# PROJECT REPORT:

# HEX AI Maze Solver With Multiple Search Algorithms


**Supervisor :** **Sir** FAROOQ ZAIDI
Sir TALHA SHAHID

**Student's Name :** YASAL KHAN & FILZA SALMAN

**Student's ID :** 22k6004 & 22k5011S

# Table of Contents

# 1. Project

We are implementing 5 search algorithms:

- **Breadth-first search**
- **Uniform-cost search**
- **Iterative deepening search** that uses depth-first tree search as a core component and avoids loops by checking a new node against the current path.
- **Greedy-best first search** using the Manhattan distance as a heuristic.
- **Graph-search A*** uses the Manhattan distance as a heuristic
- **Using filing with input file**

- The format of the input file:

  - First line: the size of the maze width, height.
  - Second line: the position of the Source and Goal. For example: 2 2 1916 meaning source point is (2, 2) and goal point is (19, 16).
  - Third line: the number of obstacles in the maze.
  - The next following line defines the obstacle by the rule:
    - The obstacle is a Convex polygon.
    - A polygon is a set of points that are next to each other clockwise. The last point will be implicitly concatenated with the first point to form a valid convex polygon.

- The output:

  - Graphical representation of polygons.
  - Representation of expanded node.
  - Cost of the expanded node. (cost is 1 at every step)
  - Representation of path from source to goal.
  - Cost of the path. (cost is 1 at every step)

## 2. The Searching algorithms:

a. The Example:

- The maze we will use to run the algorithms is a matrix of size 20x20. The starting point of the arrow is located at point (2, 2) and he have to try to find the shortest path to reach the goal point at (17, 16). Besides, there are also some obstacles in the maze too.

- The input file should have the information below:

```
20 20
2 2 17 16
4
3 14 8 9 3 4
9 6 14 6 14 1 9 1
9 18 13 17 13 13 8 12
16 12 16 8 12 8
```

b. Breath First Search:

- Bearth First Search (*BFS*) is a traversing algorithm that we start at a selected node (starting point) and traverse the graph through exploring all the neighbor nodes and using a queue to keep track of the child nodes that were encountered but not yet explored. In this problem, we also need to use a boolean visited list to avoid processing a node twice and we will explore all the nodes until we find the destination (goal point).

- Algorithm evaluation:

  - *Pros*: BFS is common algorithm, easy to implement and also guarantee to find the optimal solution to reach the goal.
  - *Cons*: it takes a lot of memory to store the discovered node and also consumes so much time to find a solution

- Conclusion:

Breath First Search is a great algorithm to start learning about path finding solution and also makes sure to find the optimal path. But it requires a lot of memory and time, so we need to continue on finding better solutions to both these sides.

c. Uniform Cost Search:

- Uniform Cost Search (*UCS*) is an uninformed search algorithm that uses the lowest cumulative cost to find a path from the source to the destination. Like Breath First Search, it expands the node through all neighboring nodes but this time it selects the next node based on the lowest cumulative cost of reaching that node. In this problem, the cost of all paths is equal to 1, so UCS will behave like BFS.

- Algorithm evaluation:

  - *Pros*: UCS is good way to find the lowest cost path from start to destination in weighted graph because it always considers the path with the least cost to choose.
  - *Cons*: Same like BFS, it needs more memory to keep track of explored nodes and also takes much time because it considers all possible lowest cost paths

- Conclusion:

Uniform Cost Search is similar to Dijikstra's algorithm. Based on the lowest cumulative cost, it can guarantee to find the shortest path with the lowest possible cost although it still does not solve the time and memory problem of the BFS

d. Iterative Deepening Search:

- Iterative Deepening Search (*IDS*) is graph searching strategy that it explores nodes as comprehensively as BFS at each iteration but it requires less memory than BFS. IDS uses a depth-limit version of depth-first search that it prevent DFS from going beyond the given depth. This way helps DFS act as BFS starts exploring nearby nodes around to find the target before increasing the depth limit

Algorithm evaluation:

- *Pros*: IDS take advantage of both DFS in finding targets at long distances with less memory and BFS in considering all neighboring nodes.
- *Cons*: When the target has not been found, it is also time consuming to repeat the DLS algorithm checking again from the first node. Besides, in this problem of finding a way in the maze, DFS's choice of path does not guarantee the best possible result that leads to IDS in this case is inefficiency.

- Conclusion:

IDS is a combination of DFS and BFS by finding the path to the destination but requires less memory than BFS. Although it takes a lot of time, but if you consider carefully that it can repeatedly access the upper-level nodes mutiple time and the number of these nodes is usually smaller than bottom-level ones, so this cost is not significant. Then IDS is still one of the most effective solutions to find the shortest path.

e. Greedy Best First Search:

- Greedy Best First Search (*GBFS*) is a search algorithm that always selects the path it evaluates the best at the moment. To evaluate the best path, it use a heuristic function. In this problem, we will use the Manhattan distance as a heuristic.

- Example of Greedy Best First Search:

- Algorithm evaluation:

  - *Pros*: The GBFS algorithm concept is easy to understand and implement where it only needs to choose the path as close to the destination as possible on each move. This also saves time and memory because no other paths need to be considered
  - *Cons*: The fact that it is “greedy” and does not care about other paths at a time leads to the path it takes which may not be optimal.

- Conclusion:

GBSF is a fundamental idea of informed search strategy. By choosing the calculated move, it greatly reduces the time and memory for redundant moves. However, it does not guarantee the least cost path so there is still need for improvement of this informed search idea.

f. Graph-search A*:

- A* (pronounced as "A star") Search is one of the most popular and best technique used in pathfinding and graph traversals nowadays.

- Example of Graph-search A*:

- Algorithm evaluation:

  - *Pros*: A* Search is complete and optimal. It is used to solve complex search problems with few nodes need to be expanded and it ensures the most optimal path is found.
  - *Cons*: It's completeness depends on that the branching factor is finite and every action has fixed cost. Besides, heuristic fucntion need to be admissible otherwise the path it found may not be optimal.

- Conclusion:

A* algotithm is almost the perfect solution in graph traversals and pathfinding problem. It builds on the principles of Dijkstra's shortest path algorithm to provide a faster way to reach the goal and rarely take a redundant step. Its effectiveness also depends on some conditions such as the heuristic function should be admissible.

# 3. Program :

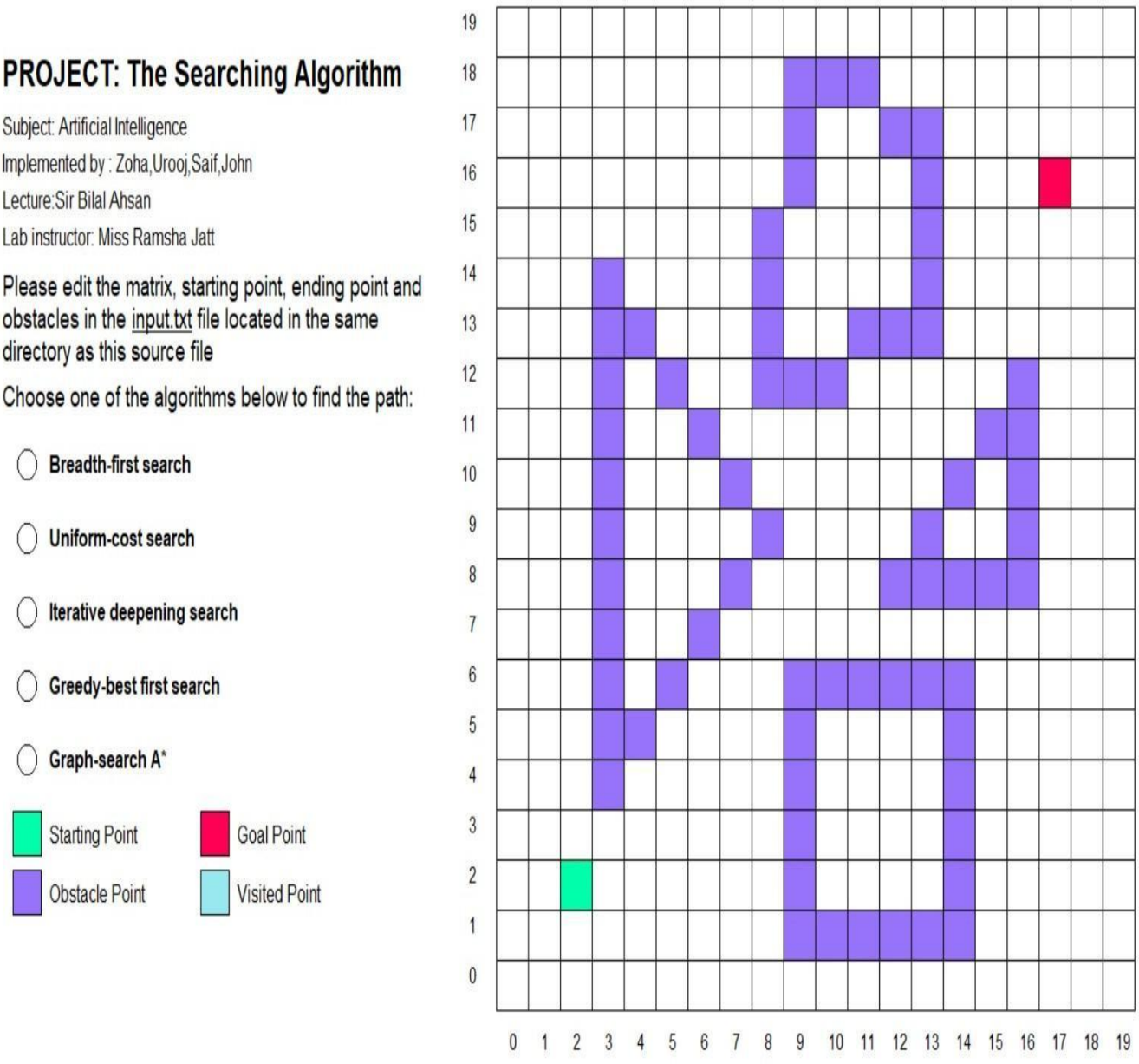


*Image 2: Choose Breath First Search*

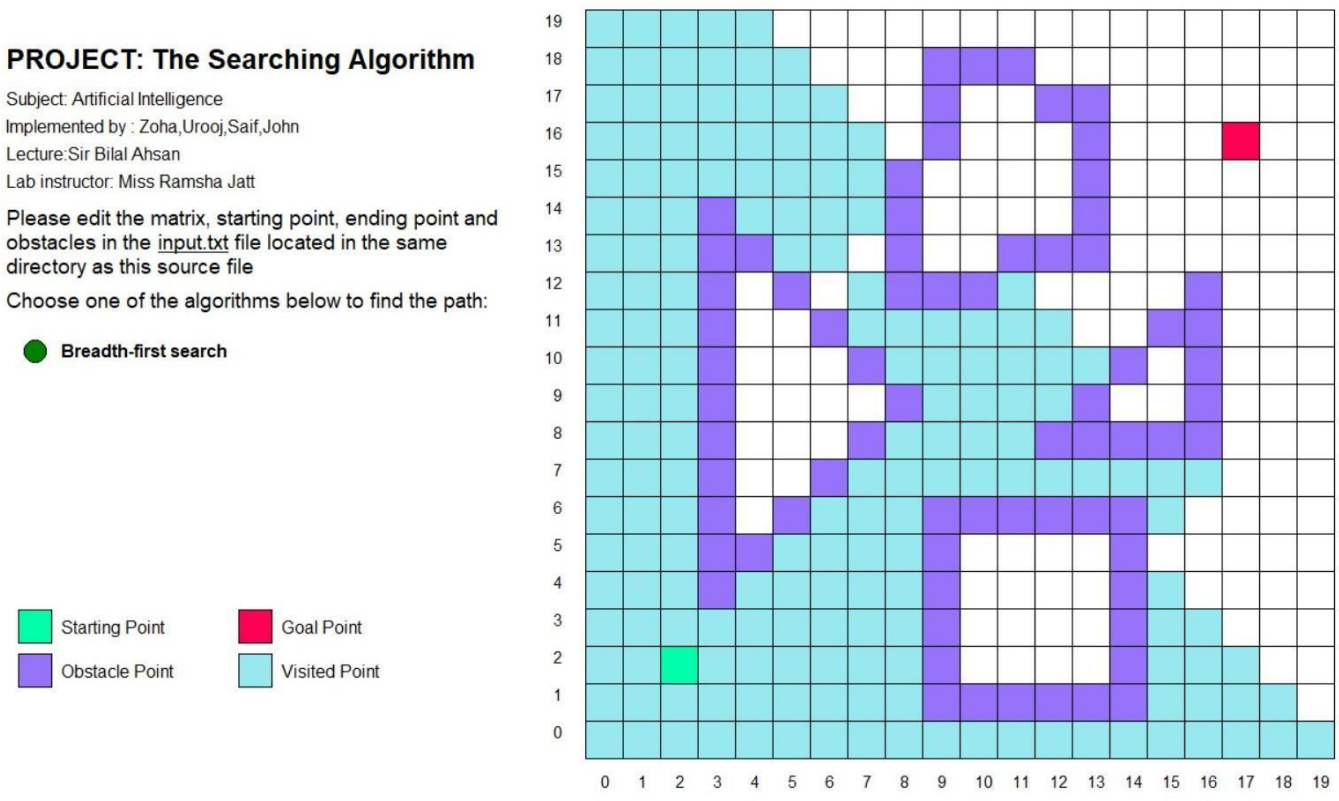


*Image 3: After completing the algorithm, it shows the path and the search cost*