0,0},
           {58,  900, 160,0,2,0,0},
           {601, 996, 160,0,2,0,0},  //360b
           {59,  960, 117,0,2,0,0},
           {60,  983, 117,0,2,0,0},
           {61, 1013, 129,0,0,2,0},
           {63, 1130, 160,0,2,0,0},
           //{91,  924, 185,0,0,0,2},
           {50,  475,  28,0,0,2,0},
           };

  /*
  * Arcs between rooms.
```

```
* Represented as {from_room, to_room, cost}
* Cost is 1 for normal arcs and 99 for doors
*/
private int[][] arcs = {
          { 1,  2, 1},
          { 1, 42, 1},
          { 2,  3, 1},
          { 2, 42, 1},
          { 3,  4, 1},
          { 3,431, 1},
          { 4,  5, 1},
          { 5,  6, 1},
          { 6,  7, 1},
          { 6,461, 1},
          { 7,  8, 1},
          { 8, 90, 1},
          { 8, 10, 1},
          {10, 11, 1},
          {10, 31, 1},
          {11, 32, 1},
          {11, 12, 1},
          {12, 35, 1},
          {12, 13, 1},
          {13, 14, 1},
          {14, 54, 1},
          {14, 15, 1},
          {15, 16, 1},
          {16, 56, 1},
          {16, 17, 1},
          {17, 58, 1},
          //{17, 91, 1},
          {42,431, 1},
          {431,43, 99},
          {43, 44, 1},
          {461,46, 99},
          {46, 45, 1},
          {46, 47, 1},
          {90, 30, 1},
          {30, 49, 99},
          {49, 48, 99},
          {49, 50, 99},
          //{30, 31, 1},
          {31, 32, 1},
          {32, 33, 99},
          {35, 34, 99},
          {34, 36, 1},
          {36, 37, 1},
          {54, 56, 1},
          {56, 58, 1},
          {58, 601, 1},
          {601, 60, 99},
          {601, 61, 99},
          {61, 60, 1},
          {60, 59, 1},
          {25, 22, 1},
          {22, 21, 1},
          {21, 20, 1},
          {25, 26, 1},
          {26, 63, 1},
          {25,  24, 99},
          {22,  63, 1},
          {20,  601, 1},
          //{20,  91, 1},
          {59,  60, 1},
          //{91,  58, 1},
          {17,  58, 1},
          {16,  17, 1},
          {16,  15, 1},
          {16,  56, 1},
          {15,  14, 1},
          {14,  54, 1},
          {13,  14, 1},
          {12,  13, 1},
          {12,  35, 1},
          {12,  11, 1},
          {35,  34, 99},
          {35,  32, 1},
```

```
            {34,  36, 1},
            {36,  37, 1},
            };

  private Node n0;
  private double pixelsToWorldScale = 27.99;       //Set to the wheel movement of Marvin per pixel
on the map

  /**
  * Returns a list of door passages from start location to goal location
  */
  public ArrayList getDoorPassages(ArrayList l, int start, int end){
    int newFromRoom = start;
    int newToRoom;
    ArrayList returnList = new ArrayList();

    for(int i=l.size(); i>0; i--){
      newToRoom = Integer.parseInt(l.get(i-1).toString());

      for(int j=0; j<arcs.length; j++){
        if( (arcs[j][0]==newFromRoom && arcs[j][1]==newToRoom)||(arcs[j][0]==newToRoom &&
arcs[j][1]==newFromRoom) ){
          if(arcs[j][2] == 99) returnList.add(newFromRoom);
        }
      }

      newFromRoom = newToRoom;
    }
    newToRoom = end;

    for(int j=0; j<arcs.length; j++){
      if( (arcs[j][0]==newFromRoom && arcs[j][1]==newToRoom)||(arcs[j][0]==newToRoom &&
arcs[j][1]==newFromRoom) ){
        if(arcs[j][2] == 99) returnList.add(newFromRoom);
      }
    }

    return returnList;
  }

  /**
  * A-star search algorithm that returns a path from start to goal location
  */
  public ArrayList findPath(int start, int goal){

    ArrayList returnList = new ArrayList();

    //A* search algorithm

    //Step 1
    int [] n0Array = findRoom(start);
    n0 = new Node(n0Array[0],n0Array[1],n0Array[2],null,0);
    ArrayList open = new ArrayList();
    open.add(n0);


    //Step 2
    ArrayList closed = new ArrayList();


    //Step 3
    while(!open.isEmpty()){

      //Step 4
      Node n = (Node)open.remove(0);


      //Step 5
      if(n.getNr() == goal){

        while(n.getParent() != null){
          n = n.getParent();
          if(n.getNr() != start) returnList.add(n.getNr());
        }

        return returnList;
```

```
      }
      else{

        //Step 6
        for(int i=0; i<arcs.length; i++){
          int newChildRoomNumber = 0;
          int newChildArcCost = 0;
          if(arcs[i][0] == n.getNr()){
            newChildRoomNumber = arcs[i][1];
            newChildArcCost = arcs[i][2];
          }
          else if(arcs[i][1] == n.getNr()){
            newChildRoomNumber = arcs[i][0];
            newChildArcCost = arcs[i][2];
          }

          if(newChildRoomNumber != 0 && n.notAncestor(newChildRoomNumber)){
            int[] newNodeArray = findRoom(newChildRoomNumber);
            Node nodeToAdd = new Node(newNodeArray[0], newNodeArray[1], newNodeArray[2], n,
newChildArcCost);

            if(!memberOf(nodeToAdd,open) && !memberOf(nodeToAdd,closed)){
             open.add(nodeToAdd);
            }
            else if(memberOf(nodeToAdd,open)){
             Node m = getFromList(newChildRoomNumber,open);
             if(m.calcPathCost(newChildArcCost, n) < m.calcPathCost()){
                    m.setNewParent(n);
                    m.setNewCost(newChildArcCost);
             }
            }
            else if(memberOf(nodeToAdd,closed)){
             Node m = getFromList(newChildRoomNumber,closed);
             if(m.calcPathCost(newChildArcCost, n) < m.calcPathCost()){
                    m.setNewParent(n);
                    m.setNewCost(newChildArcCost);
             }
            }
          }
        }

        //Step 7
        ArrayList orderedOpen = orderOpen(open);
        open = orderedOpen;

        closed.add(n); //Part of step 4
      }
    }
    return null;
  }

  /**
  * Command-generator that creates commands for the robot.
  */
  public ArrayList getCommands(ArrayList l,int start, int end){
    ArrayList returnList = new ArrayList();
    int newToRoom, newFromRoom;
    int
fromX=0,toX=0,fromY=0,toY=0,toRoomWallWest=0,toRoomWallNorth=0,toRoomWallEast=0,toRoomWallSouth=0;
    newFromRoom = start;

    for(int i=l.size(); i>0; i--){
      newToRoom = Integer.parseInt(l.get(i-1).toString());
      for(int j=0; j<rooms.length; j++){
        if(rooms[j][0] == newToRoom){
          toX = rooms[j][1];
          toY = rooms[j][2];
          toRoomWallWest=rooms[j][3];
          toRoomWallNorth=rooms[j][4];
          toRoomWallEast=rooms[j][5];
          toRoomWallSouth=rooms[j][6];
        }
        else if(rooms[j][0] == newFromRoom){
          fromX = rooms[j][1];
          fromY = rooms[j][2];
        }
      }
```

```
      int[] newCommandArray = new int[8];
      newCommandArray[0] = (int) ((toX-fromX)*pixelsToWorldScale);
      newCommandArray[1] = (int) ((toY-fromY)*pixelsToWorldScale);
      newCommandArray[2] = newFromRoom;
      newCommandArray[3] = newToRoom;
      newCommandArray[4] = toRoomWallWest;
      newCommandArray[5] = toRoomWallNorth;
      newCommandArray[6] = toRoomWallEast;
      newCommandArray[7] = toRoomWallSouth;

      returnList.add(newCommandArray);

      newFromRoom = newToRoom;
    }

    newToRoom = end;

    for(int j=0; j<rooms.length; j++){
      if(rooms[j][0] == newToRoom){
        toX = rooms[j][1];
        toY = rooms[j][2];
        toRoomWallWest=rooms[j][3];
        toRoomWallNorth=rooms[j][4];
        toRoomWallEast=rooms[j][5];
        toRoomWallSouth=rooms[j][6];
      }
      else if(rooms[j][0] == newFromRoom){
        fromX = rooms[j][1];
        fromY = rooms[j][2];
      }
    }
    int[] newCommandArray = new int[8];
    newCommandArray[0] = (int) ((toX-fromX)*pixelsToWorldScale);
    newCommandArray[1] = (int) ((toY-fromY)*pixelsToWorldScale);
    newCommandArray[2] = newFromRoom;
    newCommandArray[3] = newToRoom;
    newCommandArray[4] = toRoomWallWest;
    newCommandArray[5] = toRoomWallNorth;
    newCommandArray[6] = toRoomWallEast;
    newCommandArray[7] = toRoomWallSouth;

    returnList.add(newCommandArray);

    return returnList;
  }

  /**
  * Helping method for the A-star searcher
  */
  private Node getFromList(int nr, ArrayList l){
    for(int i=0; i<l.size(); i++){
      if( ((Node)l.get(i)).getNr() == nr ) return ((Node)l.get(i));
    }
    return null;
  }

  /**
  * Helping method for the A-star searcher
  */
  private boolean memberOf(Node n, ArrayList l){
    int nr = n.getNr();
    for(int i=0; i<l.size(); i++){
      if( ((Node)l.get(i)).getNr() == nr ) return true;
    }
    return false;
  }

  /**
  * Helping method for the A-star searcher
  */
  private ArrayList orderOpen(ArrayList old){

    ArrayList ordered = new ArrayList();

    while(!old.isEmpty()){
      Node n = (Node)old.remove(0);
      int insertAt = ordered.size();
```

```
      if(ordered.isEmpty()){
        ordered.add(n);
      }
      else{
        for(int i=0; i<ordered.size(); i++){
          if( ((Node)ordered.get(i)).getFHat() > n.getFHat() ){
            insertAt = i;
            i = ordered.size();
          }
        }
        ordered.add(insertAt, n);
      }
    }

    return ordered;
  }

  /**
  * Returns room information for a specific room
  */
  private int[] findRoom(int nr){
    for(int i=0; i<rooms.length; i++){
      if(rooms[i][0] == nr){
        return rooms[i];
      }
    }
    return null;
  }

  /**
  * Returns only the coordinates of a specific room
  */
  public int[] getRoomCoords(String r){
    int nr = Integer.parseInt(r) - 300;
    int[] returnInt = new int[2];
    for(int i=0; i<rooms.length; i++){
      if(rooms[i][0] == nr){
        returnInt[0] = rooms[i][1];
        returnInt[1] = rooms[i][2];
        return returnInt;
      }
    }
    returnInt[0] = 0;
    returnInt[1] = 0;
    return returnInt;
  }
}
```

AnswerGenerator.java

```
import java.util.ArrayList;
import java.io.*;
import java.util.Calendar;
import java.awt.*;
import javax.swing.*;
import java.util.Map;
import java.util.HashMap;

/**
 * This class generates answers for Marvin to the user. Methods are usually called from MarvinDialog.
All answers
 * can be generated in either Norwegian or English, given by a boolean param.
 *
 * @author   Ole Hartvigsen
 */
public class AnswerGenerator{

   private MarvinDialog md;                  //Marvin GUI module
   private Context c;
   private Tql tql;
   private A_StarPathPlanner searcher;              //Path Planner

   private JFrame mapFrame;
   private Map locationDescriptions;

   /**
    * Simple constructor that sets a referance to a MarvinDialog object and creates a
A_StarPathPlanner object
    * which is stored in 'searcher'.
    *
    * @param md         The MarvinDialog object that has created this object.
    * @param c  Reference to the context parser
    * @param tql        Reference to the TQL parser
    */
   public AnswerGenerator(MarvinDialog md, Context c, Tql tql){
      this.md = md;
      this.c = c;
      this.tql = tql;
      searcher = new A_StarPathPlanner();
      initMapDescriptions();
   }

   /**
    * This method generates answers to person-related questions.
    *
    * @param topicAndType      The question topic and type
    * @param record            Person record
    *
    * @returnAnswer as String
    */
   public String generatePersonAnswer(String[] topicAndType, Record record){
      String returnString = "";

      //Specific person informations
      if(topicAndType[1].equals(MarvinDialog.TELEPHONE)){
         returnString += "Du har spurt om telefonnummeret til "+record.getFullName()+".\n";
         if(record.getPhoneNumber()==0) returnString += "Jeg vet ikke hva telefonnummeret til
"+record.getFullName()+" er.";
         else returnString += "Telefonnummeret til "+record.getFullName()+" er
"+record.getPhoneNumber()+".";
      }
      else if(topicAndType[1].equals(MarvinDialog.ADDRESS)){
         returnString += "Du har spurt om adressen til "+record.getFullName()+".\n";
         if(record.getStreet().equals("")) returnString += "Jeg vet ikke hva adressen til
"+record.getFullName()+" er.";
         else returnString += "Adressen til "+record.getFullName()+" er "+record.getStreet()+".";
      }
      else if(topicAndType[1].equals(MarvinDialog.DEPARTMENT)){
         returnString += "Du har spurt om avdelingen til "+record.getFullName()+".\n";
         if(record.getDepartment().equals("")) returnString += "Jeg vet ikke hva avdelingen til
"+record.getFullName()+" er.";
         else returnString += "Avdelingen til "+record.getFullName()+" er
```

```
"+record.getDepartment()+".";
     }
     else if(topicAndType[1].equals(MarvinDialog.TITLE)){
       returnString += "Du har spurt om tittelen til "+record.getFullName()+".\n";
       if(record.getTitle().equals("")) returnString += "Jeg vet ikke hva tittelen til
"+record.getFullName()+" er.";
       else returnString += "Tittelen til "+record.getFullName()+" er "+record.getTitle()+".";
     }

     //Location
     else if(topicAndType[1].equals(MarvinDialog.ROUTEPLAN) ||
          topicAndType[1].equals(MarvinDialog.SHOW) ||
          topicAndType[1].equals(MarvinDialog.PLACE) ||
          topicAndType[1].equals(MarvinDialog.GO) ||
          topicAndType[1].equals(MarvinDialog.ACCOMPANY) ||
          topicAndType[1].equals(MarvinDialog.DEFAULT)){
       returnString += "Du har spurt om hvor kontoret til "+record.getFullName()+" er.\n";

       //Determine if we should store the location as a 'memory' answer
       if(!(record.getRoomNumber().equals("") && record.getBuilding().equals(""))){
         md.storeLocation(new Location(record.getRoomNumber(), record.getBuilding()));
       }

       //No room number in record
       if(record.getRoomNumber().equals("")){
         //No building found
         if(record.getBuilding().equals("")){
           returnString += "Jeg vet ikke hvor "+record.getFullName()+" er.";
         }
         //Building found
         else{
           returnString += record.getFullName()+" har kontor på "+record.getBuilding()+".";
           GlosMap tempGM = new GlosMap();
           TrondheimMap tempTM = new TrondheimMap();
           if(tempGM.buildingOnGlos(record.getBuilding())){
             returnString += "\n"+record.getBuilding()+" er her på Gløshaugen. På kartet ser du
hvor den er.";
             showGlos(record.getBuilding());
           }
           else if(tempTM.buildingInTrondheim(record.getBuilding())){
             returnString += "\nPå kartet ser du hvor i Trondheim det er.";
             showTrondheim(record.getBuilding());
           }
           else returnString += "Jeg vet ikke hvor denne bygningen er.";
         }
       }
       //Room number found
       else{
         //IT-bygget
         if(record.getBuilding().equals("IT-bygget")||record.getBuilding().equals("IT-bygg")){
           if(record.getRoomNumber().startsWith("3")){

             //This floor
             if(roomOnThisFloor(record.getRoomNumber())){
              returnString += record.getFullName()+" har kontor på rom "+record.getRoomNumber()+" i
denne etasjen.\n";
              returnString += giveDirections(topicAndType[1], record.getRoomNumber());
             }
             else{
              returnString += record.getFullName()+" har kontor på rom "+record.getRoomNumber()+" i
denne bygningen.\n";
              returnString += "Jeg vet ikke hvor dette rommet er, men det skal være i denne
etasjen.";
             }
           }
           else{
             String floor;
             if(record.getRoomNumber().startsWith("0")) floor = "kjelleren";
             else floor = record.getRoomNumber().charAt(0)+". etasje";
             returnString += record.getFullName()+" har kontor på rom "+record.getRoomNumber()+" i
"+floor+". Du er nå i 3. etasje.";
           }
         }
         //No building found
         else if(record.getBuilding().equals("")){
           returnString += record.getFullName()+" har kontor på rom "+record.getRoomNumber()+", men
jeg vet ikke i hvilken bygning.";
```

```
          }
          //Other building
          else{
            returnString += record.getFullName()+" har kontor på rom "+record.getRoomNumber()+" i
"+record.getBuilding()+".";
            //Building on Gløshaugen
            GlosMap tempGM = new GlosMap();
            TrondheimMap tempTM = new TrondheimMap();
            if(tempGM.buildingOnGlos(record.getBuilding())){
              returnString += "\n"+record.getBuilding()+" er her på Gløshaugen. På kartet ser du
hvor den er.";
              showGlos(record.getBuilding());
            }
            else if(tempTM.buildingInTrondheim(record.getBuilding())){
              returnString += "\nPå kartet ser du hvor i Trondheim det er.";
              showTrondheim(record.getBuilding());
            }
            else returnString += "Jeg vet ikke hvor denne bygningen er.";
          }
        }
      }
      return returnString;
    }

    /**
    * Method for generating bus answers
    */
    public String generateBusAnswer(){
      if(c.getAnswerString().equals("")) return "Telebuster gav ikke noe buss-relatert svar.";
      return c.getAnswerString();
    }

    /**
    * Method for generating answer based on Telebuster focus
    */
    public String generateFocusAnswer(){
      if(c.getFocus().equals("sn")) return "Vet du etternavnet?";
      else if(c.getFocus().equals("givenname")) return "Vet du fornavnet?";
      else return "Vennligst gi meg mer informasjon om personen.";
    }

    /**
    * Method for generating answer based on previously stored location
    */
    public String generateAnswerFromLocation(String[] topicAndType, Location location){
      String returnString = "";

      if(location.getRoomNumber().equals("")){
        returnString += "Du lurer på hvor "+location.getBuilding()+" er.\n";
        GlosMap tempGM = new GlosMap();
        TrondheimMap tempTM = new TrondheimMap();
        if(tempGM.buildingOnGlos(location.getBuilding())){
          returnString += "Denne bygningen er her på Gløshaugen. På kartet ser du hvor den er.";
          showGlos(location.getBuilding());
        }
        else if(tempTM.buildingInTrondheim(location.getBuilding())){
          returnString += "\nPå kartet ser du hvor i Trondheim det er.";
          showTrondheim(location.getBuilding());
        }
        else returnString += "Jeg vet ikke hvor denne bygningen er.";
      }
      else{
        returnString += "Du lurer på hvor "+location.getRoomNumber()+" i "+location.getBuilding()+"
er.\n";
        GlosMap tempGM = new GlosMap();
        TrondheimMap tempTM = new TrondheimMap();
        if(roomOnThisFloor(location.getRoomNumber()) && (location.getBuilding().equals("")||
location.getBuilding().equals("IT-bygget")||location.getBuilding().equals("IT-bygget"))){
          returnString += giveDirections(topicAndType[1], location.getRoomNumber());
        }
        else if(location.getBuilding().equals("IT-bygget")||location.getBuilding().equals("IT-
bygg")){
          returnString += "Dette rommet er i denne bygningen, men ikke i denne etasjen.";
        }
        else if(tempGM.buildingOnGlos(location.getBuilding())){
          returnString += "\nDenne bygningen er her på Gløshaugen. På kartet ser du hvor den er.";
          showGlos(location.getBuilding());
```

```
      }
      else if(tempTM.buildingInTrondheim(location.getBuilding())){
        returnString += "\nPå kartet ser du hvor i Trondheim det er.";
        showTrondheim(location.getBuilding());
      }
      else returnString += "Jeg vet ikke hvor denne bygningen er.";
    }

    return returnString;
  }

  /**
  * Method for generating specific answer
  */
  public String generateSpecificAnswer(String[] topicAndType){
    String returnString = "";
    String answeredLocation = "";

    if(topicAndType[0].equals("marvin") || topicAndType[0].equals("robot")){
      returnString += "Roboten skal være ved "+(String)locationDescriptions.get(new
Integer(md.getMarvinLocation()))+".";
      answeredLocation = ""+(300+md.getMarvinLocation());
    }
    else if(topicAndType[0].equals("tuc")){
      returnString += "Jeg er her ved inngangen til 3. etasje på IT-bygget";
      answeredLocation = "325";
    }
    else if(topicAndType[0].equals("I") || topicAndType[0].equals("self")){
      returnString += "Du står ved inngangen til 3.etasje på IT-bygget";
      answeredLocation = "325";
    }
    else if(topicAndType[0].equals("exit")){
      returnString += "Utgangen på IT-bygget er i 1. etasje.";
    }
    else if(topicAndType[0].equals("stair")){
      returnString += "Trappen er rett her ved siden av deg.";
      answeredLocation = "325";
    }
    else if(topicAndType[0].equals("lift")){
      answeredLocation = "390";
      returnString += "Du lurer på hvor heisen er.\n";
      returnString += giveDirections(topicAndType[1], answeredLocation);
    }
    else if(topicAndType[0].equals("toilet")){
      returnString += "Du lurer på hvor det nærmeste toalettet er.\n";
      answeredLocation = "335";
      returnString += giveDirections(topicAndType[1], answeredLocation);
    }
    else if(topicAndType[0].equals("gentlementoilet")){
      returnString += "Du lurer på hvor herretoalettet er.\n";
      answeredLocation = "335";
      returnString += giveDirections(topicAndType[1], answeredLocation);
    }
    else if(topicAndType[0].equals("ladiestoilet")){
      returnString += "Du lurer på hvor dametoalettet er.\n";
      answeredLocation = "332";
      returnString += giveDirections(topicAndType[1], answeredLocation);
    }
    else if(topicAndType[0].equals("noon")){
      returnString += "Jeg vet ingenting om middag.\n";
    }
    else if(topicAndType[0].equals("entry")){
      returnString += "Du er ved inngangen til 3. etasje i IT-bygget.";
      returnString += showEntry();
      answeredLocation = "325";
    }
    else if(topicAndType[0].equals("date") || topicAndType[0].equals("day")){
      returnString += answerToday();
    }
    else if(topicAndType[0].equals("clock") || topicAndType[0].equals("time")){
      returnString += answerTime();
    }
    else if(topicAndType[0].equals("roomnumber")){
      String roomNumber = tql.getInverseIs("room");
      if(roomOnThisFloor(roomNumber)){
        answeredLocation = roomNumber;
        returnString += "Du lurer på hvor rom "+answeredLocation+" er.\n";
```

```
        returnString += giveDirections(topicAndType[1], answeredLocation);
      }
      else{
        returnString += "Dette rommet finnes ikke i denne etasjen.";
      }
    }
    else if(topicAndType[0].equals("quit")){
      if(mapFrame != null) mapFrame.dispose();
      if(topicAndType[1].equals(MarvinDialog.YOUAREWELCOME)){
        returnString += "Bare hyggelig.";
      }
      else if(topicAndType[1].equals(MarvinDialog.BYE)){
        returnString += "Ha det bra.";
      }
      else{
        returnString += "Ha det fint.";
      }
    }

    //Store location answered
    if(!(answeredLocation.equals(""))){
      md.storeLocation(new Location(answeredLocation, ""));
    }
    return returnString;
  }

  /**
  * Returns a String about the current time
  */
  private String answerTime(){
    String returnString = "";

    returnString += "Klokken er ";
    Calendar time = Calendar.getInstance();
    if(time.get(Calendar.HOUR_OF_DAY) >= 10){
      returnString += time.get(Calendar.HOUR_OF_DAY) + ":";
    }
    else{
      returnString += "0" + time.get(Calendar.HOUR_OF_DAY) + ":";
    }
    if(time.get(Calendar.MINUTE) >= 10){
      returnString += time.get(Calendar.MINUTE) + ".";
    }
    else{
      returnString += "0" + time.get(Calendar.MINUTE) + ".";
    }

    return returnString;
  }

  /**
  * Returns a String about the current date
  */
  private String answerToday(){
    String returnString = "";

    Calendar time = Calendar.getInstance();
    if(time.get(Calendar.DAY_OF_WEEK) == Calendar.MONDAY) returnString += "Det er mandag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.TUESDAY) returnString += "Det er tirsdag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.WEDNESDAY) returnString += "Det er onsdag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.THURSDAY) returnString += "Det er torsdag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.FRIDAY) returnString += "Det er fredag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY) returnString += "Det er lørdag ";
    else if(time.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY) returnString += "Det er søndag ";

    returnString += "den "+time.get(Calendar.DAY_OF_MONTH)+". ";

    if(time.get(Calendar.MONTH) == Calendar.JANUARY) returnString += "januar.";
    else if(time.get(Calendar.MONTH) == Calendar.FEBRUARY) returnString += "februar.";
    else if(time.get(Calendar.MONTH) == Calendar.MARCH) returnString += "mars.";
    else if(time.get(Calendar.MONTH) == Calendar.APRIL) returnString += "april.";
    else if(time.get(Calendar.MONTH) == Calendar.MAY) returnString += "mai.";
    else if(time.get(Calendar.MONTH) == Calendar.JUNE) returnString += "juni.";
    else if(time.get(Calendar.MONTH) == Calendar.JULY) returnString += "juli.";
    else if(time.get(Calendar.MONTH) == Calendar.AUGUST) returnString += "august.";
    else if(time.get(Calendar.MONTH) == Calendar.SEPTEMBER) returnString += "september.";
    else if(time.get(Calendar.MONTH) == Calendar.OCTOBER) returnString += "oktober.";
```

```
    else if(time.get(Calendar.MONTH) == Calendar.NOVEMBER) returnString += "november.";
    else if(time.get(Calendar.MONTH) == Calendar.DECEMBER) returnString += "desember.";

    return returnString;
  }

  /**
  * Returns a String with directions. Also calls robot guiding if requested from type.
  */
  private String giveDirections(String type, String destination){
    String returnString = "";

    if(destination.equals("325")){
      returnString += showEntry();
    }
    else{
      int locationAskedFor = Integer.parseInt(destination) - 300;
      if(type.equals(MarvinDialog.ROUTEPLAN) || type.equals(MarvinDialog.SHOW) ||
type.equals(MarvinDialog.PLACE) || type.equals(MarvinDialog.DEFAULT) ||
tql.questionClass()==MarvinDialog.EXPLAIN){

        int[] tempCoords = searcher.getRoomCoords(destination);
        if(tempCoords[1] > 180){
          returnString += locationDescriptions.get(new Integer(locationAskedFor))+" ligger nedi
gangen, på venstre side";
        }
        else{
          returnString += locationDescriptions.get(new Integer(locationAskedFor))+" ligger nedi
gangen, på høyre side";
        }
        ArrayList doors = searcher.getDoorPassages(searcher.findPath(25, locationAskedFor), 25,
locationAskedFor);
        while(!doors.isEmpty()){
          int tempDoor = ((Integer)doors.remove(0)).intValue();
          String tempStr = ""+locationDescriptions.get(new Integer(tempDoor));
          tempStr = tempStr.toLowerCase();
          returnString += ", gjennom døren merket "+tempStr;
        }
        returnString += ".\n";
        String tempStr = ""+locationDescriptions.get(new Integer(locationAskedFor));
        tempStr = tempStr.toLowerCase();
        returnString += "På kartet ser du hvor "+tempStr+" er.";
        showIt3(destination);
      }
      else if(type.equals(MarvinDialog.GO) || type.equals(MarvinDialog.ACCOMPANY)){
        String tempStr = ""+locationDescriptions.get(new Integer(locationAskedFor));
        tempStr = tempStr.toLowerCase();
        returnString += "Roboten vil nå få beskjed om å gå til "+tempStr+", for så å returnere til
inngangen.\n";
        returnString += giveRobotTask(destination);
      }
    }

    return returnString;
  }

  /**
  * Call It3Map to show entry
  */
  private String showEntry(){
    String returnString = "På kartet ser du hvor du er, ved inngangen til 3. etasje i IT-bygget.";
    showIt3(null);
    return returnString;
  }

  /**
  * Method for creating a task for the robot
  */
  public String giveRobotTask(String s){

    //If the robot is not at the entrance
    if(md.getMarvinLocation() != 25){
      return "Roboten er allerede ute på et oppdrag. Den er for tiden ved
"+locationDescriptions.get(new Integer(md.getMarvinLocation()))+".";
    }
    else{
      int roomAskedFor = Integer.parseInt(s);
```

```
      ArrayList toGoal = searcher.findPath(25, roomAskedFor-300);
      ArrayList fromGoal = searcher.findPath(roomAskedFor-300, 25);
      ArrayList toGoalCommands = searcher.getCommands(toGoal, 25, roomAskedFor-300);
      ArrayList fromGoalCommands = searcher.getCommands(fromGoal, roomAskedFor-300, 25);

      ArrayList c = new ArrayList();
      while(!toGoalCommands.isEmpty()){
        c.add(toGoalCommands.remove(0));
      }
      while(!fromGoalCommands.isEmpty()){
        c.add(fromGoalCommands.remove(0));
      }
      //ArrayList c = searcher.getCommands(toGoal, md.getMarvinLocation(), roomAskedFor-300);

      //Create path-string to write to file, then write it to file named 'path'
      String commandsToWrite ="";
      while(!c.isEmpty()){
        int[] newCom = (int[]) c.remove(0);
        commandsToWrite += "From="+newCom[2]+",To="+newCom[3]+
                           ",{"+newCom[0]+","+newCom[1]+","+newCom[4]+","+
                           newCom[5]+","+newCom[6]+","+newCom[7]+"}\n";
      }
      commandsToWrite += ".";

      try{
        FileWriter fw = new FileWriter(MarvinDialog.pathFile);
        fw.write(commandsToWrite);
        fw.close();
      }
      catch(Exception e){
        System.out.print("Exception trying to close filewriter for path: \n");
        System.out.println(e.toString());
      }

      String outputString;
      outputString = "Ok, roboten vil få beskjed om å gå følgende sti: ";
      String outputString2 = "3" + md.getMarvinLocation() ;
      while(!toGoal.isEmpty()){
        if(Integer.parseInt(toGoal.get(toGoal.size()-1).toString()) < 10){
          outputString2 += " -> 30" + toGoal.remove(toGoal.size()-1).toString();
        }
        else{
          outputString2 += " -> 3" + toGoal.remove(toGoal.size()-1).toString();
        }
      }
      outputString2 += " -> " + roomAskedFor;
      while(!fromGoal.isEmpty()){
        if(Integer.parseInt(fromGoal.get(fromGoal.size()-1).toString()) < 10){
          outputString2 += " -> 30" + fromGoal.remove(fromGoal.size()-1).toString();
        }
        else{
          outputString2 += " -> 3" + fromGoal.remove(fromGoal.size()-1).toString();
        }
      }
      outputString2 += " -> 325";

      //return outputString+outputString2;
      return "";
    }
  }

  /**
  * Method for killing map frames
  */
  public void killMap(){
    if(mapFrame!=null) mapFrame.dispose();
  }

  /**
  * Method for checking if room is on this floor
  */
  private boolean roomOnThisFloor(String s){
    if(s.equals("301") || s.equals("302") || s.equals("303") ||
       s.equals("304") || s.equals("305") || s.equals("306") ||
       s.equals("307") || s.equals("308") ||
       s.equals("310") || s.equals("311") || s.equals("312") ||
```

```
      s.equals("313") || s.equals("314") || s.equals("315") ||
      s.equals("316") || s.equals("317") || s.equals("320") ||
      s.equals("321") || s.equals("322") || s.equals("324") ||
      s.equals("325") || s.equals("326") ||
      s.equals("330") || s.equals("331") || s.equals("332") ||
      s.equals("333") || s.equals("334") || s.equals("335") ||
      s.equals("336") || s.equals("337") || s.equals("342") ||
      s.equals("343") || s.equals("344") || s.equals("345") ||
      s.equals("346") || s.equals("347") || s.equals("348") ||
      s.equals("349") || s.equals("350") ||
      s.equals("354") || s.equals("356") || s.equals("358") ||
      s.equals("359") || s.equals("360") || s.equals("361") ||
      s.equals("363") ) return true;
   else return false;
}

/**
* Shows map of it-building 3rd floor
*/
private void showIt3(String roomNumber){
   JPanel cp = new JPanel();
      if(roomNumber==null) cp.add(new It3Map());
      else cp.add(new It3Map(roomNumber));
      if(mapFrame != null) mapFrame.dispose();
   mapFrame = new JFrame("Map");
   mapFrame.setSize(new Dimension(500,800));
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   mapFrame.setLocation(screenSize.width-500,0);
   mapFrame.setContentPane(cp);
   mapFrame.setFocusableWindowState(false);
   mapFrame.setResizable(false);
   mapFrame.setVisible(true);

   md.setLocation(0,0);
}

/**
* Shows map of the Gløshaugen campus
*/
private void showGlos(String building){
   JPanel cp = new JPanel();
   cp.add(new GlosMap(building));
   if(mapFrame != null) mapFrame.dispose();
   mapFrame = new JFrame("Gloshaugen");
   mapFrame.setSize(new Dimension(660,800));
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   mapFrame.setLocation(screenSize.width-660,0);
   mapFrame.setContentPane(cp);
   mapFrame.setFocusableWindowState(false);
   mapFrame.setResizable(false);
   mapFrame.setVisible(true);

   md.setLocation(0,0);
}

/**
* Shows a map of Trondheim
*/
private void showTrondheim(String location){
   JPanel cp = new JPanel();
      cp.add(new TrondheimMap(location));
      if(mapFrame != null) mapFrame.dispose();
   mapFrame = new JFrame("Trondheim");
   mapFrame.setSize(new Dimension(560,400));
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   mapFrame.setLocation(screenSize.width-560,0);
   mapFrame.setContentPane(cp);
   mapFrame.setFocusableWindowState(false);
   mapFrame.setResizable(false);
   mapFrame.setVisible(true);

   md.setLocation(0,0);
}

/**
* Initializes the hash map of locations
*/
```

```
  private void initMapDescriptions(){
     locationDescriptions = new HashMap();
     locationDescriptions.put(new Integer(25), "Inngangen");
     locationDescriptions.put(new Integer(22), "Rom 322");
     locationDescriptions.put(new Integer(21), "Rom 321");
     locationDescriptions.put(new Integer(20), "Rom 320");
     locationDescriptions.put(new Integer(63), "Rom 363");
     locationDescriptions.put(new Integer(26), "Rom 326");
     locationDescriptions.put(new Integer(24), "Rom 324");
     locationDescriptions.put(new Integer(17), "Rom 317");
     locationDescriptions.put(new Integer(16), "Rom 316");
     locationDescriptions.put(new Integer(15), "Rom 315");
     locationDescriptions.put(new Integer(14), "Rom 314");
     locationDescriptions.put(new Integer(13), "Rom 313");
     locationDescriptions.put(new Integer(12), "Rom 312");
     locationDescriptions.put(new Integer(11), "Rom 311");
     locationDescriptions.put(new Integer(10), "Rom 310");
     locationDescriptions.put(new Integer(8), "Rom 308");
     locationDescriptions.put(new Integer(7), "Rom 307");
     locationDescriptions.put(new Integer(6), "Rom 306");
     locationDescriptions.put(new Integer(5), "Rom 305");
     locationDescriptions.put(new Integer(4), "Rom 304");
     locationDescriptions.put(new Integer(3), "Rom 303");
     locationDescriptions.put(new Integer(2), "Rom 302");
     locationDescriptions.put(new Integer(1), "Rom 301");
     locationDescriptions.put(new Integer(42), "Rom 342");
     locationDescriptions.put(new Integer(431), "Rom 343b");
     locationDescriptions.put(new Integer(43), "Rom 343");
     locationDescriptions.put(new Integer(44), "Rom 344");
     locationDescriptions.put(new Integer(461), "Rom 346b");
     locationDescriptions.put(new Integer(45), "Rom 345");
     locationDescriptions.put(new Integer(46), "Rom 346");
     locationDescriptions.put(new Integer(47), "Rom 347");
     locationDescriptions.put(new Integer(90), "Heisen");
     locationDescriptions.put(new Integer(30), "Rom 330");
     locationDescriptions.put(new Integer(49), "Rom 349");
     locationDescriptions.put(new Integer(48), "Rom 348");
     locationDescriptions.put(new Integer(31), "Rom 331");
     locationDescriptions.put(new Integer(32), "Det lille dametoalettet");
     locationDescriptions.put(new Integer(33), "Rom 333");
     locationDescriptions.put(new Integer(34), "Rom 334");
     locationDescriptions.put(new Integer(35), "Herretoalettet");
     locationDescriptions.put(new Integer(36), "Rom 336");
     locationDescriptions.put(new Integer(37), "Rom 337");
     locationDescriptions.put(new Integer(54), "Rom 354");
     locationDescriptions.put(new Integer(56), "Rom 356");
     locationDescriptions.put(new Integer(58), "Rom 358");
     locationDescriptions.put(new Integer(601), "Rom 360b");
     locationDescriptions.put(new Integer(59), "Rom 359");
     locationDescriptions.put(new Integer(60), "Rom 360");
     locationDescriptions.put(new Integer(61), "Rom 361");
     locationDescriptions.put(new Integer(63), "Rom 363");
     locationDescriptions.put(new Integer(50), "Det store dametoalettet");
  }

  /**
  * Returns a string that says robot commands have been cleared
  */
  public String getClearedCommands(){
     return "Alle kommandoer til roboten slettet.";
  }

  /**
  * Returns a string that says robot location has been changed
  */
  public String getSetMarvinLocation(int room){
     return ("Marvins lokasjon satt til "+ (room+300));
  }

  /**
  * Returns a string saying where the robot is
  */
  public String getLocation(int room){
     return ("Roboten skal være ved "+(String)locationDescriptions.get(new Integer(room))+". Bruk
kommando '|location xxx' hvis dette ikke stemmer.");
  }
}
```

Context.java

```
import java.io.StreamTokenizer;
import java.util.StringTokenizer;
import java.io.StringReader;

public class Context{

   private String firstName;
   private String lastName;
   private String focus;
   private String answerString;
   private Record record;
   private boolean bus;                                          //Determines if topic is bus

   public Context(){
      record = new Record("");
   }

   public void parseNewAnswer(String s){

      resetContext();
      boolean inContextField = false;
      boolean inAttributesField = false;
      boolean inFocusField = false;
      boolean inFrameField = false;

      StreamTokenizer st = new StreamTokenizer(new StringReader(s));

      try{
         while(st.ttype != StreamTokenizer.TT_EOF){
            if(st.sval != null && st.sval.equals("Context")){
               inContextField = true;
            }

            if(inContextField && st.sval != null && st.sval.equals("focus")){
               inFocusField = true;
            }
            else if(inContextField && st.sval != null && st.sval.equals("frame")){
               inFrameField = true;
            }
            else if(inContextField && inFrameField && st.sval != null &&
st.sval.equals("attributes")){
               inAttributesField = true;
            }

            if(st.sval != null && st.sval.equals("topic")){
                //Check for topic bus
               st.nextToken();
               if(st.sval != null && st.sval.equals("bus")){
                  bus = true;
                  extractAnswerString(s);
               }
            }

            if(inContextField && inFocusField){
               boolean goOn = true;
               boolean getFocus = false;
               while(goOn){
                  if(st.sval != null && st.sval.equals("frame")){
                   inFocusField = false;
                   inFrameField = true;
                   goOn = false;
                  }
                  else if(st.sval != null && st.sval.equals("attributes")){
                   goOn = false;
                   getFocus = true;
                  }
                  st.nextToken();
               }
               if(getFocus){
                  st.nextToken();
                  st.nextToken();
                  focus = st.sval;
```

```
            inFocusField = false;
          }
        }
        else if(inContextField && inFrameField && inAttributesField){
          if(st.sval != null && st.sval.equals("givenname")){
            st.nextToken();
            firstName = st.sval;
          }
          if(st.sval != null && st.sval.equals("sn")){
            st.nextToken();
            lastName = st.sval;
          }
          if(st.sval != null && st.sval.equals("itemsfound")){
            inAttributesField = false;
          }
        }
        st.nextToken();
      }
    }
    catch(Exception e){
      System.out.println("Exception while performing nextToken() in Context.parseNewAnswer():");
      System.out.println(e.toString());
    }
    record = new Record(s);
  }

  private void extractAnswerString(String s){
    StringTokenizer st = new StringTokenizer(s);
    boolean inAnswerField = false;
    answerString = "";
    while(st.hasMoreTokens()){
      if(st.nextToken().equals("Answer")){
        boolean done = false;
        while(!done){
          String tempo = st.nextToken();
          if(tempo.equals("Context")) done=true;
          else if(tempo.equals("***")){}
          else answerString += tempo+" ";
        }
      }
    }
  }

  public boolean bus(){
    if(!answerString.equals("")) return bus;
    return false;
  }

  public boolean personFound(){
    if(record.personFound()) return true;
    else return false;
  }

  public boolean missingInformation(){
    if(( (firstName!=null && lastName==null)||(firstName==null && lastName!=null) )&& focus!=null)
return true;
    else return false;
  }

  public String getAnswerString(){
    return answerString;
  }

  public String getFirstName(){
    return firstName;
  }

  public String getLastName(){
    return lastName;
  }

  public String getName(){
    return firstName+" "+lastName;
  }

  public String getDepartment(){
    return record.getDepartment();
```

```
  }

  public String getFullName(){
    return record.getFullName();
  }

  public int getPhoneNumber(){
    return record.getPhoneNumber();
  }

  public String getBuilding(){
    return record.getBuilding();
  }

  public String getRoomNumber(){
    return record.getRoomNumber();
  }

  public String getTitle(){
    return record.getTitle();
  }

  public String getStreet(){
    return record.getStreet();
  }

  public String getRecordString(){
    return record.recordString();
  }

  public String getFocus(){
    return focus;
  }

  public Record getRecord(){
    return (Record)record.clone();
  }

  private void resetContext(){
    firstName = null;
    lastName = null;
    focus = null;
    bus = false;
    answerString = "";
  }
}
```

GlosMap.java

```
import javax.swing.*;
import java.awt.geom.*;
import java.util.ArrayList;
import java.lang.Thread;


public class GlosMap extends JLabel{

   private String[][] mapCoords = {
                        //Hovedbygningen
                        {"Hvedbygningen", "219", "116"},
                        {"Hovedgygget", "219", "116"},
                        {"Hovedbygn.øst", "219", "116"},
                        {"Hovedbygningen", "219", "116"},
                        {"Hovedbygningen306B", "219", "116"},
                        {"Hovedbygn.", "219", "116"},
                        {"Hovedbygn", "219", "116"},
                        {"Hovedbyggningen", "219", "116"},
                        {"Hovedbygget", "219", "116"},
                        {"Hovedbygget 2 etg.", "219", "116"},
                        {"Hovedbygg", "219", "116"},
                        {"Hovedb.4etg.", "219", "116"},

                        //Sentralbygg
                        {"Stripa", "216", "313"},
                        {"Sentralbygg 1", "216", "313"},
                        {"Sentralbygg I", "216", "313"},
                        {"Sentrbygg I", "216", "313"},
                        {"SEntralbygg", "216", "313"},
                        {"Sentralbygg", "216", "313"},
                        {"Sentralbygg ", "216", "313"},
                        {"Sentralbygg 2", "217", "266"},
                        {"Sentralbygg II", "217", "266"},

                        //Varmeteknisk
                        {"Varmetek Lab", "264", "159"},
                        {"Varmet.lab" , "245", "149"},
                        {"Varmetekn.lab" , "245", "149"},
                        {"Varmetekn Lab" , "245", "149"},
                        {"Varmeteknisk lab" , "245", "149"},
                        {"Varmeteknisk Lab 4.etg" , "245", "149"},
                        {"Varmeteknikk" , "245", "149"},
                        {"Varmetekn." , "245", "149"},
                        {"Varmetek Lab" , "245", "149"},

                        //IT syd
                        {"IT-bygg syd", "155", "334"},

                        //Kjel
                        {"VTL, Kjelhuset", "273","181"},
                        {"Kjelhuset", "273","181"},
                        {"Kjelhus", "273","181"},

                        //Gamle kjemi
                        {"Gamle Kjemi", "214", "215"},
                        {"Gamle kjemi", "214", "215"},
                        {"Gml.kjemi", "214", "215"},

                        //Gamle fysikk
                        {"Gml.Fysikk", "172", "291"},
                        {"Gml.fysikk", "172", "291"},
                        {"Gml fysikk", "172", "291"},
                        {"Gl. Fysikk", "172", "291"},
                        {"Gl.fysikk", "172", "291"},
                        {"G-fysikk", "172", "291"},
                        {"G-Fysikk", "172", "291"},
                        {"Gamle fysikkt", "172", "291"},
                        {"Gamle fysikk 3. etg.", "172", "291"},
                        {"Gamle Fysikk", "172", "291"},
                        {"Gamle fysikk", "172", "291"},
                        {"Gamle fysik", "172", "291"},
```

```
//Strømningsteknikk
{"Strømninksteknikk", "282", "217"},
{"Strømningstekn.lab", "282", "217"},
{"Strømningsteknisk lab", "282", "217"},
{"Strømningsteknikk", "282", "217"},
{"Strømningstekn", "282", "217"},
{"Strømningslab.", "282", "217"},
{"Strøminsteknisk", "282", "217"},

//Elektro
{"Elektro A", "160", "166"},
{"Elektroblokk A", "160", "166"},
{"ElektroblokkA", "160", "166"},
{"Elektro B", "117", "152"},
{"Elektro B II", "117", "152"},
{"Elektro b", "117", "152"},
{"Elekstro B", "117", "152"},
{"Elektroblokk B", "117", "152"},
{"Elektro C", "118", "130"},
{"Elektro D", "93", "129"},
{"Elektroblokk D", "93", "129"},
{"Elektro E", "168", "203"},
{"Elektro E-blokk", "168", "203"},
{"Elektro F", "167", "219"},
{"Elktro F", "167", "219"},
{"Elektro G", "183", "192"},

//Sintef Energi
{"SINTEF Energiforskning", "147", "250"},
{"SINTEF Energiforsk", "147", "250"},
{"Sintef Energiforsk", "147", "250"},

//Realfagbygget
{"Realfgbygget", "208", "443"},
{"Realfagsbygget", "208", "443"},
{"Realfagsbygg", "208", "443"},
{"Realfag", "208", "443"},
{"Realfagbygg", "208", "443"},
{"Realfagbyggg", "208", "443"},
{"Realfagbygget", "208", "443"},
{"REALFAGBYGG", "208", "443"},
{"Real", "208", "443"},
{"Ralfagsbygget", "208", "443"},

{"Sætra", "115", "88"},

//Kjemi 1
{"Kjemi", "148", "389"},
{"Kjemihall, R25", "148", "389"},
{"Kjemihall", "148", "389"},
{"Kjemihallen", "148", "389"},
{"Kjemiblokk I", "148", "389"},

//Kjemi 2
{"Kjemi II Vrimlehall", "172", "389"},
{"Kjemi II-IIi", "172", "389"},
{"Kjemi II", "172", "389"},
{"Kjemiblokk II", "172", "389"},

//Kjemi 3
{"Kjemi III/IV", "201", "389"},
{"Kjemi III", "201", "389"},
{"Kjemiblokk III", "201", "389"},
{"Kjemiblokk III-IV", "201", "389"},
{"Kjemiblokk 3", "201", "389"},
{"Kjemibl.iii", "201", "389"},

//Kjemi 4
{"Kjemi mellombygg 4/5", "226", "389"},
{"Kjemi IV", "226", "389"},
{"Kjemiblokk IV", "226", "389"},
{"Kjemiblokk 4", "226", "389"},
{"Kjemi 4", "226", "389"},

//Kjemi 5
{"Kjemi V", "253", "389"},
{"Kjemihall V", "253", "389"},
```

```
                    {"Kjemiblokk V", "253", "389"},
                    {"Kjemiblokk 5", "253", "389"},
                    {"Kjemiblokk 5-407", "253", "389"},
                    {"Kjemi 5", "253", "389"},

                                        };

  ImageIcon it3;
  private Ellipse2D.Double circle;
  private Ellipse2D.Double startDot;
  private Line2D.Double youreHereLine;
  private Line2D.Double goalLine;

  private ArrayList pathList;
  private int goalText_y;
  private boolean buildingFound = false;

  public GlosMap(String building){
    super(new ImageIcon("gloshaugen.gif"));
    int[] locationCoords = new int[2];
    for(int i=0; i<mapCoords.length; i++){
      if( ((String)mapCoords[i][0]).equals(building)){
        locationCoords[0] = Integer.parseInt(mapCoords[i][1]);
        locationCoords[1] = Integer.parseInt(mapCoords[i][2]);
        buildingFound = true;
      }
    }
    if(buildingFound) setLocationCircle(locationCoords[0]-25,locationCoords[1]-15);
    startDot = new Ellipse2D.Double(140,304,10,10);
    youreHereLine = new Line2D.Double(145,309,600,309);
  }

  //Dummy constructor
  public GlosMap(){

  }

  public boolean buildingOnGlos(String s){
    for(int i=0; i<mapCoords.length; i++){
      if(s.equals(mapCoords[i][0])) return true;
    }
    return false;
  }

  public void paintComponent(Graphics g){
    clear(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setColor(Color.red);
    if(buildingFound){
      g2d.draw(circle);
      g2d.draw(goalLine);
      g2d.drawString("Du skal hit!", 550, goalText_y);
    }
    g2d.fill(startDot);
    g2d.draw(youreHereLine);
    g2d.drawString("Du er her!", 550, 300);
  }

  public void setLocationCircle(int x, int y){
    circle = new Ellipse2D.Double(x,y,50,30);
    goalLine = new Line2D.Double(x+50, y+15, 600, y+15);
    goalText_y = y+5;
  }

  protected void clear(Graphics g){
    super.paintComponent(g);
  }

  protected Ellipse2D.Double getCircle(){
    return (circle);
  }
}
```

It3Map.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.ArrayList;
import java.lang.Thread;


public class It3Map extends JLabel{

   private int[][] mapCoords = {
                       { 1, 103, 41},
                       { 2, 84, 56},
                       { 3, 84,87},
                       { 4, 84,132},
                       { 5, 84,158},
                       { 6, 84,179},
                       { 7, 84,214},
                       { 8, 84,242},
                       {10, 84,303},
                       {11, 84,336},
                       {12, 84,365},
                       {13, 84,399},
                       {14, 84,424},
                       {15, 84,454},
                       {16, 84,486},
                       {17, 84,520},
                       {20, 84,609},
                       {21, 84,644},
                       {22, 84,671},
                       {25, 90,717},
                       {24, 48,719},
                       {26, 102,720},
                       {63, 120,689},
                       {61, 145,624},
                       {60, 163,603},
                       {59, 162,586},
                       {58, 120,546},
                       {56, 120,489},
                       {54, 120,424},
                       {35, 120,369},
                       {32, 120,322},
                       {31, 120,304},
                       {33,130,330},
                       {34,145,354},
                       {36,160,354},
                       {37,175,354},
                       {30,135,269},
                       {42, 121, 54},
                       {43,159,80},
                       {44,159,111},
                       {45,159,142},
                       {46,159,173},
                       {47,159,203},
                       {48,219,237},
                       {49,224,269},
                       {50,231,303},
                       {90,134,241},
                       {99,63,725}
                                              };

   ImageIcon it3;
   private Ellipse2D.Double circle;
   private Ellipse2D.Double startDot;
   private Line2D.Double youreHereLine;
   private Line2D.Double goalLine;

   private ArrayList pathList;
   private int goalText_y;
   private int tempX;
   private int tempY;

   private boolean showGoal;
```

```
  public It3Map(String roomNumber){
    super(new ImageIcon("it3.jpg"));
    int[] locationCoords = new int[2];
    int nr = Integer.parseInt(roomNumber)-300;
    for(int i=0; i<mapCoords.length; i++){
      if(mapCoords[i][0] == nr){
        locationCoords[0] = mapCoords[i][1];
        locationCoords[1] = mapCoords[i][2];
      }
    }
    setLocationCircle(locationCoords[0]-25,locationCoords[1]-15);
    startDot = new Ellipse2D.Double(85,712,10,10);
    youreHereLine = new Line2D.Double(90,717,400,717);
    tempX = 75;
    tempY = 815;
    showGoal = true;
  }

  public It3Map(){
    super(new ImageIcon("it3.jpg"));
    int[] locationCoords = new int[2];
    startDot = new Ellipse2D.Double(85,712,10,10);
    youreHereLine = new Line2D.Double(90,717,400,717);
    tempX = 75;
    tempY = 815;
    showGoal = false;
  }

  public void paintComponent(Graphics g){
    clear(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setColor(Color.red);
    if(showGoal){
      g2d.draw(circle);
      g2d.draw(goalLine);
      g2d.drawString("Du skal hit!", 360, goalText_y);
    }
    g2d.fill(startDot);
    g2d.draw(youreHereLine);
    g2d.drawString("Du er her!", 360, 710);

    g2d.setColor(Color.green);
  }

  public void setLocationCircle(int x, int y){
    circle = new Ellipse2D.Double(x,y,50,30);
    goalLine = new Line2D.Double(x+50, y+15, 400, y+15);
    goalText_y = y+5;
  }

  protected void clear(Graphics g){
    super.paintComponent(g);
  }

  protected Ellipse2D.Double getCircle(){
    return (circle);
  }
}
```

LanguageUnderstanding.java

```
import java.io.StreamTokenizer;
import java.io.StringReader;

public class LanguageUnderstanding{

   private Tql tql;
   private Context c;
   private String[] topicAndType;

   private String[] personRelatedTopics =
   {
      "person", "roomfree", "firstname", "lastname", "agent", "man", "woman", "telephone", "address",
"title", "department"
   };

   private String[] specificTopics =
   {
      "roomnumber", "marvin", "robot", "tuc", "I", "self", "exit", "entry", "stair", "lift",
"toilet", "clock", "day", "date", "time", "quit", "gentlementoilet", "ladiestoilet", "noon"
   };

   private String[] unspecificLocation =
   {
      "location"
   };

   public LanguageUnderstanding(Tql tql, Context c){
      this.tql = tql;
      this.c = c;
   }

   public boolean isPersonRelated(String s){
      for(int i=0; i<personRelatedTopics.length; i++){
         if(s.equals(personRelatedTopics[i])) return true;
      }
      return false;
   }

   public boolean isSpecific(String s){
      for(int i=0; i<specificTopics.length; i++){
         if(s.equals(specificTopics[i])) return true;
      }
      return false;
   }

   public boolean isUnspecificLocation(String s){
      for(int i=0; i<unspecificLocation.length; i++){
         if(s.equals(unspecificLocation[i])) return true;
      }
      return false;
   }

   public String[] getTopicAndType(){
      topicAndType = new String[3];
      topicAndType[1] = MarvinDialog.DEFAULT;
      topicAndType[2] = "";

      //Topic bus
      if(c.bus()){
         topicAndType[0] = "bus";
      }
      //A 'which'-question has been asked.
      else if(tql.questionClass() == MarvinDialog.WHICH){
         classWhich();
      }
      //A 'test'-question has been asked.
      else if(tql.questionClass() == MarvinDialog.TEST){
         classTest();
      }
      //A 'new'-phrase has been uttered
      else if(tql.questionClass() == MarvinDialog.NEW){
         classNew();
```

```
    }
    //An 'explain'-phrase has been uttered
    else if(tql.questionClass() == MarvinDialog.EXPLAIN){
      classExplain();
    }
    //A 'do'-phrase has been uttered
    else if(tql.questionClass() == MarvinDialog.DO){
      classDo();
    }

    return topicAndType;
  }

  public String getTopic(){
    String[] temp = getTopicAndType();
    return temp[0];
  }

  private void classWhich(){
    topicAndType[2] += "which";

    String whichContent = tql.getWhichContent();
    String whatIs = tql.whatIs(whichContent);

    normalAnswers(whatIs);
  }

  private void classTest(){
    topicAndType[2] += "test";

    String testType = tql.getFirstRelationType();
    String whatIs;
    if(testType.equals("knowthing")){
      topicAndType[2] += "->knowthing";
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(testType.equals("nrel")){
      topicAndType[2] += "->nrel";
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(testType.equals("know")){
      topicAndType[2] += "->know";
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(testType.equals("be1")){
      topicAndType[2] += "->be1";
      whatIs = tql.whatIs(tql.getUnknown());
      normalAnswers(whatIs);
    }
    else if(testType.equals("show")){
      topicAndType[2] += "->show";
      topicAndType[1] = MarvinDialog.SHOW;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(testType.equals("accompany")){
      topicAndType[2] += "->accompany";
      topicAndType[1] = MarvinDialog.ACCOMPANY;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(testType.equals("go")){
      topicAndType[2] += "->go";
      topicAndType[1] = MarvinDialog.GO;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else{
      topicAndType[2] += "->"+testType;
      whatIs = tql.whatIs(tql.getThirdAtom());
      normalAnswers(whatIs);
```

```
    }
  }

  private void classNew(){
    topicAndType[2] += "new";

    String newType = tql.getFirstRelationType();
    String whatIs;
    if(newType.equals("go")){
      topicAndType[2] += "->go";
      topicAndType[1] = MarvinDialog.GO;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(newType.equals("nrel")||newType.equals("wonder")||newType.equals("believe")||
newType.equals("meet")){
      topicAndType[2] += "->"+newType;
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else{
      topicAndType[2] += "->"+newType;
      whatIs = tql.whatIs(tql.getThirdAtom());
      normalAnswers(whatIs);
    }
  }

  private void classExplain(){
    topicAndType[2] += "explain";

    String explainType = tql.getFirstRelationType();
    String whatIs;
    if(explainType.equals("go")){
      topicAndType[2] += "->go";

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(explainType.equals("nrel")){
      topicAndType[2] += "->nrel";
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else{
      topicAndType[2] += "->"+explainType;
      whatIs = tql.whatIs(tql.getThirdAtom());
      normalAnswers(whatIs);
    }
  }

  private void classDo(){
    topicAndType[2] += "do";

    String doType = tql.getFirstRelationType();

    //This if-sentence removes any relation found as first that doesn't provide information about
what to 'do'
    if(doType.equals("nrel") || doType.equals("be1") || doType.equals("know")){
      tql.getUnknownAndRemove();
      doType = tql.getFirstRelationType();
    }

    String whatIs;
    if(doType.equals("knowthing")){
      topicAndType[2] += "->knowthing";
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(doType.equals("nrel")){
      topicAndType[2] += "->nrel";
      tql.getUnknownAndRemove();
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(doType.equals("know")){
      topicAndType[2] += "->know";
```

```
      tql.getUnknownAndRemove();
      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(doType.equals("be1")){
      topicAndType[2] += "->be1";
      whatIs = tql.whatIs(tql.getUnknown());
      normalAnswers(whatIs);
    }
    else if(doType.equals("show")){
      topicAndType[2] += "->show";
      topicAndType[1] = MarvinDialog.SHOW;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(doType.equals("accompany")){
      topicAndType[2] += "->accompany";
      topicAndType[1] = MarvinDialog.ACCOMPANY;

      whatIs = tql.whatIs(tql.getUnknownAndRemove());
      normalAnswers(whatIs);
    }
    else if(doType.startsWith("quit(")){
      topicAndType[2] += "->"+doType;
      topicAndType[0] = "quit";
      if(quitContains(doType).equals("youarewelcome")) topicAndType[1] =
MarvinDialog.YOUAREWELCOME;
      else if(quitContains(doType).equals("bye")) topicAndType[2] = MarvinDialog.BYE;
    }
    else{
      topicAndType[2] += "->"+doType;
      whatIs = tql.whatIs(tql.getThirdAtom());
      normalAnswers(whatIs);
    }
  }

  private void normalAnswers(String w){
    String whatIs = w;

    if(whatIs == null || whatIs.equals("empty")){
      if(tql.personExists()){
        topicAndType[2] += "->person\n";
        topicAndType[0] = "person";
      }
      else if(tql.roomExists()){
        topicAndType[2] += "->room("+tql.getRoom()+")\n";
        if(tql.getRoom().startsWith("free")) topicAndType[0] = "roomfree";
        else topicAndType[0] = "roomnumber";
      }
      else{
        topicAndType[2] += "->empty/null\n";
        if(topicAndType[1].equals(MarvinDialog.PLACE)||
          topicAndType[1].equals(MarvinDialog.GO)||
          topicAndType[1].equals(MarvinDialog.ROUTEPLAN)||
          topicAndType[1].equals(MarvinDialog.ACCOMPANY)){
          topicAndType[0] = "location";
        }
        else topicAndType[0] = "null";
      }
    }
    else if(whatIs.equals("place")){
      if(topicAndType[1].equals(MarvinDialog.DEFAULT)) topicAndType[1] = MarvinDialog.PLACE;
      topicAndType[2] += "->place";
      whatIs = tql.whatIs(whatIs, whatIs);
      normalAnswers(whatIs);
    }
    else if(whatIs.equals("office")){
      topicAndType[2] += "->office";
      whatIs = tql.whatIs(whatIs, whatIs);
      normalAnswers(whatIs);
    }
    else if(whatIs.equals("floor")){
      topicAndType[2] += "->floor";
      whatIs = tql.whatIs(tql.getInverseIs(whatIs), whatIs);
      normalAnswers(whatIs);
    }
```

```
    else if(whatIs.equals("meeting")){
      topicAndType[2] += "->meeting";
      whatIs = tql.whatIs(whatIs, whatIs);
      normalAnswers(whatIs);
    }
    else if(whatIs.equals("routeplan")){
      topicAndType[1] = MarvinDialog.ROUTEPLAN;
      topicAndType[2] += "->routeplan";
      whatIs = tql.whatIs(whatIs, whatIs);
      normalAnswers(whatIs);
    }
    else if(whatIs.equals("street")){
      topicAndType[1] = MarvinDialog.ROUTEPLAN;
      topicAndType[2] += "->street";
      whatIs = tql.whatIs(whatIs, whatIs);
      normalAnswers(whatIs);
    }
    else if(whatIs.equals("room") || whatIs.equals("meetingroom")){
      topicAndType[2] += "->room\n";

      String tempRoomString = tql.getInverseIs(whatIs);
      if(tempRoomString.startsWith("free")) topicAndType[0] = "roomfree";
      else topicAndType[0] = "roomnumber";
    }
    else if(whatIs.equals("telephone")){
      topicAndType[1] = MarvinDialog.TELEPHONE;
      topicAndType[2] += "->telephone\n";
      topicAndType[0] = "telephone";
    }
    else if(whatIs.equals("address")){
      topicAndType[1] = MarvinDialog.ADDRESS;
      topicAndType[2] += "->address\n";
      topicAndType[0] = "address";
    }
    else if(whatIs.equals("title")){
      topicAndType[1] = MarvinDialog.TITLE;
      topicAndType[2] += "->title\n";
      topicAndType[0] = "title";
    }
    else if(whatIs.equals("department")){
      topicAndType[1] = MarvinDialog.DEPARTMENT;
      topicAndType[2] += "->department\n";
      topicAndType[0] = "department";
    }
    else if(isPersonRelated(whatIs) || isSpecific(whatIs)){
      topicAndType[2] += "->"+whatIs+"\n";
      topicAndType[0] = whatIs;
    }
    else{
      if(tql.isA(whatIs) == null){
        topicAndType[2] += "->unknown("+whatIs+")\n";
        topicAndType[0] = "unknown";
      }
      else{
        whatIs = tql.isA(whatIs);
        normalAnswers(whatIs);
      }
    }
  }

  private String quitContains(String s){
    String contains = "";
    StreamTokenizer st = new StreamTokenizer(new StringReader(s));
    st.ordinaryChar('(');
    st.ordinaryChar(')');
    st.ordinaryChar(',');

    try{
      while(st.ttype != '('){
        st.nextToken();
      }
      int parentesCounter = 1;
      while(parentesCounter > 0){
        st.nextToken();
        if(st.sval != null) contains += st.sval;
        else if(st.ttype == StreamTokenizer.TT_NUMBER) contains += (int)st.nval;
        else if(st.ttype == ',') contains += ",";
```

```
          else if(st.ttype == '('){
             contains += "(";
             parentesCounter++;
          }
          else if(st.ttype == ')'){
             if(parentesCounter > 1) contains += ")";
             parentesCounter--;
          }
        }
      }
      catch(Exception e){

      }
      return contains;
    }
}
```

Location.java

```
public class Location{

   private String number;
   private String building;

   public Location(String n, String b){
      number = n;
      building = b;
   }

   public String getRoomNumber(){
      return number;
   }

   public String getBuilding(){
      return building;
   }
}
```

MarvinDialog.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Calendar;
import se.sics.jasper.*;
import java.io.*;
import java.lang.Thread;



/**
 * This is the main class in Marvin's GUI module. It handles the graphical user interface and
communicates
 * with the other parts of the module, as well as with Telebuster through PrologConnection. It also
has some
 * connection with the Robot's Controller through flat files.
 */
public class MarvinDialog extends JFrame implements ActionListener{

   //Files
   public static final String pathFile = "./robotsim/controllers/path";
   public static final String marvinFile = "./robotsim/controllers/marvin";
   public static final String questionFile = "/directinput1";
   public static final String answerFile = "/directoutput1";
   public static final String telebusterPath = "telebuster.sav";

   //Question types
   public static final String SHOW = "show";
   public static final String PLACE = "place";
   public static final String GO = "go";
   public static final String ACCOMPANY = "accompany";
   public static final String VISIT = "visit";
   public static final String ROUTEPLAN = "routeplan";
   public static final String TELEPHONE = "telephone";
   public static final String ADDRESS = "address";
   public static final String TITLE = "title";
   public static final String DEPARTMENT = "department";
   public static final String YOUAREWELCOME = "youarewelcome";
   public static final String BYE = "bye";
   public static final String DEFAULT = "default";

   //TQL classes
   public static final int WHICH = 0;
   public static final int TEST = 1;
   public static final int NEW = 2;
   public static final int EXPLAIN = 3;
   public static final int DO = 4;
   public static final int DUNNO = 9;
   public static final int NO_QUESTION = 10;

   //Dialog states
   public static final int NEW_CONVERSATION = 0;
   public static final int MORE_INFORMATION = 1;
   public static final int UNKNOWN_LOCATION = 2;

   //Marvin's "memory" of previous conversations
   private Record oldRecord;
   private Location oldLocation;
   private String[] oldTopicAndType;
   private int dialogueState;

   private JTextArea output;
   private JTextField input;
   private JButton answerButton;
   private JScrollPane scrollPane;

   private PrologConnection pc;
   private AnswerGenerator ag;
   private Context context;
   private Tql tql;
   private LanguageUnderstanding lu;
```

```
private String currentAnswer;
private int screenWidth;

public boolean details = false;

/**
 * The constructor sets up the GUI. It also makes instances of PrologConnection, AnswerParser and
 * AnswerGenerator.
 */
public MarvinDialog(){

   //Window
   super("Marvin");
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   screenWidth = screenSize.width;
   if(screenSize.width<=1024)      setSize(new Dimension(550,220));
   else if(screenSize.width==1280) setSize(new Dimension(550,350));
   else if(screenSize.width==1600) setSize(new Dimension(500,420));
   setDefaultCloseOperation(EXIT_ON_CLOSE);
   setLocation(0,0);
   setResizable(false);

   //Initialize other modules
   context = new Context();
   tql = new Tql();
   ag = new AnswerGenerator(this, context, tql);
   lu = new LanguageUnderstanding(tql, context);

   //Initialize memory
   oldRecord = null;
   oldLocation = null;
   oldTopicAndType = null;
   dialogueState = NEW_CONVERSATION;

   //Output text area
   if(screenSize.width<=1024) output = new JTextArea("", 10, 30);
   else if(screenSize.width==1280) output = new JTextArea("", 18, 30);
   else if(screenSize.width==1600) output = new JTextArea("", 22, 30);
   scrollPane = new JScrollPane(output);
   output.setEditable(false);
   output.setFocusable(false);
   this.getContentPane().add(scrollPane, BorderLayout.NORTH);

   //Input text field
   input = new JTextField();
   input.addActionListener(this);
   input.addMouseListener(new inputMouseListener(this));
   this.getContentPane().add(input, BorderLayout.CENTER);

   setVisible(true);

   //Initialize Prolog Connection
   try{
      Prolog p = Jasper.newProlog(null,null,telebusterPath);
      pc = new PrologConnection(p, this, questionFile, answerFile);
      printMsg("Prolog connection established");
   }
   catch(Exception e){
      printMsg("Error! Could not establish connection with Telebuster!");
   }

   //Initialize Robot location
   setMarvinLocation(25);
   if(details) printMsg(ag.getLocation(getMarvinLocation()));

   //Initialize Simulator
   setRobotNotReady();
   try { Runtime.getRuntime().exec("startrobot.bat"); } catch(IOException ex) { }
   printMsg("Waiting for simulator to get ready");
   while(!robotReady()){
      try{
         Thread.sleep(500);
      }
      catch(Exception e){

      }
   }
```

```
    printMsg("Robot simulator ready!\n\n");
    //ag.createTestRun();

    //Make dialog window ready for input
    printMsg("Hei, jeg er Marvin. Hvordan kan jeg hjelpe deg?");
    setFocusable(true);
    requestFocus();
    input.requestFocusInWindow();
    input.setText("Skriv her");
    input.selectAll();
  }

  /**
  * Gets location from the information file of Marvin
  *
  * @return   Marvin's location
  */
  public int getMarvinLocation(){
    int locat = 0;
    try{
      FileReader fr = new FileReader(MarvinDialog.marvinFile);
      File infile = new File(MarvinDialog.marvinFile);
      char[] cbuf = new char[(int)infile.length()];
      fr.read(cbuf);
      fr.close();

      String marvinFile = "";
      for(int i=0; i<cbuf.length; i++){
        marvinFile += cbuf[i];
      }

      StreamTokenizer st = new StreamTokenizer(new StringReader(marvinFile));
      st.lowerCaseMode(true);
      int tokenType = st.nextToken();
      while(st.ttype != StreamTokenizer.TT_EOF && (st.sval == null || !
((st.sval).equals("location")))){
          st.nextToken();
      }
      if(tokenType == StreamTokenizer.TT_EOF) locat = 0;
      else if(tokenType == StreamTokenizer.TT_WORD && st.sval.equals("location")){
        st.nextToken();
        tokenType = st.nextToken();
        if (tokenType == StreamTokenizer.TT_NUMBER) locat = (int)st.nval;
      }
      else locat = 0;
    }
    catch(Exception e){
      System.out.println("Error trying to read Marvin's location from file marvin");
      e.printStackTrace(System.out);
    }

    return locat;
  }

  public void endConversation(){
    pc.processQuery("query('nor 123 adjø.').");
  }

  public void selectAllInput(){
    input.selectAll();
  }

  public void storeLocation(Location l){
    oldLocation = l;
  }

  /**
  * Actions to be performed when MarvinDialog is triggered as an actionListener.
  *
  * @param e  ActionEvent
  */
  public void actionPerformed(ActionEvent e){

    //A system call has been entered.
    if(checkSys(input.getText())){
      printMsg(input.getText(), false);
      processSystemCall();
```

```
    }

    //New text was entered and Marvin has not offered guidance
    else if(e.getSource()==input){
      printMsg(input.getText(), false);
      processNormalQuestion();
    }
  }

//---------------------------------Private methods-------------------------------------//


  /**
  * This method is called from actionPerformed, if a normal question has been asked.
  */
  private void processNormalQuestion(){

    String newText = checkTerm(input.getText());                                    //Check if there is
a termination character

    newText = "query('nor 123 "+ newText + "').";                                   //Make prolog query

    currentAnswer = pc.processQuery(newText);                                               //Get
answer from Telebuster

    if(details) printMsg(currentAnswer);

    if(currentAnswer.equals("no answer")){                                                  //If no
answer from Telebuster

      printMsg("\nTelebuster forstod ikke spørsmålet.");                //Print message
      endConversation();
    }
    else{
      ag.killMap();

      context.parseNewAnswer(currentAnswer);                                                //Parse
context from Telebuster answer

      if(context.personFound()){                                                                    //If
a new person was found

        oldRecord = context.getRecord();                                            //Store the new
record as old record
      }

      tql.parseNewAnswer(currentAnswer);                                                    //Parse
TQL-expression from Telebuster answer

      if(details) printMsg(tql.getElementStrings());

      String[] topicAndType = new String[3];

      if(dialogueState==NEW_CONVERSATION){                                                  //If
the dialogue is in 'new conversation'-state

        topicAndType = lu.getTopicAndType();                                        //Get topic and
type from language understanding
        if(details) printMsg("State: New conversation.");
        if(details) printMsg(topicAndType[0]+", "+topicAndType[1]+": "+topicAndType[2]);

      }
      else if(dialogueState==MORE_INFORMATION){                                             //If
the dialogue is in 'get more information'-state

        topicAndType = oldTopicAndType;                                                 //Get topic
and type from previous question
        if(details) printMsg("State: More information.");
        if(details) printMsg("Old question:");
        if(details) printMsg(topicAndType[2]);
      }
      else if(dialogueState==UNKNOWN_LOCATION){                                             //If
the dialogue is in 'unknown location'-state

        topicAndType = oldTopicAndType;                                                 //Copy old
topic and type
```

```
          topicAndType[0] = lu.getTopic();                                          //Get only the
topic from language understanding
          if(details) printMsg("State: Unknown location.");
          if(details) printMsg(topicAndType[2]);
        }
        else{

          printMsg("Programfeil: Dialog-manageren vet ikke hvilken state dialogen er i.");
        }

        ////////////////////////// Is topic familiar?
//////////////////////////////////////////////////////////////////
        if(topicAndType[0]==null || topicAndType[0].equals("null") ||
topicAndType[0].equals("unknown")){                                  //If topic is unknown

          printMsg("\nDu har spurt om noe jeg ikke skjønner.");
          endConversation();
          dialogueState = NEW_CONVERSATION;
        }

        ////////////////////////// Is topic bus?
//////////////////////////////////////////////////////////////////
        else if(topicAndType[0].equals("bus")){                                  //If topic
is bus

          printMsg("\n"+ag.generateBusAnswer());           //Generate answer, no record needed
          endConversation();
          dialogueState = NEW_CONVERSATION;
        }

        ////////////////////////// Is topic person-related?
//////////////////////////////////////////////////////////////
        else if(lu.isPersonRelated(topicAndType[0])){                            //If topic
is person-related

          if(context.personFound()){                                                    //If
person is found in new record

            printMsg("\n"+ag.generatePersonAnswer(topicAndType, context.getRecord()));
  //Generate answer based on new record
            //oldRecord = context.getRecord();                                  //Store the new
record as old record
            endConversation();
  //End conversation in Telebuster
            dialogueState = NEW_CONVERSATION;
  //Set dialogue state to 'new conversation'
          }
          else if(context.missingInformation()){                                 //If
Telebuster is missing information about person

            printMsg("\n"+ag.generateFocusAnswer());                             //Generate
answer based on Telebuster focus
            oldTopicAndType = topicAndType;                                  //Store the
question topic and type
            dialogueState = MORE_INFORMATION;
  //Set dialogue state to 'more information'

          }
          else if(oldRecord!=null){                                              //If there
is a previously stored record

            printMsg("\n"+ag.generatePersonAnswer(topicAndType, oldRecord));
  //Generate answer based on previously stored record
            endConversation();
  //End conversation in Telebuster
            dialogueState = NEW_CONVERSATION;
  //Set dialogue state to 'new conversation'
          }
          else if(!(context.getFocus().equals(""))){
            printMsg("\n"+ag.generateFocusAnswer());                             //Generate
answer based on Telebuster focus
            oldTopicAndType = topicAndType;                                  //Store the
question topic and type
            dialogueState = MORE_INFORMATION;
  //Set dialogue state to 'more information'
          }
          else{
```

```
            printMsg("\nProgramfeil: Personinformasjon ikke funnet og Telebuster har ikke spurt om
mer informasjon");
          }
        }

        ////////////////////////// Is topic about unspecific location?
////////////////////////////////////////////////
        else if(lu.isUnspecificLocation(topicAndType[0])){                          //If topic is about
an unspecific location

          if(oldLocation!=null){                                                                    //If
there is a previously stored location

            printMsg("\n"+ag.generateAnswerFromLocation(topicAndType, oldLocation));     //Generate
answer based on previously stored location
            endConversation();
   //End conversation in Telebuster
            dialogueState = NEW_CONVERSATION;
   //Set dialogue state to 'new conversation'
          }
          else{
   //If there is no previously stored location

            printMsg("\nHvor mener du?");                                                   //Answer
"where?"
            oldTopicAndType = topicAndType;                                         //Store the
question topic and type
            dialogueState = UNKNOWN_LOCATION;
   //Set the dialogue state to 'more information'
          }
        }

        ////////////////////////// Is topic about something specific?
////////////////////////////////////////////////
        else if(lu.isSpecific(topicAndType[0])){                                    //Is topic
is about something specific

          printMsg("\n"+ag.generateSpecificAnswer(topicAndType));         //Generate answer, no
record needed
          endConversation();
   //End conversation in Telebuster
          dialogueState = NEW_CONVERSATION;
   //Set the dialogue state to 'new conversation'
        }
      }

      input.selectAll();
   }

   /**
   * This method is called when a system call has been made.
   * System calls start with the character '|'
   *
   * @param s  The system call that has been made.
   */
   private void processSystemCall(){
      String s = input.getText();
      StreamTokenizer st = new StreamTokenizer(new StringReader(s));
        st.ordinaryChar('|');
      try{
        st.nextToken();
      }
      catch(Exception e){
        System.out.println("Exception trying nextToken() in checkSys()");
        System.out.println(e.toString());
      }
      if(st.ttype == '|'){
        try{
          st.nextToken();
        }
        catch(Exception e){
          System.out.println("Exception trying nextToken() in checkSys()");
          System.out.println(e.toString());
        }

        //User has set a new location for Marvin
```

```
    if(st.sval != null && (st.sval).equals("location")){

      while(st.ttype != StreamTokenizer.TT_NUMBER && st.ttype != StreamTokenizer.TT_EOF){
        try{
          st.nextToken();
        }
        catch(Exception e){
          e.printStackTrace(System.out);
        }
        if(st.ttype == StreamTokenizer.TT_EOF){
          printMsg("Not a valid location");
        }
        else if(st.ttype == StreamTokenizer.TT_NUMBER){
          if((int)st.nval > 363 || (int)st.nval < 300) printMsg("Not a valid location");
          else setMarvinLocation(((int)st.nval) - 300);
        }
      }
    }
    //User wants to clear the commands file to the robot
    else if(st.sval != null && (st.sval).equals("clear")){
      try{
        //Clear path file
        FileWriter fw = new FileWriter(MarvinDialog.pathFile);
        fw.write(".");
        fw.close();
        printMsg(ag.getClearedCommands());
      }
      catch(Exception e){
        e.printStackTrace(System.out);
      }
    }
    //Turning details on/off
    else if(st.sval != null && (st.sval).equals("details")){
      if(details){
        details = false;
        printMsg("Details turned off");
      }
      else if(!details){
        details = true;
        printMsg("Details turned on");
      }
    }

    //User has made an illegal system call
    else printMsg("System call not understood");
  }
  else printMsg("System call not understood");

  input.selectAll();
}

/**
* Checks if the question asked has a terminating character and puts in a
* terminating character if not.
*/
private String checkTerm(String s){
  StreamTokenizer st = new StreamTokenizer(new StringReader(s));
  st.ordinaryChar('.');
  st.ordinaryChar('?');
  st.ordinaryChar('!');
  try{
    st.nextToken();
  }
  catch(Exception e){
    System.out.println("Exception trying nextToken() in checkTerm()");
    System.out.println(e.toString());
  }
  int previous_type = st.ttype;
  while(!(st.ttype == StreamTokenizer.TT_EOF)){
    previous_type = st.ttype;
    try{
      st.nextToken();
    }
    catch(Exception e){
      System.out.println("Exception trying nextToken() in checkTerm()");
      System.out.println(e.toString());
    }
```

```
    }

    if(!(previous_type == '.' || previous_type == '?' || previous_type == '!')){
      s += '.';
    }
    return s;
  }

  /**
  * Checks if the input text was a system call
  */
  private boolean checkSys(String s){
    StreamTokenizer st = new StreamTokenizer(new StringReader(s));
    st.ordinaryChar('|');
    try{
      st.nextToken();
    }
    catch(Exception e){
      System.out.println("Exception trying nextToken() in checkSys()");
      System.out.println(e.toString());
    }
    if(st.ttype == '|') return true;
    else return false;
  }

  /**
  * Simple method to print a string to the textarea.
  *
  * @param s  String to be printed out.
  */
  private void printMsg(String s, boolean b){

    String timeString = "";

    //Some code lines to add a nice time stamp
    Calendar time = Calendar.getInstance();
    if(time.get(Calendar.HOUR_OF_DAY) >= 10){
      timeString += time.get(Calendar.HOUR_OF_DAY) + ":";
    }
    else{
      timeString += "0" + time.get(Calendar.HOUR_OF_DAY) + ":";
    }
    if(time.get(Calendar.MINUTE) >= 10){
      timeString += time.get(Calendar.MINUTE) + ":";
    }
    else{
      timeString += "0" + time.get(Calendar.MINUTE) + ":";
    }
    timeString += " | ";

    //Output to the text area
    if(b) output.append(timeString + " Marvin>\n"+ s + "\n");
    else output.append(timeString + " User> "+ s + "\n");

    //Makes the text area scroll down to the bottom
    JScrollBar bar = scrollPane.getVerticalScrollBar();
    bar.setValue(bar.getMaximum());
    output.setCaretPosition(output.getText().length());
  }

  private void printMsg(String s){

    //Output to the text area
    output.append(s + "\n");

    //Makes the text area scroll down to the bottom
    JScrollBar bar = scrollPane.getVerticalScrollBar();
    bar.setValue(bar.getMaximum());
    output.setCaretPosition(output.getText().length());
  }


  private void setRobotNotReady(){
    try{
      String toWrite ="no";

      String marvinFile = "";
```

```
      FileReader fr = new FileReader(MarvinDialog.marvinFile);
      File infile = new File(MarvinDialog.marvinFile);
      char[] cbuf = new char[(int)infile.length()];
      fr.read(cbuf);
      fr.close();
      //Buffer to string
      for(int i=0; i<cbuf.length; i++){
        marvinFile += cbuf[i];
      }
      StringBuffer sb = new StringBuffer(marvinFile);
      int writePosition = 0;

      for(int i=0; i<sb.length(); i++){
        if(i+8 > sb.length()){
          System.out.println("Didn't find 'SimulatorReady' in file marvin");
          i=sb.length();
        }
        else if(sb.substring(i, i+14).equals("SimulatorReady")){
          writePosition = i+15;
          i = sb.length();
        }
      }
      if(writePosition != 0){
        sb = sb.replace(writePosition, writePosition+3, toWrite);
        FileWriter fw = new FileWriter(MarvinDialog.marvinFile);
        fw.write(sb.toString());
        fw.close();
      }
    }
    catch(Exception e){
      System.out.println("Error writing to file marvin");
      e.printStackTrace(System.out);
    }
  }

  /**
  * Method that sets the Marvin location.
  *
  * @param location          New location for Marvin.
  */
  private void setMarvinLocation(int location){
    try{
      String toWrite ="";
      if(location < 10) toWrite = "  "+location;
      else if(location <100) toWrite = " "+location;
      else toWrite = ""+location;

      String marvinFile = "";
      FileReader fr = new FileReader(MarvinDialog.marvinFile);
      File infile = new File(MarvinDialog.marvinFile);
      char[] cbuf = new char[(int)infile.length()];
      fr.read(cbuf);
      fr.close();
      //Buffer to string
      for(int i=0; i<cbuf.length; i++){
        marvinFile += cbuf[i];
      }
      StringBuffer sb = new StringBuffer(marvinFile);
      int writePosition = 0;

      for(int i=0; i<sb.length(); i++){
        if(i+8 > sb.length()){
          System.out.println("Didn't find 'Location' in file marvin");
          i=sb.length();
        }
        else if(sb.substring(i, i+8).equals("Location")){
          writePosition = i+10;
          i = sb.length();
        }
      }
      if(writePosition != 0){
        sb = sb.replace(writePosition, writePosition+3, toWrite);
        FileWriter fw = new FileWriter(MarvinDialog.marvinFile);
        fw.write(sb.toString());
        fw.close();
        if(details) printMsg(ag.getSetMarvinLocation(location));
      }
```

```
      }
      catch(Exception e){
        System.out.println("Error writing to file marvin");
        e.printStackTrace(System.out);
      }
    }

    private boolean robotReady(){
      try{
        FileReader fr = new FileReader(MarvinDialog.marvinFile);
        File infile = new File(MarvinDialog.marvinFile);
        char[] cbuf = new char[(int)infile.length()];
        fr.read(cbuf);
        fr.close();

        String marvinFile = "";
        for(int i=0; i<cbuf.length; i++){
          marvinFile += cbuf[i];
        }

        StreamTokenizer st = new StreamTokenizer(new StringReader(marvinFile));
        while(st.ttype != StreamTokenizer.TT_EOF && !(st.sval != null &&
st.sval.equals("SimulatorReady"))){
          st.nextToken();
        }
        if(st.ttype == StreamTokenizer.TT_WORD && st.sval.equals("SimulatorReady")){
          st.nextToken();
          st.nextToken();
          if (st.ttype == StreamTokenizer.TT_WORD){
            if(st.sval.equals("yes")) return true;
          }
        }
      }
      catch(Exception e){
        System.out.println("Error trying to read robot ready from file marvin");
        e.printStackTrace(System.out);
      }

      return false;
    }


//---------------------------------Main method-----------------------------------------//
    /**
    * Main method makes an instance of itself.
    */
    public static void main(String args[]){
      new MarvinDialog();
    }
}

class inputMouseListener implements MouseListener{

    private MarvinDialog md;

    public inputMouseListener(MarvinDialog md){
      super();
      this.md = md;
    }

    public void mouseReleased(MouseEvent e) {
      md.selectAllInput();
    }

    public void mousePressed(MouseEvent e) {
      }
      public void mouseEntered(MouseEvent e) {
      }
      public void mouseExited(MouseEvent e) {
      }
      public void mouseClicked(MouseEvent e) {
      }
}
```

Node.java

```
import java.lang.Math;

public class Node{

   private int roomNumber;
   private int xPos;
   private int yPos;
   private Node parent;
   private int fHat;
   private int cost;

   public Node(int number, int x, int y, Node parent, int cost){
      roomNumber = number;
      this.parent = parent;
      xPos = x;
      yPos = y;
      this.cost = cost;

      calcFHat();
   }

   public int calcPathCost(){
      int r = 0;

      if(parent == null) r = 0;
      else{
         r = cost + parent.calcPathCost();
      }
      return r;
   }

   public int calcPathCost(int newCost, Node newParent){
      int r = 0;

      if(newParent == null) r = 0;
      else{
         r = newCost + newParent.calcPathCost();
      }
      return r;
   }

   private void calcFHat(){
      if(parent == null) fHat = 0;
      else{
         fHat = parent.getFHat() + (int) (
              Math.sqrt(
                 (xPos-parent.getX())*(xPos-parent.getX())+(yPos-parent.getY())*(yPos-parent.getY())
              )
            );
      }
   }

   public boolean notAncestor(int a){

      boolean recur = true;

      if(roomNumber == a) recur = false;
      else if(parent == null) recur = true;
      else{
         recur = parent.notAncestor(a);
      }
      return recur;
   }

   public void setNewCost(int c){
      cost = c;
   }

   public void setNewParent(Node n){
      parent = n;
   }
```

```
  public int getFHat(){
     return fHat;
  }

  public int getNr(){
     return roomNumber;
  }

  public Node getParent(){
     return parent;
  }

  public int getX(){
     return xPos;
  }

  public int getY(){
     return yPos;
  }

  public int getCost(){
     return cost;
  }
}
```

PrologConnection.java

```
import se.sics.jasper.*;
import java.util.HashMap;
import java.io.*;

public class PrologConnection{

  private Prolog p;
  private MarvinDialog md;
  private FileWriter fw;
  private FileReader fr;

  private String questionFile;
  private String answerFile;
  private String currentDir;


  public PrologConnection(Prolog p, MarvinDialog md, String qF, String aF){
    this.p = p;
    this.md = md;
    this.questionFile = qF;
    this.answerFile = aF;

    File dir1 = new File (".");
    currentDir = "";
    try {
      currentDir = dir1.getCanonicalPath();
    }
    catch(Exception e) {
      e.printStackTrace();
    }

    StringBuffer sb = new StringBuffer(currentDir);
    for(int i=0; i<sb.length(); i++){
      Character newChar = new Character(sb.charAt(i));
      if( newChar.equals('\\') ){
        sb.setCharAt(i, '/');
      }
    }

    currentDir = sb.toString();
  }

  /**
  * Method for processing queries to Telebuster.
  */
  public String processQuery(String s){

  //Write question to question file
    try{
      fw = new FileWriter(currentDir+questionFile);
    }
    catch(IOException e){
      System.out.println("IOException trying to open filewriter for "+currentDir+questionFile+":
\n");
      System.out.println(e.toString());
    }
    try{
      fw.write(s);
    }
    catch(IOException e){
      System.out.println("IOException trying to write "+currentDir+questionFile+": \n");
      System.out.println(e.toString());
    }
    try{
      fw.close();
    }
    catch(Exception e){
      System.out.print("Exception trying to close filewriter for "+currentDir+questionFile+": \n");
      System.out.println(e.toString());
    }

  //Write default answer to answer file
```

```
    try{
         fw = new FileWriter(currentDir+answerFile);
       }
       catch(IOException e){
         System.out.println("IOException trying to open filewriter for "+currentDir+answerFile+":
\n");
         System.out.println(e.toString());
       }
       try{
         fw.write("no answer");
       }
       catch(IOException e){
         System.out.println("IOException trying to write "+currentDir+answerFile+": \n");
         System.out.println(e.toString());
       }
       try{
         fw.close();
       }
       catch(Exception e){
         System.out.println(e.toString());
    }

  //Perform query
    try{
      p.query("direct_run('" + currentDir+questionFile + "','" + currentDir+answerFile + "').",
null);
    }
    catch(Exception e){
      System.out.println("Exception while querying 'direct_run('" + currentDir+questionFile + "','"
+ currentDir+answerFile + "').:");
      System.out.println(e.toString());
    }

    String result = readAnswer();

    return result;
  }

  /**
  * Reads answer from the answer file.
  */
  public String readAnswer(){

    String result = "";

  //Read from newans
    try{
      fr = new FileReader(currentDir+answerFile);
    }
    catch(IOException e){
      System.out.println("IOException trying to open filereader for newans:");
      System.out.println(e.toString());
    }
    File infile = new File(currentDir+answerFile);
    char[] cbuf = new char[(int)infile.length()];
    try{
      fr.read(cbuf);
    }
    catch(IOException e){
      System.out.println(e.toString());
    }
    try{
      fr.close();
    }
    catch(Exception e){
      System.out.println(e.toString());
    }
    for(int i=0; i<cbuf.length; i++){
      result += cbuf[i];
    }

  //Return the result
    return result;
  }
}
```

Record.java

```
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.lang.*;


public class Record implements Cloneable{

  private boolean personFound;

  private String department;
  private String fullName;
  private int phoneNumber;
  private String building;
  private String roomNumber;
  private String title;
  private String street;

  public Record(String s){

    extractDepartment(s);
    extractName(s);
    phoneNumber = 0;
    extractPhone(s);
    extractBuildingAndRoom(s);
    extractTitle(s);
    extractStreet(s);
    personFound = personFound(s);
  }

  private boolean personFound(String s){
    if(fullName.equals("")) return false;
    else return true;
  }

  private void extractDepartment(String s){
    department = "";
    try{
      StreamTokenizer st = new StreamTokenizer(new StringReader(s));
      st.ordinaryChar(' ');

      while(!(st.sval != null && st.sval.equals("record")) && st.ttype != StreamTokenizer.TT_EOF){
        st.nextToken();
      }
      while(!(st.sval != null && st.sval.equals("ou")) && st.ttype != StreamTokenizer.TT_EOF){
        st.nextToken();
      }
      while(!(st.sval != null && st.sval.equals("cn")) && st.ttype != StreamTokenizer.TT_EOF){
        if(st.sval != null && st.sval.equals("ou")) department += "\n";
        else if(st.sval != null) department += st.sval;
        if(st.ttype == ' ') department += " ";
        st.nextToken();
      }
    }
    catch(Exception e){
      System.out.println(e.toString());
    }
  }

  private void extractName(String s){
       fullName = "";
       try{
         StreamTokenizer st = new StreamTokenizer(new StringReader(s));
         st.ordinaryChar(' ');
         st.ordinaryChar(',');

         while(!(st.sval != null && st.sval.equals("record")) && st.ttype !=
StreamTokenizer.TT_EOF){
           st.nextToken();
         }
         while(!(st.sval != null && st.sval.equals("cn")) && st.ttype != StreamTokenizer.TT_EOF){
           st.nextToken();
         }
```

```
        st.nextToken();
        while(st.ttype != ',' && st.ttype != StreamTokenizer.TT_EOF){
          if(st.sval != null) fullName += st.sval;
          if(st.ttype == ' ') fullName += " ";
          st.nextToken();
        }
      }
      catch(Exception e){
        System.out.println(e.toString());
      }
  }

  private void extractPhone(String s){
       try{
         StreamTokenizer st = new StreamTokenizer(new StringReader(s));
         st.ordinaryChar(',');

         while(!(st.sval != null && st.sval.equals("record")) && st.ttype !=
StreamTokenizer.TT_EOF){
           st.nextToken();
         }
         while(!(st.sval != null && st.sval.equals("telephonenumber")) && st.ttype !=
StreamTokenizer.TT_EOF){
           st.nextToken();
         }
         while(st.ttype != ',' && st.ttype != StreamTokenizer.TT_EOF){
           if(st.ttype == StreamTokenizer.TT_NUMBER) phoneNumber = (int)st.nval;
           st.nextToken();
         }
       }
       catch(Exception e){
         System.out.println(e.toString());
       }
  }

  private void extractBuildingAndRoom(String s){
         building = "";
         roomNumber = "";
         try{
           StreamTokenizer st = new StreamTokenizer(new StringReader(s));
           st.ordinaryChar(' ');
           st.ordinaryChar(',');
           st.ordinaryChar('*');
           st.ordinaryChar('-');
           st.ordinaryChar('0');
           st.ordinaryChar('1');
           st.ordinaryChar('2');
           st.ordinaryChar('3');
           st.ordinaryChar('4');
           st.ordinaryChar('5');
           st.ordinaryChar('6');
           st.ordinaryChar('7');
           st.ordinaryChar('8');
           st.ordinaryChar('9');

           while(!(st.sval != null && st.sval.equals("record")) && st.ttype !=
StreamTokenizer.TT_EOF){
             st.nextToken();
           }
           while(!(st.sval != null && st.sval.equals("roomnumber")) && st.ttype !=
StreamTokenizer.TT_EOF){
             st.nextToken();
           }
           st.nextToken();
           while(st.ttype != '*' && st.ttype != StreamTokenizer.TT_EOF){
             if(st.sval != null) building += st.sval;
             else if(st.ttype == ' ') building += " ";
             else if(st.ttype == '-') building += "-";
             st.nextToken();
           }
           while(st.ttype != ',' && st.ttype != StreamTokenizer.TT_EOF){
             if(st.ttype == '0') roomNumber += "0";
             if(st.ttype == '1') roomNumber += "1";
             if(st.ttype == '2') roomNumber += "2";
             if(st.ttype == '3') roomNumber += "3";
             if(st.ttype == '4') roomNumber += "4";
             if(st.ttype == '5') roomNumber += "5";
```

```
            if(st.ttype == '6') roomNumber += "6";
            if(st.ttype == '7') roomNumber += "7";
            if(st.ttype == '8') roomNumber += "8";
            if(st.ttype == '9') roomNumber += "9";
            st.nextToken();
          }
        }
        catch(Exception e){
          System.out.println(e.toString());
        }

    if(building.startsWith("Dragv")) building = "Dragvoll";
  }

  private void extractTitle(String s){
        title = "";
        try{
          StreamTokenizer st = new StreamTokenizer(new StringReader(s));
          st.ordinaryChar(',');
          st.ordinaryChar(' ');

          while(!(st.sval != null && st.sval.equals("record")) && st.ttype !=
StreamTokenizer.TT_EOF){
            st.nextToken();
          }
          while(!(st.sval != null && st.sval.equals("title")) && st.ttype !=
StreamTokenizer.TT_EOF){
            st.nextToken();
          }
          st.nextToken();
          while(st.ttype != ',' && st.ttype != StreamTokenizer.TT_EOF){
            if(st.sval != null) title += st.sval;
            else if(st.ttype == ' ') title += " ";
            st.nextToken();
          }
        }
        catch(Exception e){
          System.out.println(e.toString());
        }
  }

  private void extractStreet(String s){
          street = "";
          try{
            StreamTokenizer st = new StreamTokenizer(new StringReader(s));
            st.ordinaryChar(',');
            st.ordinaryChar(' ');
            st.ordinaryChar('.');

            while(!(st.sval != null && st.sval.equals("record")) && st.ttype !=
StreamTokenizer.TT_EOF){
             st.nextToken();
            }
            while(!(st.sval != null && st.sval.equals("street")) && st.ttype !=
StreamTokenizer.TT_EOF){
             st.nextToken();
            }
            st.nextToken();
            while(st.ttype != ']' && st.ttype != StreamTokenizer.TT_EOF){
             if(st.sval != null) street += st.sval;
             else if(st.ttype == ' ') street += " ";
             else if(st.ttype == '.') street += ".";
             else if(st.ttype == StreamTokenizer.TT_NUMBER) street += (int)st.nval;
             st.nextToken();
            }
          }
          catch(Exception e){
            System.out.println(e.toString());
          }
  }

  public String recordString(){
    return fullName+department+"\nTelefonnummer: "+phoneNumber+"\nKontor: "+roomNumber+" i
"+building+"\nStilling: "+title+"\nAdresse: "+street;
  }

  public String getDepartment(){
```

```
    return department;
  }

  public String getFullName(){
    return fullName;
  }

  public int getPhoneNumber(){
    return phoneNumber;
  }

  public String getBuilding(){
    if(building=="") return street;
    else return building;
  }

  public String getRoomNumber(){
    return roomNumber;
  }

  public String getTitle(){
    return title;
  }

  public String getStreet(){
    return street;
  }

  public boolean personFound(){
    return personFound;
  }

  public Object clone(){
    try{
      return super.clone();
    }
    catch(Exception e){

    }
    return null;
  }
}
```

Tql.java

```
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.util.ArrayList;

public class Tql{

  private ArrayList tqlElements;

  public void parseNewAnswer(String s){
       resetTql();

       StreamTokenizer st = new StreamTokenizer(new StringReader(s));

       st.ordinaryChar(',');
       st.ordinaryChar('*');
       st.ordinaryChar('(');
       st.ordinaryChar(')');
       st.ordinaryChar(' ');
       st.ordinaryChar('/');

       try{
         while(st.ttype != StreamTokenizer.TT_EOF){
           if(st.sval != null && st.sval.equals("TQL")){
             st.nextToken();
             while(st.ttype == '*' || st.ttype == ' '){
              st.nextToken();
             }
             while(st.ttype != '*'){
              String newElementString = "";
              int parentesCounter = 0;
              while((st.ttype != ',' && st.ttype != '*') || parentesCounter > 0){
                     if(st.sval != null){
                            newElementString += st.sval;
                     }
                     else{
                            if(st.ttype == StreamTokenizer.TT_NUMBER){
                                   newElementString += (int)st.nval;
                            }
                            else if(st.ttype == ' '){
                                   newElementString += " ";
                            }
                            else if(st.ttype == ','){
                                   newElementString += ",";
                            }
                            else if(st.ttype == '('){
                                   newElementString += "(";
                                   //inParentes = true;
                                   parentesCounter ++;
                            }
                            else if(st.ttype == ')'){
                                   newElementString += ")";
                                   //inParentes = false;
                                   parentesCounter --;
                            }
                            else if(st.ttype == '/'){
                                   newElementString += "/";
                            }
                     }
                     st.nextToken();
              }
              tqlElements.add(new TqlElement(newElementString));
              st.nextToken();
             }
           }
           st.nextToken();
         }
       }
       catch(Exception e){
         System.out.println("Exception while performing nextToken() in Context.parseNewAnswer():");
         System.out.println(e.toString());
       }
```

```
       //In the end, we remove all the event/real/free(x) elements, since we don't need them to
extract the meaning we are after
       for(int i=0; i<tqlElements.size(); i++){
         TqlElement testElement = (TqlElement)tqlElements.get(i);
         ArrayList testList = (ArrayList) testElement.getAtoms();
         if( ((String)(testList.get(0))).equals("event")&&
            ((String)(testList.get(1))).equals("real")){
              tqlElements.remove(i);
              i--;
            }
       }
   }

   public String getElementStrings(){
     String returnString = "";
     if(!tqlElements.isEmpty()){
       for(int i=0; i<tqlElements.size(); i++){
         returnString += ((TqlElement)tqlElements.get(i)).getElementString() + "\n";
       }
       return returnString;
     }
     else return "No elements";
   }

   public int questionClass(){
     if(!tqlElements.isEmpty()){
       String firstElementString = ((TqlElement) tqlElements.get(0)).getElementString();
       StreamTokenizer st = new StreamTokenizer(new StringReader(firstElementString));
       try{
         st.nextToken();
       }
       catch(Exception e){}
       if(st.sval.equals("which")) return MarvinDialog.WHICH;
       else if(st.sval.equals("test")) return MarvinDialog.TEST;
       else if(st.sval.equals("new")) return MarvinDialog.NEW;
       else if(st.sval.equals("do")) return MarvinDialog.DO;
       else if(st.sval.equals("explain")) return MarvinDialog.EXPLAIN;
       else return MarvinDialog.DUNNO;
     }
     else{
       return MarvinDialog.NO_QUESTION;
     }
   }

   public void resetTql(){
     tqlElements = new ArrayList();
   }

   ////////////////////////////////////////
   //Methods mainly used by AnswerGenerator//
   ////////////////////////////////////////

   /**
   * Returns the content of a which(x)-element
   */
   public String getWhichContent(){
     TqlElement firstElement = (TqlElement) tqlElements.get(0);
     return firstElement.getWhichContent();
   }

   /**
   * Returns the type of the first relation element. Used to find what type of 'test'-question the
user has asked
   */
   public String getFirstRelationType(){
     for(int i=0; i<tqlElements.size(); i++){
       TqlElement testElement = (TqlElement) tqlElements.get(i);
       if(testElement.isRelation()){
         return testElement.getFirstAtom();
       }
     }
     return null;
   }

   /**
   * Return what something is, based on the current TQL-expression. This method will only go one
depth before it returns what something is.
```

```
   */
   public String whatIs(String s){
      return findRelation(s, "", "", true);
   }

   public String whatIs(String s, String bad){
      return findRelation(s, "", bad, true);
   }

   /**
   * Returns 'inverse is'. If we have an element "free(1)isa place", and String s == place, the
method returns free(1)
   */
   public String getInverseIs(String s){
      for(int i=0; i<tqlElements.size(); i++){
        TqlElement testElement = (TqlElement) tqlElements.get(i);
        if(testElement.isIs()){
          ArrayList atoms = testElement.getAtoms();
          if( ((String)(atoms.get(2))).equals(s) ){
            return (String)atoms.get(0);
          }
        }
      }
      return null;
   }

   /**
   * Returns the unknown element in a relation. Also removes the element.
   */
   public String getUnknownAndRemove(){
      String returnString;
      for(int i=0; i<tqlElements.size(); i++){
        TqlElement testElement = (TqlElement) tqlElements.get(i);
        if(testElement.isRelation()){
          returnString = getUnknown(testElement);
          if(returnString == null) returnString = getLast(testElement);    //If there are no
unknowns, take the last atom
          tqlElements.remove(i);
          return returnString;
        }
      }
      return null;
   }

   /**
   * Returns the unknown element in a relation. Also removes the element.
   */
   public String getUnknown(){
      String returnString;
      for(int i=0; i<tqlElements.size(); i++){
        TqlElement testElement = (TqlElement) tqlElements.get(i);
        if(testElement.isRelation()){
          returnString = getUnknown(testElement);
          if(returnString == null) returnString = getLast(testElement);                          //If
there are no unknowns, take the last atom
          return returnString;
        }
      }
      return null;
   }

   /**
   * Returns the third atom from the first relation element.
   */
   public String getThirdAtom(){
      for(int i=0; i<tqlElements.size(); i++){
        TqlElement testElement = (TqlElement) tqlElements.get(i);
        if(testElement.isRelation()){
          ArrayList atoms = testElement.getAtoms();
          return (String)atoms.get(2);
        }
      }
      return null;
   }

   /**
   * Returns a room number, if one exists.
```

```
  */
  public String getRoom(){
    String returnString;
    for(int i=0; i<tqlElements.size(); i++){
      TqlElement testElement = (TqlElement) tqlElements.get(i);
      if(testElement.isIs()){
        ArrayList atoms = testElement.getAtoms();
        if( ((String)atoms.get(2)).equals("room") )       return ((String)atoms.get(0));
      }
    }
    return null;
  }

  /**
  * Checks if the String s is the first atom in a 'isa'-element and returns the third atom if found
  */
  public String isA(String s){
    String returnString;
    for(int i=0; i<tqlElements.size(); i++){
      TqlElement testElement = (TqlElement) tqlElements.get(i);
      if(testElement.isIs()){
        ArrayList atoms = testElement.getAtoms();
        if( ((String)atoms.get(0)).equals(s) )    return ((String)atoms.get(2));
      }
    }
    return null;
  }


  /**
  * Returns true if the TQL-expression contains a person, firstname or lastname
  */
  public boolean personExists(){
    for(int i=0; i<tqlElements.size(); i++){
      TqlElement testElement = (TqlElement) tqlElements.get(i);
      if(testElement.isIs()){
        ArrayList atoms = testElement.getAtoms();
        String testIs = (String)atoms.get(2);
        if(testIs.equals("person") || testIs.equals("firstname") || testIs.equals("lastname"))
return true;
      }
    }
    return false;
  }

  /**
  * Returns true if the TQL-expression contains a room
  */
  public boolean roomExists(){
    for(int i=0; i<tqlElements.size(); i++){
      TqlElement testElement = (TqlElement) tqlElements.get(i);
      if(testElement.isIs()){
        ArrayList atoms = testElement.getAtoms();
        String testIs = (String)atoms.get(2);
        if(testIs.equals("room")) return true;
      }
    }
    return false;
  }


  ////////////////////////////////////////
  //Private methods //////////////////////
  ////////////////////////////////////////

  /**
  * This method is called recursivly to find what the content of String s actually is.
  *
  * @param badAnswer         True if the method should not return String s as an answer.
  * @param previousAnswer    Is used to avoid going into an infinite loop.
  */
  private String findRelation(String s, String previousAnswer, String dontAnswer, boolean
badAnswer){
    if(s==null) return null;
    else if(!s.startsWith("free(") && !s.startsWith("(") && !Character.isDigit(s.charAt(0)) && !
badAnswer){
      if(!s.equals(dontAnswer)) return s;
```

```
      else return null;
    }
    else if(s.startsWith("(")){
      return "person";
    }
    else{
      for(int i=0; i<tqlElements.size(); i++){
        TqlElement testElement = (TqlElement) tqlElements.get(i);
        //First check the "isa" elements
        if(testElement.isIs()){
          ArrayList atoms = testElement.getAtoms();
          if( ((String)(atoms.get(0))).equals(s) ){
            if(!((String)atoms.get(2)).equals(dontAnswer)){
             return (String)atoms.get(2);
            }
          }
        }
        //If the content of String s wasn't found in the "isa" elements, check the relation
elements
        if(testElement.isRelation()){
          ArrayList atoms = testElement.getAtoms();
          for(int j=0; j<atoms.size(); j++){
            if( ((String)atoms.get(j)).equals(s) ){
             tqlElements.remove(i);
             String nextToFind = getRelationFromElement(atoms, s);
             if(nextToFind != null && nextToFind.equals(previousAnswer)) return "empty";
             else return findRelation(getRelationFromElement(atoms, s), s, dontAnswer, false);
            }
          }
        }
      }
      return null; //No relation found
    }
  }

  /**
  * This method is called with the atoms from a TQL-element and a String as argument, and should
return
  * whatever thing within the TQL-element that the String is related to.
  */
  private String getRelationFromElement(ArrayList atoms, String s){
    String relationType = (String)atoms.get(0);
    if(relationType.equals("be1")){
      if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(2);
      else return (String)atoms.get(1);
    }
    else if(relationType.equals("be2")){
      if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(2);
      else if( ((String)atoms.get(3)).equals(s) ) return (String)atoms.get(2);
      else{
        return (String)atoms.get(1);
      }
    }
    else if(relationType.equals("sit")){
      if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(2);
      else return (String)atoms.get(1);
    }
    else if(relationType.equals("live") || relationType.equals("work")){
      if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(2);
      else return (String)atoms.get(1);
    }
    else if(relationType.equals("has")){
      if( ((String)atoms.get(3)).equals(s) ) return (String)atoms.get(4);
      else if( ((String)atoms.get(4)).equals(s) ) return (String)atoms.get(3);
      else if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(3);
      else if( ((String)atoms.get(2)).equals(s) ) return (String)atoms.get(4);
      else return null;
    }
    else if(relationType.equals("srel")){
      if( ((String)atoms.get(3)).equals(s) ) return (String)atoms.get(4);
      else if( ((String)atoms.get(4)).equals(s) ) return (String)atoms.get(3);
      else if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(3);
      else if( ((String)atoms.get(2)).equals(s) ) return (String)atoms.get(4);
      else return null;
    }
    else if(relationType.equals("nrel")){
      if( ((String)atoms.get(4)).equals(s) ) return (String)atoms.get(5);
```

```
      else if( ((String)atoms.get(5)).equals(s) ) return (String)atoms.get(4);
      else if( ((String)atoms.get(2)).equals(s) ) return (String)atoms.get(4);
      else if( ((String)atoms.get(3)).equals(s) ) return (String)atoms.get(5);
      else return null;
    }
    else if(relationType.equals("event")){
      if( ((String)atoms.get(1)).equals(s) ) return (String)atoms.get(2);
      else return (String)atoms.get(1);
    }
    else if(relationType.equals("knowthing") || relationType.equals("tell")){
      if( ((String)atoms.get(3)).equals(s) ) return (String)atoms.get(2);
      else return (String)atoms.get(3);
    }
    else return null;
  }

  private String getUnknown(TqlElement testElement){
    ArrayList atoms = testElement.getAtoms();
    for(int i=0; i<atoms.size(); i++){
      if( ((String)atoms.get(i)).startsWith("free(") ) return (String)atoms.get(i);
    }
    return null;
  }

  private String getLast(TqlElement testElement){
    ArrayList atoms = testElement.getAtoms();
    return ((String)atoms.get(atoms.size()-1));
  }
}
```

Tql.java

```
import java.io.StreamTokenizer;
import java.io.StringReader;
import java.util.ArrayList;

public class TqlElement{

  public static final int CLASS = 0;
  public static final int IS = 1;
  public static final int RELATION = 2;

  String elementString;
  String firstAtom;
  int type;
  ArrayList atoms;

  /**
  * Constructor that puts all the atoms in the TQL-element into an ArrayList. Also sets what kind
of TQL-element this is (class, is, relation),
  * and sets the first atom to String firstAtom for easier access.
  */
  public TqlElement(String s){
    this.elementString = s;
    atoms = new ArrayList();
    StreamTokenizer st = new StreamTokenizer(new StringReader(elementString));
    st.ordinaryChar('(');
    st.ordinaryChar(')');
    st.ordinaryChar(',');
    st.ordinaryChar('/');
    st.ordinaryChar(' ');
    st.ordinaryChar(92);

    String newElement ="";
    boolean done = false;

    //The parenteses counter routine is used to be able to separate atoms in Strings like
(tore,amble)isa person
    boolean parenteses = false;
    boolean fnutt = false;
    int parentesesCounter = 0;

    try{
      st.nextToken();

      while(!done){
        if((st.ttype == ' ' || st.ttype == '/' || st.ttype == StreamTokenizer.TT_EOF) && !
parenteses && !fnutt){
          if(!newElement.equals("")) atoms.add(newElement);
          parenteses = false;
          parentesesCounter = 0;
          newElement = "";
        }
        else if((parenteses && parentesesCounter == 0)){
          if(!newElement.equals("")) atoms.add(newElement);
          if(st.sval != null) newElement = st.sval;
          else newElement = "";
          parenteses = false;
          parentesesCounter = 0;
        }
        else if(st.sval != null) newElement += st.sval;
        else if(st.ttype == StreamTokenizer.TT_NUMBER){
          newElement += (int)st.nval;
        }
        else if(st.ttype == '('){
          newElement += "(";
          parenteses = true;
          parentesesCounter++;
        }
        else if(st.ttype == ')'){
          newElement += ")";
          parentesesCounter--;
        }
        else if(st.ttype == 92){
```

```
            newElement += "'";
            if(fnutt) fnutt = false;
            else fnutt = true;
          }
          else if(st.ttype == ',') newElement += ",";

          if(st.ttype == StreamTokenizer.TT_EOF) done = true;
          else st.nextToken();
        }
      }
      catch(Exception e){
        System.out.println("Exception trying st.nextToken() in constructor for TqlElement");
        System.out.println(e.toString());
      }

      if(!atoms.isEmpty()){
        firstAtom = (String)atoms.get(0);
        //Check if this is a class-defining TQL-element
        if(firstAtom.startsWith("which(") || firstAtom.equals("test") || firstAtom.equals("new") ||
firstAtom.equals("explain") || firstAtom.equals("do")){
          type = CLASS;
        }
        //Check if this is an 'is' TQL-element
        else if(atoms.size() > 1 && ((String)atoms.get(1)).equals("isa")){
          type = IS;
        }
        //If it's not a class defining element, it has to be a relation element
        else{
          type = RELATION;
        }
      }
    }

    public boolean isClass(){
      if(type == CLASS) return true;
      else return false;
    }

    public boolean isIs(){
      if(type == IS) return true;
      else return false;
    }

    public boolean isRelation(){
      if(type == RELATION) return true;
      else return false;
    }

    public String getWhichContent(){
      String contains = "";
      String whichString = (String)atoms.get(0);
      StreamTokenizer st = new StreamTokenizer(new StringReader(whichString));
      st.ordinaryChar('(');
      st.ordinaryChar(')');
      st.ordinaryChar(',');

      try{
        while(st.ttype != '('){
          st.nextToken();
        }
        int parentesCounter = 1;
        while(parentesCounter > 0){
          st.nextToken();
          if(st.sval != null) contains += st.sval;
          else if(st.ttype == StreamTokenizer.TT_NUMBER) contains += (int)st.nval;
          else if(st.ttype == ',') contains += ",";
          else if(st.ttype == '('){
            contains += "(";
            parentesCounter++;
          }
          else if(st.ttype == ')'){
            if(parentesCounter > 1) contains += ")";
            parentesCounter--;
          }
        }
      }
      catch(Exception e){
```

```
      }
      return contains;
   }

   public String getFirstAtom(){
      return firstAtom;
   }

   public ArrayList getAtoms(){
      return atoms;
   }

   public String getElementString(){
      return elementString;
   }
}
```

TrondheimMap.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.ArrayList;
import java.lang.Thread;


public class TrondheimMap extends JLabel{

   private String[][] mapCoords = {
                       {"Dragvoll", "325", "230"}
                                          };

   ImageIcon it3;
   private Ellipse2D.Double circle;
   private Ellipse2D.Double startDot;
   private Line2D.Double youreHereLine;
   private Line2D.Double goalLine;

   private ArrayList pathList;
   private int goalText_y;
   private boolean locationFound = false;

   public TrondheimMap(String location){
      super(new ImageIcon("trondheim.jpg"));
      int[] locationCoords = new int[2];
      for(int i=0; i<mapCoords.length; i++){
        if( ((String)mapCoords[i][0]).equals(location)){
          locationCoords[0] = Integer.parseInt(mapCoords[i][1]);
          locationCoords[1] = Integer.parseInt(mapCoords[i][2]);
          locationFound = true;
        }
      }
      if(locationFound) setLocationCircle(locationCoords[0]-25,locationCoords[1]-15);
      startDot = new Ellipse2D.Double(173,178,10,10);
      youreHereLine = new Line2D.Double(178,183,515,183);
   }

   //Dummy constructor
   public TrondheimMap(){

   }

   public boolean buildingInTrondheim(String s){
      for(int i=0; i<mapCoords.length; i++){
        if(s.equals(mapCoords[i][0])) return true;
      }
      return false;
   }

   public void paintComponent(Graphics g){
      clear(g);
      Graphics2D g2d = (Graphics2D)g;
      g2d.setColor(Color.red);
      if(locationFound){
        g2d.draw(circle);
        g2d.draw(goalLine);
        g2d.drawString("Du skal hit!", 465, goalText_y);
      }
      g2d.fill(startDot);
      g2d.draw(youreHereLine);
      g2d.drawString("Du er her!", 465, 170);
   }

   public void setLocationCircle(int x, int y){
      circle = new Ellipse2D.Double(x,y,50,30);
      goalLine = new Line2D.Double(x+50, y+15, 465, y+15);
      goalText_y = y+5;
   }

   protected void clear(Graphics g){
      super.paintComponent(g);
```

```
  }

  protected Ellipse2D.Double getCircle(){
    return (circle);
  }
}
```

Appendix B – Robot Controller source code

MarvinController.java

```
import edu.wsu.KepheraSimulator.RobotController;
import edu.wsu.KepheraSimulator.KSGripperStates;

import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.lang.Math;
import java.io.*;


public class MarvinController extends RobotController{

  private Map locationDescriptions;

  private static int WEST = 1;
  private static int NORTH = 2;
  private static int EAST = 3;
  private static int SOUTH = 4;
  private static int LEFT = 1;
  private static int RIGHT = 2;

  private static double ADJUSTMENT_REFINEMENT = 0.19;
  private static int DIFF_TOLERANCE = 150;

  private boolean wall_left;
  private boolean wall_right;
  private boolean wall_front;
  private boolean wall_back;

  private boolean turning;                          //True if Marvin is in a turning mode
  private boolean moving;                           //True if Marvin is in a moving mode
  private boolean verifying;
  private boolean positioning;

  private boolean forceXMovement;
  private boolean forceYMovement;

  private int xDistanceToGo;
  private int yDistanceToGo;
  private int direction;
  private int currentlyTurningTo;

  private ArrayList commandList;

  protected long theTurnDegree = 90;
  protected int state = 0;
  private int verifyState = 0;
  protected boolean slowSpeed = false; // check if we're compensating.
  protected long degreeTarget = 0;
  protected long toMove = 0;
  long startCount = 0;
  private int targetDoorLocation = 0;
  private int targetRoomNumber = 0;
  private int[] targetWalls;
  private int currentRoom = 25;
  private int[] adjustments;

  public MarvinController(){

    initMapDescriptions();
    targetWalls = new int[4];
    adjustments = new int[2];
    commandList = new ArrayList();
    xDistanceToGo = 0;
    yDistanceToGo = 0;
    direction = EAST;
    turning = false;
```

```
    moving = false;
    forceYMovement = false;
    forceXMovement = false;
    state = 0;
    setWaitTime(5);

    getLocationFromFile();
    setRobotReady();
  }


  public void doWork() throws Exception {


    //Marvin isn't in turning or moving mode, but there is some distance to go in either x or y
direction
    if(!(xDistanceToGo==0 && yDistanceToGo==0) && !verifying && !turning && !moving && !
positioning){

      //North
      if(yDistanceToGo < 0 && !forceXMovement){
        if(direction != NORTH){
          turn(NORTH);
        }
        else{
          moving=true;
        }
      }
      //South
      else if(yDistanceToGo > 0 && !forceXMovement){
        if(direction != SOUTH){
          turn(SOUTH);
        }
        else{
          moving=true;
        }
      }
      //Left
      else if(xDistanceToGo < 0 && !forceYMovement){
        if(direction != WEST){
          turn(WEST);
        }
        else{
          moving=true;
        }
      }
      //Right
      else if(xDistanceToGo > 0 && !forceYMovement){
        if(direction != EAST){
          turn(EAST);
        }
        else{
          moving=true;
        }
      }
    }

    //Marvin is in turning mode
    else if(turning){
      if(state == 0) {
        degreeTarget = getLeftWheelPosition() + theTurnDegree*3;
        if(theTurnDegree > 0) currentlyTurningTo = RIGHT;               //NEW
        else if(theTurnDegree < 0) currentlyTurningTo = LEFT;           //NEW
        slowSpeed = false;
        state = 1;
      }

      if(state == 1) {

        // Sjekker om roboten har snudd langt nok.
        if(getLeftWheelPosition() == degreeTarget) {
          setMotorSpeeds(0,0);
          state = 2; // Good! Next step
        }
        else {

          // Checks if the robot has overturned and must adjust
```

```
            if (((getLeftWheelPosition() < degreeTarget) && (theTurnDegree < 0)) ||
               (getLeftWheelPosition() > degreeTarget) && (theTurnDegree > 0)) {

            //if ( ((getLeftWheelPosition() < degreeTarget) && currentlyTurningTo == LEFT) ||
     //NEW
            // ((getLeftWheelPosition() > degreeTarget) && currentlyTurningTo == RIGHT) ){
  //NEW

              long offset = (getLeftWheelPosition() - degreeTarget);

              theTurnDegree = -offset*3;
              degreeTarget = getLeftWheelPosition() - offset;
              slowSpeed = true;
            }

            int speed;
            if(slowSpeed)
              speed = 1;
            else
              speed = 2;

            // keep on turning
            if(theTurnDegree > 0) {
              setMotorSpeeds(speed, -speed); // Right turn
            }
            else {
              setMotorSpeeds(-speed, speed); // Left turn
            }

          }
        }

        if(state == 2) {
          turning = false;
         // slowSpeed = false;
          if(currentlyTurningTo == RIGHT) changeDirectionRight();
          else if(currentlyTurningTo == LEFT) changeDirectionLeft();
          theTurnDegree = 0;
          state = 0;
        }
      }

      //Marvin is in moving mode
      else if(moving){
        if(state == 0) {
          startCount = getLeftWheelPosition();
          if(direction==EAST || direction==WEST) toMove = Math.abs(xDistanceToGo);
          else if(direction== NORTH || direction==SOUTH) toMove = Math.abs(yDistanceToGo);
          else System.out.println("Error trying to move (no direction)");
            state = 1;
        }

        if(state == 1) {
          int[] readings = getAvgReadings();

          //What to do if there is a wall ahead
          if(readings[1] > 1000 || readings[2] > 1000 || readings[3] > 1000 || readings[4] > 1000){
            setMotorSpeeds(0,0);

            //If the current command involves movement in another direction
            if( (direction==NORTH || direction==SOUTH) && xDistanceToGo != 0){
              state = 2;
              forceXMovement = true;
              forceYMovement = false;
            }
            else if( (direction==EAST || direction==WEST) && yDistanceToGo != 0){
              state = 2;
              forceYMovement = true;
              forceXMovement = false;
            }

            //If the current command does not involve movement in another direction, try to avoid
the wall
            else if(readings[1] > readings[4] && direction==NORTH){ xDistanceToGo += 40;
adjustments[0] += 40;}
            else if(readings[4] > readings[1] && direction==NORTH){ xDistanceToGo -= 40;
adjustments[0] -= 40;}
```

```
            else if(readings[1] > readings[4] && direction==SOUTH){ xDistanceToGo -= 40;
adjustments[0] -= 40;}
            else if(readings[4] > readings[1] && direction==SOUTH){ xDistanceToGo += 40;
adjustments[0] += 40;}
            else if(readings[1] > readings[4] && direction==WEST){ yDistanceToGo -= 40;
adjustments[1] -= 40;}
            else if(readings[4] > readings[1] && direction==WEST){ yDistanceToGo += 40;
adjustments[1] += 40;}
            else if(readings[1] > readings[4] && direction==EAST){ yDistanceToGo += 40;
adjustments[1] += 40;}
            else if(readings[4] > readings[1] && direction==EAST){ yDistanceToGo -= 40;
adjustments[1] -= 40;}


          }

          //What to do if there is more distance to go
          else if((getLeftWheelPosition()-startCount) < toMove) {

             //If there's a very short distance to go, go slowly!!
             if(Math.abs( (getLeftWheelPosition()-startCount)-toMove ) < 10){
               setMotorSpeeds(1,1);
             }
             else setMotorSpeeds(5,5);
            }

            //If there is no distance to go and no wall ahead, update distance to go and travel
            else if((getLeftWheelPosition()-startCount) >= toMove) {
                setMotorSpeeds(0,0);
                forceXMovement = false;
           forceYMovement = false;
                state = 2;
            }
       }

       if(state == 2) {
          setMotorSpeeds(0,0);
          if(direction==WEST) xDistanceToGo += (getLeftWheelPosition()-startCount);
          else if(direction==EAST) xDistanceToGo -= (getLeftWheelPosition()-startCount);
          else if(direction==NORTH) yDistanceToGo += (getLeftWheelPosition()-startCount);
          else if(direction==SOUTH) yDistanceToGo -= (getLeftWheelPosition()-startCount);
            updateTravel((int)(getLeftWheelPosition()-startCount));
          moving = false;
          state = 0;
System.out.println("Move state finished, xDistanceToGo="+xDistanceToGo+",
yDistanceToGo="+yDistanceToGo);
          }
     }

     //Marvin is in verify location mode
     else if(verifying){
       if(verifyState == 1){
          turn(targetDoorLocation);
          verifyState = 2;
       }
       else if(verifyState == 2){
          findWalls();
          if(wallsAreCorrect()){
            System.out.println("Verified location: Room "+(300+targetRoomNumber));
            if(currentRoom != targetRoomNumber) setMsg("Verified location:
"+(String)locationDescriptions.get(new Integer(targetRoomNumber)));
            currentRoom = targetRoomNumber;
            setMarvinLocation(currentRoom);

            //Start positioning at the door
            positioning = true;
          }
          else{

            System.out.println("Couldn't verify location, assuming to be near the goal
"+targetRoomNumber+". Searching for the door.");
            setMsg("Couldn't verify location, assuming to be near
"+(String)locationDescriptions.get(new Integer(targetRoomNumber))+". Searching for the
location...");
            currentRoom = targetRoomNumber;
            setMarvinLocation(currentRoom);
```

```
            //Start positioning, attempting to find the door
            positioning = true;


            /*
            System.out.println("Couldn't verify location. The robot doesn't know where it is and
will stop.");
            while(!commandList.isEmpty()){
              commandList.remove(0);
            }
            */
          }
          targetRoomNumber = 0;
          targetDoorLocation = 0;
          verifyState = 0;
          verifying = false;
        }
      }

      //Marvin is in "position yourself at the location" mode
      else if(positioning){
        int[] readings = getAvgReadings();

//Just for debugging
for(int i=0; i<8; i++){
System.out.print(readings[i]+" ");
}
System.out.println("Direction: "+direction+"\n");

        //If there is a wall on either side, reduce corner sensor's readings for that side
        //(because it will read more than if there was no wall there, even if the robot is in the
middle of the door opening
        if(readings[0] > 800 && readings[1] > 200) readings[1] -= 200;
        if(readings[5] > 800 && readings[4] > 200) readings[4] -= 200;

        //If the difference between corner sensors is too big, adjust position
        if(Math.abs(readings[1]-readings[4]) > DIFF_TOLERANCE){
          if(readings[1] > readings[4]) addAdjustCommand(1, Math.abs(readings[1]-readings[4]));
//Right
          else if(readings[4] > readings[1]) addAdjustCommand(0, Math.abs(readings[1]-readings[4]));
//Left
        }

        //If the difference between corner sensors are ok, but the values are to small, go a bit
forward (towards the door hopefully)
        else if(Math.abs(readings[1]-readings[4]) < DIFF_TOLERANCE && (readings[1]+readings[4] <
DIFF_TOLERANCE)){
          addAdjustCommand(2, 40); //Forward
        }
        positioning = false;
      }

      //Marvin has no distance to go and isn't moving or turning. Verifying that the location is
correct
      else if(xDistanceToGo == 0 && yDistanceToGo == 0 && targetRoomNumber != 0 && !turning && !
moving && !verifying && !positioning){

        setMotorSpeeds(0,0);
        verifyState = 1;
        verifying = true;
      }

      //Get next command or read a command list from file
      else{

        setMotorSpeeds(0,0);
        if(commandList.isEmpty()){

          //Wait for 1 second, so that we aren't reading the path file all the time
          sleep(1000);
          String result ="";
          try{
            //Read to buffer
            FileReader fr = new FileReader("./controllers/path");
            File infile = new File("./controllers/path");
            char[] cbuf = new char[(int)infile.length()];
            fr.read(cbuf);
```

```
            fr.close();

            //Buffer to string
            for(int i=0; i<cbuf.length; i++){
              result += cbuf[i];
            }

            //Clear path file
            FileWriter fw = new FileWriter("./controllers/path");
            fw.write(".");
            fw.close();
          }
          catch(Exception e){
            System.out.println("Error dealing with path file in MarvinController");
            System.out.println(e.toString());
          }
          parseCommandString(result);
        }
        else if(!commandList.isEmpty()){
          getNextCommand();
        }
      }
    }

    public void executeCommands(ArrayList commands){
      commandList = commands;
    }


    public void close() throws Exception {



    }

  //-----------------------------Private methods-------------------------------------------

    private void addAdjustCommand(int dir, int diff){  //2 for forward, 1 for right, 0 for left
      getLocationFromFile();
      int[] adjustment = new int[2];
      adjustment[0] = 0; adjustment[1] = 0;

      if(dir == 0){
        if(direction == WEST) adjustment[1] = (int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == NORTH) adjustment[0] = -(int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == EAST) adjustment[1] = -(int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == SOUTH) adjustment[0] = (int)(ADJUSTMENT_REFINEMENT*diff);
      }
      else if(dir == 1){
        if(direction == WEST) adjustment[1] = -(int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == NORTH) adjustment[0] = (int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == EAST) adjustment[1] = (int)(ADJUSTMENT_REFINEMENT*diff);
        else if(direction == SOUTH) adjustment[0] = -(int)(ADJUSTMENT_REFINEMENT*diff);
      }
      else if(dir == 2){
        if(direction == WEST) adjustment[0] = -diff;
        else if(direction == NORTH) adjustment[1] = -diff;
        else if(direction == EAST) adjustment[0] = diff;
        else if(direction == SOUTH) adjustment[1] = diff;
      }

      int[] blah = new int[9];
      blah[0] = adjustment[0];
      blah[1] = adjustment[1];
      blah[2] = 0;
      blah[3] = currentRoom;
      blah[4] = currentRoom;

      for(int i=0;i<4;i++){
        blah[5+i] = targetWalls[i];
      }

      commandList.add(commandList.size(),blah);
      blah = new int[9];
    }

    private void parseCommandString(String s){
```

```
    if(s.equals(".")) System.out.println("No new commands");
    else{
      StreamTokenizer st = new StreamTokenizer(new StringReader(s));
      st.ordinaryChar('{');
      st.ordinaryChar('}');
      st.ordinaryChar(',');
      st.ordinaryChar('.');
      try{
        int tokenType = st.nextToken();
        int[] blah = new int[9];
        while(tokenType != StreamTokenizer.TT_EOF){
          if(st.ttype == StreamTokenizer.TT_WORD && st.sval.equals("From")){
            st.nextToken();
            st.nextToken();
            blah[3]=(int)st.nval;
          }
          else if(st.ttype == StreamTokenizer.TT_WORD && st.sval.equals("To")){
            st.nextToken();
            st.nextToken();
            blah[4]=(int)st.nval;
          }
          else if(st.ttype=='{'){
            st.nextToken();
            blah[0] = (int)st.nval;
            st.nextToken();
            st.nextToken();
            blah[1] = (int)st.nval;
            st.nextToken();
            st.nextToken();
            blah[5] = (int)st.nval;
            st.nextToken();
            st.nextToken();
            blah[6] = (int)st.nval;
            st.nextToken();
            st.nextToken();
            blah[7] = (int)st.nval;
            st.nextToken();
            st.nextToken();
            blah[8] = (int)st.nval;

            commandList.add(0, blah);
            blah = new int[9];
          }
          tokenType = st.nextToken();
        }
      }
      catch(Exception e){
        System.out.println("Error parsing path file in MarvinController");
        e.printStackTrace(System.out);
      }

      if(!commandList.isEmpty()){
        int[] tempStart = (int[])commandList.get(commandList.size()-1);
        int[] tempGoal = (int[])commandList.get(0);
        setMsg("New task: Go from "+(String)locationDescriptions.get(new Integer(tempStart[3]))+"
to "+(String)locationDescriptions.get(new Integer(tempGoal[4]))+".");
      }
    }
  }

  private void getNextCommand(){
    int[] blah = (int[])commandList.remove(commandList.size()-1);
    getLocationFromFile();

    if(blah[3] == currentRoom){

      //Setting the X and Y distances to go that are given in the command
      //Also trying to compensate for previous adjustments that was made due to blocking objects
      xDistanceToGo = blah[0] + adjustments[0];
      adjustments[0] = 0;
      yDistanceToGo = blah[1] + adjustments[1];
      adjustments[1] = 0;
      //targetDoorLocation = blah[2];
      targetRoomNumber = blah[4];

      System.out.println("New command! Distances to go: "+xDistanceToGo+", "+yDistanceToGo+"\n"+
                         "Going from room "+blah[3]+" to "+targetRoomNumber+". Currently at
```

```
room "+currentRoom);
       //setMsg("New command!"+(String)locationDescriptions.get(new Integer(25)));
       for(int i=0;i<4;i++){
         targetWalls[i] = blah[5+i];
         if(targetWalls[i]==2){
           if(i==0) targetDoorLocation = WEST;
           else if(i==1) targetDoorLocation = NORTH;
           else if(i==2) targetDoorLocation = EAST;
           else if(i==3) targetDoorLocation = SOUTH;
         }
       }
     }
     else{
       System.out.println("New command found, but start location ("+blah[3]+") didn't match the
robot's location "+(currentRoom)+". Command cancelled.");
     }
   }

   private void findWalls(){

     int[] getDistanceValue = getAvgReadings();

     //Assume there is a wall on the left if the left sensor gives a reading bigger than 300
     if(getDistanceValue[0] > 250) wall_left = true;
     else wall_left = false;

     //Assume there is a wall in front, if the front sensors have a combined reading of more than
600, and each have a reading of more than 300
     if( (getDistanceValue[2] + getDistanceValue[3] > 500)
       &&
       (getDistanceValue[2] > 250)
       &&
       (getDistanceValue[3] > 250)
       ){
         wall_front = true;
     }
     else wall_front = false;

     //Assume there is a wall on the right if the right sensor gives a reading bigger than 300
     if(getDistanceValue[5] > 250) wall_right = true;
     else wall_right = false;

     if(getDistanceValue[6] + getDistanceValue[7] > 600) wall_back = true;
     else wall_back = false;
   }

   private boolean wallsAreCorrect(){

     boolean expectedLeft = false;
     boolean expectedRight = false;
     boolean expectedBack = false;

     if(direction==WEST){
       if(targetWalls[3]==1) expectedLeft = true;
       if(targetWalls[1]==1) expectedRight = true;
       if(targetWalls[2]==1) expectedBack = true;
     }
     else if(direction==NORTH){
       if(targetWalls[0]==1) expectedLeft = true;
       if(targetWalls[2]==1) expectedRight = true;
       if(targetWalls[3]==1) expectedBack = true;
     }
     else if(direction==EAST){
       if(targetWalls[1]==1) expectedLeft = true;
       if(targetWalls[3]==1) expectedRight = true;
       if(targetWalls[0]==1) expectedBack = true;
     }
     else if(direction==SOUTH){
       if(targetWalls[2]==1) expectedLeft = true;
       if(targetWalls[0]==1) expectedRight = true;
       if(targetWalls[1]==1) expectedBack = true;
     }

     if(
       (wall_left==expectedLeft)
       &&
       (wall_right==expectedRight)
```

```
     &&
     (wall_back==expectedBack)
     &&
     (!wall_front)

     )
      {

     return true;
   }
   else{
     return false;
   }
}

private int[] getAvgReadings(){
   int[] returnValues = new int[8];
   for(int i=0;i<8;i++){
     int summer=0;
     for(int j=0;j<10;j++){
       summer += getDistanceValue(i);
     }
     returnValues[i] = summer/10;
   }
   return returnValues;
}

private void turn(int a){
   if(a == direction){
     turning=false; theTurnDegree = 0;
   }
   else{
     if(a == WEST && direction == SOUTH) theTurnDegree = 90;
     else if(a == WEST) theTurnDegree = -90;
     else if(a == NORTH && direction == WEST) theTurnDegree = 90;
     else if(a == NORTH) theTurnDegree = -90;
     else if(a == EAST && direction == NORTH) theTurnDegree = 90;
     else if(a == EAST) theTurnDegree = -90;
     else if(a == SOUTH && direction == EAST) theTurnDegree = 90;
     else if(a == SOUTH) theTurnDegree = -90;

//   if(a != direction) turning = true;
     turning = true;
   }
}

private void changeDirectionRight(){
   if(direction==WEST) direction=NORTH;
   else if(direction==NORTH) direction=EAST;
   else if(direction==EAST) direction=SOUTH;
   else if(direction==SOUTH) direction=WEST;
}

private void changeDirectionLeft(){
     if(direction==WEST) direction=SOUTH;
     else if(direction==NORTH) direction=WEST;
     else if(direction==EAST) direction=NORTH;
     else if(direction==SOUTH) direction=EAST;
}

private void updateTravel(int t){
   try{

     String marvinFile = "";
     FileReader fr = new FileReader("./controllers/marvin");
     File infile = new File("./controllers/marvin");
     char[] cbuf = new char[(int)infile.length()];
     fr.read(cbuf);
     fr.close();
     //Buffer to string
     for(int i=0; i<cbuf.length; i++){
       marvinFile += cbuf[i];
     }
     StringBuffer sb = new StringBuffer(marvinFile);
     int writePosition = 0;

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

```
        if(i+17 > sb.length()){
          System.out.println("Didn't find 'DistanceTravelled' in file marvin");
          i=sb.length();
        }
        else if(sb.substring(i, i+17).equals("DistanceTravelled")){
          writePosition = i+18;
          i = sb.length();
        }
      }

      if(writePosition != 0){
        int readPos = writePosition;
        StringBuffer oldNumber = new StringBuffer();
        while(Character.isDigit(sb.charAt(readPos))){
          oldNumber.append(sb.charAt(readPos));
          readPos++;
        }
        String oldNumberString = oldNumber.toString();
        int oldNumberInt = Integer.parseInt(oldNumberString);
        int newNumber = oldNumberInt+t;
        String newNumberString = ""+newNumber+"\n";
        sb = sb.replace(writePosition, writePosition+newNumberString.length(), newNumberString);
        FileWriter fw = new FileWriter("./controllers/marvin");
        fw.write(sb.toString());
        fw.close();
      }



    }
    catch(Exception e){
      System.out.println("Error writing to file marvin");
      e.printStackTrace(System.out);
    }
  }

  private void setRobotReady(){
    try{
      String toWrite ="yes";

      String marvinFile = "";
      FileReader fr = new FileReader("./controllers/marvin");
      File infile = new File("./controllers/marvin");
      char[] cbuf = new char[(int)infile.length()];
      fr.read(cbuf);
      fr.close();
      //Buffer to string
      for(int i=0; i<cbuf.length; i++){
        marvinFile += cbuf[i];
      }
      StringBuffer sb = new StringBuffer(marvinFile);
      int writePosition = 0;

      for(int i=0; i<sb.length(); i++){
        if(i+8 > sb.length()){
          System.out.println("Didn't find 'SimulatorReady' in file marvin");
          i=sb.length();
        }
        else if(sb.substring(i, i+14).equals("SimulatorReady")){
          writePosition = i+15;
          i = sb.length();
        }
      }

      if(writePosition != 0){
        sb = sb.replace(writePosition, writePosition+3, toWrite);
        FileWriter fw = new FileWriter("./controllers/marvin");
        fw.write(sb.toString());
        fw.close();
      }
    }
    catch(Exception e){
      System.out.println("Error writing to file marvin");
      e.printStackTrace(System.out);
    }
  }
```

```
  private void setMarvinLocation(int location){
       try{
         String toWrite ="";
         if(location < 10) toWrite = "  "+location;
         else if(location <100) toWrite = " "+location;
         else toWrite = ""+location;

         String marvinFile = "";
         FileReader fr = new FileReader("./controllers/marvin");
         File infile = new File("./controllers/marvin");
         char[] cbuf = new char[(int)infile.length()];
         fr.read(cbuf);
         fr.close();
         //Buffer to string
         for(int i=0; i<cbuf.length; i++){
           marvinFile += cbuf[i];
         }
         StringBuffer sb = new StringBuffer(marvinFile);
         int writePosition = 0;

         for(int i=0; i<sb.length(); i++){
           if(i+8 > sb.length()){
             System.out.println("Didn't find 'Location' in file marvin");
             i=sb.length();
           }
           else if(sb.substring(i, i+8).equals("Location")){
             writePosition = i+10;
             i = sb.length();
           }
         }

         if(writePosition != 0){
           sb = sb.replace(writePosition, writePosition+3, toWrite);
           FileWriter fw = new FileWriter("./controllers/marvin");
           fw.write(sb.toString());
           fw.close();
         }



       }
       catch(Exception e){
         System.out.println("Error writing to file marvin");
         e.printStackTrace(System.out);
       }

  }

  private void getLocationFromFile(){
     int locat = 0;
         try{
           FileReader fr = new FileReader("./controllers/marvin");
           File infile = new File("./controllers/marvin");
           char[] cbuf = new char[(int)infile.length()];
           fr.read(cbuf);
           fr.close();

           String marvinFile = "";
           for(int i=0; i<cbuf.length; i++){
             marvinFile += cbuf[i];
           }

           StreamTokenizer st = new StreamTokenizer(new StringReader(marvinFile));
           st.lowerCaseMode(true);
           int tokenType = st.nextToken();
           while(st.ttype != StreamTokenizer.TT_EOF && (st.sval == null || !
((st.sval).equals("location")))){
              st.nextToken();
           }
           if(tokenType == StreamTokenizer.TT_EOF) locat = 0;
           else if(tokenType == StreamTokenizer.TT_WORD && st.sval.equals("location")){
             st.nextToken();
             tokenType = st.nextToken();
             if (tokenType == StreamTokenizer.TT_NUMBER) locat = (int)st.nval;
           }
           else locat = 0;
         }
```

```
         catch(Exception e){
            System.out.println("Error trying to read Marvin's location from file marvin");
            e.printStackTrace(System.out);
         }

      currentRoom = locat;
   }

   private void initMapDescriptions(){
      locationDescriptions = new HashMap();
      locationDescriptions.put(new Integer(25), "Entrance");
      locationDescriptions.put(new Integer(22), "Door 322");
      locationDescriptions.put(new Integer(21), "Door 321");
      locationDescriptions.put(new Integer(20), "Door 320");
      locationDescriptions.put(new Integer(63), "Door 363");
      locationDescriptions.put(new Integer(26), "Door 326");
      locationDescriptions.put(new Integer(24), "Door 324");
      locationDescriptions.put(new Integer(17), "Door 317");
      locationDescriptions.put(new Integer(16), "Door 316");
      locationDescriptions.put(new Integer(15), "Door 315");
      locationDescriptions.put(new Integer(14), "Door 314");
      locationDescriptions.put(new Integer(13), "Door 313");
      locationDescriptions.put(new Integer(12), "Door 312");
      locationDescriptions.put(new Integer(11), "Door 311");
      locationDescriptions.put(new Integer(10), "Door 310");
      locationDescriptions.put(new Integer(8), "Door 308");
      locationDescriptions.put(new Integer(7), "Door 307");
      locationDescriptions.put(new Integer(6), "Door 306");
      locationDescriptions.put(new Integer(5), "Door 305");
      locationDescriptions.put(new Integer(4), "Door 304");
      locationDescriptions.put(new Integer(3), "Door 303");
      locationDescriptions.put(new Integer(2), "Door 302");
      locationDescriptions.put(new Integer(1), "Door 301");
      locationDescriptions.put(new Integer(42), "Door 342");
      locationDescriptions.put(new Integer(431), "Door 343b");
      locationDescriptions.put(new Integer(43), "Door 343");
      locationDescriptions.put(new Integer(44), "Door 344");
      locationDescriptions.put(new Integer(461), "Door 346b");
      locationDescriptions.put(new Integer(45), "Door 345");
      locationDescriptions.put(new Integer(46), "Door 346");
      locationDescriptions.put(new Integer(47), "Door 347");
      locationDescriptions.put(new Integer(90), "Elevator");
      locationDescriptions.put(new Integer(30), "Door 330 (stairs)");
      locationDescriptions.put(new Integer(49), "Door 349");
      locationDescriptions.put(new Integer(48), "Door 348");
      locationDescriptions.put(new Integer(31), "Door 331");
      locationDescriptions.put(new Integer(32), "Door 332");
      locationDescriptions.put(new Integer(33), "Door 333");
      locationDescriptions.put(new Integer(34), "Door 334");
      locationDescriptions.put(new Integer(35), "Door 335");
      locationDescriptions.put(new Integer(36), "Door 336");
      locationDescriptions.put(new Integer(37), "Door 337");
      locationDescriptions.put(new Integer(54), "Door 354");
      locationDescriptions.put(new Integer(56), "Door 356");
      locationDescriptions.put(new Integer(58), "Door 358");
      locationDescriptions.put(new Integer(601), "Door 360b");
      locationDescriptions.put(new Integer(59), "Door 359");
      locationDescriptions.put(new Integer(60), "Door 360");
      locationDescriptions.put(new Integer(61), "Door 361");
      locationDescriptions.put(new Integer(63), "Door 363");
      locationDescriptions.put(new Integer(50), "Toilets");
   }
}
```