Code comments

The main.py file contains two equivalent functions that solve the challenge. One follows a recursive approach and the other one an iterative one.

- Recursive approach:

Starting from the snake's initial position, we calculate its head adjacent cells and check if they are valid positions (positions the snake can move to). For each valid position, we move the snake to that position and accumulate the result of calling the function recursively with this new location for the snake and decreasing the current paths' depth value.

Function's call can be a base case or an intermediate case. Base case consists of states where the remaining steps that the snake can follow (depth that we are recursively decreasing) is 0 and intermediate case when it is bigger than 0.

Base case returns 1 as it has reached the last position of a valid path from the initial position of the snake. Intermediate case calls stop when all its subcalls stop and returns the total number of distinct valid paths that have been reached in these subcalls.

- Iterative approach:

Starting from the initial snake's location, we store the snake's location and the remaining steps that it can move (depth value) in a stack. While the stack is not empty, we pop the last inserted snake and depth pair and calculate valid positions this snake can move to. For each of these positions, we move the snake to that position and store it with the remaining steps it can do minus one.

If one of the popped snake and depth pairs has a depth equal to 0, a distinct valid path has been reached by it so we add it to the total number of distinct valid paths we have found. For these states we do not calculate its valid head adjacent cells as there are no more remaining steps the snake can make.

Algorithms' complexity

Time complexity

- Recursive approach:

Let the variable d that represents the depth of the unique paths we want to count be the size of the given problem. We define its current time by the recurrence equation

$$T(d) = l * T(d-1) + 1$$
$$T(0) = 1$$

where l is the branching factor of the tree defined by the recursive calls made of the algorithm. The branching factor is equal to 3 in the worst case scenario as the maximum possible positions the snake can move to are not 4 because it can not move backwards.

Expressing this recurrence equation in terms of smaller values

$$T(d) = 3\,T(d-1) + 1 = 3\,(3\,T(d-2) + 1) + 1 =$$
$$= 3^2\,T(d-2) + 3 + 1$$

$$T(d) = 3^2 T(d-2) + 3 + 1 = 3^2(3\,T(d-3) + 1) + 3 + 1 =$$
$$= 3^3 T(d-3) + 3^2 + 3 + 1$$

For a value k

$$T(d) = 3^k T(d-k) + 1 + \sum_{i=1}^{k-1} 3^i =$$
$$= 3^k T(d-k) + 1 + \frac{3^k-1}{2}$$

Assume that $d - k = 0$, so $d = k$. This way

$$T(d) = 3^d + \frac{3^d-1}{2}$$

Thus, the worst case time complexity of this algorithm is $O(3^d)$.

- Iterative approach:

Consider a state for the problem defined by the snake's location and the remaining steps it can take from there (initially equal to depth). We traverse all the possible locations the snake can be at starting from the initial location and decreasing the depth by 1 in each movement it makes. Because the snake has a body, there is a maximum of three possible positions the snake can move to from a location. Also, the maximum length of the paths it can make is the value of depth, so we will not reach any location that requires more than that.

This formulates a tree of different states that will have a branching factor of 3 (worst case of 3 valid positions it can move to) and a maximum depth equal to the depth variable.

The algorithm's complexity can be calculated as the sum of all possible movements that can be made in each level of the state tree.

$$S = 3 + 3^3 + 3^9 + \dots + 3^d = \sum_{i=1}^{d} 3^i$$

This expression defines a partial sum with parameters $r = 3$, $a_i = 3^i$ that we can compute as

$$S = 3 \frac{1-3\hat{}d}{1-3} = 3 \frac{3^d-1}{2} = \frac{3^{d+1}-3}{2}$$

We find a upper and lower bound for this partial sum

$$S = \frac{3^{d+1}-3}{2} \geq \frac{3^{d+1}-3^d}{2} = \frac{3^d(3-1)}{2} = 3^d$$

$$S = \frac{3^{d+1}-3}{2} \leq \frac{3^{d+1}}{2} = \frac{3}{2}3^d$$

$$3^d \leq S \leq \frac{3}{2}3^d$$

Thus, the worst case time complexity of this algorithm is $O(3^d)$.

Space complexity

For both algorithms, the maximum amount of memory used will be the length of the longest path explored, that matches the value of depth. This happens as the state space's node traversal is made as a depth-first traversal, so the maximum memory required will be the d depth calls made recursively and stored in memory stack or the d snake locations stored in the programmed stack (equivalent as the parameters are stored in the stack on each recursive call).