2006-2206: TEACHING COMPUTER PROGRAMMING SKILLS TO FIRST-YEAR ENGINEERING STUDENTS USING FUN ANIMATION IN MATLAB

Ramzi Bualuan, University of Notre Dame

Ramzi Bualuan is the Director of Undergraduate Studies in the Department of Computer Science and Engineering at the University of Notre Dame. He taught all the Notre Dame freshman engineering courses between 1994 and 2000, and he designed and teaches the programming module used in the current freshman engineering course sequence.

Teaching Computer Programming Skills to First-Year Engineering Students Using Fun Animation in MATLAB

Abstract - This paper describes an approach to teach computer programming concepts to first-year engineering students. The environment of choice is Matlab, and the teaching method consists of requiring the students to create a project in which they will code a computer game with the use of functions that are provided to them. The gaming aspect of the project generates a high level of fun which enhances the learning process. The project is one of the four modules that Engineering Freshmen work on while taking their two Introduction to Engineering Systems courses. It spans half a semester, and, unlike the other three group-based projects, is individually-based.

Matlab provides a wide range of animation tools and functions that allow a user to create very nice animation and games. Mastering those tools however is not a task for the novice, and would be too overwhelming for an introduction to computer programming module. The approach presented here raises the level of abstraction for the student so that he/she will not have to deal with the lower-level details of Matlab's animation tools, but instead focus on the problem itself, and thereby develop the proper programming skills and learn important concepts such as selection, iteration, arrays, etc.

Introduction

Teaching programming skills to Freshmen engineers has been a pedagogical challenge for many colleges and universities, especially in a large classroom setting. Issues such as the choice of the language and the selection of the topics to cover have been influenced by the engineering departments and by the students themselves. Some students come in with no prior programming background, others are self-described "hacks", and many are in between. Finding a pace that fits a skill-diverse group has its problems. Furthermore, even though students do not enter their engineering departments at our institution until their Sophomore year, by the start of the Spring semester of their Freshman year most of them already have an idea of what engineering department they will choose. Finding a programming project that fits all fields of engineering is therefore quite a task. Moreover, each department in the school of engineering wants to have its say on how Freshmen engineers should develop their programming skills. These students will, after all, soon be entering their departments.

This paper presents the approach that has been used the past three years at the University of Notre Dame, in the programming module of the Introduction to Engineering Systems course sequence. The sequence consists of two courses that all Freshmen engineers must take. Those two courses consist of four modules, two per semester, each of which is project-based. In the Spring of 2004, one of the modules became a programming module, which is the focus of this paper. The course sequence mentioned above has been presented in previous conferences[1,2]. It should be noted however that the design and the pedagogical philosophy behind the development of the programming module is totally independent from and unrelated to the development of the courses themselves.

Background

At the University of Notre Dame, all incoming Freshmen are part of the "First Year of Studies" (FY), and are fully managed by that entity. Students do not officially become part of their intended major's department until their Sophomore year. The FY controls, among other things, the students' schedules. For Freshmen who are engineering intents, the normal schedule includes math, physics, and chemistry courses specifically designed for engineering students, as well as the two-course sequence mentioned above.

The class size for each of those two courses is usually between 250 and 300. The class consists of two 50-minute lectures each week, as well as a 75-minute lab (called a "learning center"), with around 30 students per lab, where students work on examples and problems. Students work on the programming module, one of the four modules in the two-course sequence, during the span of half of a semester. The module ends with a final project, which each student works on and completes individually (the remaining three modules allow group projects). A textbook[3] and class notes are used for reference. The language of choice is Matlab.

Incentive

It is no secret that students find more incentive to work on a task when they are enjoying it and having fun. In the world of programming, fun is usually associated with games. The method presented here is one where students develop a computer game in Matlab, with the use of some basic extra functions that have been provided to them.

Matlab contains some nice and powerful graphics functions and animation tools, and offers the user means to control the graphics. For a novice programmer however, manipulating Matlab's graphics is quite complex and detailed. Teaching those graphics as-is would be rather counter-productive, as students would then spend too much time on the little details and complexities related to the graphics functions, instead of on the fundamental skills that we are trying to teach them.

Setup

The solution therefore is simply to hide those details from the students by creating wrappers around the graphics functions. The wrapper functions – which we will call "project functions" – act as black boxes to the students, who therefore only have to worry about what the functions' inputs and outputs are, and what the functions do. Due to the nature of Matlab, students do have access to the project functions' source codes. Most of them do not care to even bother to look, but some are very interested in how things are done "under the hood", and have the incentive to do things on their own. Incidentally, at the end of the module, and after finishing their project, students have the option to write a game of their choice, which is not graded but for which the top three games get an award. About ten percent of the class take on that challenge, to very impressive results.

In a given year, several project functions are created for the programming project. They usually include: one to create a window, given its dimensions; one to create a ball, given its radius, with control over its color; other functions create objects of some different geometrical shapes; each object has a 'handle' associated with it (a value returned by the function that creates it), which is used to manipulate the object. Other functions include: the ability to move an object in the *x* or the *y* direction, given the object's handle and the distance in pixels; a function to hide an object; and functions that are specific to the particular project chosen in a given year. Occasionally, some of the project functions create objects that react to various key presses or to mouse clicks, adding a level of excitement to the development of the game.

Example

In the Spring of 2006, students had to implement the video game "Pong". The project functions that were provided included: createWindow, createPaddles, xMove, yMove, hide, drawBall, getCenter, setTitle. The functions require some parameters as input, which set the window dimensions, the paddles' dimensions, and the ball size. Animation is done by moving the objects and iterating. Some of the game's graphic features, as well as the paddles' response to key presses, are hidden inside the functions mentioned above, and are totally transparent to the students.

Figure 1 below shows a snapshot of the game at a time when player 1 (left) is leading player 2 (right) 3-2, and player 2 is about to return the ball. The ball bounces off a paddle as a function of its point of impact, and bounces off the side court (horizontal lines) just as a mirror reflection. A point is scored if the ball goes past a backcourt (vertical walls), and a new ball is then served.

Figure 1: snapshot of the Pong game

Figure 2 below shows descriptions of some of the project functions provided in the Pong game. As mentioned above, students basically treat them as black boxes, as if they were part of Matlab's intrinsic functions. Some students are actually surprised when they find out that the functions are not part of Matlab.

`createWindow`:
usage: `createWindow (wid, ht)`
purpose: Generates a window whose width is `wid` pixels and height is `ht` pixels.

`createPaddles`:
usage: `[p1 p2] = createPaddles (s1, s2, t)`
purpose: Creates two paddles of length `s1` (left paddle) and `s2` (right paddle) thickness `t` on the left and right sides of the window (the window must already have been created). The paddles are set up to move based on some key presses (do a `help` on the function to learn more). Returns handles to the paddles (`p1` is a handle to the left paddle, `p2` to the right paddle)

`drawBall`:
usage: `h = drawBall (xc, yc, r)`
purpose: Draws a ball (filled circle) centered is at (`xc`, `yc`) pixels, and with a radius of `r`. Returns a *handle* `h`, i.e. a variable that can help identify the ball being created. The handle can be used by some of the functions below.

`xMove`:
usage: `xMove (h, dx)`
purpose: Moves the object whose handle is `h` by `dx` pixels horizontally
(`dx` may be either positive or negative)

`yMove`:
usage: `yMove (h, dy)`
purpose: Moves the object whose handle is `h` by `dy` pixels vertically
(`dy` may be either positive or negative)

`getCenter`:
usage: `[x y] = getCenter (h)`
purpose: Returns the coordinates, in pixels, of the center of the object whose handle is `h`

Figure 2: a sample of the Pong game project functions, and their descriptions

In implementing the Pong game, students must make proper use of the programming fundamentals they learn. For instance, in determining when the ball hits a wall or a paddle, students need to have a good understanding of the `if` statement. In creating the animation, in continuing the game, loops must be used correctly. An implementation of the game where the program keeps statistics of players, or one with multiple balls, would necessitate usage of arrays.

Students also have to use their math skills to solve some of the problems. For example, in determining the incremental *x* and *y* displacements of a ball bouncing off a paddle (the requirements document specifies the bounce angles based on the point of impact), they

must use math to make sure that the ball's path conforms to the requirement, and to keep the ball at a constant speed.

And all throughout the game's development, many of the complex details are hidden from the students, and intentionally so. The level of abstraction at which students work is not at the low level of details of Matlab's graphics functions, nor at a high level of a pseudocode, a flowchart, or a UML implementation. It is at a level where students can understand what is going on, where their focus is totally on the project, and where they can concentrate on developing their programming skills and learn the fundamentals that we want our Freshmen engineers to grasp. Furthermore, students see a direct correlation between what they write and what they see. This latter point is essential as it gives students direct immediate feedback.

Figure 3 below shows a simple example where students need to use their `if`'s and their loops in order to animate a ball that bounces off the four walls of a window. The ball angles in this example are simply ± 45°, in either direction. The students develop this code early in the semester, at a time when they are getting comfortable with the project functions, and where they are learning to apply their knowledge of selection and loop structures by solving a basic problem. The animation their program produces gives them direct visual confirmation that they have correctly applied the programming structures that they have learned.

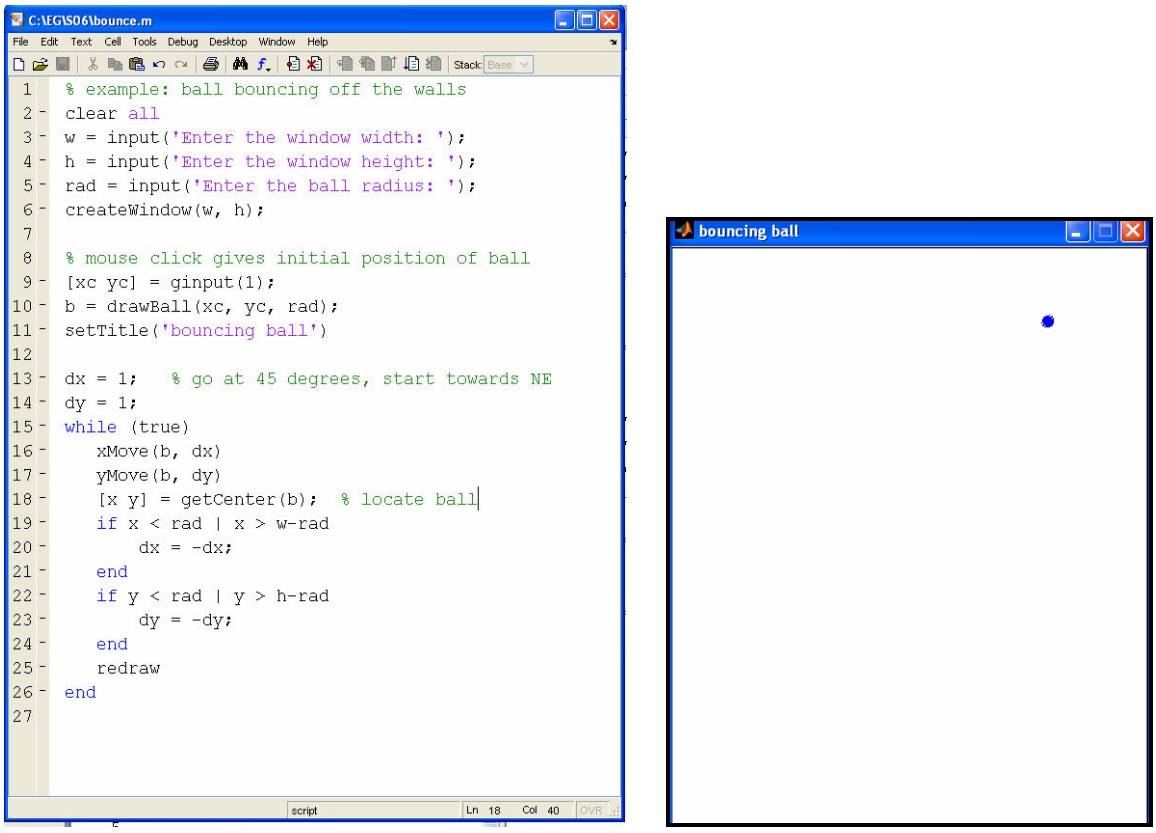

Figure 3: sample code to generate a ball that bounces off the walls, and its output

Choice of Matlab

Several colleges have used Matlab in their introductory programming courses[4,5]; others have developed a course that uses fun animation[6,7]. Our approach does the fun animation in Matlab, and students create a video game in the process. Unlike other approaches, the graphics here are simple and there are no extra hardware devices needed. Our method also intentionally shies away from including any engineering topics in the project, mainly because the module's main focus is the development of programming skills. The author used this approach several years ago in the freshman course that was the predecessor to the current two-course sequence, and to great success (back then students wrote in C, used Solaris machines, and wrapper functions were written for the X window system). When the decision was made around three years ago that one of the modules of the new engineering two-course sequence needed to be a programming module, the author adapted the previous approach to fit the current situation. Many changes had to be made to that adaptation however, mainly due to the new course objectives of the current two-course offering, to the fact that Matlab was the language of choice, and to the realization that the pedagogical paradigms for teaching programming skills to engineering Freshmen have changed since the late 1990's.

Matlab was chosen as a compromise, given that several of our engineering departments, though not all, require that their students master that language/tool. Departments that do not use Matlab have their own courses at the Sophomore level where their language of choice is taught. Our main objective here is for students who are going through the programming module to develop the proper fundamental programming skills, which are common to most languages. So whereas a computer scientist will most likely disagree with the choice of Matlab for teaching a first computer language, the intent for our Freshmen engineering courses is to provide students with enough programming skills, mostly at the fundamental level, that will allow them to solve engineering problems. Many of those skills will carry over to other languages.

Conclusion

The response from students has been very positive and enthusiastic. It is always a pleasure for faculty to see their students have fun when working on the projects. The very nature of a computer game is precisely based on fun, which provides an environment where students can learn in a more relaxed atmosphere. For a situation such as the one presented in this paper (large class, Freshmen, very diverse in their skills level), the environment must also include tools that are easily accessible to the students. By providing them with some wrapper functions, a whole world of animation and computer games can be accessible to students, without any increase in complexity, leading to a very effective method to teach the programming fundamentals.

Whereas the approach described above has been used on numerous occasions and in many institutions, we believe that it has rarely been done in Matlab. Our experience with using this method the past three years is very encouraging.

References

1. Brockman, J., Fuja, T. Batill, S., "A Multidisciplinary Course Sequence for First-Year Engineering Students," *2002 ASEE Annual Conference and Exposition*, Montreal, Quebec, Canada, June 2002.

2. McWilliams, L., Silliman, S., Pieronek, C. "Modifications to a Freshman Engineering Course Based on Student Feedback," *2004 ASEE Annual Conference and Exposition*, Salt Lake City, Utah, June 2004.

3. Kuncicky, D., *Matlab Programming*, Prentice-Hall, 2003.

4. Herniter, M., Pangasa, R., Scott, D., "Teaching Programming Skills with Matlab", *2001 ASEE Annual Conference and Exposition*, Albuquerque, New Mexico, June 2001.

5. Azemi, A., "Using Matlab to Teach the Introductory Computer-Progamming Course for Engineers, *2004 ASEE Annual Conference and Exposition*, Salt Lake City, Utah, June 2004.

6. Cole, W., Everbach, E., McKnight, S., Ruane, M., Tadmor, G., "Teaching Computers to Engineering Freshmen Through a 'High-Tech Tools and Toys Laboratory' ", *2001 ASEE Annual Conference and Exposition*, Albuquerque, New Mexico, June 2001.

7. Litkouhi, B., Naraghi, M., "And Effective Approach for Teaching Computer Programming to Freshman Engineering Students, *2001 ASEE Annual Conference and Exposition*, Albuquerque, New Mexico, June 2001.