Adding Breadth to CS1 and CS2 Courses Through Visual and Interactive Programming Projects

Ricardo Jiménez-Peris
Universidad Politécnica de Madrid
Facultad de Informática
208060 Madrid, Spain
rjimenez@fi.upm.es

Sami Khuri
San José State University
Dept. of Mathematics and Computer Science
San José, CA 95192, USA
khuri@cs.sjsu.edu

Marta Patiño-Martínez
Universidad Politécnica de Madrid
Facultad de Informática
208060 Madrid, Spain
mpatino@fi.upm.es

Abstract

The aim of programming projects in CS1/CS2 is to put in practice concepts and techniques learnt during lectures. Programming projects serve a dual purpose: first, the students get to practice the programming concepts taught in class, and second, they are introduced to an array of topics that they will cover later in their computer science education.

In this work, we present programming projects we have successfully used in CS1/CS2. These topics have added breadth to CS1/CS2 as well as whetted our students' appetite by exposing them to concurrent programming, event-driven programming, graphics management and human-computer interfaces, data compression, image processing and genetic algorithms.

We also include the background material, such as tools and libraries we have provided our students to render the more difficult projects amenable to our introductory computer science classes.

1. Introduction

One of the challenges that CS1/CS2 instructors are faced with is to motivate students to work on the programming concepts they have learned during the lectures. Quite often, students are discouraged because of the nature of the problems they are asked to implement. They fail to see the application of these programs in the real world. We believe that assigning challenging programming projects that capture their attention can solve the problem. In this paper, we propose a set of programming project topics aimed at motivating students in some attractive areas of computer science. These projects serve a dual purpose. First, the students get to practice the programming concepts taught in class, and second, they are introduced to an array of topics that they will cover later in their computer science education.

Imbedding the CS1/CS2 curricula with additional topics that students will be studying later in their computer science education is not new. Holmes et al. [5] for instance, focus on the breadth of topics to which computability might apply by incorporating concept tutorials on various topics such as security, error-correcting codes, and system analysis, in their introductory courses.

Using the implementation of games to motivate CS1 students is also not new. Pargas et al. [7] use the game of Nim to make the students learn "how to develop a strategy, decide on the best structure and algorithms to implement the strategy, and refine the design and implementation through testing". Similarly, Adams [1] uses a 2-person game, "Chance-It" as a project to help students understand object-oriented concepts. We agree with his findings that a carefully devised game motivates students and providing them with code framework gives them a good starting point.

Some of the projects we introduce are, at first glance, very challenging. We show how the difficulties can be overcome by having the instructor provide some background material that alleviates some of the difficulties associated with the projects. Depending on the project, the instructor's assistance might consist in providing tools, or libraries of functions that can get the students started on the projects. We also describe the concepts the students will be practicing and the major benefits they will get from the proposed projects.

Other works in the literature that share the same goal as ours include Robergé [8] who presents a set of programming projects aimed to have a visual impact so as to motivate students. His work does not have a graphical user interface and is not interactive, while our projects are. Fell et al. [3,4] and Astrachan et al. [2] describe a set of programming projects in the area of image enhancement

SIGCSE '99 3/99 New Orleans, LA, USA

and data compression. We propose two projects in the same area, but the projects themselves are very different.

The paper is organized as follows. Section 2 presents a framework to support interactive games as programming projects solving concurrency and graphic management. Section 3 introduces "table" games appropriate for data structures courses, and describes how to support mouse based interfaces. Section 4 presents elementary image treatment, including quadtree compression, and Section 5 uses genetic algorithms. Our concluding remarks are in Section 6.

2. Object-Oriented and Concurrent Programming with Interactive Games

Video game programming is one of the most motivating topics for computer science students. Interactive games can be used as software design programming projects. These projects can be solved with ADTs (abstract data types) or object-oriented design where students can practice these techniques. Program design is difficult to teach because it is a very abstract topic based on heuristics learnt from experience. The kind of games presented in this section consists of a collection of interacting objects. This fact eases the task of identifying design abstractions in ADT as well as in object-oriented frameworks. The student is faced with the task of choosing the right design from many different ones.

The first challenge is concurrency management; interactive games must simultaneously move several objects and respond to input from the keyboard. From our experience, introductory students usually cannot handle concurrent programming. We provide them with a library of simple functions and a skeleton of the main loop. Two primitives are needed to control the keyboard and two more to make automatic movements:

- KeyPressed() : Boolean that tells whether a key has been pressed or not.
- ReadKey(var key : Char) reads a key without echo.
- TimeElapsed(): Boolean that returns true if the countdown has finished.
- StartCountdown(count : Integer) registers the current time and the duration of the countdown.

The first two functions are available in most compilers. The last two can be readily built from a "get time" function.

What follows is the skeleton of the loop that simultaneously moves objects and reads pressed keys from the keyboard:

```
while not TimeElapsed () & NOT KeyPressed() Do
        (* Do Nothing *)
end; (* while *)
if KeyPressed() then ReadKey(key)
        (* Process the read key *)
end; (* if *)
if TimeElapsed () then StartCountDown(time);
        (* Automatic Movement *)
end (* if *)
```

The empty loop tests if a key has been entered or if the time between two successive movements has elapsed. When at least one of these events happens, the loop ends and the events are treated. Upon entering a key, it must be read and the associated action performed. When the countdown finishes, it must be restarted and the automatic movement performed.

The second problem to be solved is the graphics management. Some games are more amenable to ASCII text representation, in which case it is sufficient to provide a function that displays a character with a particular color at a given screen position. To facilitate graphic management, we provide the students with elementary functions, such as starting the graphic mode, returning to text mode, and drawing a pixel in some color in a given screen position. In some programming projects, it is also useful to provide primitives to display either simple graphic elements that allow constructing the required figures, or higher level primitives that display a complete element of the game.

In the next section, we introduce games we have successfully used over the years as CS1/CS2 projects.

2.1 Asteroids

This game consists of a set of asteroids (circles) and a spaceship (triangle). The user controls the spaceship and must destroy all the asteroids without crashing.

All asteroids are initially of the same size, but when hit, the asteroid splits into two pieces of smaller sizes (Figure 1). After multiple hits, the asteroid size reaches a certain threshold and disintegrates. Each asteroid has a speed and a direction, which change when it is hit. The direction after the hit changes by +/- 45°.

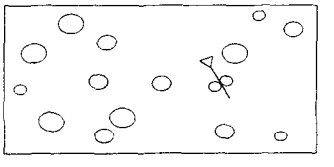

Figure 1. Spaceship splitting an asteroid

The user can control the spaceship's direction (degrees) and speed (measured in pixels per second). The shot range is also measured in pixels.

Through this project, students develop skills in ADT and object-oriented design. Its adequacy for object-oriented design stems from the fact that:

- The three objects (asteroids, spaceships and shots) have many similarities that can be exploited with the use of inheritance (they are mobile objects with different peculiarities).

- The collection of all the objects (the three types) can be stored in a polymorphic list. It will have to make use of dynamic binding.

2.2 Tetris

Tetris consists of a rectangular board and various pieces, as seen in Figure 2, which fall from the top. The pieces can be rotated 90° and shifted horizontally while they fall. This project can be displayed in text or graphic mode. In text mode, the block char "■" can be used as a building block for the pieces. The board can be drawn with semi-graphic characters. In graphic mode, any simple graphic library will suffice.

This constitutes a good programming project to practice program design. In an object-oriented framework, each piece can be implemented as a class inheriting from a root abstract class. The interface between the board and the pieces is an interesting problem that can be solved with different approaches, depending on the landing surface of a falling piece.

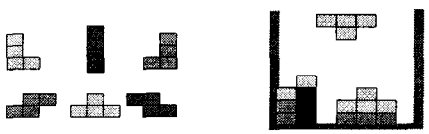

Figure 2. Tetris pieces and game snapshot

The game offers a wide number of possible additions and variations. Score accumulation can be added. After a predefined score, the speed of falling pieces could be increased. With different speeds, an initial configuration with some pieces already at the bottom of the board could be loaded from a file. A help feature showing the next piece can be added to the project.

We have found that this project tests the students' knowledge of loops, array management and configuration file management for the different scenarios associated with each speed.

3. Event-Oriented Programming with "Table" Games

We have used "table" games as programming projects in CS2. They usually require a more intensive manipulation of data structures than the games of the previous section. For example, card games usually require heaps, queues, double-ended queues and variations of these data structures. The graphic user interface and the interaction between the layer and the game pose some challenges for these projects.

We provide the students with primitives to display each card in a card game. The interface can be performed through the keyboard and/or the mouse. The keyboard primitives of the previous section can also be used here. Additional primitives needed are:

- GetMousePosition(var x: Integer; var x: Integer). Returns current mouse pointer.
- IsMouseButtonPressed(LeftButton: Boolean). Determines whether a mouse button has been pressed or not.
- ShowMousePointerAt(cursor: KindOfCursor, x, y: Integer). Shows a mouse pointer at (x, y), deleting the previous mouse pointer. KindOfCursor can be an enumerated type providing the different shapes available for the mouse pointer.

Through these games, we teach the fundamentals of user interfaces, such as drag and drop cards, click and event-oriented programming. We use a control loop similar to the one presented in Section 2, for a drag and drop interface:

```
GetMousePosition(x, y)
while not IsMouseButtonPressed(true) do
  GetMousePosition(x, y)
  ShowMousePointerAt(normal, x, y)
end; (* while *)
if "is any object at (x, y) to drag" then
  while IsMouseButtonPressed(true) do
    GetMousePosition(newX, newY);
    ShowMousePointerAt(dragging, newX, newY)
    if "object at (x, y) can be dropped at (newX, newY)"
      then
        "Move object at (x, y) to (newX, newY)"
      else "Show error message"
    end (* if *)
  end; (* while *)
end (* if *)
```

The card game solitaire, for instance, provides an excellent source for programming projects. Students can practice ADT or object-oriented design and implementation. The visually attractive display of cards is achieved by providing a primitive that displays a card at a particular position with the face up or down. The image of the cards can be captured from Windows' game "Solitaire", by copying the window content to the clipboard and then editing and saving it in bmp format. Other card games, and non-card games, such as domino, and Wari (a two-player board game from Egypt), can be implemented by using our outlined drag and drop interface.

4. Edge Enhancement and Quadtree Image Compression

One of the most attractive areas of computer science is the manipulation of images. This is due to its graphic appearance and its interesting applications in medicine, spatial missions, etc. In this section, we present two projects that involve simple enhancement algorithms and image compression. These projects allow students to

practice loops, one and two dimensional arrays, files, complex dynamic data structures (quadtrees), etc. CS1/CS2 students will need some assistance to read image files and display them.

4.1 Edge Enhancement Algorithms

Image enhancement is the process of applying techniques to emphasize and sharpen image features for display and analysis. Contrast enhancing and histogram equalization [3] are simple enough to be covered in CS1/CS2 courses. Edge enhancement algorithms also,, could be easily tackled in introductory computer science classes.

Image edge enhancement reduces an image and focuses only on the edge details of the image (see Figure 3). It is often used as a preprocessing step where the edge outlines of objects within the image are subsequently used for feature or object recognition. Edge detection is based on the relationship a pixel has with its neighboring pixels. An edge is defined by the discontinuity in gray-level values. For example, to detect vertical edges, we can shift an image to the left by one pixel and then subtract it from the original image. Two adjacent pixels with very different brightness values will yield a very large brightness value, while neighboring pixels with very similar shades of gray will yield a small brightness value. The shifting and difference method is basically achieved by manipulating 3x3 arrays. This project allows the students to learn two-dimensional array manipulation, sequential file reading and writing, loops, etc.

4.2 Quadtree Image Compression

Reading and manipulating images in various formats, such as tiff and jpeg, is too difficult for introductory students. Our approach has been to write a program that reads ms-windows bitmap files (bmp) and to use a multi-format graphic converter (there are many free ones on the web) to transform images from different formats to bmp. A sample bmp file reader can be found in [9] with the source code included in a floppy disk. In this section, we concentrate on the compression of bitmap images only.

Figure 3. Original and edge enhanced images

Compressing bitmap images can be achieved by using quadtrees: where a node either is a leaf or has exactly four children. The compression paradigm is based on the assumption that if a randomly selected pixel in the image is black let's say, then there is a good chance that its immediate neighbors are also black. The compression algorithm thus scans the bitmap image looking for areas full of pixels with the same color.

The compression algorithm takes as input a bitmap image and outputs a quadtree. The complexity of the bitmap image will affect the size of the tree. The top-down construction first builds a root node. The bitmap image is divided into four quadrants, each corresponding to a child of the root node. Thus, the leftmost child of the root node in Figures 5 and 6 correspond to the block of 4x4 pixels in the upper left quadrant of Figure 4, while the rightmost child of the root node corresponds to the block of 4x4 pixels in the upper right quadrant of Figure 4.

The compression algorithm then checks each quadrant:

- Any quadrant that consists of pixels of the same color becomes a leaf node with that color.
- Any quadrant whose pixels are not all of the same color is recursively divided into four smaller subquadrants.

The algorithm stops when there are no more quadrants of different values.

The 8x8 bitmap image of Figure 4 is compressed into the 57-node quadtree of Figure 6. Forty-three nodes are leaves (9x4=36 at the lowest level and seven at the level before the last) and the other fourteen are interior nodes that contain pointers.

The actual quadtree produced by the compression algorithm is represented in Figure 6. Note that the quadrant numbering used in all figures i$\begin{pmatrix} 1 & 4 \\ 2 & 3 \end{pmatrix}$.

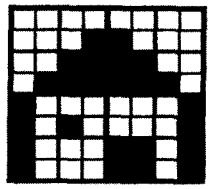

Figure 4. Image before compression

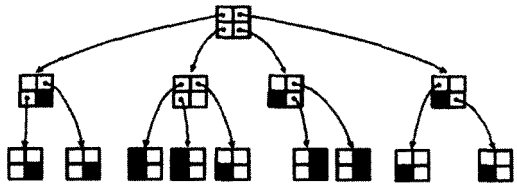

Figure 5. Quadtree representing the compressed image

As mentioned in the introduction of this section, students can be given either bitmap images to test their programs or images in other forms with the bitmap converter that will produce bmp files. So the bitmap converter can be used as a preprocessing step to convert the given image in bitmap form after which they can use their programs to perform the compression, which in essence translates into constructing the quadtree.

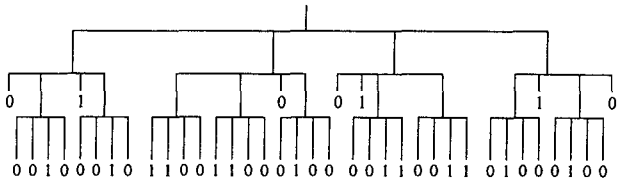

Figure 6. Quadtree implementation of Figure 5

Benefits that students can get from this project, besides being introduced to data compression, include:

- Implementing the recursive, top-down compression algorithm mentioned earlier that takes as input a bitmap image, similar to Figure 4, and produces a quadtree, similar to Figure 6.
- Implementing the decompression algorithm that produces a bitmap image from a quadtree.
- Writing a non-recursive algorithm that essentially starts by constructing a complete quadtree, thus assuming that no quadrant consists of the same color and then checks the assumption. Whenever the assumption is false and a quadrant consists of the same color, the four nodes are deleted from the tree and their parent becomes a leaf. The process is bottom-up, starting from the leaves and ending at the root node.

In summary, through the above compression, students are exposed to the recursive top-down construction of non-binary dynamic trees, which necessitate the manipulation of interesting data structures. They will also be introduced to the bottom-up compression algorithm. A comparison of both approaches is valuable, since top-down and bottom-up construction of trees are found in many other fields of computer science, such as the parsing phase in the compiling process, the testing stage in software engineering and in dynamic programming.

5. Genetic Algorithms

Information-processing systems that have certain performance characteristics that mimic biology, such as neural networks and evolutionary computation, have gained popularity. Genetic algorithms belong to the second category and are a very good candidate for CS1/CS2 projects. They are general search algorithms that perform best for problems with very large search space and for which more traditional methods have failed to find optimal solutions. It would be very challenging to have the students implement a genetic algorithm. Instead, we have used a software package, LibGA [6], for our projects. It is a library of routines, in which the various genetic operators for reproduction, crossover, and mutation, can be set and changed by using a configuration file, thus eliminating the need of recompiling. It also offers a user-friendly interface.

To use LibGA, one has to encode the problem at hand by using an appropriate representation (binary strings most often), and a fitness function that computes the "strength" of each string in the population. We discuss the representation and fitness function in class, and then ask the students to do the implementation and perform experimental runs with LibGA.

Besides being exposed to non-traditional algorithms, optimization problems, such as knapsack and scheduling problems, arrays, loops, file I/O, and some elementary probability theory, the students learn how to incorporate their own code in an existing software package. This is a very valuable experience since they will be faced with similar situations in their academic studies as well as later in their jobs.

6. Conclusion

We present a set of topics we have used as a source for visual and interactive projects to motivate our CS1/CS2 students. We introduce several sample-programming projects in each area, and the background material we provide to assist the students in their projects. The background material makes our more challenging projects amenable to CS1/CS2 students. The programming projects presented in this paper have allowed our students to practice the programming concepts taught in class and have introduced an array of topics that they will encounter later in their computer science education.

7. References

1. Adams, J. C. Chance-It: An Object-Oriented Capstone Project for CS1. *Proceedings of SIGCSE'98*, 1998, pp.10-14.
2. Astrachan, O. and Rodger, S. Animation, Visualization, and Interaction in CS1 Assignments. *Proceedings of SIGCSE'98*, 1998, pp. 317-321.
3. Fell, H. J. and Proulx V. K. Exploring Martian Planetary Images. C++ Exercises for CS1. *Proceedings of SIGCSE'97*, 1997, pp. 30-34.
4. Fell, H. J., Proulx V. K. and Rasala, R. Scaling: A Design Pattern in Introductory Computer Science Courses. *Proceedings of SIGCSE'98*, 1998, pp. 326-330.
5. Holmes, G. and Smith, T. C. Adding Some Spice to CS1 Curricula. *Proceedings of SIGCSE'97*, 1997, pp. 204-208.
6. LibGA: http://euler.mcs.utulsa.edu/~corcoran/libga.html
7. Pargas, R. P., Underwood, J. N. and Lundy, J. C. Tournament Play in CS1. *Proceedings of SIGCSE'97*. 1997, pp. 214-218.
8. Robergé, J. Creating Programming Projects with Visual Impact. *Proceedings of SIGCSE'92*. 1992, pp. 230-234.
9. Swan, T. Borland Pascal 7.0 *Programming for Windows*. Borland Bantam, 1993.