**ACI Assignment 1 – Invasion Game**

**Assignment Group – 75**

|  |  |  |
| --- | --- | --- |
| **Name** | **BITS Id** | **Contribution** |
| Avi Krishna Srivatsava | 2020FC04492 | 100% |
| Dola Tejesh | 2020FC04459 | 100% |
| Kommajyosula VNS Kanth | 2020FC04120 | 100% |
| Nareshkumar P | 2020FC04122 | 100% |
| T Navya Madhavi | 2020FC04007 | 100% |

**Player Agent Environment:**

**PEAS Descriptor**

Performance Measure: Maximum no. of land nodes visited using short path and short time.

Environment: Players, Board, Land cell, water cell

Actuators: move, stop

Sensors: cell value (0,1)

**Task Environment**

Agent: Goal Based Agent

Partially Observable

Stochastic

Semi dynamic

Multi Agent

Sequential

Discrete

**Algorithm chosen for the problem statement:**

Depth First Search (DFS)

**Reason for choosing DFS Algorithm:**

Both BFS and DFS uninformed searches suits for the problem statement. Both searches explore the adjacent edges and try to perform action on the edges

1. BFS considers all neighbors so it’s not suitable for decision tree used in puzzle games. But DFS is more suitable for decision tree. As with one decision, we need to traverse further to augument the decision. If we reach the conclusion we won.

2. DFS is more suitable when there are solutions away from source.

Hence, we are using Depth First Search (DFS) algorithm to solve the problem statement.

**Script steps:**

Step 1: Provide input using a user defined function *def run\_Invasion\_Game(N,M,nmmatrix)* .

Where N denotes number of rows, M denotes number of columns and nmmatrix denotes array of rows

Step 2: Convert the array of rows data into a matrix and assign it to a board variable.

*for i in range(N):*

*row = list(nmmatrix.split()[i])*

*board.append(row)*

Step 3: Call a user defined function *def connected\_components(board, N, M)* to start traverse from the unvisited cell value ‘1’.

Step 3a: call a user defined function *def dfs\_algo(board, visited, c, i, j, N, M)* to traverse using depth first search algorithm

Inside dfs\_algo function, Its recursively traverse to each unvisited adjacent cell value ‘1’ till it finds only visited ‘1’ and ‘0’ in the adjacent cells.

Step 4: Once the Step 3a is stopped, no. of ‘1’s visited by the current iteration assign it to *components[]* array variable and the next iteration for player 2 will start traversing to the adjacent cells.

Step 5: Step 3, 3a and 4 will continue till all the ‘1’ is visited in the given matrix array.

Step 6: Get the player2 visited ‘1’ count by using the below lines

*l = connected\_components(board, N, M)*

*ans = 0*

*for i in range(len(l)):*

*if(i%2 == 1):*

*ans += l[i]*

**Space and Time Complexity:**

Time Complexity: **O(NxM)** where N=no.of rows in the matrix board and M=no.of columns in the matrix board

Space complexity: worst case **O(M×N)** in case that the board is filled with lands where DFS goes by M×N deep.

**Output:**

