Periféricos y Dispositivos de Interfaz Humana

UNIVERSIDAD DE GRANADA

CURSO 2024 - 2025

Práctica 2. Uso de bibliotecas de programación de interfaces de usuario en modo texto

GABRIEL VICO ARBOLEDAS
RAÚL RODRÍGUEZ RODRÍGUEZ

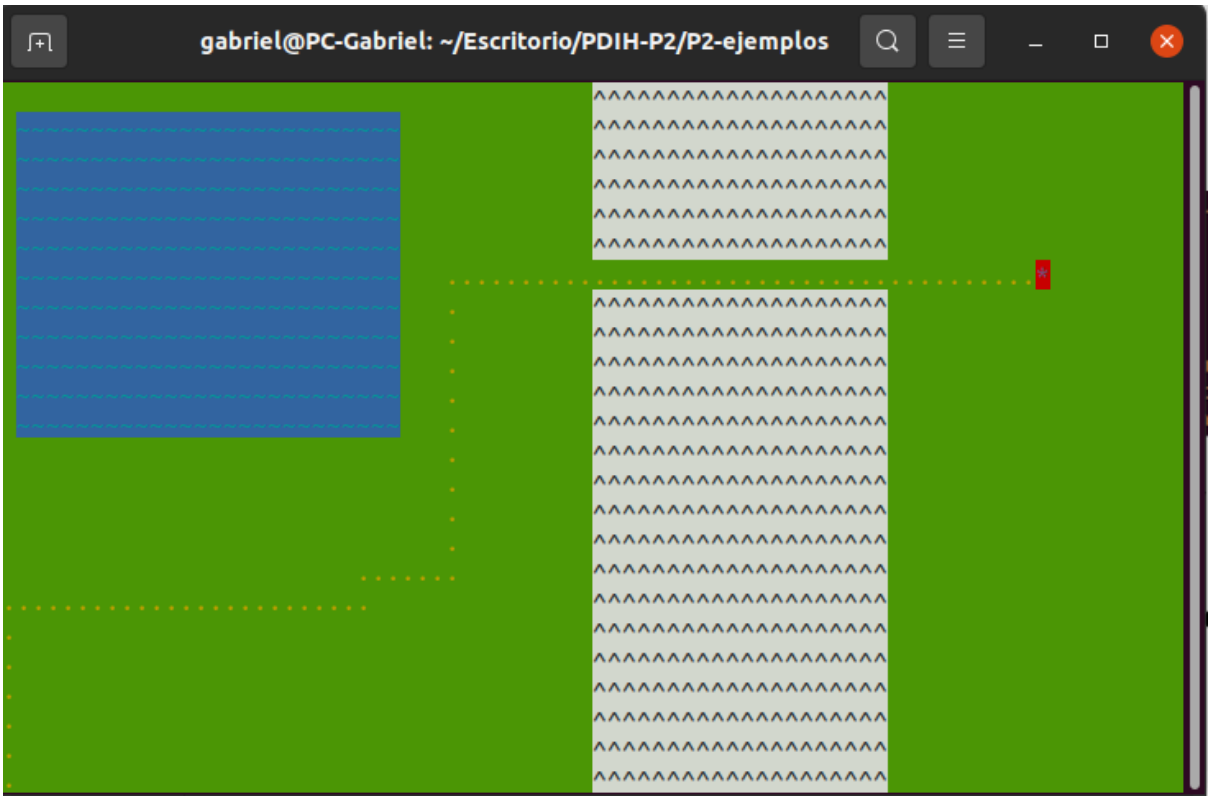
Índice

1. Ejercicios obligatorios

1.1 Instalar la librería ncurses, usar programas de ejemplo

```
gabriel@PC-Gabriel:~/Escritorio/PDIH-P2/P2$ sudo apt-get install libncurses5-dev
 libncursesw5-dev
[sudo] contraseña para gabriel:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
libncurses5-dev ya está en su versión más reciente (6.2-0ubuntu2.1).
libncursesw5-dev ya está en su versión más reciente (6.2-0ubuntu2.1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 195 no actualizados.
```

Para comprobar el correcto funcionamiento de la librería vamos a probar algún ejemplo, como puede ser el de aventura:



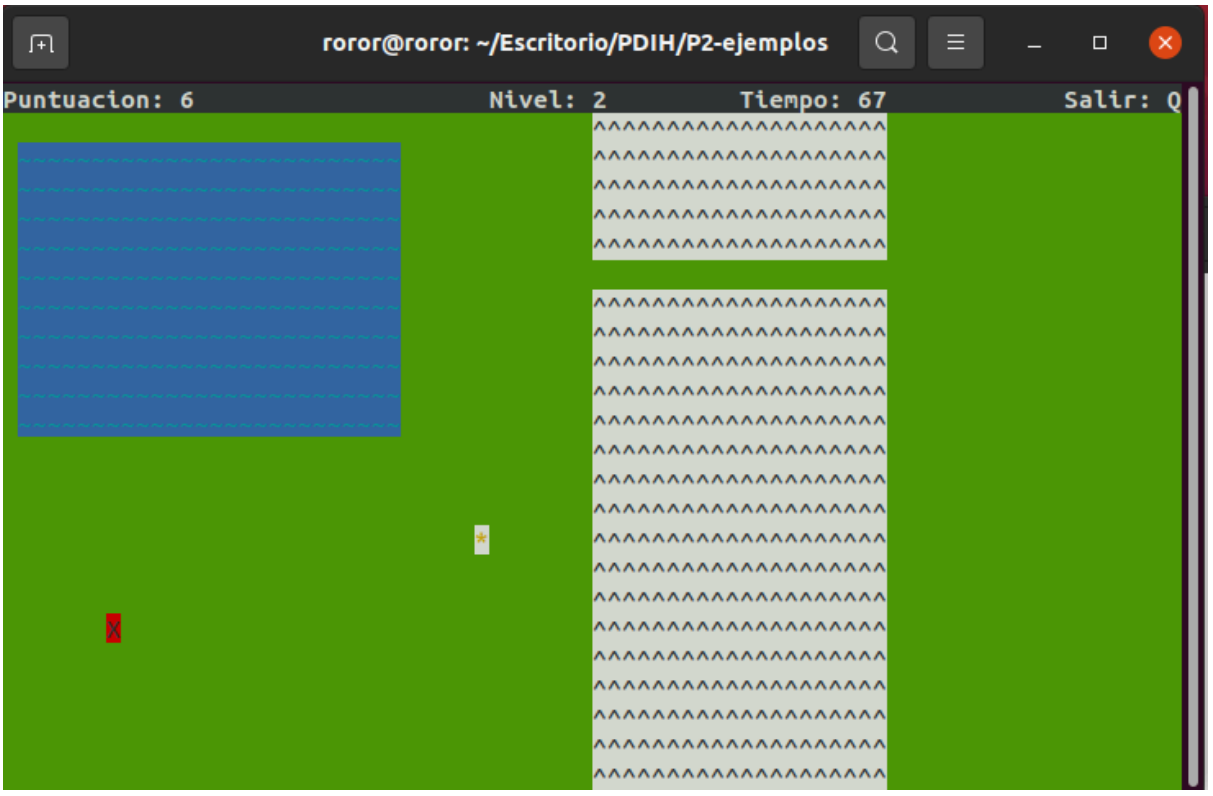
1.2 Crear un juego sencillo.

NCursed Knight: Un Juego de Aventuras en Terminal

En el mundo de *NCursed Knight*, te sumergirás en una emocionante aventura en modo texto donde la exploración y el combate se entrelazan para crear una experiencia única. Como un valiente caballero, deberás recorrer un vasto territorio lleno de peligros, desde imponentes montañas hasta misteriosos lagos y extensas llanuras. Tu objetivo principal será enfrentarte a los temibles monstruos que aparecen aleatoriamente en el mapa, poniendo a prueba tu destreza y estrategia.

El combate en *NCursed Knight* es sencillo pero desafiante. Para derrotar a tus enemigos, simplemente deberás acercarte a ellos, pero no te confíes. Cada monstruo derrotado hará que los siguientes sean más rápidos y escurridizos, creando una progresión de dificultad que mantendrá la emoción durante toda la partida. A medida que avances en el juego, irás alcanzando nuevos niveles que reflejarán tu habilidad como caballero.

El tiempo corre en tu contra en esta aventura. Tendrás un límite para demostrar tu valentía y habilidad, lo que añade una capa extra de emoción a cada decisión que tomes. Los controles intuitivos, que puedes manejar tanto con las teclas direccionales como con WASD.



Empezamos inicializando los aspectos que tendrán el cursor para evitar parpadeos en los personajes del juego además de los colores con los diferentes pares que posee cada uno y los elementos del mapa:

```c
/* initialize curses */
initscr();
keypad(stdscr, TRUE);
cbreak();
noecho();
curs_set(0);
nodelay(stdscr, TRUE);

/* initialize colors */
if (has_colors() == FALSE) {
    endwin();
    printf("Your terminal does not support color\n");
    exit(1);
}

start_color();
init_pair(GRASS_PAIR, COLOR_YELLOW, COLOR_GREEN);
init_pair(WATER_PAIR, COLOR_CYAN, COLOR_BLUE);
init_pair(MOUNTAIN_PAIR, COLOR_BLACK, COLOR_WHITE);
init_pair(PLAYER_PAIR, COLOR_YELLOW, COLOR_WHITE);
init_pair(MONSTRUO_PAIR, COLOR_BLACK, COLOR_RED);
init_pair(WELCOME_PAIR, COLOR_WHITE, COLOR_GREEN);
init_pair(END_PAIR, COLOR_WHITE, COLOR_GREEN);
```

Tras esto entramos en el bucle principal, donde tiene lugar el juego. Primero se lanza la pantalla de inicio y se reinician los indicadores de la partida como la puntuación y el nivel, en el caso de que el jugador quisiera empezar una nueva partida desde la pantalla de fin de partida. Después se inicializa el lugar de *spawn* del jugador y del monstruo:

```c
while(jugar_de_nuevo && !salir) {

    mostrar_bienvenida(&salir);
    puntuacion = 0, nivel = 0;

    if (!salir) {

        clear();

        /* initialize the quest map */
        draw_map();

        /* start player at lower-left */
        y = LINES - 1;
        x = 0;

        /* inicializar la posición del monstruo random */
        srand(time(NULL));
        respawn_monstruo(&y_monstruo, &x_monstruo);
```

Tras esto se inicializan los puntos de partida del movimiento del monstruo con el dato tipo struct tv, el cuál sirve para modificar la velocidad del movimiento del monstruo para los diferentes niveles según el tiempo que haya pasado. También se inicializa el tiempo de inicio de la partida con start_time y tiempo_actual, para controlar la duración de la misma, y con time_left el tiempo restante de partida. Finalmente, se pinta la interfaz de información ,con la puntuación, nivel y el tiempo restante, en la pantalla, donde se va actualizando en cada bucle.

El dato tipo struct timeval tv está definida en sys/time.h, y se compone de lo siguiente:

```
struct timeval {
        time_t      tv_sec;   // Segundos desde la época (1/1/1970)
        suseconds_t tv_usec;  // Microsegundos adicionales (0-999999)
};
```

Para nuestro juego, sirve para lo siguiente:

1. Medir intervalos de tiempo con precisión de microsegundos
2. Controlar la velocidad del monstruo independientemente del bucle principal

Donde el flujo completo del movimiento del monstruo es:

1. Obtener tiempo actual: Se usa gettimeofday() para obtener el tiempo preciso
2. Comparar intervalos: Se verifica si ha pasado el tiempo suficiente (monster_speed)
3. Movimiento del monstruo: Si puede moverse:
    - Se elige dirección aleatoria
    - Se calcula nueva posición
    - Se verifica que sea posición válida
    - Se actualizan las coordenadas
4. Actualizar temporizador: Se guarda el momento del último movimiento

```
gettimeofday(&tv, NULL);
last_monster_move = tv.tv_sec * 1000000 + tv.tv_usec;

start_time = get_current_time_us();

do {
    tiempo_actual = get_current_time_us();
    time_left = tiempo_partida - (tiempo_actual - start_time) / 1000000L;
    if (time_left < 0) time_left = 0;

    draw_game_info(puntuacion, time_left, nivel);

        /* Mover monstruo según su velocidad */
        gettimeofday(&tv, NULL);
        int current_time = tv.tv_sec * 1000000 + tv.tv_usec;
        if (current_time - last_monster_move >= monster_speed) {

                    /* Detectar colisión DIRECTAMENTE por coordenadas */
                    if (y == y_monstruo && x == x_monstruo) {
                        puntuacion++;
                        refresh();
                        respawn_monstruo(&y_monstruo, &x_monstruo);

                        mvprintw(LINES/2, COLS/2 - 10, "¡Monstruo derrotado!");
                refresh();
                pausa_segundos(1000000, &start_time);


                        /* Aumentar dificultad progresivamente */
                        if (puntuacion == 3){
                                monster_speed = 400000;
                                subida_nivel(nivel, &start_time);
                                nivel++;
                }
                                if (puntuacion == 6){
                                        monster_speed = 300000;
                                        subida_nivel(nivel, &start_time);
                            nivel++;
      }
          }

movimiento_monstruo(&y_monstruo, &x_monstruo, &last_monster_move, monster_speed);
```

Posteriormente, se comprueba si el jugador alcanza al monstruo, comprobando sus coordenadas, y si se da el caso, se incrementa en 1 la puntuación, se refresca la pantalla y se hace un respawn el monstruo, además de escribir un mensaje por pantalla, informando del hecho con una pequeña pausa, donde se controla con la función pausa_segundos, para hacer que el juego sea justo y no baje el tiempo de partida mientras el jugador no puede moverse por la pausa.

También se comprueba la puntuación del jugador, donde si llega a diferentes nº de monstruos derrotados, se sube la dificultad, disminuyendo el tiempo que debe estar el monstruo sin moverse, haciendo que se incremente su velocidad.

```
/* Mover monstruo según su velocidad */
gettimeofday(&tv, NULL);
int current_time = tv.tv_sec * 1000000 + tv.tv_usec;
if (current_time - last_monster_move >= monster_speed) {

            /* Detectar colisión DIRECTAMENTE por coordenadas */
            if (y == y_monstruo && x == x_monstruo) {
                puntuacion++;
                refresh();
                respawn_monstruo(&y_monstruo, &x_monstruo);

                mvprintw(LINES/2, COLS/2 - 10, "¡Monstruo derrotado!");
        refresh();
        pausa_segundos(1000000, &start_time);


                /* Aumentar dificultad progresivamente */
                if (puntuacion == 3){
                        monster_speed = 400000;
                        subida_nivel(nivel, &start_time);
                        nivel++;
        }
                        if (puntuacion == 6){
                                monster_speed = 300000;
                                subida_nivel(nivel, &start_time);
```

En el final del bucle, se comprueba que la partida no haya llegado a su fin o si no se pulsa el botón de fin de partida (q), donde por defecto muestra la pantalla de fin de partida:

```
        if (time_left <= 0) {
            clear();
            draw_game_info(puntuacion, 0, nivel);
            mvprintw(LINES/2, COLS/2 - 10, "¡Tiempo terminado!");
            mvprintw(LINES/2 + 1, COLS/2 - 10, "Puntuación final: %d", puntuacion);
            mvprintw(LINES/2 + 2, COLS/2 - 10, "Nivel alcanzado: %d", nivel);
            refresh();
            getch();
            break;
        }

        usleep(10000); // Pequeña pausa para reducir CPU
    }
    while (ch != 'q' && ch != 'Q');

f (!salir) {
        mostrar_fin_juego(puntuacion, nivel, &jugar_de_nuevo);
    }
```

Ahora se explica como funcionan las diferentes funciones necesarias para que funcione el juego. Primero una función para limitar que casillas son válidas para moverse:

```
int is_move_okay(int y, int x)
{
        if (y == 0) return 0;  // La línea 0 (información) no es transitable

    int testch = mvinch(y, x) & A_CHARTEXT;
    return (testch == GRASS || testch == EMPTY || testch == MONSTRUO);
}
```

La función movimiento_monstruo(...) se encarga del movimiento que realizará el monstruo, actualizando las coordenadas del monstruo, el tiempo del movimiento que realizó el mismo y el movimiento que hará, que será random y debe ser a una casilla legal:

```c
void movimiento_monstruo(int *my, int *mx, int *last_move_time, int monster_speed)
{
    int direccion = rand() % 4;
    int nuevo_y = *my, nuevo_x = *mx;

    switch (direccion) {
        case 0: if (nuevo_y > 0) nuevo_y--; break;
        case 1: if (nuevo_y < LINES-1) nuevo_y++; break;
        case 2: if (nuevo_x > 0) nuevo_x--; break;
        case 3: if (nuevo_x < COLS-1) nuevo_x++; break;
    }

    // Verificar si la nueva posición es válida
    if (is_move_okay(nuevo_y, nuevo_x)) {
        *my = nuevo_y;
        *mx = nuevo_x;
    }

    // Actualizar el tiempo del último movimiento
    struct timeval tv;
    gettimeofday(&tv, NULL);
    *last_move_time = tv.tv_sec * 1000000 + tv.tv_usec;
}
```

respawn_monstruo(...) se encarga de dar la nueva posición en la que debe reaparecer el monstruo cuando lo derrotan:

```c
void respawn_monstruo(int *y_monstruo, int *x_monstruo)
{
    do {
        *y_monstruo = rand() % LINES;
        *x_monstruo = rand() % COLS;
    } while (!is_move_okay(*y_monstruo, *x_monstruo));
}
```

subida_nivel (...) se encarga de mostrar un mensaje por pantalla informando al jugador el incremento de dificultad:

```c
void subida_nivel (int n, long *start_time){
       mvprintw(LINES/2, COLS/2 - 10, "¡Nivel %d completado!", n);
       mvprintw(LINES/2 + 2, COLS/2 - 10, "Ahora los monstruos se hacen más fuertes");
   refresh();
   pausa_segundos(2000000, start_time);
}
```

draw_game_info(...) se encarga de pintar la interfaz con los datos de la partida:

```c
void draw_game_info(int score, long time_left, int nivel) {
    // Dibujar la información en la primera línea
    attron(A_BOLD);
    mvhline(0, 0, ' ', COLS);
    mvprintw(0, 0, "Puntuacion: %d", score);
    mvprintw(0, COLS/2 - 7, "Nivel: %d", nivel);
    mvprintw(0, COLS - 30, "Tiempo: %02ld", time_left);
    mvprintw(0, COLS - 8, "Salir: Q");
    attroff(A_BOLD);
}
```

pausa_segundos(...) se encarga de sumar el tiempo de pausa de los mensajes mostrados por pantalla que bloquean el movimiento del jugador, incrementando el tiempo de incio de la partida, para no hacer que sea injusto:

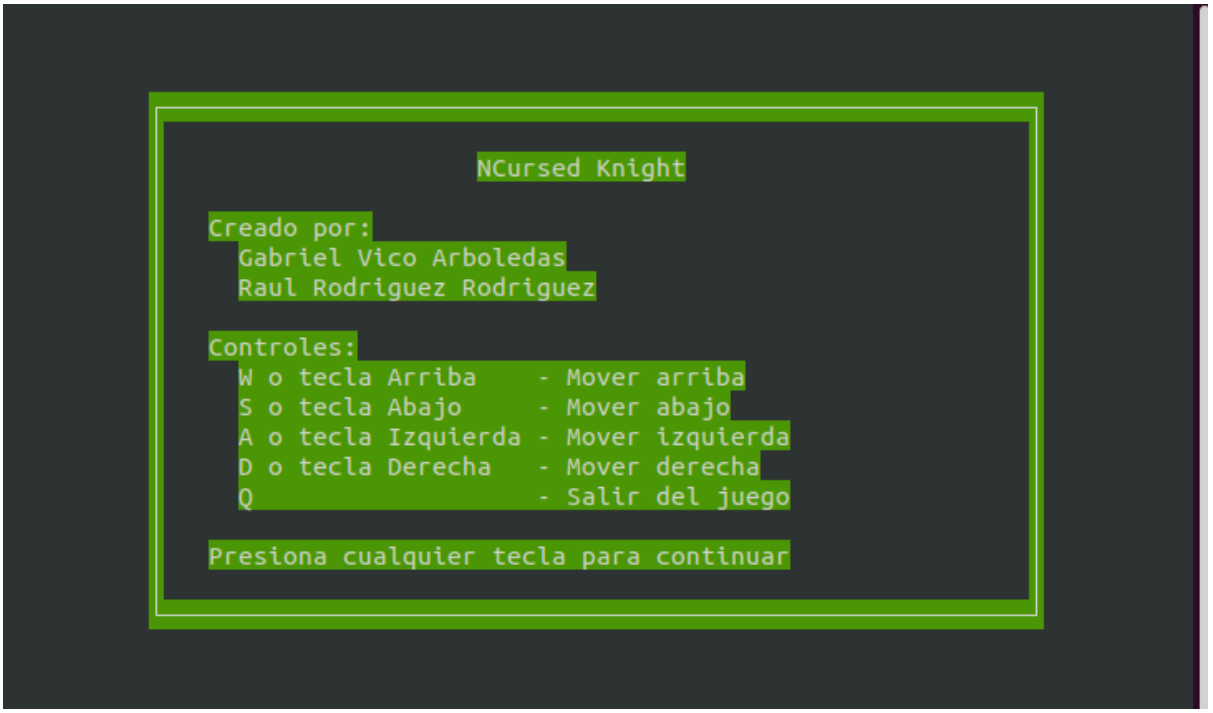
```c
void pausa_segundos(long microsegundos, long *start_time) {
    long tiempo_inicio_pausa = get_current_time_us();
    usleep(microsegundos);
    *start_time += (get_current_time_us() - tiempo_inicio_pausa);
}
```

2. Ejercicios opcionales

2.1 Pantalla de bienvenida al iniciar el juego

Al ejecutar el juego, los jugadores son recibidos por una pantalla de inicio elegante y funcional, diseñada para sumergirlos de inmediato en la aventura. Con un fondo verde y texto blanco, la interfaz muestra:

- El título del juego, **NCursed Knight**, centrado en la parte superior.
- Los créditos de los desarrolladores.
- Una sección clara de **Controles**, explicando los movimientos básicos y cómo salir del juego.
- Un mensaje invitando al jugador a presionar cualquier tecla para comenzar.



El código de la pantalla de inicio es el siguiente:

```c
void mostrar_bienvenida(int *salir) {
    nodelay(stdscr, FALSE);
    clear();
    attron(COLOR_PAIR(WELCOME_PAIR));

    int height = 18;
    int width = 60;
    int start_y = (LINES - height) / 2;
    int start_x = (COLS - width) / 2;

    // Dibujar borde
    for(int y = start_y; y < start_y + height; y++) {
        mvaddch(y, start_x, ACS_VLINE);
        mvaddch(y, start_x + width - 1, ACS_VLINE);
    }
    for(int x = start_x; x < start_x + width; x++) {
        mvaddch(start_y, x, ACS_HLINE);
        mvaddch(start_y + height - 1, x, ACS_HLINE);
    }
    mvaddch(start_y, start_x, ACS_ULCORNER);
    mvaddch(start_y, start_x + width - 1, ACS_URCORNER);
    mvaddch(start_y + height - 1, start_x, ACS_LLCORNER);
    mvaddch(start_y + height - 1, start_x + width - 1, ACS_LRCORNER);

    // Título del juego centrado
    mvprintw(start_y + 2, start_x + (width - 15)/2, "NCursed Knight");

    // Información de creadores
    mvprintw(start_y + 4, start_x + 4, "Creado por:");
    mvprintw(start_y + 5, start_x + 6, "Gabriel Vico Arboledas");
    mvprintw(start_y + 6, start_x + 6, "Raul Rodriguez Rodriguez");

    // Controles - alineado a la izquierda
    mvprintw(start_y + 8, start_x + 4, "Controles:");
    mvprintw(start_y + 9, start_x + 6, "W o tecla Arriba    - Mover arriba");
    mvprintw(start_y + 10, start_x + 6, "S o tecla Abajo     - Mover abajo");
    mvprintw(start_y + 11, start_x + 6, "A o tecla Izquierda - Mover izquierda");
    mvprintw(start_y + 12, start_x + 6, "D o tecla Derecha   - Mover derecha");
    mvprintw(start_y + 13, start_x + 6, "Q                   - Salir del juego");

    // Mensaje de continuar - alineado con "Controles"
    mvprintw(start_y + 15, start_x + 4, "Presiona cualquier tecla para continuar");

    attroff(COLOR_PAIR(WELCOME_PAIR));
    refresh();

    int ch = getch();
    nodelay(stdscr, TRUE);
    if (ch == 'q' || ch == 'Q') {
        *salir = 1;
    }
}
```

La función mostrar_bienvenida() implementa la pantalla inicial utilizando la biblioteca ncurses. Establece las dimensiones del cuadro de diálogo, calculando su posición centrada en la terminal mediante las variables LINES y COLS. Para el marco se emplean caracteres especiales como ACS_VLINE y ACS_HLINE, mientras que los atributos de color se definen con COLOR_PAIR(WELCOME_PAIR), configurando un fondo verde con texto blanco.

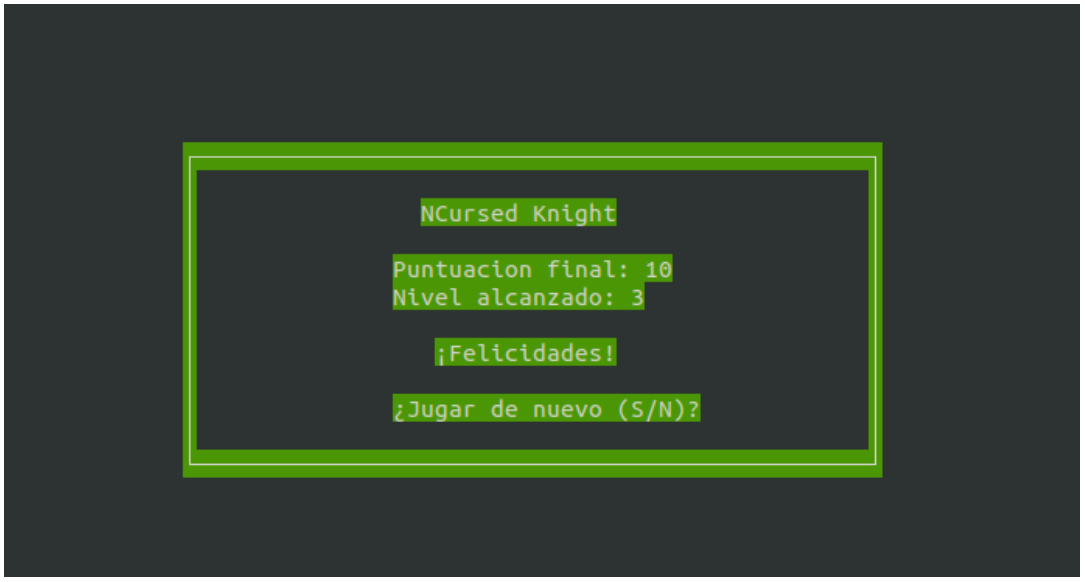
El contenido se realiza mediante llamadas a mvprintw(y,x) que posicionan cada elemento textual. La gestión de eventos se basa en getch() para capturar la entrada del usuario, con un caso especial que detecta la tecla 'Q' para salir inmediatamente. La función actualiza la pantalla con refresh() tras completar todos los dibujos, asegurando una visualización limpia.

2.2 Pantalla de resumen al finalizar el juego

Al concluir cada partida en NCursed Knight, el juego presenta una pantalla final que resume tu desempeño. Este menú no solo celebra tus logros, sino que te ofrece la posibilidad de volver a jugar.

En la pantalla final se muestra:

- Tu puntuación total, reflejando cada monstruo derrotado
- El nivel máximo alcanzado, indicador de tu progresión
- Mensaje de felicitación y mensaje de volver a jugar



El código de la pantalla de final es el siguiente:

```c
void mostrar_fin_juego(int score, int nivel, int *jugar_de_nuevo) {
    int ch;
    clear();

    do {
        attron(COLOR_PAIR(END_PAIR));

        int height = 12;
        int width = 50;
        int start_y = (LINES - height) / 2;
        int start_x = (COLS - width) / 2;

        // Dibujar borde
        for(int y = start_y; y < start_y + height; y++) {
            mvaddch(y, start_x, ACS_VLINE);
            mvaddch(y, start_x + width - 1, ACS_VLINE);
        }
        for(int x = start_x; x < start_x + width; x++) {
            mvaddch(start_y, x, ACS_HLINE);
            mvaddch(start_y + height - 1, x, ACS_HLINE);
        }
        mvaddch(start_y, start_x, ACS_ULCORNER);
        mvaddch(start_y, start_x + width - 1, ACS_URCORNER);
        mvaddch(start_y + height - 1, start_x, ACS_LLCORNER);
        mvaddch(start_y + height - 1, start_x + width - 1, ACS_LRCORNER);

        // Título del juego centrado
        mvprintw(start_y + 2, start_x + (width - 15)/2, "NCursed Knight");
        mvprintw(start_y + 4, start_x + (width - 20)/2, "Puntuacion final: %d", score);
        mvprintw(start_y + 5, start_x + (width - 20)/2, "Nivel alcanzado: %d", nivel);
        mvprintw(start_y + 7, start_x + (width - 14)/2, "¡Felicidades!");
        mvprintw(start_y + height - 3, start_x + (width - 20)/2, "¿Jugar de nuevo (S/N)?");

        attroff(COLOR_PAIR(END_PAIR));
        refresh();

        ch = getch();
        if (ch == 'q' || ch == 'Q') {
            *jugar_de_nuevo = 0;
            return;
        }
    } while(ch != 's' && ch != 'S' && ch != 'n' && ch != 'N');

    *jugar_de_nuevo = (ch == 's' || ch == 'S');
}
```

La función mostrar_fin_juego() dibuja una pantalla de resultados centrada (50x12 caracteres) con bordes usando ACS_VLINE/HLINE y colores verde/blanco (END_PAIR). Muestra puntuación, nivel y opciones (S/N) con mvprintw(), usando un bucle que solo acepta estas teclas. Actualiza la pantalla con refresh() y se integra con el sistema mediante parámetros (puntuación, nivel).

Si pulsamos “S” nos vuelve a la pantalla de bienvenida para iniciar una nueva partida, y si pulsamos “N” o “Q”, el programa finaliza.