Othello

alumne1, alumne2

Generated by Doxygen 1.8.13

Thu Jun 18 2020 17:10:57

Contents

Chapter 1

Othello

Descripción general

Se ha de diseñar e implementar el juego del Othello.

El Othello es un juego de estrategia de dos jugadores: negras y blancas. Se juega sobre un tablero monocolor de 64 casillas, de 8 por 8. Existen variantes con tamaños de tableros de 10 por 10, e incluso con tableros irregulares de 6 por 7 y 7 por 8.

Los jugadores disponen de 64 fichas bicolores, negras de un lado y blancas del otro. Por conveniencia, cada jugador tiene de antemano 32 fichas, pero no le pertenecen y debe cederlas a su adversario si a éste no le quedan.

El objetivo del Othello (o Reversi), es tener más fichas de nuestro color que el adversario al final de la partida.

Comienza el juego el jugador con las fichas negras. En su turno, cada jugador debe colocar una ficha de su color sobre una casilla vacía del tablero, adyacente a una ficha contraria. Además, debe rodear una o varias fichas contrarias entre la ficha que se coloca y otra ficha del mismo color ya colocada sobre el tablero (en cualquier dirección: horizontal, vertical o diagonal). A continuación voltea de su color las fichas que acaba de flanquear. Las fichas no se retiran del tablero ni se mueven de una casilla a otra.

Si en su turno de juego un jugador no puede colocar una ficha que voltee alguna ficha contraria según las reglas, este jugador debe pasar su turno al adversario. Asimismo, si es posible cualquier volteo de alguna ficha, no se puede pasar el turno y se debe jugar.

El final se produce cuando ningún jugador puede realizar más movimientos o cuando todas las casillas están ocupadas.

Para más información, consultar la documentación adjunta (ver Referencias).

Diseño de juego

Especificaciones adicionales

A la hora de diseñar el juego, únicamente consideraremos tableros regulares de, como máximo 10 por 10 casillas. Por defecto el tablero será estándard, es decir, de 8 por 8.

Cada jugador puede ser un usuario o la máquina, que juega de forma automática. Cuando se solicitan los datos de un jugador, se preguntará si es una persona o la máquina.

Se diseñará utilizando **diseño descendente**. Se podrán utilizar todos los procedimientos implementados anteriormente, así como las librerías estándar presentadas en la asignatura (ver Referencias).

La implementación debe seguir, en general, el estilo indicado en el documento $**$"C Style Guide", NASA, August 1994$**$. Especialmente, se deben seguir las indicaciones subrayadas.

Después de realizar una lectura de datos de teclado, se debe **vaciar el buffer de teclado** para evitar lecturas incorrectas de información.

Menú inicial

Al iniciar el juego se mostrará al usuario un menú inicial con las siguientes posibles opciones:

- `N` - Crear un nuevo juego. Identifica a los jugadores, inicializa los tableros y toda la infor mación necesaria para poder comenzar un nuevo juego (turnos, puntuaciones...). No juega, sólo prepara el juego.
- `C` - Carga de un fichero, si existe, toda la información del último juego guardado. Los juegos se guardan en un fichero de tipo binario.
- `J` - Juega. Esta opción sólo estará visible y se podrá seleccionar si el existe un juego no finalizado:
  - porque previamente se ha creado uno nuevo (`N`),
  - porque previamente se ha cargado un juego guardado (`C`), o
  - porque se quiere continuar con un juego pausado. Cuando se está jugando, después de cada jugada, se ofrecerá la posibilidad de:
  - `P` - pausar el juego y volver al menú inicial.
  - `C` - continuar el juego hasta el final sin volver a preguntar.
  - `S` - seguir con la siguiente jugada. Después de la misma volverá a preguntar.
- `G` - Guarda toda la información de un juego en un fichero **binario**.
  - sólo se puede guardar un juego creado que **no haya finalizado**.
  - sólo se guarda un único juego.
- `R`- Permite visualizar los records guardados en el fichero de records (de tipo **texto**). Se preguntará al usuario el número de récords que desea ver. Como máximo se podrán visualizar 10 récords, ordenados o no. Si el usuario indica el valor 0, se entenderá que desea visualizar todos los récords almacenados en el fichero, sin ningún tipo de orden.
- `S`- Salir. Finaliza el programa y devuelve el control al sistema operativo.

El programa controlará que las opciones que se muestren sean únicamente las que puede elegir el usuario en cada momento del juego. También controlará que la opción elegida sea válida.

Únicamente se saldrá del juego cuando se elija la opción ´S´ (Salir).

Juego de pruebas

Para cada procedimiento nuevo se deberá diseñar el juego de pruebas que valide su correcto funcionamiento de acuerdo con las especificaciones.

No es necesario incluir el juego de pruebas de todos aquellos procedimientos que se han validado con anterioridad o que forman parte de una librería estándar.

Para diseñar el juego de pruebas, se utilizarán pruebas unitarias siguiendo el estándard del DEIM (ver Referencia).

Documentación

Todos los procedimientos diseñados deberán estar documentados siguiendo los estándares de Doxygen y Markdown.

- `http://www.doxygen.com/`
- `https://www.markdownguide.org/getting-started`

[Ref]: Referencias

- [1] `https://es.wikipedia.org/wiki/Reversi`
- [2] Descubriendo el Othello (`https://tinyurl.com/DescubriendoOthello`)
- [3] ¿Qué es el Othello y cómo se juega? (`https://youtu.be/XHIDHOojUtE`)

Chapter 2

Diseño

Author

andrea.francoj@estudiants.urv.cat
paula.segala@estudiants.urv.cat

Diseño de los procedimientos para el juego del Othello.

A continuación se debes detallar el análisis y el diseño de los procedimientos implementados. Para cada procedimiento se indicará el nombre y, a continuación, se incluirá una breve descripción y el pseudocódigo del algortimo siguiendo las indicaciones de la asignatura.

Chapter 3

Test List

Module Disponibles

Tests del algoritmo lee_caracter_cadena()

Descripción	Entradas		Salidas Esperadas	OK?
	cadena	stdin	letra	
Prueba humo	sS	s	s	
	Hola	l	l	
	ttstst	ujt	t	

Inicializa una matriz

Tests del algoritmo tb_inicializa_matriz()

Descripción	Entradas			Salidas Esperadas	OK?
	nfilas	mcols	caracter	matriz	
Prueba humo	2	2	'a'	matriz 1	
	3	3	'm'	matriz 2	

Tests del algoritmo tb_busca_matriz() | Descripción | Entradas | | | | Salidas Esperadas |OK?| |----------—|:------—:|:---------—:|:------—:|:------—:|:-----:|:—:|:---------—:|—| | | matriz | nfilas | mcols | caracter| f | c | encontrado | | | Prueba humo | matriz 1 | 2 | 2 | '$' | 0 | 1 | cierto | | | | matriz 2 | 3 | 3 | 'm' | 2 | 3 | | |

Module S11

Test del procedimiento entero_aleatorio_acotado()

Test del procedimiento caracter_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	'b'	'd'	['b'..'d']	
Índices inversos (x10)	'd'	'b'	['b'..'d']	

Test del procedimiento siguiente_jugador()

Descripción	Entradas		Salidas Esperada	OK?
	jugador	num_jugadores	j. siguiente	

Descripción	Entradas		Salidas Esperada	OK?
Prueba humo	1	3	2	
	5	7	6	
	4	5	0	

Test del procedimiento ot_cambia_color()

Descripción	Entradas	Salidas Esperada	OK?
	color_ficha	resultado	
Prueba humo	FICHA_BLANCA	FICHA_NEGRA	
	FICHA_NEGRA	FICHA_BLANCA	
	't'	FICHA_VACIA	

Test del tb_dentro_limites()

Descripción	Entradas				Salidas Esperada	OK?
	f	c	nfilas	mcols	resultado	
Prueba humo	1	2	10	10	cierto	
	5	7	6	6	falso	
	3	3	5	5	cierto	
	4	4	2	2	falso	

Tests del algoritmo tb_busca_orientacion()

Descrip-ción	Entradas						Salidas Esper-adas				OK?
	matriz	nfilas	mcols	fila	col	caracter	orientacion	trobat	fila	col	
Prueba humo	matriz	3	4	0	0	'b'	SURO↩ ESTE	cierto	2	2	
Fuera límites∗	matriz1	3	3	2	0	'$'	OESTE	falso	-	-	
	matriz1	3	4	2	2	'$'	OESTE	cert	0	1	

Tests del algoritmo tb_orientaciones()

Descripción	Entradas						Salidas Esperadas		OK?
	matriz	nfilas	mcols	fila	col	caracter	orientaciones	resultado	
Prueba humo	sopa	5	4	2	1	'O'	NORESTE, SURESTE, SUR, -1	3	
Fuera límites (∗)	sopa	4	3	2	2	'O'	SURESTE, -1	1	
	sopa	3	3	0	0	'l'	OESTE, -1	1	

Module S12

Test del procedimiento fechas_iguales()

Descripción	Entradas		Salidas Esperada	OK?
	f1	f2	resultado	
Prueba humo	10, 3, 2020	10, 3, 2020	cierto	
	15, 3, 1994	17, 6, 1999	falso	

Compara fechas.

Test del procedimiento compara_fechas()

Descripción	Entradas		Salidas Esperada	OK?
	f1	f2	resultado	
Prueba humo	10, 3, 2020	5, 3, 2020	1	

Test del procedimiento ot_escribe_record()

Descripción	Entradas	Salidas Esperada	OK?
	registro	pantalla	
Prueba humo	reg1	1-feb-2020 'O' 38 "Anna"	
	reg2	17-dec-2017 '@' 40 "Maria"	

Test del procedimiento ot_escribe_records()

Descripción	Entradas		Salidas Esperada	OK?
	records	num_records	resultado	
Prueba humo	1	3	2	
	1	8	14	

Test del procedimiento escribe_fecha()

Descripción	Entradas	Salidas Esperada	OK?
	fecha	resultado	
Prueba humo	2-1-2020	1-ene-2020	
	17-6-2020	17-jun-2020	

Tablero del Othello.

Test del procedimiento ot_tablero_inicial()

Descripción	Entradas		Salidas Esperada	OK?
	tablero	dim	resultado	
Prueba humo	tablero	8	(imagen)	

Tests del algoritmo ot_lee_jugador()

Descripción	Entradas			Salidas Esperadas	OK?
	nombre	m/h	color	jugador	
Prueba humo	"Lluis"	H	FICHA_NEGRA		
	"maquina_BLANCO"	M	FICHA_BLANCO		

Tests del algoritmo ot_obtener_datos_jugador()

Descripción	Entradas	Salidas Esperadas						OK?
	jugador	nombre	H/M	ficha_color	fila	col	puntuacion	

Descripción	Entradas	Salidas Esperadas						OK?
Prueba humo	jugador1	"Jon"	H	FICHA_NEG↩ RA	6	2	30	

| |jugador2 | "maquina_BLANCO"| M | FICHA_BLANCA| 7 | 5 | 29 | || |jugador3 | "Andrea"| H | FICHA_BLANCA| 1 | 1 | 35 |

Module S13

Test del procedimiento ot_inicio()

Descripción	Entradas	Salidas Esperada	OK?
	op	resultado	
Prueba humo	'N' 'J' 'S'	ok	
Jugar sin crear	'J'	opción errónea	
Jugar sin crear	'N', 'J','S' ,'P','G'	partida guardada	

Test del procedimiento ot_guarda_juego()

Descripción	Entradas	Salidas Esperada	OK?
	tablero dim jugadores turno	resultado	
Prueba humo	juego recien iniciado	cierto	
Prueba humo	no se ha podido acceder al fichero	false	

Test del procedimiento ot_recupera_juego()

Descripción	Entradas		Salidas Esperada	OK?
	fichero	tablero dim jugadores turno	resultado	
Prueba humo	juego.dat	juego recien iniciado	cierto	
	juego.dat	no se ha encontrado el archivo	falso	
	juego.dat	fichero vacio	falso	

Ordena los records

Tests del algoritmo ot_ordena_records()

Descripción	Entradas		Salidas Esperadas	OK?
	records	dim	resulado	
Prueba humo	record_ot inicial	3	record_ot (ordenada)	
	record_ot inicial	3	record_ot (ordenada)	

Module S14

Test del procedimiento ot_decide_casilla_auto()

Descripción	Entradas		Salidas Esperada				OK?
	tablero	dim	f	c	jugador.ultima_tirada	resultado	
Prueba humo	tablero inicial	8	4	5	(4, 5)	cierto	
Prueba humo	tablero inicial	8	3	2	(3, 2)	cierto	

Test del procedimiento ot_decide_casilla_manual()

Descripción	Entradas		Salidas Esperada				OK?
	tablero	dim	f	c	jugador.ultima_tirada	resultado	
Prueba humo	tablero inicial	8	4	5	(4, 5)	cierto	
Prueba humo	tablero inicial	8	6, 3	1, 2	(3, 2)	cierto	

Test del procedimiento ot_crea_juego()

Descripción	Entradas		Salidas Esperada	OK?
	dim	datos jugadores	valores de salida	
Prueba humo	8	prueba1	datos de salida correctos	
Prueba humo	15	prueba1	datos de salida correctos/ dim = 8	

Global test_caracter_aleatorio_acotado (void)

Test del procedimiento caracter_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	'b'	'd'	['b'..'d']	

Global test_entero_aleatorio_acotado (void)

Test del procedimiento entero_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	1	3	[1..3]	

Global test_ot_cambia_color (void)

Test del procedimiento ot_cambia_color()

Descripción	Entradas	Salidas Esperada	OK?
	color_ficha	resultado	
Prueba humo	FICHA_BLANCA	FICHA_NEGRA	
Ficha negra	FICHA_NEGRA	FICHA_BLANCA	
Error	'x'	FICHA_VACIA	

Global test_siguiente_jugador (void)

Test del procedimiento siguiente_jugador()

Descripción	Entradas		Salidas Esperada	OK?
	jugador	num_jugadores	j. siguiente	
Prueba humo	1	3	2	
Da la vuelta	1	2	0	

Global test_tb_dentro_limites (void)

Test del tb_dentro_limites()

Descripción	Entradas				Salidas Esperada	OK?
	f	c	nfilas	mcols	resultado	
Prueba humo	1	2	10	10	cierto	

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Procedimientos disponibles

Procedimientos disponibles

vacia_buffer_teclado ()

Elimina los caracteres del buffer de teclado.

```
acció vacia_buffer_teclado () és
var
    ch: caràcter;
fvar
inicio
    $ Recorrer el buffer hasta llegar al final o encontrar el salto de línea (\n)
    llegir (ch);
    mentre (no farxiu()) i (ch <> '\n') fer
        llegir (ch);
    fmentre
facció
```

Lee un carácter

Diseña un procedimiento que lea un carácter de teclado. El carácter leído ha de pertenecer a una cadena que se pasa como parámetro, si no pertenece se volverá a pedir el carácter. La definició del procedimiento es la siguiente:

```
/* Lee un carácter de teclado. El carácter ha de pertenecer a una cadena.
 *
 * @param cadena (Ref: taula[] de caràcter) Cadena que contine los caracteres válidos.
 * @param texto (Ref: taula[] de caràcter) Texto informativo que se muestra al usuario.
 * @return (caràcter) Devuelve un carácter que se encuentra entre los de la cadena.
 */
función lee_caracter_cadena (
    /* Parámetro de entrada*/   cadena: taula[] de caràcter,
                                texto: taula[] de caràcter ) devuelve caràcter és;
```

Análisis

¿Qué sabes o necesitas saber para resolver el ejercicio? ¿Sería conveniente diseñar algún otro procedimiento adicional? Revisar el ejercicio 4.13 ii. Si utilizas un procedimiento que ya está probado, no es necesario volver a probarlo.

Diseño

Completa el algoritmo.

```
/* Lee un carácter de teclado. El carácter ha de pertenecer a una cadena.
 *
 * @param cadena (Ref: taula[] de caràcter) Cadena que contine los caracteres válidos.
 * @param texto (Ref: taula[] de caràcter) Texto informativo que se muestra al usuario.
 * @return (caràcter) Devuelve un carácter que se encuentra entre los de la cadena.
 */

función lee_caracter_cadena(
    /* Parámetro de entrada*/       cadena: taula[] de caràcter,
                                    texto: taula[] de caràcter ) devuelve caràcter és
var
    /* Definición de variables locales si las necesitas. */
    letra: carácter;        /* Tipo carácter */
    cont: enter;
    pertenece_cad: boolea;
fvar
inicio
    /* 1. Inicialización de variables. */
    cont := 0;
    pertenece_cad := false;
    /* 2. Completa el algoritmo */
    llegir(letrea);
    mentres (pertenece_cad <> true) fer
        si (cadena[cont] = letra) llavors
            pertenece_cad := true;
        sino
            si (cadena[cont] = '\0') llavors
            llegir(letra);
            cont := -1;
        fsi
        cont := cont + 1;
    fmentre

    /* Dato devuelto */
    retorna (letra);

ffunció
```

Juego de pruebas lee_caracter_cadena()

En este caso tenemos dos opciones: o el carácter pertenece, y acabamos o no pertenece. En este último caso podemos tener una secuencia de caracteres que no pertenezcan, pero el último debe pertenecer.

Añade como mínimo, una prueba al juego de pruebas.

Test Tests del algoritmo lee_caracter_cadena()

Descripción	Entradas		Salidas Esperadas	OK?
	cadena	stdin	letra	
Prueba humo	sS	s	s	
	Hola	l	l	
	ttstst	ujt	t	

Inicializa una matriz

Diseña un procedimiento que inicialice toda una matriz de caracteres con un carácter dado.

La definició del procedimiento es la siguiente:

```
/* Inicializa una matriz con un carácter.
```

```
*
* @param matriz (Ref: taula[][] de caràcter) Matriz a inicializar.
* @param nfilas (Valor: enter) Número de filas de la matriz (nfilas > 0).
* @param mcols (Valor: enter) Número de columans de la matriz (mcols > 0).
* @param caracter (Valor: caràcter) Caracter con el que se inicializa.
*/
acció tb_inicializa_matriz (
    /* Parámetros de salida */   var matriz: taula[][] de caràcter,
    /* Parámetros de entrada */ nfilas: enter, mcols: enter, caracter: caràcter) és;
```

Análisis

¿Qué sabes o necesitas saber para resolver el ejercicio?

Diseño

Completa el algoritmo.

```
/* Inicializa una matriz con un carácter.
*
* @param matriz (Ref: taula[][] de caràcter) Matriz a inicializar.
* @param nfilas (Valor: enter) Número de filas de la matriz (nfilas > 0).
* @param mcols (Valor: enter) Número de columans de la matriz (mcols > 0).
* @param caracter (Valor: caràcter) Caracter con el que se inicializa.
*/
acció tb_inicializa_matriz(
    /* Parámetros de salida */   var matriz: taula[][] de caràcter,
    /* Parámetros de entrada */ nfilas: enter, mcols: enter, caracter: caràcter) és
var
    /* Definición de variables locales si las necesitas. */
    contf:enter;
    contc: enter;

fvar
inicio  /* Código */
    contf:=0;
    contc:=0;

    mentres (nfilas <> contf)fer
        mentres(mcols <> contc) fer
            matriz[contf][contc] := caracter;
            contc := contc + 1;
        fmentres
    contf := contf + 1;
    fmentres

facció
```

Juego de pruebas tb_inicializa_matriz()

En este caso el juego de pruebas es muy simple, porque basta con probar que la matriz se inicializa bien, sin pasarnos de los límites. Podemos probar con una matriz cualquiera y con la matriz más pequeña 1x1.

Añade como mínimo, una prueba al juego de pruebas.

matriz 1	
a	a
a	a

Test Tests del algoritmo tb_inicializa_matriz()

Descripción	Entradas			Salidas Esperadas	OK?
	nfilas	mcols	caracter	matriz	

Descripción	Entradas			Salidas Esperadas	OK?
Prueba humo	2	2	'a'	matriz 1	
	3	3	'm'	matriz 2	

Busca en una matriz

Diseña un procedimiento busque un determinado carácter dentro de una matriz de caracteres. El procedimiento nos devolverá si lo ha encontrado o no, y en caso de que lo haya encontrado, la fila y la columna dónde se encuetra el carácter.

La definició del procedimiento es la siguiente:

```
/* Busca en una matriz un determinado carácter e
*  indica la fila y la columna donde se encuentra.
*
* @param matriz (Ref: taula[][] de caràcter) Matriz donde buscamos.
* @param nfilas (Valor: enter) Número de filas de la matriz (nfilas > 0).
* @param mcols (Valor: enter) Número de columans de la matriz (mcols > 0).
* @param f (Ref: enter) Fila donde se ha encontrado el caracter buscado.
* @param c (Ref: enter) Columna donde se ha encontrado el caracter buscado.
* @param caracter (Valor: caràcter) Caracter que se está buscando.
* @return (booleà) Retorna cierto si lo ha encontrado y falso en caso contrario.
*/
funció tb_busca_matriz (
    /* Parámetros de entrada */     matriz: taula[][] de caràcter,
                                        nfilas: enter, mcols: enter,
    /* Parámetros de salida */      var f: enter, var c: enter,
    /* Parámetros de entrada */     caracter: caràcter) retorna booleà és;
```

Análisis

¿Qué sabes o necesitas saber para resolver el ejercicio? ¿Como aplicarías los esquemas de búsqueda y recorrido al caso de las matrices?

Diseño

Completa el algoritmo.

```
/* Busca en una matriz un determinado carácter e
*  indica la fila y la columna donde se encuentra.
*
* @param matriz (Ref: taula[][] de caràcter) Matriz donde buscamos.
* @param nfilas (Valor: enter) Número de filas de la matriz (nfilas > 0).
* @param mcols (Valor: enter) Número de columans de la matriz (mcols > 0).
* @param f (Ref: enter) Fila donde se ha encontrado el caracter buscado.
* @param c (Ref: enter) Columna donde se ha encontrado el caracter buscado.
* @param caracter (Valor: caràcter) Caracter que se está buscando.
* @return (booleà) Retorna cierto si lo ha encontrado y falso en caso contrario.
*/
funció tb_busca_matriz(
    /* Parámetros de entrada */     matriz: taula[][] de caràcter,
                                        nfilas: enter, mcols: enter,
    /* Parámetros de salida */      var f: enter, var c: enter,
    /* Parámetros de entrada */     caracter: caràcter) retorna booleà és;
var
    /* Definición de variables locales si las necesitas. */
    trobat: boolea;
    trobat := fals;
    f := 0;
    c := 0;

fvar
inicio  /* Código */

    mentres (f <> nfilas) i (trobat <> cert) fer
        mentres (c <> mcols) i (trobat<> cert) fer
```

```
            si (matize [f][c] = caracter) llavors
                trobat :=  cert;
            sino
            c := c + 1;
            fsi
        fmentres
    f := f + 1;
    fmentres

ffunció
```

Juego de pruebas tb_busca_matriz()

En este caso se trata de un tipo de pruebas binario: lo encuenta o no lo encuentra. Por lo tanto, deberíamos plantear ambos casos. Puesto que es un caso particular de tratamiento de secuencias, deberíamos considerar el caso de que se encuentre en la primera posición de la matriz `matriz[0][0]` o en la última `matriz[nfilas-1][mcols-1]` Podemos probar con una matriz cualquiera y con la matriz más pequeña 1x1.

Añade como mínimo, una prueba al juego de pruebas.

matriz 1	
a	$
a	a

Test Tests del algoritmo tb_busca_matriz() | Descripción | Entradas | | | | Salidas Esperadas |OK?| |---------- ------— |:------—:|:---------—:|:------—:|:------—:|:-----:|:—:|:---------—:|—| | | matriz | nfilas | mcols | caracter| f | c | encontrado | | | Prueba humo | matriz 1 | 2 | 2 | '$' | 0 | 1 | cierto | | | | matriz 2 | 3 | 3 | 'm' | 2 | 3 | | |

6.2 Sesión 11

Functions

- int siguiente_jugador (int jugador, int num_jugadores)
- int entero_aleatorio_acotado (int min, int max)
- char caracter_aleatorio_acotado (char inf, char sup)
- char ot_cambia_color (char color_ficha)
- bool tb_dentro_limites (int f, int c, int nfilas, int mcols)
- bool tb_busca_orientacion (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int ∗f, int ∗c, int orientacion, char car)
- int tb_orientaciones (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int f, int c, char caracter, int orientaciones[ ])

6.2.1 Detailed Description

Entero aleatorio acotado.

Diseña un procedimiento que proporcione un número aleatorio, acotado entre dos valores, ambos incluidos↩ : [mínimo..máximo]. Los valores mínimo y máximo pueden estar desordenados, es decir, puede ocurrir que màximo $<$ mínimo. Se tendrá que comprobar en el procedimiento.

La descripción del procedimiento es la siguiente:

```
/* Determina un número aleatorio acotado [min..max].
 * Los valores mínimo y máximo pueden no estar ordenados.
 *
 * @param min (Valor: entero) Valor mínimo que puede tomar el número aleatorio.
 * @param max (Valor: entero) Valor máximo que puede tomar el aleatorio.
 * @return (entero) Devuelve un aleatorio perteneciente a [min..max].
 */
funció entero_aleatorio_acotado (
            /* Parámetros de entrada */ min: enter, max: enter) retorna enter és;
```

Análisis

¿Qué sabes a la hora de resolver el procedimento?

Para el problema necesitamos que el usuario no pase los numeros y comprobar que estan en oreden.Y utilizar la libreria srand() para buscar un numero aleatorio entre el min y max. El numero que lo devolveremos como resultado.

Diseño

Completa el algoritmo.

```
/* Determina un número aleatorio acotado [min..max].
 * Los valores mínimo y máximo pueden no estar ordenados.
 *
 * @param min (Valor: entero) Valor mínimo que puede tomar el número aleatorio.
 * @param max (Valor: entero) Valor máximo que puede tomar el aleatorio.
 * @return (entero) Devuelve un aleatorio perteneciente a [min..max].
 */
funció entero_aleatorio_acotado (
        /* Parámetros de entrada */ min: enter, max: enter) retorna enter és
        var
                    /* Definición de variables locales si las necesitas. */
                    num_ale:enter;
                    x: enter;
        fvar
        inicio  /* Código */
                   x := max;
                    si (max < min) llavors
                                max := min;
                                min := max;
                    fsi

                    srand(time(NULL);
                    num_ale := aleatorio %(max-min+1)+min;

                    retorna(num_ale);
        ffunció
```

Juego de pruebas entero_aleatorio_acotado()

Para validar las pruebas, utilizaremos la misma semilla, así nos aseguramos que la secuencia de aleatorios es siempre la misma: semilla (1);

Añade como mínimo, una prueba al juego de pruebas.

Test Test del procedimiento entero_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	1	3	[1..3]	
Índices inversos (x10)	3	1	[1..3]	
	4	2	[2..4]	

Carácter aleatorio acotado.

Diseña un procedimiento que proporcione un carácter aleatorio, acotado entre dos valores, ambos incluidos, siguiendo el orden de la tabla ASCII: [inferior..superior]. Los valores inferior y superior pueden estar desordenados, es decir, puede ocurrir que superior $<$ inferior. Se tendrá que comprobar en el procedimiento.

La descripción del procedimiento es la siguiente:

```
/* Determina un carácter aleatorio acotado [inf..sup]. Los valores inferiores y
 * superiores se determinan según el código ascii correspondiente.
 * Pueden no estar ordenados.
 *
 * @param inf (Valor: carácter) Carácter inferior que limita los carácteres válidos.
 * @param sup (Valor: carácter) Carácter superior que limita los carácteres válidos.
 * @return (carácter) Devuelve un aleatorio perteneciente a [inf..sup].
 */
funció caracter_aleatorio_acotado (
                      /* Parámetros de entrada */ inf: caràcter, sup: caràcter) retorna caràcter és;
```

Análisis

¿Qué sabes a la hora de resolver el procedimento?

Para resolver el problema el usuario nos pasara 2 caracteres. Comprobaremos cual es el max y el min, y buscaremos un numero aleatorio con la libreria srand() entre inf y sup, y devolveremos el resultado.

Diseño

Completa el algoritmo.

```
/* Determina un carácter aleatorio acotado [inf..sup]. Los valores inferiores y
* superiores se determinan según el código ascii correspondiente.
* Pueden no estar ordenados.
*
* @param inf (Valor: carácter) Carácter inferior que limita los carácteres válidos.
* @param sup (Valor: carácter) Carácter superior que limita los carácteres válidos.
* @return (carácter) Devuelve un aleatorio perteneciente a [inf..sup].
*/
funció caracter_aleatorio_acotado (
                        /* Parámetros de entrada */ inf: caràcter, sup: caràcter) retorna caràcter és
var
            /* Definición de variables locales si las necesitas. */
            lletra_ale:caracter;
            x: caracter;

fvar
inicio  /* Código */
          x := sup;
            si (inf>sup) llavors
                        sup := inf;
                        inf := x;
            fsi

            srand(time(NULL);
            lletra_ale:= aleatorio %(sup-inf+1)+inf;

            retorna(lletra_ale);

ffunció
```

Juego de pruebas caracter_aleatorio_acotado()

Para validar las pruebas, utilizaremos la misma semilla, así nos aseguramos que la secuencia de aleatorios es siempre la misma: semilla (1);

Añade como mínimo, una prueba al juego de pruebas.

Test Test del procedimiento caracter_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	'b'	'd'	['b'..'d']	
Índices inversos (x10)	'd'	'b'	['b'..'d']	

Siguiente jugador.

Diseña un procedimiento para pasar el turno al siguiente jugador. Los jugadores siempre se identificarán con un valor entero, empezando desde 0, y los turnos estarán ordenados de forma creciente.

Por ejemplo, si tenemos tres jugadores, estos se identificarán con los números: 0, 1 y 2. Si es el turno del jugador 1, el siguiente jugador será el 2. Pero si es el turno del jugador 2, el siguiente jugador será el 0.

La descripción del procedimiento es la siguiente:

```
/* Determina el jugador que tendrá el turno siguiente.
 *
 * @param jugador (Valor: entero) Identificador del jugador que tiene el turno actualmente.
 * @param num_jugadores (Valor: entero) Número total de jugadores del juego. Si el valor es inferior o
       igual a uno, no cambia el turno.
 * @return (entero) Devuelve el identificador del siguiente jugador.
 */
funció siguiente_jugador (
                        /* Parámetros de entrada */   jugador: enter, num_jugadores: enter)

               retorna enter és;
```

Análisis

¿Qué sabes a la hora de resolver el procedimento?

Para diseñar el procedimiento la funcion nos dara numero de jugadores y quien es el turno. Con condicional entaremos en el turno del jugador y procederemos a pasar al siguiente hasta el maximo numero de concursantes donde volvera al inicio.

Diseño

Completa el algoritmo.

```
/* Determina el jugador que tendrá el turno siguiente.
*
* @param jugador (Valor: entero) Identificador del jugador que tiene el turno actualmente.
* @param num_jugadores (Valor: entero) Número total de jugadores del juego. Si el valor es inferior o igual
       a uno, no cambia el turno.
* @return (entero) Devuelve el identificador del siguiente jugador.
*/
funció siguiente_jugador (
                        /* Parámetros de entrada */   jugador: enter, num_jugadores: enter)

               retorna enter és
var
            /* Definición de variables locales si las necesitas. */

fvar
inicio  /* Código */

            si (num_jugadores <= 1) llavors
                        jugador := jugador;

            sino si(num_jugadores > (jugador+1)) llavors
                        jugador:= jugador+1;
            sino
                        jugador := num_jugadores - (jugador+1);
            fsi

            retorna(jugador);

ffunció
```

Juego de pruebas siguiente_jugador()

En este caso es interesante tratar los límites, ¿qué ocurre si es el primer jugador o el último jugador? ¿Y si sólo hay un jugador?

Añade como mínimo, una prueba al juego de pruebas.

Test Test del procedimiento siguiente_jugador()

Descripción	Entradas		Salidas Esperada	OK?
	jugador	num_jugadores	j. siguiente	
Prueba humo	1	3	2	
	5	7	6	
	4	5	0	

Cambia de color.

Este es un procedimiento que nos proporcione el color contrario al que se pasa por parámetro, es decir, si pasamos la ficha negra, nos devolverá la ficha blanca, y si pasa la ficha negra, nos devolverá la blanca. Si se produce un error, devolverá la ficha vacía.

La descripción del procedimiento es la siguiente:

```
/* @enun fichas_t.
* Determina los símbolos para representar las fichas en el tablero.
*/
constants
            FICHA_BLANCA <- 'O';
            FICHA_NEGRA <- '@';
            FICHA_VACIA <- ' ';
fconstants

/* Cambia el color de la ficha: si la ficha era de color era negro,
 * nos devuelve el color blanco (FICHA_BLANCA) y si era blanco, nos devuelve negro (FICHA_NEGRA).
 * Si el color no es correcto, nos devolverá la FICHA_VACIA.
 *
 * @param color_ficha (Valor: caràcter) Color actual de la ficha.
 * @return (caràcter) Devuelve el color contrario al recibido como parámetro o FICHA_VACIA en caso de
       error.
*/
funció ot_cambia_color (
                               /* Parámetros de entrada */   color_ficha: caràcter) retorna caràcter és;
```

Análisis

¿Qué sabes a la hora de resolver el procedimento?

Para resolver utilizaremos un menu de opcio que siempre nos devuelva lo contario. Y un condicional para cuando se introduzca un caracter erroneo.

Diseño

Completa el algoritmo.

```
/* @enun fichas_t.
* Determina los símbolos para representar las fichas en el tablero.
*/
constants
            FICHA_BLANCA <- 'O';
            FICHA_NEGRA <- '@';
            FICHA_VACIA <- ' ';
fconstants

/* Cambia el color de la ficha: si la ficha era de color era negro,
* nos devuelve el color blanco (FICHA_BLANCA) y si era blanco, nos devuelve negro (FICHA_NEGRA).
* Si el color no es correcto, nos devolverá la FICHA_VACIA.
*
* @param color_ficha (Valor: caràcter) Color actual de la ficha.
* @return (caràcter) Devuelve el color contrario al recibido como parámetro o
*                                                    FICHA_VACIA en caso de error.
*/
funció ot_cambia_color (
```

```
                                /* Parámetros de entrada */   color_ficha: caràcter) retorna caràcter és;
var
            /* Definición de variables locales si las necesitas. */
            resultado: caracter;
fvar
inicio  /* Código */
            resultado := FICHA_VACIA;

          si (resultado = FICHA_BLANCA) llavors
            resultado := FICHA_NEGRA
          fsi

          si (resultado = FICHA_NEGRA) llavors
            resultado := FICHA_BLANCA;
          fsi

            retona(resultado);

ffunció
```

Juego de pruebas ot_cambia_color()

Es un procedimiento sencillo de probar, ya que sólo hay dos posibilidades.

Añade como mínimo, una prueba al juego de pruebas.

Test Test del procedimiento ot_cambia_color()

Descripción	Entradas	Salidas Esperada	OK?
	color_ficha	resultado	
Prueba humo	FICHA_BLANCA	FICHA_NEGRA	
	FICHA_NEGRA	FICHA_BLANCA	
	't'	FICHA_VACIA	

Dentro de los límites.

Diseña un procedimiento que compruebe si las coordenadas de una determinada casilla de la matriz (f, c), están dentro de los límites de la matriz considerando todos los lados de la matriz. Si están dentro de los límites, nos devolverá cierto, y en caso contrario, nos devolverá falso.

La descripción del procedimiento es la siguiente:

```
/* Determinan si una fila y columna dada están dentro de los límites de una matriz [nfilas][mcols]
 *
 * @param f (Valor: entero) Número de la fila.
 * @param c (Valor: entero) Número de la columna.
 * @param nfilas (Valor: entero) Número máximo de filas de la matriz.
 * @param mcols (Valor: entero) Número máximo de columnas de la matriz.
 * @return (booleà) Devuelve cierto si la fila pertenece a [0..nfilas) y
 *                              columna pertenece a [0..mcols) y falso en caso contrario.
*/
funció tb_dentro_limites (
            /* Parámetros de entrada */            f: enter, c: enter, nfilas, mcols)

               retorna booleà és;
```

Análisis

En este caso intentamos comprobar que la fila y columna que se pasan por parámetros estén dentro de los límites de la matriz. Por tanto, deberemos comprobar que no se salga por ninguno de los costados: izquierda el numero de filas debe ser igual o inferior al dado por el usuario. decendiente el numero de columnas debe ser igual o inferior al dado por el usario.

Diseño

Completa el algoritmo.

```
/* Determinan si una fila y columna dada están dentro de los límites de una matriz [nfilas][mcols]
*
* @param f (Valor: entero) Número de la fila.
* @param c (Valor: entero) Número de la columna.
* @param nfilas (Valor: entero) Número máximo de filas de la matriz.
* @param mcols (Valor: entero) Número máximo de columnas de la matriz.
* @return (booleà) Devuelve cierto si la fila pertenece a [0..nfilas) y
*                              columna pertenece a [0..mcols) y falso en caso contrario.
*/
funció tb_dentro_limites (
                              /* Parámetros de entrada */            f: enter, c: enter, nfilas, mcols)

                                 retorna booleà és
var
            /* Definición de variables locales si las necesitas. */
            pertenece: boolea;
fvar
inicio  /* Código */
            pertenece:= fals;

            si (((nfilas>f)i(f>=0))i((mcols>c)i(c>=0))) llavors
                        pertenece := cierto;
            fsi

            retona(pertenece);
ffunció
```

Juego de pruebas tb_dentro_limites()

Puesto que el resultado correcto depende de varias condiciones, debemos comprobar, por lo menos, que ocurre si se da cada una de las condiciones.

Añade como mínimo, una prueba al juego de pruebas.

Test Test del tb_dentro_limites()

Descripción	Entradas				Salidas Esperada	OK?
	f	c	nfilas	mcols	resultado	
Prueba humo	1	2	10	10	cierto	
	5	7	6	6	falso	
	3	3	5	5	cierto	
	4	4	2	2	falso	

Busca en la matriz siguiendo la orientación.

En el ejercicio 3.41 de la colección de problemas se propone un sistema similar a los puntos cardinales para examinar matrices.

En el **ejercicio 6.43 iv** de la colección de problemas, se propone buscar un determinado carácter siguiendo una orientación específica.

Diseña un procedimiento genérico que, a partir de una determinada posición inicial (fila, col), y una orientación concreta (NORTE, SUR...), busque un determinado carácter y nos diga si se ha encontrado o no. En el caso de que encuentre el carácter, en los parámetros fila y col, nos devolverá la casilla en la que se ha encontrado.

Notas:

- La casilla inicial (fila, col) no se evalúa.
- Cuidado con los límites de la matriz.
- Podéis utilizar cualquier procedimiento visto anteriormente, en éste o en cualquier otro laboratorio.

La descripción del procedimiento es la siguiente:

```
/*  MCOLS_MAX
Numero máximo de columnas de las matrices.
*/
constants MCOLS_MAX <- 10;  fconst

/* Busca un carácter en una matriz de nfilas x mcols, a partir de una determinada posicion (fila,col).
 * La busqueda se realiza en una determinada orientacion.
 * No se evalua la casilla inicial (fila, col)
 * El procedimiento devuelve si existe o no. En caso de que existe,
 * la casilla donde se encuentra el elemento se devuelve a partir de los mismos parametros de entrada fila
       y col.
 *
 * @param matriz (Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
 * @param nfilas (Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX i > 0).
 * @param mcols (Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX i > 0).
 * @param fila (Ref: entero) Fila inicial y, fila en la que se encuentra.
 * @param col (Ref: entero) Columna inicial y, columna en la que se encuentra.
 * @param car (Valor: char) Carácter que se desea buscar.
 * @param orientacion (Valor: enter) Orientación en la que se va a buscar.
 * @return (booleà) Devuelve cierto si se ha encontrado el carácter y
 *                                                  falso en caso contrario.
*/
funció tb_busca_orientacion(
                        /* Parámetros de entrada */             matriz: taula [][MCOLS_MAX] de caràcter,

                           nfilas: enter, mcols: enter,
                        /* Parámetros de entrada/salida */  var fila: enter, var col: enter,
                        /* Parámetros de entrada */             car: caràcter, orientacion: enter) retorna
       booleà és;
```

Análisis

¿Qué sabes o necesitas saber para resolver el ejercicio? ¿ Qué procedimientos anteriores pueden resultarte útiles?

Para hacer este procedimiento necesitamos recorrer la matriz según los parámetros que establece el usuario y comprobar si estos se adecuan al los interpuestos por el programa. Si no es así nos dará falso. En caso contrario se recorrerá a partir de los puntos que se nos haya indicado y de los límites que haya establecido el usuario y mostraremos si se ha encontrado el carácter introducido, donde y la orientación que se ha inicializado.

Diseño

Completa el algoritmo.

```
/* Busca un carácter en una matriz de nfilas x mcols, a partir de una determinada posicion (fila,col).
* La busqueda se realiza en una determinada orientacion.
* No se evalua la casilla inicial (fila, col)
* El procedimiento devuelve si existe o no. En caso de que existe,
* la casilla donde se encuentra el elemento se devuelve a partir de los mismos parametros de entrada fila y
       col.
*
* @param matriz (Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
* @param nfilas (Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX i > 0).
* @param mcols (Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX i > 0).
* @param fila (Ref: entero) Fila inicial y, fila en la que se encuentra.
* @param col (Ref: entero) Columna inicial y, columna en la que se encuentra.
* @param car (Valor: char) Carácter que se desea buscar.
* @param orientacion (Valor: enter) Orientación en la que se va a buscar.
* @return (booleà) Devuelve cierto si se ha encontrado el carácter y
*                                                  falso en caso contrario.
*/
```

```
funció tb_busca_orientacion(
                        /* Parámetros de entrada */            matriz: taula [][MCOLS_MAX] de caràcter,

                           nfilas: enter, mcols: enter,
                        /* Parámetros de entrada/salida */  var fila: enter, var col: enter,
                        /* Parámetros de entrada */            car: caràcter, orientacion: enter) retorna
       booleà és
    var
            /* Definición de variables locales si las necesitas. */
            trobat: boolea;
            rang:boolea;
            int incc, incf, f, c: entero;

    fvar
    inicio  /* Código */
      f := fila;
      c := col;
      trobat  := false;

      tb_calcula_incrementos(orientacion, incf,incc);

      rang = tb_dentro_limites(f, c, nfilas, mcols);

    mentres ((rang=cierto)i(trobat=falso)) fer
       f=f+incf;
       c=c+incc;
       rang = tb_dentro_limites(f, c, nfilas, mcols);
       si(rang = cierto)
           trobat := (matriz[f][c] = caracter);
    fsi
    fmentres

    retorna (trobat);
ffunció
```

Juego de pruebas tb_busca_orientacion()

En este caso, como se trata de buscar, siempre tendremos que probar que ocurre cuando se existe o cuando no existe el elemento buscado. Por otro lado, cuando se trabaja con matrices, se ha de validar que no se accede fuera de la matriz, por ejemplo, buscando en una matriz más pequeña que la definida originalmente.

También puede resultar interesante evaluar qué ocurre si el carácter buscado está al principio (casilla siguiente a la dada) o al final (en el límite de la matriz), que serían los casos equivalentes a buscar al principio de una secuencia y al final.

Añade como mínimo, una prueba al juego de pruebas.

matriz 1			
a	a	a	a
a	a	a	a
a	a	b	$

Test Tests del algoritmo tb_busca_orientacion()

Descripción	**Entradas**						**Salidas Esperadas**				**OK?**
	matriz	nfilas	mcols	fila	col	caracter	orientacion	trobat	fila	col	
Prueba humo	matriz	3	4	0	0	'b'	SURO↩ ESTE	cierto	2	2	
Fuera límites∗	matriz1	3	3	2	0	'$'	OESTE	falso	-	-	
	matriz1	3	4	2	2	'$'	OESTE	cert	0	1	

(∗) En este caso le pasamos al procedimiento que la matriz tiene una columna menos, por lo tanto, no debería acceder a la columna 3, y, por lo tanto, no debería encontrar el carácter. Si lo encuentra, es que accedemos fuera de los límites de la matriz.∗∗

Busca posibles orientaciones.

En el **ejercicio 6.43 iii** de la colección de problemas, se propone diseñar un algoritmo que, a partir de una casilla (fila, col), nos diga el número total a veces que se ha encontrado el carácter buscado y todas las orientaciones donde se encuentra (almacenadas a una tabla). Por ejemplo en la sopa de letras siguiente, si buscamos la letra 'O', a partir de la casilla (2, 0), nos dará tres posibles orientaciones: NORTE, SUR y SUDOESTE. Utilizaremos el valor -1 como centinela en la tabla para indicar que no hay más orientaciones.

La tabla que se pasa siempre tendrá espacio suficiente para guardar todas las orientaciones posibles y el centinela (-1).

Notas:

- La casilla inicial (fila, col) no se evalúa.
- Cuidado con los límites de la matriz.
- Podéis utilizar cualquier procedimiento visto anteriormente, en éste o en cualquier otro laboratorio.

La descripción del procedimiento es la siguiente:

```
/*  MCOLS_MAX
Numero máximo de columnas de las matrices.
*/
constants MCOLS_MAX <.- 10;  fconst

/* A partir de una posición de la matriz (fila y columna determinadas),
 * crea una tabla con todas las orientaciones en las que la casilla adyacente
 * contiene el carácter que se pasa como parámetro.
 * Devuelve el número de veces que se ha encontrado el carácter. La casilla del centro no se evalúa.
 *
 * @param matriz (Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
 * @param nfilas (Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX i > 0).
 * @param mcols (Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX i > 0).
 * @param fila (Valor: entero) Fila central a partir de la cual buscamos.
 * @param col (Valor: entero) Columna central a partir de la cual buscamos.
 * @param car (Valor: char) Carácter que se desea buscar.
 * @param orientaciones (Ref: taula[] de enter) Orientación en la que se va a buscar.
 * @return (enter) Devuelve el número de veces que se ha encontrado el carácter
 *                              en la casillas vecinas.
 */

funció tb_orientaciones (
                        /* Parámetros de entrada */             matriz: taula [][MCOLS_MAX] de caràcter,

                           nfilas: enter, mcols: enter,

                           fila: enter, col: enter,

                           car: caràcter,
                        /* Parámetros de salida */                  orientaciones: taula[] d'enter) retorna
       enter és;
```

Análisis

En este caso hay que presetar especial cuidado con los límites de la tabla, ya que no siempre una casilla está rodeada de otras, por ejemplo, ¿qué sucede si buscamos a partir de la casilla (0,0)?

¿Qué sabes o necesitas saber para resolver el ejercicio? ¿ Qué procedimientos anteriores pueden resultarte útiles?

Para encontrar las letras que el usuario pide necesitamos hacer un recorrido de toda la tabla a partir del punto que nos habra dado, comprobamos si esta en los limites. A partir de esto buscamos la orientacion a los lados de la letra con una funcion. Se ira guardando hasta que se termine el bucle y añadiremos el santinella. Por ultimo nos retornara le numero de letras encontrado.

Diseño

Completa el algoritmo.

```
/*  MCOLS_MAX
Numero máximo de columnas de las matrices.
*/
constants MCOLS_MAX <.- 10;  fconst

/* A partir de una posición de la matriz (fila y columna determinadas),
* crea una tabla con todas las orientaciones en las que la casilla adyacente
* contiene el carácter que se pasa como parámetro.
* Devuelve el número de veces que se ha encontrado el carácter. La casilla del centro no se evalúa.
*
* @param matriz (Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
* @param nfilas (Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX i > 0).
* @param mcols (Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX i > 0).
* @param fila (Valor: entero) Fila central a partir de la cual buscamos.
* @param col (Valor: entero) Columna central a partir de la cual buscamos.
* @param car (Valor: char) Carácter que se desea buscar.
* @param orientaciones (Ref: taula[] de enter) Orientación en la que se va a buscar.
* @return (enter) Devuelve el número de veces que se ha encontrado el carácter
*                              en la casillas vecinas.
*/

funció tb_orientaciones (
                        /* Parámetros de entrada */             matriz: taula [][MCOLS_MAX] de caràcter,

                           nfilas: enter, mcols: enter,

                           fila: enter, col: enter,

                           car: caràcter,
                        /* Parámetros de salida */                  orientaciones: taula[] d'enter) retorna
       enter és;
var
            /* Definición de variables locales si las necesitas. */
             f, c, orientacion, i: enter;
fvar
inicio  /* Código */

i = 0;
orientacion = 0;


/* Añadir el centinla */
  per(f=fila-1;f<=fila+1;f++) fer

    per(c=col-1;c<=col+1;c++) fer
      si(tb_dentro_limites(f, c, nfilas, mcols)) llavors
          si((matriz[f][c]=caracter)i((f<>fila)o(c<>col))) llavors
              orientaciones[i]:=orientacion;
              i := i +1;
          fsi
      fsi
      orientacion := orientacion + 1;
  fper
fper

orientaciones[i] := -1;

retorna (i);
ffunció
```

Juego de pruebas tb_orientaciones()

En este caso, como se trata de buscar, siempre tendremos que probar que ocurre cuando se existe o cuando no existe el elemento buscado. Por otro lado, cuando se trabaja con matrices, se ha de validar que no se accede fuera de la matriz, sobre todo en el caso de buscar a partir de posiciones situadas en los extremos de la matriz.

También puede resultar interesante evaluar qué ocurre si el carácter buscado está al principio (casilla siguiente a la dada) o al final (en el límite de la matriz), que serían los casos equivalentes a buscar al principio de una secuencia y al final.

Añade como mínimo, una prueba al juego de pruebas.

sopa			
L	A	S	U
O	Z	I	M
S	T	D	O
O	O	F	R
I	A	A	N

Test Tests del algoritmo tb_orientaciones()

Descripción	Entradas						Salidas Esperadas		OK?
	matriz	nfilas	mcols	fila	col	caracter	orientaciones	resultado	
Prueba humo	sopa	5	4	2	1	'O'	NORESTE, SURESTE, SUR, -1	3	
Fuera límites (∗)	sopa	4	3	2	2	'O'	SURESTE, -1	1	
	sopa	3	3	0	0	'I'	OESTE, -1	1	

6.2.2 Function Documentation

6.2.2.1 caracter_aleatorio_acotado()

```
char caracter_aleatorio_acotado (
            char inf,
            char sup )
```

Determina un carácter aleatorio acotado [inf..sup]. Los valores inferiores y superiores se determinan según el código ascii correspondiente. Pueden no estar ordenados.

Parameters

inf	(Valor: carácter) Carácter inferior que limita los carácteres válidos.
sup	(Valor: carácter) Carácter superior que limita los carácteres válidos.

Returns

(carácter) Devuelve un aleatorio perteneciente a [inf..sup].

```
96 {
97     /* Variables locales */
98     char letra;
99     char x;
100
101     /* La semilla se ha generado en el principal!!!
102     no llamar a srand()!
103     */
104
105     /* Código */
106     x = sup;
107      if(sup<inf) {
108         sup = inf;
```

```
109         inf = x;
110      }
111         letra = rand() % (sup-inf + 1) + inf;
112
113      return (letra);        /* Dato devuelto */
114 }
```

6.2.2.2 entero_aleatorio_acotado()

```
int entero_aleatorio_acotado (
              int min,
              int max )
```

Determina un número aleatorio acotado [min..max]. Los valores mínimo y máximo pueden no estar ordenados.

Parameters

min	(Valor: entero) Valor mínimo que puede tomar el número aleatorio.
max	(Valor: entero) Valor máximo que puede tomar el aleatorio.

Returns

(entero) Devuelve un aleatorio perteneciente a [min..max].

```
67 {
68     /* Variables locales */
69     int numero;
70     int x;
71
72     /* La semilla se ha generado en el principal!!!
73     no llamar a srand()!
74     */
75
76     /* Código */
77     x = max;
78     if(max<min) {
79         max = min;
80         min = x;
81     }
82     numero = rand() % (max-min + 1) + min;
83
84     return numero;        /* Dato devuelto */
85 }
```

6.2.2.3 ot_cambia_color()

```
char ot_cambia_color (
              char color_ficha )
```

Cambia el color de la ficha: si la ficha era de color era negro, nos devuelve el color blanco (FICHA_BLANCA) y si era blanco, nos devuelve negro (FICHA_NEGRA). Si el color no es correcto, nos devolverá la FICHA_VACIA.

Parameters

color_ficha	(Valor: caràcter) Color actual de la ficha.

Returns

(caràcter) Devuelve el color contrario al recibido como parámetro o FICHA_VACIA en caso de error.

```
26 {
27     /* Variables locales */
28     char ficha;
29     ficha = FICHA_VACIA;
30 
31     /* Codigo */
32     if (color_ficha == FICHA_NEGRA){
33         ficha = FICHA_BLANCA;
34     }
35      if (color_ficha == FICHA_BLANCA){
36         ficha = FICHA_NEGRA;
37     }
38 
39     return ficha;
40 }
```

6.2.2.4 siguiente_jugador()

```
int siguiente_jugador (
              int jugador,
              int num_jugadores )
```

Determina el jugador que tendrá el turno siguiente.

Parameters

jugador	(Valor: entero) Identificador del jugador que tiene el turno actualmente.
num_jugadores	(Valor: entero) Número total de jugadores del juego. Si el valor es inferior o igual a uno, no cambia el turno.

Returns

(entero) Devuelve el identificador del siguiente jugador.

```
42 {
43     int siguiente;
44 
45     /* Escribe la expresión correspondiente */
46     if (num_jugadores == 1){
47         siguiente = jugador;
48     }
49     else if(num_jugadores > (jugador+1)){
50         siguiente = jugador + 1;
51     }
52     else{
53         siguiente = num_jugadores - (jugador+1);
54     }
55 
56     return siguiente;      /* Datos de salida */
57 }
```

6.2.2.5 tb_busca_orientacion()

```
bool tb_busca_orientacion (
              char matriz[][MCOLS_MAX],
              int nfilas,
              int mcols,
              int * f,
              int * c,
              int orientacion,
              char car )
```

Busca un carácter en una matriz de nfilas x mcols, a partir de una determinada posicion (fila,col). La busqueda se realiza en una determinada orientacion. No se evalua la casilla inicial (fila, col) El procedimiento devuelve si existe o no. En caso de que existe, la casilla donde se encuentra el elemento se devuelve a partir de los mismos parametros de entrada fila y col. 0:NOROESTE, 1:NORTE, 2:NORESTE, 3:OESTE, 4:CENTRO, 5:ESTE, 6:SUROESTE, 7:SUR, 8:SURESTE

Parameters

matriz	(Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
nfilas	(Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX i > 0).
mcols	(Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX i > 0).
f	(Ref: entero) Fila inicial y, fila en la que se encuentra.
c	(Ref: entero) Columna inicial y, columna en la que se encuentra.
orientacion	(Valor: enter) Orientación en la que se va a buscar.
car	(Valor: char) Carácter que se desea buscar.

Returns

(booleà) Devuelve cierto si se ha encontrado el carácter y falso en caso contrario.

```
57 {
58     /* Variables locales */
59     bool iguales,rang;           /* Variable booleana */
60     int incc, incf,i,j;
61
62     /* Inicializa las variables */
63     i = *f;
64     j = *c;
65     /* Completa el código: búsqueda o recorrido? */
66     iguales = false;
67
68     tb_calcula_incrementos(orientacion, &incf,&incc);
69
70     rang = tb_dentro_limites(i, j, nfilas, mcols);
71
72     while ((rang==true)&&(iguales==false))
73     {
74        i=i+incf;
75        j=j+incc;
76
77        rang = tb_dentro_limites(i, j, nfilas, mcols);
78
79        if((rang==true)&&(matriz[i][j] == car)){
80
81            iguales = true;
82
83            *f=i;
84            *c=j;
85        }
86     }
87
88     /* Devuelve el resultado */
89     return iguales;
90 }
```

6.2.2.6 tb_dentro_limites()

```
bool tb_dentro_limites (
            int f,
            int c,
            int nfilas,
            int mcols )
```

Determinan si una fila y columna dada están dentro de los límites de una matriz [nfilas][mcols]

Parameters

f	(Valor: entero) Número de la fila.
c	(Valor: entero) Número de la columna.
nfilas	(Valor: entero) Número máximo de filas de la matriz.
mcols	(Valor: entero) Número máximo de columnas de la matriz.

Returns

(booleà) Devuelve cierto si la fila pertenece a [0..nfilas) y columna pertenece a [0..mcols) y falso en caso contrario.

```
27 {
28     bool res = false;
29 
30     /* Escribe la expresión correspondiente */
31     if (((nfilas>f)&&(f>=0))&&((mcols>c)&&(c>=0))){
32         res = true;
33     }
34 
35     return res;
36 }
```

6.2.2.7 tb_orientaciones()

```
int tb_orientaciones (
            char matriz[][MCOLS_MAX],
            int nfilas,
            int mcols,
            int f,
            int c,
            char caracter,
            int orientaciones[] )
```

A partir de una posición de la matriz (fila y columna determinadas), crea una tabla con todas las orientaciones en las que la casilla adyacente contiene el carácter que se pasa como parámetro. Devuelve el número de veces que se ha encontrado el carácter. La casilla del centro no se evalúa. 0:NOROESTE, 1:NORTE, 2:NORESTE, 3:OESTE, 4:CENTRO, 5:ESTE, 6:SUROESTE, 7:SUR, 8:SURESTE

Parameters

matriz	(Ref: tabla[][MCOLS_MAX] de carácter) Matriz de caracteres donde buscamos.
nfilas	(Valor: entero) Número real de filas de la matriz (<= NFILAS_MAX).
mcols	(Valor: entero) Número real de columnas de la matriz (<= MCOLS_MAX).

Parameters

f	(Valor: entero) Fila.
c	(Valor: entero) Columna.
caracter	(Valor: char) Carácter que se desea buscar.
orientaciones	(Ref: taula[] de enter) Orientación en la que se están volteando las fichas. Como máximo habrá 9 orientaciones.

Returns

(enter) Número total de posibles orientaciones encontradas.

```
110 {
111     int fila,col,orientacion, i;
112 
113     /* 1. Recorrer el sub-cuadrado, y para cada celda:
114      - Si es válida, el carácter coincide y no es la casilla central, añadir a la tabla.
115      */
116     i = 0;
117     orientacion = 0;
118 
119     /* Añadir el centinla */
120     for (fila =f-1;fila<=f+1;fila++)
121     {
122         for(col=c-1;col<=c+1;col++)
123         {
124             if(tb_dentro_limites(fila, col, nfilas, mcols))
125             {
126                 if((matriz[fila][col]==caracter)&&((fila!=f)||(col!=c)))
127                 {
128                     orientaciones[i]=orientacion;
129                     i++;
130                 }
131             }
132             orientacion++;
133         }
134     }
135 
136     /* Añadir centinela*/
137     orientaciones[i] = -1;
138 
139     /* Devolvemos el resultado */
140     return (i);
141 }
```

6.3 Sesión 12

Functions

- void escribe_fecha (fecha_t *fecha)
- void lee_coordenada_acotada (coord_t *coord, coord_t *coord_ini, coord_t *coord_fi)
- void ot_tablero_inicial (char tablero[ ][MCOLS_MAX], int dim)
- void ot_lee_jugador (jugador_t *jugador, char color)
- void ot_obtener_datos_jugador (jugador_t *jugador, char nombre[ ], int *color, bool *maquina, int *fila, char *col, int *puntuacion)

6.3.1 Detailed Description

Fechas iguales.

```
Define un tipos de datos para guardar una fecha (día, mes y año) y diseña un procedimiento que reciba dos fech
```

Análisis

```
¿Qué sabes a la hora de resolver el procedimento?

Para hecer el procedimiento habra una tipus de datos con dia, mes y año de tipus de dia. Despues compararemos

## Diseño

Completa el algoritmo.

~~~~
/**
*  fecha_t
* Información de una fecha
*/
tipus
    registre fecha_t és
        dia, mes , año: enter;
    fregistre
ftipus

/*
Compara dos fechas, f1 y f2, y devuelve cierto si son iguales.

@param f1 (Ref: fecha_t) Primera fecha a evaluar.
@param f2 (Ref: fecha_t) Segunda fecha a evaluar.
@return (booleà) Devuelve cierto si las fechas son iguales y
falso en caso contrario.
*/
funció fechas_iguales (
    /* Parámetros de entrada */ f1: fecha_t, f2: fecha_t) retorna booleà és
var
    /* Definición de variables locales si las necesitas. */
    iguales: boolea;
fvar
inicio  /* Código */
  iguales = falso;
    si ((f1.dia=f2.dia) i (f1.mes=f2.mes) i (f1.año=f2.año))
            iguales = cierto;
    fsi
    retorna(iguales);
ffunció
~~~~
```

Juego de pruebas fechas_iguales()

```
Añade como mínimo, una prueba al juego de pruebas, recuerda que en el caso de que resultado dependa de varias
```

Test Test del procedimiento fechas_iguales()

Descripción	Entradas		Salidas Esperada	OK?
	f1	f2	resultado	
Prueba humo	10, 3, 2020	10, 3, 2020	cierto	
	15, 3, 1994	17, 6, 1999	falso	

Compara fechas.

```
Diseña un procedimiento que reciba dos fechas y nos devuelva un valor entero según la relación que exista entr
1 - si la primera fecha es mayor que la segunda (más reciente).
0 - si las dos fechas son iguales.
-1 - si la primera fecha se menor que la segunda (más antigua).

La descripción del procedimiento es la siguiente:
```

Análisis

```
¿Qué sabes a la hora de resolver el procedimento?

El procedimiento lo resolveremos de forma que las fechas sean un tipus dia, mes y año en entero y la comparare
```

Diseño

```
Completa el algoritmo.

~~~~
/*
Compara dos fechas, f1 y f2, y devuelve un valor entero según la relación existente entre ellas:
1 - f1 > f2  La primera fecha es más reciente que la segunda.
0 - f1 = f2  Ambas fechas son iguales.
-1 - f1 < f2  La segunda fecha es más antigua que la segunda.

@param f1 (Ref: fecha_t) Primera fecha a evaluar.
@param f2 (Ref: fecha_t) Segunda fecha a evaluar.
@return (enter) Devuelve un valor según la relación existente entre las fechas:
- 1 si f1 es más reciente que f2.
- 0 si ambas son iguales.
- -1 si f1 es más antigua que f2.
*/
funció compara_fechas (
        /* Parámetros de entrada */ f1: fecha_t, f2: fecha_t) retorna enter és
var
    /* Definición de variables locales si las necesitas. */
    resultado: enter;
    iguales: boolea;
fvar
inicio  /* Código */
    iguales = falso;
    iguales = fechas_iguales(f1,f2);
    si (iguales = cierto) llavors
            resultado = 0;
            sino si ((f1.año)>(f2.año) llavors
                    resultado = 1;
                    sino si ((f1.año)<(f2.año) llavors
                        resultado = -1;
                    sino si ((f1.mes)>(f2.mes) llavors
                        resultado = 1;
                    sino si ((f1.mes)<(f2.mes)) llavors
                        resultado = -1;
                    sino si ((f1.dia)>(f2.dia)) llavors
                        resultado = 1;
                    sino
                        resultado = -1;
    fsi

    retorna (resultado);
ffunció
~~~~
```

Juego de pruebas compara_fechas()

Añade como mínimo, una prueba al juego de pruebas.

Test Test del procedimiento compara_fechas()

Descripción	Entradas		Salidas Esperada	OK?
	f1	f2	resultado	
Prueba humo	10, 3, 2020	5, 3, 2020	1	

Escribe record.

Diseña un nuevo tipo de datos para guardar la información de un récord:

- Nombre
- Color de la ficha
- Puntuación
- Fecha

Diseña un procedimiento para mostrar los datos de un registro por pantalla siguiendo el siguiente formato:

dd-mmm-aa color puntuación nombre

Análisis

¿Qué sabes a la hora de resolver el procedimento?

Para diseñar el procedimiento un tipus que contenga nombre y color de ficha que sera tipus caracter, puntuacio

Diseño

Completa el algoritmo.

```
~~~~
/**
*  record_t
* Información de una fecha
*/
tipus
    registre record_t és
        nom, ficha_color: caracter;
        puntuacion: entero;
        fecha: fecha_t;
    fregistre

    registre fecha_t és
            dia,año:enter;
            mes:caracter;
    fregistre

ftipus
/**
Imprime por pantalla la información de un récord siguiendo el siguiente formato:
dd-mmm-aa  Color   Puntuación  Nombre
Separados por tabulación y con salto de Línea al final.

@param record (Ref: record_t) Registro con toda la información del récord.
*/
acció ot_escribe_record ()
    /* Parámetros de entrada */ record: record_t) és
var
    /* Definición de variables locales si las necesitas. */
```

```
fvar
inicio  /* Código */
var
/* Definición de variables locales si las necesitas. */

fvar
inicio  /* Código */
    escribe_fecha(record.fecha\t);

    si(record.color = FICHA_BLANCA) llavors
            escriure("BLANCAS"\t);
        sino
            escriure("NEGRAS"\t);
    fsi

    escriue(record.puntuacio\t);
    escriure(record.nom\n);

facció
~~~~
```

Juego de pruebas ot_escribe_record()

```
Para hacer las pruebas, se utilizará el siguente registro (reg1):

- "Anna"
- 'O'
- 38
- 1-2-2020

(reg2)
- "Maria"
- '@'
- 40
- 17-dec-2017
```

Test Test del procedimiento ot_escribe_record()

Descripción	Entradas	Salidas Esperada	OK?
	registro	pantalla	
Prueba humo	reg1	1-feb-2020 'O' 38 "Anna"	
	reg2	17-dec-2017 '@' 40 "Maria"	

Escribe records.

```
Diseña un procedimiento para imprimir todos los registros de una tabla de registros. Al final indicará el núme
```

Análisis

```
¿Qué sabes a la hora de resolver el procedimento?

Para resolver el pricedimiento recorreremos la tabla y analizaremos la puntuacion que contienen los registros
```

Diseño

```
Completa el algoritmo.

~~~~
/**
Imprime un número determinado de los récords guardados en la tabla.

@param records (Ref: tabla[] de record_t) Tabla con los records, ordenados o no.
@param num_records (Valor: entero) Número máximo de registros a imprimir.
*/
acció ot_escribe_records
    /* Parámetros de entrada */     records: taula[] de record_t,
                                    num_records: enter) és
var
    /* Definición de variables locales si las necesitas. */
    record_max:enter;
    i:enter;
fvar
inicio  /* Código */

    i := 0;
    pos := 0;
    record_max:=record[i].puntuacio;
    mentres(i <> num_records)fer
            si(record_max<record[i].puntuacio)
                    record_max:=record[i].puntuacio;
                    pos := i;
            fsi
            i:= i+1;
    fmentres
    escriure(record[pos].nom);
    escriure(record[pos].puntuacio);
    escriure(record[pos].color);
    escriure(record[pos].fecha);
facció
~~~~
```

Juego de pruebas ot_escribe_records()

```
En este caso, la prueba será una comprobación del resultado que se muestra por pantalla.
Añade como mínimo, una prueba al juego de prueba
```

Test Test del procedimiento ot_escribe_records()

Descripción	Entradas		Salidas Esperada	OK?
	records	num_records	resultado	
Prueba humo	1	3	2	
	1	8	14	

```
# Procedimentos para la práctica:

# Escribe la fecha

Define un procedimiento para mostrar por pantalla la fecha formateada:
dd-mmm-aaaa     donde mmm es el mes indicado con tres letras.

La fecha será del tipo fecha_t.

## Análisis

¿Qué sabes a la hora de resolver el procedimento?
```

Sesión 12 Para resolver el procedimientos utilizaremos un menu de opcions donde el mes en entero de devuelva

en la abreviación correspondiente

Diseño

```
Completa el algoritmo.

~~~~
/**
Muestra por pantalla la fecha formateada: dd-mmm-aaaa

@param fecha (Ref: fecha_t) Fecha a escribir por pantalla.
*/
acció escribe_fecha
    /* Parámetros de entrada */     fecha: fecha_t) és
var
    /* Definición de variables locales si las necesitas. */

fvar
inicio  /* Código */
opcion (fecha.mes):
    1: escriure (fecha.dia,"ene",fecha.any);
    2: escriure (fecha.dia,"feb",fecha.any);
    3: escriure (fecha.dia,"mar",fecha.any);
    4: escriure (fecha.dia,"abr",fecha.any);
    5: escriure (fecha.dia,"may",fecha.any);
    6: escriure (fecha.dia,"jun",fecha.any);
    7: escriure (fecha.dia,"jul",fecha.any);
    8: escriure (fecha.dia,"ago",fecha.any);
    9: escriure (fecha.dia,"sep",fecha.any);
    10: escriure (fecha.dia,"oct",fecha.any);
    11: escriure (fecha.dia,"nov",fecha.any);
    12: escriure (fecha.dia,"dic",fecha.any);
fopcion
ffunció
~~~~
```

Juego de pruebas escribe_fecha()

```
En este caso, la prueba será una comprobación del resultado que se muestra por pantalla.
Añade como mínimo, una prueba al juego de prueba:
2-1-2020
```

Test Test del procedimiento escribe_fecha()

Descripción	Entradas	Salidas Esperada	OK?
	fecha	resultado	
Prueba humo	2-1-2020	1-ene-2020	
	17-6-2020	17-jun-2020	

Tablero del Othello.

```
Diseña un procedimiento para crear el tablero incial para jugar al Othello.
```

Análisis

```
En este caso es importante determinar cuales son las casillas centrales del tablero.
```

Diseño

```
Completa el algoritmo.

~~~~
/**
Inicializa el tablero de juego inicial, colocando las primeras fichas en el lugar correcto.

@param tablero (Ref: tabla de carácter) Tablero con la situación del juego.
@param dim (Valor: entero) Dimensión real con la que se jugará.
*/
acció ot_tablero_inicial (
    /* Parámetros de salida */      var tablero: taula [][MCOLS_MAX] de caràcter,
    /* Parámetros de entrada */     dim: enter) és
var
    /* Definición de variables locales si las necesitas. */
    i,f,c: enter;


fvar
inicio  /* Código */

 si(dim<MCOL_MAX)llavors
    f := 0;
    mentre (f<dim) fer
        c:= 0;
        mentre (c< dim) fer
            tablero[f][c] := FICHA_VACIA;
    fmentre

        si (dim = 8) llavors
            tablero[3][3] := FICHA_BLANCA;
            tablero[3][4] := FICHA_NEGRA;
            tablero[4][3] := FICHA_NEGRA;
            tablero[4][4] := FICHA_BLANCA;
        fsi
        si (dim = 8) llavors
            tablero[4][4] := FICHA_BLANCA;
            tablero[4][5] := FICHA_NEGRA;
            tablero[5][4] := FICHA_NEGRA;
            tablero[5][5] := FICHA_NEGRA;

        fsi
facció
~~~~
```

Juego de pruebas ot_tablero_inicial()

```
En este caso, la prueba será una comprobación del resultado que se muestra por pantalla.
Añade como mínimo, una prueba al juego de pruebas.
```

Test Test del procedimiento ot_tablero_inicial()

Descripción	Entradas		Salidas Esperada	OK?
	tablero	dim	resultado	
Prueba humo	tablero	8	(imagen)	

Lee jugador

```
Diseña un nuevo tipo de datos para guardar la información de un jugador:

- Nombre
```

```
- Color de la ficha
- Si es una máquina o un humano
- Puntuación
- Coordenadas de la última tirada (de tipo coord_t)

Diseña un procedimiento para leer todos los datos del jugador de teclado, excepto el color que se recibirá com

En caso de que el jugador sea una máquina, se creará un nombre estándar: maquina_color, siendo color, el color
```

Análisis

```
Si bien existe un procedimento de una biblioteca/librería estándar, este procedimiento tiene la carácterística
Evalúa la posibilidad de crear un procedimiento específico para leer cadenas de teclado.
```

Diseño

```
Completa el algoritmo.

~~~~
/**
*  jugador_t
* Información de una fecha
*/
tipus
    registre jugador_t és
        nom: taula[10] de caracter;
        color_ficha: caracter;
        persona: boolea;
        puntuacion: enter;
        coordenada: cordenada_t;
    fregistre

    registre cordenada_t és
        filas,col: enter;
    fregistre
ftipus
/**
Permite dar valor inicial a los datos de un jugador:
1. si es o no una máquina,
2. nombre: si es humano pedirlo, o maquina_color si es máquina
3. color asignado,
4. última tirada = (-1, '-') Coordenada no válida.
5. puntuación inicial = 2

@param jugador (Ref: jugador_t) Información del jugador.
@param color (Valor: carácter) Color asigando al jugador.
*/
acció ot_lee_jugador (  jugador, char color);
        /* Parámetros de salida */      var jugador: jugador_t,
        /* Parámetros de entrada */     color: caràcter) és
var
    /* Definición de variables locales si las necesitas. */
    jugador.persona := falso;

    escriure("Jugador(0) o maquina(1)?");
    llegir(jugador.persona);

    si(jugador.persona = cert) llavors
                    si (color = BLANCO)
                            jugador.nom := maquina_BLANCO;
                    sino
                            jugador.nom := maquina_NEGRO;
    fsi

    si (jugador.persona = fals) llavors
                    escrure (Nom?);
                    llegir (jugador.nom);
```

```
  fsi


    jugador.color_ficha := color;

    jugador.coordenada.fila := -1;
    jugador.coordenada.col := -1;

    jugador.puntuacion := 2

facció
~~~~
```

Juego de pruebas ot_lee_jugador()

Comprobaremos que los datos introducidos por teclado se han guardado correctamente en la variable adecuada.

Test Tests del algoritmo ot_lee_jugador()

Descripción	Entradas			Salidas Esperadas	OK?
	nombre	m/h	color	jugador	
Prueba humo	"Lluis"	H	FICHA_NEGRA		
	"maquina_BLANCO"	M	FICHA_BLANCO		

Obtener datos del jugador

En ocasiones nos interesa obtener la información de un jugador de forma independiente. Diseña un procedimiento

Análisis

¿Qué sabes o necesitas saber para resolver el ejercicio? ¿ Qué procedimientos anteriores pueden resultarte úti

Para resolver el ejercicio podemo reutilizar el procedimient ot_lee_jugadores, obtenemos los datos del jugador

Diseño

Completa el algoritmo.

```
~~~~
/**
Recupera la información de un jugador: nombre, puntuación...

@param jugador (Ref: jugador_t) Datos del jugador.
@param nombre (Ref: taula[] de caràcter) Nombre del jugador.
@param color (Ref: enter) Color del jugador.
@param maquina (Ref: booleà) Cierto si es tipo máquina y falso si es humano.
@param fila (Ref: enter) Fila de la última tirada [1..dim]
@param col (Ref: caracter) Columna de la última tirada ['A'..dim]
@param puntuacion (Ref: enter) Puntuación del jugador.
*/
tipus
    registre jugador_t és
        nom: taula[10] de caracter;
        color_ficha: caracter;
        persona: boolea;
        puntuacion: enter;
```

```
        coordenada: cordenada_t;
    fregistre

    registre cordenada_t és
        filas,col: enter;
    fregistre
ftipus
acció ot_obtener_datos_jugador (
        /* Parámetros de entrada */         jugador: jugador_t,
        /* Parámetros de salida */      var nombre: taula[] de caràcter,
                                            caracter, var maquina: boolea, var fila: enter, var col: enter, va
                                            ) és
var
    /* Definición de variables locales si las necesitas. */

fvar
inicio  /* Código */

    nombre := jugador.nom;
    maquina := jugador.person;
    fila := jugador.cordenada.fila;
    col := jugador.coordenada.col;
    puntuacion := jugador.puntuacion;

facció
~~~~
```

Juego de pruebas ot_obtener_datos_jugador()

```
Definiremos un jugador con la siguiente información (jugador1):

- "Jon"
- H
- FICHA_NEGRA
- (6,2)
- 30


Y comprobaremos que la salida es correcta.

Añade como mínimo, una prueba al juego de pruebas.
```

Test Tests del algoritmo ot_obtener_datos_jugador()

Descripción	Entradas	Salidas Esperadas						OK?
	jugador	nombre	H/M	ficha_color	fila	col	puntuacion	
Prueba humo	jugador1	"Jon"	H	FICHA_NE↩ GRA	6	2	30	

| |jugador2 | "maquina_BLANCO"| M | FICHA_BLANCA| 7 | 5 | 29 | || |jugador3 | "Andrea"| H | FICHA_BL↩ ANCA| 1 | 1 | 35 |

6.3.2 Function Documentation

6.3.2.1 escribe_fecha()

```
void escribe_fecha (
            fecha_t * fecha )
```

Muestra por pantalla la fecha formateada: dd-mmm-aaaa

Parameters

fecha	(Ref: fecha_t) Fecha a escribir por pantalla.

```
129 {
130     switch (fecha->mes){
131         case 1: printf("%d-ene-%d", fecha->dia, fecha->anyo);
132         break;
133         case 2: printf("%d-feb-%d", fecha->dia, fecha->anyo);
134         break;
135         case 3: printf("%d-mar-%d", fecha->dia, fecha->anyo);
136         break;
137         case 4: printf("%d-abr-%d", fecha->dia, fecha->anyo);
138         break;
139         case 5: printf("%d-may-%d", fecha->dia, fecha->anyo);
140         break;
141         case 6: printf("%d-jun-%d", fecha->dia, fecha->anyo);
142         break;
143         case 7: printf("%d-jul-%d", fecha->dia, fecha->anyo);
144         break;
145         case 8: printf("%d-ago-%d", fecha->dia, fecha->anyo);
146         break;
147         case 9: printf("%d-sep-%d", fecha->dia, fecha->anyo);
148         break;
149         case 10: printf("%d-oct-%d", fecha->dia, fecha->anyo);
150         break;
151         case 11: printf("%d-nov-%d", fecha->dia, fecha->anyo);
152         break;
153         case 12: printf("%d-dic-%d", fecha->dia, fecha->anyo);
154         break;
155 
156     }
157 }
```

6.3.2.2 lee_coordenada_acotada()

```
void lee_coordenada_acotada (
             coord_t * coord,
             coord_t * coord_ini,
             coord_t * coord_fi )
```

Lee las componentes de una coordenada de teclado. Después de leer el buffe debe quedar vacío.

Parameters

coord	(Ref: coord_t) Guarda la coordenada leída por teclado (fila, col).
coord_ini	(Ref: coord_t) Valores inferiores que puede tomar la coordenada.
coord_fi	(Ref: coord_t) Valores superiores que pueden tomar la coordenada. Siempre superiores a coord_min.

```
168 {
169     /* Reemplazalo con tu código */
170     fp_lee_coordenada_acotada(coord, coord_ini, coord_fi);
171 }
```

6.3.2.3 ot_lee_jugador()

```
void ot_lee_jugador (
             jugador_t * jugador,
             char color )
```

Permite dar valor inicial a los datos de un jugador:

1. si es o no una máquina,
2. nombre: si es humano pedirlo, o maquina_color si es máquina
3. color asignado,
4. última tirada = (-1, '-') coordenada no válida.
5. puntuación inicial = 2

Parameters

jugador	(Ref: jugador_t) Información del jugador.
color	(Valor: carácter) Color asigando al jugador.

```
99 {
100     //Variables
101     char maquina;
102     bool maquina_b;
103
104     //Inicializacion de variables
105     maquina = 'c';
106
107     //Codigo
108
109     while((maquina != 'H')&&(maquina != 'M')&&(maquina != 'h')&&(maquina != 'm')){
110     printf("\nH (Humano) o M (Maquina)? ");
111     scanf(" %c",&maquina);
112     }
113
114     if ((maquina == 'M')||(maquina == 'm')){
115         maquina_b = true;
116
117         if (color == FICHA_BLANCA){
118             strcpy (jugador -> nombre, "Maquina_Blancas");
119         }
120         else {
121             strcpy (jugador -> nombre, "Maquina_Negras");
122         }
123     }
124     if ((maquina == 'H')||(maquina == 'h')){
125         maquina_b = false;
126
127         printf("Indica el nombre: ");
128         vacia_buffer_teclado();
129         fgets (jugador -> nombre, CADENA_MAX, stdin);
130         strtok (jugador -> nombre, "\n");
131     }
132
133     //Guardar datos en registro
134
135     jugador->color = color;
136     jugador->maquina = maquina_b;
137     jugador->puntuacion = 2;
138     jugador->ultima_tirada.fila = -1;
139     jugador->ultima_tirada.col = '-';
140
141 }
```

6.3.2.4 ot_obtener_datos_jugador()

```
void ot_obtener_datos_jugador (
            jugador_t * jugador,
            char nombre[],
            int * color,
            bool * maquina,
            int * fila,
            char * col,
            int * puntuacion )
```

Recupera la información de un jugador: nombre, puntuación...

Parameters

jugador	(Ref: jugador_t) Registro con todos los datos del jugador.
nombre	(Ref: taula[] de caràcter) Cadena con el nombre del jugador.
color	(Ref: enter) Color con el que juega el jugador.
maquina	(Ref: booleà) Cierto si es tipo máquina y falso si es humano.
fila	(Ref: enter) Fila de la última tirada [1..dim]
col	(Ref: caracter) Columna de la última tirada ['A'..dim]
puntuacion	(Ref: enter) Puntuación del jugador.

```
155 {
156     // Variables
157     int x;
158 
159     //Codigo
160     strcpy(nombre, jugador->nombre);
161     *color = jugador->color;
162     *maquina = jugador->maquina;
163 
164 
165     *fila = jugador->ultima_tirada.fila + 1;
166     x = jugador->ultima_tirada.col + 1;
167     switch(x){
168         case 1: *col='A';
169         break;
170         case 2: *col='B';
171         break;
172         case 3: *col='C';
173         break;
174         case 4: *col='D';
175         break;
176         case 5: *col='E';
177         break;
178         case 6: *col='F';
179         break;
180         case 7: *col='G';
181         break;
182         case 8: *col='H';
183         break;
184     }
185     *puntuacion = jugador->puntuacion;
186 }
```

6.3.2.5 ot_tablero_inicial()

```
void ot_tablero_inicial (
            char tablero[][MCOLS_MAX],
            int dim )
```

Inicializa el tablero de juego inicial, colocando las primeras fichas en el lugar correcto.

Parameters

tablero	(Ref: tabla de carácter) Tablero con la situación del juego.
dim	(Valor: entero) Dimensión real con la que se jugará.

```
54 {
55
56     if (dim == 8) //Tablero dimension 8
57     {
58         tb_inicializa_matriz (tablero, 8, 8, FICHA_VACIA);
59
60         tablero[3][3]=FICHA_BLANCA;
61         tablero[3][4]=FICHA_NEGRA;
62         tablero[4][3]=FICHA_NEGRA;
63         tablero[4][4]=FICHA_BLANCA;
64     }
65
66     if (dim == 9) //Tablero dimension 9
67     {
68         tb_inicializa_matriz (tablero, 9, 9, FICHA_VACIA);
69
70         tablero[3][3]=FICHA_BLANCA;
71         tablero[3][4]=FICHA_NEGRA;
72         tablero[4][3]=FICHA_NEGRA;
73         tablero[4][4]=FICHA_BLANCA;
74     }
75
76     if (dim == 10) //Tablero dimension 10
77     {
78         tb_inicializa_matriz (tablero, 10, 10, FICHA_VACIA);
79
80         tablero[4][4]=FICHA_BLANCA;
81         tablero[4][5]=FICHA_NEGRA;
82         tablero[5][4]=FICHA_NEGRA;
83         tablero[5][5]=FICHA_BLANCA;
84     }
85 }
```

6.4 Sesión 13

Functions

- void ot_muestra_record (jugador_t jugadores[DIM_MAX])
- void ot_opcion_j (char tablero[ ][MCOLS_MAX], jugador_t jugadores[DIM_MAX], int dim, int turno)
- void ot_inicio (void)
- bool ot_guarda_juego (char nom_fichero[ ], char tablero[ ][MCOLS_MAX], int dim, jugador_t jugadores[ ], int turno)
- bool ot_recupera_juego (char nom_fichero[ ], char tablero[ ][MCOLS_MAX], int ∗dim, jugador_t jugadores[ ], int ∗turno)
- void ot_ordena_records (record_t records[ ], int dim)

6.4.1 Detailed Description

Incio del juego.

```
Organiza las diversas opciones del juego.
Muestra un menú al usuario con las diversas opciones de juego, y organiza todos los procedimientos de acuerdo

Al iniciar el juego se mostrará al usuario un menú inicial con las siguientes posibles opciones:
- 'N' - Crear un nuevo juego. Identifica a los jugadores, inicializa los tableros y toda la infor       mación
- 'C' - Carga de un fichero, si existe, toda la información del último juego guardado. Los juegos se guardan e
- 'J' - Juega. Esta opción sólo estará visible y se podrá seleccionar si el existe un juego no finalizado:
- porque previamente se ha creado uno nuevo ('N'),
- porque previamente se ha cargado un juego guardado ('C'), o
- porque se quiere continuar con un juego pausado.
Cuando se está jugando, después de cada jugada, se ofrecerá la posibilidad de:
- 'P' - pausar el juego y volver al menú inicial.
- 'C' - continuar el juego hasta el final sin volver a preguntar.
- 'S' - seguir con la siguiente jugada. Después de la misma volverá a preguntar.
- 'G' - Guarda toda la información de un juego en un fichero **binario**.
- sólo se puede guardar un juego creado que **no haya finalizado**.
- sólo se guarda un único juego.
- 'R'- Permite visualizar los records guardados en el fichero de records (de tipo **texto**). Se preguntará al
- 'S'- Salir. Finaliza el programa y devuelve el control al sistema operativo.

El programa controlará que las opciones que se muestren sean únicamente las que puede elegir el usuario en cad

Únicamente se saldrá del juego cuando se elija la opción ´S´ (Salir).
```

Análisis

```
¿Qué sabes a la hora de resolver el procedimento?
Para resplver el procedimento utilizaremos diferentes funcione y accines realizadas. Junto con eso pediremos a
```

Diseño

```
Completa el algoritmo.

~~~~
/*
Procedimiento inicial del juego.

Muestra un menú al usuario con las diversas opciones de juego, y organiza todos los procedimientos de acuerdo
*/
acció ot_muestra_record (jugadores: taula[DIM_MAX] de jugador_t) és
```

```
     save: booleà;
     num_record: enter;
     records[DIM_MAX]: record_t;

     si (jugadores[0].puntuacion > jugadores[1].puntuacion) llavors
          escriure ("Guardando record!\n");
          save := ot_guarda_record(FICH_RECORDS, jugadores[0].nombre, jugadores[0].puntuacion, jugadores[0].co
     sino
          escriure ("Guardando record!\n");
          save := ot_guarda_record (FICH_RECORDS, jugadores[1].nombre, jugadores[1].puntuacion, jugadores[1].c
     fsi

    num_record := ot_recupera_records (FICH_RECORDS, records, DIM_MAX);

     si (num_record > 0) llavors
          ot_ordena_records(records, num_record);
        ot_escribe_records(records, num_record);
    sino
         escriure ("No hay records guardados!\n");
     fsi

     si (save = false) llavors
         escriure ("No se ha podido guardar\n");
     fsi


acció ot_opcion_j (tablero: taula[] de caracter, [MCOLS_MAX], jugadores: taula[DIM_MAX] de jugador_t, dim: ent

     fin, save: boolea;
     op: caracter;

     fin := false;

     escriure ("Jugar!");
     system(T_LIMPIA_PANTALLA); ((NO SE COM ES AMB PSEUDO))
     fp_ot_muestra_info_juego(tablero, dim, jugadores, turno);

     fer( ((NO SE SI ES POSA '(' ))
          escriure ("\nP - Pausar el jugo y volver al menu inicial.\nC - Continuar el juego hasta al final sin
          llegir (op);
          vacia_buffer_teclado(); ((NO SE COM ES AMB PSEUDO))

          opcio (op)
               'C':
                    mentre (fin<>true) fer
                         fp_ot_muestra_info_juego (tablero, dim, jugadores, turno);
                         fin := ot_juega(tablero, dim, jugadores, &turno);
                              si (fin<>true) llavors
                                   ot_muestra_record (jugadores);
                              fsi
                         fmentre

               'S':
                    fp_ot_muestra_info_juego (tablero, dim, jugadores, turno);
                    fin := ot_juega (tablero, dim, jugadores, &turno);

                    si (fin=true) llavors
                         ot_muestra_record (jugadores);
                    fsi
          )mentre (fin=true);
     )mentre (fin =(op <> 'P')); ((AQUESTS DOS MENTRES NO EM QUADREN, UN VA AMB EL fer( però LALTRE NOSE))

     si (fin=false) llavors
          escriure ("Guardando partida!\n");
               save := ot_guarda_juego (FICH_JUEGO, tablero, dim, jugadores, turno);
     fsi

     si (save=false) llavors
          escriure ("No se ha podido guardar\n");
     fsi
faccio
```

```
accio ot_inicio (void) es
     dim, turno, num_record: enter;

     jugadores: taula[DIM_MAX] de jugador_t;

     records: taula[DIM_MAX] de record_t;

     op: caracter;
     tablero: taula [NFILAS_MAX][MCOLS_MAX] de caracter;

     save: boolea;

     system(T_LIMPIA_PANTALLA); ((RTTT))

     fer(

     escriure ("Escoge opcion:\nN - Crear juego.\nC - Cargar partida.\nR - Recuperar records.\nS - Salir.\n\n"
     scanf(" %c", &op);

     vacia_buffer_teclado(); ((RTTT))
     opcio (op)
          'N':
               system(T_LIMPIA_PANTALLA); ((RTTT))
               ot_crea_juego(tablero, &dim, jugadores, &turno);
               ot_opcion_j (tablero,jugadores, dim, turno);
          'C':
                system(T_LIMPIA_PANTALLA); ((RTTT))
                save := ot_recupera_juego (FICH_JUEGO,tablero,&dim, jugadores,&turno);

               si (save = false) llavors
                    escriure ("Partida no encontrada, crea una partida\n");
               sino
                    ot_opcion_j (tablero,jugadores, dim, turno);
               fsi

          'R':
               num_record := ot_recupera_records (FICH_RECORDS, records, DIM_MAX);
               si (num_record > 0) llavors
                    ot_ordena_records(records, num_record);
                    ot_escribe_records(records, num_record);
               sino
                    escriure ("No hay records guardados!\n");
               fsi
     )mentre(op <> 'S');
faccio
~~~~
```

Juego de pruebas ot_inicio()

```
Al trarse de un menú que organiza el juego, se deberá probar diferente secuencia de opciones y comprobar que l
```

Test Test del procedimiento ot_inicio()

Descripción	Entradas	Salidas Esperada	OK?
	op	resultado	
Prueba humo	'N' 'J' 'S'	ok	
Jugar sin crear	'J'	opción errónea	
Jugar sin crear	'N', 'J','S' ,'P','G'	partida guardada	

Guardar el juego

```
Diseña un procedimiento para guardar el juego actual de Othello y poder recuperarlo más adelante para seguir j
```

```
La información imprescindible para poder jugar es la siguiente:

- El tablero de juego.
- La dimensión con la que se está jugando
- Toda la información de los jugadores
- A quién le corresponde el turno.

Si el juego se ha guardado correctamente, el procedimiento nos devolverá cierto y si no se ha podido guardar,
```

Análisis

```
¿Qué es importante tener en cuenta a la hora de diseñar el algoritmo?
Para realizar este procedimiento deberemos tener en cuenta que el fichero no sea nulo. Despues guardaremos las
```

Diseño

```
Completa el algoritmo.

~~~~
/*
Guarda toda la información de un juego en un fichero binario, para poder recuperalo posteriormente. La informa

@param nom_fichero (Ref: tabla[] de carácter) Nombre del fichero binario en el que se guardará el juego.
@param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Tablero con la situación del juego.
@param dim entero (Valor: entero) Dimensión real con la que se jugaba.
@param jugadores (Ref: tabla[] de jugador_t) Tabla con la información de cada uno de los jugadores.
@param turno (Valor: entero) Información de que jugador tiene el turno para comenzar a jugar.
@return (boolano) Devuelve cierto si el juego se ha guardado correctamente y falso en caso contrario.
*/
funció ot_guarda_juego  (
/* Parámetros de entrada */ nom_fichero: taula[] de caràcter,
                                tablero: taula [][MCOLS_MAX] de caràcter,
                                dim: enter,
                                jugadores: taula[] de jugador_t,
                                turno: enter) retorna booleà és

var
                             / Definición de variables locales si las necesitas.
        guardado:boolea;
        f:f1,f2;
        i,j: enter;
fvar

inicio  / Código /
    guardado:=FALS;

    f<-fopen(nom_fichero,"w");
                si (f != NULL) llavors
                        escriuref (f,dim);
                        escriuref (f,turno);
                        escribe_matriz(f2,tablero);
                        escribe_jugador(f2,jugadores);
                        guardado=CERT;
                sino
                        guardado=FALS;
                fsi
        retorna (guardado);
  ffunció

~~~~
```

Juego de pruebas ot_guarda_juego()

```
En este caso probaremos a guardar un juego y posteriormente comprobaremos que se ha guardado correctamente.
También es conveniente comprobar que controla correctamente los errores más comunes de acceso a fichero: no se
Para comprobar que el juego se ha guardado correctamente, se necesita el procedimiento que recupere el juego g

Por ejemplo, podemos guardar un juego recien creado: tablero inicial, jugadores tipo máquina (Blancas y Negras
```

Test Test del procedimiento ot_guarda_juego()

Descripción	Entradas	Salidas Esperada	OK?
	tablero dim jugadores turno	resultado	
Prueba humo	juego recien iniciado	cierto	
Prueba humo	no se ha podido acceder al fichero	false	

Recupera el juego

```
Diseña un procedimiento para recuperar el último juego de Othello guardado para seguir jugado. Sólo se podrá r

La información imprescindible para poder jugar es la siguiente:

- El tablero de juego.
- La dimensión con la que se está jugando
- Toda la información de los jugadores
- A quién le corresponde el turno.

Si el juego se recupera correctamente, el procedimiento nos devolverá cierto y si no se ha podido recuperar, n
```

Análisis

```
¿Qué es importante tener en cuenta a la hora de diseñar el algoritmo?
Para realizar este procedimiento deberemos tener en cuenta que el fichero no sea nulo y que no este vacio. Des
```

Diseño

```
Completa el algoritmo.

~~~~
/*
Recupera toda la información de un juego guardado en un fichero binario. Puede ocurrir que el fichero esté vac

@param nom_fichero (Ref: tabla[] de carácter) Nombre del fichero binario en el que está guardado el juego.
@param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Tablero con la situación del juego.
@param dim (Ref: entero) Dimensión real con la que se jugaba.
@param jugadores (Ref: tabla[] de jugador_t) Tabla con la información de cada uno de los jugadores.
@param turno (Ref: entero) Información de que jugador tiene el turno para comenzar a jugar.
@return (booleano) Devuelve cierto si se ha recuperado el juego correctamente o falso en caso contrario.
*/
funció ot_recupera_juego  (
    /* Parámetros de entrada */ nom_fichero: taula[] de caràcter,
    /* Parámetros de salida */  var tablero: taula [][MCOLS_MAX] de caràcter,
                                var dim: enter, var jugadores: taula[] de jugador_t,
                                var turno: enter) retorna booleà és
var
        f:ficher;
        guardado:boolea;
fvar
inicio  / Código /
f<-obrir(nom_fichero,"r");
```

```
            si (f <> NULL) llavors
                    llegirf(f,dim);
                    mentre(no farxiu(f)) fer
                             llegirf(f,turno);
                             leer_matriz(f2,tablero);
                             leer_jugador(f2,jugadores);
                         fmentre
                       guardado:=CERT;
                  sino
                       guardado:=FALS;
              fsi
                retorna(guardado);
                ffunció
~~~~
```

Juego de pruebas ot_recupera_juego()

```
En este caso probaremos a recuperar un juego que hemos guardado anteriormente y comprobaremos que se ha recupe
También es conveniente comprobar que controla correctamente los errores más comunes de acceso a fichero: no se

Guardamos en un fichero un juego recien creado: "juego.dat"
```

Test Test del procedimiento ot_recupera_juego()

Descripción	Entradas		Salidas Esperada	OK?
	fichero	tablero dim jugadores turno	resultado	
Prueba humo	juego.dat	juego recien iniciado	cierto	
	juego.dat	no se ha encontrado el archivo	falso	
	juego.dat	fichero vacio	falso	

Ordena los records

```
Ordena una tabla de registros tipo `record_t`.
Los registros se ordenarán en orden **decreciente** de puntuación mayor a menor. En el caso de que las puntuac
```

Análisis

```
¿Qué algoritmo utlizarás para ordenar los records? ¿Cómo compararás las fechas?
```

Diseño

```
Completa el algoritmo.

~~~~
/*
Ordena una tabla de records. Se ordenarán por puntuación, de mayor a menor. En caso de puntuaciones iguales, s

@param records (Ref: tabla[] de record_t) Tabla que contiene los records a ordenar.
@param dim (Valor: entero) Dimensión de la tabla, número de récords que contiene la tabla.
*/
acció ot_ordena_records (
        /* Parámetros de entrada */         records: taula[] de record_t,
        /* Parámetros de entrada */     dim: enter) és
var
    /* Definición de variables locales si las necesitas. */
    max:enter;
    i:enter;
```

```
    pos:enter;
    record_t aux;
fvar
inicio  /* Código */
    i = 0;
    aux := records[0];

    mentres (i < dim) fer
        si (aux.puntuacio < records[i+1].puntuacio)llavors

                aux := records[i];
                records[i] := records[i+1];
                records[i+1] := aux;

                sino si (aux.puntuacio = record[i+1].puntuacio) llavors

                    si (compara_fecha(aux.fecha,record[i+1].fecha))

                        aux := records[i];
                        records[i] := records[i+1];
                        records[i+1] := aux;

                    fsi

                fsi

        fsi
facció
~~~~
```

Juego de pruebas ot_ordena_records()

```
Para probar si el procedimiento ordena correctamente los registros, podemos considerar una tabla (record_ot) c

|record_ot inicial          | record_ot ordenada |
|---------------------------|--------------------|
| "j1", 2-3-2020, 100, '@'  | "j2", 3-3-2020, 150, 'O' |
| "j2", 3-3-2020, 150, 'O'  | "j3", 3-2-2020, 150, '@' |
| "j3", 3-2-2020, 150, '@'  | "j1", 2-3-2020, 100, '@' |

|record_ot inicial          | record_ot ordenada |
|---------------------------|--------------------|
| "j1", 2-3-2020, 100, '@'  | "j2", 3-2-2020, 150, 'O' |
| "j2", 3-3-2020, 120, 'O'  | "j3", 3-3-2020, 120, '@' |
| "j3", 3-2-2020, 150, '@'  | "j1", 2-3-2020, 100, '@' |

También convendría comprobar que funciona correctamente si la tabla está vacía y que no accede fuera de la mis

Añade como mínimo, una prueba al juego de pruebas.
```

Test Tests del algoritmo ot_ordena_records()

Descripción	Entradas		Salidas Esperadas	OK?
	records	dim	resulado	
Prueba humo	record_ot inicial	3	record_ot (ordenada)	
	record_ot inicial	3	record_ot (ordenada)	

6.4.2 Function Documentation

6.4.2.1 ot_guarda_juego()

```
bool ot_guarda_juego (
             char nom_fichero[],
             char tablero[][MCOLS_MAX],
             int dim,
             jugador_t jugadores[],
             int turno )
```

Guarda toda la información de un juego en un fichero binario, para poder recuperalo posteriormente. La información se guarda en el fichero en el siguiente orden: dimensión, turno, tablero, jugadores.

Parameters

nom_fichero	(Ref: tabla[] de carácter) Nombre del fichero binario en el que se guardará el juego.
tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Tablero con la situación del juego.
dim	entero (Valor: entero) Dimensión real con la que se jugaba.
jugadores	(Ref: tabla[] de jugador_t) Tabla con la información de cada uno de los jugadores.
turno	(Valor: entero) Información de que jugador tiene el turno para comenzar a jugar.

Returns

(boolano) Devuelve cierto si el juego se ha guardado correctamente y falso en caso contrario.

```
391 {
392     // Variables
393     bool res;
394     FILE *f;
395
396     /* Se debe respetar el orden a la hora de guardar los dato: es muy IMPORTANTE */
397     // Codigo
398     f = fopen(nom_fichero,"wb");
399     res = false;
400
401     //Guardar datos en fichero binario
402     if (f != NULL){
403        res = true;
404             fwrite(&dim, sizeof(int), 1 , f);
405             fwrite(&turno, sizeof(int), 1 , f);
406             fwrite(tablero, sizeof(char), NFILAS_MAX*MCOLS_MAX , f);
407             fwrite(jugadores,sizeof(jugador_t),2,f);
408         fclose(f);
409     }
410
411     return (res);
412 }
```

6.4.2.2 ot_inicio()

```
void ot_inicio (
             void  )
```

Procedimiento inicial del juego. Muestra un menú al usuario con las diversas opciones de juego, y organiza todos los procedimientos de acuerdo con las diversas opciones. Controlará que se ejecutan de forma correcta.

```
318                     {
319
320     //Variables
321     int dim, turno, num_record;
322
323     jugador_t jugadores[DIM_MAX];
324
325     record_t records[DIM_MAX];
326
327     char op, tablero [NFILAS_MAX][MCOLS_MAX];
328
329     bool save;
330
331
332    system(T_LIMPIA_PANTALLA);
333
334    //Menu principal
335    do
336    {
337     printf ("Escoge opcion:\nN - Crear juego.\nC - Cargar partida.\nR - Recuperar records.\nS - Salir.\n\n
    ");
338     scanf(" %c", &op);
339
340     //Realitzar l'opcio impresa per teclat
341     vacia_buffer_teclado();
342
343     switch (op)
344     {
345          case 'N': //Nueva partida
346               system(T_LIMPIA_PANTALLA);
347               ot_crea_juego(tablero, &dim, jugadores, &turno);
348               ot_opcion_j (tablero,jugadores, dim, turno);
349               break;
350
351          case 'C': // Cargar pertida
352                system(T_LIMPIA_PANTALLA);
353                save = ot_recupera_juego (FICH_JUEGO,tablero,&dim, jugadores,&turno);
354
355                if(save == false)
356                {
357                    printf("Partida no encontrada, crea una partida\n");
358                }
359                else
360                {
361                    ot_opcion_j (tablero,jugadores, dim, turno);
362                }
363                break;
364
365          case 'R': //Mostrar records
366              num_record = ot_recupera_records (FICH_RECORDS, records, DIM_MAX);
367              if (num_record > 0)
368                {
369                    ot_ordena_records(records, num_record);
370                    ot_escribe_records(records, num_record);
371                }
372             else
373                {
374                    printf ("No hay records guardados!\n");
375                }
376    }
377   }while(op != 'S'); // Salir
378 }
```

6.4.2.3 ot_muestra_record()

```
void ot_muestra_record (
             jugador_t jugadores[DIM_MAX] )
```

Guarda y muestra al usuario los records

Parameters

jugadores	(Valor: jugador_t) Registro con todos los datos del jugdor

```
204                                                 {
205
206   //Variables
207      bool save;
208      int num_record;
209      record_t records[DIM_MAX];
210
211     //Comprobar quien ha hecho más puntos
212
213     if (jugadores[0].puntuacion > jugadores[1].puntuacion)
214      {
215         printf("Guardando record!\n");
216         save = ot_guarda_record (FICH_RECORDS, jugadores[0].nombre, jugadores[0].puntuacion
     , jugadores[0].color);
217
218      }
219
220     else
221     {
222         printf("Guardando record!\n");
223         save = ot_guarda_record (FICH_RECORDS, jugadores[1].nombre, jugadores[1].puntuacion
     , jugadores[1].color);
224     }
225
226     // Mostrar los records por pantalla
227     num_record = ot_recupera_records (FICH_RECORDS, records, DIM_MAX);
228
229     if (num_record > 0)
230     {
231         ot_ordena_records(records, num_record);
232         ot_escribe_records(records, num_record);
233     }
234     else
235     {
236         printf ("No hay records guardados!\n");
237     }
238
239     if (save == false)
240     {
241         printf("No se ha podido guardar\n");
242     }
243
244  }
```

6.4.2.4 ot_opcion_j()

```
void ot_opcion_j (
              char tablero[][MCOLS_MAX],
              jugador_t jugadores[DIM_MAX],
              int dim,
              int turno )
```

Parameters

tablero	(Ref: tabla de carácter) Tablero con la situación del juego.
dim	(Valor: entero) Dimensión real con la que se jugará.
jugadores	(Valor: jugador_t) Registro con todos los datos del jugdor
turno	(Valor: entero) Información de que jugador tiene el turno para comenzar a jugar.

```
254 {
255     //Variables
256     bool fin, save;
257     char op;
258
259     //Iniciacion de variables
260
261     fin = false;
262
```

```
263     //Menu jugar
264        system(T_LIMPIA_PANTALLA);
265        fp_ot_muestra_info_juego(tablero, dim, jugadores, turno);
266
267         do
268         {
269             printf("\nP - Pausar el jugo y volver al menu inicial.\nC - Continuar el juego hasta al final
    sin volver a preguntar.\nS - Seguir con la siguiente jugada. Depues de la misma volvera a preguntar.\n");
270             scanf(" %c",&op);
271             vacia_buffer_teclado();
272
273         switch(op)
274             {
275                 case 'C': // Continuar sin menu
276                 while (fin != true)
277                     {
278                         fp_ot_muestra_info_juego(tablero, dim, jugadores, turno);
279                         fin = ot_juega(tablero, dim, jugadores, &turno);
280
281                         if (fin == true){
282                             ot_muestra_record (jugadores);
283                         }
284                     }
285                     break;
286
287                 case 'S': // Siguiente turno
288                     fp_ot_muestra_info_juego(tablero, dim, jugadores, turno);
289                     fin = ot_juega(tablero, dim, jugadores, &turno);
290
291                     if (fin == true){
292                         ot_muestra_record (jugadores);
293                     }
294                     break;
295
296              }while(fin == true);
297
298         }while((op != 'P')); // Pausar
299
300         //Guardado de partida si no se ha terminado
301         if (fin == false)
302              {
303                 printf("Guardando partida!\n");
304                 save = ot_guarda_juego (FICH_JUEGO, tablero, dim, jugadores, turno);
305              }
306
307         // Mensaje si no se ha podio guardar
308         if (save == false)
309             {
310                 printf("No se ha podido guardar\n");
311             }
312
313 }
```

6.4.2.5 ot_ordena_records()

```
void ot_ordena_records (
              record_t records[],
              int dim )
```

Ordena una tabla de records. Se ordenarán por puntuación, de mayor a menor. En caso de puntuaciones iguales, se ordenará por fecha, de más reciente a más antigua.

Parameters

records	(Ref: tabla[] de record_t) Tabla que contiene los records a ordenar.
dim	(Valor: entero) Dimensión de la tabla, número de récords que contiene la tabla.

```
460 {
461     // variables
```

```
462     int i, fecha;
463     record_t j;
464     bool canvi;
465 
466     // Inicializacion de variables
467     i = 0;
468     fecha = 0;
469     canvi = false;
470 
471     //Codigo
472     if (dim < 2){
473         i = dim;
474     }
475 
476     else{
477         while (i!=dim){
478             if (records[i].puntos < records[i+1].puntos){
479                 j = records[i];
480                 records[i] = records[i+1];
481                 records[i+1] = j;
482                 canvi = true;
483             }
484             else if (records[i].puntos == records[i+1].puntos){
485                 fecha = compara_fechas(&records[i].fecha, &records[i+1].fecha);
486 
487                 if (fecha == -1){
488                     j = records[i];
489                     records[i] = records[i+1];
490                     records[i+1] = j;
491                     canvi = true;
492 
493                 }
494             }
495             if (canvi != true){
496                 i++;
497             }
498             else if (canvi == true){
499                     i=0;
500             }
501             canvi = false;
502         }
503     }
504 
505 }
```

6.4.2.6 ot_recupera_juego()

```
bool ot_recupera_juego (
              char nom_fichero[],
              char tablero[][MCOLS_MAX],
              int * dim,
              jugador_t jugadores[],
              int * turno )
```

Recupera toda la información de un juego guardado en un fichero binario. Puede ocurrir que el fichero esté vacío. La información se guarda en el fichero en el siguiente orden: dimensión, turno, tablero, jugadores.

Parameters

nom_fichero	(Ref: tabla[] de carácter) Nombre del fichero binario en el que está guardado el juego.
tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Tablero con la situación del juego.
dim	(Ref: entero) Dimensión real con la que se jugaba.
jugadores	(Ref: tabla[] de jugador_t) Tabla con la información de cada uno de los jugadores.
turno	(Ref: entero) Información de que jugador tiene el turno para comenzar a jugar.

Returns

(booleano) Devuelve cierto si se ha recuperado el juego correctamente o falso en caso contrario.

```
425 {
426     // Variables
427     bool res=false;
428     FILE *f;
429 
430     /* Cuidado con el orden, es muy IMPORTANTE */
431 
432     //Leer datos de fichero binario
433     f = fopen(nom_fichero, "rb");
434     if (f != NULL)
435     {
436        fread (dim, sizeof(int), 1, f);
437         if (feof(f)){
438             res = false;
439         }
440         else{
441             fread (turno, sizeof(int), 1, f);
442             fread(tablero, sizeof(char), NFILAS_MAX*MCOLS_MAX , f);
443             fread (jugadores, sizeof(jugador_t), 2, f);
444             res = true;
445         }
446         fclose (f);
447     }
448 
449     return (res);
450   }
```

6.5 Sesión 14

Functions

- bool ot_decide_casilla_auto (char tablero[ ][MCOLS_MAX], int dim, int ∗f, int ∗c, jugador_t ∗jugador)
- void ot_obtener_num_fila_col (int ∗fila, char col_ll, int ∗col)
- bool ot_decide_casilla_manual (char tablero[ ][MCOLS_MAX], int dim, int ∗f, int ∗c, jugador_t ∗jugador)
- void ot_crea_juego (char tablero[ ][MCOLS_MAX], int ∗dim, jugador_t jugadores[ ], int ∗turno)
- bool ot_juega (char tablero[ ][MCOLS_MAX], int dim, jugador_t jugadores[ ], int ∗turno)

6.5.1 Detailed Description

Decide casilla automáticamente

```
Diseña un procedimiento que genere una tirada (fila y columna) de forma aleatoria. La casilla seleccionada deb

Disponéis de un procedimiento que, a partir del tablero y un color, proporciona una lista de coordenadas válid

`(2, 3), (3, 2), (4, 5), (5, 4)`

~~~~
/* Crea una lista con todas las casillas que son válidas para colocar una ficha de un color determinado.
*
* @param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Tablero de juego.
* @param dim (Valor: entero) Dimensión real con la que se juega (>0).
* @param color (Valor: entero) Color de la ficha que se desea colocar.
* @param casillas (Ref: tabla[] de coord_t) Tabla con todas las casillas válidas para jugar.
* @return (entero) Devuelve el número de casillas válidas para jugar.
*/
funció fp_ot_lista_posibles (
        /* Parámetros de entrada */     tablero: taula[][MCOLS_MAX] de caràcter,
                                        dim: enter, color: caràcter,
        /* Parámetros de salida */  coord_t casillas[]) retorna enter és;
~~~~
```

Análisis

```
¿Utilizarías algún otro procedimiento adicional?
Para resolver el procedimento utilizaremos la funcion ofrecida para saber que casillas son posibles. Estas pos
```

Diseño

```
Completa el algoritmo.

~~~~
/* Decide la casilla dónde colocar la ficha del jugador de forma automática.
 *
 * @param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Es el tablero del juego.
 * @param dim (Valor: entero) Dimensión con la que se está jugando.
 * @param f (Ref: entero) Es la fila de la matriz dónde se colocará la ficha.
 * @param c (Ref: entero) Es la columna de la matriz dónde se colocará la ficha.
 * @param jugador (Ref: jugador_t) Es la información de jugador.
 * @return (booleano) Devuelve cierto si existe una casilla válida para jugar o
 *                  falso en caso contrario.
*/
funció ot_decide_casilla_auto (
        /* Parámetros de entrada */     tablero: taula [][MCOLS_MAX] de caràcter,
        /* Parámetros de entrada */ dim: enter,
        /* Parámetros de salida */ var f: enter, var c: enter,
```

```
        /* Parámetros de salida */ var jugador: jugador_t) retorna booleà és;
        var
            /* Definición de variables locales si las necesitas. */
            valida: booleà;
            coord_t casillas_posibles[MCOLS_MAX*MCOLS_MAX];
          i,x: enter;
        fvar
        inicio  /* Código */
            valida := true;
            i := 0;
        x :=0 ;

          x := fp_ot_lista_posibles (tablero, dim, jugador.color, casillas_posibles);

          i := entero_aleatorio_acotado (0, x-1);

          f := casillas_posibles[i].fila;
          c := casillas_posibles[i].col;

         jugador.ultima_tirada.fila := casillas_posibles[i].fila;
         jugador.ultima_tirada.col := casillas_posibles[i].col;

         si (x=0) llavors
             valida := false;
         fsi

         retorna (valida);
        ffunció
~~~~
```

Juego de ot_decide_casilla_auto()

```
Para comprobar si funciona, se puede crear un tablero con las posibilidades controladas:

- con una única posibilidad
- con ninguna posibilidad
- con varias posibilidades, como por ejemplo, el tablero incial.
```

Test Test del procedimiento ot_decide_casilla_auto()

Descripción	Entradas		Salidas Esperada				OK?
	tablero	dim	f	c	jugador.ultima_tirada	resultado	
Prueba humo	tablero inicial	8	4	5	(4, 5)	cierto	
Prueba humo	tablero inicial	8	3	2	(3, 2)	cierto	

Decide casilla de forma manual

```
Diseña un procedimiento que solicite al usuario una tirada (fila y columna). El procedimiento debe validar que
Además, actualizará la última tirada en el registro jugador.

Disponéis de un procedimiento que, a partir del tablero, el color y y un color, nos indica si una jugada (coor

~~~~
/* Valida si la jugada es correcta, es decir si se puede colocar
 * la ficha del color indicado en la coordenada.
 *
 * @param tablero (Ref: tabla de carácter) Tablero de juego.
 * @param dim (Valor: entero) Dimensión real con la que se juega.
 * @param coord (Ref: coord_t) Coordenada donde se quiere colocar la ficha.
 * @param color (Valor: char) Color de la ficha que se desea colocar.
 * @return (booleano) Devuelve cierto si es válida la jugada y falso en caso contrario.
*/
funció fp_ot_jugada_valida (
```

```
        /* Parámetros de entrada */ tablero: taula [][DIM_MAX] de caràcter, int dim,
        /* Parámetros de entrada */ coord: coord_t, color: enter) retorna booleà és;
~~~~
```

Análisis

```
¿Necesitas algún otro procedimiento adicional?
Para realizar la funcion necesitaremos el procedimiento para obtener los datos introducidos por el usuario de

## Diseño

Completa el algoritmo.

~~~~
*/
 /**
 Recupera la información del jugador en parametros de matiz

 *@param fila (Ref: enter) Fila entrada per teclat [1..dim]
 *@param col_ll (Valor: caracter) Columna entrada per teclat ['A'..dim]
 *@param col (Ref: enter) Columna pasada a numero [1..dim]
 */
acció ot_obtener_num_fila_col (var fila: enter, col_ll: caracter, var col: enter) és

            fila := fila - 1;

            opcio(col_ll)
            'A': col:= 0;
                'B': col:= 1;
                'C': col:= 2;
                'D': col:= 3;
                'E': col:= 4;
                'F': col:= 5;
                'G': col:= 6;
                'H': col:= 7;
            fopcio
facció

/* Solicita al jugador la casilla dónde colocar la ficha.
*
* @param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Es el tablero del juego.
* @param dim (Valor: entero) Dimensión con la que se está jugando.
* @param f (Ref: entero) Es la fila de la matriz dónde se colocará la ficha.
* @param c (Ref: entero) Es la columna de la matriz dónde se colocará la ficha.
* @param jugador (Ref: jugador_t) Es la información de jugador.
* @return (booleano) Devuelve cierto si existe una casilla válida para jugar o
*                    falso en caso contrario.
*/
funció ot_decide_casilla_manual (
    /* Parámetros de entrada */     tablero: taula [][MCOLS_MAX] de caràcter,
    /* Parámetros de entrada */ dim: enter,
    /* Parámetros de salida */ var f: enter, var c: enter,
    /* Parámetros de salida */ var jugador: jugador_t) retorna booleà és;
    var
    /* Definición de variables locales si las necesitas. */
        valida: booleà;
        c_ll: caracter;
        p, cont: enter;
        coord_t casillas_posibles[MCOLS_MAX*MCOLS_MAX];

        valida := false;
        cont := 0;
    fvar
inicio  /* Código */
p := fp_ot_lista_posibles (tablero, dim, jugador.color, casillas_posibles);

llegir(f);
llegir(c_ll);

ot_obtener_num_fila_col (f, c_ll, c);
```

```
mentre((valida = false)i(p <> 0)) fer

  si((casillas_posibles[cont].fila = f) i (casillas_posibles[cont].col = c)) llavors

            jugador.ultima_tirada.fila := f;
            jugador.ultima_tirada.col := c;
            valida := true;
     sino
    cont:= cont +1;
     fsi

     si (cont = p) llavors
          llegir(f);
      llegir(c_ll);
      ot_obtener_num_fila_col (f, c_ll, c);
      cont := 0;
     fsi
    retorna (valida);
ffunció
~~~~
```

Juego de ot_decide_casilla_manual()

```
Para comprobar si funciona, se puede crear un tablero con las posibilidades controladas:

- con una única posibilidad
- con ninguna posibilidad
- con varias posibilidades, como por ejemplo, el tablero incial.
```

Test Test del procedimiento ot_decide_casilla_manual()

Descripción	Entradas		Salidas Esperada				OK?
	tablero	dim	f	c	jugador.ultima↩ _tirada	resultado	
Prueba humo	tablero inicial	8	4	5	(4, 5)	cierto	
Prueba humo	tablero inicial	8	6, 3	1, 2	(3, 2)	cierto	

Crea un nuevo juego.

```
Diseña un procedimiento para crear un nuevo juego de Othello que permita, posteriormente, jugar. Se deberán in

- dimensión del tablero
- colocar las fichas para iniciar el juego
- determinar que jugador tiene el primer turno (juega con negras)
- pedir los datos del jugador (humano o persona)
```

Análisis

```
¿Qué procedimientos, diseñados anteriormente, pueden resultarte útiles?

Para realizar el procedimiento utilizaremos otras funciones o acciones realizadas anteriormente como ot_tabler
```

Diseño

```
Completa el algoritmo.
```

```
~~~~
/* Inicializa todos los elementos necesarios para poder jugar al Othello:
 * - dimensión de los tableros,
 * - inicializar el tablero de juego
 * - determinar de forma aleatoria que jugador tiene el turno, es decir,
 *   que jugador que juegará con las negras.
 * - leer los datos de los jugadores,
 *
 * @param tablero (Ref: tabla[][MCOLS_MAX] de carácter) Tablero de juego que se inicializará
 * con la posición inicial del juego.
 * @param dim (Ref: entero) Dimensión real con la que se jugará. Si es erróneo, por defecto será de 8x8.
 * @param jugadores (Ref: tabla de jugador_t) Tabla con la información de cada uno de los jugadores.
 * @param turno (Ref: entero) Información de qué jugador tiene el turno para comenzar a jugar.
*/
accio ot_crea_juego(tablero: taula[][MCOLS_MAX] de caracter, var dim: enter, jugadores: taula[] de jugador_t,

  var
     aux: jugador_t;
     color: caracter;
     num: enter;
    fvar

  inici
     color := FICHA_NEGRA;

     escriure ("Dimension del tablero[8..10]: ");
     llegir (dim);

     si ((dim <> 8) i (dim <> 9) i (dim <> 10)) llavors
          escriure ("La dimension sera 8\n");
                dim := 8;
     fsi

     ot_tablero_inicial(tablero, dim);

     num := entero_aleatorio_acotado(1, 2);

     si (num = 1) llavors
            color := FICHA_BLANCA;
     sino si (num =2)
          color := FICHA_NEGRA;
     fsi

     ot_lee_jugador(aux, color);
     jugadores[0] := aux;

     color := ot_cambia_color(color);

     ot_lee_jugador(aux, color);
     jugadores[1] := aux;

     si (jugadores[0].color = FICHA_NEGRA) llavors
          turno := 0;
     sino
          turno := 1;
     fsi
faccio
~~~~
```

Juego de pruebas ot_crea_juego()

```
Para hacer las pruebas se introduce diversos valores para comporbar que se guardan correctamente. Si utilizamo

prueba1:

- dim = 8
- jugador1 - maquina
- jugador2 - humano ("Anna")
```

Test Test del procedimiento ot_crea_juego()

Descripción	Entradas		Salidas Esperada	OK?
	dim	datos jugadores	valores de salida	
Prueba humo	8	prueba1	datos de salida correctos	
Prueba humo	15	prueba1	datos de salida correctos/ dim = 8	

6.5.2 Function Documentation

6.5.2.1 ot_crea_juego()

```
void ot_crea_juego (
            char tablero[][MCOLS_MAX],
            int * dim,
            jugador_t jugadores[],
            int * turno )
```

Inicializa todos los elementos necesarios para poder jugar al Othello:

- dimensión de los tableros,
- determinar de forma aleatoria que jugador tiene el turno, es decir, que jugador juega con las negras.
- leer los datos de los jugadores,
- inicializar el tablero de juego

Parameters

tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Tablero de juego que se inicializará con la posición inicial del juego.
dim	(Ref: entero) Dimensión real con la que se jugará. Si es erróneo, por defecto será de 8x8.
jugadores	(Ref: tabla[] de jugador_t) Tabla con la información de cada uno de los jugadores.
turno	(Ref: entero) Información de qué jugador tiene el turno para comenzar a jugar.

```
670 {
671     //Variables
672     jugador_t aux;
673     char color;
674     int num;
675 
676     // Inicializacion
677 
678     color = FICHA_NEGRA;
679 
680     //Codigo
681 
682         // Pedir dimencion
683         printf("Dimension del tablero[8..10]: ");
684         scanf("%d", dim);
685 
686         if ((*dim != 8) && (*dim != 9) && (*dim != 10)){ //Dimension erronia sera 8
687                 printf("La dimension sera 8\n");
688                 *dim = 8;
689         }
690 
691         ot_tablero_inicial(tablero, *dim); // Inicializar tablero
692 
```

```
693         num = entero_aleatorio_acotado(1, 2); // Numero aleatrio
694
695         if (num == 1){ //Ficha aleatoria
696             color = FICHA_BLANCA;
697         }
698         else if (num == 2){
699             color = FICHA_NEGRA;
700         }
701
702         //Guardar color
703         ot_lee_jugador(&aux, color);
704         jugadores[0]= aux;
705
706         color = ot_cambia_color(color);
707
708         ot_lee_jugador(&aux, color);
709         jugadores[1] = aux;
710
711         // Determinar turno
712        if(jugadores[0].color == FICHA_NEGRA){
713               *turno = 0;
714
715        }
716        else{
717           *turno = 1;
718        }
719 }
```

6.5.2.2 ot_decide_casilla_auto()

```
bool ot_decide_casilla_auto (
              char tablero[][MCOLS_MAX],
              int dim,
              int * f,
              int * c,
              jugador_t * jugador )
```

Decide la casilla dónde colocar la ficha del jugador de forma automática.

Parameters

tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Es el tablero del juego.
dim	(Valor: entero) Dimensión con la que se está jugando.
f	(Ref: entero) Es la fila de la matriz dónde se colocará la ficha.
c	(Ref: entero) Es la columna de la matriz dónde se colocará la ficha.
jugador	(Ref: jugador_t) Es la información de jugador.

Returns

(booleano) Devuelve cierto si existe una casilla válida para jugar o falso en caso contrario.

```
523 {
524     /* Variables locales */
525     bool valida;
526     coord_t casillas_posibles[MCOLS_MAX*MCOLS_MAX];
527     int i,x;
528
529     /* Inicializar */
530     valida = true;
531     i = 0;
532     x = 0;
533
534     /* Crea una lista de casillas posibles para el jugador */
```

```
535     x = fp_ot_lista_posibles (tablero, dim, jugador->color, casillas_posibles);
536     /* Determina casillas */
537     i = entero_aleatorio_acotado (0, x-1);
538 
539     *f = casillas_posibles[i].fila;
540     *c = casillas_posibles[i].col;
541 
542     jugador->ultima_tirada.fila = casillas_posibles[i].fila;
543     jugador->ultima_tirada.col = casillas_posibles[i].col;
544 
545     if (x == 0)
546     {
547         valida = false;
548     }
549 
550    /* Devuelve el resultado */
551     return (valida);
552 }
```

6.5.2.3 ot_decide_casilla_manual()

```
bool ot_decide_casilla_manual (
              char tablero[][MCOLS_MAX],
              int dim,
              int * f,
              int * c,
              jugador_t * jugador )
```

Permite al jugador decidir la casilla en la que desea colocar la ficha. Antes de retornar, comprueba que la casilla es una casilla válida.

Parameters

tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Es el tablero del juego.
dim	(Valor: entero) Dimensión con la que se está jugando.
f	(Ref: entero) Es la fila de la matriz dónde se colocará la ficha.
c	(Ref: entero) Es la columna de la matriz dónde se colocará la ficha.
jugador	(Ref: jugador_t) Es la información de jugador.

Returns

(booleano) Devuelve cierto si existe una casilla válida para jugar o falso en caso contrario.

```
597 {
598     /* Variables locales */
599     bool valida;
600     char c_ll;
601     int p, cont;
602     coord_t casillas_posibles[MCOLS_MAX*MCOLS_MAX];
603 
604     valida = false;
605     cont = 0;
606 
607     /* Crea una lista de casillas posibles para el jugador */
608     p = fp_ot_lista_posibles (tablero, dim, jugador->color, casillas_posibles);
609 
610     /* Solicita una jugada y comprueba que sea válida */
611 
612             printf("Fila [1...dim]: ");
613             scanf("%d", f);
614 
615             printf("Columna [MAJ]: ");
616             scanf(" %c", &c_ll);
```

```
617
618     ot_obtener_num_fila_col (f, c_ll, c);
619
620
621      while((valida == false)&&(p != 0)){
622
623          if((casillas_posibles[cont].fila == *f)&&(casillas_posibles[cont].col == *c)){
624
625              jugador->ultima_tirada.fila = *f;
626              jugador->ultima_tirada.col = *c;
627
628              valida = true;
629          }
630          else{
631              cont++;
632          }
633
634
635          if(cont == p){
636
637              printf("Casilla no valida.\n\nFila (1...dim): ");
638              scanf("%d", f);
639
640              printf("Columna (MAJ): ");
641              scanf(" %c", &c_ll);
642
643              ot_obtener_num_fila_col (f, c_ll, c);
644
645              cont = 0;
646          }
647
648      }
649      if (p == 0){
650          printf("No hay mas movimientos!");
651      }
652
653      /* Devuelve el resultado */
654      return (valida);
655 }
```

6.5.2.4 ot_juega()

```
bool ot_juega (
              char tablero[][MCOLS_MAX],
              int dim,
              jugador_t jugadores[],
              int * turno )
```

Procedimiento principal del juego. Es el que determina la dinámica del juego: cada jugador elije su jugada y pasa el turno al siguiente. El juego termina cuando no quedan casillas o cuando ninguno de los jugadores tiene una jugada posible. Cuando acaba, se guarda el récord del ganador en el fichero.

Parameters

tablero	(Ref: tabla[][MCOLS_MAX] de carácter) Es el tablero del juego.
dim	(Valor: entero) Dimensión con la que se está jugando.
jugadores	(Ref: tabla[] de jugador_t) Es la información de los jugadores.
turno	(Ref: entero) Indica el jugador que tiene el turno de juego.

Returns

(booleano) Devuelve cierto si el juego ha acabado y falso en caso contrario.

```
731 {
```

```
732     bool fin;
733
734     /* Reemplazalo con tu código */
735     fin = fp_ot_juega (tablero, dim, jugadores, turno);
736
737     return (fin);
738 }
```

6.5.2.5 ot_obtener_num_fila_col()

```
void ot_obtener_num_fila_col (
             int * fila,
             char col_ll,
             int * col )
```

Camia la letra intoducida por el jugador por el numero correspondiente en el tablero del [0...dim-1]

Parameters

fila	(Ref: enter) Fila entrada per teclat [1..dim]
col_ll	(Valor: caracter) Columna entrada per teclat ['A'..dim]
col	(Ref: enter) Columna pasada a numero [1..dim]

```
562 {
563     // Codidigo
564     *fila = *fila - 1;
565
566     switch(col_ll){
567         case 'A': *col= 0;
568         break;
569         case 'B': *col= 1;
570         break;
571         case 'C': *col= 2;
572         break;
573         case 'D': *col= 3;
574         break;
575         case 'E': *col= 4;
576         break;
577         case 'F': *col= 5;
578         break;
579         case 'G': *col= 6;
580         break;
581         case 'H': *col= 7;
582         break;
583     }
584 }
```

6.6 Procedimientos no evaluables.

Functions

- bool fechas_iguales (fecha_t *f1, fecha_t *f2)
- int compara_fechas (fecha_t *f1, fecha_t *f2)
- void ot_escribe_record (record_t *record)
- void ot_escribe_records (record_t records[ ], int num_records)
- bool ot_guarda_record (char nom_fichero[ ], char nombre[ ], int puntuacion, char color)
- int ot_recupera_records (char nom_fichero[ ], record_t records[ ], int dim)
- bool pertenece_cadena (char caracter, char cadena[ ])
- char lee_caracter_cadena (char cadena[ ], char texto[ ])
- void tb_inicializa_matriz (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, char caracter)
- bool tb_busca_matriz (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int *f, int *c, char caracter)
- void tb_calcula_incrementos (int orientacion, int *incf, int *incc)

6.6.1 Detailed Description

6.6.2 Function Documentation

6.6.2.1 compara_fechas()

```
int compara_fechas (
            fecha_t * f1,
            fecha_t * f2 )
```

Compara dos fechas, f1 y f2, y devuelve un valor entero según la relación existente entre ellas: 1 - f1 > f2 La primera fecha es más reciente que la segunda. 0 - f1 = f2 Ambas fechas son iguales. -1 - f1 < f2 La segunda fecha es más antigua que la segunda.

Parameters

f1	(Ref: fecha_t) Primera fecha a evaluar.
f2	(Ref: fecha_t) Segunda fecha a evaluar.

Returns

(enter) Devuelve un valor según la relación existente entre las fechas:

- 1 si f1 es más reciente que f2.
- 0 si ambas son iguales.
- -1 si f1 es más antigua que f2.

```
211 {
212     /* Definición de variables locales si las necesitas. */
213     int resultado = 0;
214     bool iguales = false;
215 
216     /* Código */
```

```
217     iguales = fechas_iguales(f1, f2);
218
219     if (iguales == true){
220         resultado=0;
221     }
222     else if ((f1->anyo)>(f2->anyo)) {
223         resultado = 1;
224     }
225     else if ((f1->anyo)<(f2->anyo)) {
226         resultado = -1;
227     }
228     else if ((f1->mes)>(f2->mes)) {
229         resultado = 1;
230     }
231     else if ((f1->mes)<(f2->mes)){
232         resultado = -1;
233     }
234     else if ((f1->dia)>(f2->dia)){
235         resultado = 1;
236     }
237     else{
238         resultado = -1;
239     }
240     return (resultado);
241 }
```

6.6.2.2 fechas_iguales()

```
bool fechas_iguales (
             fecha_t * f1,
             fecha_t * f2 )
```

Compara dos fechas, f1 y f2, y devuelve cierto si son iguales.

Parameters

f1	(Ref: fecha_t) Primera fecha a evaluar.
f2	(Ref: fecha_t) Segunda fecha a evaluar.

Returns

(enter) Devuelve cierto si las fechas son iguales y falso en caso contrario.

```
188 {
189     bool iguales = false;
190
191     if ((f1->dia == f2->dia) && (f1->mes == f2->mes) && (f1->anyo == f2->anyo)){
192         iguales = true;
193     }
194     return (iguales);
195 }
```

6.6.2.3 lee_caracter_cadena()

```
char lee_caracter_cadena (
             char cadena[],
             char texto[] )
```

Lee un carácter de teclado. Valida que el carácter pertenezca a una determinada cadena. Después de ejecutar este procedimiento, el buffer queda vacío.

Parameters

cadena	(Ref: taula[] de caràcter) Cadena que contiene los caracteres que el usuario puede elegir.
texto	(Ref: taula[] de caràcter) Texto que se muestra al usuario.

Returns

(caràcter) Devuelve el carácter tecleado por el usuario.

```
189 {
190     /* Variables locales */
191     char letra;
192     bool per;
193     printf ("%s", texto);
194     scanf ("%c", &letra);
195     /* Código */
196     per = pertenece_cadena (letra, cadena);
197     while (per != true)
198     {
199         scanf (" %c", &letra);
200         per = pertenece_cadena (letra, cadena);
201     }
202
203     /* Retornar el resultado */
204     return (letra);
205 }
```

6.6.2.4 ot_escribe_record()

```
void ot_escribe_record (
             record_t * record )
```

Imprime por pantalla la información de un récord siguiendo el siguiente formato: dd-mmm-aa Color Puntuación Nombre Separados por tabulación y con salto de Línea al final.

Parameters

record	(Ref: record_t) Registro con toda la información del récord.

```
753 {
754     //Codigo
755     escribe_fecha(&record->fecha);
756     // Cambia color banco a negro
757     if(record->color == FICHA_BLANCA)
758     {
759         printf("\tBlancas\t");
760     }
761     // Cambia color negro a blanco
762     else
763     {
764         if(record->color == FICHA_NEGRA)
765         {
766             printf("\tNegras\t");
767         }
768     }
769     printf("%d\t", record->puntos);
770     printf("%s\n", record->nombre);
771 }
```

6.6.2.5 ot_escribe_records()

```
void ot_escribe_records (
            record_t records[],
            int num_records )
```

Imprime un número determinado de los récords guardados en la tabla.

Parameters

records	(Ref: tabla[] de record_t) Tabla con los records, ordenados o no.
num_records	(Valor: entero) Número máximo de registros a imprimir.

```
780 {
781     /* Definición de variables locales si las necesitas. */
782     int i;
783     i=1;
784     record_t aux;
785     /* Código */
786     aux.puntos = records[0].puntos;
787 
788     strcpy(aux.nombre, records[0].nombre);
789 
790     aux.color=records[0].color;
791 
792     while(i<=num_records)
793     {
794         ot_escribe_record(&records[i-1]);
795         if(records[i-1].puntos<records[i].puntos)
796         {
797             aux.puntos=records[i].puntos;
798 
799             strcpy(aux.nombre, records[i].nombre);
800 
801             aux.color=records[i].color;
802         }
803 
804         i++;
805     }
806 
807     /* Resumen final */
808     printf ("---------------------------------------\n");
809     printf ("Total records: %d\n", i-1);
810     printf ("Mejor ha sido %s (%c), con %d puntos\n", aux.nombre, aux.color, aux.puntos);
811 }
```

6.6.2.6 ot_guarda_record()

```
bool ot_guarda_record (
            char nom_fichero[],
            char nombre[],
            int puntuacion,
            char color )
```

Guarda la puntuación, nombre y color de las fichas del jugador ganador en el fichero de texto. También guarda la fecha actual. La información en el fichero sigue el siguiente formato:

puntuación-mm-aa

Parameters

nom_fichero	(Ref: tabla[] de caractér) Nombre del fichero que contiene todos los récords de los juegos.
nombre	(Ref: tabla[] de carácter) Nombre del jugador.
puntuacion	(Valor: entero) Puntuación conseguida por el jugador, es decir, número de fichas de su color al final de la partida.
color	(Valor: carácter) Color de las fichas con las que jugaba.

Returns

(booleano) Devuelve cierto si el récord se ha podido añadir al fichero y falso en caso contrario.

```
827 {
828     bool res;
829 
830     /* No entra en la evaluación de la práctica */
831     res = fp_ot_guarda_record(nom_fichero, nombre, puntuacion, color);
832 
833     return (res);
834 }
```

6.6.2.7 ot_recupera_records()

```
int ot_recupera_records (
              char nom_fichero[],
              record_t records[],
              int dim )
```

Recupera del fichero los records (ordenados o no), indicando el número de récords reales que se ha recuperado del fichero. Muestra un mensaje de error si el fichero no existe y devuelve -1. ERROR: FICHERO NO EXISTE

Parameters

nom_fichero	(Ref: tabla[] de carácter) Nombre del fichero que contiene los récords.
records	(Ref: tabla[] de record_t) Tabla donde es guardan los récords.
dim	(Valor: entero) Dimensión máxima de la tabla de récords. ($>$0)

Returns

(entero) Devuelve el número de récords guardados en la tabla o -1 si se ha producido un error.

```
849 {
850     int num_records;
851 
852     /* No entra en la evaluación de la práctica */
853     num_records = fp_ot_recupera_records (nom_fichero, records, dim);
854 
855     return (num_records);
856 }
```

6.6.2.8 pertenece_cadena()

```
bool pertenece_cadena (
            char caracter,
            char cadena[] )
```

Devuelve cierto si el carácter se encuentra en la cadena.

Parameters

caracter	(Valor: caràcter) Caracter a buscar.
cadena	char (Ref: taula[] de caràcter) Cadena que contiene los caracteres donde se busca.

Returns

(booleà) Devuelve cierto si el carácter se encuentra en la cadena y falso en caso contrario.

```
157 {
158     /* Variables locales */
159     int i;
160     bool trobat;
161
162     /* Código */
163     i = 0;
164     trobat = false;
165     while ((cadena[i] != '\0') && !(trobat))
166     {
167         if (cadena[i] == caracter)
168         {
169             trobat = true;
170         }
171         else
172         {
173             i = i + 1;
174         }
175     }
176     /* Retornar el resultado */
177     return (trobat);
178 }
```

6.6.2.9 tb_busca_matriz()

```
bool tb_busca_matriz (
            char matriz[][MCOLS_MAX],
            int nfilas,
            int mcols,
            int * f,
            int * c,
            char caracter )
```

Busca en una matriz un determinado carácter e indica la fila y la columna donde se encuentra.

Parameters

matriz	(Ref: taula[][MCOLS_MAX] de caràcter) Matriz donde buscamos.
nfilas	(Valor: enter) Número de filas de la matriz (nfilas $>$ 0).
mcols	(Valor: enter) Número de columans de la matriz (mcols $>$ 0).
f	(Ref: enter) Fila donde se ha encontrado el caracter buscado.
c	(Ref: enter) Columna donde se ha encontrado el caracter buscado.
caracter	(Valor: caràcter) Caracter que se está buscando.

Returns

(booleà) Retorna cierto si lo ha encontrado y falso en caso contrario.

```
252 {
253     /* Definición de variables addicionals, si necesitas */
254     bool trobat;
255 
256     /* 1. Inicialización de variables. */
257     *f=0;
258     *c=0;
259     trobat = false;
260 
261     /* 2. Completa el algoritmo */
262     while ((*f < nfilas)&&(trobat != true)){
263         *c = 0;
264         while((*c < mcols)&&(trobat != true)){
265             if (matriz[*f][*c] == caracter){
266                 trobat = true;
267             }
268             else{
269             *c = *c + 1;
270             }
271         }
272         if(trobat != true){
273         *f = *f + 1;
274         }
275     }
276 
277     return (trobat);
278 }
```

6.6.2.10 tb_calcula_incrementos()

```
void tb_calcula_incrementos (
              int orientacion,
              int * incf,
              int * incc )
```

Determina los incrementos de fila y columna para recorrer una matriz siguiendo la orientación indicada.

Parameters

orientacion	(Valor: enter) Identificación de la orientación.
incf	(Ref: enter) Incremento aplicable a las filas.
incc	(Ref: enter) Incremento aplicable a las columnas.

```
290 {
291     /* No entra en la evaluación de la práctica */
292     fp_tb_calcula_incrementos(orientacion, incf, incc);
293 }
```

6.6.2.11 tb_inicializa_matriz()

```
void tb_inicializa_matriz (
              char matriz[][MCOLS_MAX],
              int nfilas,
```

```
            int mcols,
            char caracter )
```

Inicializa una matriz con un carácter.

Parameters

matriz	(Ref: taula[][MCOLS_MAX] de caràcter) Matriz a inicializar.
nfilas	(Valor: enter) Número de filas de la matriz (nfilas $>$ 0).
mcols	(Valor: enter) Número de columans de la matriz (mcols $>$ 0).
caracter	(Valor: caràcter) Caracter con el que se inicializa.

```
217 {
218     /* Definición de variables addicionals, si necesitas */
219     int f, c;
220 
221     /* 1. Inicialización de variables. */
222     f = 0;
223     c = 0;
224 
225     /* 2. Completa el algoritmo */
226     while (f < nfilas){
227         c = 0;
228         while (c < mcols){
229             matriz[f][c] = caracter;
230             c++;
231         }
232         f++;
233     }
234 }
```

Chapter 7

File Documentation

7.1 disenyo.md File Reference

7.2 especificaciones.md File Reference

7.3 generales.c File Reference

Procedimientos de uso general.

```
#include "generales.h"
```

Functions

- void vacia_buffer_teclado (void)
- void pausa (void)
- int siguiente_jugador (int jugador, int num_jugadores)
- int entero_aleatorio_acotado (int min, int max)
- char caracter_aleatorio_acotado (char inf, char sup)
- void escribe_fecha (fecha_t *fecha)
- void lee_coordenada_acotada (coord_t *coord, coord_t *coord_ini, coord_t *coord_fi)
- bool fechas_iguales (fecha_t *f1, fecha_t *f2)
- int compara_fechas (fecha_t *f1, fecha_t *f2)

7.3.1 Detailed Description

Procedimientos de uso general.

Authors

paula.segala@estudiants.urv.cat andrea.francoj@estudiants.urv.cat

Version

1.0

Date

27/6/19

7.3.2 Function Documentation

7.3.2.1 pausa()

```
void pausa (
            void  )
```

Pausa la ejecución y espera que el usuario pulse la tecla de return/enter.

```
25 {
26     printf("\nPulsa \"Enter\" para continuar! ");
27     vacia_buffer_teclado();
28 }
```

7.3.2.2 vacia_buffer_teclado()

```
void vacia_buffer_teclado (
            void  )
```

Elimina los caracteres del buffer de teclado.

```
16 {
17     char ch;
18     while (((ch = getchar()) != EOF) && (ch != '\n'));
19 }
```

7.4 othello.c File Reference

Implementación de los procedimientos específicos para el juego del Othello.

```
#include "othello.h"
```

Functions

- char ot_cambia_color (char color_ficha)
- void ot_tablero_inicial (char tablero[ ][MCOLS_MAX], int dim)
- void ot_lee_jugador (jugador_t *jugador, char color)
- void ot_obtener_datos_jugador (jugador_t *jugador, char nombre[ ], int *color, bool *maquina, int *fila, char *col, int *puntuacion)
- void ot_muestra_record (jugador_t jugadores[DIM_MAX])
- void ot_opcion_j (char tablero[ ][MCOLS_MAX], jugador_t jugadores[DIM_MAX], int dim, int turno)
- void ot_inicio (void)
- bool ot_guarda_juego (char nom_fichero[ ], char tablero[ ][MCOLS_MAX], int dim, jugador_t jugadores[ ], int turno)
- bool ot_recupera_juego (char nom_fichero[ ], char tablero[ ][MCOLS_MAX], int *dim, jugador_t jugadores[ ], int *turno)
- void ot_ordena_records (record_t records[ ], int dim)
- bool ot_decide_casilla_auto (char tablero[ ][MCOLS_MAX], int dim, int *f, int *c, jugador_t *jugador)
- void ot_obtener_num_fila_col (int *fila, char col_ll, int *col)
- bool ot_decide_casilla_manual (char tablero[ ][MCOLS_MAX], int dim, int *f, int *c, jugador_t *jugador)
- void ot_crea_juego (char tablero[ ][MCOLS_MAX], int *dim, jugador_t jugadores[ ], int *turno)
- bool ot_juega (char tablero[ ][MCOLS_MAX], int dim, jugador_t jugadores[ ], int *turno)
- void ot_escribe_record (record_t *record)
- void ot_escribe_records (record_t records[ ], int num_records)
- bool ot_guarda_record (char nom_fichero[ ], char nombre[ ], int puntuacion, char color)
- int ot_recupera_records (char nom_fichero[ ], record_t records[ ], int dim)

7.4.1 Detailed Description

Implementación de los procedimientos específicos para el juego del Othello.

Authors

paula.segala@estudiants.urv.cat andrea.francoj@estudiants.urv.cat

Version

1.0

Date

4/7/19

Actualizar los datos del autor, versión y fecha con vuestros datos personales.

7.5 tablas.c File Reference

Procedimientos generales para la trabajar con tablas, cadenas y matrices.

```
#include "tablas.h"
```

Functions

- bool tb_dentro_limites (int f, int c, int nfilas, int mcols)
- bool tb_busca_orientacion (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int ∗f, int ∗c, int orientacion, char car)
- int tb_orientaciones (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int f, int c, char caracter, int orientaciones[ ])
- bool pertenece_cadena (char caracter, char cadena[ ])
- char lee_caracter_cadena (char cadena[ ], char texto[ ])
- void tb_inicializa_matriz (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, char caracter)
- bool tb_busca_matriz (char matriz[ ][MCOLS_MAX], int nfilas, int mcols, int ∗f, int ∗c, char caracter)
- void tb_calcula_incrementos (int orientacion, int ∗incf, int ∗incc)

7.5.1 Detailed Description

Procedimientos generales para la trabajar con tablas, cadenas y matrices.

Authors

paula.segala@estudiants.urv.cat andrea.francoj@estudiants.urv.cat

Version

1.0

Date

27/6/19

7.6 tests_othello.c File Reference

Tests unitarios para probar los procedimientos de la práctica Othello.

```
#include "unit_test_ansi_C.h"
#include "generales.h"
#include "tablas.h"
#include "othello.h"
```

Functions

- void test_siguiente_jugador (void)
- void test_entero_aleatorio_acotado (void)
- void test_caracter_aleatorio_acotado (void)
- void test_tb_dentro_limites (void)
- void test_ot_cambia_color (void)
- float test_runner_othello (void)

Variables

- int general_ok
- int general_num_tests

7.6.1 Detailed Description

Tests unitarios para probar los procedimientos de la práctica Othello.

Authors

nom.cognom@estudiants.urv.cat

Version

1.0

Date

6/6/19

7.6.2 Function Documentation

7.6.2.1 test_caracter_aleatorio_acotado()

```
void test_caracter_aleatorio_acotado (
            void  )
```

Test Test del procedimiento caracter_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	'b'	'd'	['b'..'d']	

```
108 {
109     /* Declaración de variables locales */
110     char name[] = "entero_aleatorio_acotado";
111     int i, resultado;
112 
113     /* Inicialización variables */
114     general_ok = 0;
115     general_num_tests = 0;
116 
117     report_error("Test Suite:");
118     report_error(name);
119 
120     /* Prueba 1: Prueba de humo */
121     for (i=0; i<10; i++)
122     {
123         resultado = caracter_aleatorio_acotado('b', 'd');
124         if (assert_int_less('a', resultado,  "Test 1: Prueba de humo mayor") && assert_int_less(resultado,
      'e', "Test 1: Prueba de humo menor"))
125         {
126             general_ok++;
127         }
128         general_num_tests++;
129     }
130 
131     /* Prueba 2: ____________________ */
132 
133 
134 
135 
136     report_results(name,general_num_tests,general_ok);
137 }
```

7.6.2.2 test_entero_aleatorio_acotado()

```
void test_entero_aleatorio_acotado (
             void  )
```

Test Test del procedimiento entero_aleatorio_acotado()

Descripción	Entradas		Salidas Esperada	OK?
	min	max	aleatorio	
Prueba humo (x10)	1	3	[1..3]	

```
66 {
67     /* Declaración de variables locales */
68     char name[] = "entero_aleatorio_acotado";
69     int i, resultado;
70 
71     /* Inicialización variables */
72     general_ok = 0;
73     general_num_tests = 0;
74 
75     /* Semilla fija para repetir las pruebas */
76     srand(1);
77 
78     report_error("Test Suite:");
79     report_error(name);
80 
81     /* Prueba 1: Prueba de humo */
82     for (i=0; i<10; i++)
83     {
84         resultado = entero_aleatorio_acotado(1, 3);
85         if (assert_int_less(0, resultado, "Test 1: Prueba de humo mayor") && assert_int_less(resultado, 4,
```

```
      "Test 1: Prueba de humo menor"))
86         {
87             general_ok++;
88         }
89         general_num_tests++;
90     }
91
92     /* Prueba 2: _____________________ */
93
94
95
96
97     report_results(name,general_num_tests,general_ok);
98 }
```

7.6.2.3 test_ot_cambia_color()

```
void test_ot_cambia_color (
              void  )
```

Test Test del procedimiento ot_cambia_color()

Descripción	Entradas	Salidas Esperada	OK?
	color_ficha	resultado	
Prueba humo	FICHA_BLANCA	FICHA_NEGRA	
Ficha negra	FICHA_NEGRA	FICHA_BLANCA	
Error	'x'	FICHA_VACIA	

```
185 {
186     /* Declaración de variables locales */
187     char name[] = "ot_cambia_color";
188     char resultado;
189
190     /* Inicialización variables */
191     general_ok = 0;
192     general_num_tests = 0;
193
194     report_error("Test Suite:");
195     report_error(name);
196
197     /* Prueba 1: Prueba de humo */
198     resultado = ot_cambia_color(FICHA_BLANCA);
199     if (assert_char_equal(resultado, FICHA_NEGRA, "Test 1: Prueba de humo"))
200     {
201         general_ok++;
202     }
203     general_num_tests++;
204
205     /* Prueba 2: _____________________ */
206
207     report_results(name,general_num_tests,general_ok);
208 }
```

7.6.2.4 test_runner_othello()

```
float test_runner_othello (
              void  )
```

Test_runner para los procedimientos de la práctica.

Returns

(real) Porcentaje de pruebas superadas.

```
217 {
218     float ok, num_tests, correctos;
219     char resultado[50];
220 
221     ok = 0;
222     num_tests = 0;
223 
224     /* Test siguiente_jugador() */
225     test_siguiente_jugador();
226     ok +=general_ok;
227     num_tests +=general_num_tests;
228 
229     /* Test entero_aleatorio_acotado() */
230     test_entero_aleatorio_acotado();
231     ok +=general_ok;
232     num_tests +=general_num_tests;
233 
234     /* Test caracter_aleatorio_acotado() */
235     test_caracter_aleatorio_acotado();
236     ok +=general_ok;
237     num_tests +=general_num_tests;
238 
239     /* Test tb_dentro_limites() */
240     test_tb_dentro_limites();
241     ok +=general_ok;
242     num_tests +=general_num_tests;
243 
244     /* Test ot_cambia_color() */
245     test_ot_cambia_color();
246     ok +=general_ok;
247     num_tests +=general_num_tests;
248 
249     /* Resumen */
250     correctos = 100*ok/num_tests;
251     sprintf (resultado, "\nCorrectos = %0.2f%c\n", correctos, '%');
252     report_stdout(resultado);
253 
254     return correctos;
255 }
```

7.6.2.5 test_siguiente_jugador()

```
void test_siguiente_jugador (
            void  )
```

Test Test del procedimiento siguiente_jugador()

Descripción	Entradas		Salidas Esperada	OK?
	jugador	num_jugadores	j. siguiente	
Prueba humo	1	3	2	
Da la vuelta	1	2	0	

```
29 {
30     /* Declaración de variables locales */
31     char name[] = "siguiente_jugador";
32     int resultado;
33 
34     /* Inicialización variables */
35     general_ok = 0;
36     general_num_tests = 0;
37 
38     report_error("Test Suite:");
39     report_error(name);
40 
```

```
41     /* Prueba 1: Prueba de humo */
42     resultado = siguiente_jugador(1, 3);
43     if (assert_int_equal(resultado, 2, "Test 1: Prueba de humo"))
44     {
45         general_ok++;
46     }
47     general_num_tests++;
48
49     /* Prueba 2:  */
50
51
52
53
54     report_results(name,general_num_tests,general_ok);
55 }
```

7.6.2.6 test_tb_dentro_limites()

```
void test_tb_dentro_limites (
             void  )
```

Test Test del tb_dentro_limites()

Descripción	Entradas				Salidas Esperada	OK?
	f	c	nfilas	mcols	resultado	
Prueba humo	1	2	10	10	cierto	

```
147 {
148     /* Declaración de variables locales */
149     char name[] = "siguiente_jugador";
150     bool resultado;
151
152     /* Inicialización variables */
153     general_ok = 0;
154     general_num_tests = 0;
155
156     report_error("Test Suite:");
157     report_error(name);
158
159     /* Prueba 1: Prueba de humo */
160     resultado = tb_dentro_limites(1, 2, 10, 10);
161     if (assert_true(resultado, "Test 1: Prueba de humo"))
162     {
163         general_ok++;
164     }
165     general_num_tests++;
166
167     /* Prueba 2: _____________ */
168
169
170
171     report_results(name,general_num_tests,general_ok);
172 }
```

7.6.3 Variable Documentation

7.6.3.1 general_num_tests

```
int general_num_tests
```

7.6.3.2 general_ok

```
int general_ok
```

Procedimientos y variables auxiliares para las pruebas.

Index