2009 Sixth International Conference on Information Technology: New Generations

Teaching Object-Oriented Programming with Games

Lu Yan
School of Computer Science, University of Hertfordshire
Hatfield, Hertfordshire AL10 9AB, UK

Abstract

First-year students in CS/IT curriculum are often overwhelmed by the introduction to programming module, which is a mandatory component for the whole study program. In this paper, we discuss the difficulties students may encounter in this module and propose a novel approach to teaching programming to new programmer with games. We reflect on our experiences on making the programming module friendly and improving the success rate among new programmers. We present our learning theory, teaching methodology and assistive software with technical details.

Key Words: object-oriented, programming, game, greenfoot, bluej.

1. Introduction and background

Introduction to programming is a traditional and compulsory module for first-year CS/IT students. Though the content of this module has evolved a great deal over years due to the advance and trend of computing technology as well as the rise and fall of different programming languages, it is a common observation that teaching programming to new programmers is hard. However, since this module is the foundation for advanced level modules, it is always a challenge for the module team to effectively and efficiently deliver it.

Nowadays, object-oriented programming is the mainstream in computing industry, so Java is often offered as the first language for first-year students to learn. Though Java is a modern language without history burden and unnecessarily complex syntax as C++, it still presents a steep learning curve for novice students.

Our students, like the majority of today's students, are well acquainted with the use of computers. Most of them have experiences with computer game, word processing software, search engine, social networking site and instant messenger. However, since CS/IT program has no formal course prerequisites, students are not necessarily acquainted with programming; in fact few of them have prior experiences of programming, nor are those experiences object-oriented.

2. Difficulties from both student's and teacher's views

Learning introduction to programming is hard. It is not mainly due to the lack of prior computing experience, but the lack of problem-solving skills. Computer engineers/scientists usually need to apply logic thinking and problem-solving skills when writing a piece of software. However, students are often not well-trained with those skills in high school, which puts them in a difficult position when enrolled into university studies.

Abstract concepts and thinking may be another difficulty students will face in learning object-oriented programming. Class and object are core concepts in this module, but it is fairly hard to find the equivalence in real life. Students therefore struggle to comprehend even more complex concepts such as inheritance and polymorphism if they cannot grasp the basic class and object thinking way.

Teaching introduction to programming is hard. Introduction to programming is not about abstract object-oriented concepts and practical programming skills alone, but really about the interactions and interweaving of both. So any approach without balancing the two sides will not succeed in practice. Moreover, as this is the first programming module for new students, it is inevitable to include issues from software process, data structure, project management, operating systems and even mathematics, which are definitely not the focus but the background of this module.

978-0-7695-3596-8/09 $25.00 © 2009 IEEE
DOI 10.1109/ITNG.2009.13

Another difficulty in teaching this module arises from keeping students attention and interests throughout the module. Traditionally, programming is taught in a way of creating simple text-based programs. Most of the teachers around my age shall still remember the infamous `HelloWorld` example:

```
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

which was *de facto* first program students would learn from any Java textbook. However, today's students are grown up in an environment where computer are already part of their life. Text-based programs are neither impressive nor familiar to them. Most of them have little experience with command line, but mouse-assisted graphic environments. So, those example programs are not attractive to them, and actually foreign to them.

3. A bit of learning theory

The UK Professional Standards Framework for teaching and supporting learning in higher education [1] sets out a framework and detailed standards to align our teaching practice. The core concept is to be a reflective professional [2-6].

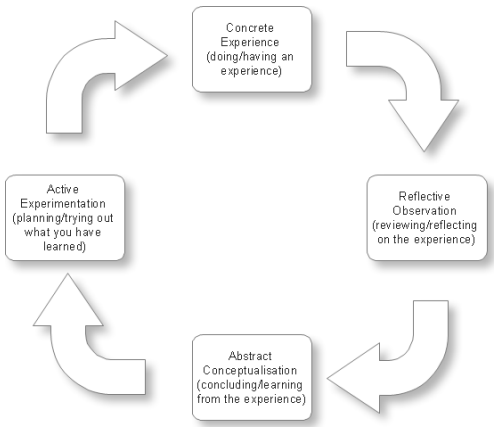

Figure 1. Kolb learning cycle

Reflective practice is important as it enables us to learn from our experiences. Kolb [7] developed a theory of experiential learning called the Kolb Cycle or the Learning Cycle in Figure 1. The cycle comprises four stages: concrete experience, reflective observation, abstract conceptualization, active experimentation; it can be entered at any point but all stages should be followed in a sequence for a successful learning process to take place [8].

The Kolb cycle implies that it is not enough to have an experience to learn, but the reflection after the experience does. It is critical to reflect on the experience to formulate concepts which can be applied to new environments. Finally, the learning will be tested in new situations and new settings. In this way, theory, action, reflection and practice are linked into a dynamic cycle and complement each other [9-12].

We have applied the Kolb cycle as an inspiration and guidance in the introduction to programming module delivery, rather than approaching the problem in a simple and linear way.

4. Our reflection and related works

There are two main aims in the introduction to programming module: object-oriented thinking and Java programming skills. The former, as we believe, can be taught successfully with satisfying results by focusing on fundamental principles of object-orientation, for instance, via metaphors.

A pioneer study by Sims-Knight *et al*. shows that it is possible to teach object-oriented design without programming [19]: students played an extant computer game and then watched an expert demonstrate how to design that game. The expert explained the design considerations in an object-oriented way and students then created their own designs for this game. This approach reinforced our idea on using games in programming teaching.

Zhu *et al*. proposed a way to teach object-oriented programming (C++) by putting methodology first and language second [16]. They proposed to introduce the object concept by observing the real world. This proposal coincides with our approach on using metaphors.

The other aim, Java programming skills, can be taught in a modern, interesting and attractive way. Our philosophy is to use graphics to capture and retain the interest of students (i.e., let students explore programming skills in a graphically rich environment). Besides non-text based examples, simple games could be used as examples. Since most of our students have played some kind of games, they are familiar with the common game rules and no background information on games is needed.

Game maker [20] is an integrated environment for learning object-oriented design. Users can create games without writing a single line of code with game maker's drag-and-drop techniques. This idea has been a major inspiration for us.

Chen *et al*. reported an extensive case study on design and execution of an object-oriented programming laboratory course [15]. In that course, students were required to implement a small-to-medium scale interactive computer game in one semester, making use of a game framework. Similar courses were also reported in [17-18, 21]. We appreciate those approaches to teaching object-oriented programming with games, but we do believe that we should be careful with the selection of games, and don't be too ambitious at the initial point since our students are new programmers.

5. Object-oriented thinking in Greenfoot

We have adopted a GUI-centered approach to teaching object-orientation concepts. Our view is that before letting students to create a trivial application, whether a toy program or a simple game, it is important to ensure students truly understand the basic concepts of object-oriented notions and really manipulate them think in the object-oriented way. We achieved that with an assistive software environment called Greenfoot [13].

5.1. Brief introduction to Greenfoot

The Greenfoot system is a framework and environment to create interactive, simulation-like applications in a two-dimensional plane [13]. Greenfoot allows implemention of and interaction with objects in the context of scenarios.

Once objects are created in Greenfoot, they can interactively be placed into a Greenfoot world, and users can directly play with these objects and invoke methods in those objects.

The Greenfoot system is also an integrated development environment (IDE): it contains an editor, a compiler and a debugger. The underlying runtime and compiler uses standard Java. Greenfoot classes are standard Java classes.

5.2. The wombat scenario

One classic example of teaching object-orientation is the wombat scenario. A Greenfoot version of this is shown in Figure 2. As in this scenario, wombats live in a grid area and move freely with commands. The world also consist some rocks and leaves. When a wombat meets a rock, it will turn to other directions; when a wombat meets a leaf, it will eat that leaf.

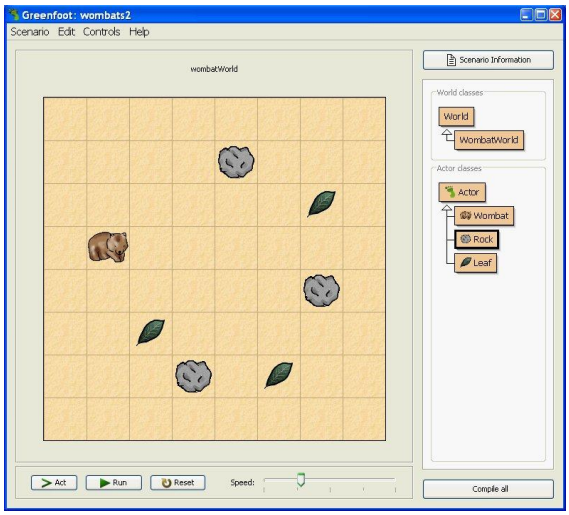

Figure 2. The wombat scenario

When using this scenario, students would typical be given an initial world, which contains an instance of wombat and possibly rocks and leaves.

5.3. Student activities

Students can perform the same activities as the teacher does, but we do encourage students explore this new environment proactively and link those important object-oriented concepts such as object, class and method to this mini-game.

- Observing behaviors: when students use Greenfoot for the first time, they will typically interact with an initial world given by the teacher. For example, the initial view would be similar to Figure 2. Students can press the Run button to observe how the wombat moves, turns and eats leaves.
- Invoking methods: students can then directly play with an individual object. For instance, students will right click the wombat icon and all available methods in the wombat object would show as Figure 3. Students will then try all those methods.
- Instantiating objects: as the next step, students will create a new world by instantiating objects. Students select a constructor by right click a class from the actor list, as shown in Figure 4. The mouse cursor will then show the icon of the object, which can be placed into the world.

After students have instantiated several objects and played the interactions among different objects, the

teacher will then bring up the concept of class and object, and indentify the relationship between class and object.

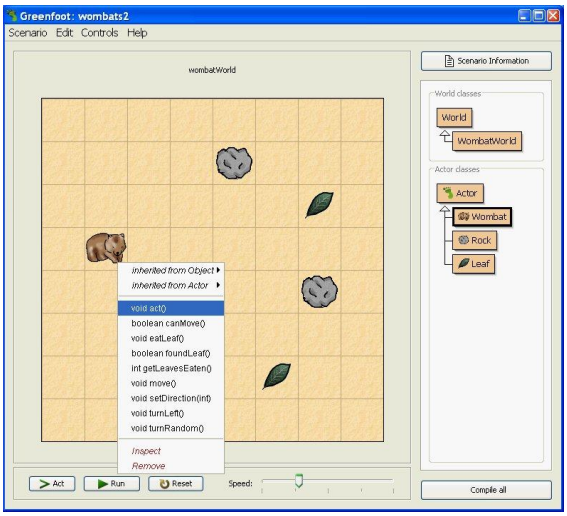

Figure 3. Invoking methods

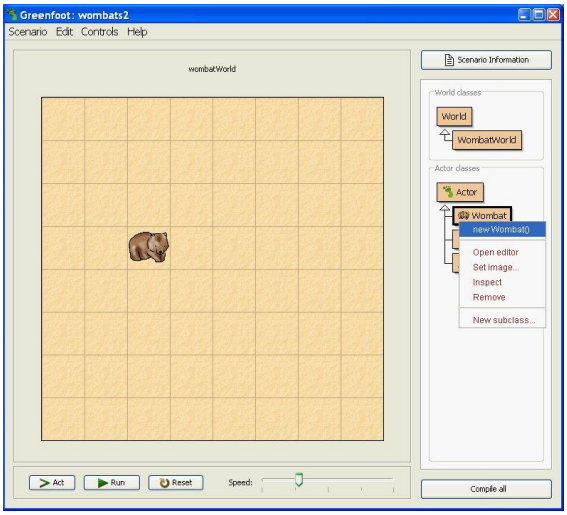

Figure 4. Instantiating objects

6. Java programming in BlueJ

We have selected BlueJ [14] as the IDE to teach students Java programming skills. BlueJ is a programming environment specifically designed for education. Besides GUI-centric design, BlueJ also encourages students to define classes and their relationships with an UML-like notation. We believe it is good for teaching both programming language skills and software engineering principles.

6.1. Brief introduction to BlueJ

BlueJ is a software environment developed to support the learning and teaching of object-oriented programming [14]. It can also be deemed as an ultra-light IDE for small scale Java development, though it lacks many advanced features comparing to NetBeans and Eclipse.

Like Greenfoot [13], students can interactively instantiate and test objects. All objects have a simple representation on an object bench. It is also possible to inspect these objects, and execute their methods.

Another significant advantage of BlueJ is the clear separation of the concepts of classes and objects. Object-oriented concepts such as class, object and method are represented visually and in its interaction design in the interface.

6.2. The picture game

Instead of the monotonic `HelloWorld`, the first Java program in BlueJ is a graphically rich one, the picture game. Figure 5 shows the BlueJ interface, class diagram and object bench of the game. This program displays a picture of house like Figure 6 on the screen, which is built from three basic shapes: circle, square and triangle.

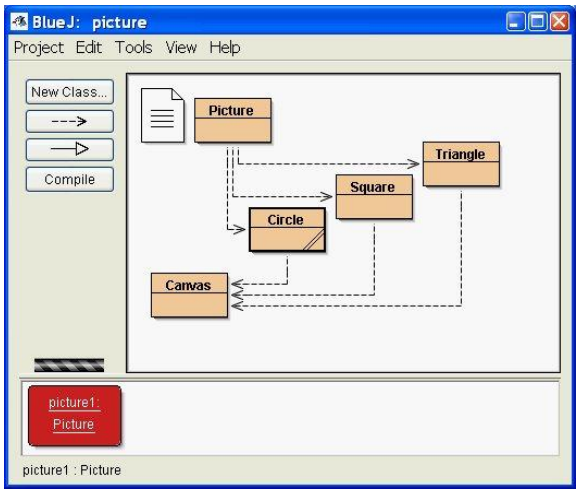

Figure 5. The BlueJ interface

Since students have already played with the Greenfoot environment, they will explore the BlueJ in a very similar way. With the foundation of object-oriented concepts, they usually have no difficulty in understanding class diagram and object bench in Figure 5. Now they are ready for some real programming tasks.

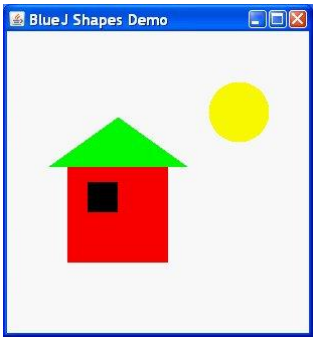

Figure 6. The picture game

6.3. Student activities

As a preliminary warm-up, based on what they have learned in Greenfoot, students are encouraged to explore BlueJ environment proactively and try at least three things:

- Observing behaviors
- Invoking methods
- Instantiating objects

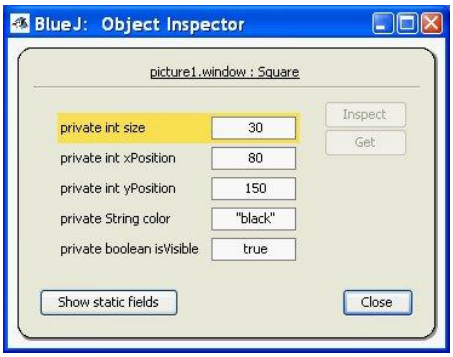

Figure 7. Inspecting objects

The teacher will then instruct them to explore advanced features of BlueJ, and stepwise modify the code of picture to generate a picture of their own design.

- Inspecting objects: students are told to right click an object and select the inspect option from the pop-up menu. An object inspector window like Figure 7 would appear, and complete set of values for the object's fields would be listed. Now the teacher will explain important object-oriented concepts such as field, parameter, property and reference to students.

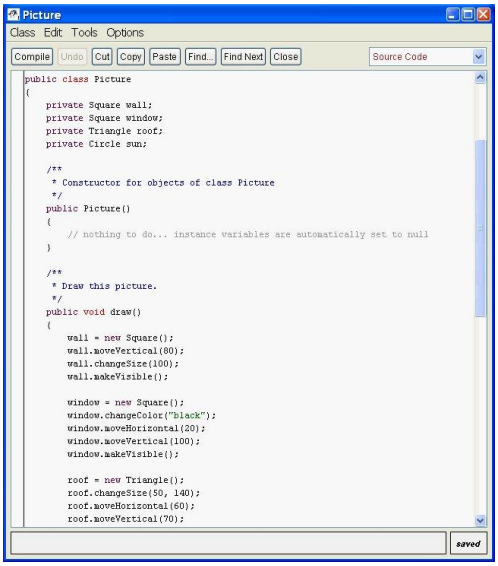

Figure 8. Modifying code

So far we covered how student can play with objects without looking at the source code. The next step is to modify the existing class in picture and learn how to write methods properly.

- Modifying code: students will double-click a class and the integrated editor would pop up with the source code like Figure 8. First, students are required to mimic the code of the picture class, but their resulting image should be significant different than the original one. Then, students are allowed to modify code freely to create more realistic pictures and animations such as sunset.

7. Concluding remarks

We have presented our approach to teaching object-oriented programming with games. This approach may be characterized by (1) object-orientation first, programming second (2) graphically rich examples (3) game as the trigger.

The proposed approach provides a positive and supportive atmosphere in which students can learn the principles of object-oriented programming. It keeps students interested and reinforces those programming principles.

Acknowledgements

This work was part of the CPAD program running at the University of Hertfordshire in 2008.

References

[1] The UK Professional Standards Framework for teaching and supporting learning in higher education, 2006. http://www.heacademy.ac.uk

[2] H. Fry, S. Ketteridge and S. Marshall S. A Handbook for Teaching & Learning in Higher Education: Enhancing Academic Practice (2nd Edition), Routledge, London, 2003.

[3] J. Biggs. Teaching for Quality Learning at University (2nd Edition), Open University Press, Berkshire, 2003.

[4] P. Ramsden. Learning to Teach in Higher Education, Routledge, London, 1995.

[5] G. Gibbs. Learning by Doing: A guide to Teaching and Learning Methods, Oxford Further Education Unit, Oxford Polytechnic, Oxford, 1988.

[6] N. Hatton and D. Smith. Reflection in Teacher Education, in *Teaching and Teacher Education*, vol. 11, pp. 33-49, 1995.

[7] D.A. Kolb. Experiential Learning experience as a source of learning and development, Prentice Hall, New Jersey, 1984.

[8] D. Boud, R. Keogh and D. Walker. Reflection: Turning Experience in to Learning, London: Kogan Page, 1985.

[9] A. Brockbank and I. McGill. Facilitating Reflective Learning in Higher Education, SHRE/Open University Press, Buckingham, 1998.

[10] J. Cowan. On Becoming an Innovative University Teacher Reflection in Action, SRHE/OU, Buckingham, 1998.

[11] J. Moon. Reflection in Learning and Professional Development Theory and Practice, Kogan Page, London, 1999.

[12] D. Schon. The Reflective Practitioner How Professionals Think in Action, Avebury, London, 1991.

[13] The Greenfoot project. http://www.greenfoot.org

[14] The BlueJ project. http://www.bluej.org/

[15] W.-K. Chen and Y. C. Cheng. Teaching Object-Oriented Programming Laboratory with Computer Game Programming, in *IEEE Transactions on Education*, vol. 10, no. 3, pp. 197-203, IEEE Press, 2007.

[16] H. Zhu and M. Zhou. Methodology first and language second: A way to teach object-oriented programming, in *Proc. 18th Annu. Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03)*, Anaheim, CA, 2003.

[17] J. D. Bayliss and S. Strout. Games as a "flavor" of CS1, in *Proc. 37th Special Interest Group Computer Science Education Tech. Symp. (SIGCSE'06)*, Houston, TX, 2006.

[18] C. Alphonce and P. Ventura. Using graphics to support the teaching of fundamental object-oriented principles in CS1, in *Proc. 18th Annu. Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'03)*, Anaheim, CA, 2003.

[19] J. E. Sims-Knight and R. L. Upchurch. Teaching Object-Oriented Design without Programming: A Progress Report, in *Computer Science Education*, vol. 4, no. 1, pp. 135-156, Routledge, 1993.

[20] M. Overmars. Learning object-oriented design by creating games, in *IEEE Potentials*, vol. 23, no. 5, pp. 11-13, IEEE Press, 2005.

[21] J. Ryoo, F. Fonseca and D. S. Janzen. Teaching Object-Oriented Software Engineering through Problem-Based Learning in the Context of Game Design, in *Proc. IEEE 21st Conference on Software Engineering Education and Training (CSEET'08)*, Charleston, South Carolina, 2008.