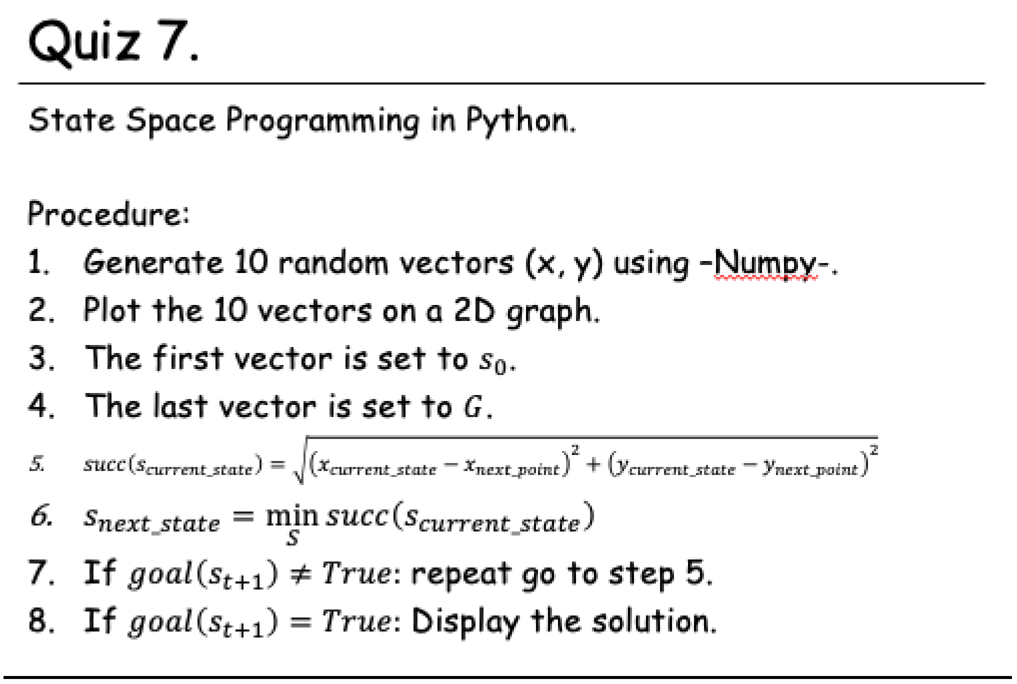
Name: Brendon Wong Tiing Sheng
Student Id: BCS22020021

<u>QUIZ 3</u>



Quiz 7.

State Space Programming in Python.

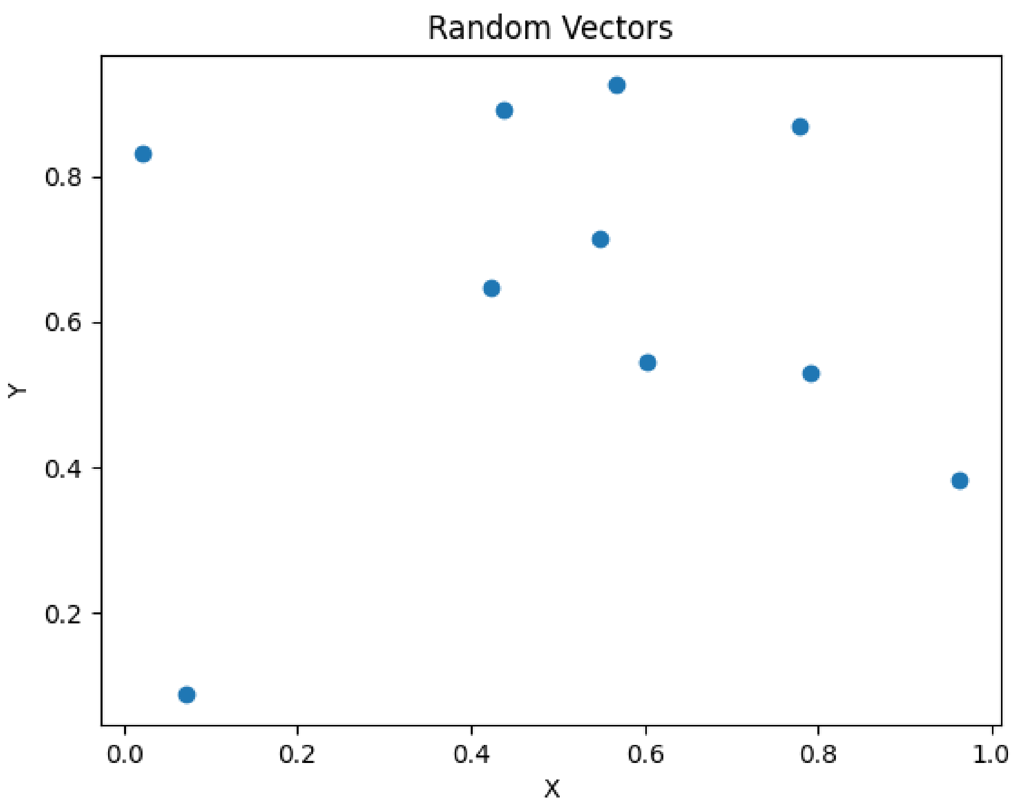
Procedure:

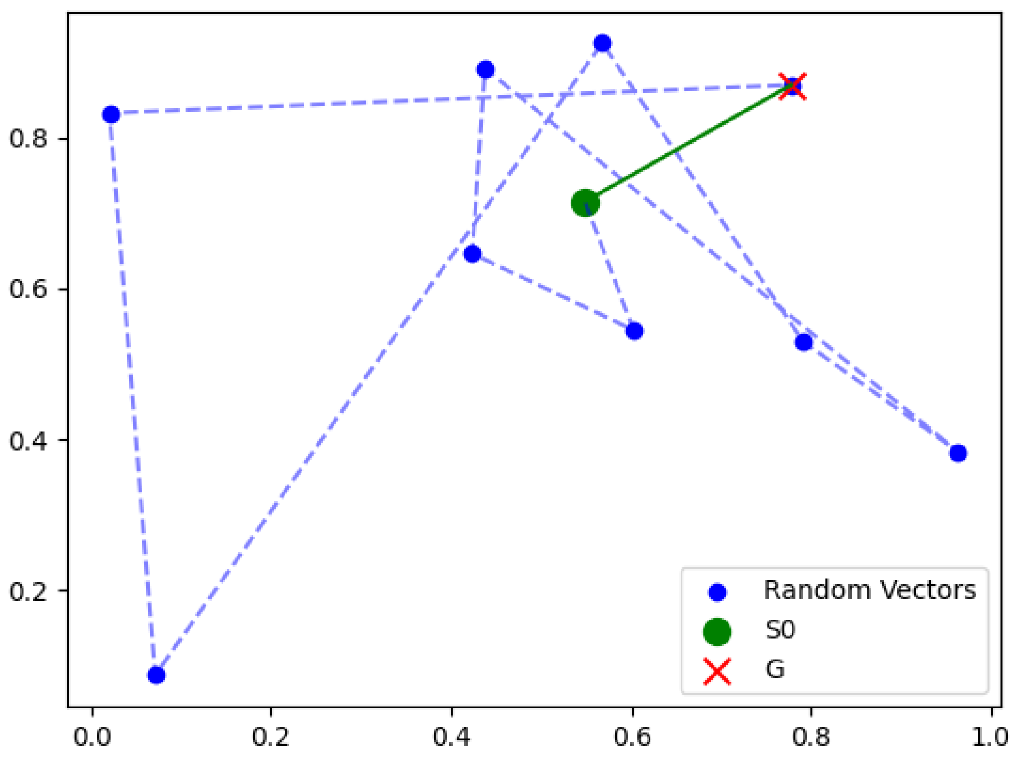
1. Generate 10 random vectors (x, y) using -Numpy-.
2. Plot the 10 vectors on a 2D graph.
3. The first vector is set to s_0.
4. The last vector is set to G.
5. $succ(s_{current\_state}) = \sqrt{(x_{current\_state} - x_{next\_point})^2 + (y_{current\_state} - y_{next\_point})^2}$
6. $s_{next\_state} = \min_{S} succ(s_{current\_state})$
7. If $goal(s_{t+1}) \neq True$: repeat go to step 5.
8. If $goal(s_{t+1}) = True$: Display the solution.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from heapq import heappop, heappush

# Step 1: Generate 10 random vectors (x, y) using NumPy
np.random.seed(0)
vectors = np.random.rand(10, 2)

# Step 2: Plot the 10 vectors on a 2D graph
plt.scatter(vectors[:, 0], vectors[:, 1])
plt.title('Random Vectors')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

# Step 3: Set the first vector as S0 and the last vector as G
S0 = tuple(vectors[0])
G = tuple(vectors[-1])

# Step 4: Calculate the Euclidean distance
def euclidean_distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] -
point2[1]) ** 2)

# Step 5: Implement Dijkstra's algorithm
def dijkstra(start, goal, vectors):
    distances = {tuple(vector): np.inf for vector in vectors}
    distances[start] = 0
    queue = [(0, start)]

    while queue:
        current_dist, current_state = heappop(queue)

        if current_state == goal:
            break

        for vector in vectors:
            if tuple(vector) != current_state:
                dist = euclidean_distance(np.array(current_state),
np.array(vector))
                new_dist = current_dist + dist
                if new_dist < distances[tuple(vector)]:
                    distances[tuple(vector)] = new_dist
                    heappush(queue, (new_dist, tuple(vector)))

    return distances

# Step 6: Apply Dijkstra's algorithm and get the path
distances = dijkstra(S0, G, vectors)
path = [G]
current = G
while current != S0:
    for vector in vectors:
        if euclidean_distance(np.array(current), np.array(vector)) +
distances[tuple(vector)] == distances[current]:
            path.insert(0, tuple(vector))
            current = tuple(vector)
            break

# Step 7: Display the path
for i, point in enumerate(path):
    print(f"Step {i+1}: {point}")

# Step 8: Plot the path
x = [point[0] for point in vectors]
y = [point[1] for point in vectors]
plt.scatter(x, y, label="Random Vectors", color='blue')
plt.plot(x, y, color='blue', alpha=0.5, linestyle='--')
plt.scatter(S0[0], S0[1], label="S0", color='green', marker='o', s=100)
plt.scatter(G[0], G[1], label="G", color='red', marker='x', s=100)
plt.plot([point[0] for point in path], [point[1] for point in path],
color='green')
plt.legend()
plt.show()
```

Result:




0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
1.0
Random Vectors
S0
G