Turtlet – Motivating Students with Visualization

Jussi Kasurinen
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, Finland
+358 5 621 2860
jussi.kasurinen@lut.fi

Uolevi Nikula
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, Finland
+358 5 621 2839
uolevi.nikula@lut.fi

Mika Purmonen
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Lappeenranta, Finland
+358 5 621 2860
purmonen@lut.fi

ABSTRACT

The fundamentals of programming are a field that extensively uses different kinds of tools to enhance learning experience. These tools come in several sizes, offering wide range of different services or approaches to the teaching of introductory programming curricula. Even the basic taxonomy for CSE tools classifies over fifty different tools and software packages.

Even still, it is unfortunate that at the same time, computer science and programming in particular, suffer from high drop-out rates and falling student grades. Students seem to be disinterest on programming because of several new concepts and structures have to be learned before anything visually impressive can be created. This problem is intensified by the new multimedia environments such as video games and applets, as they provide instant access to exciting and impressive possibilities. On the other hand, the "old school" programming is required to acquire experience in general programming. So could there be tools, which could address both the student motivation and still preserve the elements of "real" programming?

This paper describes a visualization tool creation project to demonstrate programming concepts on lecture environments, with ability to be used as a programming assignment in the course laboratory exercises. Further in paper we present the design principles and technical structure of the application. We also present some results from applying the tool to the course contents, and discuss the meaning of collected results and observations.

Keywords

Python, Fundamentals, Programming, Motivation, Visualization

1. INTRODUCTION

The fundamentals of programming and computer science in general are an area that actively develops and employs different teaching applications. In fact, Kelleher and Pausch [7] taxonomy defines and introduces over 50 different teaching applications, focusing on different aspects of learning programming. These tools also use several different approaches to learning: while smallest tools may include only additional libraries, which must be embedded to the source code, largest programs overtake entire programming environment offering complete compiler and debugging tools. There are even complete commercial game titles, which can be considered computer science education tools.

One of the subgroups of computer science education is visual programming tools [7]. These tools apply animation, visual hints, even sound to support student learning and ease code creation process. This approach capitalizes the concept of approaching different learning methods and enhancing motivation and interest towards learning programming concepts. In some cases these objectives are approached without any actual programming language, but employing different visual tools and modular meta-programming [7].

However, learning to actually program this way is not axiomatic. Usually novice programmers understand programming as series of commands or cluster of separate functions, while the experts create a mental models and approaches based on the accumulated experience and compatible, together associable structures [2,9]. This experience, which is required to create and implement programming models, or feasible approaches to programming problems, is gained by designing "real" functional code, not by following presentations or visual tools [15]. However, the positive results on learning programming concepts with visual programming tools and pre-programming methods [10] cannot be disputed.

On the other hand, computer science and programming courses suffer from high drop-out rates and falling student grades. For example, Guzdial and Soloway [4] offer an explanation for this behavior: students lose interest on programming because of several complex models and structures have to be learned before anything visually impressive can be created. The students can memorize the constructs, but without motivation for doing so the result are usually less than satisfactory. As video gaming and Internet has become popular, offering multimedia and interactivity, the technologically oriented programming – data manipulation - is no longer interesting or motivating for the students. In short, simple command line outputs and data filtering are not exiting, and students require more motivational and imposing tasks to upkeep the interest towards programming.

To respond to these issues in our introductory programming course, a decision to search for suitable tool was coined. Results from the review on existing software tools were underwhelming, as Python as a programming language did not have many visualization tool options available. Additionally, we had just revised our programming courses successfully [5] and decided against changing the programming language or teaching approach again. Therefore it was decided that we had to develop the tool ourselves. Our objective for development of a new visualization tool was to address two issues with one Python-based tool. Firstly, it should offer demonstration capabilities for lecturer to

introduce programming concepts like logical structures and iteration, but also enable students to exercise real programming by extending the command base with new visualization operations.

Rest of the paper is constructed as follows: chapter 2 describes the design and development process for the tool and chapter 3 describes the tool structure. Chapter 4 focuses on student response and results regarding the tool, whereas chapter 5 discusses the development process and future concept. Finally, chapter 6 closes the paper with conclusions.

2. DEVELOPMENT PROCESS

The original concept for the tool came from the literature review and specifically LOGO [8] or Karel the Robot [11], teaching languages focused on visual presentation of structures, primarily known from their applications of "turtle graphics". The existing Python translation of Karel the Robot called Guido van Robot [3] was rejected because it used simplified Python syntax which was deemed unnecessary distraction and also disallowed simple extension of command base beyond predefined basic operations. As our initial concept was to create visual demonstrations for Python-based programming course, we also wanted to use the tool for duration of the course, not just for the first few weeks.

Initial concept of using only standard library tools did not work as intended as the designed functionalities required better tools for drawing graphics. Therefore Python Imaging Library [13] was included to the tool library for visualization module in addition to the Tkinter interface, which is build-in standard for GUI building in Python. This however, meant that the software had to be distributed either bundled with build-in interpreter, or include additional documentation for installation and troubleshooting.

The idea was first examined with proof-of-concept prototypes, which were developed rapidly prototyping different functionalities over the duration of few days. These initial prototypes were then used to draft requirements for the final product, which was developed over few months in summer 2007 as a Bachelor's Thesis [12] for the department. The development process was overseen by the course lecturer, who would be the designated main user for demonstration tool.

As the demonstration tool was produced to a stage where it could be tested as a full prototype, the extension for student activity was considered feasible option and included to the functional elements. The primary concept for the exercising interface was to enable the students to write visualization extensions with real, functional Python source code. The student-created command module was automatically detected and included to the command base if required files were present at the tool startup. Because the target students were first year students with no prior knowledge on programming, this design was chosen as it was considered user friendly enough

3. ARTEFACT

3.1 Visualization tool

As the figure 1 shows, the visualization tool consists of four core modules. These modules are the main components of the tool, and each of them has specific tasks. The GUI module operates the user interface, which is illustrated in figures 2, 3 and 5. Control module operates the internal mechanics, delivering the GUI events to the command module, which then interprets the commands to the visualization module, which updates the map. These modules are mandatory for the tool, and used when tool is used as a visualization aid in lectures.

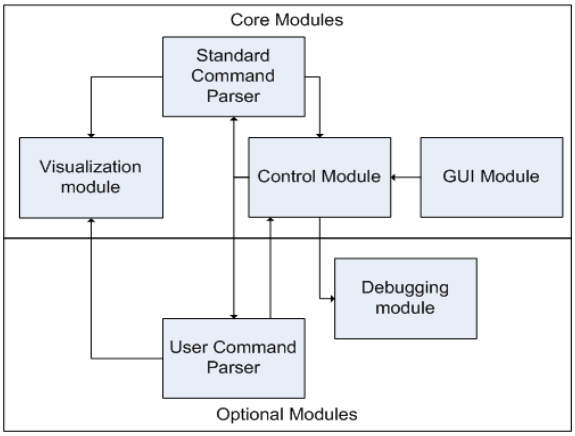

Figure 1: Turtlet internal structure and module relations.

The module structure is constructed so that only control module has feedback loop to observe system status. The core modules are produced with object-oriented programming paradigm, and distributed in byte-encoded form to enhance tool performance on Python interpreter. The user command parser and debugging module, which the students use and have access to, have module interface that allows procedural programming.

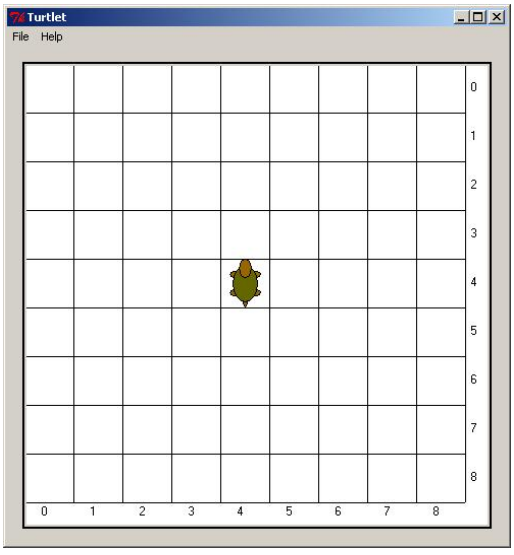

Figure 2: Turtlet map screen.

The user interface consists of two main windows, map screen and command screen, which are represented at the figures 2 and 3. The user interface is similar to other interactive visualization tools like Logo [8] or Karel the Robot [11]. The command screen has also an alternative button interface (figure 3) with simplified command base to allow quick and simple lecture demonstrations.

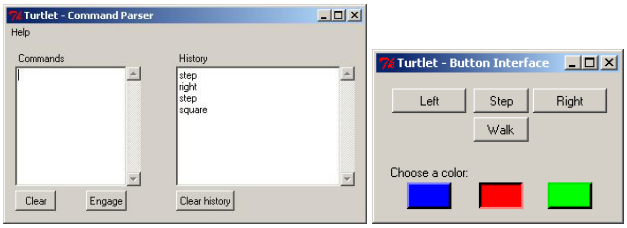

Figure 3: Turtlet command screen in left and button interface in right.

For student programming project, the tool can be extended with additional modules. As the standard commands are deliberately simple and low-level operations, the students can be instructed to create their own commands for more advanced operations. For this activity, the students can override and extend the existing command module by defining which commands control module forwards to the user command parser. Then the user parser instructs the visualization module to complete the command by telling what to draw on the screen. These additions are only required to work with the command parser, as the Tkinter syntax was considered too complicated for the novice students to grasp. However it should be mentioned that the button interface can be extended, should the course in the future include GUI-exercises to the course syllabus. The architecture also allows overlaying of a debugging module to control error handling procedures in a case where user-defined command module is malformed and causes internal errors.

```
import control
import visualization

control.MyCommands = ["square"]

def Parser(command):
    if command == "square":
        for i in range(0,4):
            visualization.TurnLeft()
            for j in range(0,2):
                visualization.TakeStep()
        return command

    else:
        return "Invalid command:"+command
```

Figure 4: User-defined command that draws 2 by 2 step square.

In the figure 4 we can see example of user command parser. The control module has internal MyCommands-list, which includes all commands forwarded to the user parser instead of standard command parser. In the user parser, there is a function called Parser, which takes the given command as a string parameter. Then, the given parameter is analyzed by a series of logical tests to identify the given command and when identified, the visualization module is instructed accordingly. For the advanced exercises, the command parameter may also include additional attributes for the command, like length or color. In this case, the given command “square” produces series of four left-hand turns and two steps, creating a leftward square on the screen. Finally the status is returned to the control module by echoing the command signaling that the command was successful, or returning error message to alert the user.

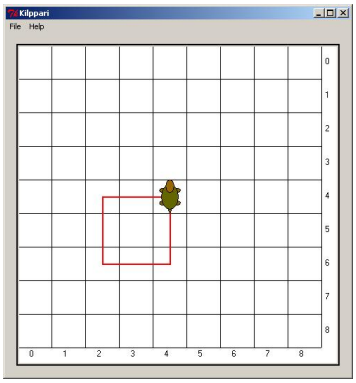

Figure 5: Turtlet map screen after successfully completing student command.

3.2 Submission evaluation tool

Evaluation of student submissions is a time consuming routine task that is necessary but not the most interesting assignment. This problem is also much more severe when using graphical user interface, as the automation, that usually eases the workload, can be sometimes difficult to apply. Since our Fundamentals of Programming course had about 150 students, we found manual evaluation of the weekly exercises so toilsome that we did not want to do it, and rather spent our time developing a separate evaluation tool (Figure 6).

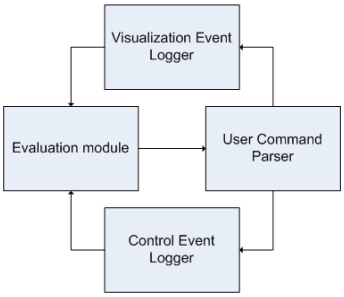

Figure 6: Structure of submission evaluation system.

The evaluation system consists of three elements, which are the evaluation module itself and event loggers for both visualization and control module calls. The system offered the same module interface to the command parser as the actual system, and recorded module calls to visualization and control as strings. Therefore the system simplified the evaluation process from user interface operations to simple string comparisons. For our development project, a prototype of the evaluation tool was successfully embedded in an existing commercial VLE system [16].

Technically the evaluation module triggers a list of predefined command inputs, of which the user-created command module reacts. These reactions are then recorded by the event loggers as data logs for the evaluation module. If the reactions to the commands correspond to the logged events, the submission is accepted. During the tests, the evaluation module also searches the data logs for additional unacceptable conditions, such as malformed implementations of commands or unsuitable order of events. If any errors are found, the evaluation module produces

error message with instruction on how to fix the submitted command parser.

For example, the Square command should result in a data log consisting of 4 turn events and 8 step events, in order of turn-step-step or step-step-turn. Similarly, taking N steps should produce N step events in the log. The required commands were individually implemented to the evaluation module as the student command parser was expected to perform more functions over the time. Obviously in all scenarios, some general rules like no error messages or no reaction to faulty command were also tested in addition to the command-related erroneous conditions.

4. RESULTS

Turtlet was used for the first time to visualize the fundamentals of programming during the academic year of 2007. The tool was first used as a visualization aide for the basic programming structures in the lectures for the first 7 weeks, and then applied to the laboratory exercises. In the exercises, the students created and extended their own command parser in five weekly phases. After these initial phases were complete, the student could start the final programming project, which required implementation of several new commands and command parameters in addition to those created earlier.

The interest and usability of the visualization tool was surveyed at the course week 10 in tool survey, right after the programming project requirements were introduced. 50 students answered to this voluntary survey. Statistically this meant only 35% of the student population, but statistical comparison of student lecture participation records and final grades had no difference compared to the non-answering students, so the survey results were considered representative of the entire student population and therefore analyzed for feedback. The programming project was additionally evaluated in the final survey for the course, to which 82% of the students (91 students) with possibility for final grade answered.

The detailed analysis on student activity and course results [6] indicated mixed opinions regarding the tool. In tool survey, he visualization tool got cautiously positive review as a programming tool. 35% of the students gave positive feedback concerning the interest as a project assignment, with 45% being indifferent. Similarly, 30% thought that Turtlet exercises were interesting with 50% being indifferent. 39% of the students also thought that the ability to create own commands and test them with visualization was interesting feature, whilst 10% of respondents had actually created own commands in addition of course assignments. In general, the results indicated that approximately 35% of the students did like the tool, 45% did not take strong stand on either direction and 20% of the students did not like it for various reasons.

In the final survey, 81% of the students thought that the course project with Turtlet had been useful, with 55% saying that the project was “moderately difficult” and 31% “moderately easy”. As for the usage as a lecture demonstration tool, the results were not so impressive: while 24% found the demonstrations interesting, 33% gave negative response.

Besides statistical data, several students also gave additional comments on Turtlet in the surveys. The cautiously positive feelings and mixed opinions towards the tool were also present at there:

- “The tool is nice, but only for so long…”
- “Turtlet is boring and unpractical for engineering student. I hoped for something more reusable. ”
- “I think that it [Turtlet] is a concrete and well-suited tool for learning programming.”
- “Turtlet was interesting.”
- “The project could be something that offers more room for creativity.”
- “This new project seems much more interesting [compared to the prior years].”

General consensus seemed to be that tool was mostly positive addition. However, some comments mentioned that Turtlet was “unpractical for engineering students” or something similar, leading us to the conclusion that at least our engineering students are not easily impressed with game-like teaching applications. In fact, comparing previous programming knowledge to the opinions regarding Turtlet as a programming tool, there was a definite trend; those students, who had previous programming knowledge thought that the tool exercises were easier, and more importantly, less interesting.

5. DISCUSSION

From technical point of view, the tool was successful and worked as intended. Even though the preliminary testing had been somewhat limited, we only discovered few minor bugs in the tool, and embedding process to the learning environment was completed in timely manner. Some of the support modules had series of problems, but besides that, the software worked without requiring major maintenance or revisions, although debugging module was left out of the initial package and offered as an add-on for those who would need it.

Unforeseen finding was that the students were much more critical against the new system than we expected. Although the prior courses did not apply any tool of this kind, the system was technically sufficient and the distributed program did not have any major flaws, the students still had mixed feelings towards using it. However, there is always some resistance to changes in infrastructure. Additionally, it could be argued that students tend to complain on negative aspects more often than give positive feedback. Therefore we are not too concerned over the lukewarm reviews, as the somewhat positive feedback means that the tool overcame the change resistance.

One of the possible reasons for the negative feedback could be the student-perceived difficulty and target audience for the tool. It could be reasonably expected that some students – especially those with prior knowledge on programming - can view the visualization tool as a toy aimed to the lowest common denominator of students, and may get bored with the lecture demonstrations. As the amount of students with prior programming knowledge (43%) is considerable, and that the engineering students were not impressed with the tool, resulting in an average of 0.3 (in scale of 1-5) grade lower score on interest, this theory seems plausible.

Technically some of the modules were bit underwhelming. For example, the debugging module and evaluation module sometimes misinformed students and caused additional work for the laboratory assistants. Even though the evaluation module did recognize valid submissions, the error feedback was sometimes spotty and in several occasions, criticizes by students for being cryptic. The debugging module had problems finding the real causes for errors, but this was usually caused by the Python interpreter itself. The programming assignments also caused some collateral problems: as the programming tasks had to clearly define the desired student command parser, some of the students felt that the project did not allow creativity in the answers. This requirement also affected the plagiarism detection system in our virtual learning environment, which reported false positive results for the first time since the introduction of the detection tools. Although the student opinions were not entirely positive, none of the comments indicated that the earlier project would have been better. In addition, the more detailed report [6] on the course outcome analyses the collected material further, indicating similar results.

As for the future research on the tool, our results and literature [1,2,10,15 etc…] indicate that there are needs for visual demonstrations and exercises in the first programming course. Obviously, some of the modules need revisions, and the error feedback system needs development. The students also complained that the service did not allow enough creativity, an issue that should be hastily addressed. For example, one concept would be to give Turtlet support to custom maps or operations such as drawing obstacles, values or objects, and command interface to pick up or drop them on the map screen. This way, the exercises could implement more elaborate assignments like algorithm visualization and problem solving with programming.

6. CONCLUSION

Our initial concept was to combine visualization and real programming interface to a Python-based programming tool. Our search for suitable, existing software lead us to develop the tool ourselves. Although the developed end-product worked mostly flawlessly and received cautiously positive reception, there were still some pitfalls which require attention in the future.

Most of all, the students need to have enough to do when using the tool to stay interested. Visual tasks may be more interesting compared to traditional small-scale programming exercises with 25-50 lines of code, but in our case the practical restrictions caused too many functional restrictions, hindering the motivational effect to some degree.

In our opinion, design and implementation of this teaching tool was positive experience, which provided useful content for the fundamental programming course and most importantly, increased the students interested towards programming. The project has so far given us new design ideas to improve Turtlet in the future.

7. REFERENCES

[1] Boada, I. Soler, J., Prados, F., Poch J. A teaching/learning support tool for introductory programming courses. 2004. Proc. 5th International Conference on Information Technology Based Higher Education and Training, pages 604-609, Istanbul, Turkey.

[2] Crosby, M. Stelovsky, J. Subject Differences in the Reading of Computer Algorithms. 1989. *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems*, Elsevier.

[3] Guido van Robot, The Programming Language, http://gvr.sourceforge.net/index.php, retrieved 24.7.2008

[4] Guzdial, M., Soloway, E., 2002. Teaching the Nintendo Generation to Program. Communications of the ACM, Vol 45(4), pages 17-21

[5] Kasurinen, J. Nikula, U. 2007. Revising The First Programming Course – The Second Round. Proc. Reflektori2007 Engineering Education Symposium, pages 92-101, Espoo, Finland.

[6] Kasurinen, J., Purmonen, M. and Nikula, U, 2008. Study on Visualization in Introductory Programming, Accepted for the 20th Annual Psychology of Programming Interest Group Conference, PPIG 2008, Lancaster, UK, 10.-12.9.2008.

[7] Kelleher, Caitlin and Pausch, Randy, 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, Volume 37(2), pages 83 – 137.

[8] Logo Foundation, The http://el.media.mit.edu/Logo-foundation/logo/, retrieved 14.8.2008.

[9] McKetihen, K., Reitman, J.S., Rueter H.H. and Hirtle S.C.. 1981. Knowledge Organization and Skill Differences in Computer Programs. Cognitive Psychology 13, pages 307-325.

[10] Nevalainen, S. Sajaniemi, J. An Experiment on Short-term Effects of Animated versus Static Visualization of Operation on Program Perception. Proc. 2006 International Workshop on Computing Education Research, pages 7-16, Canterbury, UK.

[11] Pattis, R.E. Roberts, J. Stehlik, M. 1997. *Karel the Robot: A Gentle Introduction to the Art of Programming*, 2. edition, Wiley.

[12] Purmonen, M. 2007. Development of Educational Program for Teaching of Introductory Programming (in Finnish) Bachelor's Thesis, Department of Information Technology, Lappeenranta University of Technology.

[13] Python Imaging Library (PIL), http://www.pythonware.com/products/pil/, retrieved 14.8.2008

[14] Python Software Foundation, www.python.org, retrieved 14.8.2008.

[15] Robins A., Rountree J., Rountree N. 2003. Learning and Teaching Programming: A Review and Discussion. Computer Science Education Vol. 13(2), 137-172.

[16] Viope Solutions, www.viope.fi, retrieved 14.8.2008.

Turtlet – Instructions

1. TOOL WEBPAGE

Turtlet currently has a webpage, which may be found at address

http://www2.lut.fi/~jukasuri/Kilppari/

These pages contain all of the current promotional material for the tool, few different download options and a quick tutorial for getting accustomed to creating Turtlet exercises.

Because the tool modules are distributed in a readable form instead of byte-encoded, we have to enforce some limitations on the source package distribution. These limitations have been added to avoid unwanted distribution of the file *LTYToiminta.py* as it has several similarities with our current programming exercises and the project assignment. The download links can be activated with a following user name and password pair:

user name: Koli08

password: CallingReviewers

Otherwise the files are free for distribution. Turtlet source is under the GNU General Public License version 3 or any later version.

2. DOCUMENTATION

Turtlet documentation beyond this document and the dedicated webpage are currently only available in Finnish. This also applies to the source code comments and tool-included help documentation.

If your web browser is unable to play the embedded network video files, the three minute video is also downloadable from the tool webpage. There is a direct link to the video file below the video window.

3. INSTALLATION

Turtlet can be installed with two options, test installation and full installation. The difference between these two methods is that the test installation only supports lecture demonstrations and can be used in a computer that has no Python interpreter. Full installation offers extendable command base – Turtlet exercises – and support for additional modules like debugging-module or submission evaluation tool. Please also observe that you will need the full installation to be able to complete the Turtlet tutorial.

For full installation, you need to have three installation packages, a Python interpreter version 2.5, Python Imaging Library and Numeric Python NumPy –modules. They can be downloaded either from the Turtlet webpage as a one package or accessed from following web pages:

Python 2.5: http://www.python.org

Imaging Library: http://www.pythonware.com/products/pil/

Numpy: http://numpy.scipy.org/

Please observe that you download right versions of the packages, as the modules for example version 2.6 interpreter do not work or even install to the version 2.5. Additionally, please remember to install the interpreter before the modules.

In Windows environment with full installation, Turtlet can be started by either running the *start.cmd* or opening the file *ohjaus.py* in the IDLE editor bundled with the Python interpreter and selecting *Run -> Run Program.*

Test installation starts by double-clicking the *Turtlet.exe.* Additionally, for test installation you do not need any of the above packages, as Turtlet in this installation file is a standalone version.

4. TUTORIAL

The tutorial is available on the tool webpage. In this tutorial we will implement new command on the *omaToiminta.py* module with Python programming.

If you are unfamiliar with the Python programming, there is one unusual aspect that should be mentioned, the indentation. As Python uses indentation to replace parenthesis characters on the source syntax, the programmer has to be careful with handling the code. By using a source code editor that supports Python syntax, in our experience there are only few inconveniences, but do not try to create Python-code with Notepad. Use the interpreter-bundled IDLE-editor instead.

Also as a curious side note, you may notice that the file *omaToiminta.py* includes the definition for several English commands. This is because the English Koli-demonstration version uses omaToiminta-module to implement the English commands of the parser. By removing the omaToiminta.py-file Turtlet reverts back to only Finnish commands.