Ajedrez de Alicia

Integrantes

Alejandro Marin Hoyos - 2259353-3743
Kevin Alexander Marin Henao - 2160364 - 3743
Carlos Alberto Camacho - 2160331 - 3743
Harrison Inney Valencia - 2159979 - 3743
Manuel Antonio Vidales Duran - 2155481 - 3743
Juan David Valencia Montalvo - 2160103 - 3743

Inteligencia Artificial

Universidad del Valle
Tuluá, Valle del Cauca
Octubre 2024

Interfaz gráfica

ARCHIVO constantes.py CLASE Pieza

constantes.py

```python
class Pieza:
    PEON = 'PEON'
    TORRE = 'TORRE'
    CABALLO = 'CABALLO'
    ALFIL = 'ALFIL'
    DAMA = 'DAMA'
    REY = 'REY'

class Color:
    BLANCO = 'BLANCO'
    NEGRO = 'NEGRO'
```

Define las constantes fundamentales utilizadas en el desarrollo de un juego de ajedrez. Estas constantes representan los tipos de piezas y los colores disponibles en el juego. Su propósito es facilitar la comprensión del código, mejorar la legibilidad y evitar errores al referenciar piezas o colores.

Clases y constantes

Clase: Pieza

La clase Pieza agrupa las constantes que representan los diferentes tipos de piezas de ajedrez. Cada pieza está asociada a una cadena de texto que identifica claramente su tipo.

PEÓN: Representa al peón, la pieza más básica del ajedrez.
TORRE: Representa a la torre, una pieza de largo alcance que se mueve en líneas rectas horizontales o verticales.
CABALLO: Representa al caballo, única pieza que puede saltar sobre otras y se mueve en forma de "L".
ALFIL: Representa al alfil, que se mueve exclusivamente en diagonales.
DAMA: Representa a la dama, la pieza más poderosa con movimientos combinados de torre y alfil.
REY: Representa al rey, la pieza central del juego cuya captura o amenaza pone fin a la partida.

Clase: Color

La clase Color agrupa las constantes que representan los colores de las piezas y los jugadores. Estas constantes definen los dos bandos que participan en una partida de ajedrez.

BLANCO: Representa el color blanco, asociado a un jugador y sus piezas.
NEGRO: Representa el color negro, asociado al oponente y sus piezas.

ARCHIVO IA.py

método __init__(self, color, profundidad=4):

```
def __init__(self, color, profundidad=4):
    Constructor de la clase IA (Inteligencia Artificial).
    :param color: El color de la IA (BLANCO o NEGRO).
    :param profundidad: La profundidad máxima para la búsqueda Minimax.
    """
    self.color = color
    self.profundidad = profundidad
    # Tablas de posición para evaluar la ubicación de las piezas
    self.tablas_posicion = {
        Pieza.PEON: [
            [0,  0,  0,  0,  0,  0,  0,  0],
            [50, 50, 50, 50, 50, 50, 50, 50],
            [10, 10, 20, 30, 30, 20, 10, 10],
            [5,  5, 10, 25, 25, 10,  5,  5],
            [0,  0,  0, 20, 20,  0,  0,  0],
            [5, -5,-10,  0,  0,-10, -5,  5],
            [5, 10, 10,-20,-20, 10, 10,  5],
            [0,  0,  0,  0,  0,  0,  0,  0]
        ],
        Pieza.CABALLO: [
            [-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]
        ],
```

El constructor de la clase IA inicializa los atributos necesarios para implementar una inteligencia artificial en un juego de ajedrez. Recibe dos parámetros:

1. **color**: Especifica el color de las piezas controladas por la IA, que puede ser BLANCO o NEGRO.
2. **profundidad**: Es un parámetro opcional que define la profundidad máxima para la búsqueda en el algoritmo Minimax. Representa el número de jugadas (niveles) que la IA evaluará. Su valor por defecto es 4.

El constructor asigna estos valores a los atributos **self.color y self.profundidad.**

Además, inicializa un diccionario llamado **self.tablas_posicion**, que contiene tablas de posición específicas para cada tipo de pieza del ajedrez: peón, caballo, alfil, torre, dama y rey. Estas tablas son representaciones del tablero de ajedrez de 8x8 y asignan valores

numéricos a cada posición. Los valores positivos indican posiciones más favorables para la pieza, mientras que los valores negativos corresponden a posiciones menos favorables.

Las tablas son utilizadas como heurísticas para evaluar la calidad de las posiciones de las piezas en el tablero. De esta forma, la IA puede tomar decisiones basadas en la ubicación estratégica de las piezas.

Por ejemplo:

- **Los peones** tienen más valor cuando avanzan hacia filas más altas, ya que se acercan a la promoción.
- **Los caballos** son más efectivos cuando están en posiciones centrales del tablero, donde tienen más movilidad.
- **El rey** tiene valores que favorecen su protección al inicio del juego y su movimiento central en el final de la partida.

método evaluar_tablero(self, tablero):

```
def evaluar_tablero(self, tablero):
    """
    Evalúa el estado del tablero para determinar qué tan favorable es para la IA.
    :param tablero: El estado actual del tablero de ajedrez.
    :return: Un valor numérico que representa la calidad del tablero desde la perspectiva de la IA.
    """
    valor = 0
    # Valores de las piezas (aproximados por su poder relativo)
    valores_piezas = {
        Pieza.PEON: 100,
        Pieza.CABALLO: 320,
        Pieza.ALFIL: 330,
        Pieza.TORRE: 500,
        Pieza.DAMA: 900,
        Pieza.REY: 20000
    }
```

```
for tablero_num in [1, 2]:  # Iterar por ambos tableros (uno para cada jugador)
    for fila in range(8):  # Iterar sobre las filas
        for columna in range(8):  # Iterar sobre las columnas
            pieza = tablero.obtener_pieza(tablero_num, fila, columna)  # Obtener la pieza en la posic
            if pieza:
                valor_base = valores_piezas[pieza[0]]  # Obtener el valor base de la pieza
                multiplicador = 1 if pieza[1] == self.color else -1  # Determinar si la pieza es alia

                # Evaluar el valor material
                valor += valor_base * multiplicador

                # Evaluar el valor posicional de la pieza
                fila_eval = fila if pieza[1] == Color.BLANCO else 7 - fila  # Ajustar fila según el c
                valor_posicion = self.tablas_posicion[pieza[0]][fila_eval][columna]
                valor += valor_posicion * multiplicador

                # Bonificaciones adicionales para el peón
                if pieza[0] == Pieza.PEON:
                    # Penalización por peones doblados
                    peones_columna = sum(1 for f in range(8)
                                         if tablero.obtener_pieza(tablero_num, f, columna) == pieza)
                    if peones_columna > 1:
                        valor -= 20 * multiplicador
```

```
                    if peones_columna > 1:
                        valor -= 20 * multiplicador

                    # Penalización por peones aislados
                    peones_adyacentes = False
                    for c in [columna-1, columna+1]:
                        if 0 <= c < 8:
                            for f in range(8):
                                if tablero.obtener_pieza(tablero_num, f, c) == pieza:
                                    peones_adyacentes = True
                                    break
                    if not peones_adyacentes:
                        valor -= 30 * multiplicador
```

```
# Evaluar la seguridad del rey
for tablero_num in [1, 2]:
    rey_encontrado = False
    for fila in range(8):
        for columna in range(8):
            pieza = tablero.obtener_pieza(tablero_num, fila, columna)
            if pieza and pieza[0] == Pieza.REY and pieza[1] == self.color:
                rey_encontrado = True
                # Penalizar si el rey está en jaque
                if tablero.esta_en_jaque(fila, columna, tablero_num):
                    valor -= 100
                break
        if rey_encontrado:
            break

return valor
```

La función **evaluar_tablero** realiza una evaluación del estado actual del tablero de ajedrez y devuelve un valor numérico que representa qué tan favorable es la posición para la inteligencia artificial (IA).

Parámetros

- tablero:
  Es una representación del estado actual del tablero de ajedrez, que incluye las posiciones de las piezas de ambos jugadores.

Retorna

- int:
  Un valor numérico que representa la evaluación total del tablero.
- Los valores positivos indican que la posición es favorable para la IA.
- Los valores negativos indican una posición favorable para el oponente.

Descripción del funcionamiento

La evaluación combina múltiples factores como el material, la posición de las piezas y la seguridad del rey. La evaluación general se calcula iterando sobre cada posición del tablero y evaluando las piezas presentes.

- **Valores de las piezas**
  Se definen valores numéricos aproximados para cada tipo de pieza, basados en su poder relativo en ajedrez:

  Peón: 100

  Caballo: 320

  Alfil: 330

  Torre: 500

  Dama: 900

  Rey: 20000

- **Evaluación del material y la posición**
  Para cada posición en el tablero:

  Se obtiene el valor base de la pieza.

  Se multiplica por +1 si la pieza pertenece a la IA, y por -1 si pertenece al oponente.

Se agrega un valor adicional basado en la posición de la pieza, utilizando las tablas de posición correspondientes.

Para los peones, se aplican penalizaciones adicionales:

Peones doblados: Se penaliza si hay más de un peón en la misma columna.

Peones aislados: Se penaliza si no hay peones aliados en columnas adyacentes.

- **Seguridad del rey**
  La función también evalúa la seguridad del rey:

  Se penaliza con -100 si el rey está en jaque.

  Se verifica únicamente la posición del rey aliado.

Proceso general

1. Iterar sobre ambos tableros (tablero 1 y tablero 2, para ambos jugadores).
2. Analizar cada casilla del tablero:

   Obtener la pieza que ocupa la posición.

   Evaluar su valor base, valor posicional y aplicar bonificaciones o penalizaciones especiales.

3. Verificar la seguridad del rey.
4. Sumar los valores calculados y retornar la evaluación final.

Ejemplo de comportamiento

Si el tablero tiene más material y mejores posiciones para la IA, la función retornará un valor positivo. Si el oponente tiene ventaja, el valor será negativo.

Casos especiales

- Peones doblados: Peones que ocupan la misma columna son penalizados.
- Peones aislados: Peones sin apoyo en columnas adyacentes reciben penalización adicional.
- Rey en jaque: Se agrega una penalización importante para indicar el peligro inmediato de la posición del rey..

Minimax

Para gestionar el minimax de la aplicación se toma en cuenta los siguientes elementos de la clase IA

```python
def minimax(self, tablero, profundidad, alfa, beta, es_maximizador, movimiento_anterior=None):
    if profundidad == 0:
        return self.evaluar_tablero(tablero)
    movimientos = self.obtener_todos_movimientos(tablero,
                  self.color if es_maximizador else
                  (Color.NEGRO if self.color == Color.BLANCO else Color.BLANCO))

    movimientos = self.ordenar_movimientos(tablero, movimientos)
    if es_maximizador:
        mejor_valor = float('-inf')
        for movimiento in movimientos:
            tablero_temp = tablero.copiar_tablero()
            if tablero_temp.realizar_movimiento(movimiento):
                valor = self.minimax(tablero_temp, profundidad - 1, alfa, beta, False, movimiento)
                mejor_valor = max(mejor_valor, valor)
                alfa = max(alfa, mejor_valor)
                if beta <= alfa:
                    break  # Poda beta
        return mejor_valor
    else:
        mejor_valor = float('inf')
        for movimiento in movimientos:
            tablero_temp = tablero.copiar_tablero()
            if tablero_temp.realizar_movimiento(movimiento):
                valor = self.minimax(tablero_temp, profundidad - 1, alfa, beta, True, movimiento)
                mejor_valor = min(mejor_valor, valor)
                beta = min(beta, mejor_valor)
                if beta <= alfa:
                    break  # Poda alfa
        return mejor_valor
```

Se llama a la función **"*ordenar_movimientos*"** que se encarga de priorizar movimientos que capturan piezas enemigas y favorece movimientos que llevan piezas hacia el centro del tablero. Esto mejora la eficiencia del algoritmo al analizar primero los movimientos más prometedores

Se llama a la función ***"Obtener_mejor_movimiento"*** , su función es generar todos los movimientos posibles para la IA. Ordena los movimientos usando ***ordenar_movimientos***. Ejecuta minimax para cada movimiento y elige el que tenga el mayor valor.

La función ***"Obtener_todos_movimientos"*** que genera una lista de todos los movimientos válidos para un jugador de cierto color y también itera sobre cada pieza del tablero y consulta los movimientos legales.

El algoritmo Minimax es el corazón de la toma de decisiones en la IA, ya que analiza todos los movimientos posibles hasta una cierta profundidad, alternando entre maximizar las jugadas de la IA y minimizar las del oponente. Para hacerlo, necesita los movimientos legales del tablero, que se generan y simulan en copias temporales del juego sin alterar el

original. Cada estado resultante es evaluado usando la función de valoración del tablero, que considera tanto el valor de las piezas como su posición y factores estratégicos, como la seguridad del rey. Para mejorar la eficiencia, el Minimax utiliza poda Alfa-Beta, que descarta movimientos innecesarios, y puede priorizar jugadas prometedoras como capturas o avances al centro mediante ordenación de movimientos. Al final, el algoritmo selecciona el movimiento que ofrece el mejor resultado calculado, guiando a la IA hacia decisiones óptimas en cada turno.

método ordenar_movimientos(self, tablero, movimientos):

```python
def ordenar_movimientos(self, tablero, movimientos):
    """Ordena los movimientos para mejorar la eficiencia de la poda"""
    movimientos_valorados = []
    for mov in movimientos:
        valor = 0
        # Priorizar capturas
        pieza_destino = tablero.obtener_pieza(mov[0], mov[2][0], mov[2][1])
        if pieza_destino:
            valor += 10
        # Priorizar movimientos al centro
        centro_fila = abs(3.5 - mov[2][0])
        centro_col = abs(3.5 - mov[2][1])
        valor += (7 - (centro_fila + centro_col)) / 2
        movimientos_valorados.append((mov, valor))

    return [x[0] for x in sorted(movimientos_valorados, key=lambda x: x[1], reverse=True)]
```

La función ***ordenar_movimientos*** organiza una lista de movimientos posibles para priorizar aquellos que son más prometedores, lo cual mejora la eficiencia de algoritmos de búsqueda como poda alfa-beta al explorar primero los movimientos más valiosos.

Parámetros

- **tablero:**
  Representa el estado actual del tablero de ajedrez. Se utiliza para verificar las piezas en las posiciones de destino de los movimientos.
- **movimientos:**
  Una lista de movimientos posibles que la IA puede realizar.
  Cada movimiento está representado como una tupla con la siguiente estructura:
  (***tablero_num***, ***posición_origen***, ***posición_destino***), donde:

  tablero_num indica el tablero correspondiente.

  posición_origen es una tupla (fila, columna) con la posición inicial.

  posición_destino es una tupla (fila, columna) con la posición final.

Retorno

- list:
  Una lista de movimientos ordenados de mayor a menor prioridad, según su valor.

Descripción del funcionamiento

La función asigna un valor heurístico a cada movimiento para ordenarlos en función de su relevancia. Se consideran dos factores principales al calcular este valor:

1. Prioridad a movimientos de captura
   Si el movimiento lleva a capturar una pieza enemiga, se incrementa el valor del movimiento en +10.
2. Prioridad a movimientos hacia el centro del tablero
   Se da prioridad a los movimientos que acercan la pieza al centro del tablero (filas y columnas centrales). La distancia al centro se calcula usando:

   abs(3.5 - fila_destino) y abs(3.5 - columna_destino)
   La suma de estas distancias se ajusta y se utiliza para penalizar movimientos alejados del centro.

Proceso general

1. Iterar sobre cada movimiento en la lista movimientos.
2. Calcular un valor heurístico basado en:

   Si la posición de destino contiene una pieza enemiga (+10).

   La proximidad de la posición de destino al centro del tablero.

3. Almacenar cada movimiento junto con su valor en una lista temporal ***movimientos_valorados***.
4. Ordenar la lista de movimientos en orden descendente según los valores asignados.
5. Retornar únicamente la lista de movimientos ordenados, descartando los valores.

Ejemplo de comportamiento

Dado los movimientos:

- Movimiento A → destino contiene una pieza enemiga.
- Movimiento B → destino está más cerca del centro.

La función asignará un mayor valor a:

1. El movimiento A por capturar una pieza.
2. El movimiento B por avanzar hacia el centro del tablero.

Finalmente, la lista se ordenará para explorar primero los movimientos con mayor prioridad.

método obtener_mejor_movimiento(self, tablero):

```python
def obtener_mejor_movimiento(self, tablero):
    mejor_movimiento = None
    mejor_valor = float('-inf')
    alfa = float('-inf')
    beta = float('inf')

    movimientos = self.obtener_todos_movimientos(tablero, self.color)
    movimientos = self.ordenar_movimientos(tablero, movimientos)

    for movimiento in movimientos:
        tablero_temp = tablero.copiar_tablero()
        if tablero_temp.realizar_movimiento(movimiento):
            valor = self.minimax(tablero_temp, self.profundidad - 1, alfa, beta, False, movimiento)
            if valor > mejor_valor:
                mejor_valor = valor
                mejor_movimiento = movimiento
            alfa = max(alfa, mejor_valor)
    print(mejor_movimiento)
    return mejor_movimiento
```

La función ***obtener_mejor_movimiento*** calcula y devuelve el mejor movimiento posible para la Inteligencia Artificial (IA) en una posición dada del tablero de ajedrez. Utiliza el algoritmo **Minimax** con poda **alfa-beta** para evaluar y seleccionar el movimiento más favorable.

Parámetros

- **tablero**:
  Objeto que representa el estado actual del tablero de ajedrez.
  Debe proporcionar métodos para copiar el tablero y realizar movimientos.

Atributos utilizados

- **self.color**: El color de las piezas de la IA (blanco o negro).
- **self.profundidad**: Profundidad máxima para la búsqueda en el algoritmo Minimax.

Retorno

- ***mejor_movimiento***:
  El movimiento seleccionado como el más favorable desde la perspectiva de la IA.
  Se representa como una tupla con la estructura:
  (*tablero_num*, *posición_origen*, *posición_destino*).

Descripción del funcionamiento

Inicialización de variables:

- ***mejor_movimiento*** guarda el mejor movimiento encontrado durante la búsqueda.
- ***mejor_valor*** almacena el valor máximo de evaluación obtenido (inicialmente −∞-\infty−∞).
- alfa y beta inicializan los límites para la poda alfa-beta.

Generación y ordenamiento de movimientos:

- Se generan todos los movimientos posibles con el método **obtener_todos_movimientos**.
- Los movimientos se ordenan usando la función **ordenar_movimientos** para priorizar aquellos más prometedores (ej., capturas o posiciones centrales).

Evaluación de movimientos:

- Iterar sobre cada movimiento ordenado.
- Crear una copia temporal del tablero con **tablero.copiar_tablero()** para no modificar el estado actual.
- Realizar el movimiento en el tablero temporal con **realizar_movimiento.**

Llamada al algoritmo Minimax:

- Llamar a la función minimax para evaluar el movimiento desde la perspectiva de la profundidad restante.
- El parámetro False indica que en la siguiente capa, es el turno del oponente (minimización).

Selección del mejor movimiento:

- Si el valor obtenido del Minimax es mayor que el **mejor_valor** registrado:Actualizar **mejor_valor** y **mejor_movimiento.**
- Actualizar el valor de **alfa** para la poda alfa-beta.

Retorno del resultado:

- Imprime el mejor movimiento encontrado.
- Retorna el movimiento como resultado.

método obtener_todos_movimientos(self, tablero, color):

```python
def obtener_todos_movimientos(self, tablero, color):
    """
    Obtiene todos los movimientos posibles para un color dado
    """
    movimientos = []
    for tablero_num in [1, 2]:
        for fila in range(8):
            for columna in range(8):
                pieza = tablero.obtener_pieza(tablero_num, fila, columna)
                if pieza and pieza[1] == color:
                    movs = tablero.movimientos_pieza(pieza[0], (fila, columna), tablero_num)
                    for mov in movs:
                        movimientos.append((tablero_num, (fila, columna), mov))
    return movimientos
```

La función **obtener_todos_movimientos** genera y devuelve una lista con todos los movimientos legales posibles para un color específico en un tablero de ajedrez. La función itera a través de todas las posiciones del tablero y busca las piezas que pertenecen al jugador del color especificado.

Parámetros

- **tablero**:
  Objeto que representa el estado actual del tablero de ajedrez.
  Proporciona métodos para obtener piezas y sus movimientos válidos.
- **color**:
  El color de las piezas cuyos movimientos se desean calcular.
  Puede ser **Color.BLANCO** o **Color.NEGRO**.

Retorno

- **movimientos**:
  Lista de tuplas con la estructura:
  (tablero_num, (fila_origen, columna_origen), (fila_destino, columna_destino))
  Cada tupla representa un movimiento legal encontrado en el tablero.

Descripción del funcionamiento

1. **Inicialización de la lista de movimientos**:
   Se crea una lista vacía **movimientos** que almacenará todos los movimientos legales encontrados.

2. **Iteración por los tableros**:
   El bucle **for tablero_num in [1, 2]** itera sobre ambos tableros (si se juega con múltiples tableros, por ejemplo, en variantes del ajedrez).
3. **Recorrer las posiciones del tablero**:
   - Los bucles anidados recorren cada celda del tablero de ajedrez, iterando sobre todas las filas (0-7) y columnas (0-7).
4. **Verificación de piezas**:
   - Se obtiene la pieza en la posición actual mediante **tablero.obtener_pieza(tablero_num, fila, columna)**.
   - Si existe una pieza y su color coincide con el parámetro **color**, se procede a calcular sus movimientos.
5. **Cálculo de movimientos de la pieza**:
   - Se obtienen todos los movimientos válidos para la pieza usando **tablero.movimientos_pieza(pieza[0], (fila, columna), tablero_num)**.
   - Cada movimiento es una coordenada destino **(fila_destino, columna_destino)**.
6. **Almacenamiento de movimientos**:
   - Por cada movimiento válido, se agrega una tupla con la información del movimiento a la lista **movimientos**.
7. **Retorno de la lista**:
   La lista **movimientos** se devuelve al final con todos los movimientos legales encontrados.

Estructura de salida

La lista de movimientos tendrá elementos en la siguiente forma:

(tablero_num, (fila_origen, columna_origen), (fila_destino, columna_destino))

- **tablero_num**: Número del tablero donde ocurre el movimiento (1 o 2).
- **fila_origen, columna_origen**: Posición inicial de la pieza.
- **fila_destino, columna_destino**: Posición destino a la que se puede mover la pieza.

Estructuras de datos utilizadas

Diccionarios (dict):

Se utilizan para almacenar las tablas de posición de cada tipo de pieza (***self.tablas_posicion***). Cada clave corresponde a una pieza específica (como **PEON**, **CABALLO**, etc.), y el valor asociado es una matriz que evalúa posiciones estratégicas en el tablero.

También se emplean para asociar valores numéricos a las piezas (***valores_piezas***), representando su valor relativo en el juego.

Listas (list):

Las listas bidimensionales se utilizan para las tablas de posición, donde cada entrada representa el valor estratégico de una celda del tablero.
Se usan para almacenar movimientos posibles de las piezas (movimientos en ***obtener_todos_movimientos***) y ordenar esos movimientos según su valor estratégico.

Tuplas (tuple):

Los movimientos individuales se representan como tuplas ((***tablero_num***, (fila, columna), destino)), que contienen el tablero de origen, la posición inicial de la pieza y su posición de destino.
Esto facilita la manipulación y evaluación de movimientos, ya que las tuplas son inmutables y rápidas.

Matrices (listas anidadas):

Las tablas de posición son matrices (listas anidadas de 8x8) que asignan valores estratégicos a cada celda en el tablero dependiendo del tipo de pieza. Estas matrices permiten calcular ventajas posicionales fácilmente.