UNIVERSIDAD DE ANTIOQUIA

Facultad de Ingeniería

Etapa 1: Análisis y Diseño

Informática II

Parcial II

Mario Esteban Estrada Gonzalez

CC: 1233191679

Andrés Felipe Agudelo Zapata

TI: 1017926361

Docente

Aníbal José Guerra Soler

2023-2

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

SEDE MEDELLÍN

Octubre 2023

Medellín, Colombia

1.

2. Contextualización:

El juego a realizar se desarrolla en una matriz cuadrada de tamaño inicial 8x8 (pero que puede cambiar si así se decide) en la que por turnos 2 jugadores deberán de colocar fichas en las casillas disponibles, el objetivo es que al finalizar el juego se debe de contar con mayor cantidad de fichas que el adversario.

Al inicio del juego se encuentran cuatro fichas en el centro de la matriz, dos negras y dos blancas, un jugador puede colocar una ficha en caso de que se pueda generar un encierro de tipo sándwich a una o más fichas enemigas, un encierro tipo sandwich significa encerrar una o más fichas del color contrario entre dos fichas de tu color, este encierro se puede dar en cualquiera de las diagonales, de forma horizontal o de forma vertical. Si se genera un encierro de tipo sándwich, todas las fichas atrapadas cambian al color del que las encerró.

Si un jugador no cuenta con ninguna posición para generar un encierro tipo sandwich, entonces se pierde el turno, el juego debe de finalizar si ninguno de los dos jugadores cuenta con movimientos posibles o si se llena la matriz y se debe de guardar el registro de los jugadores, quien gano y con qué puntaje.

3. Análisis:

El análisis del problema se generó con la visión de “divide y vencerás”, de manera que del problema principal se desglosan problemas más específicos que al darles solución encaminan al desarrollo final del proyecto, en este orden de ideas nacen las siguientes preguntas.

¿Qué estructura de datos se usará?

Dentro de las dinámicas del juego se puede observar que está cimentado en un tablero de 8x8 filas y columnas respectivamente, las cuales se pueden representar a través de una matriz de la misma dimensión, puesto que dentro de las limitaciones presentadas no se puede hacer uso de contenedores STL se opta por usar punteros dobles a arreglos de tamaño 8 donde cada posición de la matriz corresponde al espacio que una ficha puede ocupar, para facilitar el funcionamiento se tendrán tres tipos de identificadores que son:

COLOR FICHA	SÍMBOLO
Negro	-
Blanco	*
Vacío	o

¿Qué clases se crearán?

Para representar la lógica del juego se tienen 3 clases que el correcto funcionamiento, las clases son:

Jugador:
Estos objetos representan a cada uno de los jugadores que participan en la partida

Tablero:
Contiene las reglas y el funcionamiento general del juego, además de permitir la interacción de los jugadores.

Juego:
Permite el acceso al historial de las partidas, además de dar el inicio y el fin de las mismas.

¿Cómo funciona el juego?

El funcionamiento interno del tablero se lleva a cabo en 3 partes, la primera es la creación de la matriz con las 4 fichas iniciales como se ve en la siguiente figura.

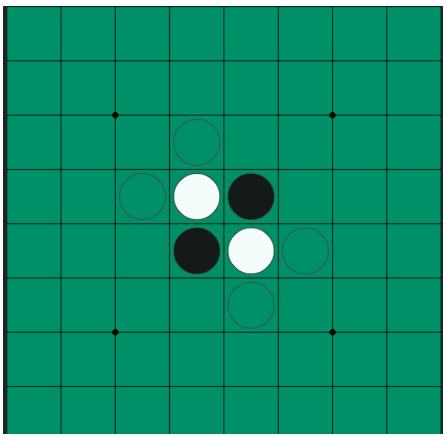

Fig 1

La segunda parte del funcionamiento requiere determinar cuáles son los movimientos posibles que el siguiente jugador puede realizar, para esto se crea la función Fichas_alredor() que determina si una posición vacía dentro del tablero, la cual será ocupada por una ficha de cierto color, está rodeada por fichas de color contrario.
A continuación a través de la función Sandwich_posible() se establece si al colocar una ficha en esa posición crea un efecto sandwich, finalmente con Insertar_Ficha() introduce una ficha en el tablero efectuando las validaciones y cambios de color respectivos

Como parte final, una vez todos los espacios del tablero hayan sido ocupados por fichas, se establecerá como finalizada la partida y se le otorgara a los jugadores la opción de volver a jugar o finalizar, sin importar cuál sea la respuesta dada, cada partida queda registrada en un archivo de texto externo al programa.

4. Diseño de la solución:

Clase Juego	**Clase Jugador**	**Clase Tablero**
Atributos: nombre, jugadores actuales, fecha y hora, puntaje, nombre, archivo, texto	**Atributos:** Turno, color, Nombre	**Atributos:** Matriz, Cantidad fichas negras, Cantidad fichas blancas, Cantidad espacios vacíos
Métodos: ● Leer archivo ● Imprimir registro ● Actualizar archivo ● Crear tablero ● Cerrar juego	**Métodos:** ● Acceder al turno	**Métodos:** ● Posibles movimientos ● Buscar alrededor ● Check movimiento Sandwich ● Insertar nueva ficha ● Validación posición ● Actualización de matriz ● Check fin del juego ● Perder turno

5. Principales Algoritmos Implementados

5.1 Adyacents celds:

Esta función se encarga de revisar las 8 celdas adyacentes a la celda inicializada en la función, e invocar la función sandwichCheck en caso de que alguna de estas tenga un carácter del otro usuario, ya que de lo contrario es imposible crear una intercesión tipo sandwich, la función sandwichCheck requiere el saber cuanto va a aumentar la x y la y, o lo que es lo mismo requiere el conocer el camino que recorre, y para obtener estos datos la función adyacent celds cuenta con 8 combinaciones diferentes de x y de y.

```
bool tablero::adyacentcelds(unsigned short fila, unsigned short columna, char turnoactual){
  char otroturn=otroturno(turnoactual);
  bool arriba,abajo,izquierda,derecha,diagonal1,diagonal2,diagonal3,diagonal4,flag;
  if(columna!=0){
    if(getvalue(fila,columna-1)==otroturn){
    arriba=sandwichCheck(fila,columna,0,-1,turnoactual);
    }else
      arriba=false;
  }else
    arriba=false;

  if (columna!=(columnas-1)){
    if(getvalue(fila,columna+1)==otroturn){
      abajo=sandwichCheck(fila,columna,0,+1,turnoactual);
    }else
      abajo=false;
  }
  else abajo=false;

  if(fila!=0){
    if(getvalue(fila-1,columna)==otroturn){
      izquierda=sandwichCheck(fila,columna,-1,0,turnoactual);
    }else
      izquierda=false;
  }
  else izquierda=false;

  if(fila!=filas-1){
    if(getvalue(fila+1,columna)==otroturn){
      derecha=sandwichCheck(fila,columna,1,0,turnoactual);}
    else
      derecha=false;
  }else derecha=false;

  if(columna!=columnas-1 && fila!=filas-1){
  if(getvalue(fila+1,columna+1)==otroturn){
    diagonal1=sandwichCheck(fila,columna,1,1,turnoactual);}
  else
    diagonal1=false;
```

```
}else diagonal1=false;

if(columna!=columnas-1 && fila!=0){
if(getvalue(fila-1,columna+1)==otroturn){
   diagonal2=sandwichCheck(fila,columna,-1,1,turnoactual);}
else
   diagonal2=false;
}else diagonal2=false;

if(columna!=0 && fila!=filas-1){
if(getvalue(fila+1,columna-1)==otroturn){
   diagonal3=sandwichCheck(fila,columna,1,-1,turnoactual);}
else diagonal3=false;
}else diagonal3=false;

if(columna!=0 && fila!=0){
if(getvalue(fila-1,columna-1)==otroturn){
   diagonal4=sandwichCheck(fila,columna,-1,-1,turnoactual);}
else diagonal4=false;
} else diagonal4=false;

if(arriba||abajo||derecha||izquierda||diagonal4||diagonal3||diagonal2||diagonal1){
flag=true;
}else
flag=false;
return flag;
}
```

5.2 sandwichCheck

La función snadwichCheck recorre la matriz en una dirección hasta que encuentre otro valor del usuario en cuyo turno nos encontramos o hasta que se encuentre con un espacio vacío o una pared de la matriz, en caso se encontrase el char del usuario se retorna un true si este no se encuentra se retorna un false.

```
bool tablero::sandwichCheck(unsigned short fila, unsigned short columna, short sumax, short
sumay,char micaracter){
   bool flag=checklimits(fila,columna,sumax,sumay);
   if(flag){
   if(checklimits2(fila,columna,sumax,sumay)&&matriz[fila+sumax][columna+sumay]=='o'){
      flag= false;
     }
     else
if(checklimits2(fila,columna,sumax,sumay)&&matriz[fila+sumax][columna+sumay]==micara
cter){
      flag=true;
     }
     else{
      flag=sandwichCheck(fila+sumax,columna+sumay,sumax,sumay,micaracter);
     }
   }
   return flag;
```

```
}
```

5.3 insert_piece

Esta función le pide al usuario una posible posición para ingresar la ficha, después de validar la entrada, la función y de saber si es un movimiento posible en una casilla vacía, busca en los 8 casos posibles de encierro de tipo sándwich y en caso de ser verdadero invoca la función change_color, esta función se invoca en cualquier encierro tipo sandwich y puede haber varios con la inserción de una sola ficha.

```
void tablero::insert_piece(player jugador){
   int z= 0;
   char colum_aux;
   short columna;
   int fila;
   while (z == 0){
      cout<<"Ingrese la  letra de la columna donde se ubicara la ficha"<<endl;
      cin >> colum_aux;
      columna=validacionentrada1(colum_aux);
      cout<<"ingerese el numero de la fila donde se ubicara la ficha "<<endl;
      cin>>fila;
      fila =validacion_entrada_fila(fila);
      fila--;
      char color = jugador.getpieza();
      if (columna != -1 && fila != -1){
         if (adyacentcelds(fila,columna,color)&&getvalue(fila,columna)!='0'){

            if (sandwichCheck(fila,columna, 0, -1,color)){//arriba{
               change_color(fila,columna, 0, -1,jugador);
            }
            if(sandwichCheck(fila,columna, 0, 1,color)){//abajo
               change_color(fila,columna, 0, 1,jugador);
            }
            if(sandwichCheck(fila,columna, -1, 0,color)){//izquierda
               change_color(fila,columna, -1, 0,jugador);
            }
            if(sandwichCheck(fila,columna, 1, 0,color)){//derecha
               change_color(fila,columna, 1, 0,jugador);
            }
            if(sandwichCheck(fila,columna, 1, 1,color)){//diagonal derecha abajo
               change_color(fila,columna, 1, 1,jugador);
            }
            if(sandwichCheck(fila,columna, 1, -1,color)){//diagonal derecha arriba
               change_color(fila,columna, 1, -1,jugador);
            }
            if (sandwichCheck(fila,columna, -1, -1,color)){//diagonal izquierda arriba
               change_color(fila,columna, -1, -1,jugador);
            }
            if(sandwichCheck(fila,columna, -1, 1,color)){//diagonal derecha abajo
               change_color(fila,columna, -1, 1,jugador);
```

```
                    }

                    z=1;
                }
            }
            else{
                cout<<"En la ubicacion ingresada no posible colocar una ficha"<<endl;
            }
        }
        cant_vacios--;
    }
```

5.4 Change_color:

La función changecolor recibe una posición de la matriz y el recorrido en el que se mueve a través de la matriz, además del valor de quien es el turno, después recorre la matriz y se va a cambiar el valor de todos los valores de la matriz que sean del valor del otro turno.

```
void tablero::change_color(unsigned short fila, unsigned short columna, short sumax, short
sumay,player jugador){
   bool flag=checklimits(fila,columna,sumax,sumay);
   unsigned short changes = 1;
   char color_opuesto;
   if (jugador.getpieza()== '*'){
      color_opuesto = '-';
   }
   else{
      color_opuesto= '*';
   }

   matriz[fila][columna]= jugador.getpieza();
   if(flag ){


      if(matriz[fila+sumax][columna+sumay]== color_opuesto){
         matriz[fila+sumax][columna+sumay]= jugador.getpieza();
         changes++;
         flag= true;
         change_color(fila+sumax,columna+sumay,sumax,sumay,jugador);
      }
      else if(matriz[fila+sumax][columna+sumay]==jugador.getpieza()){
         flag=false;
      }
      else {

      }
   }

}
```

6. Experiencia de aprendizaje

En el transcurso del diseño y la implementación se resaltan como primera discusión el hecho de establecer a la ficha como una instancia de la clase “ficha”, puesto que intuitivamente se cree que la ficha debe ser un objeto, sin embargo después de analizar la dinámica del juego notamos que el objeto ficha no tendría ningún atributo que la clase tablero ya presentara, además los métodos de cambiar color se podían hacer directamente en la matriz por tal motivo se implementó como un método del tablero ”change color”.

Después de la anterior discusión las demás clases inicialmente pensadas en el diseño fueron las que se implementaron, aunque se hicieron algunos cambios respecto a los atributos “Cant_blancas” y “Cant Negras” que inicialmente cada jugador tendria la cantidad de fichas de su color para finalmente saber quién ganó, no obstante para asignarle el valor a esos atributos era necesario utilizar métodos del tablero por lo cual se opto por dejarlo como atributos de la clase “tablero”.

Como conclusión se puede afirmar que lo primordial al momento de iniciar un nuevo proyecto es realizar el análisis respecto a las clases que se usarán, puesto que estas deben representar la realidad, hay que tener demasiado cuidado con no crear clases donde las instancias no requieran atributos y sus métodos pueden ser sustituidos por métodos de clases más generale, de la misma forma cabe recalcar como importante la comunicación entre los integrantes del proyecto puesto que pueden sobreescribir lineas de código por el uso Git.