



ugr

Universidad
de Granada

PRÁCTICA 2

USO DE BIBLIOTECAS DE PROGRAMACIÓN DE INTERFACES DE USUARIO EN MODO TEXTO

Periféricos y Dispositivos de Interfaz Humana

Autores:

Clara Sola Ruiz

Mario Casas Pérez

Índice

1. Introducción	3
2. Instalación de la librería ncurses y creación de los programas de ejemplo y comprensión de su funcionamiento.	4
• aventura.c	4
• colores.c	5
• hello.c	6
• pelotita.c	7
• pruncurses.c	8
• rebota.c	9
• rebota2.c	10
• ventana.c	11
3. Creación del juego “asteroids”	12
• Definición de constantes y estructuras	12
• Mensaje inicial de bienvenida	12
• Inicialización y actualización de asteroides	13
• Control de la nave	13
• Control de los disparos	14
• Colisiones	14
• main	15
• Salida por terminal	16
4. Creación del juego “Las aventuras de arropa”	18
• Definición de constantes y estructuras	18
• Pantallas y dibujos	18
• Inicialización de elementos del juego	19
• Movimiento y colisiones	20
• main	21
• Salida por terminal	22
5. Juego Serpiente	25
Definición de constantes y variables globales	25
• init_game()	25
• dibujar_juego()	26
• mover_serpiente()	27
• comprobar_colision()	28
• generar_comida()	28
• pantalla_bienvenida()	29
• game_over()	30
• Capturas de pantalla	30

1. Introducción

Para la realización de esta práctica hemos usado la biblioteca *ncurses*, la cual es una biblioteca de programación que provee una API que permite al programador escribir interfaces basadas en texto. Esto se hace creando una capa sobre las capacidades del terminal y proporcionando métodos para crear dichas interfaces de usuario en modo texto. Se permite crear fácilmente aplicaciones basadas en ventanas, menús, paneles y formularios (las ventanas se pueden controlar de forma independiente, facilitando su movimiento, pudiéndose incluso ocultar o mostrar).

En esta memoria se describe de forma detallada la realización de las diversas actividades descritas en el guión de la práctica 2, ejecutando todos los ejemplos proporcionados y haciendo varios juegos gracias a la biblioteca *ncurses*.

2. Instalación de la librería ncurses y creación de los programas de ejemplo y comprensión de su funcionamiento.

Hemos descargado la librería ncurses en Fedora para la realización de esta práctica.

The image shows two terminal windows from a Fedora system. The left window shows the command `sudo dnf install ncurses-devel` being executed. It displays the package details for `ncurses-devel` and `ncurses-c++-libs`, including their architecture, version, repository, and size. The right window shows the progress of the installation, including the download of the RPM files and the execution of the scriptlets.

```
[root@pclab P2]# sudo dnf install ncurses-devel
Ultima comprobación de caducidad de metadatos hecha hace 0:01:40, el jue 27 mar 2025 15:02:22.
Dependencias resueltas.
=====
Paquete      Arq.      Versión      Repositorio  Tam.
-----
Instalando:
ncurses-devel x86_64     6.4-3.20230114.fc37 updates      549 k
Instalando dependencias:
ncurses-c++-libs x86_64     6.4-3.20230114.fc37 updates       37 k
=====
Resumen de la transacción
-----
Instalar 2 Paquetes

Tamaño total de la descarga: 585 k
Tamaño instalado: 1.0 M
¿Está de acuerdo [s/N]?
```

```
s://mirrors.hostiserver.com/fedora/archive/fedora/linux/updates/37/Everything/x86_64/Packages/n/ncurses-devel-6.4-3.20230114.fc37.x86_64.rpm (IP: 45.84.31.11)
(1/2): ncurses-c++-libs-6.4-3.20230114.fc37.x86_64 34 kB/s | 37 kB 00:01
(2/2): ncurses-devel-6.4-3.20230114.fc37.x86_64 353 kB/s | 549 kB 00:01
-----
Total                                          289 kB/s | 585 kB 00:02
Ejecutando verificación de operación
Verificación de operación exitosa.
Ejecutando prueba de operaciones
Prueba de operación exitosa.
Ejecutando operación
Preparando : 1/1
Instalando : ncurses-c++-libs-6.4-3.20230114.fc37.x86_64 1/2
Instalando : ncurses-devel-6.4-3.20230114.fc37.x86_64 2/2
Ejecutando scriptlet: ncurses-devel-6.4-3.20230114.fc37.x86_64 2/2
Verificando : ncurses-c++-libs-6.4-3.20230114.fc37.x86_64 1/2
Verificando : ncurses-devel-6.4-3.20230114.fc37.x86_64 2/2

Instalado:
ncurses-c++-libs-6.4-3.20230114.fc37.x86_64
ncurses-devel-6.4-3.20230114.fc37.x86_64

¡Listo!
[root@pclab P2]#
```

Una vez instalado, vamos a descargar el fichero P2-ejemplos.zip en donde se encuentran todos los ejemplos proporcionados. A continuación, pasaremos a compilar y ejecutar aquellos programas que están en lenguaje C para ver el procedimiento de cada uno.

- aventura.c

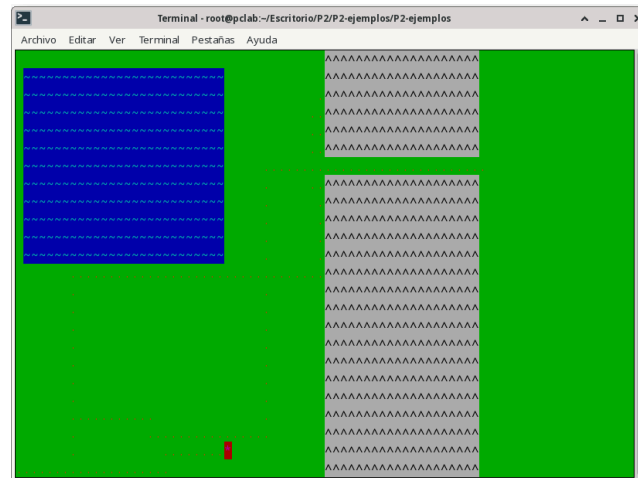
Para compilar y ejecutar el programa haremos lo siguiente:

The image shows a terminal window where the user has navigated to the `P2-ejemplos` directory. They have compiled the `aventura.c` file using `gcc` with the `-lncurses` flag, and then executed the resulting `aventura` binary.

```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo Editar Ver Terminal Pestañas Ayuda
[root@pclab P2-ejemplos]# gcc aventura.c -o aventura -lncurses
[root@pclab P2-ejemplos]# ./aventura
```

Práctica 2

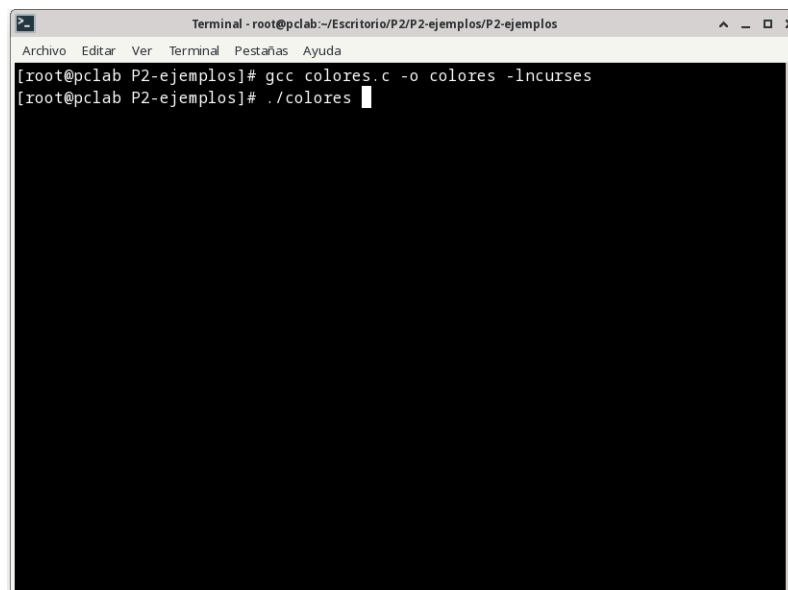
Una vez ejecutado se nos abre una ventana en donde se puede apreciar un pequeño mundo creado para un videojuego.



En él podemos apreciar que la figura roja es el muñeco que nosotros controlamos y que podemos ir moviendo por la pantalla. Además, conforme el muñeco se va moviendo, va dejando un rastro de puntitos rojos por donde ha ido pasando. La parte azul representa zona de agua, por lo que el muñeco no puede andar por esa zona. La parte gris representa una especie de muro, en donde el muñeco no podrá pasar a través de él.

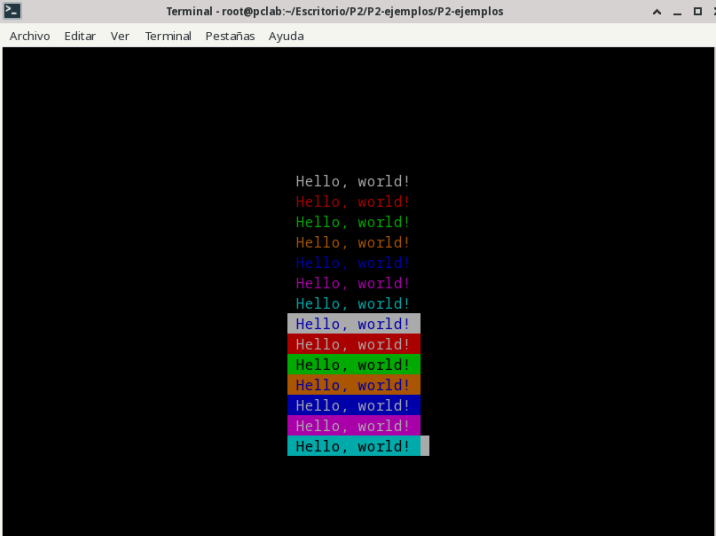
- colores.c

Para compilar y ejecutar el programa haremos lo siguiente:



Práctica 2

Una vez ejecutado se nos abre una ventana en donde se puede apreciar una serie de Hello, world! los cuales están escritos de diversos colores y fondos.



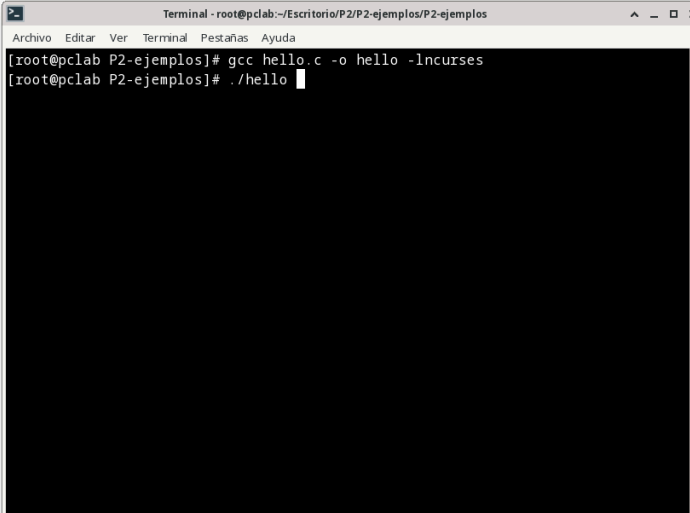
```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

Además, cuando pasa un pequeño periodo de tiempo, el programa termina por defecto, dejándonos de vuelta en la terminal.

- hello.c

Para compilar y ejecutar el programa haremos lo siguiente:

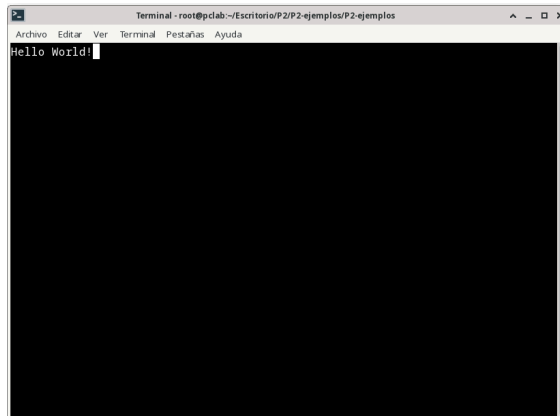


```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

[root@pclab P2-ejemplos]# gcc hello.c -o hello -lcurses
[root@pclab P2-ejemplos]# ./hello
```

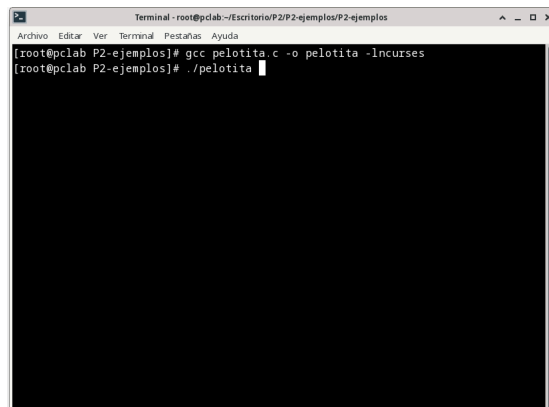
Práctica 2

Una vez ejecutado se nos abre una ventana en donde se puede apreciar el típico mensaje Hello, World!

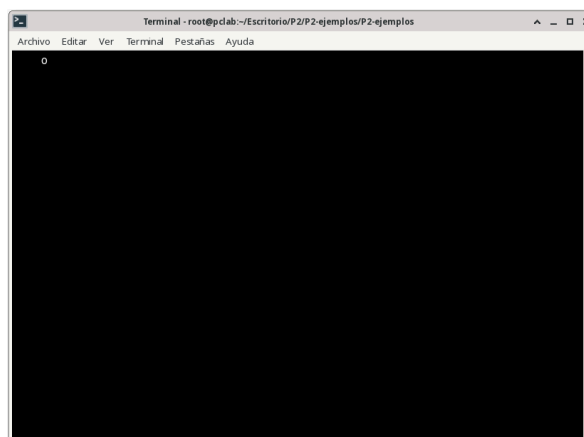


- **pelotita.c**

Para compilar y ejecutar el programa haremos lo siguiente:

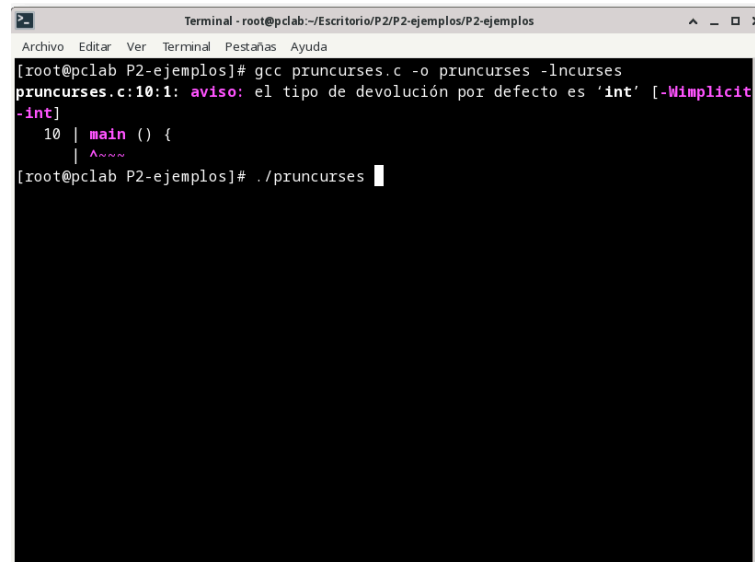


Una vez ejecutado se nos abre una ventana en donde se puede apreciar una pelota que se va moviendo en un rango específico del eje x de la terminal.



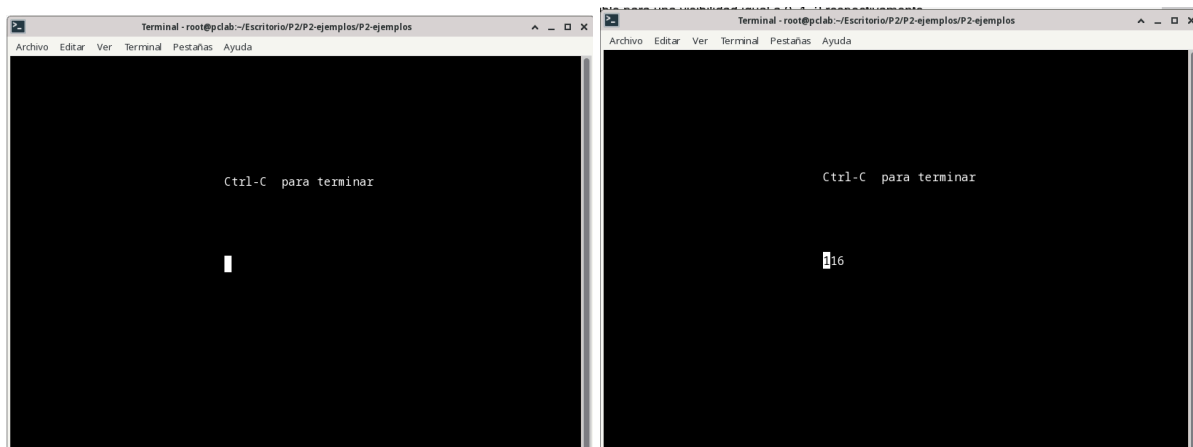
- pruncurses.c

Para compilar y ejecutar el programa haremos lo siguiente:



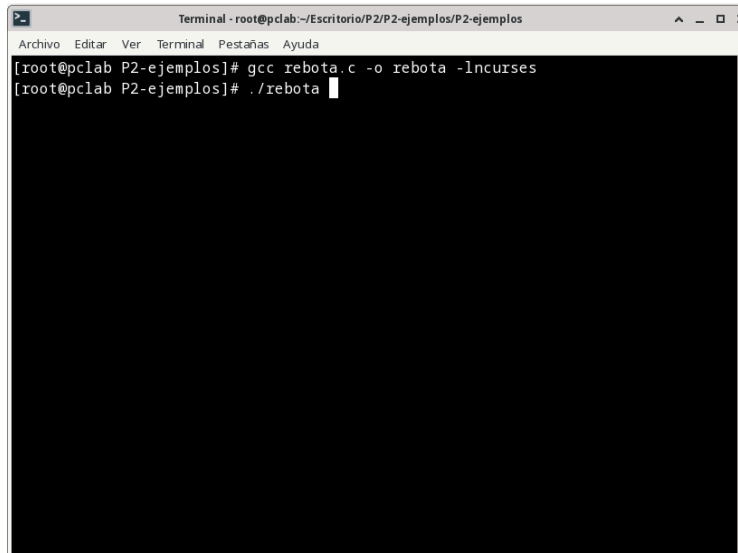
```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo Editar Ver Terminal Pestañas Ayuda
[root@pclab P2-ejemplos]# gcc pruncurses.c -o pruncurses -lncurses
pruncurses.c:10:1: aviso: el tipo de devolución por defecto es 'int' [-Wimplicit-int]
    10 | main () {
        | ^~~~~
[root@pclab P2-ejemplos]# ./pruncurses
```

Una vez ejecutado se nos abre una ventana en donde se puede apreciar un mensaje que dice que pulsemos Ctrl-C para terminar de ejecutar del programa. En otro caso, si pulsamos una tecla de manera aleatoria, saldrá un número en concreto. Podemos estar pulsando caracteres indefinidamente que nos saldrá un número en específico. El programa se cerrará cuando pulsemos Ctrl-C.



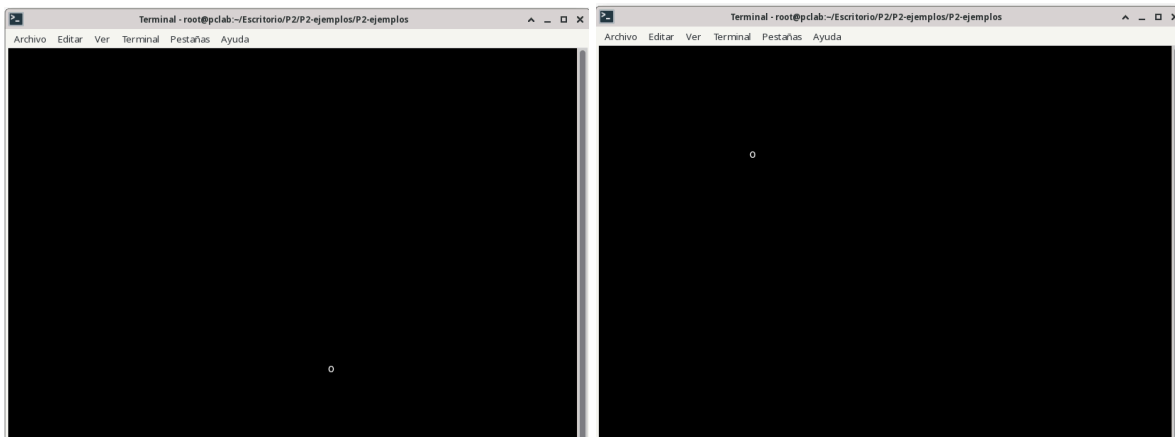
- rebota.c

Para compilar y ejecutar el programa haremos lo siguiente:



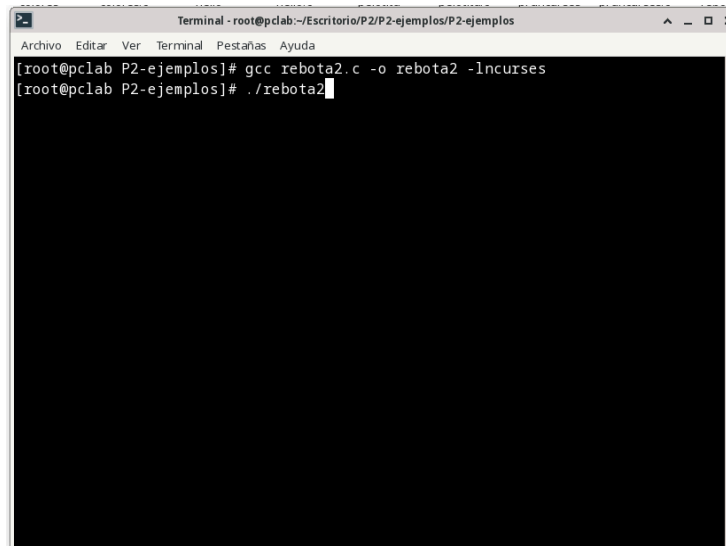
```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[root@pclab P2-ejemplos]# gcc rebota.c -o rebota -lncurses
[root@pclab P2-ejemplos]# ./rebota
```

Una vez ejecutado se nos abre una ventana en donde se puede apreciar una pelota que va rebotando a lo largo y ancho de la dicha ventana.



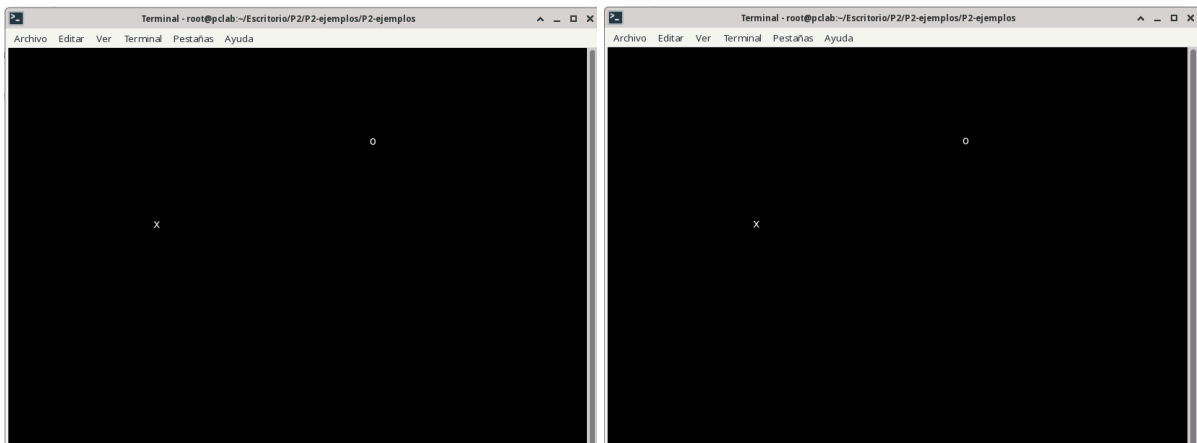
- rebota2.c

Para compilar y ejecutar el programa haremos lo siguiente:



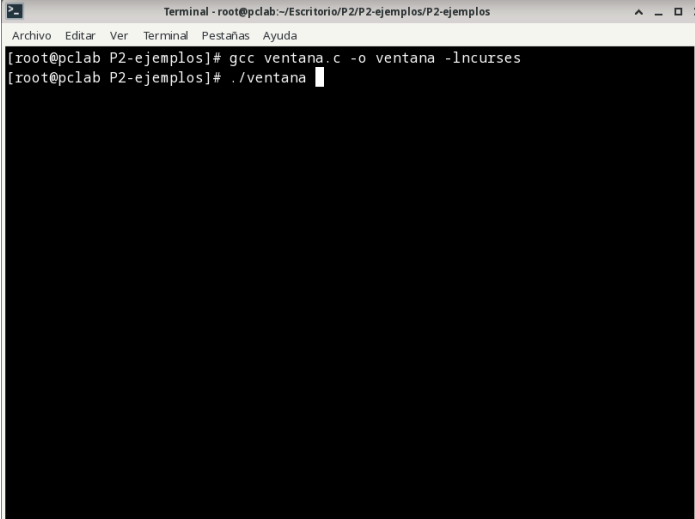
```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[root@pclab P2-ejemplos]# gcc rebota2.c -o rebota2 -lncurses
[root@pclab P2-ejemplos]# ./rebota2
```

Una vez ejecutado se nos abre una ventana en donde se puede apreciar una pelota que va rebotando a lo largo y ancho de dicha ventana. Además, aparece una x en una parte de la ventana, pareciendo un objetivo en donde la pelota deberá de tocar dicha x.



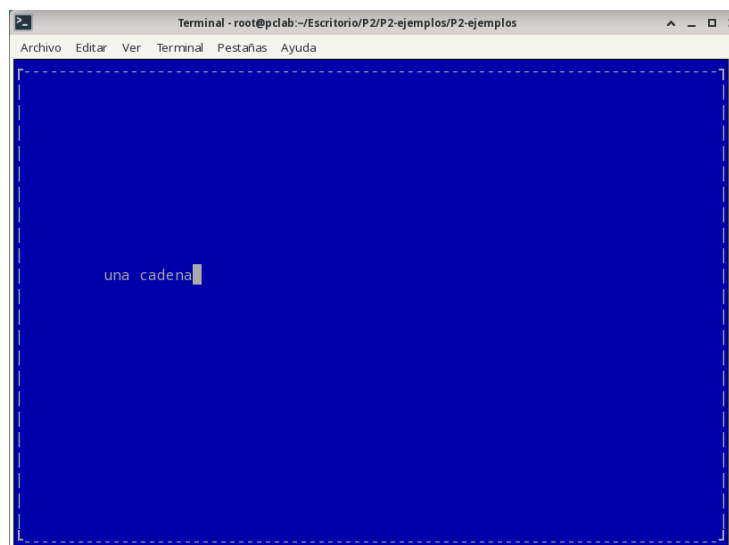
- ventana.c

Para compilar y ejecutar el programa haremos lo siguiente:



```
Terminal - root@pclab:~/Escritorio/P2/P2-ejemplos/P2-ejemplos
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[root@pclab P2-ejemplos]# gcc ventana.c -o ventana -lncurses
[root@pclab P2-ejemplos]# ./ventana
```

Una vez ejecutado se nos abre una ventana en donde se puede ver un texto que pone “una cadena”, además de que el color de fondo es azul y tiene unas líneas discontinuas como forma de recuadro, siendo las líneas del largo más alargadas que las del ancho.



3. Creación del juego “asteroids”

Hemos creado un juego de asteroides basado en terminal mediante la biblioteca ncurses para la visualización y control del juego. Se trata de que una nave vaya esquivando o eliminando con un láser aquellos asteroides que se pongan en su camino. Este programa ha sido dividido en las siguientes secciones:

- Definición de constantes y estructuras

ANCHO y *ALTO* establecen el tamaño del área del juego.

VELOCIDAD se encarga de indicar la velocidad a la que irá el juego.

MAX_ASTEROIDES es un límite de los asteroides que se pueden ver simultáneamente.

Hemos definido 3 estructuras:

- **Asteroide:** almacena la posición y su estado activo.
- **Nave:** almacena la posición de la nave (jugador).
- **Disparo:** son los láseres que dispara la nave.

- Mensaje inicial de bienvenida

Método que muestra antes de empezar el juego un mensaje de bienvenida al jugador, en donde se muestran los datos de los creadores del juego, explicando brevemente de qué va el juego y los controles necesarios para jugar.

```
void pantallaBienvenida() {
    clear();
    mvprintw(ALTO / 2 - 3, ANCHO / 2 - 10, "¡Bienvenido al juego!");
    mvprintw(ALTO / 2 - 1, ANCHO / 2 - 15, "Creado por: Clara Sola Ruiz y Mario Casas Perez.");
    mvprintw(ALTO / 2 + 1, ANCHO / 2 - 15, "Controles:");
    mvprintw(ALTO / 2 + 2, ANCHO / 2 - 15, "- Flechas: Mover");
    mvprintw(ALTO / 2 + 3, ANCHO / 2 - 15, "- Espacio: Disparar");
    mvprintw(ALTO / 2 + 4, ANCHO / 2 - 15, "- q: Salir");
    mvprintw(ALTO / 2 + 5, ANCHO / 2 - 15, "Explicacion: este juego consiste en destruir la mayor cantidad posible de asteroides para ganar. Si un asteroide choca con tu nave, pierdes.");
    mvprintw(ALTO / 2 + 7, ANCHO / 2 - 15, "Presiona cualquier tecla para empezar...");
    refresh();
    getch();
}
```

- Inicialización y actualización de asteroides

iniciaAsteroides(): se colocan los asteroides en posiciones aleatorias de la pantalla.

```
void iniciaAsteroides(Asteroides asteroides[], int n) {
    for (int i = 0; i < n; i++) {
        asteroides[i].x = rand() % ANCHO;
        asteroides[i].y = rand() % (ALTO / 2);
        asteroides[i].activo = 1;
    }
}
```

actualizaAsteroides(): hace que si los asteroides llegan a la parte de abajo de la pantalla, reaparezcan en la parte superior.

```
void actualizaAsteroides(Asteroides asteroides[], int n) {
    for (int i = 0; i < n; i++) {
        if (asteroides[i].activo) {
            asteroides[i].y++;
            if (asteroides[i].y >= ALTO) {
                asteroides[i].y = 0;
                asteroides[i].x = rand() % ANCHO;
                asteroides[i].activo = 1;
            }
        }
    }
}
```

- Control de la nave

moverNave(): permite que el jugador pueda mover la nave con las flechas del teclado.

```
void moverNave(Nave *nave, int tecla) {
    if (tecla == KEY_LEFT && nave->x > 0) nave->x--;
    if (tecla == KEY_RIGHT && nave->x < ANCHO - 1) nave->x++;
    if (tecla == KEY_UP && nave->y > ALTO / 2) nave->y--;
    if (tecla == KEY_DOWN && nave->y < ALTO - 1) nave->y++;
}
```

- Control de los disparos

iniciaDisparo(): permite que el láser salga desde la posición donde se encuentre la nave.

```
void iniciaDisparo(Disparo *disparo, Nave nave) {
    if (!disparo->activo) {
        disparo->x = nave.x;
        disparo->y = nave.y - 1;
        disparo->activo = 1;
    }
}
```

actualizaDisparo(): hace que el láser se mueva hacia arriba y que desaparezca si se sale de la pantalla.

```
void actualizaDisparo(Disparo *disparo) {
    if (disparo->activo) {
        disparo->y--;
        if (disparo->y < 0) disparo->activo = 0;
    }
}
```

- Colisiones

detectaColision(): es el método que se encarga de comprobar si el láser ha impactado en un asteroide, sumando 100 puntos si ha sido así.

```
int detectaColision(Asteroide *asteroide, Disparo *disparo) {
    if (disparo->activo && asteroide->activo &&
        disparo->x == asteroide->x && disparo->y == asteroide->y) {
        asteroide->activo = 0;
        disparo->activo = 0;
        return 100;
    }
    return 0;
}
```

detectaColisionNave(): comprueba si la nave ha chocado con un asteroide. Si esto pasa entonces el jugador ha perdido.

```
int detectaColisionNave(Nave *nave, Asteroide asteroides[], int n) {
    for (int i = 0; i < n; i++) {
        if (asteroides[i].activo && asteroides[i].x == nave->x && asteroides[i].y == nave->y) {
            return 1;
        }
    }
    return 0;
}
```

- main

Iniciamos el entorno de ncurses y mostramos el mensaje de bienvenida. Cuando el jugador pulsa una tecla se le indica que introduzca su nombre para que empiece a jugar. En un bucle, se manejan las entradas del jugador, actualizando los asteroides y disparos así como la detección de colisiones. Además usaremos mvprintw() para introducir caracteres por pantalla y usleep(VELOCIDAD) para controlar la velocidad del juego.

```

char nombre[20];
initscr();
noecho();
curs_set(FALSE);

pantallaBienvenida();

clear();

mvprintw(ALTO / 2, ANCHO / 2 - 10, "Dime tu nombre: ");
echo();
scanw("%19s", nombre);
noecho();

int jugar = 1;

while (jugar) {
    Nave nave = {ANCHO / 2, ALTO - 3};
    Asteroide asteroides[MAX_ASTEROIDES];
    iniciaAsteroides(asteroides, MAX_ASTEROIDES);
    Disparo disparo = {0, 0, 0};
    int puntuacion = 0;
    int game_over = 0;

    keypad(stdscr, TRUE);
    nodelay(stdscr, TRUE);
    srand(time(NULL));

    while (!game_over) {
        int tecla = getch();
        if (tecla == 'q') return 0;

        moverNave(&nave, tecla);
        if (tecla == ' ') iniciaDisparo(&disparo, nave);
        actualizaAsteroides(asteroides, MAX_ASTEROIDES);
        actualizaDisparo(&disparo);

        for (int i = 0; i < MAX_ASTEROIDES; i++) {
            puntuacion += detectaColision(&asteroides[i], &disparo);
        }

        if (detectaColisionNave(&nave, asteroides, MAX_ASTEROIDES)) game_over = 1;

        clear();
        mvprintw(nave.y, nave.x, "A ");

        if (disparo.activo) mvprintw(disparo.y, disparo.x, "|");
        for (int i = 0; i < MAX_ASTEROIDES; i++) {
            if (asteroides[i].activo) mvprintw(asteroides[i].y, asteroides[i].x, "**");
        }
        mvprintw(ALTO, 0, "Puntuación: %d", puntuacion);
        refresh();
        usleep(VELOCIDAD);
    }
}

```

Cuando el jugador choca con un asteroide, se muestra la pantalla de GAME OVER y se le pregunta si desea jugar de nuevo. Si pulsa “s” (o cualquier otra tecla que no sea “n”), entonces se iniciará una nueva partida, si pulsa “n” entonces se le agradecerá por haber jugador a nuestro juego y saldrá de este.

```

nodelay(stdscr, FALSE);
mvprintw(ALTO / 2, ANCHO / 2 - 5, "GAME OVER");
mvprintw(ALTO / 2 + 1, ANCHO / 2 - 8, "Puntuación: %d", puntuacion);
mvprintw(ALTO / 2 + 2, ANCHO / 2 - 12, "¿Jugar de nuevo? (s/n): ");
echo();
char respuesta;
scanw("%c", &respuesta);
noecho();
if (respuesta == 'n') jugar = 0;
}

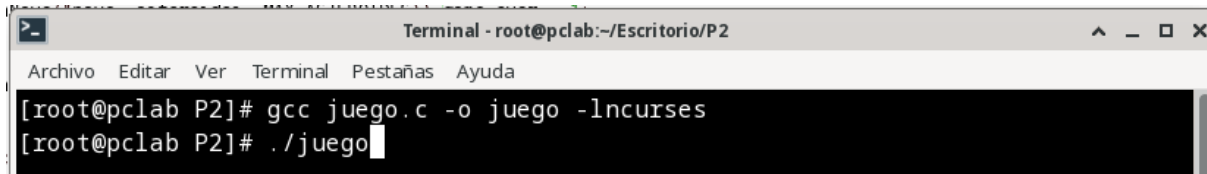
mvprintw(ALTO / 2 + 3, ANCHO / 2 - 12, "Gracias por jugar, %s!", nombre);
refresh();
getch();

endwin();
return 0;

```

- Salida por terminal

Para compilarlo y ejecutarlo, ejecutamos lo siguiente:



```
Terminal - root@pclab:~/Escritorio/P2
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[root@pclab P2]# gcc juego.c -o juego -lncurses
[root@pclab P2]# ./juego
```

Una vez iniciamos el juego se nos muestra la pantalla de inicio previamente descrita.



```
Terminal - root@pclab:~/Escritorio/P2
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

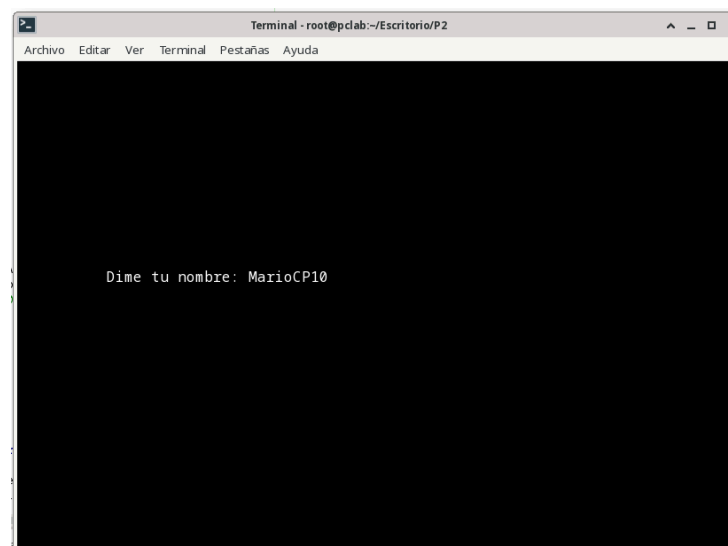
      ¡Bienvenido al juego!

Creado por: Clara Sola Ruiz y Mario Casas Perez.

Controles:
- Flechas: Mover
- Espacio: Disparar
- q: Salir
Explicacion: este juego consiste en destruir la mayor cantidad posible de asteroides para ganar. Si un asteroide choca con tu nave, pierdes.

Presiona cualquier tecla para empezar...
```

Pulsando a cualquier tecla, se mostrará en la pantalla un mensaje en donde le pedirá al jugador que inserte su nombre.

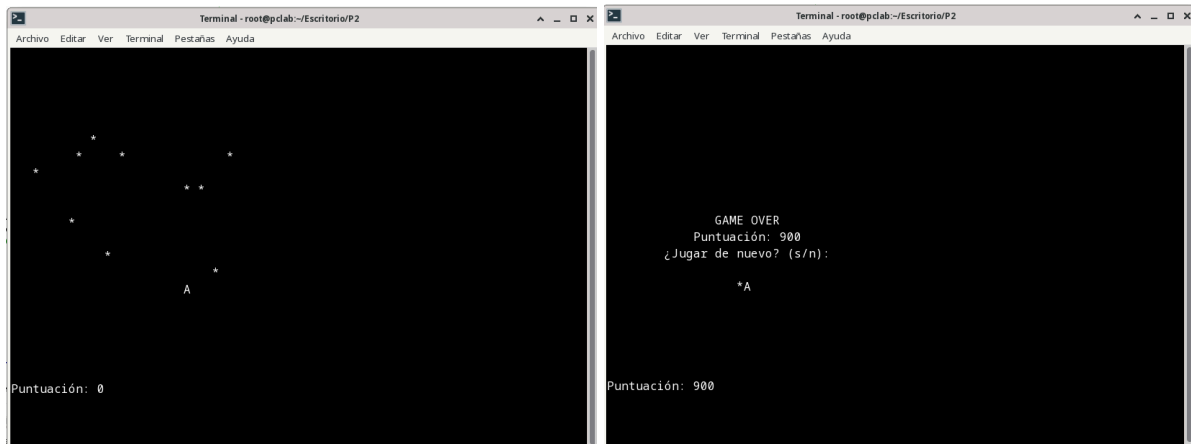


```
Terminal - root@pclab:~/Escritorio/P2
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

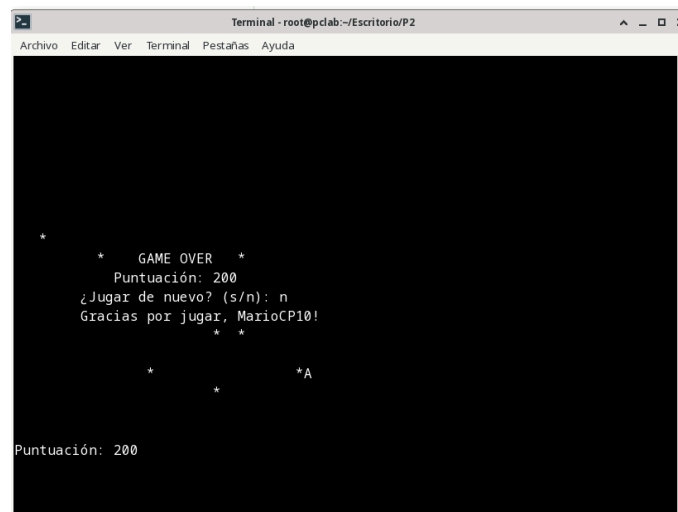
Dime tu nombre: MarioCP10
```


Práctica 2

Una vez puesto el nombre, el jugador pulsará “enter” y el juego iniciará. El jugador conforme vaya destruyendo asteroides irá ganando más puntos (100 puntos por cada asteroide destruido).



Si el jugador cuando pierde (choca con un asteroide) pulsa la tecla “s” (o cualquier otra tecla que no sea “n”) y a “enter”, se iniciará una nueva partida. En el caso de pulsar la tecla “n” y “enter”, entonces le saldrá el siguiente mensaje.



4. Creación del juego “Las aventuras de arroba”

Se ha querido crear otro juego el cual consiste en un mundo ficticio donde el personaje principal (representado mediante el símbolo @) deberá de eliminar todos los enemigos que hay en dicho mundo (representados con una E) en un tiempo límite de 2 minutos. El juego consta de 3 niveles y el jugador tendrá 3 vidas, las cuales se restaurarán en caso de que haya perdido alguna en los niveles anteriores (un enemigo le haya comido). Además, conforme el jugador vaya eliminando enemigos se irá actualizando su puntuación en el juego (cada enemigo eliminado son 100 puntos más). Este programa ha sido dividido en las siguientes secciones:

- Definición de constantes y estructuras

ANCHO y *ALTO*: determinan el tamaño del área del juego.

VELOCIDAD: controla la velocidad del juego en términos de actualización de movimientos.

Hemos definido 3 estructuras:

- **Jugador**: contiene las coordenadas del jugador, el número de vidas y la puntuación.
- **Enemigo**: contiene las coordenadas del enemigo y si está activo (si va en dirección al jugador).
- **Obstaculo**: contiene las coordenadas donde se encuentra el obstáculo correspondiente.

- Pantallas y dibujos

pantallaBienvenida(): método que muestra antes de empezar el juego un mensaje de bienvenida al jugador, en donde se muestran los datos de los creadores del juego, explicando brevemente de qué va el juego y los controles necesarios para jugar.

```
void pantallaBienvenida() {
    clear();
    mvprintw(ALTO / 2 - 3, ANCHO / 2 - 10, "¡Bienvenido al juego!");
    mvprintw(ALTO / 2 - 1, ANCHO / 2 - 15, "Creado por: Clara Sola Ruiz y Mario Casas Perez");
    mvprintw(ALTO / 2 + 1, ANCHO / 2 - 15, "Controles:");
    mvprintw(ALTO / 2 + 2, ANCHO / 2 - 15, "- Flechas: Mover");
    mvprintw(ALTO / 2 + 3, ANCHO / 2 - 15, "- Espacio: Atacar");
    mvprintw(ALTO / 2 + 4, ANCHO / 2 - 15, "- q: Salir");
    mvprintw(ALTO / 2 + 5, ANCHO / 2 - 15, "Explicacion: El bosque está lleno de enemigos y obstáculos.");
    mvprintw(ALTO / 2 + 6, ANCHO / 2 - 15, "Elimina a todos los enemigos en los 3 niveles para ganar antes de que se termine el tiempo.");
    mvprintw(ALTO / 2 + 8, ANCHO / 2 - 15, "Presiona cualquier tecla para empezar...");
    refresh();
    getch();
}
```

pantallaResumen(): muestra un mensaje de victoria o derrota al finalizar el juego con el resultado final que ha obtenido y preguntándole si quiere volver a jugar.

```
void pantallaResumen(const char *nombre, int score, int ganador, int por_tiempo) {
    clear();
    if (por_tiempo) {
        mvprintw(ALTO / 2 - 1, ANCHO / 2 - 10, "¡Se acabó el tiempo, %s!", nombre);
    } else if (ganador) {
        mvprintw(ALTO / 2 - 1, ANCHO / 2 - 10, "¡Felicidades %s, ganaste!", nombre);
    } else {
        mvprintw(ALTO / 2 - 1, ANCHO / 2 - 10, "GAME OVER, %s", nombre);
    }
    mvprintw(ALTO / 2 + 1, ANCHO / 2 - 15, "Puntuación final: %d", score);
    mvprintw(ALTO / 2 + 2, ANCHO / 2 - 15, "¿Quieres jugar de nuevo? (s/n): ");
    refresh();
}
```

dibujaBordes(): se encarga de dibujar una serie de bloques # para determinar el rango donde el jugador se podrá mover.

```
void dibujaBordes() {
    attron(COLOR_PAIR(4));
    for (int i = 0; i < ANCHO + 2; i++) {
        mvprintw(ALTO + 1, i, "#");
    }
    for (int i = 0; i <= ALTO; i++) {
        mvprintw(i, ANCHO + 1, "#");
    }
    attroff(COLOR_PAIR(4));
}
```

- Inicialización de elementos del juego

iniciaEnemigos(): se colocan los enemigos en posiciones aleatorias.

```
void iniciaEnemigos(Enemigo enemigos[], int n) {
    for (int i = 0; i < n; i++) {
        enemigos[i].x = rand() % ANCHO;
        enemigos[i].y = rand() % ALTO;
        enemigos[i].activo = 1;
    }
}
```

iniciaObstaculos(): se colocan los obstáculos en posiciones aleatorias.

```
void iniciaObstaculos(Obstaculo obstaculos[], int n) {
    for (int i = 0; i < n; i++) {
        obstaculos[i].x = rand() % ANCHO;
        obstaculos[i].y = rand() % ALTO;
    }
}
```

- Movimiento y colisiones

colisionConObstaculo(): método que se encarga que ni los enemigos ni el jugador puedan traspasar los obstáculos.

```
int colisionaConObstaculo(int x, int y, Obstaculo obstaculos[], int n) {
    for (int i = 0; i < n; i++) {
        if (x == obstaculos[i].x && y == obstaculos[i].y) return 1;
    }
    return 0;
}
```

moverJugador(): método que permite mover al jugador con las teclas de dirección del teclado.

```
void moverJugador(Jugador *jugador, int tecla, Obstaculo obstaculos[], int n_obstaculos) {
    int nuevo_x = jugador->x, nuevo_y = jugador->y;
    if (tecla == KEY_LEFT && jugador->x > 0) nuevo_x--;
    if (tecla == KEY_RIGHT && jugador->x < ANCHO - 1) nuevo_x++;
    if (tecla == KEY_UP && jugador->y > 0) nuevo_y--;
    if (tecla == KEY_DOWN && jugador->y < ALTO - 1) nuevo_y++;

    if (!colisionaConObstaculo(nuevo_x, nuevo_y, obstaculos, n_obstaculos)) {
        jugador->x = nuevo_x;
        jugador->y = nuevo_y;
    }
}
```

moverEnemigos(): método en el cual los enemigos van en dirección a donde se encuentra el jugador.

```
void moverEnemigos(Enemigo enemigos[], int n, Jugador jugador, Obstaculo obstaculos[], int n_obstaculos) {
    for (int i = 0; i < n; i++) {
        if (enemigos[i].activo) {
            int dx = jugador.x - enemigos[i].x;
            int dy = jugador.y - enemigos[i].y;
            int nuevo_x = enemigos[i].x + (dx != 0 ? (dx > 0 ? 1 : -1) : 0);
            int nuevo_y = enemigos[i].y + (dy != 0 ? (dy > 0 ? 1 : -1) : 0);

            if (!colisionaConObstaculo(nuevo_x, nuevo_y, obstaculos, n_obstaculos)) {
                enemigos[i].x = nuevo_x;
                enemigos[i].y = nuevo_y;
            }
        }
    }
}
```

atacar(): el jugador puede eliminar a los enemigos pulsando la barra espaciadora del teclado si están cerca de él. Si el jugador elimina un enemigo, se le sumará 100 puntos a su puntuación en el juego.

```
void atacar(Jugador *jugador, Enemigo enemigos[], int n) {
    for (int i = 0; i < n; i++) {
        if (enemigos[i].activo && abs(jugador->x - enemigos[i].x) <= 1 && abs(jugador->y - enemigos[i].y) <= 1) {
            enemigos[i].activo = 0;
            jugador->score += 100;
        }
    }
}
```

verificaColisionEnemigos(): si el jugador choca con un enemigo, entonces el jugador perderá una vida.

```
int verificaColisionEnemigos(Jugador *jugador, Enemigo enemigos[], int n) {
    for (int i = 0; i < n; i++) {
        if (enemigos[i].activo && jugador->x == enemigos[i].x && jugador->y == enemigos[i].y) {
            jugador->vidas--;
            enemigos[i].activo = 0;
            return 1;
        }
    }
    return 0;
}
```

- main

Iniciamos el entorno de ncurses, configurando los diversos colores de fondo y mostrando el mensaje de bienvenida. Cuando el jugador pulsa una tecla se le indica que introduzca su nombre para que empiece a jugar. En un bucle, se controlará que cuando ya no queden enemigos en un nivel, se avance al siguiente, teniendo siempre en cuenta el tiempo que le queda al jugador para completar estos 3 niveles. Además, en el bucle se estará comprobando las vidas que tiene el jugador y los enemigos restantes, así como la puntuación del jugador. Finalmente, si el jugador gana (que haya completado los 3 niveles) o pierde (que haya perdido las 3 vidas o se le haya acabado el tiempo), se le mostrará un resumen mostrando su puntuación final y preguntando al jugador si quiere volver a jugar. Usaremos mvprintw() para introducir caracteres por pantalla y usleep(VELOCIDAD) para controlar la velocidad del juego.

```
initscr();
start_color();
init_pair(1, COLOR_GREEN, COLOR_BLUE);
init_pair(2, COLOR_RED, COLOR_BLUE);
init_pair(3, COLOR_WHITE, COLOR_BLACK);
init_pair(4, COLOR_WHITE, COLOR_BLUE);
noecho();
curs_set(FALSE);
keypad(stdscr, TRUE);
srand(time(NULL));

pantallaBienvenida();

clear();
mvprintw(ALTO / 2, ANCHO / 2 - 10, "Dime tu nombre: ");
refresh();

char nombre[20];
echo();
scanw("%19s", nombre);
noecho();

int jugar = 1;
```

Práctica 2

```
while (jugar) {
    int nivel = 1;
    int enemigos_por_nivel[] = {5, 7, 10};
    int ganador = 0;

    time_t inicio_tiempo = time(NULL);
    const int TIEMPO_LIMITE = 120;
    Jugador jugador = {ANCHO / 2, ALTO / 2, 3, 0};
    while (nivel <= 3) {
        Enemigo enemigos[enemigos_por_nivel[nivel - 1]];
        Obstaculo obstaculos[nivel + 3];
        iniciaEnemigos(enemigos, enemigos_por_nivel[nivel - 1]);
        iniciaObstaculos(obstaculos, nivel + 3);

        int game_over = 0;

        while (!game_over) {
            time_t tiempo_actual = time(NULL);
            int tiempo_restante = TIEMPO_LIMITE - (int)(difftime(tiempo_actual, inicio_tiempo));

            if (tiempo_restante <= 0) {
                pantallaResumen(nombre, jugador.score, 0, 1);
                echo();
                char respuesta;
                scanw("%c", &respuesta);
                noecho();
                if (respuesta == 'n') {
                    bkgdset(COLOR_PAIR(0));
                    clear();
                    refresh();
                    endwin();
                    return 0;
                }
            }

            for (int i = 0; i < enemigos_por_nivel[nivel - 1]; i++) {
                if (enemigos[i].activo) {
                    attron(COLOR_PAIR(2));
                    mvprintw(enemigos[i].y, enemigos[i].x, "E");
                    attroff(COLOR_PAIR(2));
                }
            }

            attron(COLOR_PAIR(3));
            for (int i = 0; i < nivel + 3; i++) {
                mvprintw(obstaculos[i].y, obstaculos[i].x, "#");
            }
            attron(COLOR_PAIR(3));

            dibujaBordes();

            attron(COLOR_PAIR(3));
            mvprintw(ALTO + 2, 1, "Vidas: %d | Nivel: %d | Tiempo restante: %d segundos | Score: %d",
                jugador.vidas, nivel, tiempo_restante, jugador.score);
            attron(COLOR_PAIR(3));

            refresh();

            if (jugador.vidas <= 0) {
                game_over = 1;
                break;
            }

            int enemigos_restantes = 0;
            for (int i = 0; i < enemigos_por_nivel[nivel - 1]; i++)
                enemigos_restantes += enemigos[i].activo;

            if (enemigos_restantes == 0) {
                nivel++;
                break;
            }

            usleep(VELOCIDAD);
        }

        if (jugador.vidas <= 0)
            break;

        if (nivel > 3 && ganador == 0) {
            ganador = 1;
        }

        pantallaResumen(nombre, jugador.score, ganador, 0);
        echo();
        char respuesta;
        scanw("%c", &respuesta);
        noecho();
        if (respuesta == 'n') {
            bkgdset(COLOR_PAIR(0));
            clear();
            refresh();
            jugar = 0;
            endwin();
            return 0;
        }
    }
    endwin();
    return 0;
}

} else {
    nivel = 1;
    ganador = 0;
    inicio_tiempo = time(NULL);
    game_over = 1;
    continue;
}

}

int tecla = getch();
if (tecla == 'q') return 0;

moverJugador(&jugador, tecla, obstaculos, nivel + 3);
moverEnemigos(enemigos, enemigos_por_nivel[nivel - 1], jugador, obstaculos, nivel + 3);
if (tecla == 'a') atacar(&jugador, enemigos, enemigos_por_nivel[nivel - 1]);
verificaColisionEnemigos(&jugador, enemigos, enemigos_por_nivel[nivel - 1]);

clear();

attron(COLOR_PAIR(4));
for (int y = 0; y <= ALTO; y++) {
    for (int x = 0; x <= ANCHO; x++) {
        mvprintw(y, x, " ");
    }
}
attron(COLOR_PAIR(4));

attron(COLOR_PAIR(1));
mvprintw(jugador.y, jugador.x, "O");
attron(COLOR_PAIR(1));

if (enemigos_restantes == 0) {
    nivel++;
    break;
}

usleep(VELOCIDAD);

if (jugador.vidas <= 0)
    break;

if (nivel > 3 && ganador == 0) {
    ganador = 1;
}

pantallaResumen(nombre, jugador.score, ganador, 0);
echo();
char respuesta;
scanw("%c", &respuesta);
noecho();
if (respuesta == 'n') {
    bkgdset(COLOR_PAIR(0));
    clear();
    refresh();
    jugar = 0;
    endwin();
    return 0;
}
}
endwin();
return 0;
}
```

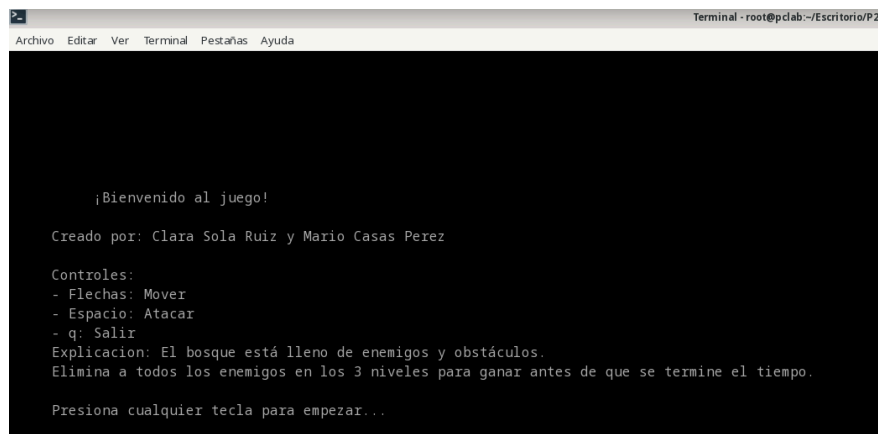
● Salida por terminal

Para compilarlo y ejecutarlo, ejecutamos lo siguiente:

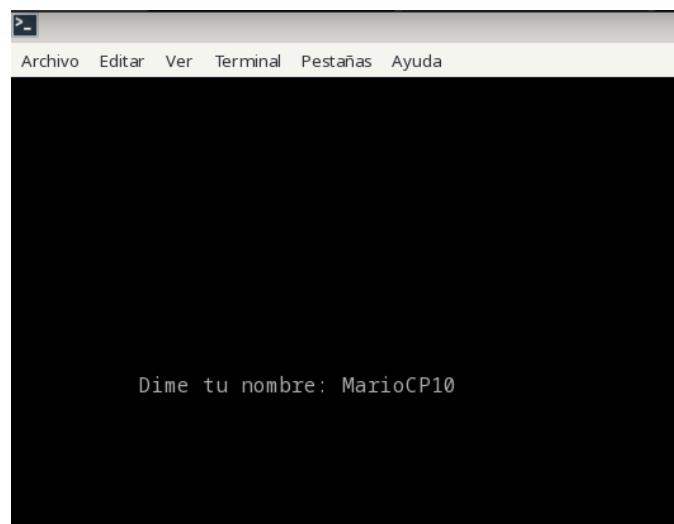
```
>
Archivo Editar Ver Terminal Pestañas Ayuda
[root@pclab P2]# gcc juego1.c -o juego1 -lncurses
[root@pclab P2]# ./juego1
```

Práctica 2

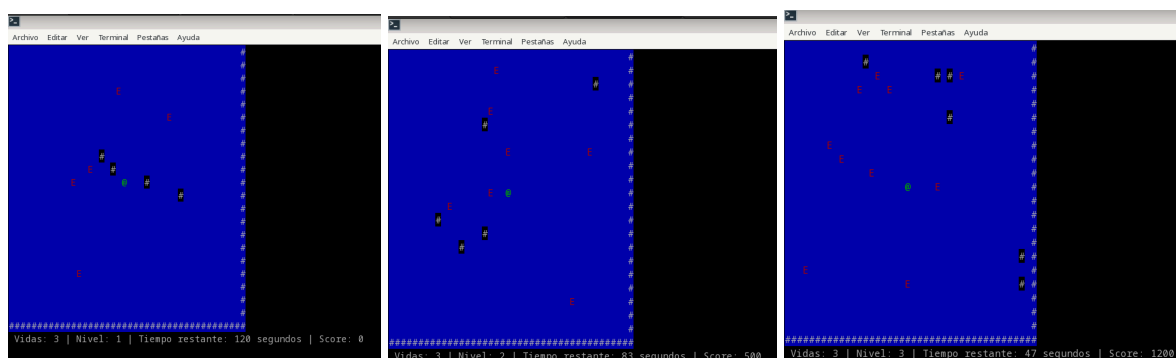
Una vez iniciamos el juego se nos muestra la pantalla de inicio previamente descrita



Pulsando a cualquier tecla, se mostrará en la pantalla un mensaje en donde le pedirá al jugador que inserte su nombre.

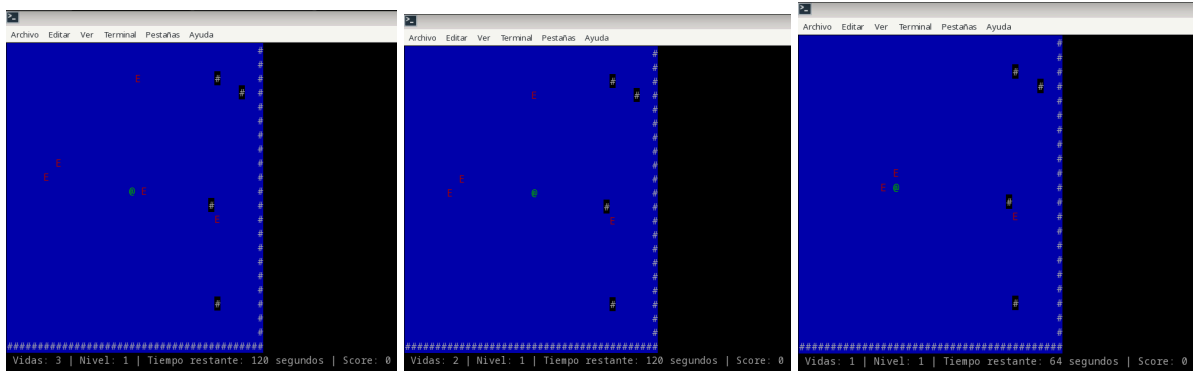


Una vez puesto el nombre, el jugador pulsará “enter” y el juego iniciará. Recordemos que hay tres niveles los cuales el jugador deberá de pasar para ganar.

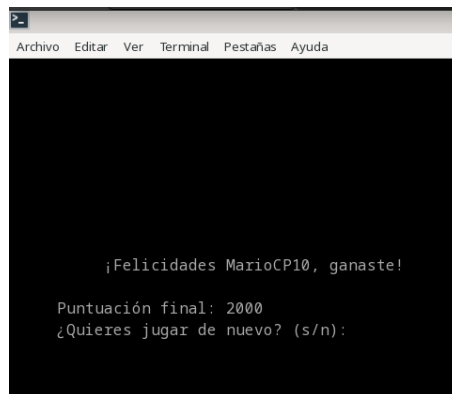


Práctica 2

Si un enemigo colisiona con el jugador, éste perderá una vida.



Si el jugador gana, se le felicitará por haber completado los 3 niveles y se le preguntará si desea jugar de nuevo. Si escribimos “s” (o cualquier otra tecla que no sea “n”), entonces se iniciará una nueva partida, si se pulsa “n” entonces saldrá del juego.



En el caso de que el jugador pierda por tiempo o por perder todas las vidas, se mostrará un mensaje indicando que ha perdido y si quiere volver a jugar de nuevo. Igual que antes, si el jugador pulsa “s” (o cualquier otra tecla que no sea “n”) entonces se iniciará una nueva partida, si se pulsa “n” entonces saldrá del juego.



5. Juego Serpiente

Este juego consiste en el famoso juego de la serpiente ("Snake") que trata de comerse una manzana, en este caso un "@" y va sumando puntos. Tiene que evitar chocarse con las paredes o consigo misma para no perder.

- Definición de constantes y variables globales

En este caso hemos definido una constante llamada *VELOCIDAD* que se encarga de definir el tiempo en microsegundos entre movimientos.

Hemos definido un **struct**, que representa un punto con sus coordenadas correspondientes *x* e *y*.

También contamos con una serie de variables globales, que son la serpiente representada por el struct de Punto y con una longitud máxima de 100 y su largo inicial de la serpiente es de 5. La comida es representada también como un punto.

Los movimientos se representan como *dir_x* y *dir_y*, y cuenta con un movimiento inicial hacia la derecha.

En este caso las variables que definen el *ANCHO* y la *ALTURA* se definen como variables globales, que serán calculadas posteriormente con la función *getmaxyx(stdscr, ALTO, ANCHO)*.

```
#define VELOCIDAD 100000 // Tiempo en microsegundos entre movimientos

typedef struct
{
    int x, y;
} Punto; // Define un tipo Punto que representa la serpiente y la comida

Punto serpiente[100];
int largo_serpiente = 5;
Punto comida;
int dir_x = 1, dir_y = 0; // Movimiento inicial hacia la derecha
int ANCHO, ALTO;
```

Ahora damos paso a las funciones:

- `init_game()`

La función como su nombre indica se encarga de inicializar el estado del juego:

- Establece el tamaño inicial de la serpiente.
- Define que la serpiente comienza moviéndose hacia la derecha.
- Coloca la serpiente en el centro de la pantalla: para ello recorre un bucle, que es el largo de la serpiente y la coloca en mitad de la pantalla, colocando el resto del cuerpo detrás de la cabeza (- *i*).
- Genera la primera comida aleatoriamente en el tablero, llamando a la función `generar_comida()`.

```
void init_game()
{
    largo_serpiente = 5; // Largo de la serpiente
    dir_x = 1;           // Mirando a la derecha
    dir_y = 0;

    // Colocar la serpiente en mitad de la pantalla
    for (int i = 0; i < largo_serpiente; i++)
    {
        serpiente[i].x = ANCHO / 2 - i;
        serpiente[i].y = ALTO / 2;
    }

    generar_comida(); // Tenemos que generar la primera comida
}
```

- `dibujar_juego()`

La función pinta el estado actual del juego en la pantalla:

- Limpiamos la pantalla con `clear()` para dibujarla desde cero.
- Dibujamos un borde alrededor de la ventana de juego utilizando `box()`.
- Se muestra la puntuación en la parte superior izquierda y esta se calcula como el largo actual de la serpiente menos el largo inicial (`largo_serpiente - 5`).
- Dibuja la comida en su posición actual (`comida.x`, `comida.y`) utilizando el símbolo @.
- Recorre todos los segmentos de la serpiente:
 - El primer segmento (la cabeza) se dibuja con el carácter O.
 - El resto del cuerpo se dibuja con el carácter o.
 - Finalmente, se refresca la pantalla con `refresh()` para aplicar todos los cambios visuales.

```

void dibujar_juego()
{
    clear();
    box(stdscr, 0, 0);

    // Mostrar el puntaje en la parte superior de la pantalla
    // La puntuación será el largo de la serpiente - 5 porque es como se empieza
    mvprintw(0, 2, " SNAKE | Puntuación: %d ", largo_serpiente - 5);

    // Pinta la comida
    mvaddch(comida.y, comida.x, '@');

    for (int i = 0; i < largo_serpiente; i++)
    {
        if (i == 0)
        {
            mvaddch(serpiente[i].y, serpiente[i].x, 'O'); // Dibuja la cabeza con 'O'
        }
        else
        {
            mvaddch(serpiente[i].y, serpiente[i].x, 'o'); // Dibuja el cuerpo con 'o'
        }
    }
    refresh();
}

```

● mover_serpiente()

Desplaza la serpiente en la dirección actual, gestiona el crecimiento si come y actualiza su cuerpo.

- Calculamos la nueva posición de la cabeza sumando *dir_x* y *dir_y* a sus coordenadas actuales.
- Se recorre el arreglo de la serpiente desde la cola hasta la cabeza y se mueve cada segmento a la posición del anterior, simulando el "deslizamiento" de su cuerpo.
- Se coloca la nueva cabeza en la posición calculada.
- Se comprueba si la cabeza ha llegado a la posición de la comida:
 - Si es así, se incrementa el largo de la serpiente.
 - Se llama a *generar_comida()* para crear una nueva comida en otra ubicación válida.

```

void mover_serpiente()
{
    // Calcular la nueva posición de la cabeza sumando las direcciones dir_x y dir_y a las coordenadas de la cabeza
    Punto new_head = {serpiente[0].x + dir_x, serpiente[0].y + dir_y};

    // Cada posición toma la posición del segmento anterior, deslizando el cuerpo, hasta que ocupe la cabeza.
    for (int i = largo_serpiente; i > 0; i--)
    {
        serpiente[i] = serpiente[i - 1];
    }

    // La nueva cabeza se coloca en la posición calculada
    serpiente[0] = new_head;

    // Comprobar si la serpiente ha comido
    if (serpiente[0].x == comida.x && serpiente[0].y == comida.y)
    {
        largo_serpiente++;
        generar_comida();
    }
}

```

- comprobar_colision()

Su propósito es comprobar si la serpiente ha chocado contra los bordes del tablero o contra sí misma.

- Se comprueba si la cabeza ha tocado los límites del borde (coordenadas fuera del área de juego). Si ocurre, devuelve 1, indicando colisión.
- Se recorre el cuerpo de la serpiente (a partir del segundo segmento) para ver si alguno de ellos coincide con la posición de la cabeza. Esto significa que la serpiente se ha mordido a sí misma.
- Si hay colisión, se devuelve 1; si no, 0.

```
int comprobar_colision()
{
    // Comprobar el choque con bordes
    if (serpiente[0].x <= 0 || serpiente[0].x >= ANCHO - 1 ||
        serpiente[0].y <= 0 || serpiente[0].y >= ALTO - 1)
        return 1;

    // Comprobar el choque consigo misma
    for (int i = 1; i < largo_serpiente; i++)
    {
        if (serpiente[i].x == serpiente[0].x && serpiente[i].y == serpiente[0].y)
            return 1;
    }
    return 0;
}
```

- generar_comida()

Genera una nueva comida en una posición aleatoria del tablero que no esté ocupada por la serpiente.

- Se utiliza un bucle que se repite hasta encontrar una posición válida que está representada por un bool, *valid*.
- Se genera una posición aleatoria dentro de los límites válidos del tablero (evitando los bordes).
- Se asume que la posición es válida, pero luego se verifica si esa posición coincide con algún segmento del cuerpo de la serpiente.
 - Si hay coincidencia, se marca como inválida y se repite el proceso.
- Una vez encontrada una posición válida, se almacena en la variable comida, con sus coordenadas.

```

void generar_comida()
{
    bool valid = false;
    while (!valid)
    {
        comida.x = rand() % (ANCHO - 2) + 1;
        comida.y = rand() % (ALTO - 2) + 1;

        valid = true; // asumimos que es válida

        // Recorremos el largo de la serpiente para no colocar comida en el cuerpo
        for (int i = 0; i < largo_serpiente; i++)
        {
            if (comida.x == serpiente[i].x && comida.y == serpiente[i].y)
            {
                valid = false; // Si choca salimos y volvemos a calcular una nueva posición de comida
                break;
            }
        }
    }
}

```

- `pantalla_bienvenida()`

Aquí se muestra la pantalla de bienvenida al jugador, enseñando los nombres de los creadores y las instrucciones que hay que seguir.

Al principio calculamos el tamaño de la pantalla para poder insertar el texto en la mitad de la terminal. Se hace uso de la función `getmaxyx(stdscr, y_max, x_max)`.

```

void pantalla_bienvenida()
{
    clear();

    int y_max, x_max;
    getmaxyx(stdscr, y_max, x_max);
    int mid_y = y_max / 2;
    int mid_x = x_max / 2;

    box(stdscr, 0, 0); // dibuja un recuadro

    mvprintw(mid_y - 4, mid_x - 20, "=== Bienvenido a SERPIENTE en NCURSES ===");
    mvprintw(mid_y - 2, mid_x - 22, "Autores: Clara Sola Ruiz y Mario Casas Perez");

    mvprintw(mid_y, mid_x - 10, "Controles del juego:");
    mvprintw(mid_y + 1, mid_x - 12, "- Flechas: Mover la serpiente");
    mvprintw(mid_y + 3, mid_x - 16, "Come comida para crecer.");
    mvprintw(mid_y + 4, mid_x - 18, "Evita chocar contigo misma o los bordes.");

    mvprintw(mid_y + 6, mid_x - 22, "Presiona cualquier tecla para comenzar...!");

    refresh();

    timeout(-1); // Espera una tecla sin límite de tiempo
    getch();     // Espera la tecla del usuario
    timeout(0);  // Vuelve al modo sin bloqueo para el juego
}

```

- game_over()

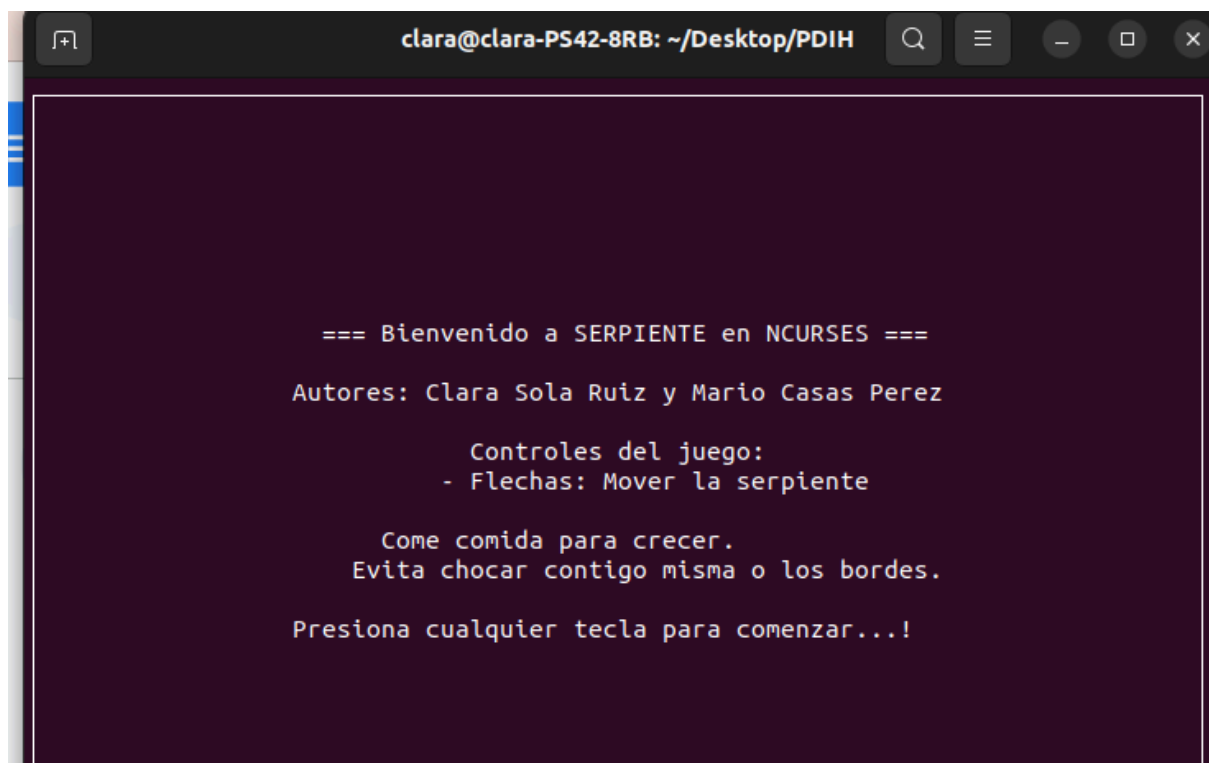
Aquí se muestra la pantalla de fin de juego con la puntuación correspondiente y se le ofrece al jugador volver a jugar o salir definitivamente del juego.

```
void game_over()
{
    clear();
    box(stdscr, 0, 0);
    mvprintw(ALTO / 2 - 2, ANCHO / 2 - 6, "¡Juego terminado!");
    mvprintw(ALTO / 2, ANCHO / 2 - 8, "Puntuación: %d", largo_serpiente - 5);
    mvprintw(ALTO / 2 + 2, ANCHO / 2 - 11, "¿Jugar otra vez? (s/n)");
    refresh();

    int c;
    while ((c = getch())) // Ver si se continua o no el juego.
    {
        if (c == 's' || c == 'S')
        {
            init_game();
            return;
        }
        else if (c == 'n' || c == 'N')
        {
            endwin();
            exit(0);
        }
    }
}
```

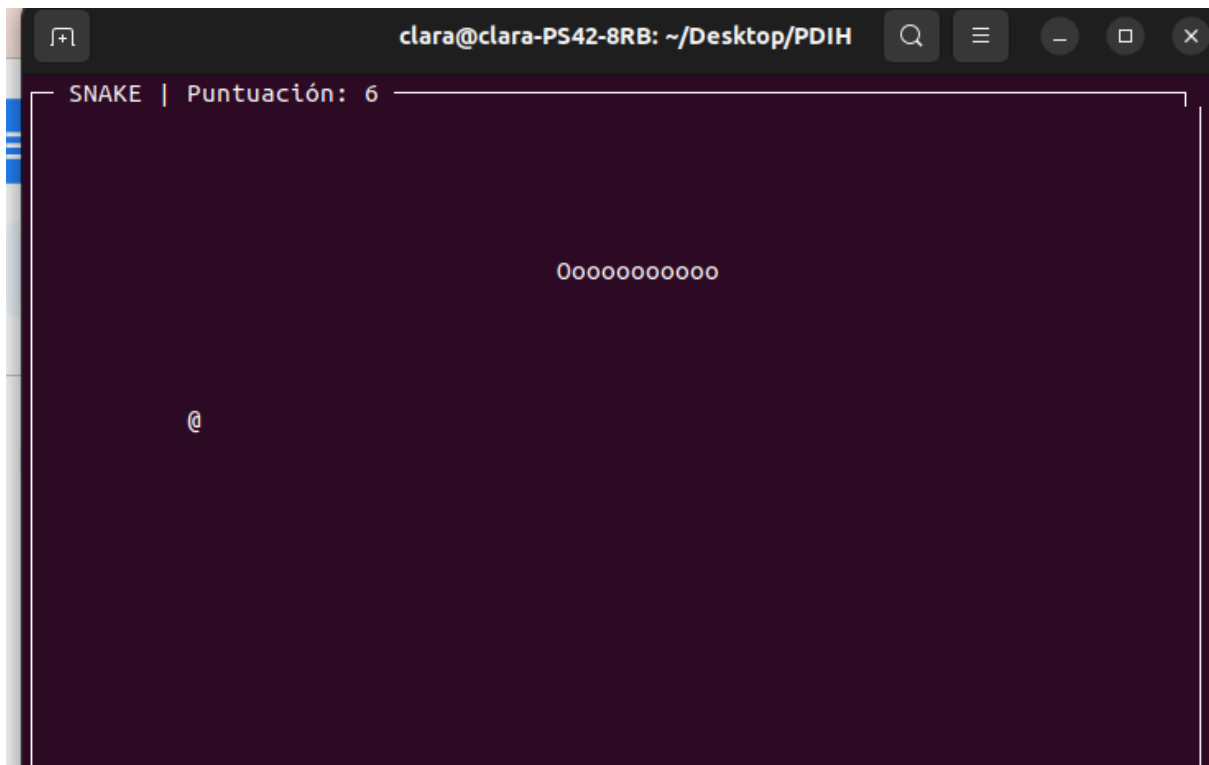
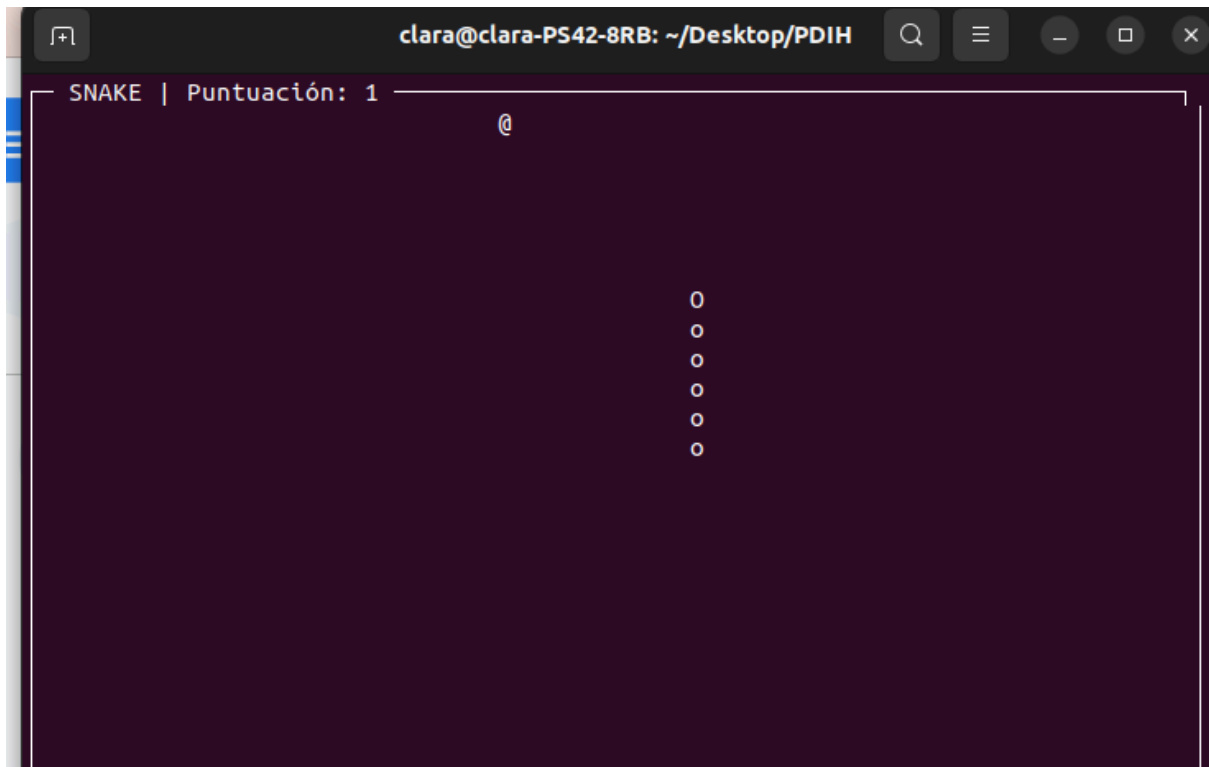
- Capturas de pantalla

Aquí podemos ver la pantalla de bienvenida:

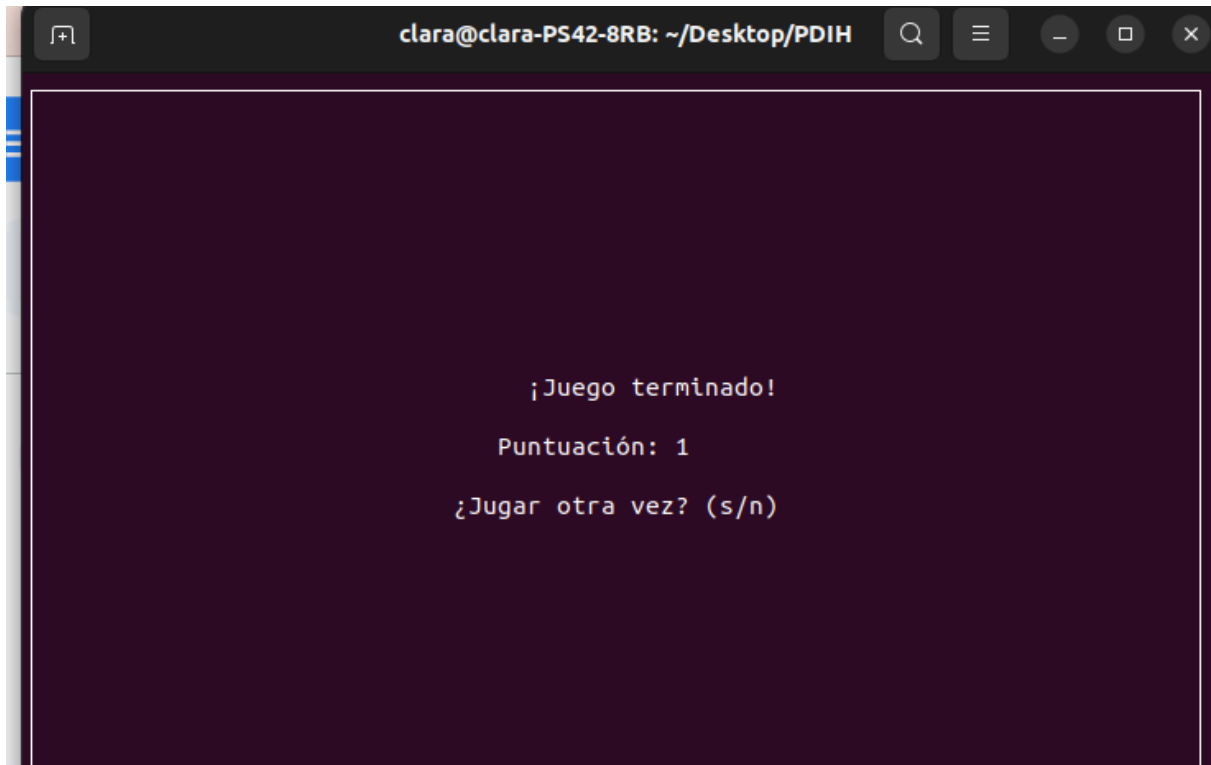


Práctica 2

Capturas de pantalla del procedimiento del juego:



Capturas de pantalla de cuando el juego ha terminado:

A terminal window with a dark purple background and white text. The window title bar shows the user 'clara' on a machine named 'clara-PS42-8RB' in the directory '~/Desktop/PDIH'. The terminal displays the message '¡Juego terminado!' followed by 'Puntuación: 1' and a prompt '¿Jugar otra vez? (s/n)'.

```
clara@clara-PS42-8RB: ~/Desktop/PDIH  
  
¡Juego terminado!  
Puntuación: 1  
¿Jugar otra vez? (s/n)
```