Assignment 3: CSP Battleship Heuristics

To generate the battleship solitaire, my code first preprocesses the input grid. Preprocessing is done as listed below from point 1 to 9.

1. Block rows/cols that satisfies constraints already, including blocking 0 constraint row/col with waters
2. Add ship to edge ships (T, B, L, R) accordingly
3. Block rows/cols that satisfy constraints because of this added ship and add surrounding Ws for each assigned ship block accordingly, including assigning 'W' to the diagonals of M grids
4. Check middle piece and if its:
    i. Top OR bottom is blocked (water), assign ship on its right AND left
    ii. Right OR left is blocked (water), assign ship on its top AND bottom
5. Block rows/cols that satisfy constraints because of this added ship and add surrounding Ws for each assigned ship block accordingly
6. Number of unassigned grid in a row/col = row/col constraint, assign ships
7. Block rows/cols that satisfy constraints because of this added ship and add surrounding Ws for each assigned ship block accordingly
8. Number of unassigned grid in a row/col = row/col constraint, assign ships
9. Block rows/cols that satisfy constraints because of this added ship and add surrounding Ws for each assigned ship block accordingly
10. Assign variable and constraints and start search

The variables and constraints are then defined. The code has ships as its variables. The variables are named by a tuple that includes the ship type (or size e.g. ship size for a destroyer is 2) and an identifier for its ship type. For example, if there are two destroyers, the variables would be: (2,0) and (2,1). The variable domain is described in a list of tuple with the format of (starting row index, starting column index, ship length, orientation of the ship (written as h or v)). A domain example: (0,0,2,'h'). The domain is reduced to the available space left after the initial preprocessing. The constraints are defined by each number of ships are available on each row or column constraint.

Finally, the search will start. Backtracking search is implemented. The search starts by assigning values to the largest ship and ends with the smallest ship. When a value is assigned to a variable, row/col blocking and surrounding grids are updated (this is the same as preprocessing step 3,5,7,9) before checking if each constraint is violated because of this assignment. State caching is also implemented where it keeps track of previous no-solution states and if the search ends up with a visited state, it stops itself from exploring the no-solution state further again.