- Tiling and Packing Algorithm
    - We implemented the “Tiling and Packing Algorithm” using the DFS method of backtracking.
    - Hexomino figures are expressed in coordinates (Flip, rotate). There are 35 figures and 1 specific duplicate figure. So, total of 36 figures and 216 cells. And, there is 3x3 empty square in the middle. Therefore, 225 cells (15x15) can be expressed.
    - In the board, each cell is expressed by true and false.
    - We consider the first cell of the board, and try all possible ways to cover it with a Hexomino. Then consider the next uncovered cell, and try all possible ways to cover it with another Hexomino. And so on, until all cells of the board are covered.

1. First, choose one of the hexomino models and fill it out from the left corner of the board.
2. Find the figure that can be filled. If the correct figure is found, the figure is placed on the right side in turn. Then save its state in the stack.
3. It proceeds until the figures are filled in by 15x15 board.

Therefore, time complexity can be converted to O(V+E).

- Three Coloring Algorithm
    - We implemented the “Three Coloring Algorithm” using the backtracking algorithm.
    - A 2D array board[V][V] (V means vertex) where V is the number of vertices in a graph. And board[V][V] is the adjacency matrix representation of the graph. If a value of board[i][j] equals 1, it means that there is a direct edge from i to j. Otherwise, if a value of board[i][j] equals 0, there is no edge from i to j.
    - Approach : Assign colors one by one to different vertices, starting from vertex 0. Before assigning a color, check whether the color is assigned already. If there is a color assignment that follows the condition, mark the color assignment as a solution. If no color assignment is possible, then backtrack.

1. First, we should make an undirected graph. It is possible to know where the figures are placed through a tiling and packing algorithm, so we need to create a 2-dimensional array that represents the location of each tile.

2. Then, the 2-dimensional array is converted into an adjacency matrix. Now, we can see the tiles adjacent to each tile.
3. Assign colors one by one to different vertices, starting from the vertex 0.

Therefore, time complexity can be converted to O(3^V).