UNIVERSIDAD
POLITÉCNICA
DE MADRID

TRABAJO INFORMÁTICA INDUSTRIAL Y COMUNICACIONES:

DISEÑO E IMPLEMENTACIÓN DEL AJEDREZ

Grupo Defensa Petrov:
Adrián Rodriguez Fernández
Manuel Campillos Vega
Adrian Juan Drag
Alonso López Carballal
Antón Souto García

ÍNDICE

Contenido

INTRODUCCION

Prefacio

El ajedrez es un juego en el que compiten dos personas, una contra la otra, en el que cada una de las cuales dispone, en un principio, de dieciséis piezas móviles colocadas sobre un tablero dividido en sesenta y cuatro casillas (ocho filas y ocho columnas). Las casillas son blancas y negras, ordenadas de forma alterna a lo largo del tablero, y constituyen las posibles posiciones que pueden tomar la totalidad de las fichas a lo largo del desarrollo del juego. Las dieciséis piezas que toma cada jugador al inicio son: Un rey, una reina, dos caballos, dos alfiles, dos torres y ocho peones.

Se trata de un juego de estrategia cuyo objetivo consiste en derrocar al rey oponente. Esta situación se da cuando alguna de las piezas se encuentra en una posición que le permita realizar un movimiento a la casilla en la que se sitúa el rey del otro bando. Cuando esto ocurre, si el rey no dispone de escapatorias, piezas aliadas que puedan derrocar a la amenazante o que el propio rey no pueda derrocar dicha pieza, se produce la situación conocida como “jaque mate”. En esta posición la partida habría terminado y el jugador ofensivo habría ganado la partida. En el caso de que exista alguna vía de escape siendo este amenazado la situación se denomina simplemente “jaque” y la partida continúa.

El ajedrez es considerado un deporte por el Comité Olímpico Internacional. La FIDE es la encargada de regular las competiciones, en las cuales cada jugador compite de forma individual o por equipos, siendo una de las más importantes las Olimpiadas de Ajedrez.

Historia

Pese a ser un deporte practicado y conocido mundialmente sus raíces y orígenes no son claros y siguen a día de hoy abiertos al debate. Pese a la duda sobre el origen y desarrollo del juego lo que si parece claro es que no es obra de una sola persona, dada su complejidad y posibilidades de combinaciones.

El ancestro primordial del ajedrez se denomina *Chaturanga,* juego que se remonta al siglo VI antes de cristo en la India, época conocida como “Era Dorada” por la gran prosperidad y avance en ciencias y artes. Las reglas exactas de este juego son desconocidas, pero lo más seguro es que se parecieran a las de su sucesor: *Shatranj.* La palabra “Chaturanga” se traduce como “cuatro divisiones militares”, nombre dado por una de las grandes obras de la literatura sánscrita a una determinada formación de batalla denominada *Mahabharata.* Este juego fue rápidamente difundido a través de rutas comerciales, llegando a Persia, el Imperio Bizantino y posteriormente a Asia. El resto es historia.

El comienzo del Ajedrez de la Edad Moderna podemos situarlo en el siglo XV. Tras la reciente llegada del juego a Europa se dan ciertas modificaciones en el juego hasta llegar al que hoy día conocemos, con la reminiscencia de la sociedad medieval plasmada en sus figuras, incluso de la sociedad árabe en el nombre de “Alfil”, que significa elefante en árabe. Las reglas de juego no fueron unificadas hasta la aparición de la imprenta a mediados de siglo. A partir de este hito se pudieron componer libros y tratados que unificaron el juego en todo el mundo.

A finales del siglo XV, en España, se introdujo una pieza fundamental para el juego hoy en día: la dama. Esta introducción supuso toda una revolución, que, impulsada por los ajedrecistas españoles, fue extendida por América y las partes de Europa en las que no tenía arraigo por el momento. Este cambio otorgó al juego un salto de dinamismo y una reducción considerable en los tiempos de partida, puesto que la dama es con diferencia la pieza más versátil e importante de cara al desarrollo del juego. Esta es capaz de moverse en todas las direcciones posibles, ambas diagonales y verticales, todas las casillas posibles sin realizar saltos entre piezas amigas o enemigas.

Desde aquella y hasta hoy el Ajedrez es considerado mucho más que un simple juego, si no también una ciencia, un arte y un deporte mental. Esto es así hasta tal punto que en España se recomendó la enseñanza obligatoria del ajedrez en los colegios en 1994. Además, es reconocido como disciplina deportiva en 156 países al reunir requisitos comunes a los deportes, tales como:

- Accesible a toda persona que quiera practicarlo.
- Función de divertimento, juego.
- Principio de rendimiento, mental.
- Regido por reglas inequívocas y limitantes.
- Fórmula de competición.
- Presencia internacional.
- Presencia de una organización plena en el ámbito deportivo: Federaciones, árbitros, resultados, rankings, etc.

Programación:

En este punto nos gustaría comenzar la explicación de nuestro programa siguiendo una progresión cronológica del proceso de programación y como fuimos haciéndolo en equipo. La programación del proyecto se llevó a cabo en equipo de manera simultánea a través de videollamada, donde partes del equipo desarrollaban tareas y otros solucionan posibles errores o mejoras. La herramienta utilizada para la sincronización ha sido GitHub, que una vez hechos a él resulta muy cómoda, permitiéndonos el trabajo en paralelo así como la vuelta atrás en caso de necesidad.

Una vez hecha esta pequeña introducción detallaremos ya el paso a paso. En primer lugar, se creó el proyecto, así como se evaluaron las posibles tareas necesarias para lograr nuestro objetivo; tras lo cual, se empezó con las clases tablero y partida. Tras la realización de estas dos clases, necesitábamos poder colocar las piezas en sus lugares correspondientes, por lo que el siguiente paso fue la creación de un vector2D para la creación de la siguiente clase, que fue la clase coordenada. Con esta clase y la clase tablero conseguimos representar las casillas.

Ya con el tablero y la posibilidad de representación, el siguiente paso fue la creación de las piezas. Para ello creamos la clase abstracta pieza y nos repartimos las clases derivadas: Rey, Reina, Torre, Alfil, Caballo y Peón. Y es tras esto, donde ya tenemos una base del trabajo que empezamos a repartirnos tareas y a trabajar en grupos más pequeños según la tarea a realizar y es así como fuimos avanzando creando las siguientes utilidades: Uso de ratón, Movimiento de piezas, Movimientos legales, Movimientos especiales (enroque, promociones de peones…), Turnos, Captura de piezas, Eliminación de piezas, Jaque, Una IA, Menús, Ficheros, Sonidos, modos de juegos, pantallas, y un largo etc.

INTERFAZ:

Nuestro juego consta de una serie de pantallas que son menús realizados con photoshop. La primera pantalla que aparecerá nada más correr el programa es la siguiente:

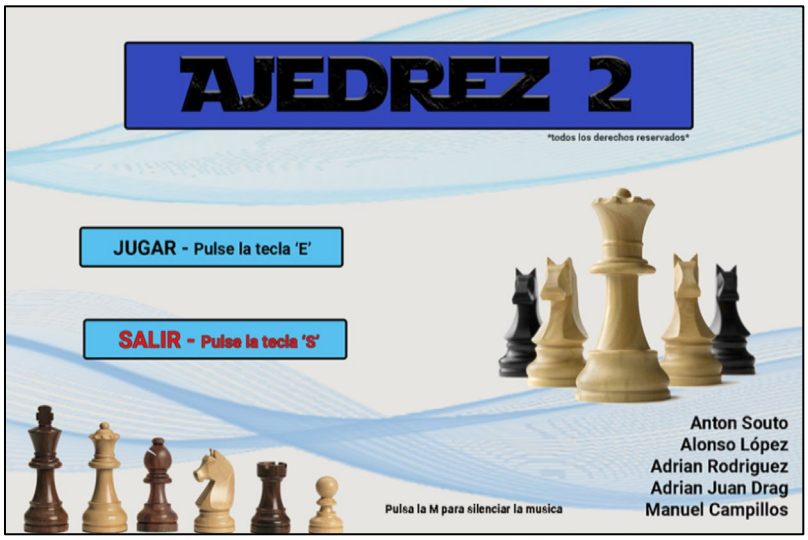

Figura 1. Menú de inicio

Esta pantalla muestra los nombres de los participantes del grupo y nos indica que para empezar a jugar deberemos darle a la tecla E y en caso de querer salir tendremos que pulsar la tecla S. Además, desde el inicio del juego tendremos música de fondo y como nos indica la pantalla podremos quitarla y ponerla con la tecla M en cualquier momento de la ejecución del programa. Una vez pulsamos la tecla E nos llevará a la siguiente pantalla que es:

Figura 1. Modos de Juego

Aquí podemos ver que nos pide que pulsemos una tecla para seleccionar el modo de juego deseado que puede ser tanto jugador contra jugador, Jugador contra IA o modos especiales que hemos diseñado.

En caso de seleccionar el modo PvE nos llevará hasta esta ventana donde podremos seleccionar el color de la IA, o lo que es lo mismo, con qué color no jugaremos.

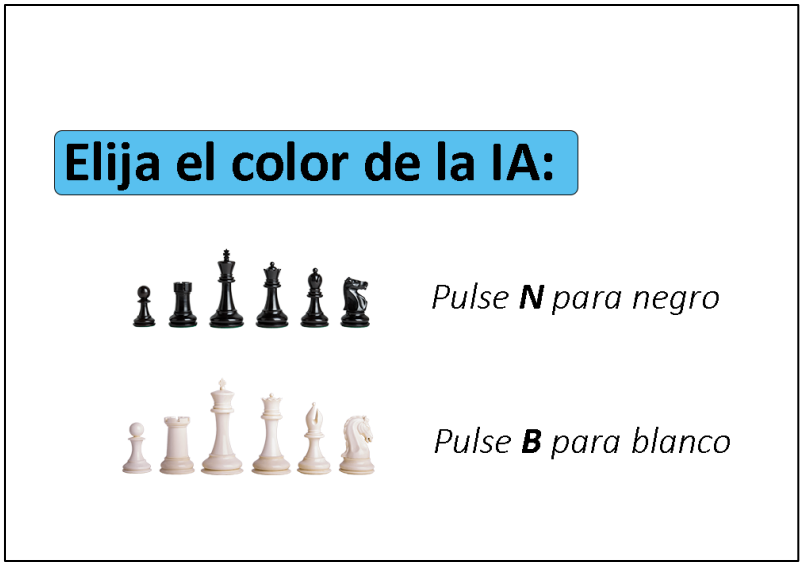

Figura 2. Selección IA

Si seleccionamos el modo loco con la tecla 3, nos llevará a una pantalla donde tenemos varios modos de juego no convencionales. Implementamos el famoso modo creado por Fisher, Ajedrez960, donde cambia la posición de las piezas distintas a los peones. Además, creamos un modo donde crea aleatoriamente 15 piezas de cualquier tipo para cada jugador, pero solo un rey para permitir el jaque mate. Por último, implementamos varios modos de juego donde todas las piezas a excepción del rey son de un mismo tipo (reinas, caballos…).

Figura 3. Otros modos de juego

Una vez elegido cualquiera de los modos, la siguiente etapa será ya la partida. Aquí podemos ver una captura de la interfaz de juego. Para su diseño nos hemos inspirado en los típicos tableros profesionales de madera y hemos buscado una jugabilidad alta en cuanto a la sencillez del diseño.

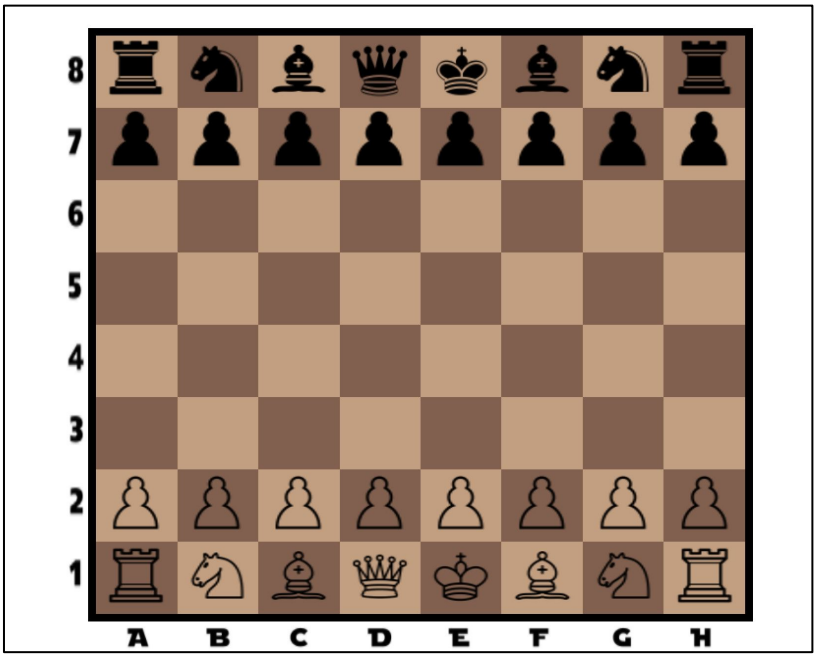

Figura 4. Tablero con sus fichas

Además, hemos añadido una pausa para su uso durante la partida, lo que nos ofrece un par de opciones para mejorar la experiencia.

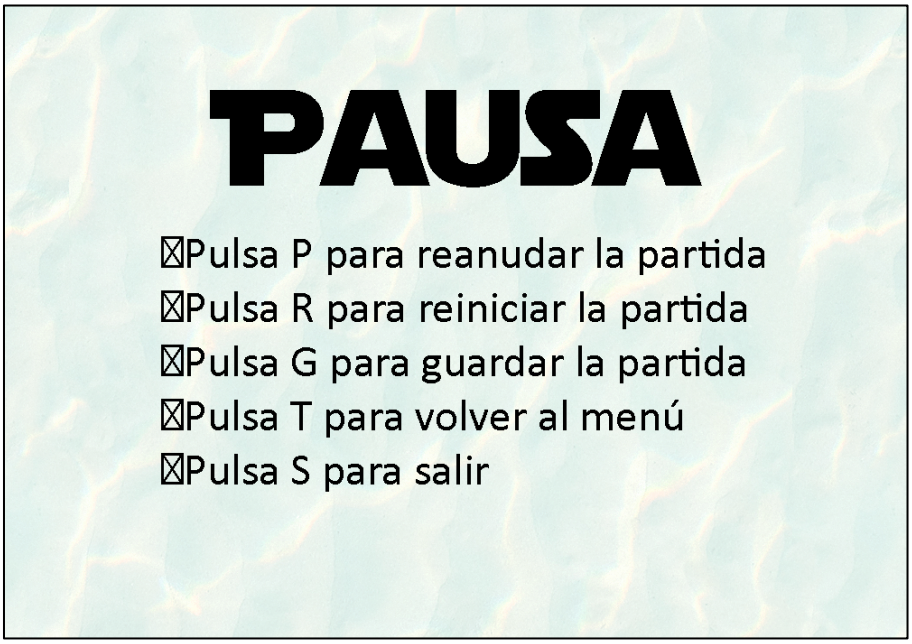

Figura 5. Menú Pausa

Otro estado es la promoción; esto es, siempre que lleguemos con un peón a la octava fila nos saldrá esta ventana para que elijamos en que pieza quiere convertirse:

Figura 6. Pantalla de selección de promoción

Por último, cuando se acaba la partida porque se produce un jaque mate, se viaja a una pantalla donde podemos guardar el historial, volver a jugar, ir al menú…

Figura 7. Pantallas de final

CÓDIGO:

Tras esta introducción, iremos comentando brevemente las diferentes clases empleadas para la realización de nuestro ajedrez. Estas son:

- Vector2D
- Coordenada
- Tablero
- Partida
- ListaPiezas
- Pieza
- Torre
- Caballo
- Alfil
- Reina
- Rey
- Coordinador

Clase Vector2D

```
#pragma once


class vector2D
{
public:
    float x;
    float y;


    vector2D(float x, float y);

};
```

La clase vector2D simplemente es un contenedor de información, que utilizamos para guardar las coordenadas en float de las piezas para dibujarlas.

Clase coordenada

```
#pragma once

#include<iostream>

enum{};
using namespace std;
class vector2D;
enum color{NEGRO, BLANCO, NONE};

class coordenada
{
    string letra;
    int numero;
public:
```

```
    //Constructores
    coordenada();
    coordenada(string letra, int numero);
    coordenada(int fila, int columna);

    //Getters
    int getFila();
    int getColumna();
    vector2D toVector();
    string getLetra() { return letra; }
    color getColorCasilla();

    //Setters
    void setCol(int columna);
    void setFil(int fila);

    //Convertidores y operadores
    string toLetraCol(int numero);
    bool operator==(coordenada coord);
    coordenada operator-(coordenada coord);

};
```

Esta clase contiene una letra de tipo *string*, que representa la columna, y un número de tipo *int* que representa la fila. Esta clase Coordenada tiene diferentes métodos que nos permiten transformar las coordenadas en formato "ajedrez", a por ejemplo, filas y columnas (de tipo *int*) o a *float*, que utilizamos para imprimir las piezas en sus coordenadas en la clase Tablero.

Clase Tablero

```
#pragma once
#include "ETSIDI.h"
#include "coordenada.h"

class tablero
{
public:
    void dibuja();
    void PintarMovPosibles(coordenada coord[], coordenada
coord2[]);
};
```

En esta clase dibujamos el tablero; es decir, pintamos las 64 casillas con sus correspondientes colores y también pintamos los movimientos posibles de cada pieza, cambiando de color las casillas afectadas.

Clase Partida

```
#pragma once
#include "ListaPiezas.h"
#include "tablero.h"
#include "coordenada.h"
#include "stdlib.h"


using namespace std;

class partida
{
    //Atributos IA
```

```
    bool existeIA = false;
    bool calculando = false;
    color colorIA = NEGRO;

    string nombreFichero = "ESTANDAR.txt";

public:
    tablero tablero;
    ListaPiezas piezas;

    //Camara
    float x_ojo;
    float y_ojo;
    float z_ojo;

    void inicializa();
    void inicializaAleatoriamente();
    void inicializaAjedrez960();
    void inicializaTipo(tipo_pieza tipo);
    void mover();
    void dibuja();

    void guardarPartida();
    void guardarHistorial();
    void promocionar(tipo_pieza tipo);

    //Mouse
    void mouse(int button, int state, int x, int y);
    void getColFilMouse(int x, int y, int &fila, int &columna);

    //Getters
    bool getJaqueMateBlanco() { return piezas.getJaqueMateBlanco(); }
    bool getJaqueMateNegro() { return piezas.getJaqueMateNegro(); }

    //Setters
    void setIA(bool IA, color colorIA);
    void setFichero(string fichero) { nombreFichero = fichero; }
};
```

En esta clase inicializamos la partida gracias a la función “inicializa” o cualquiera de los distintas alternativas para los otros modos de juego. Esta función es la encargada de cargar la posición del “ojo” de cara al tablero, poner en marcha la música que sonará mientras se esté jugando e invocar la función “cargarPartida” encargada del posicionamiento de las fichas en el tablero según un fichero donde se guardan las coordenadas y tipos de ficha. Los modos convencionales llaman a otros métodos de creación de la clase listaPiezas.

También es esta clase la encargada de dibujar la partida que estamos jugando gracias a la función “dibuja”. En ella se pinta la pantalla la ventana donde se va a representar el juego en primer lugar, posteriormente se pintan las letras y los números que nombran las casillas y por último se invoca a los métodos de las clases “ListaPiezas” y “Tablero” que gestionan su dibujo. Además, también se hace una comprobación de la variable “si” de las piezas para comprobar el turno y pintar así los movimientos posibles para la pieza seleccionada.

Con respecto a la interacción con el usuario contiene un método denominado “mouse” encargado de parametrizar la fila y la columna sobre la que se sitúa el mouse (solo si se hace click con el) y gracias a esto enviar la dirección de memoria de la pieza seleccionada (mediante otra función de esta clase denominada getColFilMouse) a una auxiliar para dibujar los movimientos posibles y posteriormente moverla a la posición deseada. Hecho esto la variable antes nombrada “si” se pondrá a false por el cambio de turno.

Además de esto se encarga de la promoción de los peones a través de la función "promocionar" seleccionando el tipo de pieza en el que se convertirá, guardar la partida invocando al método correspondiente de ListaPiezas y seleccionar el color con el que jugará la IA.

Clase ListaPiezas

```
#pragma once
#include "ETSIDI.h"
#include "pieza.h"
#include "rey.h"
#include "reina.h"
#include "caballo.h"
#include "Peon.h"
#include "Alfil.h"
#include "Torre.h"

#include <math.h>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <string>


#define MAX_PIEZAS 100


class ListaPiezas
{
private:
    //Lista
    pieza* listaPiezas[MAX_PIEZAS];
    int nPiezas = 0;
    int nPosibles=0;

    //Variables relacionadas con el enroque
    bool enroqueBlanco = true;
    bool enroqueNegro = true;
    bool torreBlancaIzq = true;
    bool torreBlancaDrc = true;
    bool torreNegraIzq = true;
    bool torreNegraDrc = true;

    //Variables relacionadas con el jaque
    int PosiblesJaque = 0;
    bool jaqueBlanco=false;
    bool jaqueNegro=false;
    bool jaqueMateBlanco = false;
    bool jaqueMateNegro = false;

    //Color de la IA
    color colorIA = NEGRO;

public:
    //Variables para pintar las coordenadas
    coordenada movimientosPosibles[64];
    coordenada coordenadaPintar[64];
    coordenada coordenadaComer[8];
    bool si = false;

    //Turno
    color proximoTurno = BLANCO;

    tipo_pieza tipo_promocion;
    tipo_pieza modo_loco;
    bool promocionar;

    pieza* apuntar;

    //Constructores
    ListaPiezas();
    ~ListaPiezas();
```

```
    //Agregar y eliminar piezas
    void crearPiezasAleatoriamente();
    void crearMismoTipo(tipo_pieza tipo);
    void crearAjedrez960();
    bool agregarPieza(pieza* pieza);
    void eliminar(int index);
    void eliminar(pieza* pieza);
    void borrarContenido();

    void guardarPartida();
    void guardarHistorial();
    void cargarPartida(string nombreFichero);

    void dibuja();

    //Getters
    bool getJaqueMateBlanco() { return jaqueMateBlanco; }
    bool getJaqueMateNegro() { return jaqueMateNegro; }

    //Funciones del enroque
    void enroque(pieza* pieza, int fila, int columna);              //Realiza el enroque, moviendo la
torre correspondiente
    void anularEnroque(pieza* pieza, int fila, int columna);        //Si se mueve alguna de las piezas
del enroque, impide que se realice

    //Movimiento de las piezas
    void moverPieza(pieza* pieza, int fila, int columna);           //Hace las comprobaciones necesarias
para mover la pieza
    bool movimientoLegal(pieza* pieza, int fila, int columna);      //Devuelve true si es un movimiento
legal
    bool movimientoLegalJaque(pieza* pieza, int fila, int columna); //Igual que movimientoLegal pero sin
la comprobacion del jaquePosible
    bool comprobarTurno(pieza* pieza);                              //Devuelve el color del proximo
movimiento
    bool comprobarColor(int index, coordenada coord);               //Comprueba el color de la pieza que
le pasas con el de la posible pieza de esa posicion
    void movPosibles(pieza* aux);                                   //Funcion para imprimir los
movimientos posibles de las piezas al clickarlas

    //Funciones para la colision de piezas
    bool comprobarAlfil(pieza* pieza, int fila, int columna);
    bool comprobarTorre(pieza* pieza, int fila, int columna);
    bool comprobarReina(pieza* pieza, int fila, int columna);
    bool comprobarPeon(pieza* pieza, int fila, int columna);
    bool comprobarRey(pieza* pieza, int fila, int columna);

    bool comerPieza(pieza* pieza, int fila, int columna);   //Devuelve true si se puede comer una pieza

    //Funciones para el jaque
    void jaque(color Color);                                //Comprueba que haya jaque al color que se
pasa
    bool jaqueBool(color Color);                            //Lo mismo que jaque(color color) pero
devuelve true si hay jaque en vez de modificar el atributo de la clase
    bool jaquePosible(pieza* pieza,int fila, int columna);  //Comprueba que un movimiento no provoque
jaque, para impedir movimientos
    bool jaqueMate(color color);                            //Comprueba si hay jaque mate


    //Busqueda de piezas o de casillas
    pieza* buscarPieza(int fila, int columna);
    bool mirarCasilla(int fila, int columna);
    bool comprobarPieza(pieza* aux, int fila, int columna);

    void Promocion(pieza* pieza);
    void comprobarPromocion(pieza* pieza);

    //IA
    void setColorIA(color colorIA);
    void algoritmoIA();
    void algoritmoIAv2(int iteraciones, int profundidad=1);
    int evaluacion(int fila, int columna);
    int evaluacionCompleta();

    //Funciones para la IA que no funcionan
    int maxi(pieza* pieza1,color color, int profundidad, int fila, int columna);
```

```
    int mini(pieza* pieza1,color color, int profundidad, int fila, int columna);
};
```

La listaPiezas es la clase central de nuestro trabajo. Para su mejor explicación vamos a hacer una tabla con sus atributos ya que tiene muchos y variados

Nombre	Tipo	Descripción
listaPiezas[MAX_PIEZAS]	pieza*	Vector de piezas
nPiezas	int	Número de piezas
enroqueBlanco	bool	True si no se ha movido el rey blanco, para permitir o no el enroque
enroqueNegro	bool	True si no se ha movido el rey negro, para permitir o no el enroque
torreBlancaIzq	bool	True si la torre de la casilla (a1) no se ha movido, para permitir o no el enroque
torreBlancaDrc	bool	True si la torre de la casilla (h1) no se ha movido, para permitir o no el enroque
torreNegraIzq	bool	True si la torre de la casilla (a8) no se ha movido, para permitir o no el enroque
torreNegraDrc	bool	True si la torre de la casilla (h8) no se ha movido, para permitir o no el enroque
PosiblesJaque	int	Variable que almacena los posi
jaqueBlanco	bool	Variable que es true si el blanco hace jaque al negro
jaqueNegro	bool	Variable que es true si el blanco hace jaque al negro

jaqueMateNegro	bool	Variable que es true si el negro gana
jaqueMateBlanco	bool	Variable que es true si el blanco gana

Entre las funciones más destacadas podemos observar comprobar pieza donde le pasas la pieza que quieres mover y a la fila y la columna de destino. Esta función a su vez va a llamar a las funciones, comprobar alfil, comprobar torre, comprobar peón…, donde dependiendo del tipo que tenga la pieza llamará a una función o a otra para comprobar si de verdad se puede mover o si hay una pieza que le impide el movimiento. Estas funciones lo que hacen es comprobar todos sus movimientos legales y permitirlos hasta que haya una colisión, por ejemplo, que un alfil pueda moverse en diagonal hasta que "impacta" con una pieza. En comprobarRey se comprueban también las condiciones para que se produzca un enroque, esto es, que no se haya movido el rey o la torre involucradas, y que durante el movimiento no haya casillas atacadas por el rival.

Luego tenemos la función movientoLegal donde comprueba que es el turno de la pieza, que no hay una pieza de su mismo color en la casilla de destino, que sea un movimiento permitido para su tipo de pieza, y que el movimiento no te provoque un jaque. Respecto a los jaques, existe una función que te obliga a moverte a solo aquellas casillas que bloqueen el jaque. Si no existe ninguna se produce el jaque mate y se acaba la partida.

La función movPosibles le pasas una pieza y te recorre todo el tablero preguntado a movimientoLegal, casilla por casilla, si se puede mover ahí. Sí se puede, esa posición se guarda en un vector de coordenadas, que luego se pasa a tablero, que se encargará de pintar los movimientos posibles en gris.

Otra función destacable es el moverPieza, la cual comprueba que se pueda mover al sitio deseado, por medio de la funcion movimientoLegal. Si se puede, se mueve la pieza, se guarda en el historial este movimiento, y si se produce una captura, se elimina la pieza que estaba en esa casilla. También, aquí se llaman a las funciones responsables del enroque. Por ejemplo, la función enroque calcula si se produce un enroque y según cual sea, mueve la torre correspondiente. Una vez se produce el movimiento se recalculan los jaques. Para esto existe una funcion llamada jaque que calcula todos los movimientos posibles del rival, y si alguno ataca al rey pone a true la variable jaque del color involucrado. Si hay jaque, se llama a la funcion jaqueMate que comprueba si se acaba la partida, calculando los posibles movimientos que anulan el jaque. Si no existe ninguno se acaba la partida. Si no hay jaque mate, se cambia el color del proximoTurno al que corresponda.

Por otro lado, tenemos los métodos de guardar partida y de guardar historial, cuya diferencia es que el método de guardar partida guarda en un fichero con un nombre elegido por el usuario las posiciones de las piezas existentes en el tablero una vez que el usuario pausa el juego, y el método de guardar historial guarda el historial de movimientos realizados por los jugadores en el fichero Historial.txt (que sólo guarda los movimientos de la última partida) o en otro fichero con un nombre elegido por el usuario en la pantalla final.

Por otra parte, tenemos una función llamada cargarPartida. Esta función va decodificando un fichero (que tiene el mismo formato que guardarPartida) según los caracteres que se va encontrado, el primero es una letra que define a los 6 tipos de pieza que hay(K para rey, Q para reina, R para torre, p para peon, N para caballo y B para alfil), el segundo carácter es el color (0 para negro, 1 para blanco) y los dos siguientes caracteres son la columna y la fila de la ficha.

K151 significa Rey Blanco en la casilla e5

Otros métodos que tiene la listaPiezas es cargar los distintos modos de juego. Todos parten de la misma base, que es crear piezas según un numero aleatorio creado por medio de la función tiraDado de la librería ETSIDI. Según el modo correspondiente se crean unos u otros tipos de pieza.

Para la programación de la IA, creamos un método que calcula todos los movimientos posibles que puede realizar y escoge el de mayor puntuación. Esta puntuación se calcula según el valor de la pieza que come. También se tiene en cuenta que, para cada tipo de pieza, su posición es más valiosa según la casilla. Por ejemplo, que un alfil este en las diagonales principales es más valioso que se encuentre en un borde del tablero. Con este concepto se tiene en cuenta tanto la posición que le quitas al rival como la que gana la IA.

Se intentó programar un algoritmo *Minimax* que calculase con una determinada profundidad el mejor movimiento, pero no se obtuvo éxito, por lo que no se eliminó del fichero, pero tampoco se utiliza.

Para solventar alguno de los problemas que nos fuimos encontrando, tuvimos que crear dos piezas “fantasmas”, que son peones de color NONE y fuera del tablero, que ocupan la primera y la última posición del arreglo.

Clase Pieza

```
#pragma once
#include "coordenada.h"
#include "vector2D.h"
#include "freeglut.h"
#include "ETSIDI.h"
#include <fstream>
#include <string>

enum tipo_pieza {REY, REINA, ALFIL, TORRE, CABALLO, PEON};

class pieza
{
    coordenada coord;
    float altura = 8.0f, ancho = 6.0f;

protected:
    tipo_pieza tipo;
    int valor;
    color icolor;

public:

    pieza(color color, coordenada coord);
    pieza();
    ~pieza();

    coordenada getCoordenada();
    void setColumna(int columna);
    void setFila(int fila);
    int getColumna();
    int getFila();
```

```
    virtual color getColor();
    virtual void dibuja();
    virtual bool movimientoLegal(coordenada destino)=0;
    virtual void guardarHistorial() = 0;


    float getAltura() { return altura; }
    float getAncho() { return ancho; }
    virtual tipo_pieza getTipo() { return tipo; }
    int getValor() { return valor; }
    virtual int getValorPos(int fila, int columna)=0;
};
```

Esta es una clase abstracta que nos proporciona la definición para sus clases derivadas. Cada pieza tiene en primer lugar unas coordenadas para su representación, un color (blanco o negro), una altura y un ancho para dibujarlas, así como un enum que define su tipo y un int su valor. Se adjunta la tabla de valores:

Rey	**100**
Reina	90
Torre	50
Caballo	30
Alfil	30
Peón	10

Por otro lado, cada pieza tiene las siguientes funciones:

- Movimiento legal (para establecer los movimientos posibles) de tipo virtual.
- Dibuja para ilustrarlos, tambien de tipo virtual.
- Guardar historial (que guarda en un fichero los movimientos efectuados con dicha pieza), de tipo virtual.
- Funciones que nos sirven para la creación de la IA o de las piezas:
    - getTipo, que nos identifica las piezas.
    - getValor, que nos da un valor para cada tipo de pieza.
    - getValorPos, que asigna un valor a cada casilla del tablero para priorizar la mejor casilla según el tipo de pieza.

En movimiento legal podemos ver como le pasamos una coordenada destino y al ser una función virtual, tendrá que ser definida por todas sus clases derivadas, además de ser diferente en cada clase.

La función de esto es que le pasamos una coordenada y nos devuelve true si esa pieza podría moverse ahí. Posteriormente para ver si de verdad se puede mover ahí sin saltar ninguna pieza es una

función ya explicada en lista piezas donde comprueba su movimiento casilla por casilla para ver si hay una pieza en su camino.

Clase Torre

```
#pragma once
#include "pieza.h"
#include <fstream>
#include <string>

class Torre : public pieza {

    int puntPosNegras[8][8] = {
      0,  0,  0,  0,  0,  0,  0,  0,
      5, 10, 10, 10, 10, 10, 10,  5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
      0,  0,  0,  5,  5,  0,  0,  0
    };
    int puntPosBlancas[8][8] = {
      0,  0,  0,  5,  5,  0,  0,  0
     - 5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
     -5,  0,  0,  0,  0,  0,  0, -5,
      5, 10, 10, 10, 10, 10, 10,  5,
      0,  0,  0,  0,  0,  0,  0,  0,
    };

public:
    Torre(color color, coordenada coord);
    Torre();

    void dibuja();
    bool movimientoLegal(coordenada destino);
    int getValorPos(int fila, int columna);
    void guardarHistorial();
};
```

Se comenta con más detalle esta clase Torre, ya que el resto de las piezas son prácticamente iguales, donde el único cambio son los movimientos legales de cada pieza. En primer lugar podemos ver dos matrices (una para las blancas y otra para las negras) que utilizaremos para la IA, que gracias la función que se menciona antes (getValorPos) nos prioriza esas casillas con mayor valoración. Añadir que la matriz está invertida de arriba a abajo. Y por último, podemos ver el resto de atributos y funciones ya mencionadas en “Pieza”.

Clase Caballo

```
#pragma once
#include "pieza.h"
#include <math.h>
#include <fstream>
#include <string>

class caballo : public pieza {

    int puntPosNegras[8][8] = {
    -50,-40,-30,-30,-30,-30,-40,-50,
    -40,-20,  0,  0,  0,  0,-20,-40,
    -30,  0, 10, 15, 15, 10,  0,-30,
    -30,  5, 15, 20, 20, 15,  5,-30,
    -30,  0, 15, 20, 20, 15,  0,-30,
    -30,  5, 10, 15, 15, 10,  5,-30,
    -40,-20,  0,  5,  5,  0,-20,-40,
    -50,-40,-30,-30,-30,-30,-40,-50,
    };
```

```
    int puntPosBlancas[8][8] = {
    -50,-40,-30,-30,-30,-30,-40,-50,
    -40,-20,  0,  5,  5,  0,-20,-40,
    -30,  5, 10, 15, 15, 10,  5,-30,
    -30,  0, 15, 20, 20, 15,  0,-30,
    -30,  5, 15, 20, 20, 15,  5,-30,
    -30,  0, 10, 15, 15, 10,  0,-30,
    -40,-20,  0,  0,  0,  0,-20,-40,
    -50,-40,-30,-30,-30,-30,-40,-50,
    };

public:
    //Constructores
    caballo();
    caballo(color color, coordenada coord);

    void dibuja();
    bool movimientoLegal(coordenada destino);
    int getValorPos(int fila, int columna);
    void guardarHistorial();


};
```

Exactamente lo mismo que en clase torre y quizás lo más interesante de esta clase ha sido su movimiento que es distinto al resto al poder saltar, por lo que no hace falta comprobar choques. El resto de clases (Alfil, Reina) serán igual y solo mostramos su código:

Clase Alfil

```
#pragma once
#include "pieza.h"
#include <fstream>
#include <string>

class Alfil : public pieza {

    int puntPosNegras[8][8] = {
    -20,-10,-10,-10,-10,-10,-10,-20,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -10,  0,  5, 10, 10,  5,  0,-10,
    -10,  5,  5, 10, 10,  5,  5,-10,
    -10,  0, 10, 10, 10, 10,  0,-10,
    -10, 10, 10, 10, 10, 10, 10,-10,
    -10,  5,  0,  0,  0,  0,  5,-10,
    -20,-10,-10,-10,-10,-10,-10,-20,
    };

    int puntPosBlancas[8][8] = {
    -20,-10,-10,-10,-10,-10,-10,-20,
    -10,  5,  0,  0,  0,  0,  5,-10,
    -10, 10, 10, 10, 10, 10, 10,-10,
    -10,  0, 10, 10, 10, 10,  0,-10,
    -10,  5,  5, 10, 10,  5,  5,-10,
    -10,  0,  5, 10, 10,  5,  0,-10,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -20,-10,-10,-10,-10,-10,-10,-20,
    };

public:
    //Constructores
    Alfil(color color, coordenada coord);
    Alfil();

    void dibuja();
    bool movimientoLegal(coordenada destino);
    int getValorPos(int fila, int columna);
    void guardarHistorial();

};
```

Clase Reina

```
#pragma once
#include "pieza.h"
#include <fstream>
#include <string>

class reina : public pieza {
    int puntPosNegras[8][8] = {
    -20,-10,-10, -5, -5,-10,-10,-20,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -10,  0,  5,  5,  5,  5,  0,-10,
     -5,  0,  5,  5,  5,  5,  0, -5,
      0,  0,  5,  5,  5,  5,  0, -5,
    -10,  5,  5,  5,  5,  5,  0,-10,
    -10,  0,  5,  0,  0,  0,  0,-10,
    -20,-10,-10, -5, -5,-10,-10,-20
    };
    int puntPosBlancas[8][8] = {
    -20,-10,-10, -5, -5,-10,-10,-20
    - 10,  0,  5,  0,  0,  0,  0,-10,
    -10,  5,  5,  5,  5,  5,  0,-10,
      0,  0,  5,  5,  5,  5,  0, -5,
     -5,  0,  5,  5,  5,  5,  0, -5,
    -10,  0,  5,  5,  5,  5,  0,-10,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -20,-10,-10, -5, -5,-10,-10,-20,
    };


public:
    reina(color color, coordenada coord);
    reina();

    void dibuja();
    bool movimientoLegal(coordenada destino);
    int getValorPos(int fila, int columna);
    void guardarHistorial();

};
```

Clase Rey

```
#pragma once
#include "pieza.h"
#include <fstream>
#include <string>

class reina : public pieza {
    int puntPosNegras[8][8] = {
    -20,-10,-10, -5, -5,-10,-10,-20,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -10,  0,  5,  5,  5,  5,  0,-10,
     -5,  0,  5,  5,  5,  5,  0, -5,
      0,  0,  5,  5,  5,  5,  0, -5,
    -10,  5,  5,  5,  5,  5,  0,-10,
    -10,  0,  5,  0,  0,  0,  0,-10,
    -20,-10,-10, -5, -5,-10,-10,-20
    };
    int puntPosBlancas[8][8] = {
    -20,-10,-10, -5, -5,-10,-10,-20
    - 10,  0,  5,  0,  0,  0,  0,-10,
    -10,  5,  5,  5,  5,  5,  0,-10,
      0,  0,  5,  5,  5,  5,  0, -5,
     -5,  0,  5,  5,  5,  5,  0, -5,
    -10,  0,  5,  5,  5,  5,  0,-10,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -20,-10,-10, -5, -5,-10,-10,-20,
    };


public:
    reina(color color, coordenada coord);
    reina();

    void dibuja();
    bool movimientoLegal(coordenada destino);
    int getValorPos(int fila, int columna);
    void guardarHistorial();
```

```
};
```

Clase Coordinador

```
#pragma once
#include "partida.h"
#include "freeglut.h"
#include "ETSIDI.h"

enum Estado { INICIO, MODOS, JUEGO, PAUSA, JAQUEMATE_BLANCO, JAQUEMATE_NEGRO, ELECCION_IA, PROMOCIONAR
};

class Coordinador
{
public:
    //Constructores
    Coordinador();
    virtual ~Coordinador();

    void mouse(int button, int state, int x, int y);
    void Tecla(unsigned char key);

    void mueve();
    void musica();
    void dibuja();

protected:
    partida partida;
    Estado estado;

private:
    bool modoMusica=false;

};
```

En esta clase creamos una máquina de estados que nos permita pasar por cada uno de ellos con el objetivo de añadir los distintos menús que se muestran en la parte de interfaz. Para conseguir esto hacemos que la instancia del objeto de clase “partida” la ejecute este coordinador en vez de en el código principal, de forma que todo lo propiamente relacionado con el juego se ejecute en uno de los estados de esta máquina, correspondiente con el estado “JUEGO”.

Asimismo, esta clase es la encargada de gestionar la música ambiente del menú de inicio, gestionar las variables que controlan el funcionamiento del mouse y las teclas de entrada para la selección de modos de juego, etc. Todas las funciones que gestionan las variables de la clase “partida” estan instanciadas en el código principal en las respectivas funciones destinadas a ello, como pueden ser “OnTimer” para la accción de mover o “OnDraw” para dibujar el tablero.

También aquí se encuentra el código del teclado, que nos permite transaccionar entre unos estados y otros según elija el usuario. Por ejemplo, en el estado INICIO, si pulsamos E pasamos al estado MODOS.

Estados:

- **INICIO:** Abre la ventana de inicio y pinta la imagen .PNG.
- **MODOS:** Estado inmediatamente siguiente al “INICIO” si no se sale del juego. En él puedes seleccionar el tipo de juego entre los mencionados en el apartado de interfaz, en caso de 1vs1 llamará al método “inicializa” de partida y en caso de 1vsIA transitará al estado “ELECCION IA”. Si por el contrario se elije el modo “Loco” se pasa a un estado de selección adicional.

- **JUEGO:** Llama al método “dibuja” de nuestra instancia de partida y detecta un posible jaque mate o promoción de piezas para el cambio de estado. También se pasa a la lista piezas las coordenadas del ratón.
- **PAUSA**: estado de pausa, aquí podemos reiniciar la partida, volver al menú, guardar la partida, salir, etc.
- **JAQUE MATE NEGRO/BLANCO:** Gracias al método getJaqueMateBlanco/Negro se hace la transición a este estado, en el que tienes la opción de reiniciar partida o volver al INICIO. También da la opción al usuario de guardar historial, llamando al metodo “setFichero” de partida.
- **ELECCION IA:** Se llama al método setIA para elegir el color de sus piezas y se inicializa la partida.
- **PROMOCIONAR:** En caso de que el método “promocionar” de ListaPiezas devuelva un true se activa este estado en el que por selección de teclado elegimos la pieza a la que promociona el peón coronado.
- **MODOS_LOCO:** en este estado seleccionamos el modo de juego alternativo que queremos jugar.