EXPERIMENT 10: A* Search

Aim: Implement an Algorithm in Python for solving A* Search.

Code:

```
import heapq

 class Node:

   def __init__(self, position):

      self.position = position

      self.g = 0  # cost from start node to current node

      self.h = 0  # heuristic estimate of distance from current node to goal node

      self.parent = None


   def __lt__(self, other):

      return (self.g + self.h) < (other.g + other.h)


def heuristic(current, goal):

   return abs(current.position[0] - goal.position[0]) + abs(current.position[1] -
goal.position[1])


def astar(start, goal, grid):

   open_list = []

   closed_set = set()


   heapq.heappush(open_list, start)


   while open_list:

      current = heapq.heappop(open_list)
```

```
    if current == goal:
        path = []
        while current is not None:
            path.append(current.position)
            current = current.parent
        return path[::-1]  # reverse the path to start from the start node

    closed_set.add(current)

    for neighbor_pos in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
        neighbor_position = (current.position[0] + neighbor_pos[0], current.position[1] + neighbor_pos[1])

        if not (0 <= neighbor_position[0] < len(grid) and 0 <= neighbor_position[1] < len(grid[0])):
            continue

        if grid[neighbor_position[0]][neighbor_position[1]] == 1:
            continue

        neighbor = Node(neighbor_position)
        neighbor.g = current.g + 1
        neighbor.h = heuristic(neighbor, goal)
        neighbor.parent = current
```

```
            if neighbor in closed_set:
                continue
            if neighbor not in open_list:
                heapq.heappush(open_list, neighbor)
            else:
                # Update g value if this path is better
                for node in open_list:
                    if node == neighbor and node.g > neighbor.g:
                        node.g = neighbor.g
                        node.parent = neighbor.parent

    return None

# Example usage
start_node = Node((0, 0))
goal_node = Node((4, 4))

grid = [
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0]
]

path = astar(start_node, goal_node, grid)
```

```
print("Path:", path)
```

Output:

Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4)]

Result: Code has been Implemented successfully.