NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Tên :TRẦN KIM KHANH

Mssv:21110318

THỰC HÀNH 4

Giải thích từng dòng code:

Class Point sẽ xử lý các điểm sẽ:

x và y là tọa độ điểm

hàm :Xác định 2 điểm có nằm cùng phía với đường line hay không

```
Tabnine | Edit | Test | Explain | Document | Ask
def rel(self, other, line):
    return line.d(self) * line.d(other) >= 0
```

Hàm :xác định điểm hiện tại có nhìn thấy điểm other qua một đường thẳng line hay không

Nếu 1 trong d1 , d2 ,d3 không thỏa thì trả về true

```
Tabnine | Edit | Test | Explain | Document | Ask
def can_see(self, other, line):
    l1 = self.line_to(line.p1)
    l2 = self.line_to(line.p2)
    d3 = line.d(self) * line.d(other) < 0
    d1 = other.rel(line.p2, l1)
    d2 = other.rel(line.p1, l2)
    return not(d1 and d2 and d3)
```

Hàm : tạo đoạn thẳng và tính khoảng cách giữa 2 điểm

```
    def line_to(self, other):
        return Edge(self, other)

    def heuristic(self, other):
        return euclid_distance(self, other)
```

Class Edge tạo các cạnh lưu điểm đầu và cuối của cạnh

```
class Edge(object):
    def __init__(self, point1, point2):
        self.p1 = point1
        self.p2 = point2
```

hàm : Tìm điểm liền kề

```
def get_adjacent(self, point):
    if point == self.p1:
        return self.p2
    if point == self.p2:
        return self.p1
```

Hàm : tính khoảng cách hướng (signed distance) từ một điểm đến một đoạn thẳng.

```
def d(self, point):
    vect_a = Point(self.p2.x - self.p1.x, self.p2.y - self.p1.y)
    vect_n = Point(-vect_a.y, vect_a.x)
    return vect_n.x * (point.x - self.p1.x) + vect_n.y * (point.y - self.p1.y)
```

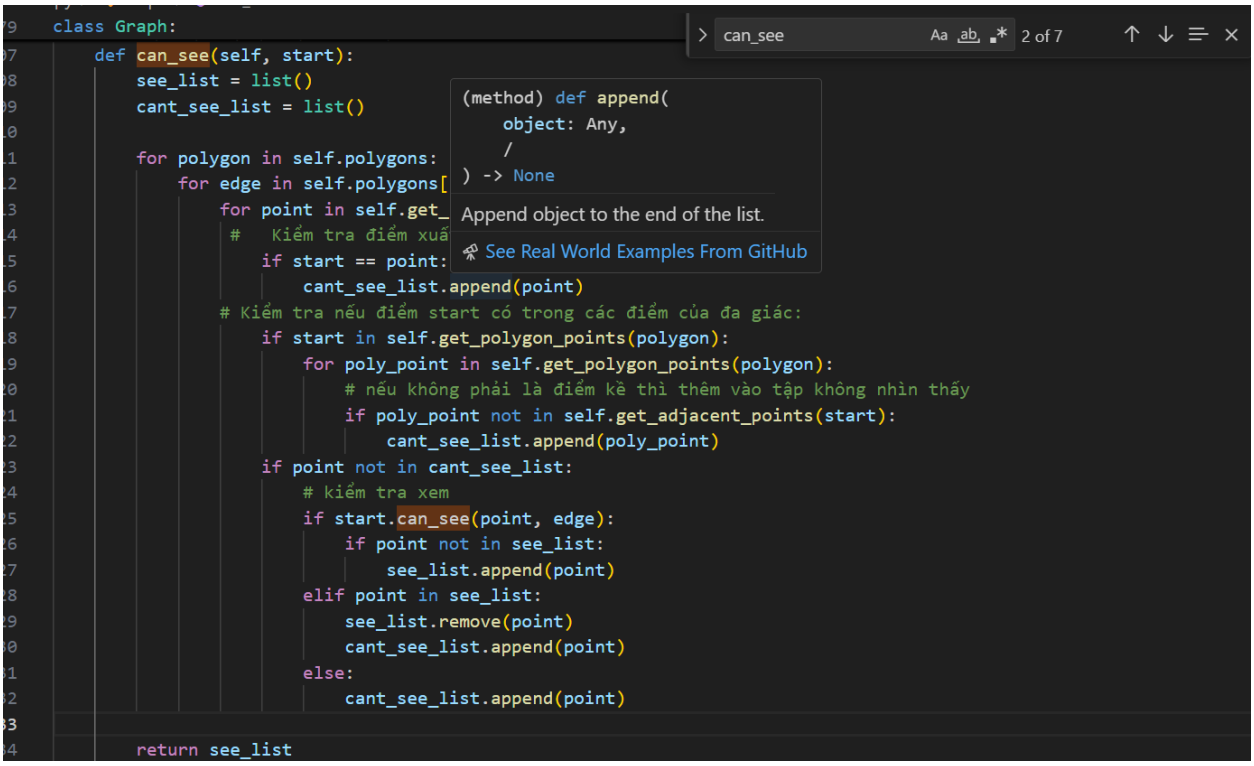
Class Graph:

self.graph: Lưu trữ các điểm trong đồ thị, mỗi điểm là khóa, giá trị là tập hợp các điểm kề với điểm đó.

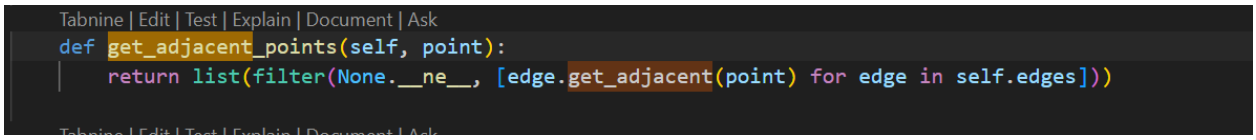
self.edges: Lưu trữ tất cả các cạnh trong đồ thị, mỗi cạnh là một đối tượng Edge.

self.polygons: Lưu trữ các đa giác, mỗi đa giác có một ID duy nhất (là khóa), giá trị là tập hợp các cạnh của đa giác đó.

pid: Biến lưu ID của đa giác, giúp phân biệt các đa giác khác nhau trong đồ thị.



Hàm get_adjacent_points trả về một danh sách các điểm kề với điểm point trong đồ thị, thông qua việc duyệt qua tất cả các cạnh và lấy các điểm kề từ mỗi cạnh.



Hàm can_see(self, start) kiểm tra tất cả các điểm có thể nhìn thấy từ điểm start

```python
class Graph:
    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self, polygons):
        self.graph = defaultdict(set)
        self.edges = set()
        self.polygons = defaultdict(set)
        pid = 0

        for polygon in polygons:
            if len(polygon) == 2:
                polygon.pop()
            if polygon[0] == polygon[-1]:
                self.add_point(polygon[0])
            else:
                for i, point in enumerate(polygon):
                    neighbor_point = polygon[(i + 1) % len(polygon)]
                    edge = Edge(point, neighbor_point)
                    if len(polygon) > 2:
                        point.polygon_id = pid
                        neighbor_point.polygon_id = pid
                        self.polygons[pid].add(edge)
                    self.add_edge(edge)
                if len(polygon) > 2:
                    pid += 1
```

hàm lấy các điểm trong đa giác

```python
Tabnine | Edit | Test | Explain | Document | Ask
def get_polygon_points(self, index):
    point_set = set()
    for edge in self.polygons[index]:
        point_set.add(edge.p1)
        point_set.add(edge.p2)
    return point_set
```

Thuật toán A* hoặc greedy tùy vào hàm func sử dụng

.

```
Tabnine | Edit | Test | Explain | Document | Ask
def search(graph, start, goal, func):
    closed = set()
    queue = PriorityQueue()
    queue.put((0 + func(graph, start), start))
    if start not in closed:
        closed.add(start)
    while not queue.empty():
        cost, node = queue.get()
        if node == goal:
            return node
        for i in graph.can_see(node):
            new_cost = node.g + euclid_distance(node, i)
            if i not in closed or new_cost < i.g:
                closed.add(i)
                i.g = new_cost
                i.pre = node
                new_cost = func(graph, i)
                queue.put((new_cost, i))

    return node
```

Thuật toán bfs:

```
Tabnine | Edit | Test | Explain | Document | Ask
def bfs(graph, start, goal):
    queue = Queue()
    queue.put(start)
    visited = set()
    visited.add(start)
    while not queue.empty():
        node = queue.get()
        if node == goal:
            return node
        for neighbor in graph.can_see(node):
            if neighbor not in visited:
                neighbor.pre = node
                visited.add(neighbor)
                queue.put(neighbor)
    return None
```

Thuật toán DFS:

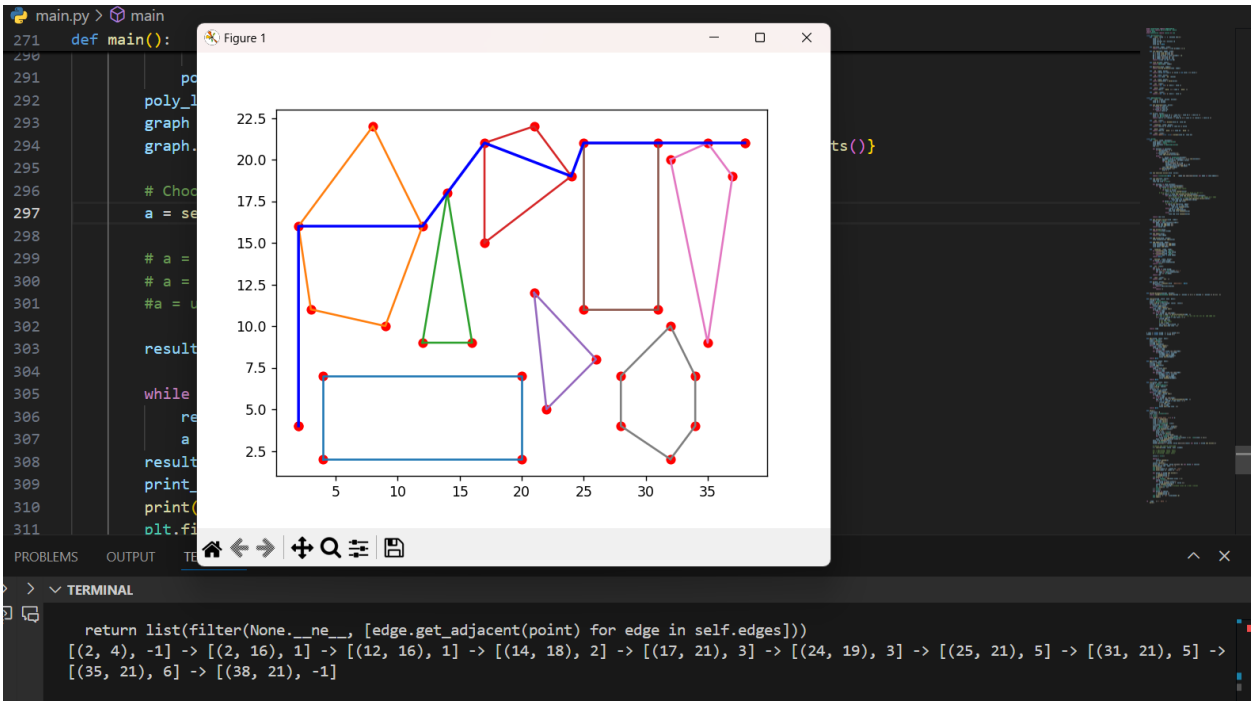
```
Tabnine | Edit | Test | Explain | Document | Ask
def dfs(graph, start, goal):
    stack = [start]
    visited = set()
    visited.add(start)
    while stack:
        node = stack.pop()
        if node == goal:
            return node
        for neighbor in graph.can_see(node):
            if neighbor not in visited:
                neighbor.pre = node
                visited.add(neighbor)
                stack.append(neighbor)
    return None
```

thuật toán USC :

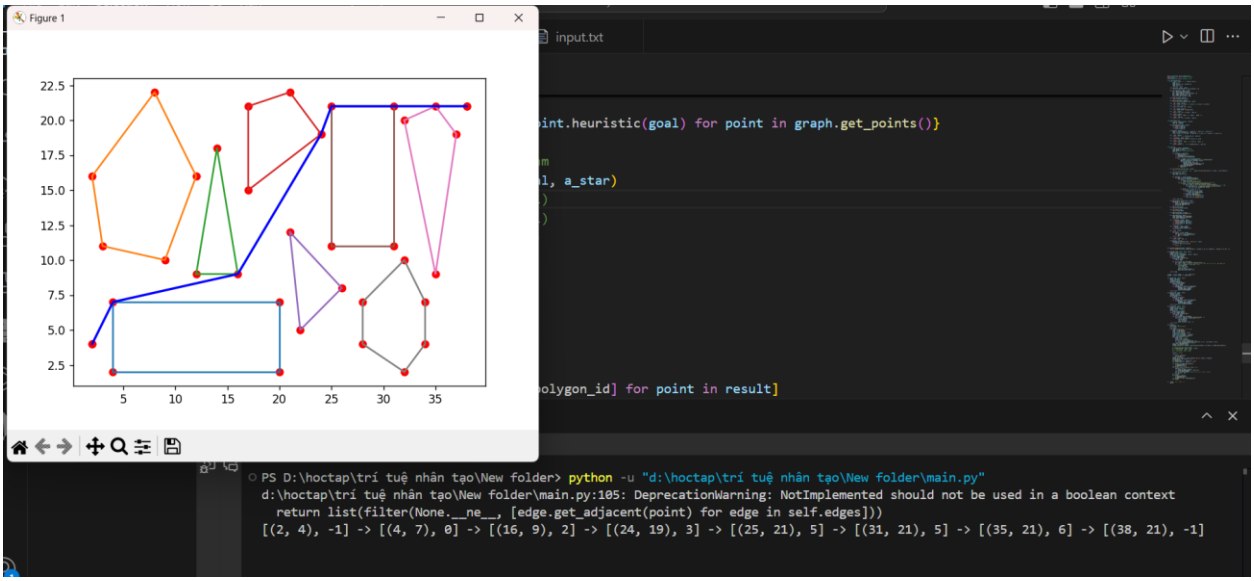
```
Tabnine | Edit | Test | Explain | Document | Ask
def ucs(graph, start, goal):
    closed = set()
    queue = PriorityQueue()
    queue.put((0, start))
    start.g = 0
    if start not in closed:
        closed.add(start)
    while not queue.empty():
        cost, node = queue.get()
        if node == goal:
            return node
        for i in graph.can_see(node):
            new_cost = node.g + euclid_distance(node, i)
            if i not in closed or new_cost < i.g:
                closed.add(i)
                i.g = new_cost
                i.pre = node
                queue.put((new_cost, i))
    return None
```

Kết quả của từng thuật toán:

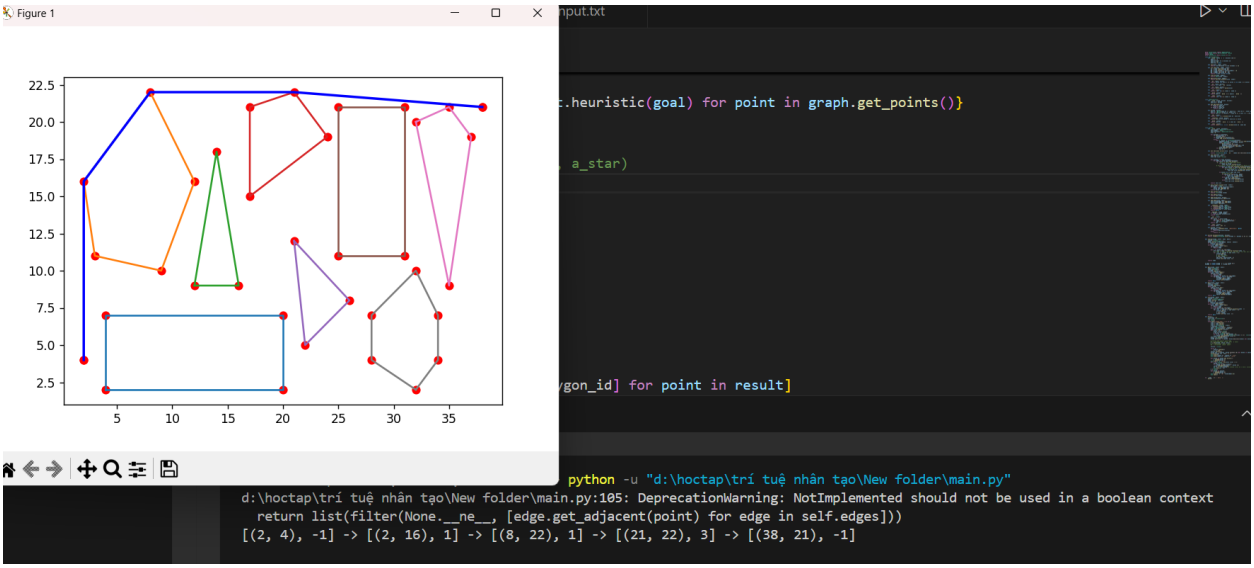
1 Tìm kiếm tham lam: Greedy Search:



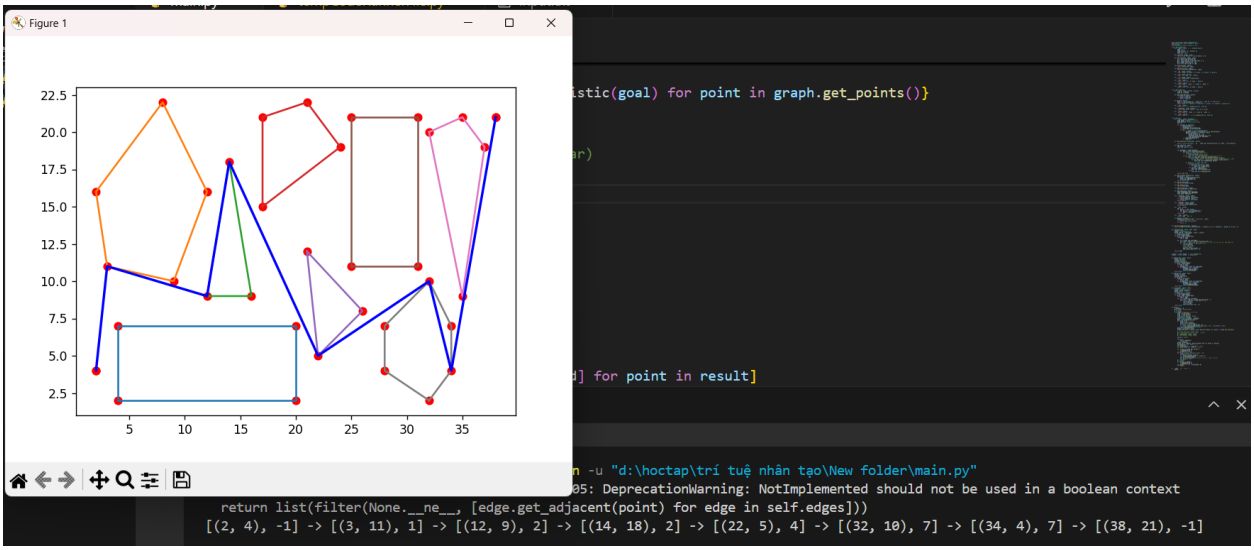
2 thuật toán A*



3 thuật toán BFS:



4 thuật toán DFS:



Thuật toán USC:

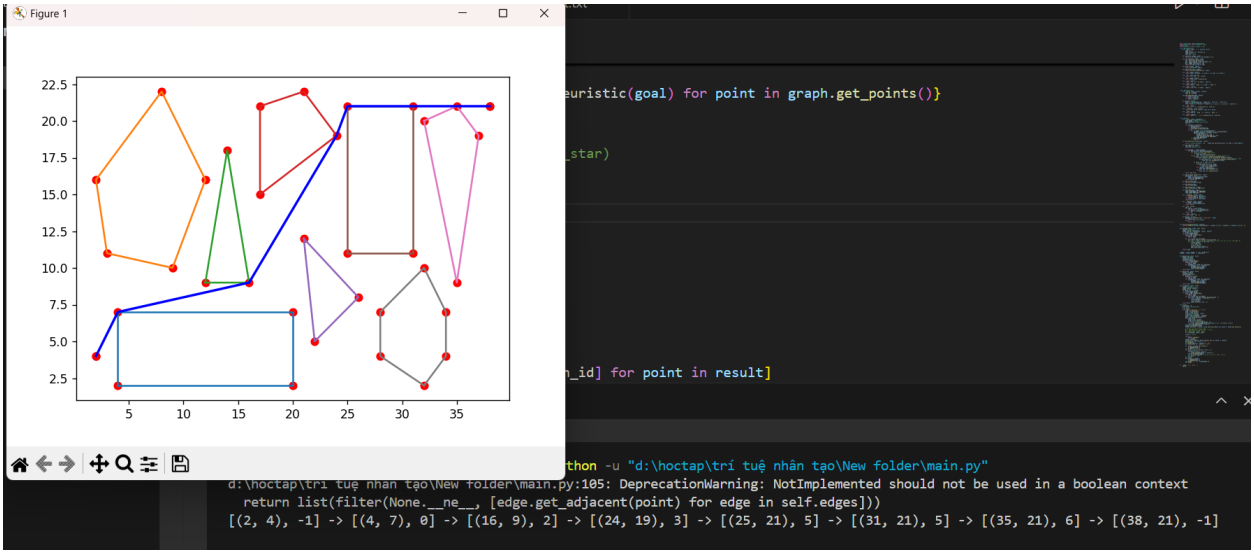


Figure 1
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
5
10
15
20
25
30
35
euristic(goal) for point in graph.get_points()}
_star)
_id] for point in result]
thon -u "d:\hoctap\trí tuệ nhân tạo\New folder\main.py"
d:\hoctap\trí tuệ nhân tạo\New folder\main.py:105: DeprecationWarning: NotImplemented should not be used in a boolean context
return list(filter(None.__ne__, [edge.get_adjacent(point) for edge in self.edges]))
[(2, 4), -1] -> [(4, 7), 0] -> [(16, 9), 2] -> [(24, 19), 3] -> [(25, 21), 5] -> [(31, 21), 5] -> [(35, 21), 6] -> [(38, 21), -1]