Taller Problema 8 Reinas

Profesor: Angel Augusto Agudelo Zapata

Andres Felipe Gordillo Guerrero

1088348241

a.gordillo@utp.edu.co

Inteligencia Artificial

Universidad tecnológica de pereira

2024 - Semestre II

El código resuelve el problema de las 8 reinas utilizando el enfoque de Satisfacción de Restricciones (CSP) con el algoritmo Min-Conflicts y el uso de la biblioteca SimpleAI. Además, proporciona funciones para mostrar visualmente el tablero de ajedrez tanto en la consola como gráficamente con Matplotlib.

Detalles del Código
Importación de Bibliotecas:

```
from simpleai.search import CspProblem
from simpleai.search.csp import min_conflicts
import matplotlib.pyplot as plt
import numpy as np
```

1.
    - `CspProblem`: Clase para definir problemas de Satisfacción de Restricciones (CSP).
    - `min_conflicts`: Algoritmo de búsqueda local utilizado para resolver el CSP.
    - `matplotlib.pyplot`: Utilizado para crear gráficos.
    - `numpy`: Biblioteca para trabajar con arrays y matrices.

Definición de Variables y Dominios:

```
variables = list(range(8))  # Variables representando cada columna
del tablero (0 a 7)
dominios = {var: list(range(8)) for var in variables}  # Dominios
son las filas (0 a 7)
```

2.
    - `variables`: Representa cada columna del tablero de ajedrez.
    - `dominios`: Dict donde cada variable tiene como dominio las filas posibles para colocar una reina.

Definición de Restricciones:

```
def restricciones(variables, valores):
    columna1, columna2 = variables
    fila1, fila2 = valores

    # Restricciones para asegurar que las reinas no se ataquen
    if fila1 == fila2 or abs(columna1 - columna2) == abs(fila1 -
fila2):
        return False
    return True
```

- `restricciones`: Función que verifica que dos reinas no se ataquen en la misma fila o diagonal.

Creación del Problema CSP:

```
problema = CspProblem(variables, dominios, [(par, restricciones) for
par in pares_de_variables])
```

- `CspProblem`: Crea un objeto que representa el problema CSP con las variables, dominios y restricciones especificadas.

Resolución del Problema:

```
resultado = min_conflicts(problema)
```

- `min_conflicts`: Algoritmo que busca la solución al CSP utilizando la estrategia de mínimos conflictos.

Mostrar el Resultado en Consola:

```
for columna, fila in resultado.items():
    print(f"Columna {columna}: Fila {fila}")
```

- Muestra las posiciones de las reinas en el tablero en formato de texto en la consola.

Función para Mostrar el Tablero en la Consola:

```
def mostrar_tablero(resultado):
    # Crear matriz para representar el tablero
    tablero = [['.' for _ in range(8)] for _ in range(8)]
    # Colocar reinas en la matriz
    for columna, fila in resultado.items():
        tablero[fila][columna] = 'Q'
    # Imprimir tablero
    for fila in tablero:
        print(' '.join(fila))
```

- `mostrar_tablero`: Genera una representación visual del tablero de ajedrez con reinas en la consola.

Función para Mostrar el Tablero Gráficamente:

```
def mostrar_tablero_grafico(resultado):
    # Crear figura y ejes
    fig, ax = plt.subplots()
    tablero = np.zeros((8, 8))  # Crear matriz 8x8 para el tablero
    # Colorear casillas alternas del tablero
    for i in range(8):
        for j in range(8):
            if (i + j) % 2 == 0:
                tablero[i, j] = 1
    ax.imshow(tablero, cmap='gray')  # Mostrar tablero con colores
alternos
    # Colocar reinas en el tablero
    for columna, fila in resultado.items():
        ax.text(columna, fila, '♛', fontsize=35, ha='center',
va='center', color='red')
    # Configurar ejes
    ax.set_xticks(np.arange(8))
    ax.set_yticks(np.arange(8))
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.grid(False)
    plt.show()  # Mostrar figura
```

- `mostrar_tablero_grafico`: Visualiza el tablero de ajedrez con reinas usando gráficos con Matplotlib.

Uso del Código

- El código resuelve el problema de las 8 reinas utilizando el CSP y el algoritmo Min-Conflicts.
- Muestra las soluciones encontradas tanto en la consola como en una representación gráfica del tablero de ajedrez.

1. Uso de SimpleAI:

SimpleAI proporciona las clases y métodos necesarios para definir y resolver problemas de satisfacción de restricciones (CSP). En particular:

- `CspProblem`: Clase que define el problema de satisfacción de restricciones. Se utiliza para establecer las variables, dominios y restricciones del problema.
- `min_conflicts`: Es el algoritmo que SimpleAI utiliza para resolver el problema de las 8 reinas. Este es un algoritmo local, similar a **hill climbing**, que ajusta las posiciones de las reinas para minimizar el número de conflictos entre ellas.

Parte relevante del código donde se utiliza SimpleAI:

```
from simpleai.search import CspProblem

from simpleai.search.csp import min_conflicts

# Creamos el problema con las variables, dominios y restricciones

problema = CspProblem(variables, dominios, [(par, restricciones) for
par in pares_de_variables])

# Resolvemos el problema utilizando el algoritmo min_conflicts

resultado = min_conflicts(problema)
```

2. Algoritmo Min-Conflicts:

El algoritmo **min_conflicts** es una variante de los algoritmos de búsqueda local como **hill climbing**. La diferencia principal es que en vez de hacer mejoras graduales hacia la mejor solución posible (hill climbing), este algoritmo selecciona una variable que esté en conflicto y la reubica en el valor que minimiza los conflictos con las demás variables.

- **Hill Climbing** y **Min-Conflicts** son similares en el sentido de que ambos son algoritmos de optimización local. En el caso del problema de las 8 reinas, ambos buscarían minimizar el número de conflictos entre reinas.

En el código:

```
resultado = min_conflicts(problema)
```

El método `min_conflicts` actúa sobre el problema `CspProblem` y realiza un proceso de búsqueda local, similar a **hill climbing**, hasta encontrar una solución sin conflictos (o con un número mínimo de conflictos).