Teaching Non-majors Computer Programming Using Games as Context and Flash ActionScript 3.0 as the Development Tools

Yue-Ling Wong
Wake Forest University
Computer Science Department
Winston-Salem, NC 27109
(336) 758 3922
ylwong@wfu.edu

ABSTRACT
This paper describes a new course developed for non-majors to teach them programming concepts using games as context and Flash ActionScript 3.0 as the programming language. While Flash provides a graphical environment, the curriculum is designed to help students not to rely on graphical aids in coding by helping them overcome the common anxiety in learning syntax and dealing with syntax errors. The curriculum design and the language choice will be discussed.

Categories and Subject Descriptors
K3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, curriculum.*

General Terms
Design, Experimentation

Keywords
Non-majors, introduction to programming, curriculum, games, Flash ActionScript.

1. INTRODUCTION
This paper describes a new course that is designed for non-majors who may be more attracted towards visual and interactive multimedia programs. The course also intends to satisfy non-majors' desire and need to learn programming concepts and practice but may hesitate to attend CS1 programming courses. For examples, many digital art students who are interested in creating interactive projects are interested in learning programming, particularly in Flash ActionScript and JavaScript.

While this new course is developed for non-majors, the concepts of programming and algorithms taught in this new course follow closely with those taught in CS1 introductory programming courses and are applicable across programming languages and applications.

Technical Report, 2008-1, Department of Computer Science, Wake Forest University. Updated: 2009-06-25

2. WHY FLASH ACTIONSCRIPT
The language chosen for the new course has to be robust and full-fledged enough to support the topics that are covered in CS1 introductory programming courses. There are several reasons for the choice of Flash ActionScript 3.0.

2.1 Optional Object-Oriented Programming (OOP)
ActionScript supports OOP but OOP is also optional. This affords us the flexibility to start the course with non-OOP. OOP often is considered as a difficult subject for most students who are novice to computer programming. In this new course, I start with non-OOP to familiarize students with syntax, and translating ideas into statements and event handlers. OOP is then introduced in the second half of the semester.

2.2 Availability of Drawing Tools
Flash environment provides the drawing tools to easily create graphics to be used in the program. This frees up the time—that normally are required to learn the drawing API—for students to learn event handlers and to program to manipulate and animate these graphics.

2.3 Easy Testing the Application
The Flash file can be tested by selecting one command (Control > Test Movie) or one keyboard shortcut. Flash creates a SWF file and automatically opens it in a Flash Player window. Any compiler errors will be displayed in the Compiler Errors panel. There is another panel for run-time errors. Although the errors are sometimes cryptic, some provide the line number for the error.

2.4 Built-in Methods for Simple Collision Detection
Collision detection between two objects is indispensable in most of the computer games. However, collision detection algorithms normally are not part of the curriculum of the introductory course. ActionScript offers an advantage of having two built-in methods for collision detection: between two objects (by using their bounding boxes) and between a point and an object. These two methods are sufficient for most situations.

The discussion of the pros and cons of these built-in methods naturally leads to a distance-based approach. Pythagorean theorem is reviewed in class and its application is explained in distance-based collision detection of circles. The concept is then translated into code in one of the labs.

2.5 Similar Syntax as Java and C++

ActionScript syntax is similar to Java and C++. This allows students to transit to these languages should they decide to continue onto the next level of computer programming classes.

To show you that Flash ActionScript 3.0 is very similar with other programming languages and has evolved to the extent that it can be used as a language to provide non-majors hands-on introductory programming concepts that are applicable across programming languages, here are some descriptions and examples of Flash ActionScript 3.0.

2.5.1.1 Data Types

Data types supported in ActionScript 3.0 include String, int, uint, Number (double precision), Array, Boolean, Object, and Movieclip.

2.5.1.2 Operators

In addition to arithmetic and logical operators, bitwise operations also are supported.

2.5.1.3 Variables

You can set up to require variable declaration. Here are two examples of variable declaration.

```
var x:int = 5;
var name:String = "Lucy";
```

2.5.1.4 Arrays

Flash ActionScript supports multidimensional arrays and array operations such as concat, join, pop, push, reverse, shift, slice, sort, splice, and unshift for arrays. Here is an example to declare an array.

```
var alphabet:Array = new Array("a","b","c");
```

2.5.1.5 Functions

Here is an example of a function definitions in ActionScript.

```
function restartGame(b:int, c:int):void
{
  score = b;
  health = c;
}
```

2.5.1.6 Event Handlers

Flash ActionScript supports mouse and keyboard events, and additional Flash-specific events. Shown below is the he basic structure of ActionScript code to handle events. The built-in method `addEventListener()` is used.

```
function eventResponse(eventObject:EventType):void
{
   // statements
}
eventSource.addEventListener(EventType.EVENT_NAME,
eventResponse);
```

2.5.1.7 Program Flow

If-statements, switch statements, for-loops, while-loops are supported in ActionScript. For example:

```
if (age < 40 || weight < 150)
{
  x = 2;
}
else
{
  x = 3;
}


switch (status)
{
  case 1:
    trace("walk");
    break;
  case 2:
    trace("jump");
  case 3:
    trace("kick");
    break;
  case 4:
    trace("duck");
    break;
  default:
    trace("stand");
}


var sum:uint = 0;
for (var i:uint = 1; i < 100;  i++) {
  sum += i;
}
```

2.5.1.8 OOP

Shown below is an example code of defining a class.

```
public class Tree
{
  public static const GRAVITY:Number = -0.2;
  public static const FRICTN:Number = 0.98;
  public static const FOCALLENGTH:int = 250;
  private var elevation:Number;

  public function Tree()
  {
    elevation = 50;
    applyLinearPerspective();
  }
  public function applyLinearPerspective()
  {
    //statements
  }
}
```

To create subclass, you use the keyword `extends`. For example:

```
class OakTree extends Tree
{
}
```

The keyword `override` is used to redefine inherited methods in ActionScript 3.0.

3. WHY GAMES AS CONTEXT

Computer games have been used in a various computer science classes [2] (such as CS1 [1, 3, 4, 5], CS2 [3], and computer

graphics [6]) as a motivator and a mean to help students understand the underlying concepts and principles. The course described in this paper shares many of the same goals and ideas. For example, the course is not intended to be a game design course. In addition, games as context are chosen as a theme in both lecture and lab components while following the set of expected outcomes of traditional CS1 courses. These courses often use Java as an introductory programming language. However, the target student population for this course is the non-majors.

Games often have multiple sprites, each of which has the same types of properties and can perform similar tasks. The concepts of OOP thus naturally apply in game programming. One of the requirements for the term project—programming a game—is to implement the basic OOP—creating classes, instantiating them to generate multiple objects, and making them perform tasks by invoking the class methods. Inheritance and polymorphism are considered as advanced topics in this course. The implementation of these two concepts in the project is considered as an extra credit. It turned out that almost everyone's project made use of subclasses and overrode inherited methods to various extent. For most projects, it was driven by the games' need to create subclasses.

4. CURRICULUM DESIGN

The curriculum of this course was guided by the expected outcomes of traditional CS1 courses that use Java as an introductory programming language. However, because of the interactive and visual nature of computer games, these characteristics play an important role in determining the order of the topics to be taught. The order may be different from the traditional courses. For example, the mouse and keyboard event handling are introduced early in the semester. Same is true for the manipulation of the x- and y-coordinates, size, location, and transparency of an object (or a Movieclip instance.) These topics usually are taught later in the semester in traditional CS1 introductory programming courses.

Furthermore, there are some additional concepts (such as gravity, friction, and collision detection) that may not be part of the traditional CS1 courses but are applied quite often in programming the games.

4.1 Learn to Read First, Then Write

When we learn a new language—a human language—we first learn to read before we start to write. When we first learn to read, we learn the spelling of single words and grammar before reading sentences. When we start to read an essay, the words and grammar start to make more sense. When we start to write in that new language, we start out with simple sentences and short essay with an idea or topic assigned to us to write about. Then we start to express our own thoughts in longer essays. This new course follows the similar approach to introduce students to a programming language.

An introduction to syntax is often introduced in bits and pieces—one keyword or construct at a time. To give the students a big picture of how all different pieces come together in a functional program, students are given a functional game program where they are asked to identify keywords and programming constructs (such as variables, constants, if-else statements, operators, and predicting values of variables) immediately after an introduction to the syntax. Students can see the code of a functional game and compile it even before they start to write any code.

4.2 Practice Reading Error Reports Early On

While computer programmers accept that typos and syntax errors are common in coding practice, students often get frustrated when their script fails only because of a misspelled word or an extra semi-colon. To help students build confidence in reading error reports and troubleshoot their code, a lab is designed such that the students are given a functional game program and asked to make a typo in the code, compile the program, and read the error. Students are also asked to have their peer to introduce two typos in the program, and they need to troubleshoot to identify and fix the errors. This lab is introduced before the students start to write code on their own. It intends to lessen the possible frustration by explaining to them that typos and syntax errors are common experience, and providing them a sandbox to practice reading and fixing errors before they debug their own code.

4.3 Introduce Vector Graphics and Bitmapped Images Followed by Animation Basics

To get students acquainted with Flash, lab exercises for drawing tools and animation were included at the beginning of the semester. Animation basics (concepts such as keyframes, tweening, and frame rate) are introduced after the introduction to vector graphics. Because Flash is a vector graphics program, it is natural to introduce vector graphics, which in turn naturally leads to a discussion of bitmapped images versus vector graphics.

4.4 Introduce Event Handling Early On

As explained earlier, the event handling needs to be introduced early on in the semester so that students can start adding interactivity even in the first game lab assignment.

4.5 Design of Labs

The first game programming lab starts in week 4. It is programming a Pong-type of game. It is scheduled to be right after an introduction to event handling. It provides students an opportunity to program with the mousemove event handler (using the mouse to control the paddle) and the frame event handler (to move the ball continuously on its own.) The lab also provides the opportunity to control the x and y properties of an object (or Movieclip instance). The built-in collision detection methods are introduced to detect when the paddle hits the ball. This also requires writing if-statements. The lab also introduce the concept of using functions to reduce code duplication.

The side-scrolling platform game requires keyboard event handling, modeling gravity effect on the hero, and collision detection. These latter two topics are introduced within a week before the lab.

The Tic-Tac-Toe lab provides students an opportunity to use a 2-dimensional array to model a game board. This lab is arranged to be right after an introduction to arrays.

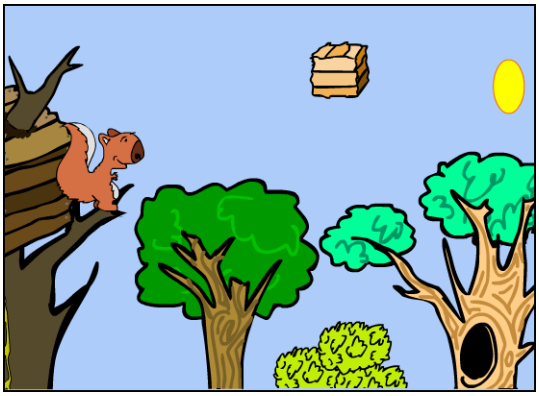
The lab right after the concepts of OOP are taught is to modify the previous Pong game lab into OOP version so that the game has multiple balls. A Ball class is created and instantiated to generate multiple Ball objects. This lab builds on its non-OOP version of

an earlier lab. The event handling and the code regarding the control of the paddle and ball control are already there. This allows students to focus on the OOP component.

The subsequent lab creates a more complex program using OOP from scratch. It involves creating a class and instantiating it to generate 100 objects. This lab also introduces concepts of linear perspective to create a 3-D environment that allows the player to use keyboard to move forward, backward, left, and right, and fly up.

The side-scrolling platform game (Figure 1) and 3-D fly-through (Figure 2) labs allow students to express their creativity of the visual elements while following the same structure of the code.

(a)[1]



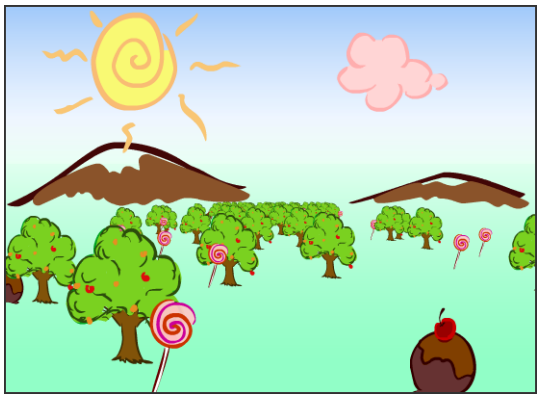
(b)[2]

Figure 1. Examples of students' side-scrolling platform lab.

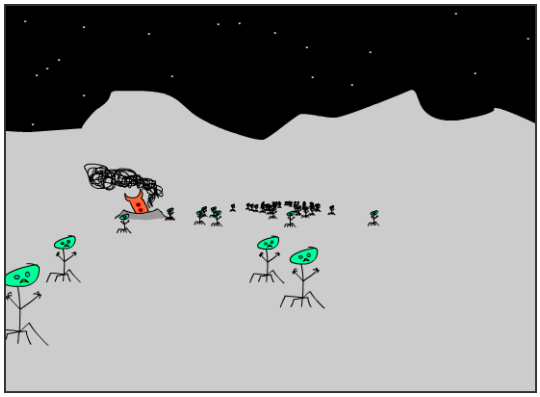
Many of the graphics created in the drawing and animation labs at the beginning of semester are used in these programming labs. This frees up students' time in the graphic creation in these labs so they can focus more on the coding.

4.6 Design of Term Project

The labs are very structured and designed to let students practice translating small pieces of ideas into code. There are no game design or planning required from the students. On the contrary, the term project is to provide students opportunity to plan and work towards their game ideas more independently. Some projects extended the labs while some were created from scratch.



(a)[3]



(b)[4]

Figure 2. Examples of students' 3-D fly-through lab.

5. RESULTS AND CONCLUSIONS

The goals of this paper are to share my experience of the new course for non-majors and generate interest and discussion about teaching computer programming within the context of computer games. The curriculum design issues discussed in this paper may serve as a model for introductory programming courses that use any programming languages.

To help students build confidence learning new programming languages on their own or taking a CS1 course in the future, the class also exposes the students to programs written in other languages, such as Java, C++, and JavaScript, at the end of the

[1] Courtesy of Corey Lons

[2] Courtesy of Kevin Crace

[3] Courtesy of Liyang Diao

[4] Courtesy of Marc Legrand

semester. The students are given Java and C++ code of functional games. The programs are written in parallel to one of the labs that they have written in ActionScript so that they can easily see the similarities. The students were asked to identify the keywords and programming constructs, and compile the code using Microsoft Visual C++ Express. In the final exam, students were asked to identify the programming constructs in Java, C++, and JavaScript code. Almost all of the students were able to identify these constructs.

6. ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Grant Numbers DUE-0127280 and DUE-0340969.

7. REFERENCES

[1] Bayliss, J. D. and Strout, S. 2006. Games as a "Flavor" of CS1. In Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (Houston, Texas, March 1-5, 2006), 500-504. ACM Press.

[2] deLaet, M., Kuffner, J., Slattery, M. C., and Sweedyk E. (moderator) 2005. Computer Games and CS Education: Why and How. In Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (St. Louis, Missouri, February 23-27, 2005), 256. ACM Press.

[3] Giguette, R. 2003. Pre-Games: Game Designed to Introduce CS1 and CS2 Programming Assignments. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (Reno, Nevada, February 19-23, 2003), 288-292. ACM Press.

[4] Leska, C. and Rabung, J. 2004. Learning O-O Concepts in CS I Using Game Projects. In Proceedings of ITICSE'04 (Leeds, United Kingdom, June 28-30, 2004), 237. ACM Press.

[5] Leutenegger, S. and Edgington, J. 2007. A Games First Approach to Teaching Introductory Programming. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, Kentucky, March 7-10, 2007). ACM Press.

[6] Sung, K., Shirley, P., and Rosenberg, B. R. 2007. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, Kentucky, March 7-10, 2007), 249-253. ACM Press.