Assignment 7

Name: Krunal Rank
Roll No: U18CO081

Consider the problem of solving crossword puzzles: fitting words into a rectangular grid. The grid, which is given as part of the problem, specifies which squares are blank and which are shaded. For each word starting square you have a list of words that can be fitted (vertical and/or across).

Crossword Solver Class:

```
import copy
import numpy as np
import cv2


class CrosswordSolver:
   def __init__(self, rows: int, cols: int, verbose: int):
       self.rows = rows
       self.cols = cols
       self.matrix = [["#" for i in range(self.cols)] for j in range(self.rows)]
       self.ids = dict()
       self.list = list()
       self.verbose = verbose

       if self.verbose == 1:
           print("Initialised CrosswordSolver with parameters:")
           print("Rows:", self.rows)
           print("Cols:", self.cols)
           print("Verbose:", self.verbose)

       self.font = cv2.FONT_HERSHEY_SIMPLEX
       self.startImageWrite = (10, 50)
       self.fontScale = 1
       self.fontColor = (255, 255, 255)
       self.lineType = 2

   def block(self, row: int, col: int):
       self.matrix[row][col] = "-"
       if self.verbose == 1:
```

```
        print("Blocked", (row, col), "in the Base State!")

    def add_id(self, row: int, col: int, id: int):
        self.ids[id] = (row, col)
        if self.verbose == 1:
            print("Added ID at", (row, col), "with ID:", id)

    def add_word(self, id: int, direction: int, word: str):
        self.list.append((word, id, direction))
        if self.verbose == 1:
            print(
                "Added Word",
                word,
                "at ID",
                id,
                "in direction",
                ("Across" if direction == 0 else "Down"),
                "!",
            )

    def check_valid_insert(
        self, state: list(list()), id: int, direction: int, word: str
    ) -> bool:
        row, col = self.ids[id]

        if direction == 0:
            for i in range(col, col + len(word)):
                if i == self.cols:
                    return False
                if state[row][i] != "#" and state[row][i] != word[i - col]:
                    return False
        else:
            for i in range(row, row + len(word)):
                if i == self.rows:
                    return False
                if state[i][col] != "#" and state[i][col] != word[i - row]:
                    return False
        return True

    def insert_word(
        self, state: list(list()), id: int, direction: int, word: str
    ) -> list(list()):
```

```
    row, col = self.ids[id]

    new_state = copy.deepcopy(state)

    if direction == 0:
        for i in range(col, col + len(word)):
            new_state[row][i] = word[i - col]
    else:
        for i in range(row, row + len(word)):
            new_state[i][col] = word[i - row]
    return new_state

def is_solved(self, state: list(list())) -> bool:

    for i in range(self.rows):
        for j in range(self.cols):
            if state[i][j] == "#":
                return False
    return True

def __visualise(self, state):
    if self.verbose == 1:
        strings = []

        for i in state:
            new_string = ""
            for j in i:
                new_string += j + " "
            strings.append(new_string)

        img = np.zeros((50 * self.rows, 40 * self.cols, 3), np.uint8)
        offset = 0
        for i in strings:
            cv2.putText(
                img,
                i,
                (self.startImageWrite[0], self.startImageWrite[1] + offset),
                self.font,
                self.fontScale,
                self.fontColor,
                self.lineType,
```

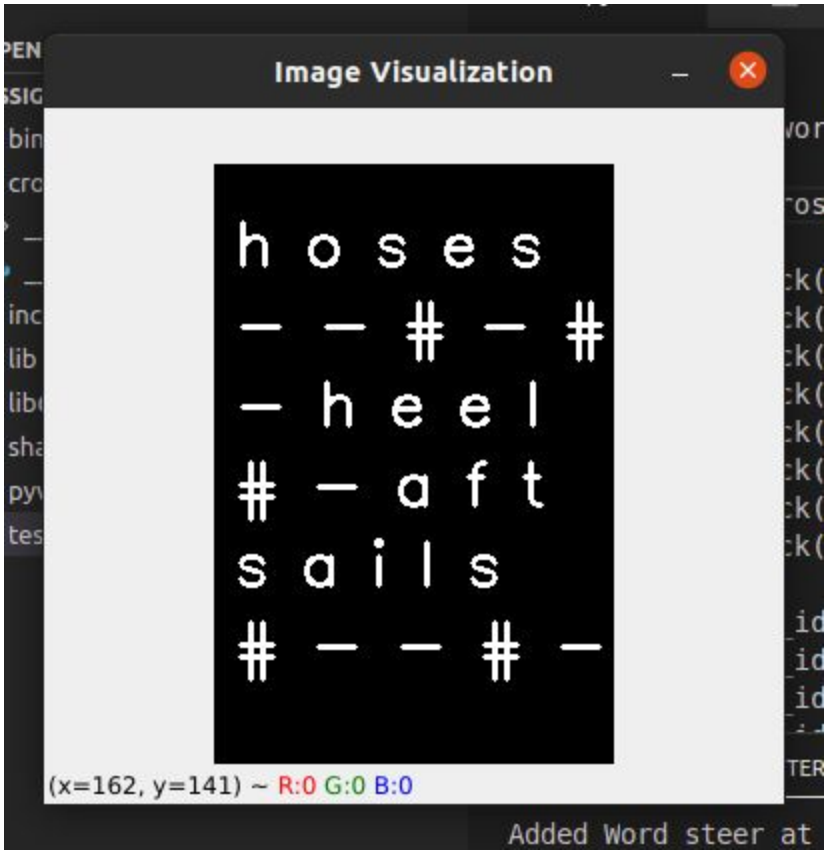
```
            )
            offset = offset + 40
        cv2.imshow("Image Visualization", img)
        cv2.waitKey(33)

def solve(self):
    state = copy.deepcopy(self.matrix)
    solved_state, result = self.__solve_crossword(state, 0)
    if result == 0:
        return [[]]

    return solved_state

def __solve_crossword(self, state: list(list()), word_idx: int):
    self.__visualise(state)
    if word_idx == len(self.list):
        if self.is_solved(state):
            return (state, 1)
        else:
            return (state, 0)

    word, id, direction = self.list[word_idx]

    if self.check_valid_insert(state, id, direction, word) == True:
        new_state = self.insert_word(state, id, direction, word)

        (solved_state, result) = self.__solve_crossword(new_state, word_idx + 1)

        if result == 1:
            return (solved_state, result)

    return self.__solve_crossword(state, word_idx + 1)
```

Tester Code:

```
from crossword_solver import CrosswordSolver

solver = CrosswordSolver(6,5,0)

solver.block(1,0)
solver.block(1,1)
solver.block(1,3)
solver.block(2,0)
solver.block(3,1)
solver.block(5,1)
solver.block(5,2)
solver.block(5,4)

solver.add_id(0,0,1)
solver.add_id(0,2,2)
solver.add_id(0,4,3)
solver.add_id(2,1,4)
solver.add_id(2,3,5)
solver.add_id(3,0,6)
solver.add_id(3,2,7)
solver.add_id(4,0,8)

solver.add_word(1,0,"hoses")
solver.add_word(1,0,"laser")
solver.add_word(1,0,"sheet")
solver.add_word(1,0,"steer")
solver.add_word(1,0,"sails")

solver.add_word(4,0,"heel")
solver.add_word(4,0,"hike")
solver.add_word(4,0,"keel")
solver.add_word(4,0,"knot")
solver.add_word(4,0,"line")

solver.add_word(7,0,"aft")
solver.add_word(7,0,"ale")
solver.add_word(7,0,"eel")
solver.add_word(7,0,"lee")
solver.add_word(7,0,"tie")
```

```
solver.add_word(8,0,"hoses")
solver.add_word(8,0,"laser")
solver.add_word(8,0,"sheet")
solver.add_word(8,0,"steer")
solver.add_word(8,0,"sails")

solver.add_word(2,1,"hoses")
solver.add_word(2,1,"laser")
solver.add_word(2,1,"sheet")
solver.add_word(2,1,"steer")
solver.add_word(2,1,"sails")

solver.add_word(3,1,"hoses")
solver.add_word(3,1,"laser")
solver.add_word(3,1,"sheet")
solver.add_word(3,1,"steer")
solver.add_word(3,1,"sails")

solver.add_word(5,1,"heel")
solver.add_word(5,1,"hike")
solver.add_word(5,1,"keel")
solver.add_word(5,1,"knot")
solver.add_word(5,1,"line")

solver.add_word(6,1,"aft")
solver.add_word(6,1,"ale")
solver.add_word(6,1,"eel")
solver.add_word(6,1,"lee")
solver.add_word(6,1,"tie")

ans = solver.solve()


print("-"*35)
for i in ans:
   for j in i:
       print(j,end=" ");
   print("")
print("-"*35)
```

Visualiser:



Solution: