UDACITY

PROJECT

Creating an AI Agent to solve Sudoku

A part of the Artificial Intelligence Program

PROJECT REVIEW

CODE REVIEW 5

NOTES

solution.py 5

```python
assignments = []
```

SUGGESTION

logging with default level ERROR could be added to debug the code. Logs can also help to understand the algorithms. Please have a look at this link : https://docs.python.org/3/howto/logging.html. Assert statements could be used too https://wiki.python.org/moin/UsingAssertionsEffectively

```python

rows = 'ABCDEFGHI'
cols = '123456789'


def cross(A, B):
    "Cross product of elements in A and elements in B."
    return [s+t for s in A for t in B]

# Define boxes, rows and cols units that will compose the grid
boxes = cross(rows, cols)
row_units = [cross(r, cols) for r in rows]
column_units = [cross(rows, c) for c in cols]
square_units = [cross(rs, cs) for rs in ('ABC','DEF','GHI') for cs in ('123','456','789')]
# Define the diagonal units
diagonal_units = [[rows[i] + cols[i] for i in range(len(rows))]] + [[rows[i]+cols[::-1][i] for i in range(len(rows))]]
```

AWESOME

Good job (y) Additional constraints for diagonal sudoku implemented successfully :)

```python

# Add the diagonal units to the unit list
unitlist = row_units + column_units + square_units + diagonal_units
units = dict((s, [u for u in unitlist if s in u]) for s in boxes)
peers = dict((s, set(sum(units[s],[])) - set([s])) for s in boxes)


def assign_value(values, box, value):
    """
    Please use this function to update your values dictionary!
    Assigns a value to a given box. If it updates the board record it.
    """

    # Don't waste memory appending actions that don't actually change any values
    if values[box] == value:
        return values

    values[box] = value
    if len(value) == 1:
        assignments.append(values.copy())
    return values

def naked_twins(values):
    """Eliminate values using the naked twins strategy.
    Args:
        values(dict): a dictionary of the form {'box_name': '123456789', ...}

    Returns:
        the values dictionary with the naked twins eliminated from peers.
```

SUGGESTION

Its a good practice to modularize the code, like according to the logic naked_twins can be split up in two methods find_twins, eliminate twins to enhance readability.

```
    """

    # Find all instances of naked twins
    # Eliminate the naked twins as possibilities for their peers

    # Find boxes with 2 candidates
    _twins = [box for box in values.keys() if len(values[box]) == 2]

    # Find the naked twins and create a list of lists
    naked_twins = [[box1, box2] for box1 in _twins for box2 in peers[box1] if set(values[box1]) == set(values[box2])]

    for i in range(len(naked_twins)):
        box1 = naked_twins[i][0]
        box2 = naked_twins[i][1]

        # Find peers for first and second twins
        peers1 = set(peers[box1])
        peers2 = set(peers[box2])

        # Join the two sets
```

AWESOME

Great work providing conceptual comments in between the method where important logic is coded. its a good practice and helps demonstrating your thought process.

```
        common_peers = peers1 & peers2

        # Remove the naked twins as candidates from the common peers
        for peer in common_peers:
            if len(values[peer]) >= 2:
                peers_over_1 = values[box1]

                for k in peers_over_1:
                    values = assign_value(values, peer, values[peer].replace(k, ''))
    return values


def grid_values(grid):
    """
    Convert grid into a dict of {square: char} with '123456789' for empties.
    Args:
        grid(string) - A grid in string form.
    Returns:
        A grid in dictionary form
            Keys: The boxes, e.g., 'A1'
            Values: The value in each box, e.g., '8'. If the box has no value, then the value will be '123456789'.
    """

    chars = []
    digits = '123456789'
    for c in grid:
        if c in digits:
            chars.append(c)
        if c == '.':
            chars.append(digits)
    assert len(chars) == 81
    return dict(zip(boxes, chars))




def display(values):
    """
    Display the values as a 2-D grid.
    Args:
        values(dict): The sudoku in dictionary form
    """

    width = 1+max(len(values[s]) for s in boxes)
    line = '+'.join(['-'*(width*3)]*3)
    for r in rows:
        print(''.join(values[r+c].center(width)+('|' if c in '36' else '')
                      for c in cols))
        if r in 'CF': print(line)
    return


def eliminate(values):
```

SUGGESTION

Its a good practice to provide your method with a docstring that helps in understanding the functioning of the method. Please have a look at this link: https://www.python.org/dev/peps/pep-0257/

```python
    solved_values = [box for box in values.keys() if len(values[box]) == 1]
    for box in solved_values:
        digit = values[box]
        for peer in peers[box]:
            # values[peer] = values[peer].replace(digit,'')

            # def assign_value(values, box, value):
            values = assign_value(values, peer, values[peer].replace(digit, ''))
    return values

def only_choice(values):
    for unit in unitlist:
        for digit in cols:
            dplaces = [box for box in unit if digit in values[box]]
            if len(dplaces) == 1:
                # values[dplaces[0]] = digit
                values = assign_value(values, dplaces[0], digit)
    return values

def reduce_puzzle(values):
    stalled = False
    while not stalled:
        # Check how many boxes have a determined value
        solved_values_before = len([box for box in values.keys() if len(values[box]) == 1])

        # Use the Eliminate Strategy
        values = eliminate(values)

        # Use the Only Choice Strategy
        values = only_choice(values)

        # Use the Naked Twins Strategy
        values = naked_twins(values)

        # Check how many boxes have a determined value, to compare
        solved_values_after = len([box for box in values.keys() if len(values[box]) == 1])

        # If no new values were added, stop the loop.
        stalled = solved_values_before == solved_values_after

        # Sanity check, return False if there is a box with zero available values:
        if len([box for box in values.keys() if len(values[box]) == 0]):
            return False
    return values

def search(values):

    # First, reduce the puzzle using the previous function
    values = reduce_puzzle(values)

    if values is False:
        return False ## Failed earlier

    if all(len(values[s]) == 1 for s in boxes):
        return values ## Solved!

    # Choose one of the unfilled squares with the fewest possibilities
    n,s = min((len(values[s]), s) for s in boxes if len(values[s]) > 1)

    # Now use recurrence to solve each one of the resulting sudokus, and

    for value in values[s]:
        new_sudoku = values.copy()
        new_sudoku[s] = value
        attempt = search(new_sudoku)
        if attempt:
            return attempt

def solve(grid):
    """
    Find the solution to a Sudoku grid.
    Args:
        grid(string): a string representing a sudoku grid.
            Example: '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
    Returns:
        The dictionary representation of the final sudoku grid. False if no solution exists.
    """
    return search(grid_values(grid))

if __name__ == '__main__':
    diag_sudoku_grid = '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
    display(solve(diag_sudoku_grid))

    try:
        from visualize import visualize_assignments
        visualize_assignments(assignments)

    except SystemExit:
        pass
    except:
        print('We could not visualize your board due to a pygame issue. Not a problem! It is not a requirement.')

```

▸ README.md

RETURN TO PATH

Student FAQ Reviewer Agreement