A Game Based Interactive Development Environment for Non-computer Science Majors

Bo Chen
College of Computer Science
Zhejiang University of Technology
Hangzhou, China
cb@zjut.edu.cn

Yan Liu
Chengdu Institute
Sichuan International Studies University
Chengdu, China
elsa.liu1212@hotmail.com

***Abstract*—Object oriented programming is useful course for non-computer science majors to understand computer language, but there are several problems in the teaching and learning procedure. We designed an interactive development environment based on Greenfoot project. The IDE makes it easy to write interactive graphical applications in Java language. Several game based program examples are introduced into course and used for lab projects. The game designing procedure provides educational tools that aid in understanding fundamental object-oriented concepts, and it is highly motivational through instant graphical feedback. We described the IDE's features, discussed the effectiveness of tool, and describe potential improvements to IDE design and implementation.**

Keywords- Object Orientd Programming; course design; game programming

I. INTRODUCTION

Java is one of the most popular programming languages in use, particularly for client-server Internet web applications. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. Java language is used in pilot course for non-CS majors with prior programming experience and an interest in building software application. This course introduces students to an object-oriented model of programming, with an emphasis on the programming approaches useful in creating software applications. Students will be expected to design, implement, and debug object-oriented programs. Topics include inheritance, user interfaces, and database access.

According to our teaching practice on Java programming for both CS and non-CS majors, there are great differences between the two types of students. Then, it is required that teachers must pay enough attention to the differences, and design different teaching ways to achieve better results.

In contrast to computer science students, the most important features of non-CS majors include:

1) Inconsistent basic level of programming

Since different major students have different computer pilot course, the basic level of students' programming skill has great disparity. For example, to arts and management major students, the pilot course only introduce basic knowledge about computer literacy, almost have no sense on program design and language. But for engineering science majors, most of them have learned some language course like Pascal or C, they only lack of knowledge of object oriented programming. And more, even in same class, there is great individual difference in designing and coding ability.

2) Low English proficiency

Most of the students only have English proficiency in general, reluctant to use development tools in English. Most of them can not read English technical documents. Even for English majors, the error messages output by compiler and linker are so "strange", students have no sense for what's the message means.

3) Lack of debug skills

In practice course, many students cannot debug the program code by themselves and have to ask teachers to correct for them. They know little about the hints in English, cannot understand the error messages from compiler. So it is very difficult to find real problem. Even they know the error messages' meaning, they lack of skills to debug and correct codes.

II. GUIDELINES

In general, Java program design includes two major components: basic knowledge and advanced knowledge. Basics knowledge includes Java basic syntax, Java's basic working principle, the focus is a Java object-oriented content. Advanced knowledge is different with the students' professional varies.

The greatest problem with teaching Java is the large number of circular dependencies of language concepts and constructs. It sometimes seems that to understand anything, you have to know everything. This characteristic makes especially the first few weeks of a course hard to deal with.

Similar with Kolling & Rosenberg, we identified five guidelines that help in Java programming course.

Guideline 1: Objects first. Start thinking everything is objects from the first day. Objects and classes are the fundamental concepts that students need to understand, whenever the students come from CS or non-CS majors.

Guideline 2: Don't start with a blank screen. If students start writing projects from scratch, they have to structure the problem first. This is not a task that students can master at the beginning.

Guideline 3: Read code. Reading code is an essential skill for any software developer. Students can learn a great deal from reading code.

Guideline 4: Use "large" projects. One of the major benefits of object orientation lies in the ability to structure problems into independent modules (classes). Show multi-class examples from the start to convey the right ideas.

Guideline 5: Show program structure. The internal structure of applications lies at the heart of understanding object-oriented programming.

The ultimate goal of programming course is to get students interested in pursuing studies in object oriented disciplines. One way to do this is to give students a positive and fun experience where they felt empowered. A game based development environment allows users to create new classes, objects, and invoke methods simply by clicking with the mouse. This greatly assists teaching in some abstract concepts, like the difference between classes and objects.

III. Related Work

Before choosing our interactive development environment we evaluated commercial, open source, and freeware tools. We decided the benefits of freeware tools outweighed any weakness. We choose tools for their functionality and facilitation of teaching introductory programming and game design, but also committed to keeping the tools free. It also allows students to continue development of their games at home and at school. We paid heavy attention on three projects.

A. *Scratch from MIT*

Scratch is a media-rich, networked environment originally designed for use in "computer clubhouses", a network of afterschool learning centers for youth from economically disadvantaged communities. Scratch's focus on media springs from this initial situation, as clubhouse participants were observed to be "creating and manipulating graphics, animations, videos, and music". Thus Scratch emphasizes media manipulation and supports programming activities that resonate with interests of youth, such as creating animated stories, games, and interactive presentations.

A Scratch project is built from a background and a number of movable sprites. Programming is based on a metaphor of building blocks, and, influenced by pervious systems such as LegoBlocks, is accomplished by dragging blocks from a palette and assembling them, like puzzle pieces, to create "stacks" of blocks. These "stacks" then determine various behaviors of the objects.

B. *BlueJ from University of Kent*

BlueJ is an integrated Java development environment specifically designed for introductory teaching. BlueJ is a full Java environment: it is built on top of a standard Java SDK and thus uses a standard compiler and virtual machine. It presents, however, a unique front-end that offers a different interaction style than other environment.

The environment's interface facilitates the discussion of object-oriented design and aids in using a true "object first" approach. BlueJ provide an easy-to-use teaching environment for the Java language that facilitates the teaching of Java to first year students. Special emphasis has been placed on visualization and interaction techniques to create a highly interactive environment that encourage experimentation and exploration.

C. *Greenfoot from University of Kent*

Greenfoot also come from University of Kent. It is a free, cross-platform programming and simulation environment built on BlueJ environment. The Greenfoot environment allows users to easily create and explore classes, instances, and members using a graphic interface.

A Greenfoot scenario divides user created classes into three categories: World, Actor, and Utility. Users customize the scenario by creating a subclass of the World class. Users then create subclasses of Actor for each type of game entity. Utility classes can be used to accomplish functionality.

IV. Game Based Environment

A. *Fully Localized Interactive Environment*

We use modified Greenfoot as our interactive development environment. The main changes on this open source project include translate the interface language from English to Chinese, and rewrite the online help module for easier access Chinese documents. We also made some wrapper class for the error message output module, so the students can understand how the program running.

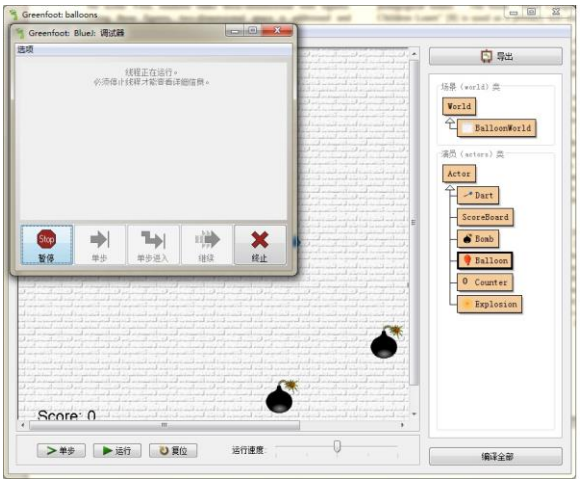

FIGURE I. LOCALIZED INTERACTIVE ENVIRONMENT

B. *Increase Interest by Game*

To attract students to programming, and continued to increase interest on software development, we use game based examples in Java course.

After a brief introduce on Java language, the course enter the game design session. Students are given a working definition of games and then challenged to create a game in the first lab project. They are introduced to the concept of critical and reflective play as a way to assess the quality of the play experience. Students are given opportunities to refine their games and have replayed on next lab session.

In the programming sessions students start with playing a pre-made game to get them comfortable using Greenfoot and then create a simple game by adding one step of complexity at a time. The Greenfoot API includes an Actor class and World class that provide many needed methods including getLocation, creating/removing entities, object intersection tests and keyboard/mouse control. We first create sprites and move them using Greenfoot API methods. Next, we make the sprites move back and forth reflecting at boundaries this introduce if-statements and adding data members for current direction and/or velocity. We then solidify understanding of the coordinate system by having students place sprites using code to spell out block letters using for-loops. Next, students are required to add keyboard control of sprites and use of the Greenfoot provided intersection test for collision detection. Finally, object members are added to keep track of game counters such as timers and the number of objects remaining in the game.

As a result, majority of students are interested to make their "own" games. To solve the game problem, they discussed with classmates and teacher assistants. This shows the power of constructivist learning model.

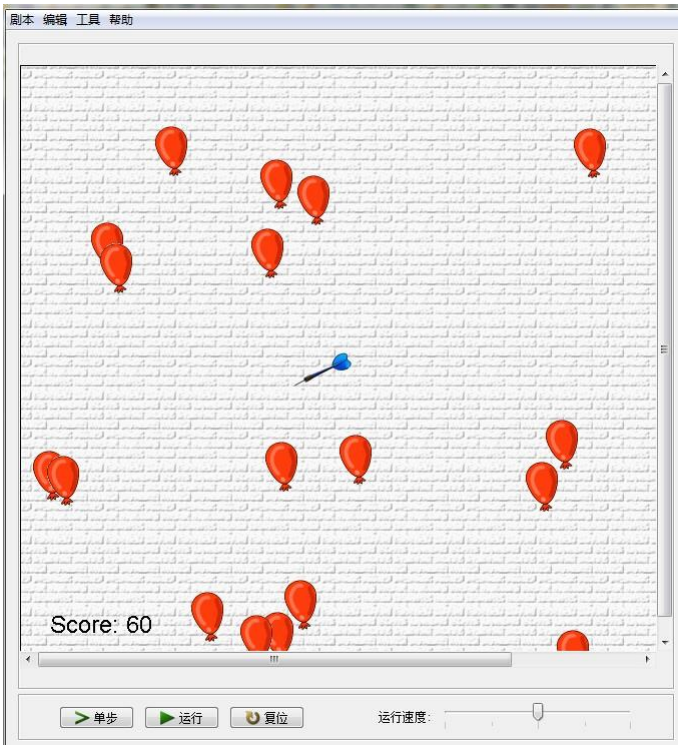

FIGURE II. STUDENTS PLAYING OWN BALLOON GAME

C. *Diving into Code*

The Greenfoot implementation is based on the BlueJ system, and many of the BlueJ tools – the editor, the debugger, Javadoc generation – are available in Greenfoot in a very similar form to BlueJ. The new version of BlueJ also has version control support for manage student's code.

The environment also supplies a view of the classes that participate in the simulation on the right side of the main window. These classes can be edited, compiled and instantiated. These actions can be accessed from the popup menu of the class.

There are two ways to creating new classes. Either by selecting New subclass from the class' popup menu, or by clicking the New Class button below the class icons. The class-browser is divided into two sections.

GreenfootObject classes are the classes that are to be visualised in the world. Their superclass – GreenfootObject – will always be shown in the class browser. The GreenfootObject class cannot be modified.

Subclasses of GreenfootObject will typically have an individual icon. This icon is shown in the representation of the class next to the class name. Greenfoot objects that do not specify an appearance have a default look defined in their superclass.

GreenfootWorld classes are classes that res are classes that represent worlds. Different worlds may exist in a single project (holding, for example, different initial populations of walls and beepers). The superclass of these – GreenfootWorld – will always be shown in the class browser.

The GreenfootWorld classes have popup menus exactly like the ones described for GreenfootObject classes. When a constructor is selected for a subclass of GreenfootWorld, the new world object will automatically replace the existing world in the main view of the Greenfoot user interface.

A class icon can be used to execute a constructor, which results in an object being created and placed on the object bench at the bottom of the main window. Once the object has been created, any public method can be executed.

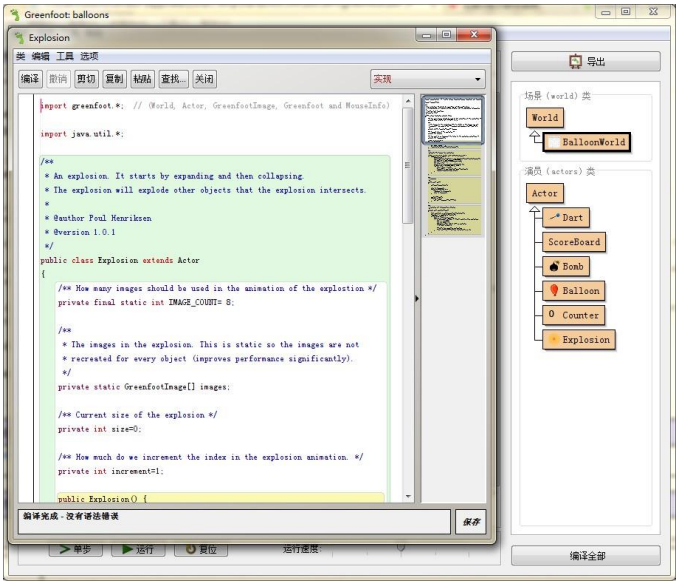

FIGURE III. CLASS HIERARCHY AND CODE EDIT WINDOW

D. *Lab Projects*

The projects involve programming a Java application based on techniques learnt in weekly lab exercises. A wide range of other applications can be fitted into the Greenfoot framework. Since drawing capabilities on the world include the drawing of

text, some objects could display a behavior that displays textual information on the screen. While this is not the main goal for greenfoot, it extends its capabilities.

The Greenroom - Greenfoot Educators Community - has many resources, and more important, the great idea. For example, we encourage biology majors to develop a scenario like the Marine Biology Case Study. This example is structurally similar to Balloon game, and does not require further detailed discussion here. One difference in use patterns is that there are often many fish involved in such a simulation. To add a larger number of fish, a constructor or a method of the world can be used. It is great experience to biology majors that the fishes join into game!

V. CONCLUSIONS

In this paper we conducted a literature review about tools designed for supporting the instructional process of OO programming. These tools are used in computer programming course for non-CS majors. We consider that the BlueJ environment and its extend Greenfoot project hold a lot of advantages. We designed an interactive development environment based on the Greenfoot.

The evaluation teaching practice showed that students highly appreciated the environment, and showed great interest on the game design. The students are impressed for its simplicity and usability. They also commented that the activities performed with this tool enhanced their knowledge and skills.

Now, the Greenfoot does not enable students to design software programs using model notation. So it cannot generate classes' code with code skeletons for further editing. On another side, students have more interest on 3D objects in the game. Our next develop plan will provide dynamic visualization of the software programs. The next generation of tools that support the instructional process of the OOP will include these features.

ACKNOWLEDGMENT

This work supported by Zhejiang Provincial Natural Science Foundation of China (Grant No. LY12F02038). And thanks Zhejiang Key Lab of Visual Media.

REFERENCES

[1] Janet E. Burge , Gerald C. Gannod , Maureen Doyle , Karen C. Davis, Girls on the go: a CS summer camp to attract and inspire female high school students, Proceeding of the 44th ACM technical symposium on Computer science education, March 06-09, 2013, Denver, Colorado, USA

[2] Distasio, Joseph, Way, Thomas, P., Inclusive Computer Science Education Using a Ready-made Computer Game Framework. ITiCSE '07, Dundee, Scotland, United Kingdom.

[3] Fajardo, R., Leutenegger, Scott T., Programming, Pixels and Play: A University Summer Game Camp To Attract Under-represented Populations to Game Development and Computer Science", Proc. Of Future Play, 2006.

[4] Ursula Wolz , Youwen Ouyang , Scott Leutenegger, Scratching the subject surface: infusing computing into K-12 curriculum, Proceedings of the 42nd ACM technical symposium on Computer science education, March 09-12, 2011, Dallas, TX, USA.

[5] Amber Settle, Computational thinking in a game design course, Proceedings of the 2011 conference on Information technology education, October 20-22, 2011, West Point, New York, USA.

[6] The Greenfoot Programming Environment, www.greenfoot.org.