Q1) A working demo of your system

Q2) The algorithm and any heuristics that you apply. Explain your algorithm and heuristics.

There are 8 so-called “Word Placer”s in the given puzzle, and the algorithm runs through them in a while loop.

In each iteration, the algorithm checks all words of the wordPlacer’s length and places the first one it deems suitable. A word is suitable iff there are no contradictions with the other placed words (i.e, for a given vertical/horizontal wordPlacer pair, the square that they clash on is of the same letter).

If no unplaced or unchecked word left for a given wordPlacer to left, this indicates that there has been misplacements previously. Hence, backtracking is performed to clear the previous words and try other possibilities in their stead before advancing again. Ideally, this method will check all possible combinations to find the possible solutions, if any exists.

The loop is over once all wordPlacers have been filled with appropriate words.

In each step, it is apparent that any word placed on a WordPlacer might not lead to the correct solution, but since the program has a memory of tried words for each wordPlacer, it can use this data as heuristics to make more educated guesses for the next step.

Q3) Provide a way to show the assignments step by step. In other words, your implementation must show the steps of the backtracking algorithm

In each iteration, the board is printed with a 1 second sleep() function in between, so while running the code – the assignments are shown step by step in the command line. I also print any details about which word is being tried for which wordPlacer and details about the backtracking whenever it is performed. Furthermore, I provided comments within the code to explain my reasoning in each logical step of the algorithm.

Q4) The full CSP problem formulation. A clear description must be given in a formal notation.

The set of variables can be formulated in various ways of thinking: each square within the board or more generally, each wordPlacer.

As such, X = {wp1, wp2…, wp8}

For each wordPlacer, the domain of values it can take is:

D = {"aft","ale","eel","heel","hike","hoses","keel","knot","laser","lee","line","line","sails”, "sheet","steer","tie"}

C = {wordPlacer.length must be equal to len(word) that fills it, for a given vertical/horizontal wordPlacer pair, the square that they clash on must be of the same letter}

Q5) The implementation details of your code. Code must be readable and understandable.

I guess the code needs to be examined for this question and I tried to make it as understandable as possible by using logical variable names and adding comments wherever necessary.

Q6) A brief description (approximately 1 page in length) of your experience programming for and participating in the tournament

Any task I do using Python is a fun one, and bonus points if it is in fact an assignment (2 birds with 1 stone!). I tried to modularize the code as much as I can, by creating different classes and implementing functionalities within each of them. I used main.py as the driver code for the given crossword puzzle, however most of the implementation is handled under the member functions of the board instance, which is a singleton, technically. It is possible that I did not use the resources in the most efficient way, however the code gets the job done.

There were cases of undeleted random letters whenever clearing a wordPlacer and it took some time to debug it. However the reason behind it was simple: while creating the tempLayout to check whether a word fits, I was assigning it to the board’s layout directly. So, even in the cases that the word did not fit, some residue letters were left in the board. Python uses assignment by reference, so the solution was to use deepCopy.

Overall, I can say 462 has been quite enjoyable in both assignments, and also provided good practice.