Name: Vikas Mane
Roll No: B1921152
Class: BE-B

Assignment 8

Title: Implement Tic-Tac-Toe using A* Algorithm

System Requirements: NetBeans IDE

Theory:

- A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. It is complete and will always find a solution if one exists. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

  f (n) = g(n) + h(n)

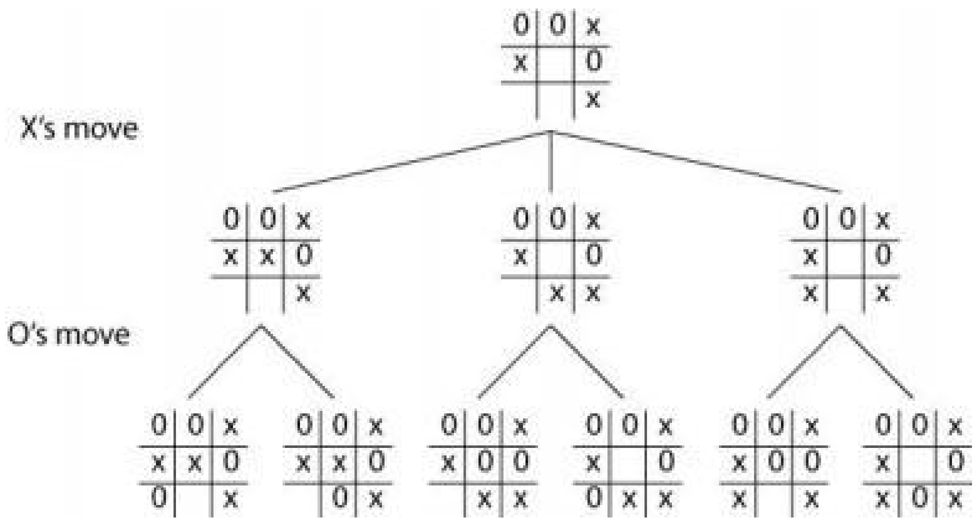
- Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have f (n) = estimated cost of the cheapest solution through n. Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n). The algorithm is identical to UNIFORM-COST-SEARCH except that A* uses g + h instead of g
- A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way
- It uses a distance-plus-cost heuristic function (usually denoted) to determine the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions: the path-cost function, which is the cost from the starting node to the current node (usually denoted) an admissible "heuristic estimate" of the distance to the goal (usually denoted ***h(x)***)
- The part of the function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.If the heuristic h satisfies the additional condition ***h(x) <= d(x,y) + h(y)*** for every edge x, y of the graph (where d denotes the length of that edge), then h is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see closed set below) and A* is equivalent to running Dijkstra's algorithm with the reduced cost.

  d'(x,y) := d(x,y) - h(x) + h(y)

- For example, since airline distance never overestimates actual highway distance, and manhattan distance never overestimates actual moves in the gliding tile.

Tic-Tac-Toe:

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



f(n)=g(n)+h(n)

Where, f(n) is evaluation function g(n) is path cost h(n) is heuristic function

A* is commonly used for the common path finding problem in applications such as games, but was originally designed as a general graph traversal algorithm.

Objective: Use A* algorithm to implement Tic Tac Toe problem

Input: Enter X or 0 then enter position you want to place X or 0 (1 to 9)

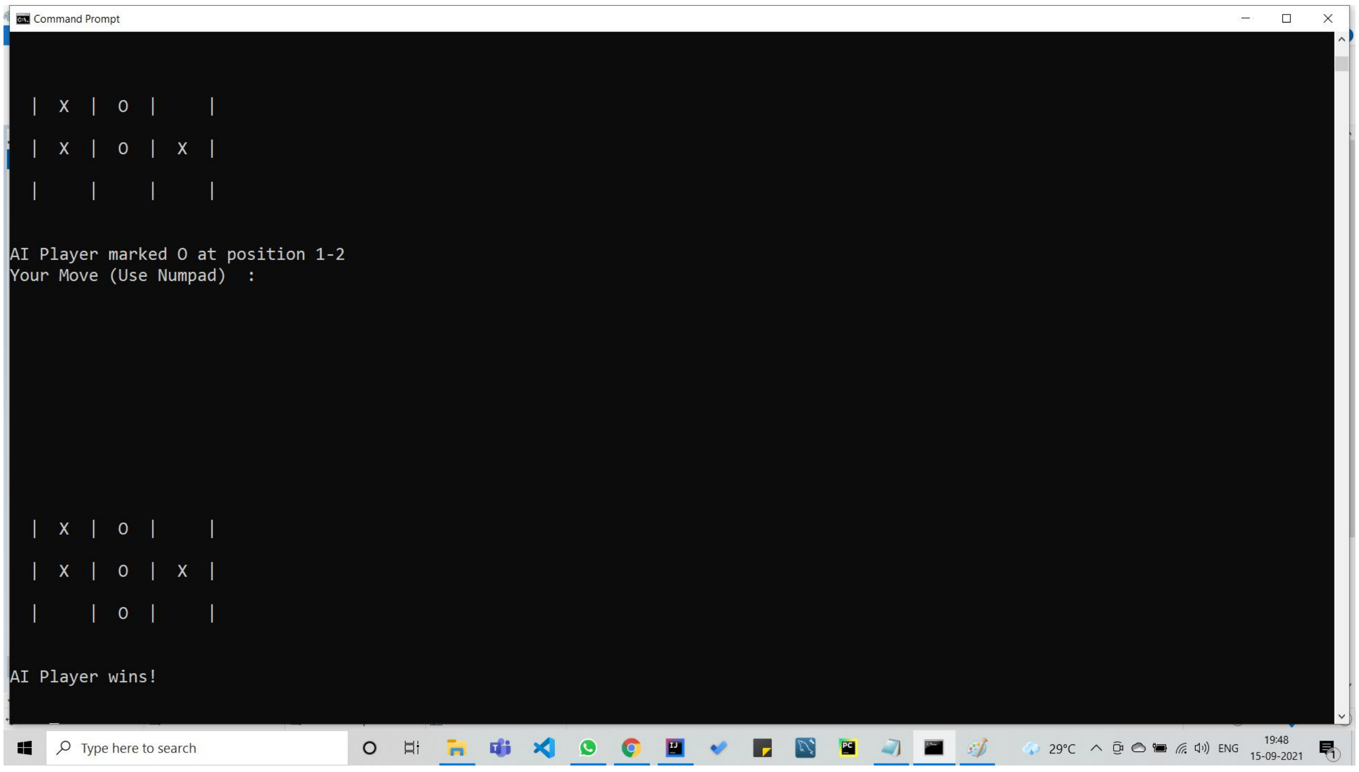
Output: Draw, you win or AI Player wins!

Result / Conclusion: Using A* algorithm we are able to solve Tic Tac Toe problem.

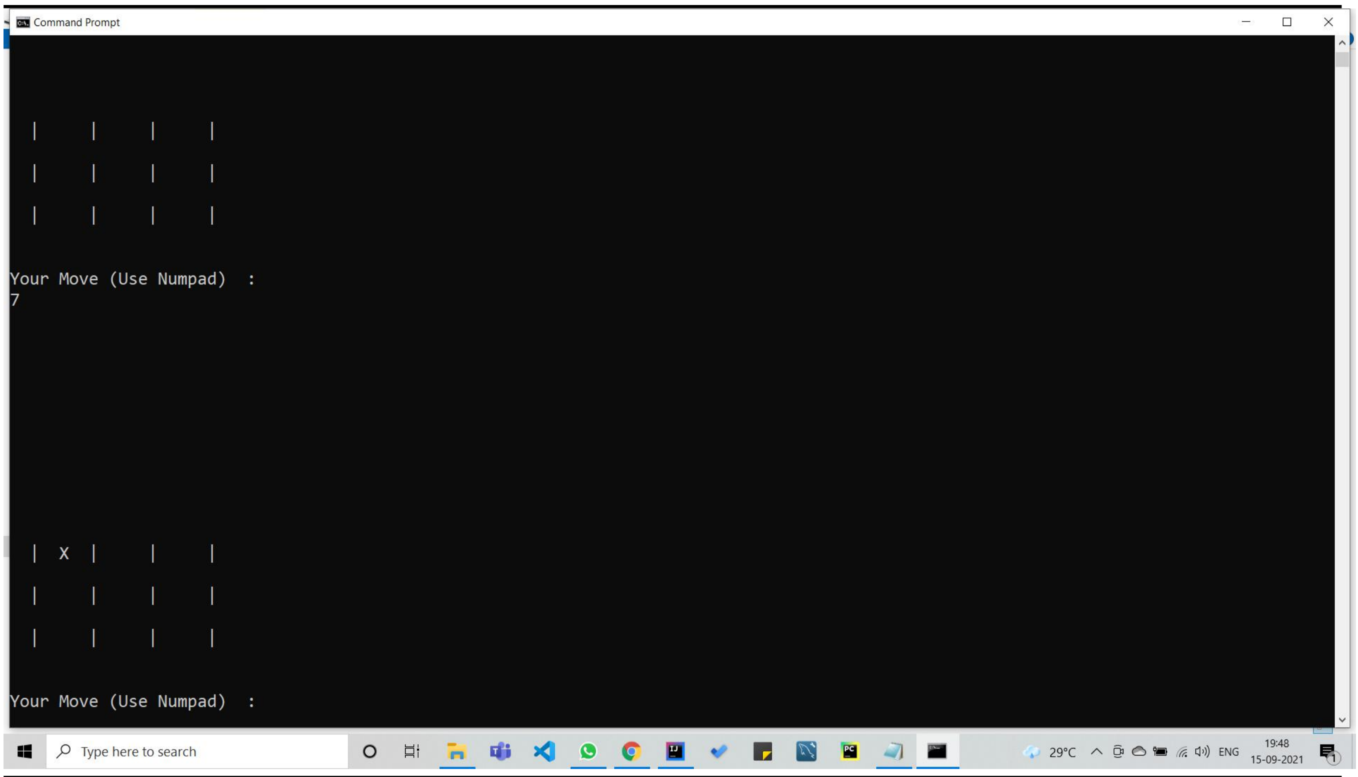
Program:

```
public static void main(String args[])
        {init();
        selectSymbol();
        play();
    }
public static void play() {                              //keep playing while
value returned from checkWin() is "nothing"
        moveCount = 0;
        printBoard();
        while (checkWin(board) == "nothing")
            {if (isHumanTurn()) {
                inputMove();
            } else {
                aiMove();
            }
            printBoard();
            moveCount++;
        }
        if (checkWin(board) == humanPlayer)
            {System.out.println("You win!");
        } else if (checkWin(board) == aiPlayer)
            { System.out.println("AI Player wins!");
        } else {
            System.out.println("Draw!");
        }
    }
```

Output:



```
 | X | O |   |

 | X | O | X |

 |   |   |   |


AI Player marked O at position 1-2
Your Move (Use Numpad)  :








 | X | O |   |

 | X | O | X |

 |   | O |   |


AI Player wins!
```



```
 |   |   |   |

 |   |   |   |

 |   |   |   |


Your Move (Use Numpad)  :
7








 | X |   |   |

 |   |   |   |

 |   |   |   |


Your Move (Use Numpad)  :
```