Teaching Programming to Liberal Arts Students — a Narrative Media Approach

Peter Bøgh Andersen*, Jens Bennedsen, Steffen Brandorff+, Michael E. Caspersen and Jesper Mosegaard

*Department of Computer Science
Aalborg University, Denmark
pba@cs.auc.dk

+Department of Information Studies
University of Aarhus, Denmark
sbrand@imv.au.dk

Department of Computer Science,
University of Aarhus, Denmark
{jbb, mec, mosegard}
@daimi.au.dk

ABSTRACT

In this paper we present a new learning environment to be used in an introductory programming course for students that are non-majors in computer science, more precisely for multimedia students with a liberal arts background.

Media-oriented programming adds new requirements to the craft of programming (e.g. aesthetic and communicative).

We argue that multimedia students with a liberal arts background need programming competences because programmability is the defining characteristic of the computer medium. We compare programming with the creation of traditional media products and identify two important differences which give rise to extra competences needed by multimedia designers as opposed to traditional media product designers. We analyze the development process of multimedia products in order to incorporate this in the learning process, and based on this we present our vision for a new learning environment for an introductory programming course for multimedia students.

We have designed a learning environment called *Lingoland* with the new skills of media programming in mind that hopefully can help alleviate the problems we have experienced in teaching programming to liberal arts students.

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computers and education– *Computer science education, Information systems education*

ITiCSE'03, ***June 30–July 2, 2003, Thessaloniki, Greece.***

General Terms

Human Factors.

Keywords

non-majors, narration, literal arts students, introductory programming, programming education.

1. INTRODUCTION

Teaching introductory programming to non-computer-science students and in particular to multimedia students with a liberal arts background is a big challenge for several reasons (an in-depth discussion of this issue can be found in [4] and [2]). Programming is not a primary interest of the students and many students consider programming to be "nerdy". Most liberal arts students are more inclined to "open-ended topics" in which analysis, discussion and interpretation are core competencies, and are less inclined to take interest in "closed, absolute topics" like math and programming. Almost all students are lacking in mathematical qualifications, or even worse: many are scared of math and typically have had very bad school experience in that subject. Consequently, most students are de-motivated already at the outset and do not possess the habits and qualifications that "normal" CS students do. For these and other reasons it is necessary to approach the task of teaching introductory programming to liberal arts students in a new and untraditional way.

Our approach is based on an innovative learning environment, Lingoland, that is used in the first weeks of learning the basics of programming in Lingo (the built-in language of Macromedia's media authoring tool, Director). This paper is a presentation of our motivation for developing Lingoland, the vision for it, as well as a brief description of the product as it has materialized so far. (It is difficult to pay justice to an interactive media like Lingoland in a paper; we therefore invite the interested reader to visit www.daimi.au.dk/lingoland/ for a more thorough presentation of the tool.)

2. WHY MULTIMEDIA STUDENTS NEED PROGRAMMING

Given that teaching programming to multimedia students is a big challenge, and that learning programming skills is very frustrating, one might consider whether programming skills are at all necessary for this group of students.

We believe it to be very important (as did Alan Kay and Adele Goldberg when they invented Smalltalk [5]); the main argument is the following [1]: for the past two hundreds years, two cultures of research have differentiated themselves. On one hand, Science sought the laws of nature and put them to use in building tools and machines. On the other hand, the Liberal Arts concerned themselves with human culture and art, and applied their knowledge to analyzing and producing media objects, such as books, plays, movies, television, and newspapers. But with the advent of the computer, something came into the world that was a tool, a machine and a medium at the same time. This fact necessitates a renegotiation of established knowledge borders. Qua machine and medium, the computer requires from its user both types of qualifications, and therefore the liberal arts need to take an interest in utilizing its characteristic features.

The basic characteristic of the computer medium is that it can be programmed. This feature distinguishes the computer from other media, and enables the interactivity that most people recognize as the unique new feature of this medium.

If multimedia students were allowed to study a curriculum without programming, they would miss the defining characteristic of one of the media of interest, which again would reduce their analyses and products concerning the computer to replicas of old media works.

3. COMPETENCES FOR CREATION OF INTERACTIVE MEDIA PRODUCTS

When we compare programming to the creation of traditional media products, we will focus on two important differences:

1. *Direct vs. indirect creation*: In many traditional media (literature, graphic art, etc.) the product is created directly by the designer. In programming, the product—the program execution—is created indirectly through the program which is a description of (an infinite number of) possible program executions. Programmers create tools rather than works.
2. *The temporal aspect:* In some media, the designer has full control over time. This is not the case in interactive computer systems in which the user determines part of the execution.

The creation of some media products (e.g. movie scripts, drama, and music compositions) differ in this respect and are similar to programming in respect to indirect creation. In these art forms the author/composer will have to keep in mind how his instructions in the manuscript or score will be understood by the actors or performers.

A programmer does not produce a program execution directly, but must plan and write a *description* of the execution. Thus, programming involves a more *analytical* and *less directly involved* stance than painting or writing.

The outcome of a programming effort is not a single tangible "thing", but rather a (possibly infinite) set of "execution instances", each different from the other, but all based on the description of the execution. Thus, the outcome of programming can be considered as more *abstract* than the result of e.g. painting.

These differences are caused by the inherent properties of the computer media, and there is no escape: the students must learn to handle them if they want to stay in the game.

However, this does not mean that pedagogical methods and curricula should be transferred wholesale from the computer science traditions to interactive media. Most programming curricula evolved in an earlier period in which computer systems were automata or tools. In such applications, predictability, robustness, and correctness are important properties. In media applications, however, new requirements are *added*: like other media, they must be understandable, entertaining, aesthetically satisfying, educational, provocative, etc. Although the traditional media skills, such as writing good texts or drawing instructive pictures, are still important, programming skills are just as indispensable, *since they allow you to exploit the special powers of computers to communicate your message.*

4. THE PROCESS OF DEVELOPING MEDIA PRODUCTS

The main function of a medium is to enable its user to create attractive interpretations of the representations it carries. The value of a media product lies in what it can mean to somebody. If we cannot understand the subject of a textbook we have bought, we have wasted our money; if the newspaper does not give us interesting news we have reason for complaint; and if the horror movie does not scare us we want our money back.

The process of writing a book resembles the development process for interactive media products and gives clues to what we want to obtain in the learning process. A writer moves phrases around and exchanges words in his manuscript until he hits upon the formulation that best expresses his thought. Similarly a painter keeps repainting and correcting until the image composition is as he envisioned. This is a basic technique in media work, called *commutation* in linguistics: replace a property of a sign by another property and observe the difference of meaning caused by the change. Repeat until satisfied.

It follows that commutation could play an important part in a media-oriented programming course, not only because it is a good pedagogical trick, but also because it is a basic professional technique the students must master.

Consequently we have to modify the traditional computer science method, and object-orientation in particular, of first building a model of the domain of interest, then devising the functionality required by the application area, and finally adding a suitable interface. Otherwise we risk constructing a product e.g. a story that nobody will take any interest in at all! The building of the model, the definition of functionality, and the design of the interface must go hand in hand.

Of course, not all aspects of an interactive media product can be written with this direct link to the interface. A sculptor must construct a steel scaffold that can maintain his sculpture; likewise, the programmer will have to construct an architecture that indirectly is a prerequisite for the effects that impact the user.

5. LINGOLAND – THE VISION

Theorists like Niklas Luhmann (cf. [8]) view communication as a "perturbation" of the already established closed system of the learner. What the teacher can do is to challenge, "irritate", the student's system; however it is the student himself that must adapt to the irritation. A consequence of this theory is the following: if you can present a programming environment that the student can understand through his current set of competences, and you are able to challenge these competences, this may cause the students to change their preconceived notions about a subject. This is another way of stating the old rule of "talking to students in a language they know". Liberal arts students often possess a number of the qualifications involved in problem solving at a general level, but most often have no explicit knowledge hereof. If we choose a metaphor already known to them, we could be halfway.

If one wants to learn a new foreign language, say German, the most efficient approach is to spend some time among Germans. We are not arguing that learning a programming language is like learning a natural language, but we find the metaphor of spending time among native speaking inhabitants when learning a language useful also when it comes to learning a programming language.

Like all metaphors, the metaphor of viewing programming education as learning a new natural language breaks down at some point. In order to support the metaphor from the outset, the programming language is presented as the language of mechanical creatures in a fictive world. Hopefully the students will then accept the primitive and strange way these machine-born creatures communicate.

In a media-oriented approach, learning a programming language means to understand the meaning effects it can offer to produce! What happens on the screen or in the loudspeakers if I change this line of code to something else? Does it change my story? Does it express what I want to express?

We envisage a learning tool, Lingoland, to support the learning of the programming language Lingo. The environment is designed to also meet the requirements of the other part of the curriculum, media design. It builds on the game-and-story metaphor. We have staged the learning process as a game where we have left various lacunae open for the students to discover and enhance. The student simply edits scripts and watches the resulting behavior changes, switching back and forth between the so-called play mode (execution) and transform mode (scripting) of the tool.

Moreover, Lingoland tries to move the students from the game illusion they know into the world of programming, from the "interface" to the "system". In the user-interface of the environment we graphically make explicit the change from "playing the game" to "programming the game" (transforming the rules of the game). In this respect it exploits the notion of "Verfremdung"[1] in Brecht's dramaturgy [3].

The programming challenges unfold when the student wanders around in the fictive world. All the inhabitants you meet "speak" and understand Lingo. You learn by observing the way they behave, and you communicate with them in their own language. In this respect Lingoland builds on Papert's well-known ideas of creating an abstract world of mathematics [6].

One of the ways programming languages differ from natural languages is the required degree of formality and precision when communicating in the language. Through their experience and education students with a liberal arts background have developed and refined a series of competences used in discussion, communication, and life in general. Such competences have a rich set of tools to deal with inaccuracies, incompleteness, and errors but still works satisfactorily in a great number of contexts. The flexibility of this relies on the experience of the individual, based on an ability to generalize and transfer conceptual structures. In contrast programming requires precision and rigid formal expressivity, as the recipient of the communication is a machine. The computer does not possess even a minimum of the customary social competences, implicitly expected from any communication partner by the computer novice. On the contrary, interpreters or compilers mercilessly track down even the smallest of syntax errors and produce disillusion, blocking the progress of the learner. The creatures of Lingoland are not human and rather unfriendly to "non-Lingoists" (i.e. the students) so students are forced to be precise in the way that they express their communication. Hopefully, this motivates formal and precise communication.

The best teaching environment is one that itself demonstrates what is being taught, i.e. the environment should itself be a good media product that stages the learning process. There are two aspects of being a good multimedia product, an external and an internal. From an external point of view, the product must be understandable, entertaining, aesthetically satisfying, educational, provocative, etc. From an internal point of view the product must posses traditional software engineering qualities such as modularity, low coupling and high cohesion, etc. Good quality is hopefully inspiring to the students, but more importantly it is necessary when demonstrating the system architecture: how interface functionality and model works together.

6. LINGOLAND – THE PRODUCT

Lingoland is a game where you control a person walking around and your mission is to rescue the world from various evil viruses that have infected all the other inhabitants. All inhabitants are products from a local software factory, and the student's final goal is to locate this plant and correct their production; only then, Lingoland will again become the peaceful place it once was. During the mission you are given a number of quests that have to be solved in order to proceed and fulfil the mission.

A typical quest in the beginning is "There is going to be a party for which we need some water, and agent Snegom is going to get the water at the well and bring it to the barn, where the party will take place. Snegom walks too far; you must change his behaviour so that he only walks 50 steps back and forth". The mission is:

[1] "Verfremdung" (or estrangement) means to historicize, that is, consider people and incidents as historically conditioned and transitory. The spectator will no longer see the characters on stage as unalterable, uninfluencable, helplessly delivered over to their fate. He will see that his man is such and such, because circumstances are such. And circumstances are such, because man is such. But he in turn is conceivable not only as he is now, but also as he might be –that is, otherwise– and the same holds true for circumstances. Hence, the spectator obtains a new attitude in the theatre. He will be received in the theatre as the great "transformer", who can intervene in the natural processes and the social processes, and who no longer accepts the world but masters it.

1. find the agent Snegom
2. look at the Lingo-code that defines his behaviour
3. change the code to make Snegom do the right thing
4. verify that the result is as desired.

Once Snegom is found, the student exits the play mode and enters the transform mode. In transform mode all the active objects in the game gets assigned a name. The student can edit the behaviour of the desired object (Snegom) by invoking an edit method on the edit tool and giving Snegom as a parameter. This helps learning the student about method calls and parameter passing.

When the edit method is invoked, the code for Snegom is shown in a little box:

```
if distance>100 then
  ...
  distance=0
end if
distance=distance+1
```

This piece of code contains an if-statement that needs to be altered in order to obtain the desired behaviour.

When returning to play mode, the behaviour of Snegom changes right away and there is an immediate feedback to the students (in case of a syntax error, the student is informed about it by the agent and help is provided).

Later in the game the same topic is re-addressed. This time the quest could be: "During the party we want to have some nice flashing lights, but the fire-fly won't flash. Find it and make it flash, then we will have a nice party". The student does the same again (find the relevant object, change the code that defines the behaviour of the object, and observe the result), but this time there is no if-statement at the beginning, the student has to figure it out by himself.

Later again, the quests are "phrased" in Lingo, and as the quests get harder, the student has to use more advanced problem solving techniques to modify and correct the creatures of Lingoland. In this way the student is introduced to the constructs of the programming language; in the beginning the tasks are fairly easy, but gradually they become harder and harder, and over time the student improves his expressability in the language.

All the elements of Lingoland are written in Lingo. This implies that the student can inspect the way they are written and learn from this. This also gives him the idea that programming can be used for writing educational software and games and hence motivates the need for learning programming.

Lingoland has a strict object-oriented user interface where all interaction takes place through method invocations on objects. Even the editor that is used to transform the Lingoland objects is an object and the interaction with it reflects that. In this way the students learns to solve a problem by finding (and, eventually, creating) suitable objects with relevant methods for solving the problem.

To support the "Verfremdung" (see §5) that challenges students not only to play but also to think, we clearly signal a mode change from play mode (see figure 1) to transform mode (transforming/creating behaviours in Lingo – see figure 2). When you exit the game and enter transform mode, a semi-transparent lid slides over the Lingoland world and presents the world as contained in a machine-like device. The fictive world turns out to be an illusion running on a machine. The student can still see the world through the lid but is no longer a participant – now he is creating or modifying it.

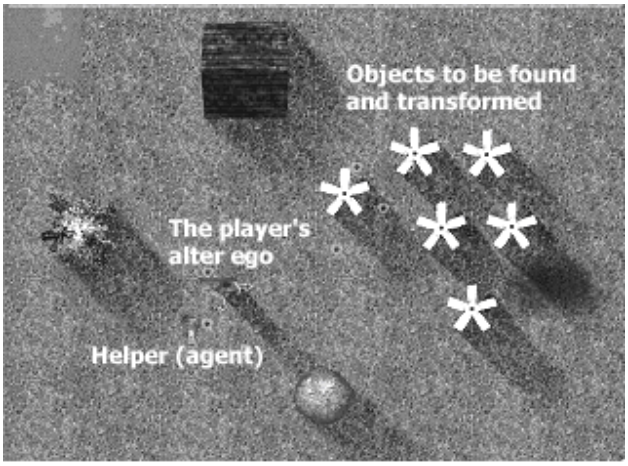

Figure 1: Lingoland (play mode)

The product is designed as a framework. In the framework there are two aspects: the game-story and the learning objectives, and in the system architecture these two aspects are clearly separated. The story and quests are generated on-the-fly according to the learning objectives defined by the teacher and the progress of the student. In the framework the teacher define learning objectives and how these objectives can be made concrete in Lingoland by quests. This implies that different pedagogies can be used. In the example above a spiral approach is use, but it is up to the designer of the story and learning objectives to make the decision. Further description of the framework design and the consequences for the learning environment will be addressed in another paper.

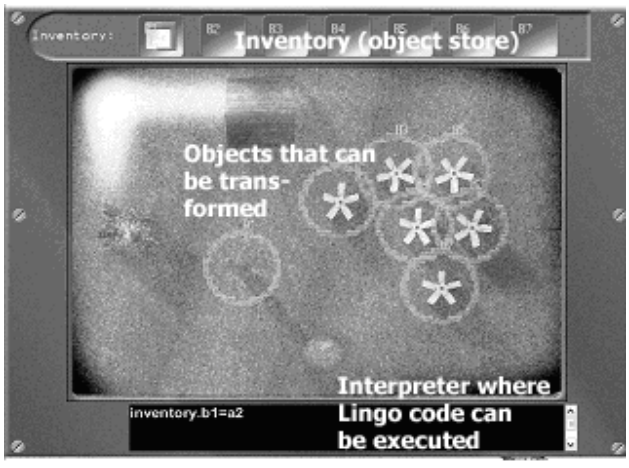

Figure 2: Lingoland (transform mode)

7. CONCLUDING REMARKS

Other approaches and tools address the task of introducing students to programming through a programmable virtual world (e.g. Turtle World [6] and Karel the Robot [7]). However, our ap-

proach differs radically in at least five respects: 1) it is a learning environment created as a professional multimedia production using different senses, 2) the notion of "Verfremdung" is used explicitly to differentiate and integrate the play mode and the transform mode of the tool, 3) the learning model is explicit in the tool, and it is changeable, 4) the internal design of the tool is created in such a way that it is suitable for inspection and modification by the students later in their learning process, and 5) the tool represents in every aspect an outstanding example of the kind of multimedia productions the students should be able to create when they graduate.

We suggest a narrative as the framework for learning good programming practices as well as a specific language (e.g. Lingo). A popular kind of narrative among students is the computer game, among which we have chosen the genre *adventure game.* By letting all learning activity relate to the fictive world, "Lingoland", and to communication with its inhabitants, the "Lingolanders", we hope to ignite the initial spark of interest of the student. We also hope to keep down frustrations of the student by three main ingredients:

1) a reasonably interesting learning environment without the hassles and hazards of most compilers and programming frameworks, which in itself demonstrates the type of media applications the students are expected to produce;

2) smooth and quick transitions between the play and transform modes gives the student immediate feed-back to every programming effort and supports *commutation* as a design technique;

3) a "spiral" learning pattern with a bearable learning curve requiring the student to learn bit by bit.

These are the main reasons we have hopes for Lingoland as a learning environment for programming courses offered to multimedia students with a liberal arts background.

Although Lingoland is not fully completed, the environment has been used with success for two years in an introductory object-oriented programming course for multimedia students. Also, the environment has been tested with a group of multimedia educators from community colleges. The reaction from both groups was very positive and encouraging. A more formal evaluation of the environment is planned.

Lingoland and various documents related to the system is available on the web: www.daimi.au.dk/lingoland/.

8. ACKNOWLEDGEMENTS

We thank IT University West for financial support.

References

[1] Andersen, P. Bøgh (forthcoming). Acting Machines. To appear in Gunnar Liestøl et al. (eds.): *Innovations – Media, Methods and Theories*. Cambr., Mass: MIT press.

[2] Bennedsen, J., *Teaching Java To Liberal Arts Students*, Java & the Internet in the Computing Curriculum Conference Proceedings 7, 2003

[3] Brecht, B. (1960). *Om Tidens Teater (Schriften zum Theater).* Gyldendal: Copenhagen. In danish.

[4] Guzdial, M. & E. Soloway (2002). *Teaching the Nintendo Generation to Program.* Communication of the ACM (4), 2002, pp. 17-21.

[5] Kay, A. & A. Goldberg (1977). *Personal dynamic media*, IEEE Computer (3), 1977, pp. 31-41.

[6] Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books: New York.

[7] Pattis, R. E (1995), *Karel the Robot*, John Wiley & Sons, Inc. 0-471-59725-2

[8] Qvortrup, L. (1998). *Det hyperkomplekse samfund* (The hyper complex society). Gyldendal: Copenhagen. In danish.