Programming Text Windows with ncurses

by Jim Hall

How to use ncurses to manipulate your terminal screen.

In my article series about programming for the text console using the ncurses library, I showed you how to draw text on the screen and use basic text attributes. My examples of Sierpinski's Triangle (see "Getting Started with ncurses") and a simple Quest adventure game (see "Creating an Adventure Game in the Terminal with ncurses") used the entire screen at once.

But what if it makes more sense to divide the screen into portions? For example, the adventure game might divide the screen to use part of it for the game map and another portion of the screen for the player's status. Many programs organize the screen into multiple parts—for instance, the Emacs editor uses an editing pane, a status bar and a command bar. You might need to divide your program's display areas similarly. There's an easy way to do that, and that's with the windows functions in ncurses. This is a standard part of any curses-compatible library.

Simple Senet

You may associate "windows" with a graphical environment, but that is not the case here. In ncurses, "windows" are a means to divide the screen into logical areas. Once you define a window, you don't need to track its location on the screen; you just draw to your window using a set of ncurses functions.

To demonstrate, let me define a game board in an unexpected way. The ancient Egyptian game *Senet* uses a board of 30 squares arranged in three rows and ten columns. Two players move their pieces around the board in a backward "S" formation, so that the board looks like this:

```
 1  2  3  4  5  6  7  8  9  10
20 19 18 17 16 15 14 13 12 11
21 22 23 24 25 26 27 28 29 30
```

Without the windows functions, you'd have to keep track of the row and column for each piece and draw them separately. Since the board is arranged in a backward "S" pattern, you'll always need to do weird math to position the row and column correctly every time you update each square on the board. But with the windows functions, ncurses lets you define the squares once, including their position, and later refer to those windows by a logical identifier.

The ncurses function `newwin()` lets you define a text window of certain dimensions at a specific location on the screen:

```
WINDOW *newwin(int nlines, int ncols, int begin_y,
 ↪int begin_x);
```

The `newwin()` function returns a pointer of type `WINDOW*` that you can store in an array for later reference. To create a *Senet* board, you can use a global array `BOARD[30]`, and write a function to define the 30 squares of the *Senet* board using windows:

```
#define SQ_HEIGHT 5
#define SQ_WIDTH 8

WINDOW *BOARD[30];
```

```
void create_board(void)
{
    int i;
    int starty, startx;

    starty = 0;
    for (i = 0; i < 10; i++) {
        startx = i * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
    }

    starty = SQ_HEIGHT;
    for (i = 10; i < 20; i++) {
        startx = (19 - i) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
    }

    starty = 2 * SQ_HEIGHT;
    for (i = 20; i < 30; i++) {
        startx = (i - 20) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
    }

    /* put border on each window and refresh */

    for (i = 0; i < 30; i++) {
        box(BOARD[sq], '2', '2');
        wrefresh(BOARD[sq]);
    }
}
```

The first part of this function uses `newwin()` to define the 30 squares. I divided this into three parts to make it obvious that the second row actually counts backward.

Text windows in ncurses don't create a "frame" to show the window on the screen. If you want to draw a frame, you can do so using one of two functions. Here, after defining the windows, the function then calls the ncurses function `box()` to draw the square on the screen. Normally, the `box()` function takes arguments for the characters to use for the vertical and horizontal borders; if you pass zero as either or both arguments, ncurses uses a default line-drawing character.

Note that instead of the usual `refresh()` function to update the screen, you need to use the window-specific `wrefresh()` function to refresh the window.

In the *Senet* game, several squares carry certain meaning. Under common *Senet* rules, players must stop on square 26 (window array element `BOARD[25]`) before they can move off the board. Square 27 (array element `BOARD[26]`) is a trap, which sends the player back to square 15 (`BOARD[14]`). Squares 28 and 29 (BOARD[27] and BOARD[28], respectively) require the player to throw a "3" or "2" exactly to move their piece off the board.

To represent these special squares, I replaced the `box()` and `wrefresh()` functions at the end of `draw_board()` with a call to my own function, `draw_square()`. This function draws a different border for the special squares:

```
/* put border on each window and refresh */

    for (i = 0; i < 30; i++) {
        draw_square(i);
```

```
    }
}

void draw_square(int sq)
{
    switch (sq) {
    case 14:                    /* revive square */
        wborder(BOARD[sq], '#', '#', '#', '#', '#', '#',
         ↪'#', '#');
        break;

    case 25:                    /* stop square */
        box(BOARD[sq], 'X', 'x');
        break;

    case 26:                    /* water square */
        box(BOARD[sq], 'O', 'o');
        break;

    case 27:                    /* 3-move square */
        box(BOARD[sq], '3', '3');
        break;

    case 28:                    /* 2-move square */
        box(BOARD[sq], '2', '2');
        break;


    default:
        box(BOARD[sq], 0, 0);
    }

    wrefresh(BOARD[sq]);
}
```

The `draw_square()` function shows two ways to draw a frame on a text window. I covered the `box()` function earlier. The other method is with the `wborder()` function, which takes separate arguments for the left, right, top and bottom edges of the window, and the upper-left, upper-right, lower-left and lower-right corners. As with `box()`, if you pass zero as any or all of the arguments, ncurses will use a default line-drawing character.

Look in the sample output to see the difference between calling `box()` and `wborder()` with different arguments. I've intentionally used different methods in my *Senet* program to show a few combinations. For example, the "stop" square uses lowercase "x" for the top and bottom borders, and uppercase "X" for the left and right borders. The "revive" square uses `wborder()` to fill the corners, while the other squares are drawn more simply with `box()`. Note that calling `box()` with character arguments still draws line graphic characters for the corners.

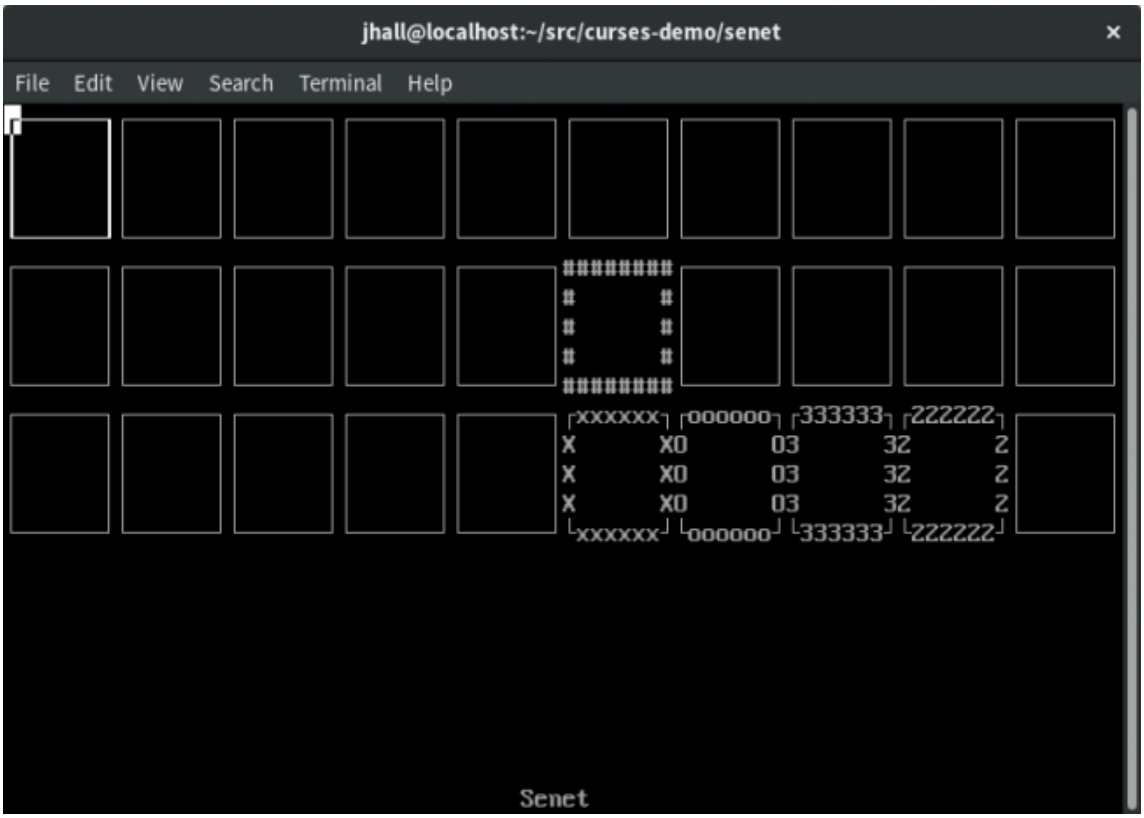

Figure 1. The Senet Board with the First Square Highlighted

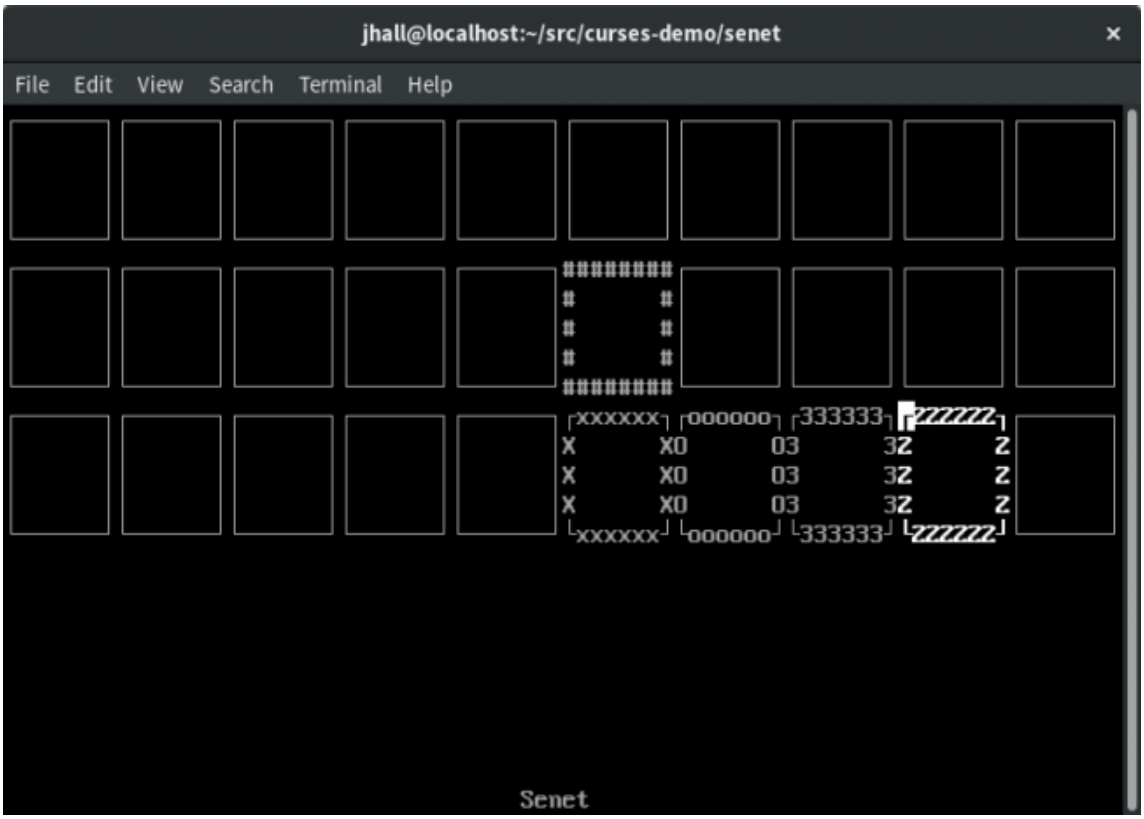

Figure 2. The Senet Board with the 29th Square Highlighted

Because this uses the windows functions, the `draw_square()` function doesn't need to know the location of each square. That's all tracked by ncurses as an attribute of the 30 windows that make up the 30 squares on the board. Once the program defines the windows, including their position, drawing to each square is a simple call to an associated "w" function, referencing the window to draw to.

For example, to draw a character in a window, you use the `waddch()` function, or `mvwaddch()`, if you want to draw at a specific location in the window. Most curses functions have a "w" partner function that operates on a specific window, such as these:

```
int move(int y, int x);
int wmove(WINDOW *win, int y, int x);

int addch(const chtype ch);
int waddch(WINDOW *win, const chtype ch);

int mvaddch(int y, int x, const chtype ch);
int mvwaddch(WINDOW *win, int y, int x, const chtype ch);
```

Full Program

Now that you've seen how to use windows to create 30 independent drawing areas, let's walk through a simple program to draw a *Senet* board and allow the user to navigate through the squares using the plus and minus keys. At a high level, the program follows these steps:

1. Initialize the curses environment.
2. Define and draw the 30 squares on the *Senet* board.
3. Loop: 1) get a key from the keyboard; 2) adjust the player's location to the previous or next square, accordingly; and 3) repeat.
4. When done, close the curses environment and exit.

Here's the program:

```
/* senet.c */

#include <stdlib.h>
#include <ncurses.h>

#define SQ_HEIGHT 5
#define SQ_WIDTH 8

void create_board(void);
void destroy_board(void);

void draw_square(int sq);
void highlight_square(int sq);

WINDOW *BOARD[30];

int main(int argc, char **argv)
{
    int key;
    int sq;

    /* initialize curses */

    initscr();
    noecho();
    cbreak();
```

```
    if ((LINES < 24) || (COLS < 80)) {
        endwin();
        puts("Your terminal needs to be at least 80x24");
        exit(2);
    }

    /* print welcome text */

    clear();

    mvprintw(LINES - 1, (COLS - 5) / 2, "Senet");
    refresh();


    /* draw board */

    create_board();

    /* loop: '+' to increment squares, '-'
       to decrement squares */

    sq = 0;
    highlight_square(sq);

    do {
        key = getch();

        switch (key) {
        case '+':
        case '=':
            if (sq < 29) {
                draw_square(sq);
                highlight_square(++sq);
            }
            break;

        case '-':
        case '_':
            if (sq > 0) {
                draw_square(sq);
                highlight_square(--sq);
            }
        }
    } while ((key != 'q') && (key != 'Q'));

    /* when done, free up the board, and exit */

    destroy_board();

    endwin();
    exit(0);
}

void create_board(void)
{
    int i;
    int starty, startx;

    starty = 0;
    for (i = 0; i < 10; i++) {
        startx = i * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
```

```
    }

    starty = SQ_HEIGHT;
    for (i = 10; i < 20; i++) {
        startx = (19 - i) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
    }

    starty = 2 * SQ_HEIGHT;
    for (i = 20; i < 30; i++) {
        startx = (i - 20) * SQ_WIDTH;
        BOARD[i] = newwin(SQ_HEIGHT, SQ_WIDTH, starty,
         ↪startx);
    }


    /* put border on each window and refresh */

    for (i = 0; i < 30; i++) {
        draw_square(i);
    }
}

void destroy_board(void)
{
    int i;

    /* erase every box and delete each window */

    for (i = 0; i < 30; i++) {
        wborder(BOARD[i], ' ', ' ', ' ', ' ', ' ', ' ', '
         ↪', ' ');
        wrefresh(BOARD[i]);

        delwin(BOARD[i]);
    }
}

void draw_square(int sq)
{
    switch (sq) {
    case 14:                    /* revive square */
        wborder(BOARD[sq], '#', '#', '#', '#', '#', '#',
         ↪'#', '#');
        break;

    case 25:                    /* stop square */
        box(BOARD[sq], 'X', 'x');
        break;

    case 26:                    /* water square */
        box(BOARD[sq], 'O', 'o');
        break;

    case 27:                    /* 3-move square */
        box(BOARD[sq], '3', '3');
        break;

    case 28:                    /* 2-move square */
        box(BOARD[sq], '2', '2');
        break;
```

```
    default:
        box(BOARD[sq], 0, 0);
    }

    wrefresh(BOARD[sq]);
}

void highlight_square(int sq)
{
    wattron(BOARD[sq], A_BOLD);
    draw_square(sq);
    wattroff(BOARD[sq], A_BOLD);
}
```

This is just the bare bones of a *Senet* game. All it does is generate a game board and allow the user to navigate through all of the squares. To keep this focused on the windows functions in ncurses, I've left out all the gameplay and rules.

The program uses only a few functions:

- `void create_board(void);` — defines the 30 squares as text windows and draws them on the screen.
- `void destroy_board(void);` — erases the 30 squares and deletes the windows.
- `void draw_square(int sq);` — draws a single square on the board. This function draws a different outline depending on the special squares used in *Senet*.
- `void highlight_square(int sq);` — highlights a square as the user navigates through each square on the board. To keep things simple, this uses the `A_BOLD` attribute instead of color.

Learning on Your Own

This program is a simple example of how to use nurses windows functions to define separate areas on the screen. The sample program is a game, but you can use this as a starting point for your own programs. Any program that requires updating multiple areas of the screen can use the windows functions.

The ncurses library provides a rich set of functions to update and access the screen in text mode. While graphical user interfaces are very cool, not every program needs to run with a point-and-click interface. If your program runs in plain-text terminals, consider using ncurses to manipulate the terminal screen.

Resources

- If you are interested in learning more about curses, the ncurses man pages provide extensive documentation on the different functions.
- For more information, including programming examples, read Pradeep Padala's "NCURSES Programming HOWTO" at the Linux Documentation Project.
- "Creating an Adventure Game in the Terminal with ncurses" by Jim Hall
- "Getting Started with ncurses" by Jim Hall
- "Programming in Color with ncurses" by Jim Hall
- "About ncurses Colors" by Jim Hall
- *Senet* (Wikipedia)