Alberto Rodríguez Pérez
alberto.rodper.2@educa.jcyl.es

MANUAL DEL PROGRAMADOR

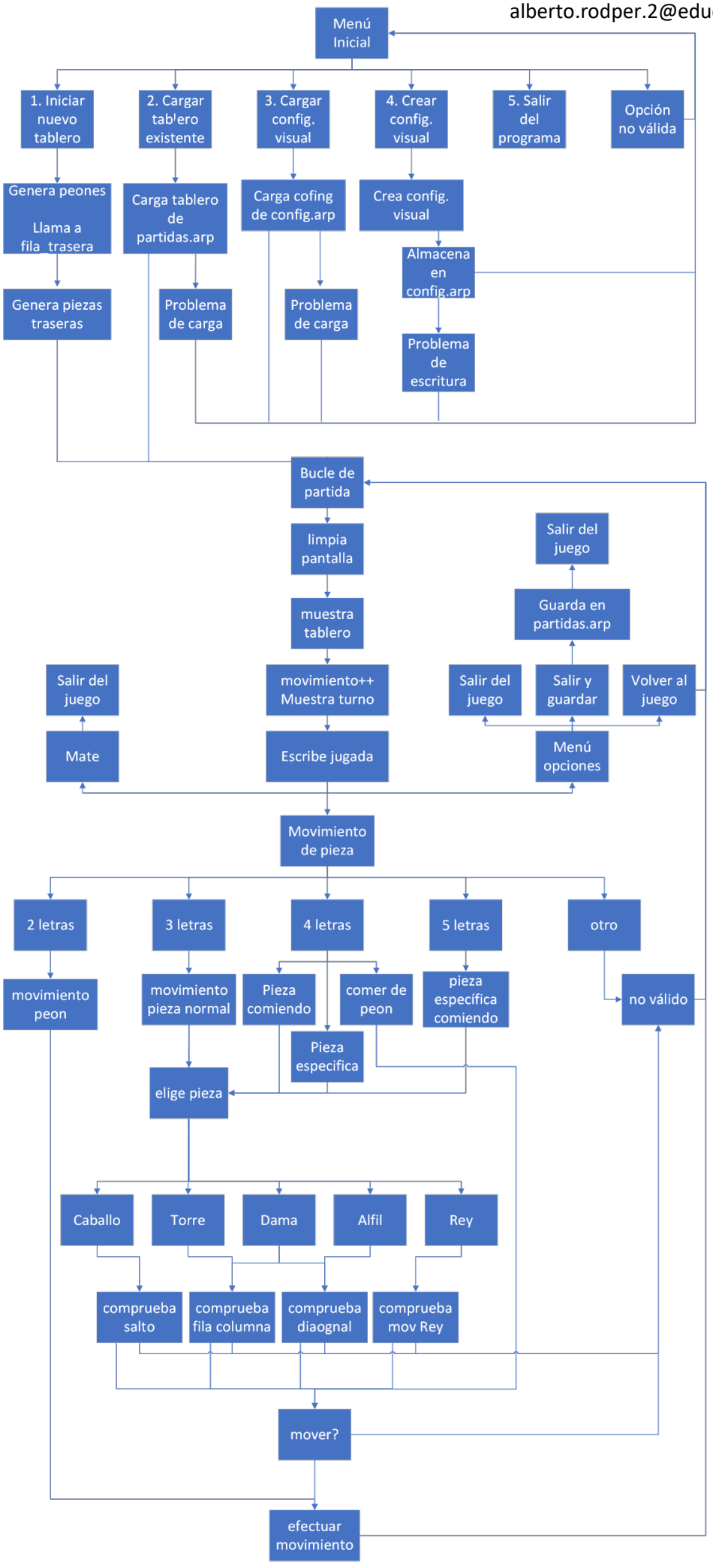
Bienvenido al manual del programador del código “Ajedrez” de Alberto Rodríguez Pérez.

En este manual, se explicará el flujo del programa y sus utilizaciones y funcionalidades para poder facilitar las posibles modificaciones que se quieran hacer.

1. Diagrama de flujo

Lo primero, es ver el flujo del programa, para ello, muestro a continuación un diagrama algo descriptivo sobre esto mismo:

Alberto Rodríguez Pérez
alberto.rodper.2@educa.jcyl.es



Alberto Rodríguez Pérez
alberto.rodper.2@educa.jcyl.es

2. Esquema y definición de clases y métodos

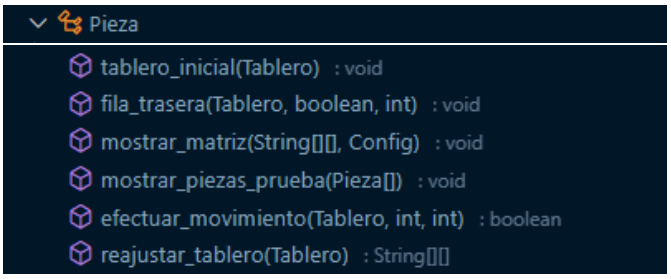
Aquí vamos a explicar el esquema de las clases utilizadas y a justificar su creación.

2.1. Ajedrez

La clase **"Ajedrez"** es la principal, desde aquí se lleva tanto el menú principal, como el flujo principal del programa y las opciones de escritura y lectura de archivos sobre la partida.

- El **main** gestiona el flujo principal.
- El **menú inicial**, muestra una pantalla de bienvenida que nos permite empezar de diferentes formas.
- El **menú de opciones** se usa para salir y guardar una vez la partida ha empezado.
- El método **nuevo movimiento** se encarga de orientar el movimiento solicitado por el usuario en función del tipo de escritura y pieza.
- El método **selecciona pieza** se utiliza para se utiliza para elegir entre las piezas que no son peones y gestionar desde ahí a qué métodos se envían.
- El método **encuentra pieza**, sirve para que, en caso de que el usuario haya especificado una por necesidad, se almacene su índice y se trabaje con él directamente.
- El método **cargar tablero** es el utilizado en caso de que el usuario quiere cargar una partida del archivo "partidas.arp". Muestras las partidas guardadas y lee el objeto "Tablero" almacenado en él, así como los siguientes 32 "Pieza" de esa partida. Invoca a Pieza.reajustar_tablero porque la lectura de la matriz tablero corrompe las posiciones (no se muy bien por qué).
- El método **guardaEstado** se utiliza para almacenar en "partidas.arp" la partida tal y como está, a este método se accede solo desde la opción "Guardar y salir" del **menu opciones.**
- **Limpiar Pantalla** tiene la función de simular un clear screen, depende de la constante "LINEAS".

2.2. Pieza



La clase "**Pieza**" es la utilizada para almacenar tanto las variables relacionadas con cada una de las piezas como los métodos utilizados para su movimiento e interactuación. Es de tipo "serializable" para poder almacenarla en los ficheros.

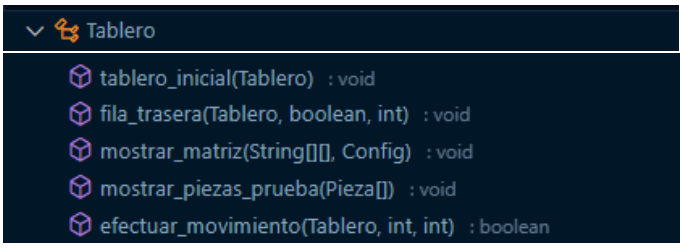
Se utilizan los atributos:

- **Nombre**: Para identificarla y colocarlas en el tablero.
- **Vivo**: Para comprobar su movimiento solo ni no han sido comidas.
- **Mate:** Utilizada solo en los reyes, para definir el bucle y mostrar al ganador.
- **posicionX**: Almacenar del 0 al 7 su posición en las columnas del tablero.
- **posicionY**: Almacenar del 0 al 7 su posición en las filas del tablero.

Los métodos utilizados (todos ellos hacen lo mismo 2 veces, una de ellas para las blancas y otra para las negras):

- **movimiento Peon** se encarga de realizar el avance de los peones, tanto de 1 como de 2 casillas.
- **comerDePeon** hace las comprobaciones para que los peones coman, no está con movimientoPeon porque tiene una movilidad diferente.
- **Movimiento Torre** hace las comprobaciones de las torres y evalúa su movimiento a través del método **comprobar fila columna.**
- **Movimiento Alfil** hace las comprobaciones necesarias de los alfiles y evalúa su movimiento a través de **comprobar diagonal.**
- **Movimiento Caballo** hace las comprobaciones necesarias de los caballos y evalúa su movimiento a través del método **comprobar salto.**
- **Movimiento Dama** hace las comprobaciones necesarias de las damas y evalúa su movimiento a través de **comprobar fila columna y comprobar fila.**
- **Movimiento Rey** comprueba que la casilla a la que se quiere ir está a uno y a través de **comprobar movimiento Rey** evalúa que en esa casilla no le darán jaque.
- **Reajustar tablero** es un método que se invoca solo desde la lectura de partidas, para solucionar un error. Los tableros leídos de los objetos se corrompen (no sé muy bien por qué) y con esto los rehago.

2.3. Tablero



La clase "**Tablero**" es la que se utiliza para almacenar todos los datos relevantes de la partida que se está jugando, sirviendo tanto para el almacenamiento como para evitar pasar decenas de cosas entre los métodos, otorgando visibilidad y claridad al código.

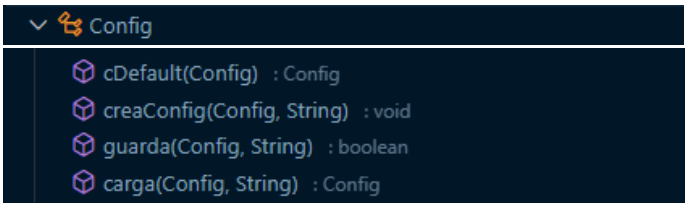
Los atributos, la parte más importante de esta clase, son:

- **Nombre:** Utilizado para distinguir el juego una vez almacenado
- **Tablero:** Matriz 8x8 de tipo cadena para mostrar la inicial de las piezas y agilizar las comprobaciones de ocupación de las casillas.
- **Piezas:** Array de 32 posiciones de tipo Pieza que almacena cada una de las piezas de la partida.
- **nuevaX:** Variable que almacena la coordenada X a la que se quiere ir. Útil para las comprobaciones.
- **nuevaY:** Variable que almacena la coordenada Y a la que se quiere ir. Útil para las comprobaciones.
- **pieza_específica:** Variable tipo int que suele estar en -1, se modifica cuando el usuario especifica un identificador por necesidad y almacena el índice de la pieza específica.
- **comer:** es un boolean que se marca a true cuando se quiere comprobar una captura, y a false cuando es un movimiento normal de avance.
- **Movimiento:** marca el número de la jugada, para saber si es el turno de las blancas o de las negras.

Los métodos utilizados son:

- **Tablero inicial** se encarga de iniciar un tablero y sus piezas como en una partida nueva. Se apoya en **fila trasera.**
- **Fila trasera** crea los objetos que no son peones a través de un switch que genera el tipo de pieza correspondiente.
- **Mostrar matriz** se encarga de hacer la visualización del tablero, aunque requiere de un método de la clase "**Config**" para saber la forma de visualización del tablero y piezas.
- **Efectuar movimiento** es un método al que recurren todos los métodos de comprobación de la clase "**Pieza**" en caso de que quieran consolidar un movimiento y no solo hacer su comprobación (se comprueban solo si se llaman desde comprobar movimiento Rey, de cualquier otra forma de pasa con intenciones de realizarlo).
- **Mostrar piezas prueba** se ha utilizado durante el desarrollo para mostrar las posiciones de las piezas en el tablero y evaluar las funcionalidades de movimiento.

2.4. Config



La clase "**Config**" se utiliza para todo el apartado de la configuración visual, como la visibilidad de las piezas en rojo dependen de que el lugar desde el que se lanza la aplicación permita códigos de color ANSI, y codifique en UTF-8, en vez de establecer la visibilidad del tablero con constantes, se establece en función de lo que marca un objeto de esta clase. Además se permite almacenar configuraciones para dar facilidades al usuario.

Los atributos de esta clase son:

- **Nombre:** Utilizado para poder identificar la configuración una vez almacenada.
- **Esquinas (infDer, infIzq, supDer, supIzq):** almacenan el carácter que se mostrará cuando haga falta imprimir una esquina del tablero-
- **Aristas (arist, aristDer, aristInf, aristIzq, aristSup):** almacenan el carácter que se mostrará cuando haga falta enseñar un punto de unión entre varias líneas.
- **Lineas (lineaVer, lineaHor):** almacena el carácter que se mostrará cuando haga falta enseñar una línea horizontal o vertical.
- **Rojo:** Almacena el código ANSI para mostrar el rojo en caso de que se pueda ver, y un * en caso de que no.
- **Blanco:** Almacena el código ANSI para volver a imprimir en blanco y * en caso de que no se codifiquen los códigos ANSI.

Los métodos utilizados son:

- Se utiliza **cDefault** cuando no se especifica otra por parte del usuario, y se muestra el tablero con caracteres especiales y colores con formato ANSI.
- **creaConfig** es invocado solo en el menú inicial, se utiliza para ir preguntando al usuario que codificaciones ve y cuales no, y utilizar el máximo aspecto gráfico posible a la hora de imprimir el tablero.
- **Guarda** se utiliza para almacenar una configuración del usuario establecida desde **creaConfig** en el fichero "Config.arp".
- **Carga** es invocado solo en el menú inicial, lee y permite elegir una configuración visual almacenada en "config.arp".

2.5. MiObjectOutputStream

Método utilizado para escribir sin cabeza si el archivo ya contiene archivos. Clase heredada de "ObjectOutputStream".

Alberto Rodríguez Pérez
alberto.rodper.2@educa.jcyl.es

3. Explicación de métodos complejos

En este apartado se va a explicar el algoritmo de métodos que puede que no sean muy entendibles cuando se leen al principio debido a que algunos pueden haber quedado un poco liosos por la gran cantidad de funcionalidades.

3.1. Ajedrez: nuevo_movimiento

Lo primero que hace es evaluar que la jugada introducida por el usuario está entre los parámetros de 2 y 5 letras, fuera de eso, no sería una jugada válida.

Se guardan en nuevaX y nuevaY los últimos 2 caracteres, pue siempre se encargan de marcar la localización a la que se quiere ir o comer.

```
partida.nuevaX = (int)(jugada.charAt(jugada.length()-2)-97);
partida.nuevaY = (int)(jugada.charAt(jugada.length()-1)-49);
```

Se resta -97 a la letra (columna) y -49 al número (fila), para almacenarlos entre 0 y 7, acorde con el tablero.

Pasamos a un switch en función de la longitud de la jugada introducida por el usuario:

En caso de 2 letras:

- Marcará movimiento de un peón exclusivamente.

En caso de 3 letras:

- Marcará movimiento de una pieza no peón.

En caso de 4 letras:

- Podrá marcar una pieza no peón comiendo (Txa8)
  - Se activará la variable “comer” a true, y las comprobaciones se harán pensando en que se quiere capturar.

- Podrá marcar un peón comiendo (dxe4)
  - Se llamará al método específico para peones capturando.

- Podrá marcar que se ha pasado un identificar de pieza específica (Cgf3)
  - Se cargará la variable “pieza_específica” con el índice correspondiente y las comprobaciones se harán teniendo en cuenta la existencia de esta misma.

En caso de 5 letras:

- Podrá marcar solo que una pieza específica quiere capturar (T3xf3)
  - Activará “comer” y pondrá el índice necesario en “pieza_específica”

Una vez efectuado el movimiento/comprobación, si se modificó “comer” o “pieza_específica” se pondrán a false o -1 respectivamente, para no contar con ello en el próximo movimiento.

3.2. Ajedrez: encuentra_pieza

Es el método encargado de almacenar el índice de una pieza, cuando el usuario a especificado una debido a que cualquiera de las 2 iguales del mismo color podían ir a la posición indicada.

Lo primero que se hace es definir si el identificar del usuario nos indica una fila o una columna para poder iniciar las comprobaciones correspondientes.

Si nos ha pasado una letra, buscaremos en toda esa columna que pieza que pueda necesitar una especificación (dejando fuera peones, damas y reyes), es la que se encuentra en esa columna y es de ese color.

```
if(escolumna){
    for(int x = 0+zona; x < 16+zona; x++){
        if(x == 3 || x == 4 || (x >=8 && x<=23) || x == 27 || x == 28) continue;

        if(partida.piezas[x].posicionX == (int)(posicion - 97) && partida.piezas[x].nombre.charAt(0) == tipo)
            return x;
    }
```

La variable “zona” se utiliza para no hacer el bucle 2 veces, uno para cada color. Si estamos en el turno blanco, zona será 0 y se buscará entre las primeras 16 piezas, si es el turno negro, zona será 16 y se buscará en las piezas a partir de la posición 16.

Si en vez de la columna nos ha especificado la fila, haremos la misma comprobación, pero en vez de comprobando la posición X de las piezas, comprobando la Y.

Una vez se encuentre, se almacenará el índice y se saldrá.

3.3. Pieza: movimientoPeon

Este método tiene 2 if para comprobar tanto en el caso de las blancas como de las negras, el primero de ellos, comprobará el movimiento normal de 1 casilla de avance. El segundo, comprobará la primera posibilidad de que avance 2 casillas al desplazarse desde su posición inicial, comprobando tanto si está en su fila inicial (2 o 7), y que la casilla intermedia de avance está vacía.

Si las comprobaciones son verdaderas, se llamará a **efectuar_movimiento** para consolidar el avance.

3.4. Pieza: movimientoTorre/Caballo/alfil/Dama/Rey

Estos métodos son en esencia iguales, con distintivos a la hora de comprobar el avance o posibilidad de ejecución, pero se pueden explicar a la vez.

```
if(!partida.comer && partida.tablero[partida.nuevaY][partida.nuevaX] != " ")
    return false;

if(partida.comer && partida.tablero[partida.nuevaY][partida.nuevaX] == " ")
    return false;
```

Lo primero que se hace es definir si se han llamado correctamente o no, si se busca capturar, se comprueba que en la casilla de destino hay una pieza, y si se busca avanzar se comprueba que está vacía (hay un espacio).

Lo siguiente que se hace es crear una variable boolean para marcar si las piezas son válidas, pues se van a comprobar todas las del color. Si la pieza está viva, se llamará a su correspondiente método de comprobación de movimiento, ya sea de filas y columnas, salto o diagonales.

```
if(valido0 && valido7)
    if(partida.pieza_específica != -1)
        pieza = partida.pieza_específica;

    else
        return false;

if(partida.pieza_específica == -1){
    if(valido0) pieza = 0;
    if(valido7) pieza = 7;
}
```

Si las dos posibilidades (si hay 2) son válidas, se devolverá que no se ha podido mover la pieza a menos que esté uno de sus índices en “pieza_específica”, es decir, que el usuario haya escrito un identificativo para saber cuál de las 2 quiere que se mueva.

Si no había una pieza específica y solo una es válida, se guardará su índice en la variable que se enviará a efectuar_movimiento.

```
boolean devolver = false;

if(pieza != -1) devolver = true;

//Si se ha encontrado una pieza y se quiere mover, se envía a efectuar_movimiento
if(mover && devolver){
    devolver = Tablero.efectuar_movimiento(partida, pieza, color);
}
return devolver;
```

Si no se ha especificado una pieza, se devolverá false, si se ha especificado, y nos han dicho que se quiere mover, se enviará la pieza correspondiente a efectuar_movimiento.

En caso del Rey, se comprueba si la casilla a la que se quiere ir está siendo atacada, eviando esa casilla con mover = false a todos los comprobantes de las piezas contrarias. Su casilla se pone como si estuviese vacía para que se pueda comprobar si se mueve a una casilla tapada por el mismo durante la simulación del movimiento.

```
saveRey = partida.tablero[partida.piezas[rey].posicionY][partida.piezas[rey].posicionX];
partida.tablero[partida.piezas[rey].posicionY][partida.piezas[rey].posicionX] = " ";
```

Alberto Rodríguez Pérez
alberto.rodper.2@educa.jcyl.es

Para finalizar, aviso de que se buscaba implementar un enroque y la coronación de las piezas pero que por el tiempo durante estas semanas de exámenes no ha dado tiempo.

Con esto concluye el manual del Ajedrez.

==========

Este código ha sido escrito completamente por Alberto Rodríguez Pérez.

Lo único aplicado no enseñado en el módulo ha sido la utilización de los códigos ANSI para dar color a las piezas negras.

La posición de los caracteres especiales del tablero dentro del código UTF-8, han sido encontradas con un bucle creado de forma personal, no sacados de internet.

Cualquier cosa, el correo de contacto es: alberto.rodper.2@educa.jcyl.es

¡¡Espero que se disfrute del juego !!