**INFORME SEGUNDO PARCIAL**

Andres Felipe Restrepo López

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

Universidad de Antioquia

2023

**Informe sobre proceso del trabajo de informática 2.**

-Como primer paso se procede a jugar eothello para poder entender su forma de jugar y cuáles son las reglas en acción, para poder ir ideando los pasos a seguir para idear las clases necesarias para poder desarrollar el juego.

Gráfico de burbujas

Descripción generada automáticamente

-El día de hoy 25/10/2023, procedí a realizar un adelanto de algunas clases que tengo pensado implementar en el desarrollo del juego todo esto lo realicen en el main para realizar pruebas, eventual mente en próximos avances organizare mejor el archivo y desarrollare las condiciones para que las fichas funcionen con la normativa correcta del juego. En el momento he creado los principales cascarones de las siguientes clases:

**-Tablero:** Esta clase está enfocada principalmente en la forma del tablero sus atributos son una matriz en este caso de 8\*8, filas y columnas.

Los métodos de **Tablero** son:

-mostrar\_tablero: Esta me permite ver como es el tablero en su estado actual.

-Tablero: El constructor de Tablero, me permite crear la matriz, luego dejar las filas y columnas en blanco y luego, llenar las fichas del centro del tablero.

-movimineto\_en\_tablero: Permite realizar cambio de fichas en el talbero.

- getMatriz: me permite traer la matriz para poder verificar algunas condiciones.

----como opcional tengo **copia\_de\_matriz,** **eliminar\_copia\_de\_matriz**,la cual me permitía crear una copia de la matriz pero por el momento no la he utilizado, puede que borre estos métodos.

**-Jugador:** Los principales atributos de esta clase son el puntaje, que muestra cuantas fichas tiene el jugador en el tablero actualmente.

El color, que está relacionado con la forma de la ficha, en este caso las blancas están relacionadas con un” \*” y las negras con un “- “.

El nombre, en el cual se guarda el respectivo nombre del jugador.

Los métodos de **Jugador** son:

-datos\_usuario: El cual nos permite establecer el tipo de ficha, el nombre del usuario y establece un puntaje en 0.

- getColor: Para obtener el color del usuario.

-getNombre: Para obtener el nombre del usuario.

-setPuntaje: Para establecer el nombre del usuario.

-getPuntaje: Para obtener el puntaje del usuario.

**-Juego:** Esta clase desarrolla como tal el juego, sus principales atributos son:

Dos instancias de la clase Jugador: **Jugador jugador1,** **Jugador jugador2**, las cuales contienen la información de la clase jugador ya mencionado anterior mente.

Una instancia de la clase Tablero: **Tablero tablero,** la cual me permite ver reflejado los distintos movimientos realizados por los usuarios.

Los métodos de **Juego** son:

-turno\_del\_jugador: En el cual se gestiona todo respecto al turno del jugador, ejemplo las restricciones para poner una ficha.

- Juego():El constructor, en el cual se pide el nombre a los jugadores.

Tengo planeado implementar dos clases más, una llamada menú, la cual gestiona un menú principal de juego y otra clase llamada archivo, la cual permite almacenar y acceder al historial de puntajes de los jugadores.

Mas adelante, continuare creando las normas para que el juego funcione correctamente.

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 26/10/2023, procedí a realizar algunas condiciones del juego entre ellas la de verificar si se puede colocar una ficha en el tablero, la mayoría de las condiciones quedaron llamadas, solo faltan algunos retoques entre ellos cuadrar las condiciones de las filas 0 y 7 y columnas 0 y 8. También falta cambiar las fichas de una línea cuando se cumple el sándwich.

Los métodos en los que trabaje fueron:

-**comprobar\_jugada\_valida:** La cual tiene algunas condiciones para poder poner las fichas.

-**verificacion\_linea\_sandwich:** La cual me permite hacer una verificación de las ocho formas en las que el sándwich, de puede dar.

Ambos métodos pertenecientes a la clase juego, en el próximo avance tengo planteado terminar las condiciones y efectuar el cambio visual de las fichas cuando ocurra el sándwich.

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 28/10/2023, realice algunas correcciones en el código, que no permitían que funcionara correctamente, también realice un cambio en el nombre de usuario el cual inicial mente estaba declarado como un char pero, lo cambie por un string para evitar problemas, otro problema que tenia era a la hora de ingresar un nombre con espacio, el cual me causaba errores, lo logre solucionar con un getline().

También modifique el código, para que el usuario ingrese números de 1 al 8 para las filas y letras mayúsculas de A al H para las columnas.

Logre plantear una posible solución para que me rellene las fichas las fichas contrarias del oponente cunado se cumple la condición de sándwich, por el momento solo lo aplique en las fichas de la posición izquierda.

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 31/10/2023, corregí y mejoré la funcionalidad de dos métodos de la **clase juego** las cuales fueron:

-**comprobar\_jugada\_valida:** primero agregue una condición extra la cual es**”** if((0 <= fila + i && fila + i < 8) && (0 <= columna + j && columna + j < 8))**”**, con esta condición me aseguro que todas las acciones de buscar una ficha siempre estén dentro del tablero, para evitar errores y estar siempre en el rango permitido.

También dentro de esta función declare un booleano llamado verificador igualado a falso, esta variable me permite recorrer todas las filas y columnas circundantes en de distancia 1 para verificar fichas opuestas para poder cumplir la condición de sándwich, si hay mínimo un sándwich, la variable verificador es igualada a true, gracias a esto la jugada es válida y puedo colocar la ficha, además me permite llenar múltiples filas y columnas en el caso que sea posible.

- **verificacion\_y\_llenado\_de\_linea\_sandwich:** Modifique el nombre de esta función, llamada anterior mente (**verificacion\_linea\_sandwich),** ya que esta ahora no solo cumple la función de verificar si se puede colocar la ficha, cuando se cumple que hay una ficha opuesta al rededor en el rango de 1 y además luego de esta fichas opuesta, le sigue una ficha del mismo jugador(en este caso el **sándwich),**si no que ahora también llenas las líneas , sean diagonales o rectas, con las fichas del jugador de turno, sustituyendo las fichas del otro jugador.

Luego de hacer estas modificaciones y actualizaciones se procedió, a realizar pruebas, hasta el momento, funciono correctamente y no encontré ningún error, se continuará haciendo pruebas.

Analizando el orden de reglas para el juego, falta un método que me permita verificar si hay al menos un lugar en el que pueda colocar una ficha valida, ya que si esto se cumple el juego puede continuar. Si, por el contrario, esto no es válido, específicamente para los 2 usuarios, el juego se termina y por defecto gana el que mayor puntaje tiene en ese momento.

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 1/11/2023,realice unas modificaciones a **comprobar\_jugada\_valida(Jugador& jugador,int fila,int columna,int seleccion)**, en la cual agregue una parámetro nuevo llamado **seleccion**, el cual me permite que la función **comprobar\_jugada\_valida(Jugador& jugador,int fila,int columna,int selección),**que también se le agrego el parámetro **seleccion**, tenga 2 funcionalidades distintas, dependiendo del valor de selección, este varia de 0 a 1, si el valor del parámetro de selección, es igual a 0, **comprobar\_jugada\_valida(),**, verifica si se cumpla la ley del **sándwich,** y también llena la línea correspondiente de fichas. Si por el contrario selección es igual a 1, **comprobar\_jugada\_valida(),**se encarga de solo de verificar si se cumple la ley del **sándwich,** esto lo hice con el propósito para poder crear un meto de la clase Juego, llamado:

**-bool verificacion\_proxima\_jugada(Jugador& jugador),**la cual me permite verificar si en el tablero , queda alguna jugada disponible, esto me permite crear una condición para poder terminar el juego.

También, corregí un error de la función:

**-verificacion\_y\_llenado\_de\_linea\_sandwich:**el cual me estaba llenando unas filas con fichas donde no debía , porque no estaba considerando los espacios en blanco. Este es un ejemplo de la condición que implemente:  
**if(tablero.getMatriz()[fila+contador\_filas][columna]==' ' && contador\_filas>1).**

En este momento, el juego está funcionando debidamente y finaliza correctamente, además cuando finaliza muestra quien gano con sus respectivos puntos. En el próximo avance, planeo trabajar en el historial de partidas, y guardar el archivo.  
adjunto imagen del juego:

Texto

Descripción generada automáticamente

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 2/11/2023, Se crearon varios métodos de la clase Juego, los cuales fueron:  
-**escribir\_archivo(string name, string info):**El cual me permite escribir un archivo, esto con el propósito de crear un .txt, llamado historial\_de\_partidas.txt, que me almacene el historial de partidas de los jugadores, el cual contiene el nombre del jugador, los puntos con los que gano y el jugador oponente, junto con su respectiva fecha de juego.

**.string leer\_archivo(string name):**Esta función me permite leer el archive para poder visualizar el historial de las partidas en consola.

Ambas funciones fueron extraídas de la practica 3.

También se creó un nuevo método de la clase Juego, llamado **menu\_principal()**, el cual me permite gestionar un menú con tres opciones distintas, las cuales son:  
**Ingrese 1 si quiere jugar**

**Ingrese 2 si quiere ver el historial de partidas**

**Ingrese 3 si quiere salir**

Esto permite tener una estructura del juego más organizada, Al crear el método **menu\_principal(),**tuve que realizar un cambio en la clase Juego, borrando el constructor por defecto, ya que este cumplía la función de asignarle los nombres y fichas a los jugadores.

Pero al crear el menú, esta se instanciaba antes del menú, por lo que cree un método nuevo llamado **definir\_datos\_jugador(),** que cumple exactamente lo que hacia el constructor de Juego, pero ahora solo se ejecuta cuando lo necesito.

Tuve un problema con la opción 1, ya que no podía poner el nombre al jugador 1, ya que en el búfer quedaba registrado un “enter”, lo logre solucionar buscando en foros y encontré **cin.ignore();** el cual elimina el buffer de entrada.

Con todas estas funcionalidades el juego ya está funcional y completo casi en su totalidad. Solo falta, organizar el código para que quede más organizado.

Texto

Descripción generada automáticamente

Anexo en el repositorio de GitHub, los avances de hoy.

-El día de hoy 6//11/2023, Organice el código, para que este estuviese mas organizado.

Para esto cree 3 **c++ Class,** con su respectivo, cpp y .h, los cuales son llamados juego, jugador y tablero. Esto se hace con el fin de tener una mejor presentación y desarrollo del código, se agregaron unas restricciones menores, se realizaron pruebas y luego se agregó el destructor en la clase tablero, **~Tablero();** para que me elimine la matriz, que tenia almacenada en la memoria dinámica.

También agregue comentarios en el código, para especificar y aclarar, el funcionamiento de las diferentes condiciones y lógicas del programa.

Texto

Descripción generada automáticamente

Anexo en el repositorio de GitHub, los avances de hoy.