MEMORIA DEL AJEDREZ DE:

Eloy Manzano Pachita 2DAM

Contenido

Introducción

En este documento voy a hablar sobre mi ejercicio de hacer, un juego de ajedrez en Python. Los movimientos y métodos que he sido capaz de implementar son: un método llamado *dibujarTablero* que haciendo uso de la librería Pygame de Python dibuja un tablero de ajedrez de 8x8 casillas y coloca cada una de las piezas en su lugar representadas por un png, un método para cada una de las distintas piezas del tablero de ajedrez que controla toda su lógica, un método llamado *moverPieza* que se encarga de determinar si un movimiento es válido en función de la ficha que estemos moviendo, un método guardar movimiento que guarda en un archivo el turno, la pieza y el movimiento que ha hecho la pieza y por ultimo un método *jugar* que ejecuta el bucle principal del juego, controla los turnos, los eventos de Pygame etc.

Primeros pasos

Lo primero que pensé al empezar a hacer mi ejercicio del ajedrez es en que estructura iba a usar para representar el tablero ya que es lo más importante al principio, y lógicamente se me vino a la cabeza un array bidimensional de 8x8 y buscando información me di cuenta de que un array como yo lo conocía no existe en Python por lo que tenía que usar una lista de listas para representarlo pero por suerte el uso es muy parecido al que yo ya conocía sobre los arrays, después poco a poco fui desarrollando mi código, la primera versión que hice fue el juego del ajedrez por consola por eso podéis ver que en mi código hay 3 métodos que finalmente no uso en la versión grafica que hice después pero me apetecía dejarlos en mi código aunque no se usen, después lo que hice como acabo de comentar fue cambiar mi código a una versión grafica en la que se usa la librería de Python Pygame para representar el tablero, las fichas del ajedrez y algunas funciones más que comentare más abajo.

Explicación del código y los métodos que lo forman

Configuración de la ventana y tamaño de las casillas

```
7   #Configuración de la ventana
8   ANCHO = 800
9   ALTO = 800
10  VENTANA = pygame.display.set_mode((ANCHO, ALTO))#Crea una ventana de ancho*alto
11  pygame.display.set_caption("Ajedrez")#Cambia el titulo
```

Lo primero que hago en mi código es crear la ventana grafica en la que se va a representar el tablero y en la que vamos a poder mover las piezas, para ello he creado dos constantes, ancho y alto en las que guardo el tamaño de la ventana, después utilizo la función pygame.display.set_mode para crear la ventana gráfica y almaceno esto en una constante llamada VENTANA, finalmente con pygame.display.set_caption asigno un título a la ventana.

```
13   #Dimensiones de las casillas
14   TAMAÑOCASILLA = ANCHO // 8 #Constante que almacena el tamaño de una casilla que es el ancho del tablero / 8
```

Aquí almaceno en una constante llamada TAMAÑOCASILLA el tamaño que van a tener las casillas que será el ancho total de la ventana entre 8 porque son 8 casillas.

Asignación de colores

```
16   #Colores
17   BLANCO = (255, 255, 255) #Constante que almacena el color blanco
18   GRIS = (210, 210, 210) #Constante que almacena el color gris
19   AZUL = (0, 0, 80) #Constante que almacena el color azul
```

Aquí almaceno en 3 constantes los colores blanco, gris y azul que los necesitare más adelante.

Creación del tablero

```
21   #Tablero
22   tablero = [ #Es un array bidimensional aunque en python es una lista de listas
23       ["t", "c", "a", "q", "k", "a", "c", "t"],
24       ["p", "p", "p", "p", "p", "p", "p", "p"],
25       [" ", " ", " ", " ", " ", " ", " ", " "],
26       [" ", " ", " ", " ", " ", " ", " ", " "],
27       [" ", " ", " ", " ", " ", " ", " ", " "],
28       [" ", " ", " ", " ", " ", " ", " ", " "],
29       ["P", "P", "P", "P", "P", "P", "P", "P"],
30       ["T", "C", "A", "Q", "K", "A", "C", "T"]
31   ]
```

El tablero es una lista de listas rellenada con la inicial de cada ficha, en minúscula las negras y en mayúscula las blancas.

Cargar las imágenes de las fichas y redimensionarlas al tamaño de las casillas

```
33  #Cargar imagenes de las piezas
34  IMAGENESPIEZAS = {
35      #Piezas blancas
36      "P": pygame.image.load("Imagenes/peonBlanco.png"), #Asocia la letra del array a la imagen
37      "T": pygame.image.load("Imagenes/torreBlanca.png"),
38      "A": pygame.image.load("Imagenes/alfilBlanco.png"),
39      "C": pygame.image.load("Imagenes/caballoBlanco.png"),
40      "Q": pygame.image.load("Imagenes/reinaBlanca.png"),
41      "K": pygame.image.load("Imagenes/reyBlanco.png"),
42
43      #Piezas negras
44      "p": pygame.image.load("Imagenes/peonNegro.png"), #Asocia la letra del array a la imagen
45      "t": pygame.image.load("Imagenes/torreNegra.png"),
46      "a": pygame.image.load("Imagenes/alfilNegro.png"),
47      "c": pygame.image.load("Imagenes/caballoNegro.png"),
48      "q": pygame.image.load("Imagenes/reinaNegra.png"),
49      "k": pygame.image.load("Imagenes/reyNegro.png"),
50  }
```

Aquí lo que hago es asociar a cada letra del tablero la imagen que le corresponde usando pygame.image.load y la ruta dentro de mi ordenador y lo guardo en un diccionario llamado *IMAGENESPIEZAS*.

```
52  #Redimensionar imágenes al tamaño de las casillas
53  for clave in IMAGENESPIEZAS: #Recorre todas las claves (letras) de IMAGENESPIEZAS
54      #Redimensiona cada imagen y la guarda en el diccionario
55      IMAGENESPIEZAS[clave] = pygame.transform.scale(IMAGENESPIEZAS[clave], (TAMAÑOCASILLA, TAMAÑOCASILLA))
56
```

Aquí hago un bucle que recorre todas las claves asociadas a las imágenes de las piezas y usando pygame.transform.scale se redimensionan al tamaño de la casilla que he calculado antes y se guardan de nuevo en el diccionario IMAGENESPIEZAS pero ya con su tamaño adecuado.

Primer método: *dibujarTablero*

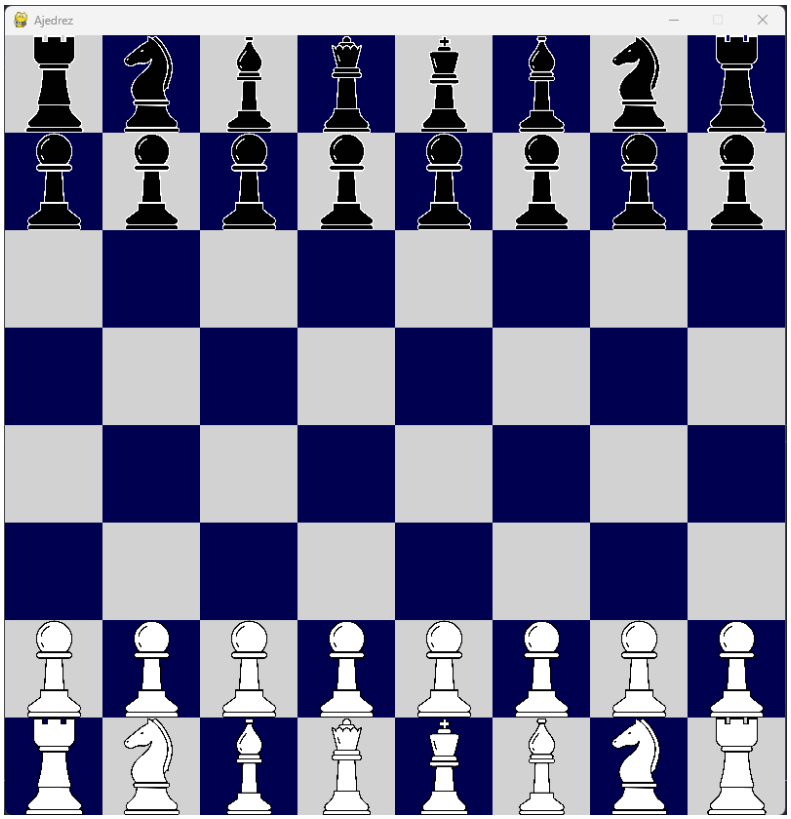
```python
#DibujarTablero
def dibujarTablero(tablero):
    for fila in range(8):#Recorrer el array de 8*8
        for col in range(8):
            #Alternar colores de las casillas
            if (fila + col) % 2 == 0: #si la suma es par
                color = GRIS
            else:
                color = AZUL
            #.rect dibuja un rectangulo en la ventana del color que toque
            pygame.draw.rect(VENTANA, color, (col * TAMAÑOCASILLA, fila * TAMAÑOCASILLA, TAMAÑOCASILLA, TAMAÑOCASILLA))

            # Dibujar piezas
            pieza = tablero[fila][col]#Accede a la pieza concreta de esa posicion del array
            if pieza != " ": #Si hay una pieza en la casilla
                #Se dibuja la imagen correspondiente a esa pieza
                VENTANA.blit(IMAGENESPIEZAS[pieza], (col * TAMAÑOCASILLA, fila * TAMAÑOCASILLA))

    pygame.display.flip()#Actualiza la pantalla

```

La función de este método es mostrar el tablero de ajedrez de manera gráfica, alternando los colores gris y azul para representar las casillas y mostrando las piezas correspondientes en sus posiciones. Primero recorre un array de 8x8 que representa el tablero con dos bucles anidados. Para alternar los colores utilizo una condición que pone el color gris a las casillas pares y el color azul a las casillas impares. Luego con pygame.draw.rect dibuja una casilla como un rectángulo del tamaño y color correspondiente. Para dibujar las piezas primero accedemos a la posición del array y comprobamos que ese hueco no esta vacío, si no lo está, usamos VENTANA.blit para cargar la imagen de esa pieza. Por último, uso pygame.display.flip para actualizar la ventana.



Segundo método: *moverPeon*

```python
#Mover Peon
def moverPeon(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen #Asignación múltiple
    filaDestino, colDestino = destino #Asignación múltiple
    pieza = tablero[filaOrigen][colOrigen] #Guarda en pieza la posición de esta

    if (jugador == "blanco" and pieza != "P") or (jugador == "negro" and pieza != "p"): #Comprueba que sea un peón negro o blanco
        print("La pieza seleccionada no es un peon valido.")
        return False

    if jugador == "blanco": #Si el jugador es blanco
        #Comprueba que solo se ha movido una posición hacia delante, comprueba que no cambia de columna porque solo se puede mover hacia delante y comprueba que la casilla de destino está vacía.
        if (filaDestino == filaOrigen - 1 and colDestino == colOrigen and tablero[filaDestino][colDestino] == " "):
            tablero[filaDestino][colDestino] = "P" #Pone P en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        #Comprueba que la fila de origen es 6 y la de destino es 4, que la columna es la misma, y que las dos posiciones por delante están vacías
        elif (filaOrigen == 6 and filaDestino == 4 and colDestino == colOrigen and tablero[5][colDestino] == " " and tablero[4][colDestino] == " "):
            tablero[filaDestino][colDestino] = "P" #Pone P en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        #Comprueba que solo se ha movido una posición hacia delante, comprueba que solo se mueve una columna a la izquierda o derecha, y comprueba que la casilla de destino tiene una pieza negra
        elif (filaDestino == filaOrigen - 1 and abs(colDestino - colOrigen) == 1 and tablero[filaDestino][colDestino].islower()):
            tablero[filaDestino][colDestino] = "P" #Pone P en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        else:
            print("No puedes comer tu propia pieza.")

    elif jugador == "negro": #Si el jugador es negro
        #Comprueba que solo se ha movido una posición hacia delante, comprueba que no cambia de columna porque solo se puede mover hacia delante y comprueba que la casilla de destino está vacía.
        if filaDestino == filaOrigen + 1 and colDestino == colOrigen and tablero[filaDestino][colDestino] == " ":
            tablero[filaDestino][colDestino] = "p" #Pone p en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        #Comprueba que la fila de origen es 1 y la de destino es 3, que la columna es la misma, y que las dos posiciones por delante están vacías.
        elif filaOrigen == 1 and filaDestino == 3 and colDestino == colOrigen and tablero[2][colDestino] == " " and tablero[3][colDestino] == " ":
            tablero[filaDestino][colDestino] = "p" #Pone p en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        # Comprueba que solo se ha movido una posición hacia delante, comprueba que solo se mueve una columna a la izquierda o derecha, y comprueba que la casilla de destino tiene una pieza blanca.
        elif filaDestino == filaOrigen + 1 and abs(colDestino - colOrigen) == 1 and tablero[filaDestino][colDestino].isupper():
            tablero[filaDestino][colDestino] = "p" #Pone p en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True
        else:
            print("No puedes comer tu propia pieza.")

    print("Movimiento no valido")
    return False
```

El método moverPeon gestiona el movimiento de los peones en el tablero de ajedrez, asegurando que los movimientos realizados sean válidos de acuerdo con las reglas del juego. En primer lugar, se asignan las coordenadas de origen y destino mediante asignación múltiple, y se identifica la pieza seleccionada en la casilla de origen. A continuación, se verifica que el jugador actual esté intentando mover un peón válido: un peón blanco ("P") para el jugador "blanco" o un peón negro ("p") para el jugador "negro". Si no es así, se muestra un mensaje de error y la función finaliza devolviendo False.

Si el jugador es blanco, se evalúan tres posibles movimientos: avanzar una casilla hacia adelante si la posición está vacía, avanzar dos casillas si el peón es la primera vez que se mueve (desde la fila 6) si las dos posiciones están vacías, o capturar una pieza enemiga en diagonal hacia adelante si la casilla contiene una pieza negra. Cada movimiento válido actualiza el tablero moviendo el peón a la posición de destino, dejando vacía la casilla de origen y devolviendo True. Se muestra un mensaje si el jugador intenta capturar su propia pieza en diagonal.

Para el jugador negro se aplican las mismas reglas, pero en la dirección opuesta: avanzar una casilla hacia adelante si está vacía, avanzar dos casillas si el peón es la primera vez que se mueve (desde la fila 1) si las dos posiciones están vacías, o capturar en diagonal una pieza enemiga blanca. Si un movimiento cumple las condiciones, el tablero se actualiza de la misma manera que antes y el método devuelve True. En caso de que no se cumpla ninguna condición, se muestra un mensaje indicando que el movimiento no es válido y se retorna False.

Este método asegura que los peones solo se muevan y coman dentro de los límites permitidos.

Tercer método: *moverTorre*

```python
#Mover Torre
def moverTorre(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen #Asignacion multiple
    filaDestino, colDestino = destino #Asignacion multiple
    pieza = tablero[filaOrigen][colOrigen] #Guarda en pieza la posicion de esta

    if (jugador == "blanco" and pieza != "T") or (jugador == "negro" and pieza != "t"): #Comprueba que sea una torre negra o blanca
        print("La pieza seleccionada no es una torre valida.")
        return False

    if filaOrigen == filaDestino or colOrigen == colDestino: #Comprueba que el movimiento es en linea recta
        if filaOrigen == filaDestino: #Si el movimiento es hacia delante o atras
            if colDestino > colOrigen: #Si el movimiento es hacia la derecha del tablero
                paso = 1
            else: #Si el movimiento es hacia la izquierda del tablero
                paso = -1
            for col in range(colOrigen + paso, colDestino, paso): #Recorre las columnas entre colOrigen y ColDestino hacia arriba o abajo
                if tablero[filaOrigen][col] != " ": #Comprueba si hay algo diferenet a un espacio en blanco en el camino
                    print("Hay una pieza en el camino.")
                    return False
        elif colOrigen == colDestino: #Si el movimiento es hacia un lado
            if filaDestino > filaOrigen: #Si el movimiento es hacia atras
                paso = 1
            else: #Si el movimiento es hacia delante
                paso = -1
            for fila in range(filaOrigen + paso, filaDestino, paso): #Recorre las filas entre filaOrigen y filaDestino hacia un lado o otro
                if tablero[fila][colOrigen] != " ": #Comprueba si hay algo diferente a un espacion en blanco en el camino
                    print("Hay una pieza en el camino.")
                    return False
        piezaDestino = tablero[filaDestino][colDestino] #Guarda en piezaDestino la posicion del tablero
        #Si la pieza de destino es igual a un espacio en blanco o a una pieza del oponente
        if piezaDestino == " " or (jugador == "blanco" and piezaDestino.islower()) or (jugador == "negro" and piezaDestino.isupper()):
            tablero[filaDestino][colDestino] = tablero[filaOrigen][colOrigen] #Reemplaza el contenido de la casilla de destino por la de origen
            tablero[filaOrigen][colOrigen] = " " #Deja la casilla de origen vacia
            return True
        else: #Si no es un espacion en blanco o una pieza del oponente
            print("No puedes comer tu propia pieza.")
            return False
    else: #Si el movimiento no es en linea recta
        print("Movimiento no valido para la torre. Debe moverse en linea recta.")
        return False
```

El método moverTorre gestiona el movimiento de las torres en el tablero de ajedrez, asegurando que los movimientos realizados sean validos de acuerdo con las reglas del juego. En primer lugar, se asignan las coordenadas de origen y destino mediante asignación múltiple y se identifica la pieza seleccionada en el tablero. Luego, se verifica que el jugador actual este intentado mover una torre valida: una torre blanca ("T") para el jugador "blanco" o una torre negra ("t") para el jugador "negro". Si no es válida, se muestra un mensaje y la función devuelve False.

El movimiento de la torre debe ser en línea recta, ya sea horizontal o vertical. Si se cumple esta condición, el método verifica que no haya piezas bloqueando el camino: Si el movimiento es horizontal, recorre las columnas intermedias entre el origen y el destino, comprobando que estén vacías. Si el movimiento es vertical, recorre las filas intermedias de manera similar.

Si encuentra una pieza en el camino, muestra un mensaje indicándolo y devuelve False. Una vez comprobado el camino, se evalúa la casilla de destino. El movimiento es válido si está vacía o contiene una pieza del oponente. En este caso, se actualiza el tablero moviendo la torre a la casilla de destino y dejando vacía la casilla de origen, devolviendo True. Si la casilla de destino contiene una pieza del mismo jugador, se muestra un mensaje indicando que no se puede comer una pieza propia y devuelve False. Por último, si el movimiento no es en línea recta, se muestra un mensaje indicando que la torre solo puede moverse en línea recta y devuelve False. Este método garantiza que las torres sigan las reglas de movimiento establecidas en el ajedrez.

Cuarto método: moverAlfil

```python
#Mover Alfil
def moverAlfil(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen #Asignacion multiple
    filaDestino, colDestino = destino #Asignacion multiple
    pieza = tablero[filaOrigen][colOrigen] #Guarda en pieza la posicion de esta

    if (jugador == "blanco" and pieza != "A") or (jugador == "negro" and pieza != "a"): #Comprueba que sea un alfil del color del jugador
        print("La pieza seleccionada no es un alfil valido.")
        return False

    #Comprueba que el numero de filas movidas sea igual que el de columnas, lo que quiere decir que se esta moviendo en diagonal
    if abs(filaDestino - filaOrigen) == abs(colDestino - colOrigen):
        if filaDestino > filaOrigen: #Si el movimiento es hacia abajo
            pasoFila = 1
        else: #Si el movimiento es hacia arriba
            pasoFila = -1
        if colDestino > colOrigen: #Si el movimiento es hacia la derecha
            pasoCol = 1
        else: #Si el movimiento es hacia la izquierda
            pasoCol = -1
        #Cambia las variables filaActual y colActual a las nuevas posiciones , si paso col es 1 va hacia la derecha y si es -1 va hacia la izquierda
        filaActual, colActual = filaOrigen + pasoFila, colOrigen + pasoCol
        while filaActual != filaDestino and colActual != colDestino: #Mientras la fila de origen y destino sean distintas y la collumna de origen y destino sean distintas:
            if tablero[filaActual][colActual] != " ": #Si 1 posicion actual dentro del bucle es distinta a un espacio en blanco
                print("Hay una pieza en el camino.")
                return False
            filaActual += pasoFila #Suma a fila actual con el valor de pasoFila que sera el valor 1 o -1
            colActual += pasoCol #Suma a  col actual con el valor de pasoCol que sera el valor 1 o -1
        piezaDestino = tablero[filaDestino][colDestino] #Asigna a pieza destino la posicion del tablero
        #Si piezaDestino es igual a un espacio en blanco, y el jugador es blanco y la pieza de destino negra o el jugador es negro y la ficha de destino blanca
        if piezaDestino == " " or (jugador == "blanco" and piezaDestino.islower()) or (jugador == "negro" and piezaDestino.isupper()):
            tablero[filaDestino][colDestino] = tablero[filaOrigen][colOrigen] #Reemplaza el contenido de la casilla de destino por la de origen
            tablero[filaOrigen][colOrigen] = " " #Pone en la posicion de origen un espacio en blanco
            return True
        else: #Si el jugador es blanco y la posicion de destino es blanca o lo mismo con negras
            print("No puedes comer tu propia pieza.")
            return False
    else: #Si el numero de filas no es igual al numero de columans quiere decir que no se esta moviendo en diagonal
        print("Movimiento no valido para el alfil. Debe moverse en diagonal.")
        return False
```

El método moverAlfil gestiona el movimiento de los alfiles en el tablero de ajedrez, verificando que cumplan con las reglas del juego. En primer lugar, asigna las coordenadas de origen y destino mediante asignación múltiple y obtiene la pieza seleccionada en el tablero. Luego, se asegura de que la pieza sea un alfil válido: "A" para el jugador "blanco" o "a" para el jugador "negro". Si no lo es, muestra un mensaje indicando el error y devuelve False.

El alfil debe moverse en diagonal, lo que significa que el número de filas y columnas desplazadas debe ser igual. Si se cumple esta condición, se determinan las direcciones del movimiento (hacia arriba o abajo, y hacia la izquierda o derecha) mediante incrementos (1) o decrementos (-1). A partir de esto, se recorre el camino desde la casilla de origen hasta la de destino, verificando que no haya piezas bloqueando el movimiento. Si encuentra un obstáculo, muestra un mensaje y devuelve False.

Una vez verificado el camino, se evalúa la casilla de destino. El movimiento es válido si está vacía o contiene una pieza del oponente. En ese caso, se actualiza el tablero moviendo el alfil a la casilla de destino y dejando vacía la casilla de origen, devolviendo True. Si la casilla de destino contiene una pieza del mismo jugador, se muestra un mensaje indicando que no se puede comer una pieza propia y devuelve False. Por último, si el movimiento no es en diagonal, se muestra un mensaje indicando que el alfil solo puede moverse en diagonal y devuelve False. Este método asegura que los movimientos del alfil sean coherentes con las reglas del ajedrez.

Quinto método: moverCaballo

```python
#Mover Caballo
def moverCaballo(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen #Asignacion multiple
    filaDestino, colDestino = destino #Asignacion multiple
    pieza = tablero[filaOrigen][colOrigen] #Guarda en pieza la posicion de esta

    #Comprueba que el jugador es blanco y la pieza no es un caballo blanco y lo mismo para el color negro
    if (jugador == "blanco" and pieza != "C") or (jugador == "negro" and pieza != "c"):
        print("La pieza seleccionada no es un caballo valido.")
        return False

    #Esto es una lista de tuplas que almacena los movimientos validos para el caballo
    movimientosValidos = [(2, 1), (2, -1), (-2, 1), (-2, -1), (1, 2), (1, -2), (-1, 2), (-1, -2)]

    #Comprueba si alguno de los movimientos validos coincide con el movimiento realizado
    if any((filaDestino == filaOrigen + df and colDestino == colOrigen + dc) for df, dc in movimientosValidos):
        piezaDestino = tablero[filaDestino][colDestino] #Guarda en piezaDestino la posicion del tablero
        #Si piezaDestino es igual a un espacio en blanco, y el jugador es blanco y la pieza de destino negra o el jugador es negro y la ficha de destino blanca
        if piezaDestino == " " or (jugador == "blanco" and piezaDestino.islower()) or (jugador == "negro" and piezaDestino.isupper()):
            tablero[filaDestino][colDestino] = tablero[filaOrigen][colOrigen] #Reemplaza el contenido de la casilla de destino por la de origen
            tablero[filaOrigen][colOrigen] = " " #Pone en la posicion de origen un espacio en blanco
            return True
        else: #Si el jugador es blanco y la posicion de destino es blanca o lo mismo con negras
            print("No puedes comer tu propia pieza.")
            return False
    else: #Si el movimiento realizado no coincide con ninguno de los movimientos validos para el caballo
        print("Movimiento no valido para el caballo.")
        return False

```

El método moverCaballo gestiona el movimiento de los caballos en el tablero de ajedrez, validando que los movimientos realizados cumplan con las reglas del juego. Comienza asignando las coordenadas de origen y destino mediante asignación múltiple y obtiene la pieza seleccionada en el tablero. Luego, verifica que la pieza seleccionada corresponda a un caballo válido: un caballo blanco ("C") si el jugador es "blanco" o un caballo negro ("c") si el jugador es "negro". Si no es válida, muestra un mensaje y devuelve False.

A continuación, define una lista de movimientos válidos para el caballo, que son sus posibles desplazamientos en forma de "L". Utiliza la función any() para comprobar si el movimiento realizado coincide con alguno de estos desplazamientos válidos. Si el movimiento es válido, evalúa la casilla de destino: el movimiento es correcto si la casilla está vacía o contiene una pieza del oponente. En ese caso, actualiza el tablero moviendo el caballo a la nueva posición y dejando la anterior vacía, devolviendo True. Si la casilla contiene una pieza del mismo jugador, muestra un mensaje indicando que no se puede comer una pieza propia y devuelve False.

Por otro lado, si el movimiento no coincide con ninguno de los desplazamientos válidos del caballo, muestra un mensaje indicando que el movimiento no es válido para el caballo y devuelve False. Este método garantiza que los caballos respeten sus reglas de movimiento específicas en el ajedrez.

Sexto método: moverRey

```python
# Mover Rey
def moverRey(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen #Asignación múltiple
    filaDestino, colDestino = destino #Asignación múltiple
    pieza = tablero[filaOrigen][colOrigen] #Guarda en pieza la posición de esta

    if (jugador == "blanco" and pieza != "K") or (jugador == "negro" and pieza != "k"): #Comprueba que sea un rey del color del jugador
        print("La pieza seleccionada no es un rey válido.")
        return False

    #Calcula la distancia de movimiento en filas y columnas
    diferenciaFila = abs(filaDestino - filaOrigen) #Calcula el valor absoluto
    diferenciaColumna = abs(colDestino - colOrigen)

    #Comprueba que el rey se mueve solo una casilla en cualquier dirección
    if diferenciaFila <= 1 and diferenciaColumna <= 1:
        piezaDestino = tablero[filaDestino][colDestino] #Asigna a piezaDestino el contenido de la casilla destino
        #Comprueba que la casilla de destino está vacía o contiene una pieza enemiga
        if piezaDestino == " " or (jugador == "blanco" and piezaDestino.islower()) or (jugador == "negro" and piezaDestino.isupper()):
            tablero[filaDestino][colDestino] = pieza #Coloca el rey en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True  # Retorna verdadero
        else:
            print("No puedes comer tu propia pieza.") #Agrega el mensaje para no comer la propia pieza
            return False #Retorna falso si el jugador intenta comer su propia pieza
    else:
        #Si el movimiento no es válido, imprime un mensaje de error
        print("Movimiento no válido para el rey.")
        return False
```

El método moverRey controla el movimiento del rey en el tablero de ajedrez, asegurándose de que cumple con las reglas específicas de esta pieza. Comienza asignando las coordenadas de origen y destino mediante asignación múltiple y recupera la pieza seleccionada en el tablero. Luego, verifica si la pieza seleccionada es un rey válido: un rey blanco ("K") si el jugador es "blanco" o un rey negro ("k") si el jugador es "negro". Si no es válida, muestra un mensaje de error y retorna False.

El método calcula la distancia de movimiento del rey en filas y columnas utilizando valores absolutos para determinar el desplazamiento. A continuación, comprueba si el rey se mueve una única casilla en cualquier dirección, ya sea horizontal, vertical o diagonal. Si este requisito se cumple, evalúa la casilla de destino. El movimiento es válido si la casilla está vacía o contiene una pieza del oponente. En este caso, actualiza el tablero colocando el rey en la nueva posición y dejando vacía la casilla de origen, devolviendo True.

Si la casilla de destino contiene una pieza del mismo jugador, muestra un mensaje indicando que no se puede capturar una pieza propia y retorna False. Por otro lado, si el movimiento no respeta las reglas del rey (es decir, no es de una casilla en cualquier dirección), se muestra un mensaje indicando que el movimiento no es válido y retorna False. Este método asegura que los movimientos del rey sean realizados correctamente según las reglas del ajedrez.

Séptimo método: moverReina

```python
# Mover Reina
def moverReina(tablero, origen, destino, jugador):
    filaOrigen, colOrigen = origen  #Asignación múltiple
    filaDestino, colDestino = destino  #Asignación múltiple
    pieza = tablero[filaOrigen][colOrigen]  #Guarda en pieza la posición de esta

    if (jugador == "blanco" and pieza != "Q") or (jugador == "negro" and pieza != "q"): #Comprueba que sea una reina blanca o negra
        print("La pieza seleccionada no es una reina válida.")
        return False

    #Calcula las diferencias de movimiento
    diferenciaFila = abs(filaDestino - filaOrigen)
    diferenciaColumna = abs(colDestino - colOrigen)

    #Comprueba si el movimiento es válido para una reina
    if diferenciaFila == diferenciaColumna or filaOrigen == filaDestino or colOrigen == colDestino:
        pasoFila = 0 #Inicializa el paso en filas
        pasoColumna = 0 #Inicializa el paso en columnas

        #Define los pasos según la dirección del movimiento
        if filaOrigen != filaDestino:
            pasoFila = 1 if filaDestino > filaOrigen else -1
        if colOrigen != colDestino:
            pasoColumna = 1 if colDestino > colOrigen else -1

        #Verifica que no haya piezas en el camino
        filaActual, colActual = filaOrigen + pasoFila, colOrigen + pasoColumna
        while filaActual != filaDestino or colActual != colDestino:
            if tablero[filaActual][colActual] != " ":
                print("Hay una pieza en el camino.")
                return False
            filaActual += pasoFila
            colActual += pasoColumna

        #Comprueba la casilla de destino
        piezaDestino = tablero[filaDestino][colDestino] #Asigna a pieza destino el contenido de la casilla destino
        if piezaDestino == " " or (jugador == "blanco" and piezaDestino.islower()) or (jugador == "negro" and piezaDestino.isupper()):
            tablero[filaDestino][colDestino] = pieza #Coloca la reina en la posición de destino
            tablero[filaOrigen][colOrigen] = " " #Deja vacía la posición de origen
            return True  #Retorna verdadero
        else: #Si la pieza destino es del mismo color, no puedes comerla
            print("No puedes comer tu propia pieza.")
            return False
    else:
        #Si el movimiento no es válido, imprime un mensaje de error
        print("Movimiento no válido para la reina.")
        return False
```

El método moverReina gestiona el movimiento de la reina en el tablero, verificando que los movimientos sean válidos conforme a las reglas de esta pieza. Comienza asignando las coordenadas de origen y destino mediante asignación múltiple y recupera la pieza seleccionada en el tablero. A continuación, valida que la pieza seleccionada sea una reina válida: una reina blanca ("Q") si el jugador es "blanco" o una reina negra ("q") si el jugador es "negro". Si no es válida, muestra un mensaje de error y retorna False.

El método calcula las diferencias absolutas entre las filas y columnas de origen y destino para determinar el tipo de movimiento. Luego, verifica si el movimiento es válido para la reina: debe ser en línea recta (horizontal o vertical) o en diagonal (misma distancia en filas y columnas). Si esta condición se cumple, se inicializan los pasos (pasoFila y pasoColumna) para recorrer el camino que la reina debe seguir. Dependiendo de la dirección del movimiento, los pasos se ajustan para avanzar o retroceder. A continuación, un bucle recorre todas las casillas entre el origen y el destino. Si alguna casilla intermedia no está vacía, el movimiento es inválido, se muestra un mensaje indicando que hay una pieza en el camino y retorna False.

Si el camino está despejado, el método comprueba la casilla de destino. El movimiento es válido si la casilla de destino está vacía o contiene una pieza enemiga. En este caso, actualiza el tablero moviendo la reina a la nueva posición y dejando vacía la casilla de origen, devolviendo True.

Si la casilla de destino contiene una pieza del mismo jugador, se muestra un mensaje indicando que no se puede capturar una pieza propia y devuelve False. Por último, si el movimiento no cumple con las reglas de la reina, se imprime un mensaje de error indicando que el movimiento no es válido y devuelve False. Este método asegura que los movimientos de la reina se realicen correctamente según las reglas del ajedrez.

Octavo método: moverPieza

```python
#Mover Pieza
def moverPieza(tablero, origen, destino, jugador):
    if origen is None or destino is None: #Comprueba que origen y destino no sean nulos
        print("Movimiento no valido")
        return False
    filaOrigen, colOrigen = origen #Se guardan los valores filaOrigen y colOrigen
    pieza = tablero[filaOrigen][colOrigen]

    if pieza == " ": #Comprueba que la pieza de origen no sea un espacio en blanco es decir que la casilla no este vacia
        print("No hay una pieza en la posicion de origen.")
        return False

    #Si el jugador y blanco y la pieza seleccionada es minuscula devuelve falso y da un mensaje de error
    if jugador == "blanco" and pieza.islower():
        print("La pieza seleccionada no pertenece al jugador BLANCO.")
        return False
    #Si el jugador es negro y la pieza seleccionada es mayuscula devuelve false y da un mensaje de error
    elif jugador == "negro" and pieza.isupper():
        print("La pieza seleccionada no pertenece al jugador NEGRO.")
        return False

    #Comprobación de la pieza y llamada al método de cada una de ellas pasandole el tablero, la casilla de origen de destino y el jugador
    if pieza.lower() == "p": #Con el .lower ignorarmos que la pieza sea minuscula o mayuscula ya que eso lo hemos comprobado arriba
        return moverPeon(tablero, origen, destino, jugador)
    elif pieza.lower() == "t":
        return moverTorre(tablero, origen, destino, jugador)
    elif pieza.lower() == "a":
        return moverAlfil(tablero, origen, destino, jugador)
    elif pieza.lower() == "c":
        return moverCaballo(tablero, origen, destino, jugador)
    elif pieza.lower() == "k":  # Nueva comprobación para el rey
        return moverRey(tablero, origen, destino, jugador)
    elif pieza.lower()== "q":
        return moverReina(tablero, origen, destino, jugador)

    print("Movimiento no implementado para esta pieza.")
    return False
```

El método moverPieza es una función que valida y dirige el movimiento de cualquier pieza de ajedrez en el tablero, asegurando que solo se intenten movimientos válidos. Primero, comprueba que las coordenadas de origen y destino no sean nulas y que la casilla de origen no esté vacía, devolviendo un mensaje de error si estas condiciones no se cumplen.

Luego, valida que la pieza seleccionada pertenezca al jugador actual, basándose en el caso de las letras: si el jugador es "blanco", la pieza debe ser mayúscula, y si es "negro", la pieza debe ser minúscula. Si no se cumple esta condición, devuelve un mensaje de error específico.

Tras estas comprobaciones iniciales, identifica el tipo de pieza en la casilla de origen y llama al método correspondiente para gestionar su movimiento (moverPeon, moverTorre, moverAlfil, moverCaballo, moverRey, o moverReina). La función distingue entre tipos de piezas ignorando si son mayúsculas o minúsculas mediante el uso de lower().

Si no se reconoce la pieza, imprime un mensaje indicando que el movimiento no está implementado y devuelve False.

Noveno método: guardarMovimiento

```python
def guardarMovimiento(turno, origen, destino, pieza):
    #Abrir o crear el archivo 'movimientos.txt' para agregar los movimientos
    with open("movimientos.txt", "a") as archivo:
        archivo.write("¡Partida comenzada!")
        #Escribir el movimiento en el archivo en el formato: Turno, pieza, origen, destino
        archivo.write("Turno: "+str(turno)+" Pieza: " +str(pieza)+ " Origen: "+str (origen)+" Destino: "+str (destino)+"\n")

```

El método guardarMovimiento permite guardar en un archivo llamado movimientros.txt el turno, el origen, el destino y la pieza de todos los movimientos que se realizan en la partida.

Decimo método: jugar

```python
#Jugar
def jugar():
    turno = "blanco" #Se pone el primer turno a blancas ya que siempre se suele empezar por este turno las partidas
    print("Turno del "+turno.upper()) #Imprimo el primer turno por consola para saber a quien le toca en cada momento
    corriendo = True #Es una variable bandera que controla cuando funciona el bucle while
    origen = None #Inicializo la varibale a nulo
    destino = None #Inicializo la varibale a nulo

    #with open("movimientos.txt","w") as archivo:
        #archivo.write("¡La partida ha comenzado! \n")

    while corriendo: #El bucle se ejcuta mientras qie la variable corriendo no se cambie a false
        dibujarTablero(tablero) #Llama al metodo dibujar tablero y le pasa por parametro el tablero

        for evento in pygame.event.get(): #Este bucle captura los eventos de pygame
            if evento.type == pygame.QUIT: #Si el evento es cerrar la ventana se detiene el juego
                corriendo = False #La variable bandera corriendo pasa a ser False
                pygame.quit() #Cierra pygame
                return
            elif evento.type == pygame.MOUSEBUTTONDOWN: #Esto maneja que el evento sea el clic del raton
                pos = pygame.mouse.get_pos() #Obtiene la posicion del raton en la ventana y lo guarda en la variable pos en forma de coordenadas
                #Convierte las coordenadas de pixeles de la anterior variable en filas y columnas del tablero dividiendo las coordenadas por el tamaño de las casillas
                fila, col = pos[1] // TAMAÑOCASILLA, pos[0] // TAMAÑOCASILLA
                if origen is None: #Si origen es none
                    origen = (fila, col) #Se asigna los valores del primer clic a la variable origen

                else: #Si ya se ha seleccionado un origen el segundo clic se guarda como destino
                    destino = (fila, col)

                    #Llama al metodo moverPieza y si devuelve true que quiere decir que se ha ejecutado cambia el turno.
                    if moverPieza(tablero, origen, destino, turno):
                        pieza = tablero[fila][col]  # Determinar la pieza que se movió
                        guardarMovimiento(turno, origen, destino, pieza)
                        if turno == "blanco":  #Cambia el turno depende de cual este
                            turno = "negro"
                        else:
                            turno = "blanco"
                        print()
                        #print("La pieza se ha movido de " + str(origen) + " a " + str(destino))
                        print()
                        print ("Turno del "+turno.upper()) #Imprime el proximo turno
                        print()

                    print("La pieza se ha movido de " + str(origen) + " a " + str(destino))


                    #with open("movimientos.txt","a") as archivo:
                        #archivo.write("Turno: "+turno+ " Origen: "+str(origen)+" Destino:"+str(destino)+"\n")

                    #else:
                        #print("Es el turno del jugador "+turno.upper())

                    #Se reestablecen las variables origen y destino para el proximo movimiento
                    origen = None
                    destino = None

    pygame.quit() #Cierra la ventana de pygame y termina el programa

#Llamar al metodo jugar
jugar()
```

El método jugar gestiona el flujo principal de la partida de ajedrez, controlando los turnos de los jugadores y permitiendo que se realicen los movimientos a través de la interacción con la interfaz de Pygame. El juego comienza con el turno del jugador blanco, y alterna entre los jugadores tras cada movimiento.

Primero, se establece que el turno inicial es del jugador blanco y se imprime en consola. Luego, el bucle principal (while corriendo) mantiene el juego en ejecución mientras la variable corriendo sea True. Dentro de este bucle, se dibuja el tablero en la pantalla y se capturan los eventos generados por el jugador mediante Pygame.

Cuando un jugador hace clic en una casilla del tablero, el método obtiene las coordenadas del clic y las convierte en índices de filas y columnas. Si aún no se ha seleccionado un origen (origen es None), el primer clic lo asigna como el origen. El segundo clic selecciona el destino.

Si se ha seleccionado tanto un origen como un destino, se llama al método moverPieza para intentar mover la pieza. Si el movimiento es válido, se guarda el movimiento en el archivo movimientos.txt utilizando el método guardarMovimiento, y luego se alterna el turno entre los jugadores, cambiando el turno de "blanco" a "negro" o viceversa.

Finalmente, las variables origen y destino se restablecen a None para esperar un nuevo movimiento. Si se detecta que el jugador ha cerrado la ventana de Pygame, el bucle se detiene y se cierra el juego.

Ejemplo de la ejecución