Notebook IndexError

```
import numpy as np
```

Define the Maze environment

```
maze = np.array([
  [0, 0, 0, 0, 0],
  [0, -1, -1, -1, 0],
  [0, 0, 0, 0, 0],
  [0, -1, -1, -1, 0],
  [0, 0, 0, 0, 0],
])
start_state = (0, 0)
goal_state = (4, 4)
```

Define Hyperparamters

```
epsilon = 0.1
learning_rate = 0.5
discount_factor = 0.9
max_episodes = 1000
max_steps = 100
```

Initialize the Q-table

```
num_rows, num_cols = maze.shape
num_actions = 4
Q = np.zeros((num_rows, num_cols, num_actions))
```

Q-learning algorithm

```
for episode in range(max_episodes):
  state = start_state
  total_reward = 0
  for step in range(max_steps):
    if np.random.uniform(0, 1) < epsilon:
      action = np.random.randint(num_actions)
    else:
      action = np.argmax(Q[state])

    # Calculate next state based on action
    if action == 0:
      next_state = (state[0] - 1, state[1])
    elif action == 1:
      next_state = (state[0] + 1, state[1])
    elif action == 2:
      next_state = (state[0], state[1] - 1)
    else:
      next_state = (state[0], state[1] + 1)

    # Check if next_state is within maze boundaries
    if 0 <= next_state[0] < num_rows and 0 <= next_state[1] < num_cols:
      # If within boundaries, update Q-value
      reward = maze[next_state]
      Q[state][action] += learning_rate * (reward + discount_factor * np.max(Q[next_state]) - Q[state][action])
      state = next_state
    else:
      # If outside boundaries, apply penalty and stay in current state
      reward = -5  # Penalty for hitting a wall
      Q[state][action] += learning_rate * (reward - Q[state][action]) # Update Q-value with penalty and no state change

    total_reward += reward
    if state == goal_state:
      break
  print(f"Episode {episode + 1}: Total reward = {total_reward}")
```

```
Episode 947: Total reward = -22
Episode 948: Total reward = -22
Episode 949: Total reward = -39
Episode 950: Total reward = -23
Episode 951: Total reward = -22
Episode 952: Total reward = -11
Episode 953: Total reward = -36
Episode 954: Total reward = -27
Episode 955: Total reward = -16
Episode 956: Total reward = -18
Episode 957: Total reward = -21
Episode 958: Total reward = -27
Episode 959: Total reward = -26
Episode 960: Total reward = -5
Episode 961: Total reward = -28
Episode 962: Total reward = -26
Episode 963: Total reward = -16
Episode 964: Total reward = -11
Episode 965: Total reward = -21
Episode 966: Total reward = -21
Episode 967: Total reward = -25
Episode 968: Total reward = -19
Episode 969: Total reward = -23
Episode 970: Total reward = -17
Episode 971: Total reward = -16
Episode 972: Total reward = -24
Episode 973: Total reward = -12
Episode 974: Total reward = -12
Episode 975: Total reward = -28
Episode 976: Total reward = -11
Episode 977: Total reward = -19
Episode 978: Total reward = -15
Episode 979: Total reward = -27
Episode 980: Total reward = -25
Episode 981: Total reward = -20
Episode 982: Total reward = -22
Episode 983: Total reward = -13
Episode 984: Total reward = -16
Episode 985: Total reward = -12
Episode 986: Total reward = -27
Episode 987: Total reward = -21
Episode 988: Total reward = -16
Episode 989: Total reward = -21
Episode 990: Total reward = -22
Episode 991: Total reward = -21
Episode 992: Total reward = -5
Episode 993: Total reward = -8
Episode 994: Total reward = -12
Episode 995: Total reward = -11
Episode 996: Total reward = -27
Episode 997: Total reward = -25
Episode 998: Total reward = -22
Episode 999: Total reward = -25
Episode 1000: Total reward = -15
```

Testing the learned policy

```
state = start_state
steps = 0
success_count = 0
while state != goal_state and steps < max_steps:
  action = np.argmax(Q[state])
  if action == 0:
    next_state = (state[0] - 1, state[1])
  elif action == 1:
    next_state = (state[0] + 1, state[1])
  elif action == 2:
    next_state = (state[0], state[1] - 1)
  else:
    next_state = (state[0], state[1] + 1)

  if 0 <= next_state[0] < num_rows and 0 <= next_state[1] < num_cols:
    state = next_state
    steps += 1
    if state == goal_state:
      success_count += 1
  else:
    break
print(f"Testing Results - Steps: {steps}, Success Rate: {success_count/max_steps}")
```

```
Testing Results - Steps: 100, Success Rate: 0.0
```