Name: Mahendra Singh Choudhary
Roll No.: B18CSE028
Course: CS323 (Artificial Intelligence)
Instructor: Dr. Yashaswi Verma
Topic: AI to automate the movement of FlappyBird from initial state to target node avoiding obstacles

Introduction :

In this project, we are solving the maze to find an optimal path for the agent to move from its current state to any state that we ask it to if such a path exists. In our problem, the initial state is assumed to be the left-most square in the top row. We can manually place obstacles in the grid and remove already existing ones. The player can move the agent in four directions, top, bottom, left and right, by one step, provided none of the four has an obstacle, i.e. the agent can move one step at a time in either of the four directions if the position where we want to move our agent is not an obstacle.
The scoring is done as the number of steps that our agent has to move in order to reach the goal state.

This *problem* can be formulated *as a search problem,* which can be solved using uninformed or informed Tree or Graph search algorithms.

To inculcate *novelty* in the problem statement, the game has been modified to include continuous queries, i.e., the game organizer can choose new goal states continuously, making the final position of the agent, in the previous query, as initial state and we will need to solve the search problem considering the already covered paths as obstacles.

Problem Definition :

The game organizer can place(remove) obstacles in(from) the grid at any point and asks the agent to move to the node that the organizer points to, taking a minimum number of steps in between the process(when the agent is moving, the grid can not be changed, i.e. the existing obstacles can not be removed and no new obstacle can be placed). The squares on the path that were covered to travel from the initial state to the goal state now act as obstacles(which can be removed all at once in addition to one at a time for the standard obstacles). The Organizer

can ask the agent to travel to different locations and for each such query, we need to show the path from the current location of the agent to the location that we are asked to move to. The agent should not move if the goal state can not be achieved.

State:

The grid, with different cubes that can be covered in three different colours (one for the travelled nodes, one for obstacle nodes, and one for open for movement positions).

Initial state: The present configuration of the grid(at the time the goal state is defined)

Goal State: The agent should be at the position that the organizer wants it to.

Transitions: The agent is allowed to move to any cube that shares one edge with the cube of the current position provided that cube has not already been travelled through and is not an obstacle.

Branching factor: 4(since a position(cube) in the grid can share an edge with four other positions(cubes) in the grid)

Goal Test: Check whether the agent agent is at the goal node or not.

Cost function: Each allowed transition costs one step.

Background Survey:

Our background survey has revealed that there does not exist any implementation for the game for the variant that we are working on. There are a few implementations available for the relaxed variant of our subproblem where the maze is generally pre-defined and the agent is not bombarded with multiple queries. Thus our problem statement is novel in its scope.
For the previously implemented approaches to relaxed variants, all kinds of search algorithms like BFS, DFS, UCS, Astar etc. A few have also approached this as a 2 agent problem where both goal and start nodes move towards each other.

Discussion:

Novelty: I have posed the problem as a continuous query maze search problem, with dynamic(can be changed in between the querries, during the game i.e. a variable maze) obstacle nodes, which has never been done before, to the best of my knowledge.

My solution could be called an engineering-based solution because I have tried to minimize the number of computations required to reach an optimal solution by using an effective algorithm with a nicely derived heuristic function

I have implemented the solution on a maze with fixed dimensions(14*14). It expands reasonably fewer nodes to find the optimal solution. The worst-case scenario arises when the goal state is inaccessible from the initial state and there are fewer obstacles near the initial state, in which case it has to do a lot of computations. For the cases where the solution exists, our algorithm gives the optimal path in quite a little time. Hence, depending on the conditions that whether or not is it provided that the problem will always have a solution, the scalability of our implementation varies.

Demonstration and validation can be found here.
The path taken is highlighted in a shade of red colour, the obstacles in black and rest in a shade of blue.

ALGORITHM:

Astar search turned out to be the most efficient algorithm to solve this problem. We used a minheap to store the nodes on the fringe. A node in our search graph is defined as a tuple - (f,(positionx,positiony),path)

'f' in the tuple is the sum of heuristic function(h) for current state, (positionx, positiony) is a pair of x and y coordinates of the square under consideration and path is the list of nodes expanded to reach the parent node of node at position (positionx, positiony). The node with a lower value of f will be popped out of the fringe before other nodes(with higher values of f). It can be proved that we always get an optimal solution using the fact that our heuristic is consistent.

Heuristic: We have used manhattan distance from the current node position to the goal node position as a heuristic function. Since there can be obstacles in our problem, the manhattan distance will always be less than or equal to the actual number of steps taken, never more, hence our heuristic is admissible. By similar logic, it can be inferred that between any two consecutive nodes, the actual cost of the path will be atleast equal to the manhattan distance between them. By extension, if the two nodes were connected, heuristic value would be one and actual number of steps taken to move between the two nodes will be atleast 1, proving our heuristic's consistency.

For a larger variant of the problem, we can treat the goal node as another agent, and both the agents will move towards each other. This will reduce the number of computations required if the solution exists(otherwise it will worsen the situation, we will need to weigh the pros and cons for the problem according to additional information to know if this will actually be better than our approach).

Algorithm Complexity:

Space Complexity: O(B^D) (because it stores all generated nodes in memory)

Time Complexity : The time complexity depends highly on the heuristic function, in the worst case of an unbounded search space it's O(B^D). It can be stated that the worst case time complexity is of the order of (number_of_states*complexity of heuristic function)

Summary and Conclusions:

In this project, I attempted at solving an interesting path-finding problem with the help of Astar search algorithm. I implemented my solution using pygame library of python and it was a learning experience. As a drawback, the algorithm tends to take more time for cases where no solution exists but it delivers very fast results in cases where a solution exists.

References:

1. https://en.wikipedia.org/wiki/A*_search_algorithm
2. https://en.wikipedia.org/wiki/Maze_solving_algorithm
3. https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/