Chapter XXIX
Teaching OOP and COP Technologies via Gaming

Chong-wei Xu
Kennesaw State University, USA

ABSTRACT

This chapter introduces an innovative pedagogical method for teaching object-oriented programming (OOP) and component-oriented programming (COP) via gaming. Going through the evolution of the three-layer gaming framework, we clearly illustrate that gaming covers almost all core features of OOP and COP technologies. Teaching OOP and COP technologies via game development not only engages students' efforts, but also opens an opportunity for involving students with industry-level projects and enhancing students' ability to brainstorm and solve real-world problems. Furthermore, gaming may play an important role in developing other applications, especially those that feature visualization and animation.

INTRODUCTION

The game industry is growing rapidly. "The worldwide game industry reached $33.5 billion in size in 2005, with expected growth to $58.4 billion by 2007, a near doubling in size in a two-year period." The hiring requirements at a typical game company are something on the order of 65% computer scientists, 30% artists, and 5% designers. In addition, government and non-game-industry corporations for the new area of serious games are also growing (Zyda, 2006). "Game-play has begun to surpass television viewing among some segments of the population. Video game development budgets are already the size of motion picture development budgets, on the order of $20 million to $100 million, with expected revenue for a hit game reaching from $250 million to more than $1 billion" (Zyda, 2007).

We would expect these data to show that the demand for computer scientists is increasing. However, internationally, computer science has experienced a 70% decrease in undergraduate enrollments since 2000 (Zyda, 2006). "Don't just play games, create them!" has become an attractive and interesting innovation in computer science curriculum. Several universities have created game degrees, while a number of other universities have set up game tracks and/or courses. Actually teaching game development is valuable not only because students and the young generation are interested in games, but also because gaming itself has the potential to revitalize and increase interest in computer science (Zyda, 2006). First of all, games have a unique role in education. Games "do all of the things that the learning scientists told us worked well" (Schollmeyer, 2007). Games support the following effective learning paradigms identified by learning science: experiential learning ("If you do it, you learn it"); inquiry-based learning ("What happens when I do this?"); self-efficacy ("If you believe you can do it, you will try longer/harder, and you will succeed more often than you would otherwise"); goal setting ("You learn more if you are working toward a well-defined goal"); and cooperation (team learning) (Mayo, 2007). Secondly, games are the integration of humanity, mathematics, physics, arts, artificial intelligence, graphics, visualization, animation, sounds, images, programming, and so on. Gaming itself is becoming a science (Zyda, 2007). Consequently, teaching game developments will promote the further study of all sciences, especially mathematics and physics, programming skills, and problem-solving abilities.

Here, we are going to concentrate on the discussion of teaching programming knowledge and skills. The traditional way of teaching programming is going over one chapter for covering one topic, such as class, object, inheritance, polymorphism, and so on. Specifically made programming exercises are used to demonstrate and explain the individual concept. Unfortunately, this traditional approach misses the connections among the different topics, and students fail to see opportunities to put these technologies together to solve a real-world problem. The drawbacks of this pedagogical method often appear on students' evaluations, such as "The program didn't do anything cool" (only see some text outputs) or "no idea what it was doing" (change programs do not affect the output) (Phelps, Bierre, & Parks, 2003), due to the fact that students cannot see the effects of their programs intuitively. Games are based on graphics, visualization, and animation. Students can easily see the cause-and-effect relationship between their programs and the results in the format of graphics and animation. In addition, visualization and animation will further catch their attention and encourage their interests.

This idea has been demonstrated by many famous projects, such as "Logo," "StarLOGO," "Karel," "Programmable Brick," "Lego RCX Brick," "Alice," "MUPPETS," and so on (Phelps et al., 2003; Bierre & Phelps, 2004). These excellent projects mainly create a 3D virtual environment for students to control a robot or a set of characters to visualize the programs' effects, to extend and practice the programming skills, and to engage students' efforts.

Considering the limitations of resources and time, we are trying to directly teach the programming language (Java) through gaming. We select the 2D simple games, such as *Worm, WormChase* (Davison, 2005), *Breakout, Othello,* puzzle games, and so forth, as examples to teach object-oriented programming (OOP) and component-oriented programming (COP) technologies. Even though these games are simple, they all include static scenes, dynamic object animation, collision detections, player controls, a score calculation system, and a level system. We apply analysis and synthesis methodology to partition a game into pieces, use incremental development strategy to build up a game step by step, and incorporate term projects with presentations to bundle all related OOP and COP concepts together for designing and implementing one project—a game. The contents of a game are rich and interesting and closely related to the industry world. The size of the project is suitable for students and feasible to be finished in one semester. In other words, game development promotes a new teaching method for not only learning, but also applying what has been