A Qualitative Case Study on the Use of Jeliot 3

Kimmo Sivula

26.8.2005

University of Joensuu
Department of Computer Science
Master´s thesis

Abstract

This thesis is the review of the development of the tool to help students to learn the basics about object-oriented programming. After that tool was finished, this research was made and this thesis was written. That tool is called Jeliot 3. In this thesis is researched how students use Jeliot 3 in the process of learning the basics about object-orinted programming, whether Jeliot 3 is a useful tool in learning, what kind of exercises are good for Jeliot 3 and does Jeliot 3 has some kind of effect on the students' motivation. Participants were involved in the ViSCoS project where high school students can study their first-year university-level computer science studies in a virtual way. In this research two students were observed while they were doing their homework as a pair. The homework belongs to the course called Programming II, where they were studying the basics about object-oriented programming. Students came once every week to the special laboratory in the building of the department of computer science. The data was collected four weeks using the program called Camtasia, which saves the view from the screen ten times in a second. Camtasia also saves sound which makes possible to watch the saved files as movies. In this thesis was used the content analysis. According to my findings the Jeliot 3 helps students to learn the basics about object-oriented programming, if they know how it can be used effectively. Jeliot 3 has also positive influence on students' motivation. Jeliot 3 has some disadvantages, like using colours in highlighting the source code, but when those are fixed Jeliot 3 can be used even more effectively.

Keywords: Jeliot, object-oriented programming, computer science education

Table of contents

1 Introduction

The Finnish writer Mika Waltari has said: *"When you are a novice you better write only about themes which touch your life, your line of business, peoples who live is the same building as you do, your friends, workers who work in the same factory as you do, work day of the same office. Then you have a chance to get into your text that `strong feeling from the life that has been lived`, without that the text you have written remains ineffective, even if this phrase has worn out as a mistake of style."*[1] The same is true in research work (Eskola & Suoranta 1999). Because I am a novice as a researcher, I decided to do my Master's thesis about something that I have studied and I am interested in. I also hope that I can be connected with this subject in the future.

I wanted to do my thesis on a topic that is related to teaching because I have studied in a computer science education program. Almost every course in the Department of Computer Science has something to do with programming. It means that programming is one of the major topics in computer science and every computer science student has to have some kind of knowledge about programming. That is the reason why teaching and learning programming are important issues in the university level computer science curriculum. Almost every time when the word *learning* is mentioned, the phrase *problems in learning* are mentioned very close after. That is the reason why teachers, including me, are interested in things that make learning easier and decrease learning problems.

[1] "Aloittelijan asteella sinun on paras kirjoittaa vain sellaisista aiheista, jotka sivuavat omaa elämääsi, toimialaasi, saman talon asukkaita, tovereitasi, saman tehtaan työläisiä, saman konttorin arkipäivää. Silloin sinulla on mahdollisuus saada tekstiisi se, `eletyn elämän väkevä tuntu`, jonka puuttuessa kirjoittamasi jää tehottomaksi, vaikka lausepari onkin kulunut tyylivirheeksi."

When I started my university studies, I did not have any knowledge of programming; I also had some problems while learning. That is one reason why I am so interested in things that can help students learn programming. In the field of computer science, the factors that support students' learning of algorithms and programming are one notable target of research (Kashihara et al. 1999).

One of these things that help learning is called *visualization*. Visualization, in the context of computer science education, refers to the animation of algorithms. One program that animates algorithms is called Jeliot 3. Jeliot 3 belongs to the Jeliot family, which consists of Eliot, Jeliot I, Jeliot 2000 and Jeliot 3. All of the products from the Jeliot family have been made for animating algorithms or programs, but Jeliot 3 is the first version of Jeliot that is made for object-oriented programming. Because Jeliot 3 is the first one in animating object-oriented programs, it needs to be evaluated and that is the reason why this research has been conducted.

Observing students while they do their homework is the modus operandi of this research. The students in this research were participants in the ViSCoS project, a project where high school students can study first year, university-level computer science (Haataja et al. 2001). The students were observed in a computer science laboratory while doing homework assigned in a course called *Programming II*. In Programming II, the students learn the basics of object-oriented programming.

The aim of this thesis is to explain (a) how *novice programmers* use Jeliot when they learn object-oriented programming. In this thesis, the term *novice programmers* refer to students who are learning the basics of object-oriented programming. The other things which are investigated in this thesis are (b) how Jeliot can be helpful in teaching object-oriented programming, (c) in which type of object-oriented programming exercises Jeliot is good for and (d) whether Jeliot

has an influence on students' motivation. All these points are connected to the topic of learning object-oriented programming with Jeliot.

The next two chapters are about the theory of this thesis. In Chapter 2, some aspects of visualization are introduced. There is information about why visualization is used, what visualization means and how it is used; however, the emphasis is on visualization's use in teaching, its possibilities and limitations, and on strategies for effective visualization. The history of the Jeliot family is recounted in Chapter 3. Chapter 3 describes why each version has been made, how it works and briefly how it has been implemented.

The last three chapters consist of information about the research that has been made. Information about methods, participants, the collection and analysis of data and reliability are given in Chapter 4. In Chapter 5 the results from the research are presented. A discussion of the results is presented in Chapter 6.

2 Visualization

In this chapter, explanations of why visualization is needed, what visualization means and how it used in computer science education are given. In this chapter, the benefits of visualization and also the criticism against it are given. In the end of this chapter, there is a discussion of how effective visualizations should be implemented and how visualizations should be used so that they are as useful as possible.

2.1 The need for visualization

While students learn the basics about programming, they experience several different kinds of problems (Ben-Bassat Levy et al. 2003). One reason they experience problems is that novices' understanding of how computers work is not clear enough (Ben-Ari 2001). Another one that novice students have difficulty understanding how a program works from the source code because they have difficulty in making mind maps (Ben-Bassat Levy et al. 2003). While students learn the basics of programming, they have to learn many different terms; in object-oriented programming there are even more new terms. Learning new terms has been found to be confusing for novice students (Ben-Ari et al. 2002b). Novices also have defective skills in basic tasks such as following the execution of the program (Ben-Ari 2001). Novice users also have problems with planning and implementing an algorithm (Hong 1998).

The first thing in the process of programming is to invent an algorithm. The second step is to implement the algorithm. During that process the user is in the process of problem solving through modeling (Crapo et al. 2000). Modeling means making a model of how an abstract matter works. According to Crapo et

al. (2000) modeling is a process whose aim is to attain a solution and create a correct mental model. Using visualization can facilitate novice users in modeling a problem (Waisel et al. 1999).

2.2 The definition of visualization

Visualization is a computer-based, interactive visual representation of information to approve the cognition; the aim of visualization is to achieve insights (Card et al. 1999). In visualization, cognition has to be understood as the process of procuring or adapting the information. One subset of visualization is software visualization. Software visualization refers to all the fields of visualizations that helps or teaches the process of software engineering (Price et al. 1993). The aim of visualization is to find simplicity from a complex totality (Petre et al. 1998).

Both *visualization* and *software visualization* are quite large-minded. Software visualization has two main parts - program visualization and algorithm visualization (Price et al. 1993). Program visualization shows program structures and actions in a visual way and algorithm visualization shows the common principles how the program works (Korhonen 2003). This thesis focuses on program visualization.

2.3 Visualization in use

Algorithm visualization was first used in the 1950s (Haibt 1959). In the beginning of this century visualization is utilized for explaining, planning and analyzing programs (Khuri 2001). In teaching, visualization has been actively used over the

past twenty years to make it easier to learn the more difficult aspects of programming (Miraftabi 2001).

In teaching, visualization can be used in many different ways: lecturers can easily explain some aspects of an algorithm, students can construct their own visualization of an algorithm as a homework, students can present visualizations in the classroom, which might lead into a discussion, students can explore algorithms in the laboratory, students' can study for a test at any time, during office hours the lecturers can answer students questions and help pose test questions (Hundhausen et al. 2002). In order for visualization to be used effectively in teaching, there has to be a cognitive connection between the search for and use of information. Using visualization can be preasented the fundamental features from the information (Petre et al. 1998).

Effective visualization helps students develop their understandings of algorithms in active learning environments (Hundhausen et al. 2002). Such modern teaching methods, like active learning environments, are incorporated closely to *constructivism,* which is the most popular theory of teaching and learning, including computer science teaching, at the moment (Ben-Ari 2001). Constructivism means that students combine new experiences with old information and produce new theories; it is learning by doing (Ben-Ari 2001). Constructivism is also the basis of the curriculum for both elementary and high school (The basis of the curriculum for elementary school 2004; The basis of the curriculum for high school 2003). Visualization encourages students to practical learning process so that they are not afraid of making mistakes while learning by doing (Khuri 2001). The same observation has been made by Kähkönen & Piiroinen (2002); they observed that students can learn by attempting and making mistakes. Visualization is a great tool for learning when it attracts student in to learning algorithms, but it should not be used alone. It should be used along other teaching models (Hundhausen et al. 2002).

2.4 The problems of using visualization in teaching

Many different ways to use visualization in teaching were introduced above. Nevertheless, some teachers do not use it because they feel that they do not have the time to learn about it, they feel that using it would take away time needed for other class activities, they feel that creating visualizations for classroom use requires too much time and effort and they feel that it is simply not educationally effective (Hundhausen et al. 2002). It has been demonstrated that in certain circumstances there is no advantage at all in using visualization in teaching (Petre & Green 1993). Some problems for using visualization can also be the technical recourses that are involved and the teachers' unwillingness to create visualizations from course materials (Miraftabi 2001).

2.5 The advantages of visualization

The advantages of using visualization in learning have been proven (Lawrence et al. 1994). Computer-based visualization is an effective support for modeling (Crapo et al. 2000). Meisalo et al. (1997) made the same conclusion. Visualization leads students' attention and helps to understand the connection between concepts and text (Mayer 1997). While students watch the visualization they understand more easily how an algorithm works and why it works because they have to think and they can see how algorithm works with different kinds of inputs (Anderson & Naps 2001). Visualization maintains students' power of concentration, clarifies complicated concepts, automates examples and presentations and increases communication between teachers and students (Khuri 2001).

According to Kähkönen & Piiroinen (2002) visualization clarifies and makes programming more concrete, makes programming more fun and easier, is a

great tool to familiarize students with programming, is experienced by students as a meaningful and useful way to learn programming and facilitates the process of starting to learn programming. Visualization should be used while teaching the things that are experienced to be difficult (Torvinen 2001). Kehoe et al. (1999) found out that visualization makes students more motivated and pleased to learn programming. Also Sutinen et al. (1997) have discovered that students are more motivated when they use visualization. While students use visualization the quality of the source code and the documentation that they have written are better (Markkanen et al. 1998). Student development is more effective and the vocabulary of the terms of programming is composed faster, which improves understanding, when they use visualization (Ben-Bassat Levy et al. 2003).

2.6 The disadvantages of visualization

The criticism against the visualization concerns the effectiveness of animation. Petre found out that novice students have insufficient skills to understand graphical displays and they might not see all of the information (1995). Graphical displays are not self-explanatory; instead students have to be taught to understand them (Petre & Green 1993). According to Hundhausen et al. (2002) in visualization a picture is not worth of thousand words. Novice students have also difficulties to understand what each graphical element images in the algorithm (Stasko et al. 1993). Visualization itself does not transfer the correct mental model of an algorithm to the student's brain (Hundhausen et al. 2002).

Visualization is not worth of using if the use and the meaning and the limits of the visualization are not realized (Petre et al. 1998). The good-quality visualization helps to learn but more important things are the form of activity and the form of learning exercises in which the visualization is used (Hundhausen et al. 2002). The weakest students feel that visualization is too confusing. The best students

feel that they do not need it (Ben-Bassat Levy et al. 2003). Petre et al. (1998) disagree by saying that professional users use visualization to converse with equals and as some kind of tool of thinking whereas novice users use visualization to see how algorithms work, for a new way of imaging the problem and to to get a glimpse of professionals' world of ideas. Experts and novices think differently and, because of that, they have different kinds of modeling needs (Crapo et al. 2000). That is the reason why visualization should offer different kinds of views for users at different levels (Petre 1995). An expert user demands from visualization a certain kind of depth of information and the possibility to influence on the function of the visualization (Khuri 2001).

2.7 Effective visualization

In order for visualization to be as effective as possible, the first step in designing visualizations should be a comprehensive user analysis (Khuri 2001). Qualities that novice users would like to have in visualizations are that the user interface should be simple to use and visualization should be continuous so that all of the source code is animated in a way that users know where all the constants and variables come from (Lattu et al. 2000). There should be the possibility to put some syntax highlighting in the code, like indentations and colored text or other text style effects, to put some remarks into the code, to control the visualization, all the connections should be visualized, for example between classes or between objects, and there should be ready-made examples (Lattu et al. 2003). Also Khuri (2001) has noticed that there should be ready-made examples so that students can see how programs work and how outputs are conducted from the inputs.

Visualization should not be used only in limited examples; rather, it should be used throughout the entire course so that it is one basic tool in classrooms and in

exercises, because during long-time use students can learn how to use the new tool (Ben-Bassat Levy et al. 2003). Students need verbal explanations while they use visualization (Kehoe et al. 1999). Mayer (1997) has concluded the same but specifies that explanations have to correspond to the visualizations to be effective. Through the help of explanations, students can understand what really happens in the visualizations and students can more easily relate the source code to the actions in the visualization (Ben-Bassat Levy et al. 2003). In order for visualization to be effective in object-oriented programming, the visualization should animate the connection between execution and the source code, animate the whole code and should be in the same window (Ben-Ari et al. 2002b).

3 The history of Jeliot

Jeliot 3 is the result of more than ten years of searching, programming and testing. It still has its own bugs and users have some ideas how it could be even better, but it is better than earlier versions of Jeliot. Jeliot 3 is the first version of Jeliot that can be used in object-oriented programming (Moreno et al. 2004).

In this chapter, a description of the earlier versions of Jeliot 3 and how they work is given. Also in this chapter is information about what the aim of each version was. This chapter, however, does not go into detail about how different versions are programmed; that information is described in the Master's thesis of Niko Myller (2004). The pedagogical information about Jeliot as well as the positive and the negative sides of visualization are explained in Chapter 2.

3.1 Eliot

The development of Eliot started in 1992 when Erkki Sutinen and Jorma Tarhio found out that students learn more when they make animations themselves instead of just watching ready-made animations. Stasko et al. (1993) has made the same conclusion. At that time both Sutinen and Tarhio were working at the University of Helsinki. With the tools which were available at that time, it took up to hundred hours to make a simple animation. That is why they decided to make a new tool for animating source code. (Sutinen et al. 1997)

The main idea behind this new tool was to make it easier to make animations. From the users' point of view, the easiest way of making animation is *self-animation*. Self-animation means that a user does not have to create any extra

code to make the animation; the tool makes the animation by itself. While the program runs, the tool makes the animation automatically.

After a few years of programming the tool called Eliot, which is the first version of Jeliot was born. It was completed in 1996 in the University of Helsinki by several students under the supervision of Erkki Sutinen and Jorma Tarhio (Ben-Ari et al. 2002a). The name Eliot comes from the Finnish word *eliöt*, which means living organisms (Lahtinen et al. 1998).

Eliot was made using the programming language C and it works only on Linux workstations or on X terminals (Sutinen et al. 1997). The algorithms that can be animated with Eliot have to be written into Eliot C language (Teräsvirta 1996). The Eliot C language is based onto basic C language, but it has a few differences (Teräsvirta 1996).

Animation in Eliot is *semi-automatic*. In this case, semi-automatic means that users have to make decisions about which data objects are visualized and how. Eliot offers a library of data types. In the Eliot library there are data types such as integer and tree. Users choose which data objects they want to be animated but the data object has to be in the data-type library. Eliot has also different kinds of visualizations for each data type. Users also have to choose one visualization for each data object. Users can choose different types of visualizations for the same data object, so the same data object can be animated differently. After making these choices, Eliot makes an animation automatically. (Sutinen et al. 1997)

Figure 1 shows the user interface of Eliot. There are only the source code window and the animation setup window. From the animation setup window, the user can choose how the data object is animated. Figure 2 shows the two different animations for an array which a user has chosen. It also shows the animation control panel. The control panel includes *run, step* and *quit* buttons and the slidebar for controlling the speed of the animation. The run button makes

the whole program run, the step button makes the program run step-by-step. As can be seen in Figure 1 and Figure 2, the user interface consists of many windows. According to Ben-Ari et al. (2002a) that can confuse novice users instead of helping them learning. (Sutinen et al. 1997)

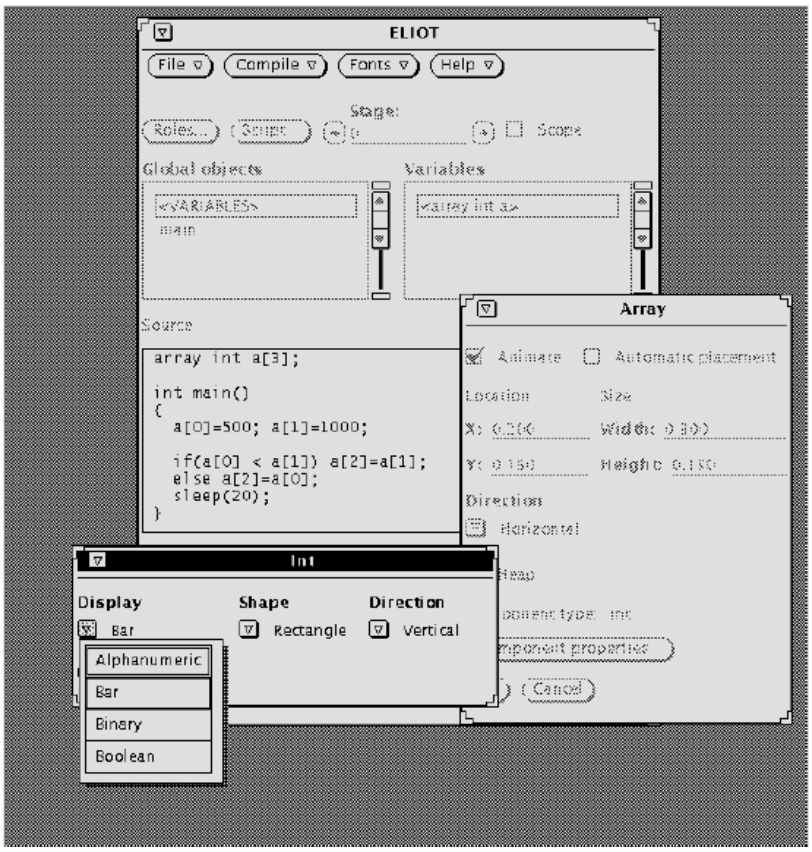

Figure1: The source code window and the animation setup windows of Eliot (Sutinen et al. 1997).

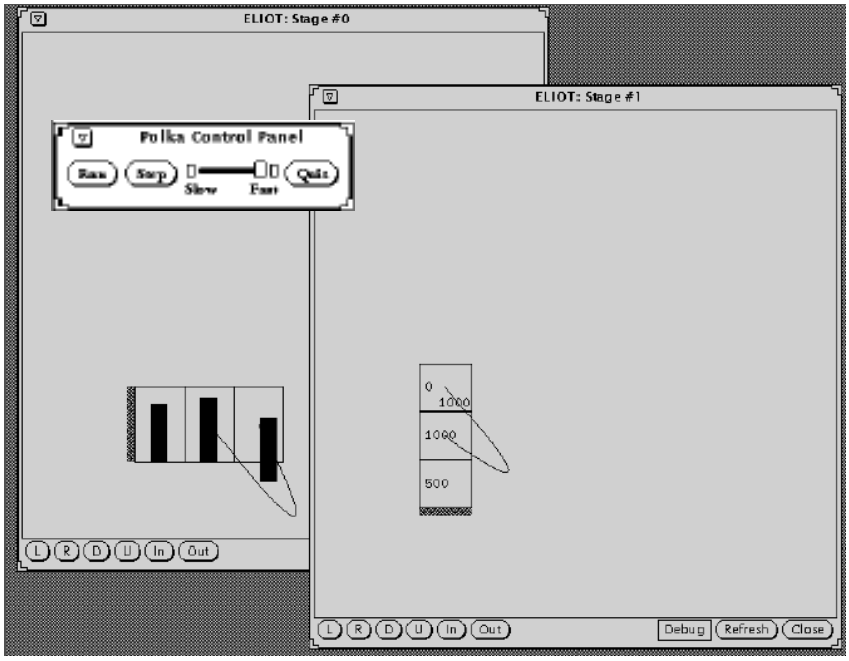

Figure 2: The animation windows and the control panel of Eliot (Sutinen et al. 1997).

3.2 Jeliot I

Because Eliot worked only on Linux workstations or on X terminals, it was not portable. That is one reason why Eliot was developed further (Ben-Ari et al. 2002a). Other reasons for its further development were the appearence of new platforms, like the World-Wide Web, new programming languages, like Java, and new methods of teaching, like constructivism. While Eliot needed to be developed, the functionality was expeacted to be the same as in Eliot (Sutinen et al. 2003).

In 1997 a new version of Eliot was finished in the University of Helsinki, under the direction of Erkki Sutinen and Jorma Tarhio, while some students made the new version (Sutinen et al. 2003). It was implemented using the programming language Java and so it got the name *Jeliot*, which is a short version of *Java Eliot*

(Haajanen et al. 1997). Later this version of Jeliot was called Jeliot I because other versions of Jeliot were completed.

As the result of development, Jeliot I was functionally similar to Eliot (Sutinen et al. 2003). Because Jeliot I was implemented using Java it could be used in the World-Wide Web environment, which made it portable as desired (Haajanen et al. 1997). It can be said that Jeliot I is Eliot in the World-Wide Web environment (Sutinen et al. 2003). An exception is that in Jeliot I, algorithms have to be written in EJava, which is a dialect of Java (Myller 2004).

Figure 3 shows the user interface of Jeliot I during animation. There are two windows, the right one is for the animation and the left one is for the source code and controlling the animation. There are also different windows for editing the code and there can be more windows for the animation and for each animation window there is an own setup window (Teräsvirta 1997).

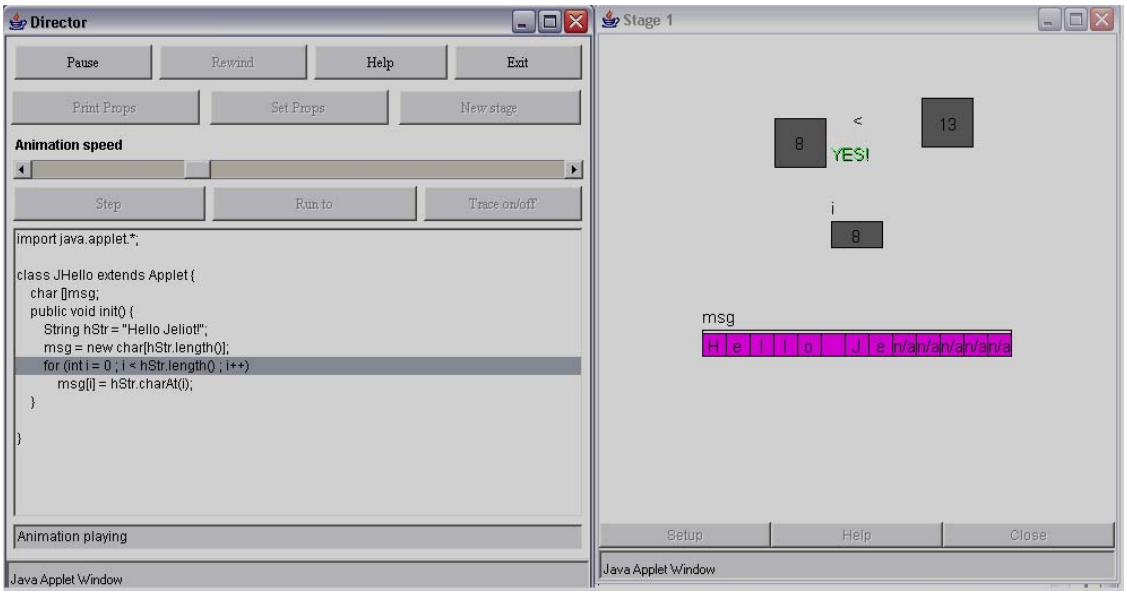

Figure 3: A screenshot of user the interface of Jeliot I during an animation.

Technically, Jeliot I and Eliot are totally different (Haajanen et al. 1997). Jeliot I uses self-animation and is semiautomatic just like Eliot, but these functions have

been implemented differently (Sutinen et al. 2003). For example the library of data types had to be done differently (Sutinen et al. 2003).

Jeliot I can still be tested and used at World-Wide Web address http://cs.joensuu.fi/~jeliot/jeliot.html.

3.3 Jeliot 2000

For novice users of Jeliot I there were too many windows in the user interface. Also it was sometimes too hard to understand the animation because all the source code was not animated (Ben-Bassat Levy et al. 2003). The two main ideas behind the development of the new version of Jeliot were to make the user interface simple for novice users and to animate the whole source code (Ben-Bassat et al. 2003). Eliot and Jeliot I are better for students who have some kind of knowledge of programming. The next version should be done for novice users (Ben-Ari et al. 2002a).

The new version of Jeliot was named Jeliot 2000 because it was based on Jeliot I and because it was created in 2000. (Ben-Bassat Levy et al. 2003). It was made at the Weizmann Institute of Science under the supervision of Mordechai Ben-Ari while Pekka Uronen was visiting there (Ben-Ari et al. 2002a).

Like Jeliot I, Jeliot 2000 was also implemented using the Java programming language because Java is portable and is often used in teaching (Ben-Bassat Levy et al. 2003). Jeliot 2000 programs have also to be written in Java language, but Jeliot 2000 understands only some Java programs (Myller 2004).

A huge change happened in the user interface of Jeliot 2000, as compared to the interface of Jeliot I. Jeliot 2000 is designed for the novice user and that is why it

consists of only one window, as can be seen in Figure 4. One window is easy for the novice users because novice users then are not as likely to be confused by the amount of different windows. The Jeliot 2000 window is separated into two parts; in the left part there is an area for the source code and in the right part there is an area for the animation. While a program runs, the user can follow, from the left part, which part of the code is animated at each moment. In the lower part of the window there are control buttons for the animation, which are labeled *step*, *play*, *pause* and *rewind*, as well as buttons for editing and compiling the code, which are labeled *edit* and *compile*. The *rewind* button rewinds the animation to the beginning of the algorithm. (Ben-Bassat Levy et al. 2003)

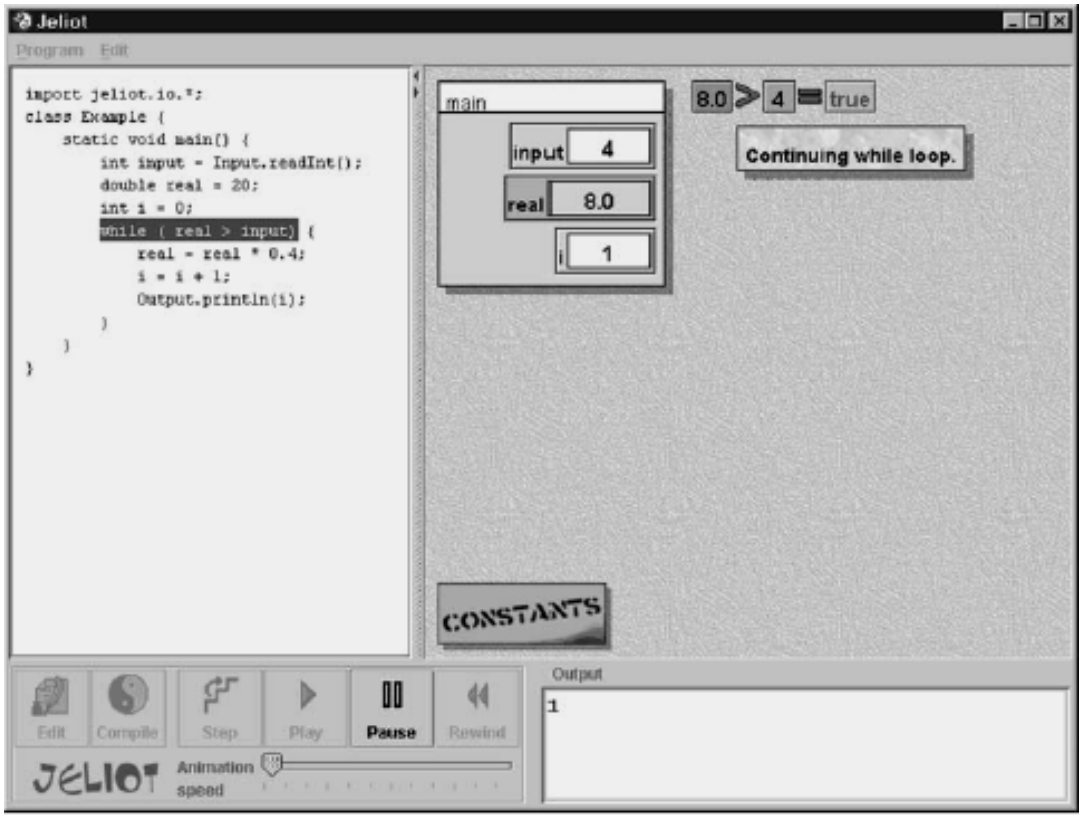

Figure 4: The user interface of Jeliot 2000 (Ben-Bassat Levy et al. 2003).

As in Eliot and Jeliot I, Jeliot 2000 data objects are still self-animated; however, animation in Jeliot 2000 is *fully automatic* (Ben-Bassat Levy et al. 2003). Fully automatic means that every item from the algorithm is animated. Every constant,

variable and statement is animated so that users know where each item comes from and how items are related to each other. (Ben-Ari et al. 2002a.)

3.4 Jeliot 3

Programs in Jeliot 2000 have to be written using the Java programming language, which is widely used in object-oriented programming, but Jeliot 2000 cannot understand objects (Ben-Ari et al. 2002a). Object-oriented programming is used to teach programming as the first language, which means that it is used by novice users (Moreno et al. 2004). That is why a new version of Jeliot was needed.

The new version of Jeliot was named Jeliot 3, because it was based on Jeliot 2000 (Moreno 2004). Niko Myller and Andrés Moreno made it at the University of Joensuu under direction of Erkki Sutinen and Mordechai Ben-Ari (Myller 2004). There exist lots of different versions of Jeliot 3 and the first one was completed in 2003 (Jeliot 3a 2005). I used the version which was available at the time of my research. The newest version of Jeliot 3 was finished in 2004 (Jeliot 3b 2005).

The user interface is similar to that of Jeliot 2000 because novice users have found it simple to use (Ben-Bassat Levy et al. 2003). The user interface of Jeliot 3 is displayed in Figure 5. The biggest changes in the user interface compared to Jeliot 2000 are line numbering and the few added menus (Myller 2004).

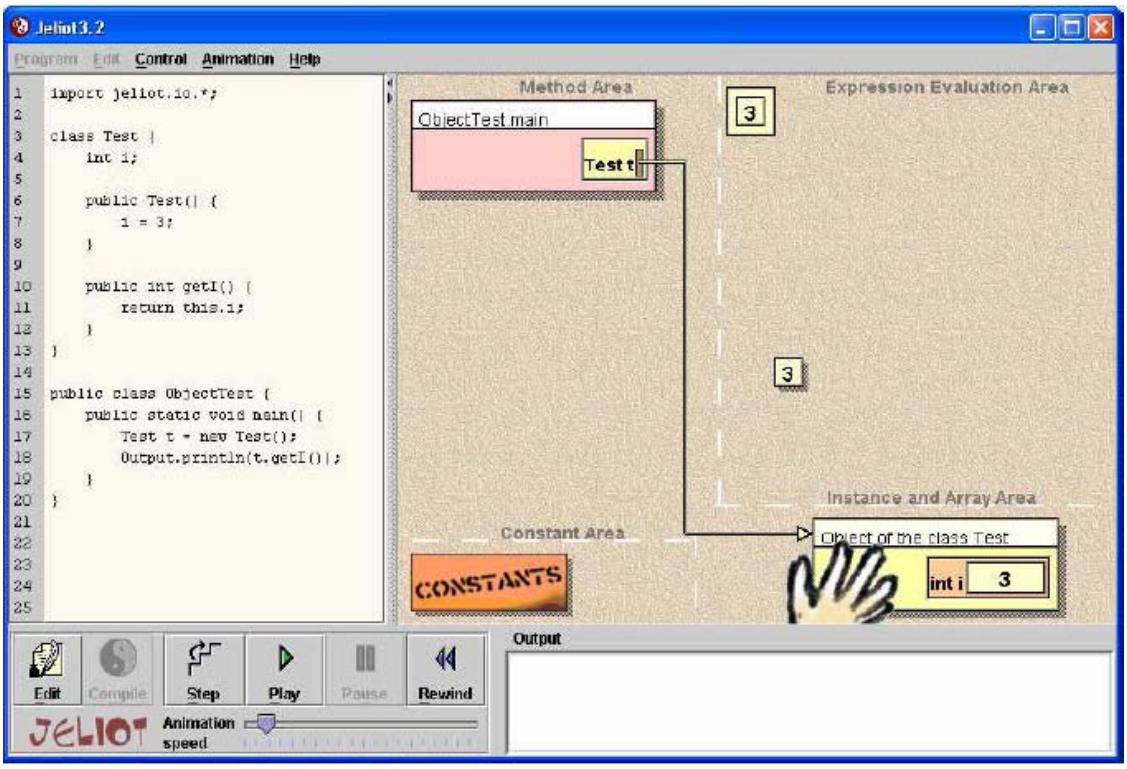

Figure 5: The user interface of Jeliot 3 (Myller 2004).

Jeliot 3 is implemented using the programming language Java because the design needed to be similar to that of Jeliot 2000. Like in Jeliot 2000, a Jeliot 3 user has to write algorithms using Java, although even Jeliot 3 cannot understand every Java program. However Jeliot 3 understands more programs than Jeliot 2000 (Myller 2004). For example the applets cannot be made using Jeliot 3 and there are some differences in the code which is written using Jeliot 3 like reading the inputs and printing the outputs.

In Jeliot 3 data objects are still self-animated and animation is fully automated because they were found useful and Jeliot 3 reused a lot of source code from Jeliot 2000 (Moreno et al. 2004). Technically the biggest improvement in Jeliot 3 is the possibility to animate object-oriented algorithms (Moreno & Myller 2003).

4 An empirical study of visualization

This chapter gives information on research procedures, participants, data collection and reliability. The results of this research are introduced in Chapter 5.

4.1 Method

According to Eskola & Suoranta (1999) it is not necessary to make a distinction between qualitative and quantitative research. Rather quantitative and qualitative research should be seen as linked together, not opposites or methods that exclusive each other's (Alasuutari 1999). Despite that in this thesis a distinction was made between them. In this thesis qualitative methods were used because they could offer information that is relevant to the phenomenon being researched. Qualitative research usually considers humans and meanings of human acts (Varto 1993). One goal of this thesis was to find out how students' use Jeliot 3; that obviously deals with humans and meanings of human actions.

There are many different kinds of qualitative procedures (Hirsjärvi et al. 2004). In this thesis, a case study[2] procedure was used because it is capable of investigating certain people acting in a specific environment (Yin 1983). Case study research can be researching a case in action. In case study research, the main goal is to understand a phenomenon more deeply (Syrjälä et al. 1996). According to Varto (1992), understanding comes about when the examiner and examinee are in the same world of living[3]. Then the examiner is a part of the context that he is researching. The *world of living* means the unity of humanity; it

[2] Metsämuuronen (2001) has translated the Finnish term tapaustutkimus to English as case study.

[3] World of living is translation from the Finnish word elämismaailma.

is a world that consists of the objects of relations between individuals, community, social interaction, set of values and humans in general. (Varto 1993.)

4.2 Participants

According to Eskola and Suoranta (1999), there are usually a small amount of participants in qualitative research. The criterion of appropriate data in qualitative research is not the amount of data, but the quality of data.

Since the aim of this research was to find out how novice users use Jeliot while learning object-oriented programming, those kinds of participants had to be found. In this thesis, participants were part of the ViSCoS project. ViSCoS refers to Virtual Studies on Computer Science (Meisalo et al. 2004). The ViSCoS project enables high school students to study the first-year university-level computer science studies of 22.5 ECTS credit points[4] in one and a half years (Haataja et al. 2001). In virtual studies, all studies happen individually, either using the Internet or certain books. Only the exam is conducted in a real classroom. ViSCoS students can get help from lots of different kinds of sources. Each of them has a tutor-teacher in their own high school, supervisors in the university and many peer students over the web and in the high school. In the ViSCoS project, a web-based learning environment is used. In that environment, there are course materials, exercises for each week, email and discussion forums. Students have to do exercises and send them to their supervisors using he electronic learning environment. Using email and discussion forums, supervisors give instruction to their students and give individual feedback about the exercises every week. (Meisalo et al. 2004.)

[4] 22.5 ECTS credit points are 15 credits in the Finnish university system (http://www.cs.joensuu.fi/viscos/).

In order for students to be able to take part in this research, they had to be a part of the ViSCoS project and they also had to be studying the course - Programming II. Programming II was chosen because in that course students learn the basics of object-oriented programming, which is an aspect of this thesis' research questions.

Because every student has to volunteer to participate in research, it was very difficult to find participants. Some students signed an agreement stating that they did not want to participate in research. I contacted by phone or email six students who were living in or near Joensuu and had studied in ViSCoS during the autumn of 2003. Most students did not want to be researched or did not study Programming II at that time.

Finally I got two students who volunteered to participate in this study. It is quite a low number, but according to Eskola & Suoranta (1999) in qualitative research the amount of the data is not the main issue. The purpose of the data is to help the researcher construct understanding from the phenomenon being researched (Eskola & Suoranta 1999).

4.3 Data collection

Data in this thesis were collected using qualitative methods (Eskola & Suoranta 1999). Through questioning and interviewing, it is possible to find out how people think and believe, but through observation it is possible to find out what really happens (Hirsjärvi 2004). Dean & Whyte (1958) have even noticed that there is difference between reports of behavior and behavior itself. Because the goal of this thesis is to see how students use Jeliot to help them to learn programming, the *observational* method had to be done. Observation means that examiner objectively watches an examinee and makes notes during the observation

(Metsämuuronen 2001). There are four different kinds of observation, but nonparticipatory observation was used in this study. In nonparticipatory observation there is no interaction between examiner and examinees. The examiner just watches and makes notes, which can be called learning-by-watching (Grönfors 1982.) Nonparticipatory observation is used in this research because it was desired that examiner concentrate totally on observation.

While collecting the data in qualitative research, the term *saturation* is used often. Saturation means that when the same themes start to appear with new examinees, then enough data have been collected (Hirsjärvi et al. 2004). In this research saturation could not be used because there were only two examinees available.

4.3.1 Planning the observation

Observation is a heavy process and it takes a lot of time to do it, so it is important to plan it well so that it does not have to be done twice (Tuomi & Sarajärvi 2002). These are the reasons why much time was spent in the planning of the observation.

One purpose of this research was to find out how students use Jeliot. The best way to observe it is to watch when examinees do their homework. The observation should be done in a place where there is good equipment for that. Normally nonparticipatory observation is made in specially defined places, like laboratories (Hirsjärvi 2004).
The method of observation has been criticized because the observer may disturb the observation situation with his presence (Hirsjärvi 2004). It means that examinees might not act naturally while there is an outsider present. That disturbion can be diminished with the examiner's long-term presence or the

examiner can act like he or she is not observing (Grönfors 1982). In this research, the latter approach to diminishing the examiner effect was used. Examinees could ask for help if they needed to, but the answers were given inaccurately.

Exercises were designed so that students would have to discuss them in pair. Discussion enabled the experimenter to better understand what the students were thinking and doing.

In the Programming II course, Jeliot had not been used before as homework. Because Jeliot 3 has some limitations[5], some of the exercises had to be changed or even new ones designed. This also took some time and needed to be planned carefully. Exercises can be found in the Appendix 1.

The exercises were planned so that the exercises became progressively more difficult. In the exercises, as much computation, like loops and ifs, and data structures, like arrays, as possible appeared. That made it possible to verify that students had already studied the course Programming I where those things were taught. In the first week, the students had to create an object and use its methods for simple calculations. In the second and the third weeks the students had to use *if* and *if-else, while, do-while* and *switch* statements. In the third week there was one exercise for inheritance also. In the last week there were exercises where students had to use arrays.

[5] Limitations have been introduced in paragraph 3.4.

4.3.2 Conducting the observations

The observations were made in a special laboratory at the department of computer science. That laboratory was the best place to conduct observations because there was a program called Camtasia. Camtasia can create videos from the desktop activities (http://www.techsmith.com/products/studio/default.asp). That program can also save sounds so the microphone was put near the screen. Camtasia saves the views from the screen and the sound in the same file so that it can be watched later as a movie. During the observation period, notes were made by the observer (Metsämuuronen 2001). The observer made the notes by pen but to make sure that everything important was stored the Camtasia software was used. The recorder saves every word and sound and it can be watched many times again (Grönfors 1982).

Observations were made in the beginning of the Programming II course because at that time the students were learning the basics of object-oriented programming; they were novice users.

The contents and implementation of the Programming II course are quite similar from year to year. The only exception was that some of the exercises had to be done using Jeliot 3 because of this thesis. Each week, students had four exercises. During the first four weeks, all students had to do two of them using Jeliot 3. The examinees came to the laboratory each week to do their homework.

4.4 Analysis of the data

In qualitative research, collecting and analyzing data are linked together so that it is hard to separate them (Grönfors 1982; Laaksovirta 1988; Metsämuuronen 2001; Syrjälä et al. 1996). According to Grönfors (1982) there are many reasons

for that. For example, the analysis has to be made by the same person who collected the data and choosing the problems and definitions are the parts of the analysis. While analyzing the data, the process of collecting the data might be even unfinished (Laaksovirta 1988).

Before it is possible to start to analyze the data, notes have to be copied (Metsämuuronen 2001). After four weeks of data collection there were almost eleven hours of Camtasia files. Some of the data was not relevant for this research, so certain selections were chosen for analysis. Only essential parts of the data were copied. The essential parts were those parts where the students had already compiled the programs and they worked with the visualization of Jeliot 3. Also the notes that have been made by pen have to be copied. (Grönfors 1982.) Analysis was conducted of these selected pieces of data.

In qualitative analysis, the aim is not to find out statistical facts. One reason for that is that very rarely there is enough potency for collecting so much data that those statistical facts can be found. One interview can easily include thirty pages to be transcribed and that takes a lot of time. The other reason is that sometimes it is not possible to collect enough data for statistical facts, and usually it is not even necessary. (Alasuutari 1999.)

There are many different kinds of ways to analyze the data (Grönfors 1982; Eskola & Suoranta 1999; Metsämuuronen 2001; Tuomi & Sarajärvi 2002). In this thesis content analysis was used. In content analysis, documents are analyzed systematically and objectively (Tuomi & Sarajärvi 2002). Documents can be letters, newspapers, reports or some other material in a text form. In this investigation, the documents were the notes resulting from the observation. When using contents analysis, the themes, topics and examiner's interpretations can be found in the documents (Grönfors 1982). Sometimes studies that have been made using contents analysis have been criticized because the examiner does not make sensible conclusions (Tuomi & Sarajärvi 2002).

In this investigation, the method of contents analysis that has been given by Syrjälä et al. (1996) was used. The first part of analysis was to make rough classification for themes and topics that come from the research problems. That can be made using mind maps or certain software. In this investigation, the rough classification consists of the themes: *the use of Jeliot 3*, and *the advantages and the disabilities of Jeliot 3*. After these rough themes and topics have taken shape, the documents are analyzed again. This time the rough themes and topics are sharpened for more detailed categories, for example in this research the theme - advantages of Jeliot 3 was sharpened to which exercises Jeliot 3 is good at and *the benefits of Jeliot 3 in object-oriented programming*. Then these categories are compared, similarities and conflicts are searched. At that point, the themes and categories are confirmed. Finally the ultimate classification is made with theory. Using theory, the data will be enriched. (Syrjälä et al. 1996.) The final classification can be found in the results chapter. The same kind of frame is introduced by Tuomi & Sarajärvi (2002).

4.5 Reliability

According to Varto (1992), reliability in research means that there are no accidental and irrelevant things in the research. In qualitative research these contingencies are eliminated from the data time being (Varto 1992). Research is deemed to be reliable if there are no conflicts (Grönfors 1982). The conflicts may come if the data and discussion are not equal to the thoughts of the examinee or if they are not linked to the theory (Syrjälä et al. 1996). The other perspective on evaluating the reliability of the qualitative research is to concentrate on the essential instrument of the research, which is the examiner itself, which means that the whole process of research has to be evaluated (Eskola & Suoranta 1999).

The reliability of the research is forwarded to the reader from the report and that is the reason why the examiner has to explain the research process in a convincing way (Syrjälä et al. 1996). The report has to be also written so that the reader can follow examiner's reasoning and reader can agree or disagree with examiner's conclusions (Mäkelä 1990).

All of these things have been considered in this research. All the time theory and data are linked together and they support each other. The report is written in detail and in that way the results are clearer and more understandable (Tuomi & Sarajärvi 2002).

5 Results

The final classification of themes and topics can be seen in this chapter. Each category has its own paragraph and each detailed category has its own sub paragraph. The detailed categories took shape from the information that appeared in the data, while the data were being analyzed.

This chapter reports the results of this investigation. The first paragraph is about how novice users use Jeliot in object-oriented programming. The students' feeling of whether Jeliot is helpful or not is explained in the second paragraph. The third paragraph describes what kinds of exercises Jeliot is the most effective at. At the end of this chapter, the effects of Jeliot in the students' motivation are presented.

5.1 The use of Jeliot 3 in the learning of object-oriented programming

5.1.1 New terms

Jeliot 3 got its present looks and functions largely from previous versions; it retained the best parts from the previous versions while some new functions were added (Myller 2004). The user interface and the visualization are implemented in Jeliot 3 so that it serves novice users (Ben-Bassat Levy et al. 2003). In this research, the students were novices in object-oriented programming but they had studied programming before so they had some kind of knowledge about constructing and implementing algorithms and using the Java programming language. It was good that they knew already the basics of

programming like loops, if and while statements and so on and they had become familiar with the capabilities of Java. But the disadvantage was that they had to change their thinking to the object-oriented mode (Ben-Ari et al. 2002b). That was found in this investigation because during the observations they had huge problems with it. One thing that may have caused these problems is the amount of the new terms (Ben-Ari et al. 2002b). It appears in the following comments by the students:

> *"We don't make any of those objects now … there are only two pieces of classes. Then we can make it using methods.*[6]*"*
>
> *"This is this playing with the new terms."*[7]
>
> *"Put simply that… damn class which we don't know… I haven't used yet enough this class thing."*[8]

While they were discussing how they should implement the program, they also had problems because they needed to explain things to one another and they did not know the right terms.

Every part of the algorithm should be visualized so that students know, for example, where all of the constants come from and how different terms like *classes, objects* and *methods* are linked together (Ben-Bassat Levy et al. 2003). This is done in Jeliot 3, but in the beginning of the observation in this research the students had problems with these subjects. They did not know what terms class, object and method meant and how they are connected.

> *"Why these have to be objects because these, shouldn't these work as methods… it is…*

---

[6] "Eipä myö tehä nyt ollenkaan noita olioita... Siinä on vaan luokkia kaks kappaletta. Sitten me voidaan tehdä metodeilla."

[7] "Tää on tää uusien termien kanssa leikkiminen."

[8] "Laita vaan se... himputin class mitä myö ei tiietä... Tätä ei oo tarpeeks käyttäny tätä luokkatsydeemiä vielä."

Well I don't know anything about it.
It's that what we don't understand yet.
By the way is it coming under what?
It comes under the class."[9]

5.1.2 The display of the visualization

In this investigation, students also had problems following the execution of the programs. It was found that the problems were more several in the beginning, which supports the theory that the visualization is a tool whose use has to be learnt and whose advantages appear over long-time use (Ben-Bassat Levy et al. 2003). The first time that students saw and tried Jeliot 3 was when they came to do those exercises for the first time. Because they were not used to use that tool, it took them a couple of weeks to learn it. The next statement was taken from the students' discussion during the execution of a program.

"Why it jumped straight there by the way?
I don't know. This is a hard one."[10]

One thing that also caused problems in following the execution of the algorithm was the speed of the visualization. In Jeliot 3, there is a possibility to choose the speed of the visualization from the scroll bar; it can be changed during the execution (Myller 2004). Students used the maximum speed all the time for the visualization. Petre found that novices have problems to noticing all the

---

[9] "Miks näitten pitää olla olioita ku nää, eikös nää ihan metodeina toimi... no se on...
Mä en tiiä ollenkaan.
Se on sitä mitä ei vielä ymmärretä.
Tulleeks se muuten mihin alle?
Se tulloo classin alle."

[10] "Miks se hyppäs sinne suoraan muuten?
En tiiä. Tää on paha."

information in a visualization (Petre 1995). That problem was exaggerated in this case because of the speed of visualization. During the observation sometimes the students had to be told to slow down the speed of the visualization. Once they discovered it by themselves.

> *"Let's look it slower. At least for me it passed teeming where the hell it… Shall we watch this thing through slowly?*
> *Okay, let's watch.*
> *Thus slower.*
> *The beginning can go faster.*
> *Not even that."*[11]

Another option for watching the visualization slower in Jeliot 3 is to use the step-by-step mode, where the program execution moves only one step at a time. When a user wants to continue watching the visualization he can watch the next step or choose with a continuous visualization. In this investigation, the students never used the step-by-step mode. Lattu et al. also noticed the same (2000). During the observation of the students, one time another student mentioned the possibility and his willingness to use the step-by-step mode.

> *"Err, it passed totally again. Slooooweeer. Furthermore. Or it could be watched step by step."*[12]

When they got used to following the animation they learnt to find the mistakes from the source code faster. Once they both started to laugh when they realized from the animation how stupid a mistake they had made.

---

[11] "Katotaan se hittaammin. Mulla ainakin meni ihan vilisten ohi toi missä hitossa se... Katottaanko hittaasti läpi tämä homma?
No katotaan.
Siis hittaammin.
Alku saa mennä nopeemmin.
Ei etes."

[12] "Ääh, meni kyllä taas ihan ohi. Hiiitaaammin. Edelleen. Tai sen vois kahtoa steppeinä."

5.1.3 Ready-made examples

While students did their exercises, they watched examples from the course material many times. It is good that students can watch and try some ready-made examples; in that way they can understand how the algorithm works by compiling it with different inputs (Khuri 2001). But in this case students did not compile those examples, they just read them and tried to understand what each line in the source code meant. Because they were novices they did not understand all of them.

> *"I don't understand what it does in that place because… Do you understand?*
> *No. But it doesn't matter if only it works."*[13]

If students would have compiled the examples, they would have understood them because those examples were put in the course material to make it easier to understand the theory.

5.1.4 The need for verbal explanation

While students did those exercises they asked for help quite a few times. They were not told the answer, but they were told the way to get there. This kind of help decreases the feelings in the students that there was an observer present (Grönfors 1982). That they asked help, supports the theory that students need explanations to make the visualization effective (Kehoe et al. 1999; Mayer 1997). Because the studies are virtual, the students do not always have a chance to get

[13] "En ymmärrä mitä se oikein tos tekee ku... Ymmärrät sie?
En. Mut ei se haittaa kuhan se toimii."

verbal explanations. So everything that they have learnt they have learnt by themselves and they need support from the teacher (Torvinen 2001).

5.2 The usefulness of Jeliot 3

Visualization supports the learning by doing method, which means that students learn by attempting and making mistakes (Khuri 2001; Kähkönen & Piiroinen 2002). In this research students also used this method. First they made a program and they watched how it worked and if it did not work correctly they fixed the algorithm. Sometimes they even encouraged each other to do first and think about the problem after that.

> *"Let's just put that return and try it, if it doesn't work then we start to think."*[14]

5.2.1 The disadvantages of Jeliot 3

The first time that students wrote programs with Jeliot 3, they compared it many times to Crimson, which they had used before. Crimson is a source code editor for programming languages, such as Java, which has syntax highlighting and other features such as undo/redo, user tools, macros, spell checker and some others (http://www.crimsoneditor.com). Their opinion was that certain things were better in Crimson, like using the colors for highlighting the source code and showing which parentheses belong together.

> *"Besides is here any chance that here come strings and ints with the code of color forth. It would be quite good.*

[14] "Laitetaan vaan se return ja kokeillaan, sit ruvetaan miettimään jos ei toimi."

It's benefit for Crimson.
Would it be here also?"[15]

"Crimson showed directly which parentheses closed each others."[16]

The students knew some basics about programming from the previous course of ViSCoS and that made them sometimes feel that they had not made the mistake in writing the program, but that the mistake was in Jeliot 3. The other reasons for that feeling might be that they knew that the new version of Jeliot 3 was for the first time in use and the characteristics of the source code in Jeliot 3.

"I slightly think that this is only caused by Jeliot.
Yeah, this might be because of Jeliot.
But we haven't been able to try this with Crimson either but…
Damn I wish we had Crimson."[17]

"But the trouble is still in Jeliot.
This is in Jeliot this problem.
This is like it would work in Crimson."[18]

---

[15] "Onks tässä muuten mitenkään et tähän tulee stringit ja intit värikoodeilla näkyviin. Ois aika hyvä.
Se on Crimsonissa plussaa.
Oiskohan se tässäkin."

[16] "Crimson näytti suoraan mitkä sulut sulkivat toisensa."

[17] "Musta vähän tuntuu, et tää johtuu vaan Jeliotista.
Tää on kyllä ehkä Jeliotista.
Ei olla kyllä päästy kokkeilemmaan kyllä Crimsonillakaan, mutta...
Vitsi kun ois Crimson."

[18] "Mutta vika on edelleen Jeliotissa.
Tämä on Jeliotissa vika.
Tämä on niinkun Crimsonissa toimis."

5.2.2 Advantages of Jeliot 3

In some cases the students thought that Jeliot 3 was better than Crimson. It is good that they found out the benefits of Jeliot 3. They liked the logic and continuity of the visualization. These feelings come from the facts that the visualization of Jeliot 3 is designed to cover every constant and variable from the source code and all connections between classes, objects and methods are visualized (Moreno et al. 2004). At least the other student thought that Jeliot 3 enriches thinking. That comes from the idea that visualization makes it easier to make the right mental model in the students' mind (Crapo et al. 2000; Meisalo et al. 1997). The students thought that Jeliot 3 was educationally effective. That was meant to be because Jeliot 3 was designed for teaching and learning object-oriented programming (Moreno & Myller 2004).

> *"But of course this is much much more logical because straight from here can be seen where is…*
> *Yes, that's true.*
> *So this just like goes goes goes goes."*[19]

> *"This of course enriches the thinking of mind."*[20]

> *"This is more educational program.*
> *That's true, this tells…"*[21]

[19] "Mutta täähän onkin tieteskin paljon paljon loogisempi kun tästä näkee suoraan missä on...
Nii näkee.
Siis tää niinkun etenee etenee etenee etenee."

[20] "Tää rikastaa mielen ajattelua."

[21] "On tää enemmän opettavainen tää ohjelma.
Nii on, tää kertoo..."

5.2.3 Jeliot 3's benefits in object-oriented programming

The results which are introduced above are not strictly connected to object-oriented programming. This paragraph tells about Jeliot 3's benefits in object-oriented programming.

In the beginning the students had problems understanding how the object should be created and how the object can call some method. They watched some ready -made examples and copied the part where object was created and method was called. After they watched the visualization for a while, one student realized what was happening in the visualization and explained it to the other.

> *"Here the object is created. And here it calls the… err... well... the method."*[22]

They also had some problems knowing from which class the object should be created. Once the students had created the object from the wrong class, they wondered why the object could not be called.

> *"This object is created from the wrong class. This object cannot call that method because it is not in the same class."*[23]

5.3 Object-oriented exercises for Jeliot 3

In order that could be reached the cognitive connection between searching and using the information and the fundamental features from the information could be adduced has to be resolved which kind of programming exercises are wanted to

---

[22] "Tässä se olio luodaan. Ja tossa se kutsuu sitä...tota...ööh... metodii."

[23] "Tää olio on luotu väärästä luokasta. Tää olio ei voi kutsua tota metodii, koska se ei oo samassa luokassa."

taught using visualization (Petre et al. 1998). The Jeliot exercises for the first four weeks' of the ViSCoS-project's Programming II course were planned so that there computations, like loops and ifs, and data structures, like arrays, appear as much as possible. That made possible the reason that students had already studied the course Programming I where those things were taught. Students' opinions are that if arrays, if statements and logical operations are difficult things to learn then visualizing those things would be helpful (Torvinen 2001).

Programming II is the follow-up course to the course Programming I. In the end of Programming I, students had been taught how to make applets using Java. In the beginning the course Programming II there are also exercises where students needed to implement applets. This sometimes appeared during the observation period when students started to make applets before they realized that applets were not need in these exercises and that it is not possible to make applets in Jeliot 3.

5.4 Jeliot 3 and the motivation

Motivation and learning are closely associated (Ikonen 2000). Motivation is a phenomenon which is directed by *activity*, which makes students behave in a certain way, aim at the direction in which the activity is headed, and *feedback*, which either helps or hinders students to complete a task (Ruohotie 1998). According to Niermeyer & Seyffert (2004) motivation can be divided into general and specific types of motivation. Everyone of us has some wishes, things and goals that motivate us and that can be called general motivation. Specific motivation makes us commit ourselves toward a certain aim. The extent of these varies between people but it greatly affects the amount of perseverance that people have towards carrying out an activity. Motivation can also be divided in to the external and internal types of motivation (Ruohotie 1998). External motivation

is objective, like articles and events, and internal motivation is subjective, like feelings. Maybe the most common example of external motivation is money and of internal motivation is satisfaction.

In a learning situation, the teacher ensures that the motivation occurs and is maintained (Ikonen 2000). The teacher has to choose studying manners which launch eagerness to learn (The basis of the curriculum for primary and secondary school 2004). The motivation for studying is born when the teacher gives students exercises that are hard enough that students get reinforcement when they solve them. At the same time the exercises have to be easy enough so that students get lots of feelings of success and the exercise has to be varied and interesting enough so that students do not get bored (Ruohotie 1998).

The knowledge or the skill that is being taught has to be relevant to the student's life. In the beginning of the process of learning, there might be a need for the external motivators like good grades or punishment. Motivation has an influence on the quantitative and the qualitative factors of learning as well as on the learning outcomes. The experiences which have taken shape from the learnt things have an influence on motivation. (Ikonen 2000.)

As has been proved, visualization increases the motivation for learning (Kehoe et al. 1999; Sutinen et al. 1997). That was also a finding in this research. The first time students did exercises they gave up quite easily.

> *"We shall let it be."*[24]
>
> *"This is not working. Let's move on to another exercise."*[25]

[24] "Annetaan olla."

[25] "Ei tää toimi. Siirrytään siihen toiseen tehtävään."

But in the end they were motivated when they worked with Jeliot 3 and they wanted to have those exercises on a floppy disk so that they could continue working with those exercise at home.

> *"This is going to take a long time.*
> *It doesn't matter, this is cool, and this is cute."*[26]
>
> *"This is, by the way, great program. Besides this I have to get on a floppy. At least I could practice with this further long time."*[27]

[26] "Tässä kestää aika pitkään.

Ei se haittaa, tää on siistii, tää on nättii."

[27] Tää on muuten hieno ohjelma. Tää muuten pittää mun saada korpulle. Ainakin saisin harjoteltuu tässä vielä pitkän aikaa."

6 Discussion

The aim of this research was to find out how novices users use Jeliot 3 and if it is useful for them. The aim was also to determine in what kind of exercises in object-oriented programming Jeliot 3 can be used and if Jeliot 3 have any effect on the students' motivation. The introduction has a discussion of the theory of visualization, how it can be used effectively in teaching and in the history of Jeliot 3. Through these, the possibilities of using visualization in teaching and how students can be helped in the process of learning object-oriented programming using Jeliot 3 was discovered.

The conclusions of this study will help to develop Jeliot 3 even further because I see Jeliot 3 as a very useful tool in teaching object-oriented programming for novices. In the conclusions I also discuss the ways to use Jeliot 3 as effectively as possible.

6.1 Interpretation of the results

Most of the results of this investigation were the same that others had found out earlier; visualization affects positively on students' motivation (Kehoe et al. 1999; Sutinen et al. 1997) and students feel that visualization helps in the process of learning (Lawrence et al. 1994). Also Kähkönen & Piiroinen (2002) have noted the advantages of visualization. All of these things are very important for novice users so that they get the inspiration to start learning programming and also to keep up that motivation.

In conclusion, my results agree with Torvinen (2001) that visualization should be used while teaching difficult things, but I also agree with Ben-Bassat Levy et al.

(2003) that visualization should be used all the time during the course so that it can be learnt and used effectively. In this investigation, students used Jeliot 3 for the first time and only for a short period of time. Both of these things made it difficult to learn Jeliot 3 so that it could have been used effectively. Students may have used Jeliot 3 better if they had had a chance to learn how to use it before the observations ware made.

Understanding how to read the visualization is not obvious; it is a skill that has to be learnt (Petre & Green 1993). Novice students do not notice all the information from the screen (Petre 1995). That is the reason why there is a speed control slidebar and the possibility to watch the visualization in a step-by-step mode in Jeliot 3. According to Lattu et al. (2000) and this investigation, students do not like to use the step-by-step mode. The students in this ivestigation preferred to put the speed of visualization to the maximum level and then realized that they had not understood what happened in the visualization. From these findings it is apparent that students should be taught how the speed of visualization can be changed, how the step-by-step mode works, as well as why they should use these characters.

The other thing that confused the students was that they had already learnt the basics of programming and they started thinking in the object-oriented way. One way to avoid this problem is to teach object-oriented programming as the first programming language. That way students do not have to change their way of thinking. But starting to learn programming with object-oriented language is also problematic. When teaching begins with objects, problems with languages like general computation (for example source code, compiling and execution) and basic terms of programming (for example variables, procedures, functions and parameters), come besides the terms of object-oriented programming (for example class and object) appear. For novice students, there might too much to handle all at the same time. (Ben-Ari et al. 2002b.)

Students had used Crimson Editor earlier in their studies. That also may have confused them a bit because every tool has its own specific characteristics. For example, students tried to do things with Jeliot 3 which are only possible to do with Crimson. It is great that there are many different kinds of useful tools for students because all of them have their own advantages and they been have designed especially for some specific use. Then each student or teacher can choose which tool is the best for him and in his use. For example, the program called BlueJ was developed for the purpose of teaching object orientation with Java. So if someone feels that Jeliot 3 is not a good tool for teaching or learning object-oriented programming, he can choose for example BlueJ.

In this investigation, the participants were involved in the ViSCoS project's Programming II course, in which students learn the basics of object-oriented programming. Observations were made during the first four weeks of the course while students did their homework. Although the exercises were designed very carefully, they were not strictly connected to the course because one of the course's topics for those four weeks was implementing a user interface. Also students had learnt in the end of the Programming I course how to implement applets and they tried to implement those also with Jeliot 3, which is not possible. Because of that I think that a course where Jeliot 3 is used should be designed so that Jeliot 3 is the main tool and that all the exercises should be solved using Jeliot 3.

During this case study, it was found that students can get new viewpoints for programming using visualization. Clearly can be seen when students get the feeling that now I understand. If you put fleas into the pot from where they can jump off if there is no cover and you close the cover for a while and then open the cover, the fleas do not jump off from the pot because they have not been able to do that for a while and they have "learnt" that they can not get out from the pot. Problems and difficult things in learning can be seen as the same kind of cover that students get used to that this is something that I can not solve. While novice

students are learning object-oriented programming I think that Jeliot 3 is a tool for them to get out from the pot.

6.2 Evaluation of the research process

From the observations, very useful data were gained, but it would have been better if there had been more participants in this research, even though there are usually only a few participants in qualitative research studies (Eskola and Suoranta 1999). More participants than two were recruited, but there were no volunteers from the ViSCoS project. Other students may have given different opinions and feelings about using Jeliot 3 and its usefulness. On the other hand more students would have meant that the observations would have taken significantly more time and that problems with timetables may have arose. Even with one pair, it was quite difficult to fit timetables together.

Also, it would have been nice to observe the students for a longer time period. A longer time would perhaps have shown how well they have learnt to use Jeliot 3 and how well they really learnt the basics of object-oriented programming. In a longer observation it would have been possible to let students do more complicated exercises, which contain, for example, nested loops.

Each week the students did two exercises without stopping and the average time that it took was two and a half hours. It is definitely too long a time for students to concentrate on designing and implementing programs. In this research the students were capable of effectively studying for one and a half hours at a time. After one and a half hours, they were more interested in going home than doing the exercises. Maybe that one and a half hours comes about from the fact that they have gotten used to working in primary, secondary and high school for at most two lessons together which is two times forty five minutes. It would have

been better if the sessions of observation had been planned so that one session lasted only one and half hours. Either each week should have had only one exercise or the students should have come twice a week for the observation.

While analyzing the data there arose some statements and topics which needed more detailed information. That information would been available had the students been interviewed after the observations had been made. That interview was impossible to do because I had to leave the country for exchange studies. After the period of exchange studies, interviewing was not appropriate because the observations had been done such a long time ago that students' memories of the observation would not have been clear anymore. With the help of the interviews the data would have been even more opulent.

6.3 Ideas for the further development of Jeliot 3

In Jeliot 3, the inputs and printing outputs have to be written differently than in standard Java. That was a little bit strange for the novice students. The commands that Jeliot 3 needs for inputs and outputs are not difficult things to learn, but it would be easier if there were no difference between normal Java and Jeliot 3. The other thing that appeared during the observation was that the letter *L*, when it is not written as capital, can be a mistake for the number 1. That is a problem because Jeliot 3 uses the font called *Times New Roman.*

Students are used to naming the variables, classes, methods and so on so that the name tells what the variables are used for. This is good, but in Jeliot 3 there is only a limited amount of space where the name can be written because the main area of the window is used for visualization. If the name has too many characters, it does not fit into the allotted space. When students have to change the names of the variables, they might not know what each variable means and it

can be very confusing, at least for novice users. I think that Jeliot 3 should cut off the end of the names and show only the beginnings of the names in the visualization.

Earlier students had used Crimson; they said that it indicates which parentheses belong together. This parentheses feature was very helpful for the students because if they forgot to put the ending parenthesis on the program it would not work correctly. For example, one problem was that if the missing ending parenthesis was from the first class, the error message told the students to put the ending parenthesis at the end of program, which is not the right place. If the ending parenthesis is put at the end of the program it compiles, but the Jeliot 3 program fails when it gets to the part with the missing parantheses. It is a big problem in Jeliot 3. If Jeliot 3 could show which parentheses belongs together, these kinds of problems would not exist.

Crimson also uses colors to highlight the source code, so that it is easier to find the key words from the source code. For example, the type of variables are colored as well as the compulsory words in loops like *while*, *else* and *if*. This would be also helpful for the students to find out more easily where the key words are and if everything is written correctly.

It would also be helpful to have a feature where it would be possible to choose which parts of the algorithm are visualized. If the mistake is in the end of the program, students have to watch all of the visualization before they get to the problematic part. In long programs, it would make the execution of the program faster. Of course there is the possibility that students do not know which parts should be visualized. The aim of the visualization in Jeliot 3 is to visualize all the source code not just some parts, but for the teacher it would be nice to have the possibility to show only some parts of the program visualization.
From the teacher's point of view, Jeliot 3 should have the possibility to highlight some parts of the algorithm. That highlighting effect could be, for example, a

different color. Then it would be easier to show to the students which are the important or difficult parts of the algorithm. There should also be the possibility to choose the size of the font because when the teacher uses the data projector to show the algorithm to the whole class in the classroom, the text should be seen at the end of the classroom, too.

6.4 Possible ideas for further research

The students worked in collaborative pairs. However, there are some problems with collaboration. For example, while one person works the other person might just sit and wait for the answers. Also, one person might just make all the decision, whether they are right or wrong. In this investigation, the students cooperated well although the stronger person tried many times to convince the other student that answers were right when in fact they were wrong. Co-operation would have also been an interesting factor to investigate, but somewhere the line has to be drawn or the research becomes too large-scale.

Interesting avenues of research connected to this thesis' topics would be comparative research between two classes where both are learning the basics of object-oriented programming but one class is using traditional methods and the other is using Jeliot 3. This might bring about good points how Jeliot 3 can be developed further and how Jeliot 3 could be used in a way that is most effective for novice students.

Other further studies could be made when Jeliot 3 is used while teaching the basics of an object-oriented programming language. For example it might be interesting to determine how suitable Jeliot 3 is for that purpose and if students feel that Jeliot 3 is helpful in the process of learning. Another question is how Jeliot 3 is used among students of different ages while learning the basics of

object-oriented programming. For example the research questions might be: Are there differences with Jeliot 3 for different aged users and is Jeliot 3 more powerful for some learners of different ages.

Jeliot 3 is designed for the novice users, but can experts use Jeliot 3 as a tool of thinking and conversiation with peers as Petre et al. (1998) say. If not should a new version of Jeliot be designed for the expert users to help them in their process of learning?

References

Alasuutari, P. 1999. 3rd edition. Laadullinen tutkimus (Qualitative research). Gummerus kirjapaino Oy: Jyväskylä.

Anderson, J. & Naps, T. 2001. A Context for the Assessment of Algorithm Visualization System as Pedagogical Tools. *Proceedings of the First Program Visualization Workshop.* Porvoo, Finland. 121-130.

Ben-Ari, M. 2001. Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching, 20(1), 45-73.

Ben-Ari, M., Myller, N., Sutinen, E. & Tarhio, J. 2002a. Perspectives on Program Animation with Jeliot. In: Diehl, S. (Ed.), Software Visualization. Vol. 2269 of Lecture Notes in Computer Science. Springer-Verlag, pp. 31–45.

Ben-Ari, M., Ragonis, N. & Ben-Bassat Levy R. 2002b. A vision of visualization in teaching object-oriented programming. Second Program Visualization Workshop. HornstrupCentret, Denmark. 84-90.

Ben-Bassat Levy, R., Ben-Ari, M. & Uronen, P. 2003. The Jeliot 2000 program animation system. Computers & Education 40/2003. 1-15.

Camtasia Studio. http://www.techsmith.com/products/studio/default.asp (Accessed May 26, 2005)

Card, S., Mackinlay, J. & Scheiderman, B. 1999. *Readings in Information Visualization: Using Vision to Think.* Morgan Kauffmann Publishers Inc., San Francisco.

Crapo, A., Waisel, L., Wallace, W. & Willemain, T. 2000. Visualization and the Process of Modeling: A Cognitive-theoretic View. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data mining.* ACM, Boston.

Crimson Editor. http://www.crimsoneditor.com (Accessed May 26, 2005)

Dean, J. & Whyte, W. 1958. How Do You Know If the Informant Is Telling the Truth? Human Organization, 17 (2).

Eskola, J. & Suoranta, J. 1999. 3rd edition. Johdatus laadulliseen tutkimukseen (Introduction to the qualitative research). Jyväskylä: Gummerus.

Grönfors, M. 1982. Kvalitatiiviset kenttätyömenetelmät (Qualitative methods for working on the field). Werner Söderström Osakeyhtiö: Juva.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T. & Vanninen, P., 1997. Animation of User Algorithms on the Web. In: Proceedings of VL'97 IEEE Symposium on Visual Languages. pp. 360–367.

Haataja, A., Suhonen, J. Sutinen, E. & Torvinen, S. 2001. High School Students Learning Computer Science over the Web. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ) 3(2), October 2001. Available in URL: http://imej.wfu.edu/articles/2001/2/index.asp (Accessed April 26, 2005).

Haibt, L. 1959. A program to draw multi-level flow charts. In: Proc. of the Western Joint Computer Conference, volume 15, 131-137.

Hirsjärvi, S., Remes, P. & Sajavaara, P. 2004. Tutki ja kirjoita (Study and write). Helsinki. Kirjayhtymä.

Hong, J. 1998. *The Use of Java as an Introductory Programming Language.* ACM Crossroads, 4.4. The ACM's First Electronic Publication, http://www.acm.org/crossroads/xrds4-4/introjava.html (Accessed May 13, 2005)

Hundhausen, C. D., Douglas, S. A. & Stasko, J. T. 2002. A Meta-Study of Algorithm Visualization Effectiveness. Journal of Visual Languages & Computing 13 (3), 259– 290.

Ikonen, O. 2000. Oppimisvalmiudet ja opetus (Readinesses for learning and teaching). PS-kustannus: Jyväskylä.

Jeliot 3a. http://cs.joensuu.fi/~jeliot/news.php (Accessed April 21, 2005)

Jeliot 3b. http://cs.joensuu.fi/~jeliot/index.php (Accessed April 21, 2005)

Kashihara, A., Terai, A. & Toyoda J. 1999. Making Fill-in-Blank Program Problems for Learning Algorithms. In Cumming, G. (ed.) *Advanced Research in Computers and Communications in Education*, IOS Press, Amsterdam, pp. 776-783.

Kehoe, C., Stasko, J. & Taylor, A. 1999. Rethinking the evaluation of algorithm animations as learning aids: an observational study. Technical report GIT-CVU-99–10, Georgia Institute of Technology.

Khuri, S. 2001. Designing Effective Algorithm Visualizations. *Proceedings of the First Program Visualization Workshop.* Yliopistopaino, Joensuu. 1-12.

Korhonen, A. 2003. Visual Algorithm Simulation. Doctoral Dissertation. Helsinki University of Technology. Department of Computer Science and Engineering.

Kähkönen, H. & Piiroinen, K. 2002. *Ohjelmoinnin opettaminen Empirica Controlilla (Teaching programming with Empirica Control).* Master's thesis, University of Joensuu, Department of Computer Science.

Laaksovirta, T. 1988. Tutkimuksen lukeminen ja tekeminen (Reading and making the research). Hakapaino Oy: Helsinki.

Lahtinen, S.-P., Sutinen, E. & Tarhio, J. 1998. Automated Animation of Algorithms with Eliot. Journal of Visual Languages and Computing 9 (3), 337–349.

Lattu, M., Meisalo, V. & Tarhio, J. 2003. A visualization tool as a demonstration aid. Computers & Education 41 (2), 133–148.

Lattu, M., Tarhio, J. & Meisalo, V. 2000. How a Visualization Tool Can Be Used-Evaluating a Tool in a Research & Development Project. In: 12th Workshop of the Psychology of Programming Interest Group. Corenza, Italy, pp. 19–32, http://www.ppig.org/papers/12th-lattu.pdf (Accessed May 19, 2005).

Lawrence, A., Badre, A. & Stasko, J. 1994. Empirically evaluating the use of animations to teach algorithms. In: Proceedings of the1994 IEEE Symposium on Visual Languages. IEEE Computer Society Press, Los Alamitos, CA, 48-54.

Markkanen, J., Saariluoma, P., Sutinen, E. & Tarhio, J. 1998. Visualization and imagery in teaching programming. In: Domingue, J., Mulholland, P. (Eds.), 10th Annual Meeting of the Psychology of Programming Interest Group. Knowledge Media Institute, Open University, Milton Keynes, UK, pp. 70–73.

Mayer, R. 1997. Multimedia learning: are we asking the right questions? Educational Psychologist, 32(1), 1–19.

Meisalo, V., Sutinen, E. & Tarhio, J. 1997. CLAP: Teaching Data Structures in a Creative Way. *Proceeding of the ITiCSE '97, Integrating Teaching into Computer Science Education* (Ed. Davies, G.), Uppsala University, Uppsala, 117-119.

Meisalo, V., Sutinen, E. & Torvinen, S. 2004. Classification of Exercises in a Virtual Programming Course. Proceedings of the 34th Frontiers in Education Conference (FIE2004), Savannah, Georgia, USA, October 20-23, Available in URL: http://fie.engrng.pitt.edu/fie2004/papers/1296.pdf (Accessed April 26, 2005).

Metsämuuronen, J. 2001. Laadullisen tutkimuksen perusteet , metodologia-sarja 4 (Basis for the qualitative research). Viro.

Miraftabi, R. 2001. *Intelligent Agents in Program Visualizations: A Case Study With Seal.* Proceedings of the First Program Visualization Workshop *(Ed. Sutinen, E.). University of Joensuu, Joensuu, 53-58.*

Moreno, A. 2004. The Design and Implementation of Intermediate Codes for Software Visualization. Master's thesis, University of Joensuu, Department of Computer Science.

Moreno, A. & Myller, N. 2003. In the Proceedings of International Conference on Networked e-learning for European Universities (Granada, Spain)

Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. 2004. Visualizing Programs with Jeliot 3. In Proceedings of the International Working Conference on Advanced Visual Interfaces, Gallipoli, Italy.

Myller, N. 2004. The Fundamental Design Issues of Jeliot 3. Master's thesis, University of Joensuu, Department of Computer Science.

Mäkelä, K. 1990. Kvalitatiivisen analyysin arviointiperusteet (Criteria of evaluation for the qualitative anlysis). In Mäkelä (ed.) Kvalitatiivisen aineiston analyysi ja tulkinta (Analysis and interpretation of the qualitative data). Helsinki: Gaudeamus.

Niermeyer, R. & Seyffert, M. 2004. Motivaatio. (Original book: Rudolf Haufe Verlag GmbH & Co. 2002. Motivation) Oy Rastor Ab: Helsinki.

Petre, M. 1995. Why looking isn't always seeing. Readership skills and graphical programming. Communications of the ACM 38 (6). 33-44.

Petre, M., Blackwell, A. & Green, T. 1998. *Cognitive Questions in Software Visualization.* Software Visualization *(Ed. Stasko, J., Domingue, M., Brown, M., Price, B.). MIT Press, Massachusetts, 453-480.*

Petre, M., & Green, T. R. G. 1993. Learning to read graphics: some evidence that 'seeing' an information display is an acquired skill. Journal of Visual Languages and Computing, 4, 55–70.

Price, B. A., Baecker, R. M. & Small, I. S. 1993. A Principled Taxonomy of Software Visualization. Journal of Visual Languages & Computing 4 (3), 211–266.

Ruohotie, P. 1998. Motivaatio, tahto ja oppiminen (Motivation, will and learning). Oy Edita Ab: Helsinki.

Stasko, J., Badre, A. & Lewis, C. 1993. Do algorithm animations assist learning: An empirical study and analysis. *The proceedings of the INERCHI '93 Conference on Human Factors in Computing Systems.* Amsterdam. 61-66.

Sutinen, E., Tarhio, J., Lahtinen S-P., Tuovinen A-P., Rautama E. & Meisalo V. 1997. Eliot - an Algorithm Animation Environment. Report A-1997-4, Department of Computer Science, University of Helsinki, Helsinki, Finland, http://www.cs.helsinki.fi/TR/A-1997/4/A-1997-4.ps.gz (Accessed April 10, 2005).

Sutinen, E., Tarhio, J. & Teräsvirta, T. 2003. Easy Algorithm Animation on the Web. Multimedia Tools and Applications 19 (2), 179–184.

Syrjälä, L., Ahonen, S. Syrjäläinen, E. & Saari, S. 1996. 3rd edition. Laadullisen tutkimuksen työtapoja (Working habits of qualitative research). Helsinki: Kirjayhtymä.

Teräsvirta, T. 1996. Eliot. http://www.cs.helsinki.fi/research/aaps/Eliot/Eliot.fi.html (Accessed April 19, 2005)

Teräsvirta, T. 1997. Jeliot. http://cs.joensuu.fi/~jeliot/www/ (Accessed April 19, 2005)

The basis of the curriculum for elementary school. 2004. (Perusopetuksen opetussuunnitelman perusteet). Helsinki: Opetushallitus. http://www.oph.fi/info/ops/pops_web.pdf (Accessed May 27, 2005)

The basis of the curriculum for high school. 2003. (Lukion opetussuunnitelman perusteet). Helsinki: Opetushallitus. http://www.edu.fi/julkaisut/maaraykset/ops/lops_uusi.pdf (Accessed May 27, 2005)

Torvinen, S. 2001. Ohjelmoinnin perusteiden oppimisen ongelmia virtuaalisessa oppimisympäristössä (Problems of learning the basics of programming in virtual learning environment). Master's thesis, University of Joensuu, Department of Computer Science.

Tuomi, J. & Sarajärvi, A. 2002. Laadullinen tutkimus ja sisällön analyysi (Qualitative research and content analysis). Helsinki: Tammi.

Varto, J. 1992. Laadullisen tutkimuksen metodologia (Methodology of qualitative research). Helsinki: Kirjayhtymä.

ViSCoS. http://www.cs.joensuu.fi/viscos/ (Accessed August 31, 2004)

Waisel, L. B., Wallace, W. A. & Willemain, T. R. 1999. Visualizing Modeling Heuristics: An Exploratory Study. Proceedings of the 1999 International Conference on Information Systems. Charlotte, North Carolina, 166-177.

Yin, R. 1983. Case research. Design and Methods. Applied Social Research Methods series vol 5. London: Sage.

Appendix 1

The exercises

1st week

- Make class Suorakulmio which counts the area of the rectangle. Program ask from the user the width and the lenght of the rectangle and prints out the area.

- Make class Valot which changes the light in traffic lights. Program asks does the user wants to light the upper or the lower light. If the red light lights and user wants to light the upper light then green light lights. If green light lights and user wants to light the lower light then red light lights. Every time when the light is changed the program prints out which colour is lighted. User tells when program ends (while or do while loop).

2nd week

- Make class Ala which can increase and decrease the length and the width of rectangle. Program tests that the class Ala works.

- Make class Laskin which has operations +, -, * and /. Program asks two doubles and the operation and prints out the result. User tells when program ends (switch case).

3rd Week

- In the material is Henkilo class. Extend from that class class Johtaja which has attributes number_of_employees and what_leeds, class Unemployee which has attributes education and previous_job and class Pensioner which has attributes age and hobbies. Make program which tests all classes.

4th week

- Make class Taulukko which counts average and frequency from the integer array. Program asks from the user 6 numbers between 1 and 10 and stores them into an array. It also prints out the average and the frequency.

- Program has ten first names and ten last names in the name database. User types one last name and class Nimitietokanta prints out all the first names and the last names which has the same last name that user typed.