Antecedentes de la propuesta

Desde hace muchos años, los juegos de mesa han sido una fuente de entretenimiento accesible y estratégica, siendo *Batalla Naval* uno de los títulos más reconocidos por su simplicidad en las reglas y su profundidad táctica. A raíz de su popularidad, múltiples adaptaciones digitales han surgido en diversas plataformas, sin embargo, muchas de estas versiones se han limitado a replicar la experiencia básica del juego, sin aportar innovaciones reales a la mecánica ni al diseño de la interacción entre los jugadores.

En este contexto, nuestro equipo propone el desarrollo de una versión renovada de *Batalla Naval* que no solo mantenga el espíritu clásico del juego, sino que también incorpore elementos modernos que aumenten su complejidad estratégica. Entre estas innovaciones destacan el uso de **cartas de poder con efectos especiales**, la **posibilidad de mover las unidades (barcos)** dentro del tablero, y la inclusión de **decisiones tácticas adicionales durante el turno del jugador**.

Estas características permiten enriquecer la experiencia del usuario, dando lugar a partidas más dinámicas, impredecibles y desafiantes, donde la planificación y la adaptación constante son claves. A diferencia de implementaciones como *Fleet Battle* o *Sea Battle 2*, que conservan una estructura estática y repetitiva, nuestra propuesta apuesta por un enfoque más activo y personalizado de juego por turnos, incluyendo un sistema de turnos automatizado, validación de condiciones y estados, y generación de retroalimentación inmediata al usuario.

Además del objetivo lúdico, este proyecto busca consolidar los conocimientos adquiridos a lo largo del curso de **Programación Estructurada**, promoviendo el desarrollo de un sistema modular escrito en lenguaje C. Se implementarán técnicas esenciales como el uso de **estructuras de datos personalizadas**, **control de flujo**, **uso de punteros**, **memoria dinámica**, y la **manipulación de matrices y archivos**, elementos fundamentales para fortalecer la lógica algorítmica y las competencias técnicas de cada integrante.

Este enfoque permitirá que el videojuego no solo sea funcional, sino también escalable, claro y mantenible. Todo el desarrollo se realizará bajo estándares definidos de codificación, empleando prácticas de trabajo colaborativo y control de versiones para asegurar la correcta evolución del sistema a lo largo de sus fases de diseño e implementación.

Descripción del producto software

El producto que se propone desarrollar es una aplicación de consola, programada en lenguaje C, que representa una versión extendida, estratégica y dinámica del clásico juego “Batalla Naval”. Este software está diseñado para partidas multijugador-locales por turnos, en donde dos jugadores competirán en un entorno textual simulando una batalla marítima táctica con nuevas mecánicas.

Cada jugador contará con su propio tablero para colocar su flota y podrá interactuar con el sistema mediante un conjunto de **cartas de poder** que determinan las acciones posibles en cada turno. Estas cartas se asignan aleatoriamente al inicio del turno y permiten ejecutar distintos tipos de acciones: atacar coordenadas del tablero enemigo, mover uno de sus barcos o activar efectos especiales como escudos, radares o ataques dobles.

La lógica del juego estará completamente condicionada por el **sistema de cartas**, lo cual garantiza partidas variadas, estratégicas e impredecibles. Cada jugador tendrá que adaptarse a la carta que reciba en su turno, eligiendo la mejor forma de aprovechar su efecto dentro del contexto de la partida.

El juego continuará en rondas alternadas hasta que un jugador logre destruir completamente la flota del oponente. Durante toda la experiencia, el sistema ofrecerá una **interfaz clara y amigable dentro del entorno de consola**, con menús organizados, retroalimentación inmediata y validaciones automáticas de cada acción.

Técnicamente, el sistema se desarrollará de manera modular, utilizando estructuras de datos para representar los barcos, tableros, cartas y jugadores. Se integrará el uso de matrices, punteros y memoria dinámica para la gestión de estados, y se adoptarán buenas prácticas de programación estructurada como parte del enfoque didáctico del proyecto.

Este software no solo tendrá un propósito lúdico, sino también académico, funcionando como un ejercicio integrador que aplica los conocimientos adquiridos durante el curso de Programación Estructurada, consolidando habilidades clave para el desarrollo de software profesional.

Objetivo general y específicos del sistema

Objetivo general

Desarrollar una aplicación de consola en lenguaje C que permita a dos jugadores competir en una partida estratégica del juego “Batalla Naval”, incorporando nuevas mecánicas como el uso de cartas de poder, movimiento dinámico de unidades, detección automática de condiciones de victoria y una estructura modular clara y funcional, como ejercicio integrador de los conocimientos adquiridos en el curso de Programación Estructurada.

Código	Objetivo específico
OE1	Diseñar e implementar estructuras de datos que representen con claridad a los barcos, los tableros de juego, el sistema de cartas de poder y los jugadores.
OE2	Programar una lógica de turnos automatizada, que controle de manera eficiente el flujo de juego, permitiendo alternancia, validación de acciones y retroalimentación visual al jugador.
OE3	Incorporar un sistema de movimiento de unidades, que permita a los jugadores modificar la posición de sus barcos durante la partida como una estrategia adicional.
OE4	Desarrollar una mecánica de cartas de poder, con efectos variables que afecten al desarrollo del juego y se gestionen de forma estructurada y justa.
OE5	Gestionar las condiciones de victoria o derrota, detectando de manera automática el estado final del juego y mostrando mensajes adecuados al jugador.
OE6	Aplicar buenas prácticas de codificación, utilizando funciones bien definidas, separación por módulos, uso correcto de punteros, matrices y estructuras.
OE7	Fomentar el trabajo colaborativo y documentado, mediante bitácoras, control de versiones con GitHub, y registro de avances individuales y grupales.
OE8	Lograr que la interfaz de texto sea clara, accesible y funcional, facilitando al usuario la interacción con el juego y comprendiendo el estado de la partida en todo momento.

Definición de Requerimientos del Sistema

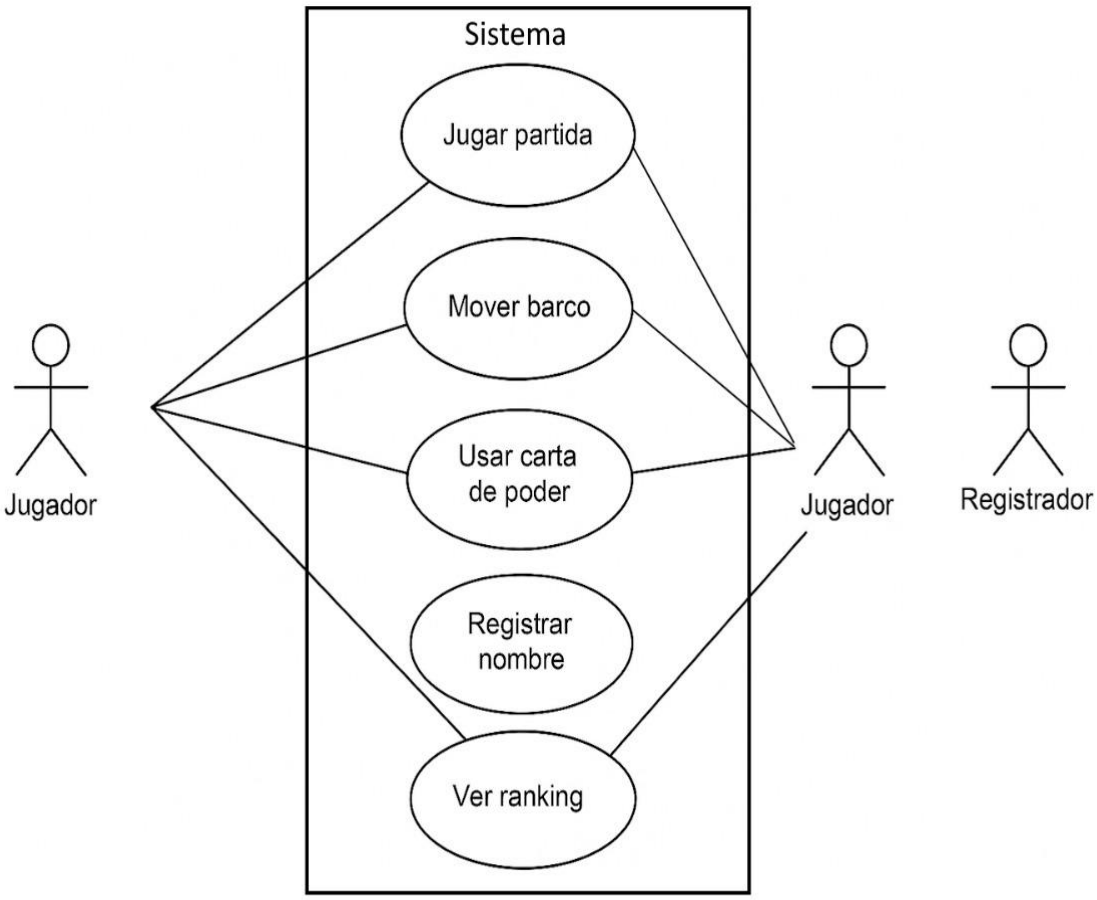
Requerimientos Funcionales

Código	Descripción
BNS-F01	El sistema permitirá registrar a dos jugadores solicitando un nombre identificador para cada uno.
BNS-F02	El sistema mostrará un menú principal con las opciones: iniciar partida, leer instrucciones y salir del juego.
BNS-F03	El sistema asignará de forma aleatoria una carta de poder a cada jugador al inicio de su turno, determinando la acción disponible.
BNS-F04	El sistema permitirá ejecutar un ataque a coordenadas del tablero enemigo solo si la carta recibida lo permite.
BNS-F05	El sistema permitirá mover un barco de posición únicamente si la carta activa del turno habilita esta acción.
BNS-F06	El sistema activará automáticamente efectos especiales (como radar, escudo o ataque doble) cuando la carta correspondiente esté activa.
BNS-F07	El sistema dará un reporte de estado de cada barco, incluyendo impactos, hundimientos y eliminación del tablero al jugador.
BNS-F08	El sistema alternará los turnos entre jugadores, mostrando con claridad cuál es el jugador activo en cada ronda.
BNS-F09	El sistema evaluará condiciones de victoria, finalizando la partida cuando uno de los jugadores pierda toda su flota.
BNS-F10	El sistema mostrará mensajes de fin de juego e invitará al jugador a salir o reiniciar la partida.

Requerimientos No Funcionales

Código	Descripción
BNS-NF01	El sistema debe ejecutarse completamente en entorno de consola, sin requerir conexión a internet.
BNS-NF02	El sistema debe ofrecer una interfaz textual clara y legible, con menús visibles y representaciones comprensibles del tablero.
BNS-NF03	El tiempo de respuesta entre las acciones del usuario debe ser inferior a 2 segundos.
BNS-NF04	El sistema debe estar modularizado en archivos independientes, cada uno asociado a un componente del juego.
BNS-NF05	El sistema debe utilizar estructuras, matrices y punteros para representar barcos y tableros.
BNS-NF06	El sistema usará Raylib y fuentes personalizadas para mejorar la experiencia visual.
BNS-NF07	La lógica del sistema debe estar separada de la visualización textual, facilitando mantenimiento y escalabilidad.
BNS-NF08	El sistema debe ser portable y compatible con compiladores compatibles con Raylib.
BNS-NF09	El sistema debe notificar al usuario sobre entradas erróneas y ofrecer retroalimentación textual inmediata.

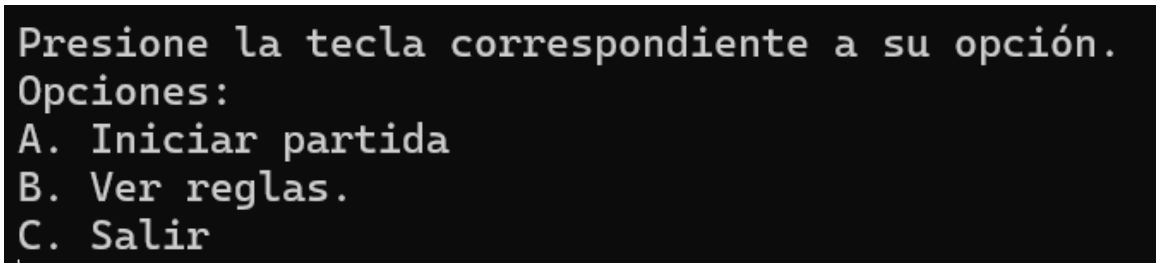
Diagrama de Casos de Uso



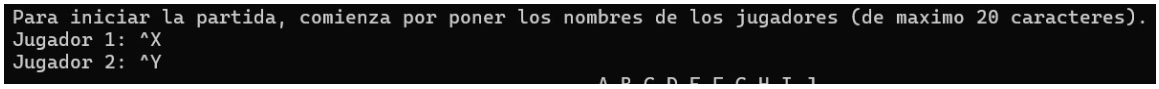
Interfaces

-Pantalla de inicio

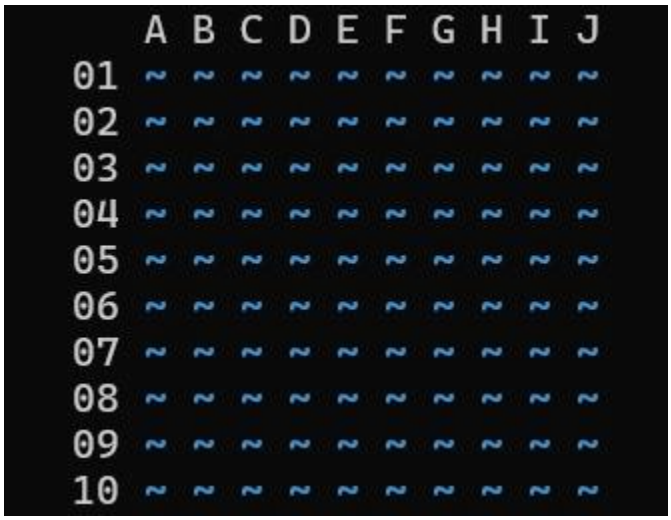
-Menú principal



-Jugadores



-Tabla



-Sistema de coordenadas

```
Coordenadas iniciales del barco (numero de fila y letra de columna, separadas por un espacio)
2 B
Coordenadas finales del barco (mismo formato).
3 B
```

```
Barco colocado exitosamente en (2, B)
   A B C D E F G H I J
01 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
02 ~ x ~ ~ ~ ~ ~ ~ ~ ~
03 ~ x ~ ~ ~ ~ ~ ~ ~ ~
04 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
05 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
06 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
07 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
08 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
09 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
10 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

Casos de Uso Detallados

CU-BNS01 - Registrar jugadores

- **Actor principal:** Jugador 1 / Jugador 2
- **Descripción:** El jugador ingresa su nombre al iniciar la partida para identificar su tablero.
- **Precondición:** Se debe haber seleccionado la opción “Iniciar partida” en el menú principal.
- **Flujo principal:**
    1. El sistema solicita el nombre para el Jugador 1.
    2. El sistema solicita el nombre para el Jugador 2.
    3. Se validan los nombres (no vacíos, no repetidos).
    4. Se guarda la información temporalmente.
- **Postcondición:** Los nombres quedan registrados para el resto de la sesión.

CU-BNS02 - Colocar barcos en el tablero

- **Actor principal:** Jugador
- **Descripción:** El jugador coloca sus barcos uno por uno en el tablero siguiendo las reglas del juego.
- **Precondición:** Los jugadores deben haber sido registrados.
- **Flujo principal:**
    1. El sistema solicita la coordenada inicial y orientación del barco.
    2. Se valida que el barco no se salga de los límites ni se solape con otro.
    3. El barco se coloca en el tablero.
    4. Se repite hasta completar todos los barcos.
- **Postcondición:** El tablero inicial del jugador queda configurado.

CU-BNS03 - Ejecutar ataque

- **Actor principal:** Jugador
- **Descripción:** El jugador elige una coordenada del tablero enemigo para intentar impactar un barco.
- **Precondición:** El jugador debe tener el turno activo.
- **Flujo principal:**
    1. El sistema solicita una coordenada (ej. B5).
    2. Se valida si hay barco enemigo en esa posición.
    3. Se informa si fue acierto o fallo.
    4. Se actualiza el tablero.
- **Postcondición:** El estado del tablero enemigo se actualiza y se cambia el turno.

CU-BNS04 - Usar carta de poder

- **Actor principal:** Jugador
- **Descripción:** El jugador activa una carta especial disponible.
- **Precondición:** El jugador debe tener cartas disponibles y estar en su turno.
- **Flujo principal:**
    1. El sistema muestra las cartas disponibles.
    2. El jugador selecciona una carta.
    3. Se valida si su uso es permitido en ese momento.
    4. Se aplica el efecto de la carta.
    5. Se actualiza el estado del juego.
- **Postcondición:** La carta es marcada como utilizada y su efecto queda reflejado.

CU-BNS05 - Mover barco

- **Actor principal:** Jugador
- **Descripción:** El jugador mueve un barco una casilla en alguna dirección válida.
- **Precondición:** Debe haber barcos vivos y el movimiento debe estar habilitado.
- **Flujo principal:**
    1. El sistema muestra los barcos del jugador.
    2. El jugador selecciona qué barco mover.
    3. Se ingresa la dirección del movimiento.
    4. Se valida que no salga del tablero ni colisione.
    5. Se actualiza la posición.
- **Postcondición:** El barco cambia de ubicación y el tablero se actualiza.

CU-BNS06 - Detectar condición de victoria

- **Actor principal:** Sistema
- **Descripción:** El sistema detecta si un jugador ha perdido todos sus barcos.
- **Precondición:** Se ha realizado una acción de ataque o finalizado un turno.
- **Flujo principal:**
    1. El sistema revisa si quedan barcos vivos en el tablero del jugador afectado.
    2. Si no hay ninguno, se declara al oponente como ganador.
    3. Se muestra mensaje final y opciones de salida o reinicio.
- **Postcondición:** El juego termina oficialmente.

Matriz de Requerimientos y Objetivos

La siguiente tabla presenta los requerimientos funcionales y no funcionales del sistema, su versión correspondiente, la fecha de validación y su relación directa con los objetivos específicos del proyecto.

Identificación	Descripción del requisito	Versión	Objetivo relacionado	Última fecha registrada
BNS-F01	El sistema permitirá registrar a dos jugadores solicitando un nombre identificador para cada uno.	1.0.0	OE8	12/05/2025
BNS-F02	El sistema mostrará un menú principal con las opciones: iniciar partida, leer instrucciones y salir del juego.	1.0.0	OE8	12/05/2025
BNS-F03	El sistema asignará de forma aleatoria una carta de poder a cada jugador al inicio de su turno.	1.0.0	OE4	12/05/2025
BNS-F04	El sistema permitirá ejecutar un ataque a coordenadas del tablero enemigo solo si la carta recibida lo permite.	1.0.0	OE5	12/05/2025
BNS-F05	El sistema permitirá mover un barco de posición solo si la carta activa del turno lo permite.	1.0.0	OE3	12/05/2025
BNS-F06	El sistema activará efectos especiales cuando la carta correspondiente esté activa.	1.0.0	OE4	12/05/2025
BNS-F07	El sistema gestionará el estado de cada barco, incluyendo impactos, hundimientos y eliminación del tablero.	1.0.0	OE1	12/05/2025
BNS-F08	El sistema alternará los turnos entre jugadores, mostrando cuál es el jugador activo en cada ronda.	1.0.0	OE2	12/05/2025
BNS-F09	El sistema evaluará condiciones de victoria, finalizando la partida cuando un jugador pierda toda su flota.	1.0.0	OE5	12/05/2025
BNS-F10	El sistema mostrará mensajes de fin de juego e invitará al jugador a salir o reiniciar la partida.	1.0.0	OE8	12/05/2025
BNS-NF01	El sistema debe ejecutarse completamente en entorno de	1.0.0	OE6	12/05/2025

	consola, sin requerir conexión a internet.			
BNS-NF02	La interfaz textual debe ser clara y legible, con menús visibles y representaciones comprensibles.	1.0.0	OE8	12/05/2025
BNS-NF03	El tiempo de respuesta entre acciones debe ser inferior a 2 segundos.	1.0.0	OE6	12/05/2025
BNS-NF04	El sistema debe estar modularizado en archivos independientes.	1.0.0	OE6	12/05/2025
BNS-NF05	Uso de estructuras, matrices y punteros para barcos y tableros.	1.0.0	OE1	12/05/2025
BNS-NF06	El sistema usará Raylib y fuentes personalizadas para mejorar la experiencia visual.	1.0.0	OE8	12/05/2025
BNS-NF07	Separación entre lógica y visualización para facilitar mantenimiento.	1.0.0	OE6	12/05/2025
BNS-NF08	El sistema debe ser portable y compatible con compiladores compatibles con Raylib.	1.0.0	OE6	12/05/2025
BNS-NF09	Retroalimentación inmediata ante entradas erróneas del usuario.	1.0.0	OE8	12/05/2025

Informe de Avance del Código respecto a los Requerimientos

Hasta la fecha de corte, el desarrollo del sistema ha avanzado conforme a lo establecido en los requerimientos funcionales y no funcionales. A continuación, se detalla un resumen de su cumplimiento:

Se han implementado los módulos de registro de jugadores, colocación de barcos con validación, y estructuras básicas del sistema

```c
struct ship
{
    int size;
    int startShip[2];
    int endShip[2];
    int *ship_status; // Pointer for dynamic allocation of status array
    char orientation; // 'H' for horizontal, 'V' for vertical, 'U' for undefined
    char direction; // 'E' for east, 'W' for west, 'N' for north, 'S' for south, 'U' for undefined
};

struct player
{
    char name[MAX_NAME_LENGTH];
    struct ship ships[NUM_SHIPS]; // Array of ships for the player
    int placed_ships;
    int sunked_enemy_ships;
    int sunked_ships;

};
```

El sistema gestiona de forma correcta el flujo de turnos, la alternancia entre jugadores, y la asignación de nombres.

```c
// funcion para iniciar la partida. Esta funcion sera como un "main secundario" para ejecutar todas las subrutinas necesarias para llevar a cabo una partida.
void partida(){
    struct player player1, player2;
    inicializar_flota(&player1);
    inicializar_flota(&player2);

        limpiar(); // se limpia el menu de opciones inicial despues de haber entrado en la funcion partida, para que no estorbe.

        // Variables para almacenar los nombres de los jugadores.
        char NombreJugador1[MAX_NAME_LENGTH], NombreJugador2[MAX_NAME_LENGTH];

        // Las siguientes lineas solicitan y capturan los nombres de los jugadores.
        printf("Para iniciar la partida, comienza por poner los nombres de los jugadores (de maximo 20 caracteres).\n");
        getchar(); // Limpiar el buffer de entrada si hay caracteres sobrantes.

    printf("Jugador 1: ");
        fgets(NombreJugador1, sizeof(NombreJugador1), stdin);
    // Elimina el salto de línea al final del nombre.
    NombreJugador1[strcspn(NombreJugador1, "\n")] = '\0';

        printf("Jugador 2: ");
        fgets(NombreJugador2, sizeof(NombreJugador2), stdin);
    // Elimina el salto de línea al final del nombre.
        NombreJugador2[strcspn(NombreJugador2, "\n")] = '\0';

        int MatrizJug1[BOARD_SIZE][BOARD_SIZE] = {0}; // Matriz para el jugador 1.
    int MatrizJug2[BOARD_SIZE][BOARD_SIZE] = {0}; // Matriz para el jugador 2.
```

El sistema de cartas de poder está en fase de diseño, por lo cual los requerimientos relacionados con acciones dependientes de cartas (ataques, defensas, movimientos) aún no han sido implementados.

Se han respetado los estándares de codificación, con funciones bien comentadas y un esquema modular organizado en archivos '.c' y '.h'.

Los requerimientos no funcionales han sido tomados como base para toda la arquitectura del sistema, incluyendo uso de estructuras, claridad en interfaz textual, y portabilidad.

Se ha implementado una lógica de validación que impide errores de entrada del usuario, tales como coordenadas fuera de rango o solapamiento de barcos

```
// Validar que las coordenadas estén dentro del rango permitido.
if (filaInicio < 0 || filaInicio >= BOARD_SIZE || filaFin < 0 || filaFin >= BOARD_SIZE ||
    colCharInicio < 0 || colCharInicio >= BOARD_SIZE || colCharFin < 0 || colCharFin >= BOARD_SIZE) {
    printf("Coordenadas fuera de rango. Asegurese de establecerlas entre 1 y 10 para filas y A-J para columnas. Intente de nuevo.\n");
    return;
}

// Validar que no haya solapamiento con otros barcos.
int solapamiento = 0;
for (int i = filaInicio; i <= filaFin; i++) {
    for (int j = colCharInicio; j <= colCharFin; j++) {
        if (matriz[i][j] != 0) {
            solapamiento = 1;
            continue;
        }
    }
    if (solapamiento) continue;
```

Se ha hecho uso de herramientas de trabajo colaborativo como GitHub, manteniendo control de versiones y registro de participación individual en una bitácora semanal.

El código está disponible en el repositorio público del equipo: https://github.com/Ozia112/proyecto_prog_estructurada

Este informe se elabora como evidencia del avance parcial del sistema respecto al cumplimiento de los requerimientos, sirviendo como registro técnico y académico de la evolución del proyecto.

Estándares de codificación

Para el desarrollo del sistema "Batalla Naval con Cartas", se implementó una estructura modular y se adoptó un conjunto de buenas prácticas basadas en el estándar **ANSI C**, con adaptaciones propias orientadas a la claridad, portabilidad y eficiencia.

A continuación, se describen los principales criterios empleados:

Nombrado de variables y estructuras

- Se usa nomenclatura en inglés y descriptiva: `startShip`, `ship_status`, `ship_body`, `MatrizJug1`.
- Las estructuras `ship` y `player` están bien definidas en `battleship_librarie.c`, con campos inicializados y comentados adecuadamente.
- Las constantes usan mayúsculas con guiones bajos (`BOARD_SIZE`, `MAX_NAME_LENGTH`), como dicta el estilo clásico de macros en C.

Uso de macros

- Se aplican `#define` para valores constantes como colores (`WaterColor`, `ShipColor`), tamaños de tablero (`BOARD_SIZE`), y caracteres de interfaz (`ship_tip`, `ship_body`).
- Esto permite modificar aspectos del sistema sin alterar la lógica, facilitando el mantenimiento.

Organización modular del código

- El código se divide en tres archivos principales:
    - `battleship.c`: contiene el `main()` y el menú principal.
    - `battleship_librarie.c`: contiene las funciones centrales del juego.
    - `battleship_librarie.h`: define los prototipos y evita redefiniciones con `#ifndef`.
- Cada módulo gestiona una parte del sistema, respetando el principio de **responsabilidad única**.

Funciones breves y comentadas

- Las funciones como `imprimirTablero`, `ponerBarcos`, `partida`, `limpiar`, están bien separadas y cumplen un propósito claro.
- Se incluyen comentarios tipo:
- ```
  // Entrada: coordenadas iniciales
  // Proceso: validación y asignación
  // Salida: matriz modificada
  ```

Estructuras de control claras

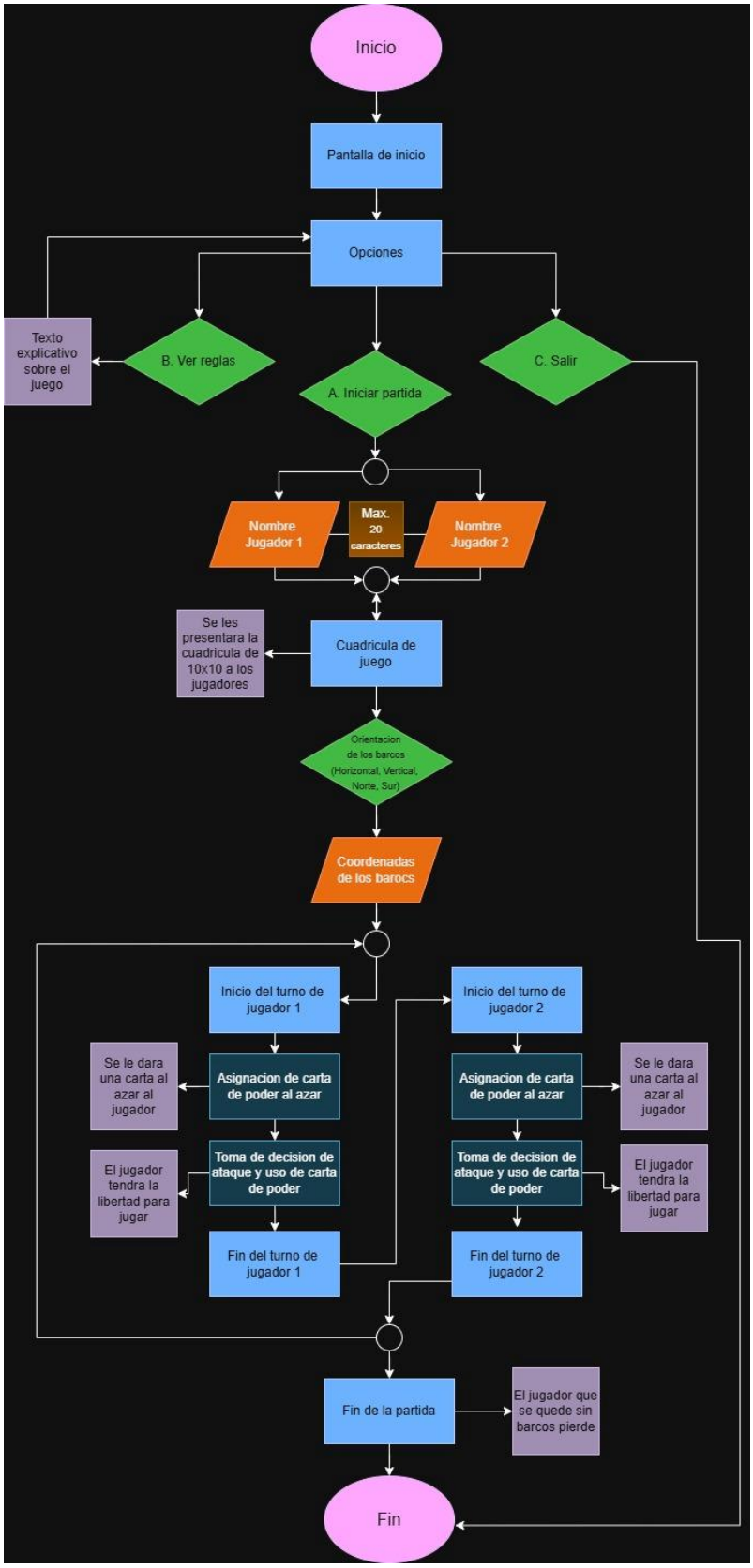
- Se evita el uso excesivo de anidamientos o condiciones ambiguas.
- Se usa `switch` en el menú principal (`main`) y `while` con condiciones claras para repetir entradas inválidas.

Bibliotecas utilizadas

- Se utilizan:
  - `stdio.h`, `stdlib.h` para entrada/salida y manejo de memoria.
  - `windows.h` para control de color y comandos de consola (`SetConsoleTextAttribute`, `Sleep`).
  - `ctype.h` y `string.h` para manejo de caracteres y cadenas.
- La librería `Raylib` será integrada en futuras versiones para representar visualmente la partida, aunque actualmente todo funciona en consola.
  ```

Modularidad

Diagrama de bloques



Proceso de Desarrollo y Organización del Equipo

El desarrollo del sistema 'Batalla Naval' se planificó de manera modular, dividiendo las tareas según las áreas funcionales del juego. Cada integrante asumió la responsabilidad de uno o más componentes, de acuerdo con sus habilidades técnicas y la distribución acordada desde el inicio del proyecto.

División de módulos y responsabilidades

Módulo	Funcionalidad Principal	Responsable
Menú e inicio de partida	Captura de nombres, inicio del juego	Nicolás Canul
Tablero y colocación	Impresión de matriz, validación de coordenadas, colocación de barcos	Alejandro Flores
Turnos y flujo del juego	Alternancia de jugadores, estructura inicial de partida	Jesús León
Documentación y estandarización	Comentarios, estilo de codificación, revisión de estructuras	Isaac Ortiz
Funciones generales y limpieza	Funciones de impresión, limpieza de consola, interacción básica	Todos

Herramientas de trabajo colaborativo

- Repositorio en GitHub para control de versiones y colaboración en línea. Cada integrante trabajó en su propio módulo desde ramas individuales.
- Reuniones semanales para definir avances, resolver problemas técnicos y revisar la integración de código.
- Uso de bitácora grupal para llevar control de actividades semanales y participación individual.

Bitácora Semanal de Actividades

Semana	Nicolás Canul	Alejandro Flores	Jesús León	Isaac Ortiz
1	Diseño y estructura del menú	Diseño base del tablero	Estructura de turnos y juego	Organización de archivos y estructuras
2	Captura de nombres y entrada de datos	Colocación de barcos y validación de solapamiento	Inicio de turnos y verificación básica	Modularización del código y limpieza
3	Pruebas de flujo del menú	Revisión de impresión de tablero	Enlace entre menú y tablero	Comentarios y revisión del estándar
4	Revisión final y documentación	Ajustes de tablero	Apoyo en integración	Documentación del estándar de codificación

Evaluación del Cumplimiento de Objetivos y Participación del Equipo

El equipo logró cumplir parcialmente los objetivos establecidos al inicio del proyecto. A continuación, se presenta una evaluación del estado de cumplimiento de cada uno de los objetivos específicos del sistema:

Objetivo Específico	Estado
Diseñar estructuras de datos para el tablero y los barcos	Cumplido
Implementar la colocación y validación de barcos	Cumplido
Programar la lógica básica de turnos	En proceso
Desarrollar el sistema de cartas de poder	No iniciado
Aplicar buenas prácticas de codificación	Cumplido
Documentar y organizar el código en módulos	Cumplido
Trabajar colaborativamente usando GitHub y bitácoras	Cumplido
Desarrollar interfaz clara y funcional en consola	Cumplido

En cuanto a la participación del equipo, todos los integrantes cumplieron con su parte asignada, mostrando compromiso y responsabilidad en el avance del sistema. Las tareas se dividieron de forma equitativa y se realizaron reuniones periódicas para discutir los avances, resolver dudas y ajustar funcionalidades. Se utilizó GitHub para el control de versiones y la colaboración técnica, y se mantuvo una bitácora para documentar las actividades semanales. El equipo trabajó con una actitud colaborativa y enfocada a cumplir los entregables del proyecto.

Evaluación de la Participación Individual

Integrante	Rol Principal	% de Contribución Estimada
Nicolás Canul	Menú, entrada de datos	25%
Alejandro Flores	Tablero, colocación de barcos	30%
Jesús León	Turnos y flujo del juego	20%
Isaac Ortiz	Documentación, estructura modular	25%

Los porcentajes reflejan la participación técnica directa en el código y la documentación. Las decisiones se tomaron en conjunto y todos participaron activamente en las reuniones semanales, en la resolución de dudas técnicas y en el control de versiones del repositorio compartido en GitHub.

Convenciones de Nomenclatura en el Proyecto

Para asegurar claridad y organización en la documentación del sistema, se ha seguido una nomenclatura sistemática para los distintos elementos clave del desarrollo:

- **BNS-F**: Hace referencia a los **Requerimientos Funcionales** del sistema.
  - **Ejemplo**: BNS-F04 indica el cuarto requerimiento funcional documentado.
- **BNS-NF**: Representa los **Requerimientos No Funcionales**.
  - **Ejemplo**: BNS-NF06 corresponde al sexto requerimiento no funcional relacionado con aspectos técnicos como rendimiento o portabilidad.
- **CU-BNS**: Indica un **Caso de Uso** asociado al sistema **Batalla Naval** (abreviado BNS).
  - **Ejemplo**: CU-BNS02 - Colocar barcos en el tablero representa el segundo caso de uso identificado en el desarrollo.

Esta nomenclatura estandarizada permite un rastreo preciso entre objetivos, requerimientos y funcionalidades implementadas, además de facilitar futuras revisiones y mantenimientos del software.

Repositorio del Proyecto

Para facilitar el seguimiento del desarrollo y la revisión del código fuente, el proyecto se ha trabajado de forma colaborativa en un repositorio en GitHub. En este repositorio se encuentran los archivos del sistema, documentación del código, bitácoras y registros de versiones, con aportaciones individuales e integraciones grupales.

Enlace al repositorio: https://github.com/Ozia112/proyecto_prog_estructurada