CSCI 6511 – Project 2 Tile Placement
Maxime Kotur

This CSP Tile Placement problem gives a grid where we have different bushes in each cell which are represented by numbers. We have 3 different types of 4x4 tiles that we can place on the grid in order to get the exact amount bushes at the end while covering them with different types of tile shapes. We have 3 types of tile shapes which are the Full block, Outer block and the El Shape block. We must use all of them, and we are given the amount of each tile shape that we have at our disposal.

To solve the given problem using CSP, we can define the following variables, domains, and constraints:
Variables: We define one variable for each 4x4 tile on the landscape grid, and assign a unique identifier to each variable. For example, for a 100x100 landscape, we will have 625 (25x25) variables. Therefore, the amount of variables will be the total amount of tile shapes given and each represent a spot that we can place a tile in.
Domains: Each variable can take one of the three shapes: full block, outer boundary, or EL shape. So, the domain of each variable is {Full, Outer, EL}.
Constraints: The constraints are defined based on the target configuration of bushes that need to be visible. For each position on the landscape, we check which tiles cover that position, and ensure that at least one of them has the shape that allows the corresponding bush to be visible. For example, if the target requires that the bushes of type 2 and 3 should be visible, and a particular position on the landscape is covered by three tiles, then at least one of those three tiles should have the outer boundary or full block shape for bushes of types 2 or 3.

To implement the CSP algorithm, we can use the following components:
Search Algorithm: For this I use a backtracking search algorithm to find the solution with a DFS approach. The search algorithm iteratively assigns a value to a variable and checks if the constraints are satisfied. If the constraints are not satisfied, it backtracks to the previous assignment and tries a different value.
Heuristics:
a. **Minimum Remaining Values (MRV):** At each step of the search, we select the variable with the smallest domain, i.e., the variable that has the fewest options left. This means that we choose the least abundant tile type. This is to ensure that we explore the most constrained variable first and reduce the search space. In order to get the MRV tile type, we calculate the total combinations of bushes while iterating and then we forward check to filter out the combinations that violate the constraints given to us. We also start with the shape which has the lowest amount of combinations first as that type would be less constrained than the other types. Furthermore, the Full block isn't applied here as a Full block will just cover all the bushes on that tile making it easier to place. This helps us reduce the domain.

b. **Least Constraining Value (LCV):** When we select a variable, we select the value that rules out the fewest options for the other variables. This is to ensure that we make the most progress towards the solution with each assignment. In order to do this, we use the MRV calculated before and we get the new targets of each combination minus the actual target. We would then get the minimum change of these new targets and use that as our least constrained value. This then gets put into our AC3 algorithm for constraint satisfaction.

Constraint Propagation using AC3: We use the AC3 (Arc Consistency 3) algorithm to ensure that the constraints are satisfied. AC3 is an algorithm that reduces the domains of variables by removing values that are inconsistent with the constraints. We use AC3 to preprocess the constraints and reduce the search space.

Here is the pseudocode for the CSP problem that I used:

Initialize domains for each variable
Preprocess the constraints using AC3
Call recursive_backtracking_search function with the current assignment, domains, and constraints
recursive_backtracking_search(assignment, domains, constraints):

a. If assignment is complete, return assignment.
b. Select a variable using MRV heuristic.
c. For each value in the domain of the selected variable, do the following:
    i. If the value is consistent with the constraints, add the variable to the assignment and update the domains based on the constraints.
    ii. Call recursive_backtracking_search with the updated assignment, domains, and constraints.
    iii. If the call to recursive_backtracking_search returns a valid assignment, return the assignment
    iv. Remove the variable and its value from the assignment and restore the domains to their previous state.
d. If no value in the domain of the selected variable satisfies the constraints, backtrack to the previous variable.

I also have added unitTests by using python's unittest module. I have added all the given testcases with solutions and compared them with my output. If the two arrays are the same, it will pass. Most testcases take less than 30 seconds to pass however some take about 3 minutes. I use the time module to calculate this. I think the amount of time is correlated to having more EL shapes as that shape gives the most combinations on average.