

Why and How to Teach Game Programming

Chong-wei Xu
Computer Science and Information Systems
Kennesaw State University
cxu@kennesaw.edu

Abstract

Game programming is interesting but tough. It could be counted as a sophisticated software development in an undergraduate curriculum because it deals with a wide range of knowledge and skills. This article discusses why and how to teach game programming in order to keep its entertaining nature and to reduce its toughness.

Key words

Game programming, Java programming, modeling, teaching-scholarly activity.

1. Introduction

Programming is a process for designing, implementing, debugging a program. The purpose of programming is to allow the developers to communicate with a computer so that the computer will follow the developers' instructions to perform a certain functions. The communication needs a language that can be understood by both sides. Obviously, English is "too complicated" and "too ambiguous" to be a standard language for computers. A programming language comes in handy. However, the programming language is only for human developers not for computers. Thus, an interpreter is needed. It is the compiler that interprets the source code written in a programming language to a corresponding machine language to make the computer understand what it should be done. Due to the fact that the computer knows nothing, the program should be written in a very detail, or very tedious way. Therefore, programming is counted as a tough work by some people, especially students or beginners.

Any language deals with syntax and semantics. A programming language is the same. In addition, a programming language has its software engineering and scientific flavors, like the mathematics language. Thus, it takes a while to learn. Some students feel difficult on the syntax in learning a programming language. Actually, the real difficulty lays on the software engineering and scientific concepts, i.e. its semantics. For example, the syntax of a simple while loop is not difficult, but applying it to perform an accurate control needs mathematical knowledge and repeated experiments. Learning a programming language is the same as learning a second language. There is no shortcut but practices, which means repeatedly talking with the computer until it "understands" what the programmer said. Definitely, an interesting topic and an exciting interaction in the talking between the developer and the computer would make the learning "interesting and easier" because an interesting topic and a meaningful interaction between the programmer and the computer will stimulate the programmer to continue the repeated experiments without giving it up. In other words, the most important factors for encouraging learners to persistently practice the programming language and exercise it

are the interesting degree of the topic and the reactions of the computer. It just likes two people talking. If the topic is very interesting and the reactions are exciting, they will continue their talking for a long time.

What topics for programming might satisfy these criteria? Games. Games are welcomed by many students because a game has a nice story and generate an active and a visual response. The player can continuously communicate with the game and the visualization and animation of the game can stimulate the player's interests with a variety of dynamic scenes and objects. Playing games like this, however, designing and implementing a game could be another story due to the fact that a game is a mixture of sciences, multimedia, arts, artificial intelligence, and so on [3]. To put all of them together is not an easy job. Fortunately, the developer is the first player of the game, he/she also involves an interesting talk. Especially, when the programming is finished, the developer will enjoy the most exciting moment. In sum, game programming is interesting but tough.

For encouraging students learning, we have offered game programming as an interesting topic. This article intends to discuss the reasons for teaching game programming and the strategies for keeping its interesting but reducing its toughness. Section 2 discusses why we teach game programming. Section 3 introduces how we teach it. Then, section 4 draws the conclusion and indicates future work.

2. Why teaching game programming

Games are interesting and attractive. Many efforts have been made to offer game programming into computer science curriculum. The articles [1], [6], and [8] are just a few examples. For teaching game programming there appear to be four approaches [9]: 1) to support foundations courses; 2) to provide specialized content at the upper level; 3) to provide a curriculum encompassing thematic approach to CS; 4) to provide trans-disciplinary experiences for CS students. We are following the second approach.

Many students only thought games are interesting before entering the course. By learning game programming, many students recognized and understood why Computer Science (CS) curriculum has so many prerequisites. Firstly, they recognized the importance of mathematics and physics. Game programming deals with many computations. Even a simple 2D game needs computations for setting up the position of multiple objects, for checking their moving speeds and wrapping them when they hit the edges of the screen, and for fitting a game into different screen sizes and different hardware limitations. The visualization clearly depicts any inaccurate computations immediately. How about 3D games? Obviously, there are no 3D games without mathematics [2]. For implementing games for handheld devices [10], J2ME does not support floating-point computations. Many students first time recognized the values of floating-point numbers. Some students found out the importance and usefulness of mathematics and went back to take more mathematics courses.

Secondly, students recognized the power of object-oriented programming. A game handles multiple different objects. In order to make every object dynamically moving,

animation is the foundation. For making animation, one or multiple threads should be created to make a game loop for controlling all objects. The game loop does three things: paint all objects, update every object's position, and sleep for a while. It requires every object can paint itself by implementing a `paint()` method and update its own position by implementing a `updatePosition()` method [4]. Factoring out these common parts, all objects form an inheritance hierarchy and all update actions form an interface. The root of the object hierarchy becomes an abstract class, all real objects correspond to concrete subclasses as Figure 1 shows. This structure facilitates the Open-Closed principle (OCP). The OCP principle is the first principle for developing large size of software. The principle says, "Software entities should be open for extension but closed for modification." [7]. That is, a software entity should be able to be extended without modifying its source code. The abstract class and interface on the root of the hierarchy implements the closed part of the OCP principle because they will not be modified and they predict all possible extensions. The concrete subclasses implement the open part for any kind of extensions. Due to the fact that all objects come from the same root of the super-class, they have the same type so that they can be instantiated and assigned into one array or one vector. For invoking the `paint()` and `updatePosition()` methods defined in every subclass, a polymorphism concept and dynamic binding mechanism is applied. Evidently, all important features of OOP technology, such as inheritance, abstract class, interface, polymorphism, thread, and so on, are put into engineering practice, which allows students having a chance to apply their OOP knowledge for a software engineering domain instead of many unrelated examples. That promotes students to deeply understand the spirit of the OOP technology.

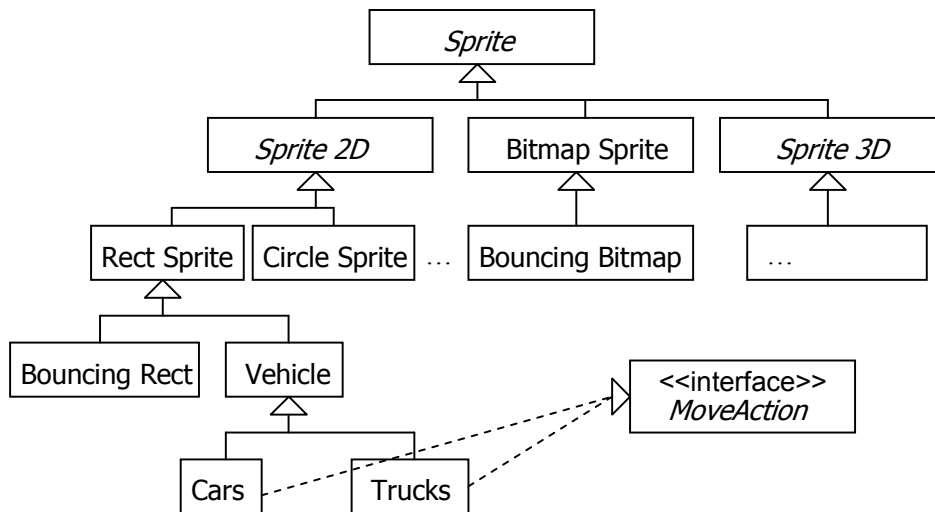


Figure 1 An example of the object inheritance hierarchy.

Thirdly, students recognized that they are the real commanders of computers. When they play a game, they felt the magic of the computer. Now, they can design and implement a game by integrating all arts, multimedia, images, and music etc. together to realize a story. They understood that the power of computers come from the power of themselves. They are no longer afraid of programming but enjoy the entertainments of programming.

3. How to teach game programming?

For entertaining programming, there is a lot of teaching-scholarly activities should be done [5]. The first and foremost is to modeling a game. The modeling means to decompose a game into modules so that each module can be implemented independently with a step-by-step refinement to complete a complicated game.

The model that we used for teaching can be depicted as Figure 2. That is, programming a game consists of the following major steps.

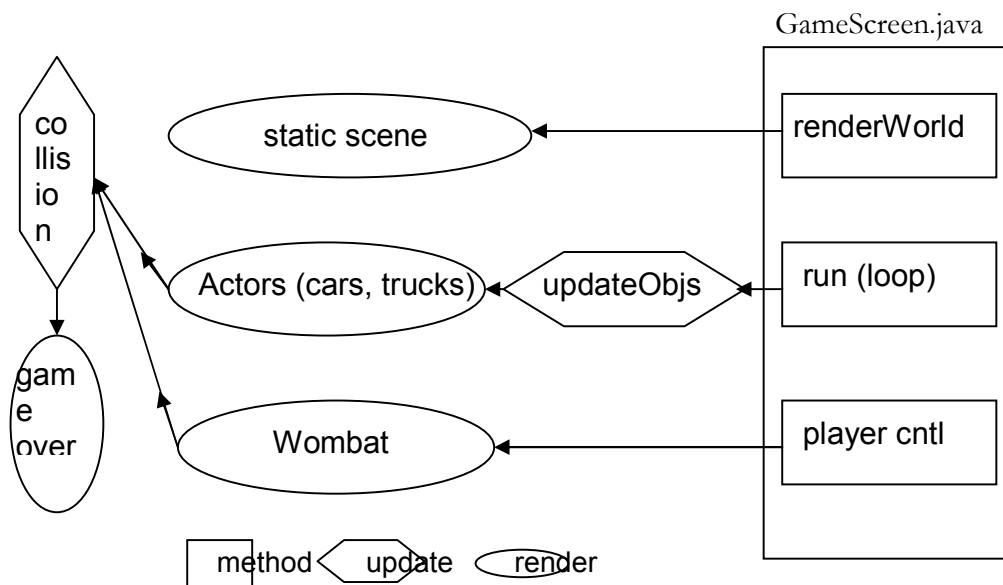


Figure 2 A model for teaching game programming.

Rendering a static scene is the first and simplest step. The static scene usually is the background of a game. It could be the decoration part of the game, some static buildings, ocean water, sand ground, or lanes of a highway. They could be an image or graphics generated by a part of the paint() method.

Then, adding all objects statically onto the static background to display a complete frame for the game. The objects are the instances of the concrete subclasses in Figure 1. They are instantiated and assigned into a data structure, such as an array, a vector, or a list. The main program, for example the GameScreen in Figure 2, invokes the repaint() to call all paint() methods implemented in every object to display itself. This step shows all objects and their initial positions.

The third step is to animate all moving objects. This step adds a Thread object and makes the game loop running. The game loop invokes updateObjs() to call all updatePosition() methods that is defined in the same interface and implemented in every object to update the positions of each of them. Each object will update its position according to its own updating algorithm based on its own moving routes and speed setting. After this step, the developers can see a complete frame with static scenes and dynamic moving objects.

The fourth step is allowing the game player to control the objects according to the story of the game. The interactive controls are based on the key strokes or mouse clicks. The events and event handling are applied for realizing all controls of the player.

The fifth step is to handle the collisions. All moving objects including the objects controlled by the game player could be collided according to the game story. The effect of a collision could be widely defined, such as changing the moving direction, crash some objects, or even stop the entire game.

The last step is a refinement or a polish of the entire game. This step could involve very broad actions, such as providing the controls for pausing the game and restarting the game, counting the scores, determining the levels of the game, and so on.

Applying this model is not only making programming easier but more important is to train students with an analysis and synthesis ability such that whenever they see a new game, they will have an idea how to decompose the game into parts and figure out what should be done step-by-step to build up the game.

4. Conclusion and future work

In an undergraduate curriculum, programming a game could be counted as a comprehensive work. Teaching game programming for beginners and students may involve a dilemma. One side is the entertainment nature of games, thus interesting and attractive. The other side is the sophistication of design and implementation. Decomposing a game into modules and integrating the modules together step-by-step is a practical method for overcoming the problem. The ability for analyzing and synthesizing a project allows many students being able to imagine and design the implementation steps for making a game. This problem solving ability is the goal of our education.

When involving 3D games, mathematics and physics become a bottleneck. How can smoothly cover the topics from vector and matrix computations to coordinate system transformations? Even provided a game engine based on mathematics and graphics, 3D programming is still tough. A better solution could be a special Game Math course plus a 3D Game Programming course. The difficulty is that it may involve a change of the curriculum. Many teaching-scholarly activities are waiting.

References

- [1] Jessica Bayliss and Sean Strout, "Games as a 'Flavor' of CS1", Proceedings of the thirty-Seventh SIGCSE technical Symposium on Computer Science Education, Houston, Texas USA, march 1-5, 2006.
- [2] David Brackeen, B. Backer, and L. Vanhelsuwe, "Developing Games in Java", New Riders, 2004.
- [3] Andrew Davison, "Java Graphics and Gaming", an online book, <http://fivedots.coe.dsu.c ac.th/~ad/ig/index.html>.

- [4] Joel Fan, E. Ries, and C. Tenitchi, “Black Art of Java Game Programming”, Waite Group Press, 1999.
- [5] John P. Flynt and O. Salem, “Software Engineering For Game Developers”, Thomson Course Technology, 2005.
- [6] Mark Lewis and Berna Massingill, “Graphical Game Development in CS2: A Flexible Infrastructure for a Semester long Project”, Proceedings of the thirty-Seventh SIGCSE technical Symposium on Computer Science Education, Houston, Texas USA, march 1-5, 2006.
- [7] R.C. Martin, “The Open-Closed Principle”, “Engineering Notebook—C++ report”, January, 1996.
- [8] Ian Parberry, Max Kazemzadeh, and Timothy Roden, “The Art and science of Game Programming”, Proceedings of the thirty-Seventh SIGCSE technical Symposium on Computer Science Education, Houston, Texas USA, march 1-5, 2006.
- [9] Ursula Wolz, Tiffany Barnes, Ian Parberry, and Michael Wick, “Digital Gaming as a Vehicle for Learning”, Proceedings of the thirty-Seventh SIGCSE technical Symposium on Computer Science Education, Houston, Texas USA, march 1-5, 2006.
- [10] Martin J. Wells, “J2ME Game Programming”, Thomson Course Technology.