AUTHOR	Skolnick, Michael M.; Spooner, David L.
TITLE	Graphical User Interface Programming in Introductory Computer Science.
SPONS AGENCY	National Science Foundation, Arlington, VA.
PUB DATE	95
CONTRACT	DUE-9354641
NOTE	9p.; In: "Emerging Technologies, Lifelong Learning, NECC '95"; see IR 017 705.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Conference Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	Computer Games; Computer Graphics; Computer Interfaces; *Computer Science Education; Higher Edu·:ation; *Introductory Courses; *Programming
IDENTIFIERS	*Graphical User Interfaces; *Macro Graphic System; Rensselaer Polytechnic Institute NY

ABSTRACT

Modern computing systems exploit graphical user interfaces for interaction with users; as a result, introductory computer science courses must begin to teach the principles underlying such interfaces. This paper presents an approach to graphical user interface (GUI) implementation that is simple enough for beginning students to understand, yet rich enough to demonstrate many important aspects of computer science. The GUI interface described is implemented using a library of C macros and provides a display window that outputs bit-mapped graphics and inputs mouse actions. The macro calls and conventions are described in the context of an implementation of Conway's Game of Life, a programming exercise found in many introductory texts. The GUI interface for the Game of Life program is detailed in the first section, in order to provide a concrete example of the capabilities of the simplified GUI library. The next section describes the life main function that sets up the buttons and associated callback functions; these callback functions are then presented. The last section gives an example of a simple paint program tha' can also be created using the GUI macro library. In addition, this paper considers how the material could be introduced in an Introduction to Computer Science course at Rensselaer Polytechnic Institute (New York). (AEF)

Graphical User Interface Programming in Introductory Computer Science

Paper presented at the NECC '95, the Annual National Educational Computing Conference (16th, Baltimore, MD, June 17-19, 1995.

paper

Graphical User Interface Programming in Introductory Computer Science

Michael M. Skolnick
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
(518) 276-6912
skolnick@cs.rpi.edu

David L. Spooner
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
(518) 276-6890
spoonerd@cs.rpi.edu

Key words: introductory computer science, programming, event-loops, graphical user interface

Abstract

Modern computing systems exploit graphical user interfaces for interaction with users. As a result, introductory computer science courses must begin to teach the principles underlying such interfaces. This paper presents an approach to graphical user interface implementation that is simple enough for beginning students to understand, yet rich enough to demonstrate many important concepts and trade-offs in computer science.

Introduction

A major problem confronting teachers of introductory computer science courses is that, while most computational environments that students are exposed to involve graphical user interfaces (GUIs), the I/O programming paradigm that they are typically introduced to is character based. The irony is that students are using and enjoying the flexibility of GUIs but from a programming point of view they could just as well be using teletype machines. It is clearly no longer acceptable to give students programming exercises where graphical information is output as lines of ASCII characters. Further, students restricted to character-based input quickly become bound up in the intricacies of awkward command languages, when a GUI interface would be more natural. At the same time, there are valid reasons for hesitating to teach GUI programming at the introductory level. Foremost is the additional complexity involved in doing event-loop programming, typically considered too difficult for the novice programmer.

While a strong case can be made for introducing GUI programming techniques into introductory computer science courses, this is not being done. The only introductory Computer Science text (of which we are aware) that integrates bit-mapped graphics is Roberts (1995). By including a graphics.h file in the C code, the programmer has access to a set of basic functions for producing bit-mapped output, e.g., MovePen, DrawLine, DrawArc. We are not aware of any introductory text (or course) that attempts to introduce both GUI input and output.

This paper will describe a simplified GUI interface that is implemented using a library of C macros and provides a display window that outputs bit-mapped graphics and inputs mouse actions. The input mouse actions can be used in a drawing framework and for the control of some (user specified) number of input buttons. The library of macros supports the MOTIF GUI interface standard, but could easily be converted to support other GUI standards. The macro calls and conventions will be described in the context of an implementation of Conway's Game of Life (Conway, 1970), a programming exercise found in many introductory texts. The macro library has been used in teaching the second computer science course at Rensselaer. The positive reactions of students have indicated to us that introducing this library to students in the course Introduction to Computer Science (CS1) would enhance this course as well. This paper will describe how this material could be introduced in CS1.

The GUI interface for the Game of Life program will be described in the first section, in order to provide a concrete example of the various capabilities of the simplified GUI library. The next section describes the life main() function that sets up the buttons and associated callback functions. The callback functions (and macro conventions for calling them) that are at the heart of the programming interface support are presented in the next section. The last section gives an example of a simple paint program that can also he creating using the GUI macro library. In all of these sections we consider how this material should be presented to CS1 students.

The Game of Life and its GUI Interface

In general, the GUI interface (as supported by the macro library) provides two regions of interaction with the program: a vertical strip of user definable buttons on the left and a larger display/drawing region to the right. For example, when the Game of Life program is invoked a window like that shown in Figure 1 appears on the screen. In the Game of Life, the graphical display/ drawing region may initially be all black, indicating that there are no live cells. The user can then drag the mouse in this region in order to draw in the cells (in white) that are considered to be alive. The Game of Life transition can now be applied to the initial array of live cells.

Basically, the Game of Life transition embodies simple rules of population support and crowding. For example, live cells stay alive if they are "sustained" by "just the right" (2 or 3) immediate neighboring cells that are alive, but if a live cell is crowded (4 or more neighbors) or too isolated (0 or 1 neighbors) then it dies. Similarly, cells can come to life if they have just the right support in their neighborhoods. What is interesting about this simple "game" is that a variety of stable configurations of "cooperating" cells tend to emerge, when successive transitions are applied to initial random configurations.

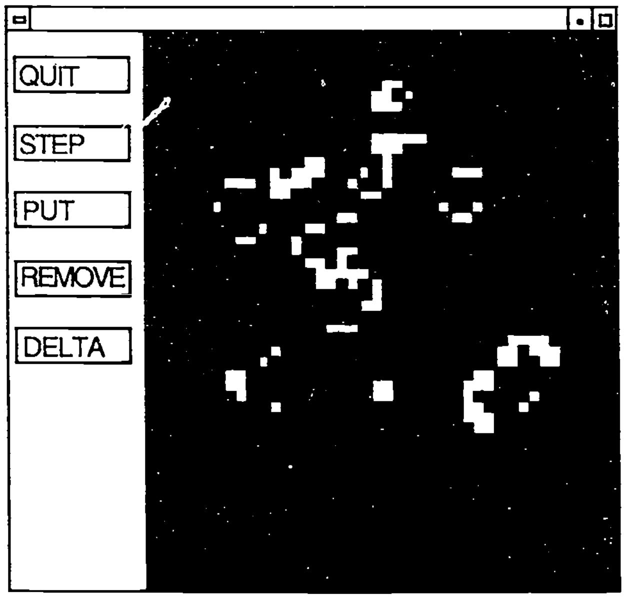

Figure 1. User initialized state of the GUI Game of Life interface

The user moves back and forth between changing states in the array using the mouse and clicking the "STEP" button to move the transition function ahead. The "PUT" and "REMOVE" buttons are used to toggle the effect of dragging the mouse across the graphics window: the "PUT" button sets a global variable that results in mouse movements bringing dead cells to life, while the "REMOVE" button sets the global variable so that live cells are removed as the mouse is dragged across the graphics window. The "DELTA" button causes program control to return to the standard input UNIX window, where the user is queried as to the number of generations to be computed whenever the "STEP" button is pressed. The default number of iterations is one. By setting the number of iterations to a value other than the default, the user creates the effect of animating the Game of Life transitions. Finally, the "QUIT" button is automatically supplied by the macro library and provides a means of exiting the program via the GUI interface (a control-C in the UNIX text window can also be used to exit the program).

When the Game of Life program is invoked the user may specify (via command line arguments) a file that initializes the array of live cells. In addition, the user can specify from standard input a sequence of coordinates of initially alive cells. The third mode of input (as described above) is via the mouse in the display/drawing region. We feel that the students benefit from exposure to all three modes of input. They need to see that a natural way to input cell states is via a mouse, but that sometimes a larger or algorithmically generated ASCII input file is more appropriate (e.g., as might be generated via a random initialization program) or that sometimes entering a few specific coordinates of alive cells is more natural (e.g., during debugging). We introduce more on UNIX filters and pipes than is typically covered in CS1 and compensate by covering named file I/O somewhat less. In addition, while pointers are not usually covered in CS1 we introduce them in the context of processing command line parameters (via argv and argc); that is, pointers are introduced solely for simple string manipulation, e.g., the students learn that argv[1] points to the string of characters comprising the first command line argument, which in turn makes a fine argument to fopen. We view it as an important general intellectual lesson that students appreciate how limiting I/O to a serial stream of characters is a strong intellectual restriction and, as such, at times is a significant limitation and at other times is a useful simplification of complexity. Thus, we are careful to ensure that students come away with some idea of key I/O trade-offs, as opposed to thinking that a GUI is always the best option or conversely that ASCII (standard or file) input is the only option. Finally, throughout the computation there is the option of communicating with the user via a stream of ASCII characters. This allowed us to choose a "minimal" set of GUI primitives to support via the macro library: a vertical array of buttons on the left and a display/drawing region to the right. With the capability to communicate with a user via buttons, a display/drawing region and standard I/O window we believe that there is sufficient coverage of the basic modalities of modern I/O interaction.

One reason that the Game of Life is often given as a programming example in many introductory texts is that it is inherently fascinating. At the same time, we have argued that a lack of true GUI programming makes interacting with one's program a frustrating experience. Beyond this issue, we often find that too many of the programming exercises in introductory texts are given in isolation and precious little is given concerning the larger scientific issues. Thus, students leave the isolated Game of Life exercise with a sense that it is "neat," but little more. In the CS1 course that we are teaching, we are approaching the teaching of this material via an integration with extensive case studies. The GUI toolkit is part of a larger project to develop an online hypermedia textbook (Skolnick & Spooner 1994) for introductory computer science courses for both majors and non-majors. This hypermedia textbook combines science and engineering case studies with workstation-based computing concepts and tools to provide students with the computing knowledge they need to successfully apply computing in their own disciplines. In the case of the Game of Life, it is a simple exercise to consider other variations on neighborhood transition functions, e.g., parity, min, average, max. This allows us to consider how these kinds of neighborhood cellular array computations are being used in areas such as image processing and as alternative scientific models of physical systems, e.g., in Toffoli and Margolus (1987) various kinds of neighborhood transitions are used to model basic thermodynamic systems. We can also approach more profound issues related to the nature of computation itself, i.e., when considering how to generalize the neighborhood transition function it is natural to consider the class of all such boolean functions, which leads to such issues as combinatorial explosion, search spaces, computational completeness and hardware boolean lookup tables.

We now consider how the GUI functionality is implemented at the programming level and will discuss the additional concepts that need to be introduced in CS1 to allow students to understand the basics of event-loop programming.

main() routine in GUI Life

Before considering the details of the code, it is useful to describe how the basic ideas of event-loop programming are presented to the students. The students are told that the event-loop program can be viewed as a mini-operating system, which handles much of the tedious details of the GUI interface. When a programmer wants to define a button, a function call must be made to the event-loop program. Certain essential information about the button must be passed to the event-loop program: the name to appear on the button and the name of the callback function to be invoked when the button is pressed. The event loop places the buttons onto the screen, detects which button is being pressed, graphically represents button presses and, most important to the programmer, transfers control to the appropriate callback function to process the actions associated with the button. Students are told that there are many different types of interface objects (widgets) that can be created, but we have provided a simple default class of such objects. They need only visualize the event-loop program as something they communicate with initially from within their main() routine and then the final action within main() is to transfer control to the event-loop, which will then be responsible for invoking the callback functions (defined by the programmer) in response to user actions.

How these issues are handled at the programming level can be seen in the main() program in Figure 2. All the Motif macro definitions and related global variables are defined in the main_gui.h include file. In the figure, macro calls are indicated by all upper case names and the absence of semi-colons at the end of the line. Since we use ANSI C, the function prototypes of the four user defined button callback functions are setup using DEF_BUTTON_FUNCS. (Some example callback function definitions will be considered below.) The purpose of the DEF_BUTTON_FUNCS macro is to hide some of the messy details of the Motif library, e.g., the expansion of the macro will result in the following prototype definition for the step function, "void step(Widget, Widget, XmPushButtonCallbackStruct *);". In general, we believe that there is benefit in covering macros more thoroughly and earlier in a CS1 course than is typically done (e.g., beyond #defines), since macros can be used to introduce basic functionality early on, providing a useful foil for the more intricate explanations of true functions and parameter passing. After processing any command line arguments–in ParseArgs–the first macro call, SETUP_MOTIF_WIDGETS, generates in the main program all the appropriate general widget definitions for the GUI interface. The four calls to the CREATE_BUTTON macro result–from the students point of view–in the event loop having all the necessary information for managing the buttons, i.e., the names of the buttons and the associated callback functions. Next, the call to

CREATE_WINDOWS results in the GUI interface window appearing on the screen, with the four user defined buttons (and the default "quit" button) and an initially empty display/draw window to the right of the buttons. The InitWindow function is called to see if the user has specified (via command line arguments) a file that initializes the array of live cells.

```
#include  "life.h"
#include  "main_gui.h"

DEF_BUTTON_FUNCS(step,  put,
        remove,  delta)

void main(int argc, char *argv[])
{
   ParseArgs(&argc,  argv);

   SETUP_MOTIF_WIDGETS
   CREATE_BUTTON("STEP",step)
   CREATE_BUTTON("PUT",put)
   CREATE_BUTTON("REMOVE",remove)
   CREATE_BUTTON("DELTA",delta)
   CREATE_WINDOWS

   InitWindow();

   TRANSFER_TO_EVENT_LOOP
}
```

Figure 2. The main() function of the Game of Life program.

The final macro call in main() is TRANSFER_TO_EVENT_LOOP, which turns over control to the event loop. This movement away from control residing in the main() program is exhilarating to many students but can be confusing to others. We find that this transition is smoother if students are introduced to programming from the very start within the context of a symbolic debugger (which we tell the students to view as the symbolic "inspector"–our attempt to counter the "pathological" perspective that debuggers are only useful when things don't work). Thus, since students are familiar with the notions of successive refinement and "plug" functions, they can verify in the "inspector" that when a button is pressed the associated callback function is indeed invoked.

We now briefly consider button callback functions followed by the callback function associated with mouse events in the display/draw portion of the GUI window.

Callback Functions

The simplest class of callback functions are those associated with buttons, because there is no information that needs to be passed by the event loop to the function. The callback function is simply called when a button is pushed. One (of the four) Game of Life callback functions is given in Figure 3. Note again the use of macro substitutions to hide some of the messy and irrelevant Motif parameters, e.g., BUTTON_ARGS.

The step function (in Figure 3) does the main work of the computation, determining the next generation of alive and dead cells (ComputeNextGeneration) and then updating the current generation and displaying any altered cells (CopyNextToCurrent). The parameters passed to these two functions–current_gen and next_gen–are arrays containing the information on alive and dead cells. Information on the programming interface for displaying the cells is given below in our discussion of the callback routine for the display/draw component of the GUI interface.

```
   void step(BUTTON_ARGS)
/* computes and displays the next
   sequence of generations */
{
   extern int NumIterations;
   int i;

   for (i=0; i<NumIterations; i++) {
        ComputeNextGeneration(current_gen,
                next_gen);
        CopyNextToCurrent(next_gen,
                current_gen);
    }
},
```

Figure 3. Two of the four button callback functions.

We now turn to the display callback function associated with actions in the graphical display/draw window. The display/draw window is where the graphical information is displayed as well as where the mouse events (used to set cells to be alive or dead) are detected. The display callback function, which handles responses to mouse events, is shown in Figure 4. Students are expected to have a display_callback function defined somewhere in their source file (or files). Thus, it is a special reserved function name that currently is "hardwired" in the gui.h macro library. It is invoked whenever a mouse down, mouse drag or mouse up event is sensed by the event loop. Note also that the gui_externs.h file contains all external references to the GUI variables that are defined in main() and any GUI macros that might be invoked by any function.

We have provided several macro definitions that allow the programmer to access mouse event information. In the display_callback function for GUI life, MOUSE_X and MOUSE_Y result respectively in the current x and y coordinates of the mouse. One of the arguments passed to display_callback (and defined in DISPLAY_CALLBACK_ARGS) is "XButtonEvent *_event"; _event is a pointer to the relevant mouse information. Thus, MOUSE_X is defined as _event->x. These kinds of macro definitions allow postponement of detailed explanations of pointers to structures until students are ready; when they are ready for this material they enjoy unfolding the meaning of the code generated by the macros.

```
#include "life.h"
#include "life_externs"
#include "gui_externs.h"

void
display_callback(DISPLAY_CALLBACK_ARGS)
{
int x,y; /* up-left corner of cell */
int row, col;

x = MOUSE_X-(MOUSE_X % Pixels_per_cell);
y = MOUSE_Y-(MOUSE_Y % Pixels_per_cell);
row = y / Pixels_per_cell;
col = x / Pixels/per_cell;
if (DrawingCells) {
   XFillRectangle(DISPLAY_WIN, x, y,
      Pixels_per_cell, Pixels_per_cell);
   XFillRectangle(DISPLAY_BUF, x, y,
      Pixels_per_cell, Pixels_per_cell);
   current_gen[row][col] = ALIVE;
}
else { /* erasing cells */
   SET_FOREGROUND_BLACK
   XFillRectangle(DISPLAY_WIN, x, y,
      Pixels_per_cell, Pixels_per_cell);
   XFillRectangle(DISPLAY_BUF, x, y,
      Pixels_per_cell, Pixels_per_cell);
   SET_FOREGROUND_WHITE
   current_gen[row][col] = DEAD;
```

```
    }
    DISPLAY_WINDOW
    } /* end of display_callback */
```

Figure 4. The display_callback function for the game of life GUI.

The first two executable lines of code in display_callback (Figure 4) compute the x and y coordinates of the upper-rightmost pixel of the rectangular region that represents the cell. The next two lines in the code compute the row and column of the cell where the mouse was clicked. If the user has set the mode to DrawingCells, the mouse click will result in a white rectangle being drawn on the screen and current_gen array that records alive and dead cells being updated. There are two calls to XFillRectangle, which is the XWindow function that draws the live cell as a white rectangle. The two calls to XFillRectangle are required because it is necessary to support two display buffers: DISPLAY_WIN refers to the display buffer containing the actual information being displayed, while DISPLAY_BUF refers to a "backup" buffer typically containing the same information. The reason for the two buffers is to deal with the refresh problem in windowing systems: whenever the current information in DISPLAY_WIN is written over by another window, there must be a "backup" buffer which is used to restore the window information when the covered window is made active again. Introducing this complexity in supporting GUI programming is necessary but students can be initially told to always have two invocations of any graphics routine, one to each buffer. (We will consider below how this additional complexity has some advantages in other settings.) There seems to be little problem in the students appreciating that if they don't write to DISPLAY_BUF (in addition to DISPLAY_WIN) then they will loose the information in the GUI display/draw window when it gets covered by some other window. Also, all of the internal details of handling this refresh issue are implemented within the macro library, so the students need not be concerned with any further implementation details. Another point concerning the design of the macro library is that there is an advantage in not hiding the fact that XFillRectangle is being called, e.g., by hiding it in a macro call or even condensing the two calls to XFillRectangle into one macro. We want to avoid the danger of abusing macros by creating an idiosyncratic version of C. In addition, there is a significant advantage in introducing students to the level of XWindow calls because they can then go to the manuals and discover many other useful graphics routines, e.g., XDrawLines, XDrawPoints, etc. It seems to us that we need to prepare CS1 students with knowledge that allows them to enter into deeper levels of system documentation. CS1 texts and courses have tended to ignore this issue.

One other application: a paint program

With the basic mechanisms of the macro library behind us, we would like to consider one final programming project that now falls within the range of a CS1 course. Figure 5 shows the screen produced by a simple paint program. The structure of the program follows that of the Game of Life program, with calls to XDrawLine and XDrawRectangle being made within display_callback. The "RECT" and "FREE" buttons set a global variable that determines the effect of mouse actions in the display/draw window.

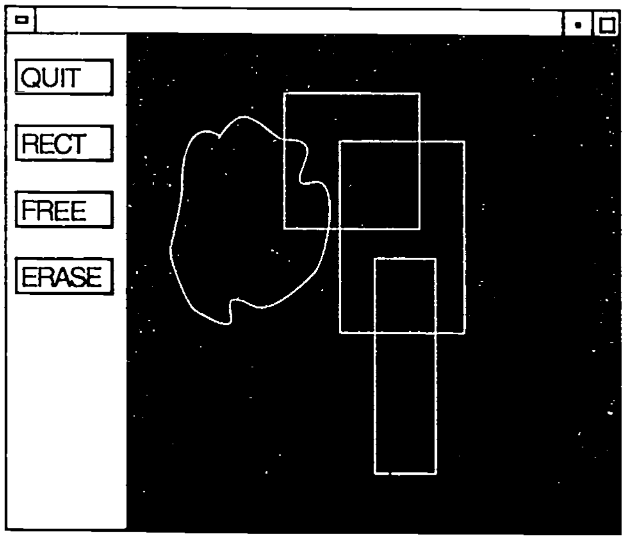

Figure 5. A sample screen of the paint program written using the macro library.

It is also worth noting that the double buffering scheme described above–involving DISPLAY_WIN and DISPLAY_BUF–has the advantage of allowing the animation of rectangle drawing motions, as well as more general animation effects. All students are expected to be able to program a simple non-animated draw function where, (1) an initial mouse down event (tested for via a MOUSE_DOWN macro predicate and associated with the left mouse button) positions the upper left corner of the rectangle, (2) mouse movement events (tested for via a MOUSE_MOVE macro predicate) result in no display changes, and (3) a final MOUSE_UP event (also tested for via a macro predicate) positions the lower right corner of the rectangle so that the rectangle can then be drawn with calls to XDrawRectangle. As a more difficult programming problem students may consider how to animate the draw, as is done in most drawing programs. This is accomplished by maintaining the state of the window prior to the mouse down event in DISPLAY_BUF and whenever a mouse movement is detected, DISPLAY_BUF is copied into DISPLAY_WIN and the rectangle defined by the current mouse position is drawn into DISPLAY_WIN. The effect is that as the mouse is dragged across the screen, the rectangle associated with the previous mouse position is erased and a new one is drawn. If the speed of buffer transfer is well matched to the speed at which the mouse is drawn across the screen, the effect is a smooth animation of the drawing. Students are truly amazed that such animation effects are within their programming skills and gain an appreciation of some basic tradeoffs between CPU power and the complexity of events that can be handled in a GUI interface.

Conclusion

In summary, the use of a simple GUI toolkit for teaching programming in an introductory computer science course offers several important advantages. It provides an exciting opportunity for students to exploit the newest functionality of modern computing technology, thereby increasing their interest and motivation to learn and experiment. It facilitates presentation of design and interface issues. And it allows students to begin to appreciate and understand the GUI components of the systems and application programs that they commonly use in other situations.

The GUI toolkit is available via anonymous ftp at "ftp.cs.rpi.edu". Use "anonymous" as the user name and your email address as the password. Once connected, change to the directory "pub/graphicslib". The "README" file in that directory contains the details of what is available and how to install and use it. In particular, the directory contains the macro library and sample code for the Game of Life and Paint programs.

Acknowledgments

This work was partially supported by the National Science Foundation, Grant Number DUE-9354641. Any opinions expressed or implied are those of the authors. Chapter 10 from Heller (1991) was where the authors learned about GUI code under MOTIF and the programs in this chapter served as the inspiration for the macro library.

References

Conway, J. H. (1970). The Game of Life. *Scientific American*, October 1970, 120.

Heller, D. (1991). *The Definitive Guides to the X Window System, Vol. 6: Motif Programming Manual*, Sebastopol, California: O'Reilly and Associates, Incorporated.

Roberts, E. (1995). *The Art and Science of C: An Introduction to Computer Science*. Reading: Addison-Wesley Publishing Company.

Skolnick, M. M. & Spooner, D. L. (1994). *Hypermedia Textbook for Computer Science I: A Case Studies Approach*. Technical Report 94-20. Troy, New York: Computer Science Department, Rensselaer Polytechnic Institute.

Toffoli, T. & Margolus, N. (1897). *Cellular Automata Machines: A New Environment for Modeling*. Cambridge: MIT Press.