MAZE

testingClass{

setup_maze(){

Get basic game information such as the number of rows and columns and the number of game walls and the start and end point to start the game

}

print_maze(){

Print the value taken in the previous function for printing on the console

}

}

Node {

Inheritance Inheritance of various features of a point location cost distance to the goal of adhering to structural constraints including not stepping out of the deterministic framework at the beginning

}

A_Star_Search {

Start(){

Start solving the game by taking the initial board to start solving by generating random numbers in the characteristic range for random movement if selected move

}

MazeFirstStart(){

Take the initial constraints of width and height and the initial array for the initial decimal value of the values of east and west and north and south and start the initial movement and if during the movement it should be checked that the point where we are is observed or we hit the wall or In four constraints

}

MazeSecondCheck(){

Drop the priority queue in a loop to test each point to get the answer. In this function we have to check the position of each point again after the previous function in which the constraints were checked to

decide if it is possible Be it our algorithm or not

}

Successor(){

Returns the objective function and the way to reach it, depending on the received array and the empty or full condition of the target.

}

8 QUEENS

simulatedAnnealing(){

1. Randomly move or alter the state
2. Assess the energy of the new state using an objective function

3. Compare the energy to the previous state and decide whether to accept the new solution or reject it based on the current temperature.
4. Repeat until you have converged on an acceptable answer

For a move to be accepted, it must meet one of two requirements

- The move causes a decrease in state energy
- The move increases the state energy but is within the bounds of the temperature. The temperature exponetially decreases as the algorithm progresses. In this way, we avoid getting trapped by local minima early in the process but start to hone in on a viable solution by the end.

}

Queen(){

Determine the characteristics of each queen and capture and set the

characteristics of each queen for use in moves

}

SimulatedAnnealingState{

Build the basic board structure and randomize the queens

getNextState(){

This method selects a random queen and then randomly draws a column.

It updates the queen, returns the page and returns to the next state until it equals the previous value

}

}

Tic Toc Toe

TicTocToe {

resetGame(){

Game recovery

}

endGame(){

The final conditions of the game are checked and implemented in this function

}

playAI(){

Place graphic work in the main function of our class

}

}

Tile {

The properties of each of these 9 items in this class must be set or taken to continue working to solve them

}

MiniMaxAlphaBeta(){

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta <= alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

}