

Cristina Díez Sobrino

# ÍNDICE

[REGLAS DEL JUEGO: 2](#_Toc17432)

[Métodos: 3](#_Toc7532)

[intro(); 3](#_Toc22870)

[main(); 3](#_Toc14681)

[tablero(); 3](#_Toc23477)

[simbolos(); 3](#_Toc29794)

[intercambio(); 3](#_Toc5704)

[Turno(); 3](#_Toc27456)

[movimientoDama(); 4](#_Toc269)

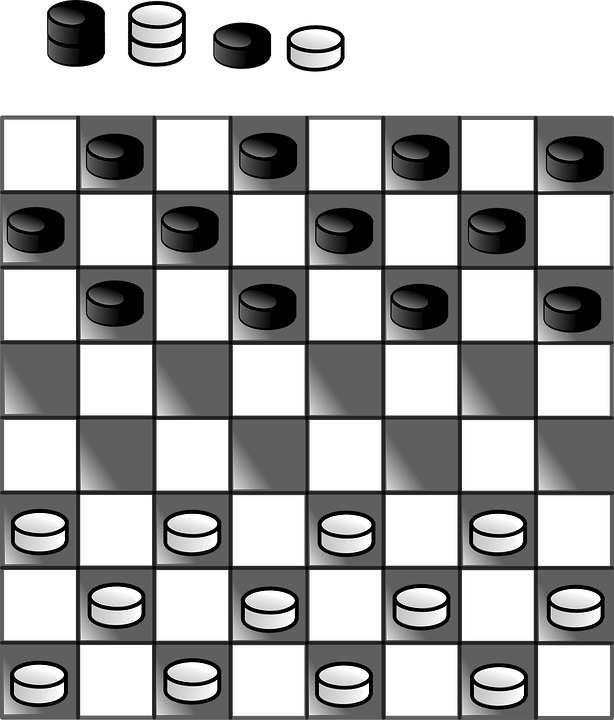
[convertirEnDama(); 5](#_Toc29564)

[comerOtraVez(); 5](#_Toc27359)

[comprobarDiagonal(); 6](#_Toc6672)

# REGLAS DEL JUEGO:

1. - Inician el juego las fichas blancas por defecto.
2. - Las fichas SOLO se mueven 1 casilla hacia ADELANTE y en DIAGONAL excepto al comer FICHA que pueden saltar.
3. - Las fichas SALTAN mas de una casilla SOLO cuando haya una ficha contraria y la siguiente este vacía.
4. - Una ficha se transforma en dama cuando llegue al lado contrario del tablero.
5. -Si es posible comer con una ficha, se comerá, si no se hace la ficha será eliminada(“soplada”).
6. - Gana quien coma mas fichas.
7. - Si pasan mas de 2 horas sin que nadie gane se declara EMPATE.



### **Métodos**:

##### intro();

Función que escribe el menú principal en el que el usuario podrá elegir una de las opciones: instrucciones, jugar o salir. (switch)

##### main();

Crea un array bidimensional matrix y lo rellena con números. Cada uno de estos números significará una cosa: 0 casilla no util, 1 casilla vacia util, 2 ficha blanca, 3 ficha negra, 4 dama blanca y 5 dama negra.

Organiza los turnos pidiendo las filas y columnas origen y destino y pasando los datos a las distintas funciones del programa para realizar las acciones.

##### tablero();

Pinta el tablero con símbolos e imprime las leyendas. También da información sobre la cantidad de piezas que le queda a cada jugador que se registra en unos contadores:

cout<<"\t\t << TABLERO DE LAS DAMAS ESPANOLAS >>"<<endl<<endl;  
cout<<"\n\t\t\t\t b = fichas BLANCAS --> " <<contFichasBlancas;  
cout<<"\n\t\t\t\t n = fichas NEGRAS --> " <<contFichasNegras;

##### simbolos();

Cambia los números por letras para mayor comprensión del jugador: b ficha blanca, n ficha negra, B dama blanca, N dama negra y v hueco vacío.

##### intercambio();

Realiza los intercambios en el matrix.

##### Turno();

Es el método que se utiliza en el turno de cada uno de los jugadores. Este método controla:

1. Que la posición que inserte el jugador sea correcta.

if ((filaOrigen < 0 && FILAS <= filaOrigen) &&  
 (colOrigen < 0 && COLUMNAS <= colOrigen)) { // Comprueba filas y col esten dentro del tablero  
 cout << "\n No ves que el tablero no tiene esa posicion ? Estas fuera del tablero\n";  
 return -1;  
}

1. Que la ficha que elija para mover sea una ficha de su color.

if (jugador == FICHAS\_BLANCAS ) {//Comprueba que muevas solo donde haya fichas blancas  
 if (matrix[filaOrigen][colOrigen] != FICHAS\_BLANCAS && matrix[filaOrigen][colOrigen] != DAMAS\_BLANCAS) {  
 cout << "\n <<< Eres CIEGO ? , No tienes fichas \*\* BLANCAS \*\* en esa posicion >>>\n";  
 return -1;  
 }

1. Que se mueva a un espacio vacío.

if (matrix[filaDestino][colDestino] != ESPACIO\_VACIO) {//Comprueba que muevas solo espacios vacios.  
 cout << "\nSolo puedes mover a un espacios VACIOS!! como tu cerebro, jeje";  
 return -1;  
}

1. Que las fichas, que no sean damas, solo puedan moverse hacia delante.

if (filaOrigen <= filaDestino) {  
 cout << "\nNo te da o que ??? No puedes mover hacia atras o quedarte en el mismo lugar\n";  
 return -1;  
}

1. Que las fichas, que no sean damas, solo puedan moverse un espacio, excepto si comen que podrán saltar. Si llegan al final del tablero se convierten en dama.

//Comprueba si es un movimiento normal, una sola posicion  
if (filaOrigen - filaDestino == -1 || filaOrigen - filaDestino == 1) {  
 //si llega al final se transforma en dama  
 if (colOrigen - colDestino == 1 || colOrigen - colDestino == -1) {  
 intercambio(matrix, filaOrigen, colOrigen, filaDestino, colDestino);  
 convertirEnDama(matrix, filaOrigen, colOrigen, filaDestino, colDestino, jugador);

[...]

if (jugador == FICHAS\_NEGRAS && matrix[comerFichaBlancas][comerFichaNegras] != FICHAS\_BLANCAS  
&& matrix[comerFichaBlancas][comerFichaNegras] != DAMAS\_BLANCAS) {  
 cout << "\nSolo puedes saltar si COMES FICHA \n";  
 return -1;

Esta función llama a otras funciones que realizan comprobaciones y movimientos. Para el control de las damas llama a la función movimientoDama.

##### movimientoDama();

Esta función guarda en una variable “jugadorEnem” al valor que tiene el jugador enemigo. Es decir, si somos el jugador 2, el enemigo será el 3 y al contrario. También utiliza un contador para las fichas del enemigo que más tarde se restarán a las del jugador contrario.

int jugadorEnem;  
int contadorEnem;  
if(jugador == FICHAS\_NEGRAS) jugadorEnem = FICHAS\_BLANCAS, contadorEnem= contFichasBlancas;  
else jugadorEnem = FICHAS\_NEGRAS,contadorEnem= contFichasNegras;

[... tras realizar los movimientos y haber podido comer fichas....]

if(jugador == FICHAS\_NEGRAS) contFichasBlancas= contadorEnem;  
else contFichasNegras =contadorEnem;

Esta función de divide en 4 determinando si el movimiento es diagonal inferior derecha, diagonal inferior izquierda, diagonal superior derecha o diagonal superior izquierda, comparando los valores de la fila y columna origen con el destino para saber en que dirección es el movimiento. Tras esto, va sumando o restando a dos variables “fila” y “columna” que sirven de guía para ver que hay en la siguiente casilla de la diagonal. Si se encuentra dos fichas juntas se dirá que no se pueden saltar dos a la vez, tiene que haber un espacio detrás de cada una. Si se encuentra una ficha amiga, se dirá que no se puede saltar nuestras propias piezas. Si no, se comerán loas piezas enemigas encontradas en el camino, se restarán en el contador enemigo y se realizará el cambio con el método intercambio y por último se mostrará el tablero con el método tablero.

//movimiento diagonal inferior izquierda  
}else if(filaOrigen<filaDestino && colOrigen<colDestino){  
 do {  
 fila++, column++;  
 //ver si hay una ficha o dama del enemigo en la casilla y si la hay comersela  
 if (matrix[fila][column] == jugadorEnem || matrix[fila][column] == jugadorEnem+2) {  
 //comprobar que no hay dos fichas seguidas en la diagonal  
 if(matrix[fila+1][column+1] != ESPACIO\_VACIO){  
 cout<<"No puedes saltar dos fichas a la vez\n";  
 return -1;  
 }else matrix[fila][column] = ESPACIO\_VACIO, contadorEnem--;  
 }else if (matrix[fila][column] == jugador || matrix[fila][column] == jugador+2) {  
 cout<<"No puedes saltar tus propias piezas\n";  
 return -1;  
 }  
 }while (fila!= filaDestino && column!=colDestino);  
}

##### convertirEnDama();

Este método comprueba si una ficha ha llegado al final del tablero y si es así la transforma en dama.

//si llega a la ultima fila transformar en dama  
if(filaDestino == 0 && jugador==FICHAS\_NEGRAS){  
 matrix[filaDestino][colDestino]=DAMAS\_NEGRAS;  
 cout<<"Tu ficha se convierte en una dama negra\n";  
}else if(filaDestino == 7 && jugador==FICHAS\_BLANCAS){  
 matrix[filaDestino][colDestino]=DAMAS\_BLANCAS;  
 cout<<"Tu ficha se convierte en una dama blanca\n";  
}

##### comerOtraVez();

Esta función se utiliza en turno(); para, después de que una ficha haya comido, se coma otra vez si hay fichas enemigas adyacentes. Hay que recordar, que si las hubiera y no se comen, la ficha que no ha comido será “soplada” (eliminada).

Se le pregunta al jugador si puede comer de nuevo. Si el jugador responde que sí, se comprobará si es así mediante el método comprobarDiagonal(); .Si realmente no puede comer , se le dirá que no mienta:

if(comprobarDiagonal(matrix,jug,destinof,destinoc)==0){  
 cout<<"NO MIENTASS!! No se puede comer mas con esa ficha";  
 return 0;

Si sí puede seguir comiendo, se le pedirá que meta un nuevo destino y se realizará la comprobación de que sea correcto. Si es correcto, come la pieza enemiga y se vuelve a preguntar con recursividad en el mismo método si se puede comer de nuevo, ya que se puede dar el caso de llegar a comer 3 piezas en un mismo turno con una ficha.

if (comerOtraVez( matrix, jugador, filaOrigen, colOrigen, filaDestino, colDestino,  
 comerFichaBlancas,comerFichaNegras, jugador)==0){  
 return 0;  
}

Si por el contrario, el jugador dice que no puede comer más, se realizará de nuevo la comprobación. Si no podía comer más con esa ficha pasará a ser el turno del contrario, en cambio, si sí se podía comer más se le avisará al usuario que esa pieza que no ha comido va a ser eliminada y pasará a ser el turno del contrario, por lo tanto habremos perdido una ficha.

//metodo que comprueba que hay en las diagonales  
int podiaComer = comprobarDiagonal(matrix, jug, destinof, destinoc);  
//si sí se podía comer se le quita la ficha  
if (podiaComer == 1) {  
 cout << "Podias comer otra vez, tu ficha sera eliminada\n";  
 if(matrix[destinof][destinoc] == FICHAS\_BLANCAS){  
 contFichasBlancas--;  
 }else if(matrix[destinof][destinoc] == FICHAS\_NEGRAS){  
 contFichasNegras--;  
 }  
 matrix[destinof][destinoc] = ESPACIO\_VACIO;

##### comprobarDiagonal();

Comprueba si hay fichas en las diagonales para poder comerlas. Depende del jugador que seamos ya que si somos el jugador de las fichas negras, buscará fichas blancas para comer y al contrario. También hay que tener en cuenta que detrás de la pieza a comer debe haber un espacio vacío para poder “saltar” con nuestra pieza, si no, no será una pieza “comible”. Para esto utilizo una variable que me indica si la pieza “podía comer” o no y otras variables guía para recorer y guardar la diagonal y ver si se cumplen los requisitos.

int fila1;//guardar fila del valor1 a comprobar si esta vacio  
int col1;//guardar columna del valor1 a comprobar si esta vacio  
int fila2;//guardar fila del valor2 a comprobar si esta vacio  
int col2;//guardar columna del valor2 a comprobar si esta vacio  
int puedeComer; //controlador de si es posible comer (1 si, 0 no)  
if (jug == FICHAS\_NEGRAS) {  
 fila1 = filaOrigen - 1;  
 col1 = colOrigen + 1;  
 fila2 = filaOrigen - 1;  
 col2 = colOrigen - 1;  
 if (matrix[fila1][col1] == FICHAS\_BLANCAS) {  
 fila1 = fila1 - 1;  
 col1 = col1 + 1;  
 if (matrix[fila1][col1] == ESPACIO\_VACIO ) {  
 puedeComer =1;  
 }  
 }if (matrix[fila2][col2] == FICHAS\_BLANCAS) {  
 fila2 = fila2 - 1;  
 col2 = col2 - 1;  
 if (matrix[fila2][col2] == ESPACIO\_VACIO ) {  
 puedeComer =1;  
 }  
 }