
FP-UNA - CONSTRUYENDO EL FUTURO

Ingeniería en informática

Lenguajes de Programación I - TQ

# Juego interactivo que propone determinar cuadrados a partir de sus vértices.

Alumna: Maria Cielito Melgarejo Baez

C.I: 5574506

**Profesor:** Dr. Diego Pedro Pinto Roa

AÑO 2023

# INDICE

# DESCRIPCION GENERAL DEL TRABAJO

El programa diseñado e implementado en lenguaje C consiste en un juego en el que los jugadores deben evitar formar cuadrados mediante la intersección de sus vértices. Además del juego en sí, el programa gestiona usuarios con estadísticas, puntajes de jugadores y ofrece la opción de cambiar el tamaño del tablero.

Este programa se juega en un tablero que puede tener un tamaño máximo de 10x10 casillas. Los jugadores se turnan para ubicar sus fichas en las casillas libres del tablero, representando los vértices de los cuadrados. Las fichas del primer jugador tienen el valor numérico 1 y las del segundo jugador tienen el valor 2. Se alterna el orden entre el jugador humano y el algoritmo de la computadora.

El objetivo del juego es evitar que el oponente forme un cuadrado marcando cuatro vértices en el tablero. El ganador es el jugador que obliga a su oponente a colocar cuatro fichas cuyos vértices se pueden unir para formar un cuadrado. Si el tablero se llena sin que ninguno de los jugadores logre formar un cuadrado, se considera un empate.

El programa utiliza archivos para almacenar la información necesaria. Utiliza un archivo "ranking.txt" para almacenar las puntuaciones de los jugadores. Además, cada jugador tiene dos archivos asociados a su nombre: "configuracion.txt", que almacena el tamaño del tablero, y "resultados.txt", que contiene los resultados de las partidas. El programa utiliza archivos para almacenar y gestionar la información necesaria.

El programa presenta un menú interactivo donde el jugador puede realizar diversas acciones:

1. Visualizar configuración del tablero: muestra la información sobre el tamaño del tablero leyendo el archivo correspondiente al jugador.
2. Configurar parámetros del tablero: permite modificar el tamaño del tablero almacenado en el archivo de configuración del jugador.
3. Ver estadísticas: muestra las estadísticas del jugador, incluyendo el porcentaje de partidas ganadas, perdidas y empatadas, la cantidad total de jugadas y el puntaje de los 5 mejores jugadores. El puntaje de cada jugador se calcula como (número de partidas ganadas - número de partidas perdidas) / partidas totales.
4. Jugar partida: inicia el juego, donde el jugador interactúa con el tablero y la computadora para colocar sus fichas. La computadora utiliza un algoritmo para realizar su jugada.
5. Ayuda: muestra el sistema de ayuda del juego.
6. Salir: permite salir del programa y finalizar el juego.

Durante el juego, el programa guarda la información actualizada en los archivos correspondientes para mantener las estadísticas y permitir el seguimiento de la última partida.

El programa puede presentar algunas limitaciones y dificultades, entre las cuales se incluyen:

- Tamaño máximo del tablero: El programa está diseñado para manejar tableros de hasta 10x10 casillas. Si se desea trabajar con tableros de tamaño mayor, sería necesario modificar el código para adaptarlo a estas dimensiones.

- Complejidad del algoritmo de la computadora: La inteligencia artificial implementada en el programa para que la computadora realice sus movimientos se basa en un algoritmo determinado. Dependiendo de la complejidad y sofisticación del algoritmo utilizado, la computadora podría tomar decisiones subóptimas en ciertas situaciones o no ofrecer un nivel de desafío adecuado para los jugadores más experimentados.

- Limitaciones en la interacción del jugador: El programa solicita al jugador las coordenadas de la casilla en la que desea colocar su ficha. Esto puede ser un proceso lento y propenso a errores si el jugador no ingresa las coordenadas correctamente. No posee una interfaz de usuario más intuitiva que facilite la interacción con el tablero.

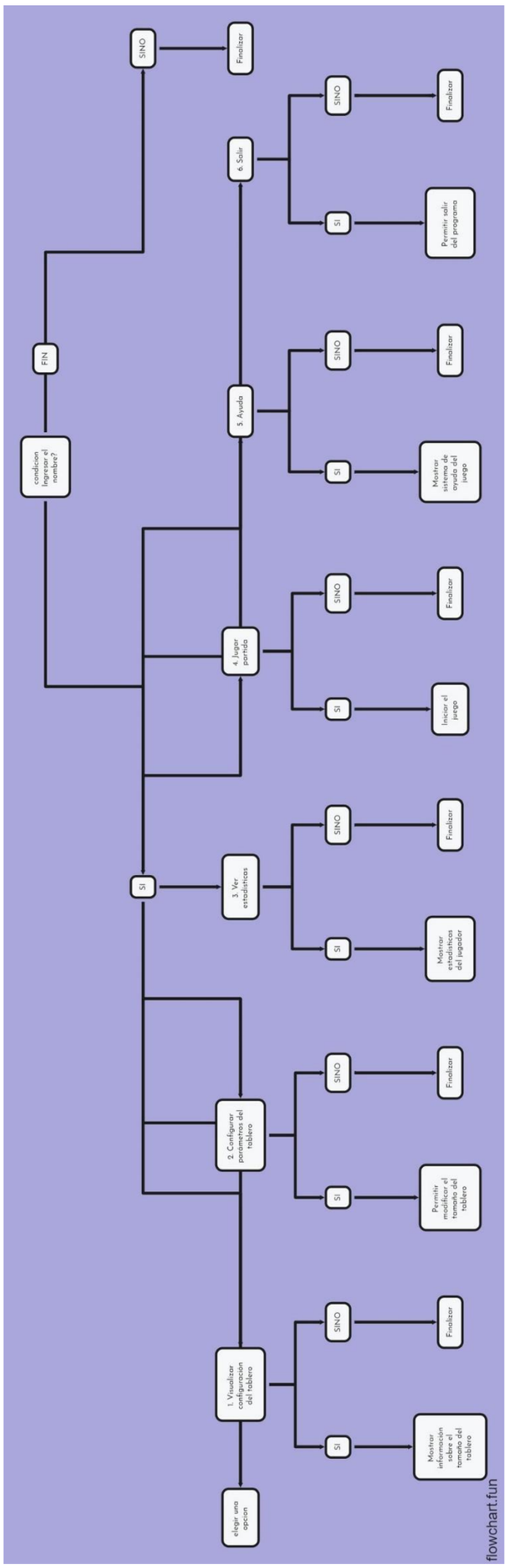
- Escalabilidad del programa: El programa está diseñado para manejar un número limitado de jugadores y partidas. En específico, cuenta con un límite de 20 jugadores. Si se desea ampliar el sistema para admitir un mayor número de jugadores o mantener un historial más extenso de partidas y estadísticas, puede requerir modificaciones en la estructura de datos y en el manejo de archivos.

- Validación de la entrada del usuario: El programa asume que el usuario ingresará datos válidos en cada interacción, como las coordenadas de las casillas. Sin embargo, no se realiza una validación exhaustiva de la entrada del usuario, lo que podría conducir a comportamientos inesperados o errores si el usuario proporciona datos incorrectos.

- Falta de interacción en tiempo real: El programa no permite la interacción en tiempo real entre los jugadores, ya que cada jugador realiza su jugada por turnos. Esto podría restarle emoción y dinamismo al juego, especialmente en partidas entre dos jugadores humanos.

- Estas limitaciones y dificultades pueden ser abordadas y mejoradas mediante la optimización del código, la implementación de validaciones adicionales, la mejora del algoritmo de inteligencia artificial y la adición de características adicionales para una mejor interacción con los jugadores.

# DIAGRAMA GENERAL



# EXPLICACIÓN DETALLADA DE MODULOS

## A. Principales definiciones de módulos y funciones presentes en el archivo "librerias.h".

1. Módulo: Entradas/Salidas
    - Funciones:
        - printf: Imprime texto en la salida estándar.
        - scanf: Lee la entrada estándar según un formato especificado.
2. Módulo: Librería estándar
    - Funciones:
        - stdlib.h:
            - malloc: Reserva un bloque de memoria dinámica.
            - free: Libera un bloque de memoria previamente reservado.
        - string.h:
            - strcpy: Copia una cadena de caracteres.
            - strlen: Obtiene la longitud de una cadena de caracteres.
3. Módulo: Booleanos
    - Tipo de dato:
        - stdbool.h:
            - bool: Tipo de dato booleano (verdadero o falso).
            - true: Valor verdadero.
            - false: Valor falso.
4. Constantes
    - JUGADOR: Valor constante para representar al jugador (1).
    - COMPUTADORA: Valor constante para representar a la computadora (2).
    - MAX_NICK_LENGTH: Longitud máxima permitida para el nombre del jugador.
    - MAX_RESULTADOS_FILENAME: Longitud máxima del nombre de archivo de resultados.
5. Estructuras de datos
    - Coordenada: Estructura que almacena las coordenadas de una posición en el tablero.
    - Nodo: Estructura que representa un nodo en una lista enlazada de coordenadas.
    - ListaCoordenadas: Estructura que representa una lista enlazada de coordenadas.
    - ResultadosJugador: Estructura que almacena los resultados de un jugador (partidas jugadas, ganadas, perdidas y empatadas).
    - Jugador: Estructura que representa un jugador con su nombre y puntaje.
6. Funciones prototipo
    - mostrarConfiguracion: Muestra la configuración del tablero del jugador.
    - configurarParametros: Permite modificar la configuración del tablero del jugador.
    - verEstadisticas: Muestra las estadísticas del jugador.
    - jugarPartida: Inicia una partida del juego.
    - mostrarAyuda: Muestra la ayuda del juego.
7. Funciones utilizadas durante el juego
    - crearTablero: Crea dinámicamente un tablero de juego con las dimensiones especificadas.
    - liberarTablero: Libera la memoria asignada para el tablero.

- verificarGanador: Verifica si un jugador ha ganado la partida, evitando las coordenadas en una lista especificada.
- tableroLleno: Verifica si el tablero de juego está completamente lleno.
- agregarCoordenada: Agrega una coordenada a una lista enlazada.
- mostrarCoordenadas: Muestra las coordenadas de una lista enlazada.
- numCoordenadas: Obtiene el número de coordenadas en una lista.
- verificarTablero: Verifica si una lista de coordenadas forma un cuadrado en el tablero.
- coordenadaExistente: Verifica si una coordenada ya existe en una lista.
- longitudLista: Obtiene la longitud de una lista enlazada.
- eliminarNodo: Elimina un nodo de una lista enlazada.
- seleccionarAleatorio: Selecciona una coordenada aleatoria de una lista enlazada.
- guardarJugador: Guarda la información de un jugador en un archivo.

## B. Principales definiciones de módulos y funciones presentes en el archivo "main.c".

1. Inclusión de librerías:
    - #include "Librerias.h": Incluye el archivo de cabecera "Librerias.h", que contiene las definiciones de módulos y funciones utilizadas en el programa.
2. Declaración de variables:
    - int filas;: Variable para almacenar el número de filas del tablero.
    - int columnas;: Variable para almacenar el número de columnas del tablero.
    - char nick[MAX_NICK_LENGTH];: Arreglo de caracteres para almacenar el nombre del jugador.
3. Función principal main():
    - Captura del nombre del jugador:
        - Se muestra el mensaje "Ingresa tu nick para comenzar el juego: " mediante printf().
        - Se lee el nombre del jugador utilizando scanf() y se guarda en la variable nick.
4. Creación del archivo de configuración:
    - Se declara una variable nombreFichero para almacenar el nombre del archivo de configuración.
    - Se utiliza strcpy() y strcat() para concatenar el nombre del jugador con "_configuracion.txt" y obtener el nombre del archivo de configuración.
    - Se intenta abrir el archivo en modo lectura ("r").
        - Si el archivo no existe (fichero == NULL), se crea el archivo en modo escritura ("w").
            - Si no se puede crear el archivo, se muestra el mensaje "Error al crear el fichero de configuración" y se retorna 1 para indicar un error.
            - Se inicializan las variables filas y columnas con valores predeterminados (10) y se escriben en el archivo utilizando fprintf().
            - Se cierra el archivo con fclose().
        - Si el archivo existe, se cierra directamente con fclose().
5. Creación del archivo de resultados:
    - Se declara una variable nombreFicheroResultados para almacenar el nombre del archivo de resultados.

- Se utiliza sprintf() para formatear el nombre del archivo de resultados con el nombre del jugador y "_resultados.txt".
- Se intenta abrir el archivo en modo lectura ("r").
    - Si el archivo no existe (ficheroResultados == NULL), se crea el archivo en modo escritura ("w").
        - Si no se puede crear el archivo, se muestra el mensaje "Error al crear el fichero de resultados" y se retorna 1 para indicar un error.
        - Se cierra el archivo con fclose().
    - Si el archivo existe, se cierra directamente con fclose().

6. Menú de opciones:
    - Se utiliza un bucle do-while para mostrar el menú y leer la opción seleccionada por el usuario.
    - Se muestra el menú con las siguientes opciones:
        1. Visualizar configuración del tablero.
        2. Configurar parámetros del tablero.
        3. Ver estadísticas del jugador.
        4. Jugar una partida.
        5. Mostrar ayuda del juego.
        6. Salir del programa.
    - Se lee la opción seleccionada por el usuario utilizando scanf() y se guarda en la variable opcion.
    - Se utiliza una estructura switch-case para ejecutar la función correspondiente según la opción seleccionada por el usuario.
        - Para las opciones del 1 al 5, se llama a las funciones correspondientes pasando el nombre del jugador como argumento.
        - Para la opción 6 (salir), el bucle se interrumpe y el programa finaliza.
7. Retorno de la función main():
    - Se retorna 0 para indicar una finalización exitosa del programa.

## C. Principales definiciones de módulos y funciones presentes en el archivo "funciones.c"

1. Función **mostrarConfiguracion(const char* nick)**:
    - Declara las variables locales:
        - **nombreFichero**: Arreglo de caracteres para almacenar el nombre del archivo de configuración.
        - **fichero**: Puntero de tipo **FILE** para manipular el archivo.
        - **filas** y **columnas**: Variables enteras para almacenar el tamaño del tablero.
    - Construcción del nombre del archivo de configuración utilizando **strcpy()** y **strcat()**.
    - Apertura del archivo en modo lectura (**"r"**) utilizando **fopen()**.
        - Si el archivo no se puede abrir (es **NULL**), muestra el mensaje "Error al abrir el fichero de configuración" y retorna.
    - Lectura del tamaño del tablero desde el archivo utilizando **fscanf()**.
    - Cierre del archivo con **fclose()**.
    - Muestra la configuración del tablero por pantalla utilizando **printf()**.
    - Imprime las filas y columnas del tablero.

2. Función **configurarParametros(const char* nick)**:
    - Declara las variables locales:
        - **nombreFichero**: Arreglo de caracteres para almacenar el nombre del archivo de configuración.
        - **fichero**: Puntero de tipo **FILE** para manipular el archivo.
        - **filas** y **columnas**: Variables enteras para almacenar el tamaño actual del tablero.
    - Construcción del nombre del archivo de configuración utilizando **strcpy()** y **strcat()**.
    - Apertura del archivo en modo lectura (**"r"**) utilizando **fopen()**.
        - Si el archivo no se puede abrir (es **NULL**), muestra el mensaje "Error al abrir el fichero de configuración" y retorna.
    - Lectura del tamaño actual del tablero desde el archivo utilizando **fscanf()**.
    - Cierre del archivo con **fclose()**.
    - Muestra la configuración actual del tablero por pantalla utilizando **printf()**.
    - Solicita al usuario el nuevo tamaño del tablero utilizando **printf()** y **scanf()**.
    - Valida el nuevo tamaño del tablero:
        - Si el número de filas o columnas es menor o igual a cero, o mayor que 10, muestra el mensaje de error y retorna.
    - Apertura del archivo en modo escritura (**"w"**) para actualizar la configuración utilizando **fopen()**.
        - Si el archivo no se puede abrir (es **NULL**), muestra el mensaje "Error al abrir el fichero de configuración" y retorna.
    - Escribe el nuevo tamaño del tablero en el archivo utilizando **fprintf()**.
    - Cierre del archivo con **fclose()**.
    - Muestra el mensaje de actualización exitosa y el nuevo tamaño del tablero por pantalla utilizando **printf()**.
3. La función verEstadisticas(const char* nick) realiza las siguientes operaciones:
    - Construye el nombre del archivo de resultados utilizando la variable nick y la función sprintf().
    - Abre el archivo de resultados en modo lectura ("r") utilizando fopen().
    - Si el archivo no se puede abrir (es NULL), muestra el mensaje "Error al abrir el fichero de resultados" y retorna.
    - Declara una variable resultados del tipo ResultadosJugador para almacenar los resultados del jugador.
    - Lee los valores de las partidas jugadas, ganadas, perdidas y empatadas desde el archivo utilizando fscanf().
    - Cierra el archivo de resultados con fclose().
    - Calcula los porcentajes de partidas ganadas, perdidas y empatadas.
    - Muestra las estadísticas del jugador por pantalla utilizando printf().
    - Abre el archivo de jugadores en modo lectura ("r") utilizando fopen().
    - Si el archivo no se puede abrir (es NULL), muestra el mensaje "Error al abrir el fichero de jugadores" y retorna.
    - Declara un arreglo de estructuras jugadores para almacenar los jugadores y sus puntajes.
    - Inicia una variable numJugadores en 0 para contar el número de jugadores procesados.
    - Declara la variable nombreJugador para almacenar temporalmente los nombres de los jugadores.
    - Lee los nombres de todos los jugadores desde el archivo "jugadores.txt" utilizando fgets().
    - Elimina el salto de línea ('\n') del nombre del jugador utilizando strcspn().
    - Construye el nombre del archivo de resultados para cada jugador y lo abre en modo lectura ("r").
    - Si el archivo no se puede abrir (es NULL), muestra el mensaje "Error al abrir el fichero de resultados para el jugador" y continúa con el siguiente jugador.
    - Lee los resultados del jugador desde el archivo de resultados utilizando fscanf().
    - Cierra el archivo de resultados del jugador.

- Calcula el puntaje del jugador actual utilizando la fórmula (partidasGanadas - partidasPerdidas) / totalPartidas.
- Almacena el nombre del jugador y su puntaje en el arreglo de jugadores.
- Incrementa el contador numJugadores.
- Si se han procesado 20 jugadores o más, se sale del bucle.
- Cierra el archivo de jugadores con fclose().
- Ordena el arreglo de jugadores según sus puntajes de forma descendente utilizando el algoritmo de burbuja.
- Muestra los puntajes de los 5 mejores jugadores por pantalla utilizando printf().

4. La función verificarJugadorExistente(const char* nombre) verifica si un jugador ya existe en el archivo "jugadores.txt". Realiza lo siguiente:
   - Abre el archivo "jugadores.txt" en modo lectura ("r") utilizando fopen().
   - Si el archivo no se puede abrir (es NULL), muestra el mensaje "Error al abrir el fichero de jugadores" y retorna false.
   - Declara una variable linea de tipo char para almacenar temporalmente cada línea del archivo.
   - Lee cada línea del archivo utilizando fgets() y almacena la línea en linea.
   - Elimina el carácter de nueva línea ('\n') al final de la línea utilizando strcspn().
   - Compara la línea con el nombre proporcionado utilizando strcmp().
   - Si son iguales, significa que el nombre ya existe en el archivo. Cierra el archivo con fclose() y retorna true.
   - Cierra el archivo con fclose() y retorna false.
5. La función guardarJugador(const char* nombre) guarda un jugador en el archivo "jugadores.txt". Realiza lo siguiente:
   - Verifica si el jugador ya existe en el archivo utilizando la función verificarJugadorExistente().
   - Si el jugador ya existe, retorna sin hacer nada.
   - Abre el archivo "jugadores.txt" en modo anexar ("a") utilizando fopen().
   - Si el archivo no se puede abrir (es NULL), muestra el mensaje "Error al abrir el fichero de jugadores" y retorna.
   - Escribe el nombre del jugador en una nueva línea en el archivo utilizando fprintf().
   - Cierra el archivo con fclose().
6. La función longitudLista(ListaCoordenadas* lista) devuelve la longitud de una lista enlazada. Realiza lo siguiente:
   - Declara una variable longitud e inicialízala en 0 para contar los nodos de la lista.
   - Inicializa una variable actual con la cabeza de la lista.
   - Recorre la lista mientras el nodo actual no sea NULL.
   - Incrementa la longitud en 1.
   - Avanza al siguiente nodo.
   - Retorna la longitud de la lista.
7. La función eliminarNodo(ListaCoordenadas* lista, Nodo* nodoEliminar) elimina un nodo de una lista enlazada. Realiza lo siguiente:
   - Verifica si el nodo a eliminar es la cabeza de la lista.
   - Si es la cabeza, actualiza la cabeza de la lista al siguiente nodo.
   - Si no es la cabeza, recorre la lista hasta encontrar el nodo anterior al nodo a eliminar.
   - Si se encuentra, actualiza el puntero siguiente del nodo anterior para saltar el nodo a eliminar.
   - Libera la memoria del nodo a eliminar utilizando free().
8. La función igualarListas(ListaCoordenadas* listaDestino, ListaCoordenadas* listaOrigen) copia los elementos de una lista de origen en una lista de destino. Realiza lo siguiente:
   - Inicializa una variable actual con la cabeza de la lista de origen.
   - Recorre la lista de origen mientras el nodo actual no sea NULL.
   - Agrega la coordenada del nodo actual a la lista de destino utilizando la función agregarCoordenada().
   - Avanza al siguiente nodo.
9. La función seleccionarAleatorio(ListaCoordenadas* lista, int* fila, int* columna) selecciona de forma aleatoria un elemento de la lista enlazada y obtiene sus coordenadas de fila y columna. Realiza lo siguiente:
   - Obtiene la longitud de la lista utilizando la función longitudLista().
   - Verifica si la longitud es mayor que 0.
   - Si es mayor que 0, continúa con el proceso de selección aleatoria.
   - Si es igual a 0, asigna valores por defecto (-1) a las variables de fila y columna y retorna.
   - Genera un índice aleatorio entre 0 y longitud-1 utilizando rand().
   - Recorre la lista hasta llegar al nodo correspondiente al índice aleatorio.
   - Inicializa una variable actual con la cabeza de la lista.
   - Utiliza un bucle for para avanzar i veces en la lista hasta llegar al nodo deseado.

- Asigna las coordenadas del nodo seleccionado a las variables de fila y columna mediante desreferenciación de punteros.
- Elimina el nodo seleccionado de la lista utilizando la función eliminarNodo().

10. La función verificarTablero(ListaCoordenadas* lista, int** tablero, int filas, int columnas) verifica si todas las coordenadas del tablero son distintas de cero y están presentes en la lista. Realiza lo siguiente:
    - Itera sobre cada coordenada del tablero utilizando dos bucles for anidados.
    - Inicializa fila en 0 y realiza un bucle mientras fila sea menor que filas.
    - Inicializa columna en 0 y realiza un bucle mientras columna sea menor que columnas.
    - Verifica si el valor de la coordenada en el tablero es igual a cero.
    - Si es igual a cero, verifica si la coordenada no está en la lista utilizando la función coordenadaExistente().
    - Si la coordenada no está en la lista, retorna 0, indicando que existe al menos una coordenada igual a cero que no está en la lista.
    - Si el bucle anterior no encuentra ninguna coordenada igual a cero que no esté en la lista, retorna 1, indicando que todas las coordenadas son válidas.
11. La función coordenadaExistente(ListaCoordenadas *lista, int x, int y) verifica si una coordenada (x, y) ya existe en la lista enlazada. Realiza lo siguiente:
    - Inicializa una variable actual con la cabeza de la lista.
    - Recorre la lista mientras el nodo actual no sea NULL.
    - Verifica si las coordenadas del nodo actual son iguales a (x, y).
    - Si son iguales, retorna 1, indicando que la coordenada ya existe en la lista.
    - Avanza al siguiente nodo.
    - Si el bucle anterior no encuentra la coordenada en la lista, retorna 0, indicando que la coordenada no existe.
12. La función verificarGanador(ListaCoordenadas* listaVerificar, int filas, int columnas, ListaCoordenadas* listaEvitar) verifica si hay un cuadrado formado por las coordenadas presentes en la listaVerificar. Realiza lo siguiente:
    - Define dos funciones internas, isCoordenadaPresente(ListaCoordenadas* lista, int x, int y) y tieneCuadrado(Nodo* cabeza).
    - La función isCoordenadaPresente verifica si una coordenada (x, y) está presente en la lista especificada.
    - La función tieneCuadrado verifica si hay un cuadrado formado por las coordenadas en la lista. Recorre la listaVerificar y para cada coordenada (x1, y1), compara con todas las demás coordenadas (x2, y2) para verificar si forman un cuadrado.
    - Calcula las coordenadas (x3, y3) y (x4, y4) del cuadrado utilizando las fórmulas dadas en el código.
    - Verifica si las coordenadas (x3, y3) y (x4, y4) cumplen con las condiciones para agregarlas a la listaEvitar utilizando la función coordenadaExistente y la función agregarCoordenada.
    - Verifica si las coordenadas (x3, y3) y (x4, y4) están presentes en la listaVerificar utilizando la función isCoordenadaPresente.
    - Si se encuentra un cuadrado que cumple con las condiciones, se imprime un mensaje de cuadrado encontrado y se retorna 1 indicando que se encontró un ganador.
    - Llama a la función tieneCuadrado pasando la cabeza de la listaVerificar.
    - Si la función tieneCuadrado retorna true (1), significa que se encontró un cuadrado y se retorna true indicando que hay un ganador.
    - Si la función tieneCuadrado retorna false (0), significa que no se encontró un cuadrado y se retorna false indicando que no hay ganador.
13. La función jugarPartida(const char* nick) implementa la lógica para jugar una partida del juego. Realiza lo siguiente:
    - Inicializa varias banderas booleanas: bandera, bandera_ganado, bandera_perdido, bandera_empatado y bandera_partida para controlar el estado del juego.
    - Llama a la función guardarJugador(nick) para guardar el nombre del jugador.
    - Crea diferentes listas enlazadas para almacenar las coordenadas del jugador, la computadora y las coordenadas a evitar.
    - Construye los nombres de los archivos de configuración, resultados y última partida utilizando el nombre del jugador.
    - Abre los archivos de configuración, resultados y última partida en los modos correspondientes (r para lectura y wb para escritura binaria).
    - Si ocurre un error al abrir los archivos, muestra un mensaje de error y retorna.
    - Lee las dimensiones del tablero del archivo de configuración.
    - Cierra el archivo de configuración.
    - Crea un tablero utilizando la función crearTablero(filas, columnas).
    - Inicializa el tablero con ceros.
    - Establece el jugador actual como JUGADOR.

- Establece la bandera partidaTerminada como false para indicar que la partida no ha terminado.
- Dentro del ciclo principal while (!partidaTerminada), se muestra el tablero actual utilizando un par de bucles for.
- Dependiendo del jugador actual, se solicita al usuario que ingrese la fila y la columna donde quiere depositar su ficha o la computadora genera una posición aleatoria.
- Se valida la posición ingresada o generada, verificando si es una posición válida y si está vacía en el tablero. En caso de ser inválida, se muestra un mensaje y se continúa con el siguiente ciclo del bucle while.
- Se actualiza el tablero con la ficha del jugador actual.
- Si el jugador actual es el jugador humano, se agrega la coordenada a la lista de coordenadas del jugador y se verifica si ha formado un cuadrado. En caso de haber formado un cuadrado, se muestra un mensaje y se establece partidaTerminada y bandera_perdido como true.
- Si el jugador actual es la computadora, se agrega la coordenada a la lista de coordenadas de la computadora y se verifica si ha formado un cuadrado. En caso de haber formado un cuadrado, se muestra un mensaje y se establece partidaTerminada y bandera_ganado como true.
- Se verifica si el tablero está lleno y, si es así, se establece partidaTerminada, bandera_empatado y bandera_partida como true, y se muestra un mensaje de empate.
- Se cambia el jugador actual alternando entre JUGADOR y COMPUTADORA.
- Después del bucle while, se muestra el tablero final utilizando los mismos bucles for.
- A continuación, se actualiza el archivo de resultados:
- Si el archivo de resultados está vacío, se establecen valores iniciales para los resultados.
- Si el archivo de resultados contiene resultados anteriores, se leen los valores existentes del archivo.
- Se actualizan los resultados según el resultado de la partida actual.
- Se guarda la información actualizada en el archivo de resultados.
- Se guarda la última partida en el archivo correspondiente.
- Se libera la memoria utilizada por el tablero.
- Se muestra un mensaje indicando que la partida ha terminado y que los resultados han sido actualizados.

14. crearTablero:
    - Esta función se encarga de crear un tablero dinámico de dimensiones filas x columnas. Utiliza la función malloc para asignar memoria para las filas y columnas del tablero y luego devuelve un puntero al tablero creado.
15. liberarTablero:
    - Esta función se utiliza para liberar la memoria asignada al tablero dinámico. Recorre todas las filas del tablero y libera la memoria asignada a cada fila. Luego libera la memoria asignada para el arreglo de punteros a filas y finalmente libera la memoria asignada para el tablero en sí.
16. tableroLleno:
    - Esta función verifica si el tablero está lleno, es decir, si todas las posiciones del tablero contienen un valor diferente de cero. Recorre todas las filas y columnas del tablero y si encuentra una posición con valor cero, devuelve false. Si no encuentra ninguna posición vacía, devuelve true.
17. agregarCoordenada:
    - Esta función se utiliza para agregar una coordenada (fila, columna) a una lista de coordenadas. Crea un nuevo nodo de tipo Nodo y asigna los valores de fila y columna a la estructura de coordenadas del nodo. Luego, verifica si la lista está vacía y si es así, establece el nuevo nodo como la cabeza de la lista. Si la lista no está vacía, recorre la lista hasta llegar al último nodo y agrega el nuevo nodo al final de la lista.
18. mostrarCoordenadas:
    - Esta función se utiliza para mostrar las coordenadas almacenadas en una lista de coordenadas. Recorre la lista de nodos comenzando desde la cabeza y muestra la fila y columna de cada coordenada en la consola. Actualiza el puntero actual al siguiente nodo en cada iteración hasta que alcanza el final de la lista.

# LISTADO DEL CODIGO FUENTE

## A. "Librerias.h".

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

// Definir las constantes para los valores de las fichas
#define JUGADOR 1
#define COMPUTADORA 2

#define MAX_NICK_LENGTH 100
#define MAX_RESULTADOS_FILENAME (MAX_NICK_LENGTH + 15)

// Estructura para almacenar las coordenadas en formato (fila, columna)
typedef struct {
    int fila;
    int columna;
} Coordenada;

typedef struct Nodo {
    Coordenada coordenada;
    struct Nodo* siguiente;
} Nodo;

typedef struct {
    Nodo* cabeza;
} ListaCoordenadas;

typedef struct {
    int totalPartidas;
    int partidasGanadas;
    int partidasPerdidas;
    int partidasEmpatadas;
} ResultadosJugador;

typedef struct {
    char nombre[MAX_NICK_LENGTH];
    float puntaje;
} Jugador;


// Funciones prototipo
void mostrarConfiguracion(const char* nick);
void configurarParametros(const char* nick);
void verEstadisticas(const char* nick);
void jugarPartida(const char* nick);
void mostrarAyuda();

//Funciones utilizadas durante el juego
int** crearTablero(int filas, int columnas);
void liberarTablero(int** tablero, int filas);
bool verificarGanador(ListaCoordenadas* listaVerificar, int filas, int columnas, ListaCoordenadas* listaEvitar);
bool tableroLleno(int** tablero, int filas, int columnas);
void agregarCoordenada(ListaCoordenadas* lista, int fila, int columna);
void mostrarCoordenadas(ListaCoordenadas* lista);
int numCoordenadas(Nodo* cabeza);
int verificarTablero(ListaCoordenadas *lista, int** tablero, int filas, int columnas);
int coordenadaExistente(ListaCoordenadas* lista, int x, int y);
int longitudLista(ListaCoordenadas* lista);
void eliminarNodo(ListaCoordenadas* lista, Nodo* nodoEliminar);
void seleccionarAleatorio(ListaCoordenadas* lista, int* fila, int* columna);
void guardarJugador(const char* nombre);
```

## B. “Main.c”

“main.c”

```
#include"Librerias.h"
int filas;
int columnas;

int main()
{
   char nick[MAX_NICK_LENGTH];
   printf("Ingresa tu nick para comenzar el juego: ");
   scanf("%s", nick);

   // Crear el archivo de configuración si no existe
   char nombreFichero[MAX_NICK_LENGTH + 15];
   FILE* fichero;

   strcpy(nombreFichero, nick);
   strcat(nombreFichero, "_configuracion.txt");

   fichero = fopen(nombreFichero, "r");
   if (fichero == NULL) {
      fichero = fopen(nombreFichero, "w");
      if (fichero == NULL) {
         printf("Error al crear el fichero de configuración.\n");
         return 1;
      }

      int filas = 10;
      int columnas = 10;
      fprintf(fichero, "%d %d", filas, columnas);

      fclose(fichero);
   } else {
      fclose(fichero);
   }

   int opcion;

   // Crear el archivo de resultados si no existe
   char nombreFicheroResultados[MAX_NICK_LENGTH + 15];
   FILE* ficheroResultados;

   sprintf(nombreFicheroResultados, "%s_resultados.txt", nick);

   ficheroResultados = fopen(nombreFicheroResultados, "r");
   if (ficheroResultados == NULL) {
      ficheroResultados = fopen(nombreFicheroResultados, "w");
      if (ficheroResultados == NULL) {
         printf("Error al crear el fichero de resultados.\n");
         return 1;
      }

      fclose(ficheroResultados);
   } else {
      fclose(ficheroResultados);
   }

   do {
      printf("\nMenu para jugar\n");
      printf("1. Visualizar configuracion del tablero\n");
      printf("2. Configurar parametros\n");
      printf("3. Ver estadisticas\n");
      printf("4. Jugar partida\n");
      printf("5. Ayuda\n");
      printf("6. Salir\n");
      printf("Elige una opcion: ");
      scanf("%d", &opcion);

      switch (opcion) {
         case 1:
            mostrarConfiguracion(nick);
            break;
         case 2:
```

```
            configurarParametros(nick);
            break;
        case 3:
            verEstadisticas(nick);
            break;
        case 4:
            jugarPartida(nick);
            break;
        case 5:
            mostrarAyuda(nick);
            break;
    }
} while (opcion != 6);

return 0;
}
```

## C. “Funciones.c”

```
#include"Librerias.h"
void mostrarConfiguracion(const char* nick)
{
    char nombreFichero[MAX_NICK_LENGTH + 15];
    FILE* fichero;
    int filas, columnas;

    // Construir el nombre del fichero de configuracion
    strcpy(nombreFichero, nick);
    strcat(nombreFichero, "_configuracion.txt");

    // Abrir el fichero en modo lectura
    fichero = fopen(nombreFichero, "r");
    if (fichero == NULL) {
        printf("Error al abrir el fichero de configuracion\n");
        return;
    }

    // Leer el tamano del tablero
    fscanf(fichero, "%d %d", &filas, &columnas);

    // Cerrar el fichero
    fclose(fichero);

    // Mostrar la configuracion
    printf("Configuracion del tablero:\n");
    printf("Filas: %d\n", filas);
    printf("Columnas: %d\n", columnas);
}

void configurarParametros(const char* nick)
{
    char nombreFichero[MAX_NICK_LENGTH + 15];
    FILE* fichero;
    int filas, columnas;

    // Construir el nombre del fichero de configuracion
    strcpy(nombreFichero, nick);
    strcat(nombreFichero, "_configuracion.txt");

    // Abrir el fichero en modo lectura
    fichero = fopen(nombreFichero, "r");
    if (fichero == NULL) {
        printf("Error al abrir el fichero de configuracion.\n");
        return;
    }

    // Leer el tamano actual del tablero
    fscanf(fichero, "%d %d", &filas, &columnas);

    // Cerrar el fichero
    fclose(fichero);

    // Mostrar la configuracion actual
```

```
    printf("Configuracion actual del tablero:\n");
    printf("Filas: %d\n", filas);
    printf("Columnas: %d\n", columnas);

    // Solicitar al usuario el nuevo tamano del tablero
    printf("Ingresa la nueva escala del tablero (filas columnas): ");
    int nuevasFilas, nuevasColumnas;
    scanf("%d %d", &nuevasFilas, &nuevasColumnas);

    // Validar el nuevo tamano del tablero
    if (nuevasFilas <= 0 || nuevasColumnas <= 0 || nuevasFilas > 10 || nuevasColumnas > 10) {
        printf("Error: La escala del tablero debe ser mayor que 0 y no puede superar 10 filas y 10 columnas.\n");
        return;
    }

    // Abrir el fichero en modo escritura para actualizar la configuracion
    fichero = fopen(nombreFichero, "w");
    if (fichero == NULL) {
        printf("Error al abrir el fichero de configuracion.\n");
        return;
    }

    // Escribir el nuevo tamano del tablero en el fichero
    fprintf(fichero, "%d %d", nuevasFilas, nuevasColumnas);

    // Cerrar el fichero
    fclose(fichero);

    printf("Se ha actualizado la configuracion del tablero.\n");
    printf("Nuevo tamano del tablero:\n");
    printf("Filas: %d\n", nuevasFilas);
    printf("Columnas: %d\n", nuevasColumnas);
}

void verEstadisticas(const char* nick)
{
    // Construir el nombre del fichero de resultados
    char nombreResultados[MAX_NICK_LENGTH + 14];
    sprintf(nombreResultados, "%s_resultados.txt", nick);

    FILE* ficheroResultados = fopen(nombreResultados, "r");
    if (ficheroResultados == NULL) {
        printf("Error al abrir el fichero de resultados.\n");
        return;
    }

    ResultadosJugador resultados;
    fscanf(ficheroResultados, "Total de partidas jugadas: %d\n", &resultados.totalPartidas);
    fscanf(ficheroResultados, "Partidas ganadas: %d\n", &resultados.partidasGanadas);
    fscanf(ficheroResultados, "Partidas perdidas: %d\n", &resultados.partidasPerdidas);
    fscanf(ficheroResultados, "Partidas empatadas: %d\n", &resultados.partidasEmpatadas);
    fclose(ficheroResultados);

    // Calcular el porcentaje de partidas ganadas, perdidas y empatadas
    float porcentajeGanadas = (float)resultados.partidasGanadas / resultados.totalPartidas * 100;
    float porcentajePerdidas = (float)resultados.partidasPerdidas / resultados.totalPartidas * 100;
    float porcentajeEmpatadas = (float)resultados.partidasEmpatadas / resultados.totalPartidas * 100;

    // Mostrar las estadísticas
    printf("Estadisticas para el jugador: %s\n", nick);
    printf("Total de partidas jugadas: %d\n", resultados.totalPartidas);
    printf("Partidas ganadas: %d (%.2f%%)\n", resultados.partidasGanadas, porcentajeGanadas);
    printf("Partidas perdidas: %d (%.2f%%)\n", resultados.partidasPerdidas, porcentajePerdidas);
    printf("Partidas empatadas: %d (%.2f%%)\n", resultados.partidasEmpatadas, porcentajeEmpatadas);

    // Mostrar los puntajes de los 5 mejores jugadores


        // Leer los nombres de todos los jugadores desde el archivo "jugadores"
    FILE* ficheroJugadores = fopen("jugadores.txt", "r");
    if (ficheroJugadores == NULL) {
        printf("Error al abrir el fichero de jugadores.\n");
        return;
    }
```

```
    Jugador jugadores[20];
    int numJugadores = 0;
    char nombreJugador[MAX_NICK_LENGTH];

    while (fgets(nombreJugador, MAX_NICK_LENGTH, ficheroJugadores) != NULL) {
        // Eliminar el salto de línea del nombre del jugador
        nombreJugador[strcspn(nombreJugador, "\n")] = '\0';

        // Construir el nombre del archivo de resultados para el jugador actual
        char nombreResultadosJugador[MAX_RESULTADOS_FILENAME];
        sprintf(nombreResultadosJugador, "%s_resultados.txt", nombreJugador);

        FILE* ficheroResultadosJugador = fopen(nombreResultadosJugador, "r");
        if (ficheroResultadosJugador == NULL) {
            printf("Error al abrir el fichero de resultados para el jugador %s.\n", nombreJugador);
            continue;
        }

        // Leer los resultados del archivo para el jugador actual
        ResultadosJugador resultadosJugador;
        resultadosJugador.totalPartidas = 0;
        resultadosJugador.partidasGanadas = 0;
        resultadosJugador.partidasPerdidas = 0;
        resultadosJugador.partidasEmpatadas = 0;

        fscanf(ficheroResultadosJugador, "Total de partidas jugadas: %d\n", &resultadosJugador.totalPartidas);
        fscanf(ficheroResultadosJugador, "Partidas ganadas: %d\n", &resultadosJugador.partidasGanadas);
        fscanf(ficheroResultadosJugador, "Partidas perdidas: %d\n", &resultadosJugador.partidasPerdidas);
        fscanf(ficheroResultadosJugador, "Partidas empatadas: %d\n", &resultadosJugador.partidasEmpatadas);
        fclose(ficheroResultadosJugador);

        // Calcular el puntaje del jugador actual
        float puntaje = (float)(resultadosJugador.partidasGanadas - resultadosJugador.partidasPerdidas) /
resultadosJugador.totalPartidas;

        // Almacenar el nombre del jugador y su puntaje en la estructura de jugadores
        strcpy(jugadores[numJugadores].nombre, nombreJugador);
        jugadores[numJugadores].puntaje = puntaje;
        numJugadores++;

        if (numJugadores >= 20) {
            break;  // Se alcanzo el máximo de jugadores a procesar
        }
    }

    fclose(ficheroJugadores);

    // Ordenar el arreglo de jugadores según sus puntajes de forma descendente
    for (int i = 0; i < numJugadores - 1; i++) {
        for (int j = 0; j < numJugadores - i - 1; j++) {
            if (jugadores[j].puntaje < jugadores[j + 1].puntaje) {
                Jugador temp = jugadores[j];
                jugadores[j] = jugadores[j + 1];
                jugadores[j + 1] = temp;
            }
        }
    }

    // Mostrar los puntajes de los 5 mejores jugadores
    printf("Puntaje de los 5 mejores jugadores:\n");
    for (int i = 0; i < numJugadores && i < 5; i++) {
        printf("%s - %.2f\n", jugadores[i].nombre, jugadores[i].puntaje);
    }

}

bool verificarJugadorExistente(const char* nombre)
{
    FILE* ficheroJugadores = fopen("jugadores.txt", "r");
    if (ficheroJugadores == NULL) {
        printf("Error al abrir el fichero de jugadores.\n");
        return false;
    }
```

```
    char linea[100];
    while (fgets(linea, sizeof(linea), ficheroJugadores) != NULL) {
        // Eliminar el carácter de nueva línea al final de la línea
        linea[strcspn(linea, "\n")] = '\0';

        if (strcmp(linea, nombre) == 0) {
            // El nombre ya existe en el archivo
            fclose(ficheroJugadores);
            return true;
        }
    }

    fclose(ficheroJugadores);
    return false;
}


void guardarJugador(const char* nombre)
{
    if (verificarJugadorExistente(nombre)) {
        //printf("El jugador ya existe en el archivo.\n");
        return;
    }

    FILE* ficheroJugadores = fopen("jugadores.txt", "a");
    if (ficheroJugadores == NULL) {
        printf("Error al abrir el fichero de jugadores.\n");
        return;
    }

    fprintf(ficheroJugadores, "%s\n", nombre);

    fclose(ficheroJugadores);
}
// Funcion para obtener la longitud de la lista enlazada
int longitudLista(ListaCoordenadas* lista) {
    int longitud = 0;
    Nodo* actual = lista->cabeza;
    while (actual != NULL) {
        longitud++;
        actual = actual->siguiente;
    }
    return longitud;
}

// Funcion para eliminar un nodo de la lista enlazada
void eliminarNodo(ListaCoordenadas* lista, Nodo* nodoEliminar) {
    if (lista->cabeza == nodoEliminar) {
        lista->cabeza = nodoEliminar->siguiente;
    } else {
        Nodo* actual = lista->cabeza;
        while (actual != NULL && actual->siguiente != nodoEliminar) {
            actual = actual->siguiente;
        }
        if (actual != NULL) {
            actual->siguiente = nodoEliminar->siguiente;
        }
    }
    free(nodoEliminar);
}

void igualarListas(ListaCoordenadas* listaDestino, ListaCoordenadas* listaOrigen)
{

    // Recorremos la lista de origen y copiamos los elementos a la lista de destino
    Nodo* actual = listaOrigen->cabeza;
    while (actual != NULL) {
        agregarCoordenada(listaDestino, actual->coordenada.fila, actual->coordenada.columna);
        actual = actual->siguiente;
    }
}

// Funcion para seleccionar un elemento aleatorio de la lista enlazada y obtener fila y columna
```

```
void seleccionarAleatorio(ListaCoordenadas* lista, int* fila, int* columna) {
    int longitud = longitudLista(lista);
    if (longitud > 0) {
        // Generar un índice aleatorio entre 0 y longitud-1
        int indice = rand() % longitud;

        // Recorrer la lista hasta llegar al nodo correspondiente al índice aleatorio
        Nodo* actual = lista->cabeza;
        for (int i = 0; i < indice; i++) {
            actual = actual->siguiente;
        }

        // Asignar las coordenadas del nodo seleccionado a fila y columna
        *fila = actual->coordenada.fila;
        *columna = actual->coordenada.columna;

        // Eliminar el nodo seleccionado de la lista
        eliminarNodo(lista, actual);
    } else {
        // Si la lista esta vacía, asignar valores por defecto a fila y columna
        *fila = -1;
        *columna = -1;
    }
}


int verificarTablero(ListaCoordenadas* lista, int** tablero, int filas, int columnas) {
  // Iterar sobre cada coordenada del tablero
  for (int fila = 0; fila < filas; fila++) {
    for (int columna = 0; columna < columnas; columna++) {
      // Verificar si el valor de la coordenada es igual a 0
      if (tablero[fila][columna] == 0) {
        // Verificar si la coordenada no esta en la lista
        if (!coordenadaExistente(lista, fila, columna)) {
          // Existe al menos una coordenada igual a 0 que no esta en la lista
          return 0;
        }
      }
    }
  }

  // No se encontro ninguna coordenada igual a 0 que no este en la lista
  return 1;
}

int coordenadaExistente(ListaCoordenadas *lista, int x, int y) {
  Nodo *actual = lista->cabeza;

  while (actual != NULL) {
    if (actual->coordenada.fila == x && actual->coordenada.columna == y) {
      // La coordenada ya existe en la lista
      return 1;
    }
    actual = actual->siguiente;
  }

  // La coordenada no existe en la lista
  return 0;
}

bool verificarGanador(ListaCoordenadas* listaVerificar, int filas, int columnas, ListaCoordenadas* listaEvitar)
{
   // printf("\nFILAS Y COLUMNAS: %d, %d\n", filas, columnas);
    Nodo* actual = listaVerificar->cabeza;

    //printf("Lista: PARA VERIFICAR EL GANADOR");
    while (actual != NULL) {
        //printf("(%d, %d) ", actual->coordenada.fila, actual->coordenada.columna);
        actual = actual->siguiente;
    }
    //printf("\n");

    int isCoordenadaPresente(ListaCoordenadas* lista, int x, int y) {
        Nodo* actual = lista->cabeza;
```

```
        while (actual != NULL) {
            if (actual->coordenada.fila == x && actual->coordenada.columna == y) {
                return 1;
            }
            actual = actual->siguiente;
        }
        return 0;
    }

    int tieneCuadrado(Nodo* cabeza) {
        Nodo* actual = cabeza;

        while (actual != NULL) {
            int x1 = actual->coordenada.fila;
            int y1 = actual->coordenada.columna;

            Nodo* siguienteNodo = actual->siguiente;
            while (siguienteNodo != NULL) {
                int x2 = siguienteNodo->coordenada.fila;
                int y2 = siguienteNodo->coordenada.columna;
                int x3 = x2 - (y2 - y1);
                int y3 = y2 + (x2 - x1);
                int x4 = x1 - (y2 - y1);
                int y4 = y1 + (x2 - x1);

                // Verificar si las coordenadas cumplen con las condiciones para agregarlas a la lista de coordenadas que la
computadora debe evitar
                 if (x3 >= 0 && x3 < filas &&
                    y3 >= 0 && y3 < columnas) {
                  if (x4 >= 0 && x4 < filas &&
                     y4 >= 0 && y4 < columnas) {
                    if (!coordenadaExistente(listaEvitar, x3, y3)) {
                      agregarCoordenada(listaEvitar, x3, y3);
                    }
                    if (!coordenadaExistente(listaEvitar, x4, y4)) {
                      agregarCoordenada(listaEvitar, x4, y4);
                    }
                    //printf("\nLas coordenadas que la computadora debe evitar son:\n");
                    mostrarCoordenadas(listaEvitar);
                  }
                }


                if (isCoordenadaPresente(listaVerificar, x3, y3) && isCoordenadaPresente(listaVerificar, x4, y4)) {
                    if ((x1 != x2 || y1 != y2) && (x1 != x3 || y1 != y3) && (x1 != x4 || y1 != y4)) {
                        printf("SE HA ENCONTRADO UN CUADRADO: (%d,%d) (%d,%d) (%d,%d) (%d,%d)\n", x1, y1,
x2, y2, x3, y3, x4, y4);
                        return 1;
                    }
                }

                siguienteNodo = siguienteNodo->siguiente;
            }

            actual = actual->siguiente;
        }

        return 0;
    }

    if (tieneCuadrado(listaVerificar->cabeza)) {
      return true;
    } else {
      return false;
    }
}


// Funcion para jugar una partida
void jugarPartida(const char* nick)
{
    bool bandera = false;
    bool bandera_ganado = false;
    bool bandera_perdido = false;
```

```
    bool bandera_empatado = false;
    bool bandera_partida = false;

    guardarJugador(nick);

    ListaCoordenadas listaCoordenadasUsuario;
    listaCoordenadasUsuario.cabeza = NULL;

    ListaCoordenadas listaCoordenadasComputadora;
    listaCoordenadasComputadora.cabeza = NULL;

    ListaCoordenadas listaCoordenadasEvitar1; //para que el jugador pierda
    listaCoordenadasEvitar1.cabeza = NULL;

    ListaCoordenadas listaCoordenadasEvitar2; //para que la computadora no pierda
    listaCoordenadasEvitar2.cabeza = NULL;

    ListaCoordenadas listaCoordenadasEvitar3; //cuando la computadora se queda sin poder evitar movimientos
    listaCoordenadasEvitar3.cabeza = NULL;

    ResultadosJugador resultados;

    // Construir los nombres de los ficheros
    char nombreConfiguracion[MAX_NICK_LENGTH + 15];
    char nombreResultados[MAX_NICK_LENGTH + 14];
    char nombreUltimaPartida[MAX_NICK_LENGTH + 16];
    sprintf(nombreConfiguracion, "%s_configuracion.txt", nick);
    sprintf(nombreResultados, "%s_resultados.txt", nick);
    sprintf(nombreUltimaPartida, "%s_ultimapartida.dat", nick);

    FILE* ficheroConfiguracion = fopen(nombreConfiguracion, "r");
    FILE* ficheroResultados = fopen(nombreResultados, "r");
    FILE* ficheroUltimaPartida = fopen(nombreUltimaPartida, "wb");

    if (ficheroConfiguracion == NULL || ficheroResultados == NULL || ficheroUltimaPartida == NULL) {
        printf("Error al abrir los ficheros.\n");
        return;
    }

    int filas, columnas;
    fscanf(ficheroConfiguracion, "%d %d", &filas, &columnas);
    fclose(ficheroConfiguracion);

    // Crear el tablero
    int** tablero = crearTablero(filas, columnas);

    // Inicializar tablero con ceros
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            tablero[i][j] = 0;
        }
    }

    int jugadorActual = JUGADOR;
    bool partidaTerminada = false;

    while (!partidaTerminada) {
        // Mostrar el tablero actual y las coordenadas del jugador
        printf("\nTablero actual:\n");
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                printf("%d ", tablero[i][j]);
            }
            printf("\n");
        }

        int fila, columna;
        if (jugadorActual == JUGADOR) {
            // Turno del jugador
            printf("Es tu turno. Ingresa la fila y columna donde quieres depositar tu ficha: ");
            scanf("%d %d", &fila, &columna);
        } else {
            // Turno de la computadora (algoritmo aleatorio)
            printf("Turno de la computadora.\n");
```

```
        // Generar una posicion aleatoria hasta encontrar una casilla vacía
        do {
            fila = rand() % filas;
            columna = rand() % columnas;

            // Verificar si la coordenada generada esta en la lista de coordenadas a evitar
            if (coordenadaExistente(&listaCoordenadasEvitar1, fila, columna) != 0 ||
coordenadaExistente(&listaCoordenadasEvitar2, fila, columna) != 0) {
                // La coordenada generada esta en la lista, buscar otra posicion
                //Verificar que exista al menos una posicion que sea igual a 0 y que no este en la lista.
            if(verificarTablero(&listaCoordenadasEvitar1,tablero,filas,columnas) == 0 ||
verificarTablero(&listaCoordenadasEvitar2, tablero,filas,columnas) == 0){

            }else{
                //Si ya no quedan posiciones en el tablero que se puedan evitar, entonces la computadora va a tomar las
posiciones en donde no pierda
                //es decir, evitara aquellas donde la computadora forme un cuadrado, y se posicionara en donde el
jugador forma un cuadrado.
                if (bandera == false){
                    //igualar la tercera lista a la primera para poder trabajar con ella en particular
                    igualarListas(&listaCoordenadasEvitar3, &listaCoordenadasEvitar1);
                    bandera = true;
                }
                seleccionarAleatorio(&listaCoordenadasEvitar3, &fila,&columna);
                //salir del bucle
                break;
                }

            }

            // La coordenada generada no esta en la lista, salir del bucle
            break;
        } while (1);
    }

    // Validar la posicion ingresada
    if (fila < 0 || fila >= filas || columna < 0 || columna >= columnas || tablero[fila][columna] != 0) {
        printf("Posicion invalida. Intenta de nuevo.\n");
        continue;
    }

  tablero[fila][columna] = jugadorActual;
      if (jugadorActual == JUGADOR) {
        agregarCoordenada(&listaCoordenadasUsuario, fila, columna);
        if(verificarGanador(&listaCoordenadasUsuario, filas, columnas, &listaCoordenadasEvitar1)){
            printf("\nHAS FORMADO UN CUADRADO. HAS PERDIDO LA PARTIDA");
            partidaTerminada = true;
            bandera_perdido = true;
        }
    } else {
        agregarCoordenada(&listaCoordenadasComputadora, fila, columna);
        if(verificarGanador(&listaCoordenadasComputadora, filas, columnas, &listaCoordenadasEvitar2)){
            printf("\nHAS GANADO. LA COMPUTADORA HA FORMADO UN CUADRADO");
            partidaTerminada = true;
            bandera_ganado = true;
        }
    }

    // Verificar si el tablero esta lleno
    if (tableroLleno(tablero, filas, columnas) == 1 && partidaTerminada == false) {
        partidaTerminada = true;
        bandera_empatado = true;
        bandera_partida = true;
        printf("\nHA RESULTADO UN EMPATE");
    }


    // Cambiar el jugador actual
    jugadorActual = (jugadorActual == JUGADOR) ? COMPUTADORA : JUGADOR;

  }

  // Mostrar el tablero final
  printf("\nTablero final:\n");
```

```
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            printf("%d ", tablero[i][j]);
        }
        printf("\n");
    }
// Actualizar el fichero de resultados
if (ficheroResultados == NULL) {
    // Establecer valores iniciales si el archivo está vacío
    resultados.totalPartidas = 0;
    resultados.partidasGanadas = 0;
    resultados.partidasPerdidas = 0;
    resultados.partidasEmpatadas = 0;
} else {
    // Leer los resultados existentes del archivo
    fscanf(ficheroResultados, "Total de partidas jugadas: %d\n", &resultados.totalPartidas);
    fscanf(ficheroResultados, "Partidas ganadas: %d\n", &resultados.partidasGanadas);
    fscanf(ficheroResultados, "Partidas perdidas: %d\n", &resultados.partidasPerdidas);
    fscanf(ficheroResultados, "Partidas empatadas: %d\n", &resultados.partidasEmpatadas);
}

// Actualizar los resultados según el resultado de la partida actual

if (bandera_partida){
        if (resultados.totalPartidas == 6421624){
            resultados.totalPartidas = 0;
        }
    resultados.totalPartidas++;
}

if (bandera_ganado) {
    resultados.partidasGanadas++;
} else if (bandera_perdido) {
    resultados.partidasPerdidas++;
} else if (bandera_empatado) {
    resultados.partidasEmpatadas++;
}

// Guardar los resultados actualizados en el archivo de resultados
ficheroResultados = fopen(nombreResultados, "w");
if (ficheroResultados == NULL) {
    printf("Error al abrir el fichero de resultados.\n");
    return;
}

        if (resultados.totalPartidas == 6421624){
            resultados.totalPartidas = 1;
        }else{
        resultados.totalPartidas = resultados.totalPartidas + 1;
        }
fprintf(ficheroResultados, "Total de partidas jugadas: %d\n", resultados.totalPartidas);
fprintf(ficheroResultados, "Partidas ganadas: %d\n", resultados.partidasGanadas);
fprintf(ficheroResultados, "Partidas perdidas: %d\n", resultados.partidasPerdidas);
fprintf(ficheroResultados, "Partidas empatadas: %d\n", resultados.partidasEmpatadas);

fclose(ficheroResultados);




    // Guardar la última partida en el fichero correspondiente
    fwrite(*tablero, sizeof(int), filas * columnas, ficheroUltimaPartida);
    fclose(ficheroUltimaPartida);

    // Liberar la memoria del tablero
    liberarTablero(tablero, filas);

    printf("Partida terminada. Los resultados han sido actualizados.\n");
}

// Funcion para crear un tablero dinamico
int** crearTablero(int filas, int columnas)
{
    int** tablero = (int**)malloc(filas * sizeof(int*));
```

```
    for (int i = 0; i < filas; i++) {
        tablero[i] = (int*)malloc(columnas * sizeof(int));
    }
    return tablero;
}

// Funcion para liberar la memoria del tablero
void liberarTablero(int** tablero, int filas)
{
    for (int i = 0; i < filas; i++) {
        free(tablero[i]);
    }
    free(tablero);
}


// Funcion para verificar si el tablero esta lleno
bool tableroLleno(int** tablero, int filas, int columnas)
{
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            if (tablero[i][j] == 0) {
                return false;
            }
        }
    }
    return true;
}

void agregarCoordenada(ListaCoordenadas* lista, int fila, int columna)
{
    Nodo* nuevoNodo = malloc(sizeof(Nodo));
    nuevoNodo->coordenada.fila = fila;
    nuevoNodo->coordenada.columna = columna;
    nuevoNodo->siguiente = NULL;

    if (lista->cabeza == NULL) {
        lista->cabeza = nuevoNodo;
    } else {
        Nodo* actual = lista->cabeza;
        while (actual->siguiente != NULL) {
            actual = actual->siguiente;
        }
        actual->siguiente = nuevoNodo;
    }
}

void mostrarCoordenadas(ListaCoordenadas* lista)
{
    Nodo* actual = lista->cabeza;
    while (actual != NULL) {
        //printf("Fila: %d, Columna: %d\n", actual->coordenada.fila, actual->coordenada.columna);
        actual = actual->siguiente;
    }
}

void mostrarAyuda()
{
    printf("=== AYUDA ===\n");
    printf("El objetivo del juego es formar cuadrados en un tablero.\n");
    printf("Cada jugador, ya sea humano o computadora, coloca una ficha en el tablero.\n");
    printf("Cuando un jugador forma un cuadrado con sus fichas, gana la partida.\n");
    printf("Si el tablero se llena y nadie ha formado un cuadrado, la partida termina en empate.\n");
    printf("Durante tu turno, ingresa la fila y la columna donde deseas colocar tu ficha.\n");
    printf("Si en algún momento necesitas ayuda, puedes usar la opcion 'ayuda' para mostrar esta informacion
nuevamente.\n");
    printf("¡Diviertete jugando!\n");
}
```

# BIBLIOGRAFIA


- Kernighan, B. W., & Ritchie, D. M. (1988). El lenguaje de programación C. Prentice Hall.
- Prata, S. (2013). C programming Absolute Beginner's Guide (3rd Edition). Que Publishing.
- Deitel, H. M., & Deitel, P. J. (2015). C How to Program (8th Edition). Pearson.
- King, K. N. (2008). C Programming: A Modern Approach (2nd Edition). W. W. Norton & Company.
- Schildt, H. (2017). C: The Complete Reference (4th Edition). McGraw-Hill Education.