UNIVERSIDAD DEL VALLE DE GUATEMALA

*Departamento de Ingeniería en Ciencias de la Computación*

*y Tecnologías de la Información*

Ing. Kimberly Marisol Barrera González

# 

**Proyecto 1**

**Snake**

Integrantes:

Adriana Martínez - 24086

Mishell Ciprian -231169

Belén Monterroso - 231497

Pablo Andrés Cabrera Argüello - 231156

**Fase 1: Análisis del Juego Snake**

* **Funcionamiento general de juego:**

Al ejecutar el programa lo primero que aparecerá será un menú con las principales opciones para el jugador. Este menú incluirá:

* Nombre del juego
* Modo de juego (1 jugador o 2)
* Selección de nivel de dificultad.
* Salir del juego.

Al elegir un modo de juego se mostrarán también unas breves instrucciones donde se explica cómo se controla la serpiente y en que consiste la dinámica. Antes de comenzar la partida, el sistema hará una pausa corta, ya sea usando un temporizador o esperando a que el jugador presione una tecla para confirmar que está listo para iniciar le juego.

Una vez iniciada la partida, la serpiente aparecerá en el centro del tablero. Durante la partida se irán generando pelotas (puntos de comida) que la serpiente podrá recolectar para alargarse y sumar su puntaje. También existirán trampas que si, son tocadas, terminaran la partid inmediatamente. El jugador pierde cuando la serpiente choca con si misma o contra las paredes u obstáculos.

* **Elementos principales del juego:** 
  + 1 o 2 serpientes, según la modalidad elegida por el usuario.
  + Pelotas de comida, que aumentan la longitud de la serpiente y suman puntos.
  + Parades que delimitan el área del tablero.
  + Trampas que al ser tocadas, finaliza la partida.
  + Un tablero donde indica el nivel de juego y el tiempo que queda para cada partida.
* **Interacción y controles:**

Al inicio, el jugador debe seleccionar si jugará solo o con un compañero. Para hacerlo, puede presionar la tecla **1** (un jugador), **2** (dos jugadores) o **3** (salir del juego).

Después, seleccionará el nivel de dificultad, lo cual define la velocidad de la serpiente y la cantidad de obstáculos que aparecerán.

* En el **modo de un jugador**, los controles son:
  + **W** → arriba
  + **A** → izquierda
  + **S** → abajo
  + **D** → derecha
* En el **modo de dos jugadores**:
  + El **jugador 1** usará las teclas **WASD**.
  + El **jugador 2** usará las **flechas de dirección** (← ↑ ↓ →).

Cuando el jugador pierde, en pantalla se mostrará un mensaje de **“GAME OVER”**.

* **Cálculos necesarios:**

Para garantizar que el juego funcione correctamente, deben realizarse algunos cálculos:

* Determinar posiciones válidas para que la comida y las trampas no aparezcan demasiado cerca de la serpiente al inicio.
* Calcular la nueva longitud de la serpiente cada vez que coma una pelota.
* Llevar un puntaje acumulado de la cantidad de pelotas comidas.
* Verificar en cada movimiento la posición de la serpiente para evitar que atraviese las paredes o se choque consigo misma.
* **Representación visual:**

Los elementos se mostrarán en la consola utilizando caracteres ASCII:

* La serpiente estará formada por bloques rectangulares consecutivos.
* Las pelotas de comida se mostrarán como puntos o círculos rellenos.
* Las trampas serán círculos huecos o algún símbolo que confunda al jugador.
* Las paredes estarán dibujadas como líneas rectangulares que delimitan el campo de juego.
* **Requisitos funcionales**

El programa debe cumplir con los siguientes requisitos:

* Mostrar correctamente el menú principal al iniciar.
* Permitir seleccionar entre uno o dos jugadores.
* Gestionar el movimiento de la serpiente y la generación de comida/trampas.
* Mostrar el puntaje y la cantidad de vidas disponibles.
* Finalizar la partida y mostrar un mensaje de “GAME OVER” cuando corresponda.
* **Investigación técnica**

Para capturar las teclas en tiempo real, usaremos ncurses, una librería de consola disponible en Linux/WSL que nos da: (1) control de pantalla para dibujar en posiciones exactas sin parpadeo y (2) entrada no bloqueante. Al iniciar, configuramos la terminal con initscr(); cbreak(); noecho(); keypad(stdscr, TRUE); nodelay(stdscr, TRUE);. Con esto, getch() devuelve ERR si no hay tecla y, cuando sí hay, entrega el carácter ('w', 'a', 's', 'd') o un código especial para flechas (KEY\_UP, KEY\_DOWN, etc.).

Tendremos un hilo de entrada dedicado que corre en bucle leyendo getch(), mapea la tecla a una dirección válida y la guarda en el estado del juego. Esa dirección se protege con un mutex de estado (pthread\_mutex\_t) para evitar condiciones de carrera con los hilos de la(s) serpiente(s) que la leen en cada “tick”. Además, como ncurses no es thread-safe, usamos un segundo mutex de pantalla alrededor de llamadas a getch(), mvaddch/mvprintw y refresh, de modo que nunca dibujen/lean dos hilos a la vez. Con este esquema el juego no se congela esperando teclado, el render es estable y los movimientos son consistentes (incluimos lógica para rechazar giros de 180°).

* **Lista de funciones** <ncurses.h>

|  |  |  |
| --- | --- | --- |
| Función | ¿ Qué hace? | Uso en el juego |
| Initscr() | Inicializa ncruses y toma control de la terminal | Arranque del modo consola para dibujar/leer teclas. |
| Endwin() | Cierra ncurses y restaura la terminal. | Evita que la consola se trabe. |
| Cbreak () | Para evitar usar enter al usar cada tecla | Input responsivo en el tiempo real. |
| Noecho() | Evita que las teclas se impriman en pantalla | Que el HUD no se ensucie con el input. |
| keypad(stdscr, TRUE) | Hace que getch() retorne códigos de flechas (KEY\_\*). | Soportar flechas para el Jugador 2. |
| nodelay(stdscr, TRUE) | Vuelve no bloqueante a getch(). | Si no hay tecla, el juego sigue (ERR). |
| timeout(ms) | Límite de espera para getch(). | Alternativa a nodelay para polling con timeout. |
| getch() | Lee la tecla: letra (w/a/s/d) o KEY\_\*; ERR si no hay. | Hilo de entrada: captura y mapea dirección. |
| mvaddch(y,x,ch) | Dibuja un carácter en (y,x). | Render de cabeza/cuerpo/comida/trampas. |
| mvprintw(y,x,fmt,...) | Imprime texto formateado en (y,x). | Puntajes, tiempo restante, mensajes. |
| refresh() | Refresca la pantalla con lo dibujado. | Actualiza cada frame en consola. |
| clear() | Limpia toda la pantalla. | Pantallas como Game Over/reinicio. |
| pthread\_create(...) | Crea un nuevo hilo POSIX. | Hilos: serpiente(s), entrada, comida, temporizador, trampas. |
| pthread\_join(...) | Espera la finalización de un hilo. | Cierre ordenado al terminar la partida. |
| pthread\_mutex\_lock(...) | Bloquea el mutex. | Bloquea el mutex. |
| pthread\_mutex\_unlock(...) | Desbloquea el mutex. | Liberar acceso compartido tras actualizar/leer. |
| pthread\_mutex\_destroy(...) | Destruye el mutex. | Limpieza al salir. |
| usleep(microseg) | Suspende el hilo en microsegundos. | Control de velocidad y polling sin bloquear el juego. |
| rand()/srand() | Aleatorios e inicialización de semilla. | Posicionar comida/trampas en celdas libres. |
| time(nullptr) | Fuente de tiempo para la semilla. | srand(time(nullptr)) al iniciar. |

* **Calendarización:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Actividad | Responsable | Tiempo estimado | Fecha de inicio | Fecha de finalización |
| Diseño del menú de inicio funcional | Adriana Martínez / Mishell Ciprian | 4 h | 8 de sept de 2025 | 8 de sept de 2025 |
| Diseño de pantalla de instrucciones | Pablo Andrés Cabrera / Belén Monterroso | 4 h | 8 de sept de 2025 | 9 de sept de 2025 |
| Representación gráfica de jugadores y objetos en ASCII (serpiente, comida, trampas, tablero) | **Todos** | 8 h | 9 de sept de 2025 | 11 de sept de 2025 |
| Diseño y visualización de tabla de puntajes destacados | Adriana Martínez / Mishell Ciprian | 3 h | 10 de sept de 2025 | 10 de sept de 2025 |
| Integración y coherencia del entorno en consola (alineación, colores, zonas) | **Todos** | 5 h | 10 de sept de 2025 | 11 de sept de 2025 |
| Preparación de demostración funcional del entorno gráfico (pruebas de navegación y control) | **Todos** | 2 h | 11 de sept de 2025 | 11 de sept de 2025 |
| Elaboración del informe escrito (portada, resumen, diseño visual, pantallas, justificación) | **Todos** | 5 h | 9 de sept de 2025 | 12 de sept de 2025 |
| Revisión final del documento y calendarización | Pablo Andrés Cabrera / Belén Monterroso | 1 h | 12 de sept de 2025 | 12 de sept de 2025 |

* **Bibliografía:**
* Robbins, A., & Robbins, S. (1996). Programming with POSIX Threads. Addison-Wesley. <https://dl.acm.org/doi/10.5555/528759>
* Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley. https://www.stroustrup.com/4th.html
* Dougherty, D., & Robbins, A. (1996). UNIX Text Processing with ncurses. O’Reilly Media. <https://www.oreilly.com/library/view/unix-text-processing/>
* Project GNU. (2023). GNU Ncurses Library (Version 6.4) – Official Manual. GNU Project. <https://invisible-island.net/ncurses/>
* The Open Group. (2018). POSIX Threads (pthreads) Programming Guide. The Open Group Base Specifications Issue 7. <https://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html>
* Sutter, H., & Alexandrescu, A. (2004). C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley. <https://www.informit.com/store/c-plus-plus-coding-standards-101-rules-guidelines-and-9780321113580>
* Welsh, M., Dalheimer, M. K., & Kaufmann, L. (2000). Linux Programming Unleashed. Sams Publishing. <https://dl.acm.org/doi/book/10.5555/518684>