**Informe final**

Para que el usuario interactúe con el programa principal, se implementan funciones declaradas en el archivo herramientas.h, dichas funciones son usadas para imprimir el menú del programa, manejar el historial de las partidas y validar las entradas del usuario por teclado.

La función imprimir\_menu, recibe como parámetro un entero y de acuerdo a este muestra por pantalla el menú asociado, de la siguiente manera:

* Evalúa con un switch el número que se recibe como parámetro
  + si el número es igual a uno se imprime el menú de inicio del programa
  + si el número es igual a dos se imprime el menú del final de cada partida

Opcion\_invalida de tipo booleano, recibe como parámetro un string, el cual será asignado por el usuario y corresponde a la opción seleccionada en cada menú, por ende esta función tiene como objetivo validar que dicha opción se encuentre entre los valores posibles

* Si la opción es igual a uno, se retorna falso
* De lo contrario y si la opción es dos, se retorna falso
* De lo contrario y si la opción es tres, se retorna falso
* De lo contrario, el valor no coincide con una opción válida dispuesta en el menú, entonces se imprime un mensaje de error y se retorna verdadero

El valor retornado es usado en un ciclo do while, dispuesto en el programa principal, que garantiza que el usuario ingresa una opción válida.

Las funciones guardar\_partida y mostrar\_historial, corresponden al manejo de archivos que contienen la información de las partidas, por ende en ambas se declara un objeto de la clase fstream

Guardar\_partida, se encarga de almacenar la información de cada juego en un archivo llamado historial.txt, recibe como parámetros la cantidad de fichas con las que se gano o se empato el juego, el nombre de los jugadores y el nombre del ganador

* Se toma la fecha y hora actual haciendo uso de la librería time.h, declarando un objeto de la clase time\_t para poder acceder al método time() con su dirección de memoria
* Creo el string con la información contenida en el formato deseado, en este caso para poder escribir sobre el archivo, se convierte la cantidad de fichas de tipo entero a string y la fecha y hora se pasa por la función ctime() que convierte el tipo time\_t a caracteres
* Se abre el archivo por medio del objeto fstream en el modo escritura y en el modo agregar al final del archivo
* Con un condicional se verifica que el archivo se abre
* Se agrega el string con la información y se cierra el archivo

Mostrar\_historial, se encarga de imprimir por pantalla la información contenida en el archivo historial.txt

* Se abre el documento en el modo lectura
* Con un condicional if se verifica si el archivo no pudo ser abierto
  + Se imprime un mensaje dado que aún no se ha jugado una partida en el juego y no se posee historial
* Nos mantenemos en un ciclo while mientras no lleguemos al final del archivo
  + Obtenemos la línea actual del archivo file que será almacenada en la variable de tipo string línea
  + Imprime la línea
* Cierra el archivo

El juego y su lógica usan 2 clases, una para representar las piezas y otra para representar

una partida, la clase casilla utiliza una variable de tipo char para representar su estado actual, en su creación este estado es ‘ ‘ para representar una casilla vacía,además de eso tiene una función setter para cambiar su estado a ‘\*’ o ‘-’ dependiendo de el jugador en turno y una función getter para obtener el valor que almacena.

La clase que representa las partidas llamada juego usa una matriz de tamaño variable para representar el tablero, el cual almacena un objeto de tipo casilla en cada una de sus posiciones, estas son las funciones que hacen parte de la clase juego:

* juego(int filas, int columnas):
  + crear una partida con un tablero de tamaño filas x columnas:
    - crea un array de apuntadores a objetos de tipo casilla de tamaño fila llamado tablero.
    - en cada espacio del array tablero crea un array de objetos tipo casilla de tamaño columna.
  + cambia las posiciones centrales para crear la posición inicial del tablero:
    - utiliza la función set\_estado de la clase casilla para cambiar el estado de los objetos casilla ubicados [(filas/2)-1][(columnas/2)-1] y [(filas/2)][(columnas/2)].
    - Se hace lo mismo para las piezas del otro jugador en las casillas [(filas/2)][(columnas/2)-1] y [(filas/2)-1][(columnas/2)].
    - Se almacena también el tamaño del tablero para su uso dentro de la clase.
* ~juego():
  + elimina un elemento de la clase juego:
    - recorre el array tablero eliminando cada uno de sus elementos.
    - elimina el array tablero.
* print\_tablero():
  + crea un string con el tablero para mostrarle al usuario:
    - crea el string “tabla” y almacena la parte superior izquierda del tablero que siempre está vacía para poder mostrar la guía del nombre de las casillas.
    - itera “columnas” veces agregando la guía de cada una de las columnas “tabla”.
    - itera “filas” veces agregando la guía de cada una de las filas y tiene un ciclo interno que recorre cada casilla en la fila obteniendo su valor mediante la función get\_estado de la clase casillas y agregándolo a “tabla”.
    - retorna el string “tabla”.
* movimiento\_valido(std::string casilla, int turno):
  + determina qué piezas son del jugador que no está en turno según el turno en el que se invoca:
    - si el turno es par el jugador en turno tiene las piezas representadas por ‘-’, por lo tanto el jugador que no está en turno tiene las piezas representadas por ‘\*’.
    - de lo contrario el jugador que no está en turno tiene las piezas representadas por ‘-’.
  + revisa que la casilla ingresada sea una casilla válida y en qué dirección están las piezas que ese movimiento cambia:
    - convierte los caracteres de ascii a la posición que representan en el tablero.
    - crea la variable de tipo string “validez” para almacenar las direcciones que el movimiento altera.
    - crea la variable de apoyo booleana “encontrado”.
    - revisa que la fila y la columna ingresados estén dentro de los límites del tablero y que la cadena tenga sólo 2 caracteres, de lo contrario establece validez a “”;
    - usa un ciclo para ver las piezas en cada una de las 8 direcciones que rodean a la casilla ingresada y actúa diferente dependiendo del estado de la casilla que encuentra:
      * si encuentra una casilla vacía esa dirección es inválida y por lo tanto termina esa comprobación y pasa a la siguiente.
      * si se encuentra con una pieza del oponente le asigna true a la variable “encontrada”.
      * si se encuentra con una pieza del jugador en turno ocurre una de dos cosas:
        + si la variable encontrada “encontrada” es true significa que ya se había encontrado con una de las piezas del oponente y por lo tanto un encierro tipo sándwich ha ocurrido y que el movimiento es válido así que agrega el número correspondiente a su dirección a “validez”.
        + de lo contrario significa que el movimiento no genera un encierro tipo sándwich y que esa dirección es invalida
    - revisa que la posición no esté ocupada por otra pieza, en caso de estarlo establece “validez” a “”.
    - retorna “validez”.
* movimiento\_valido(int fila, int columna, int turno):
  + sigue el mismo proceso que la función anterior pero recibe la casilla en forma de fila y columna y retorna un valor booleano en lugar de la dirección que afecta el movimiento.
* movimientos\_disponibles(int turno):
  + crea un string son los movimientos disponibles según cual es el jugador en turno:
    - itera sobre cada casilla haciendo llamados a movimiento\_valido(int fila, int columna, int turno).
    - en caso de que la función devuelva un valor true se transforma la coordenada a ascii y se agregan al string.
    - retorna el string para mostrarle al usuario.
* tomar\_turno(int turno, std::string casilla):
  + realiza los cambios de cada turno al tablero:
    - revisa qué piezas tiene cada jugador.
    - llama a la función movimiento\_valido(std::string casilla, int turno) para saber en qué dirección están los encierros.
    - cada una de esas direcciones se recorre y se cambian las piezas del oponente hasta que se encuentra con una casilla propia o una casilla vacía.
* contador\_j1():
  + cuenta cuántas piezas tiene el jugador 1:
    - recorre todas las casillas invocando a la función get estado de la clase casillas y al encontrarse con ‘\*’ suma 1 a la cuenta.
    - retorna la cantidad de casillas.
* contador\_j2():
  + cuenta cuántas piezas tiene el jugador 2:
    - recorre todas las casillas invocando a la función get estado de la clase casillas y al encontrarse con ‘-’ suma 1 a la cuenta.

En cuanto al ciclo principal de juego el juego se repite mientras haya un movimiento disponible o el movimiento anterior no hubiera sido un pase de turno representado en la variable game\_over:

* Revisa que haya movimientos disponibles, en caso de no haberlos se lo comunica al jugador y pasa el turno almacenando esta información en game\_over.
* Se asegura que la variable game\_over vuelva a su estado original.
* Le muestra el turno actual al jugador y que símbolo representa la pieza del jugador en turno.
* Le muestra al usuario el tablero de juego

y qué movimientos puede realizar.

* En caso de que sea el primer turno o de que el jugador se equivoque al ingresar su movimiento muestra cómo debe ser ingresado el movimiento y lo recibe hasta que sea un movimiento válido.
* De lo contrario solo recibe el movimiento
* Se ejecutan los cambios correspondientes al movimiento del usuario.
* se termina el turno.