24F-0040
Laiba
Assignment 01
Artificial Intelligence
Question 07

Github Link:

https://github.com/laibaaliraza/AI-path-finder.git

Comprehensive Report:

AI Path Finding Project using Different Search Algorithms

Introduction:

This project is about using AI search techniques to find path from start to goal on a grid. It shows how different search algorithms explore the grid and find goal. we also use matplotlib to show grid so that we can see how nodes are visited step by step and also some of the dynamic obstacles are added randomly to make it more real like working.

Problem Definition as search problem:

Initial State:
 start position is 0 0 top left corner

Actions:
 Agent can move in 8 directions
 Up, down, left, right, top-left, bottom-right

Transition Model:
 Agent moves to next cell if its not wall or obstacle

Goal State:
goal position is 14 14 bottom right corner

Path Cost:
Each move cost 1

Grid and Visualization:

Grid is 15 by 15 implemented using numpy array
each cell value means:
0 empty
2 start
3 goal
5 visited node
6 final path
7 obstacle

Library matplotlib shows the grid and updates each step so we can see how it works

Dynamic Obstacles

Random obstacles are added while algorithm is running. This make problem harder and more like real world.obstacles appear with some probability

Explanation:

Breadth First Search (BFS):

BFS works level by level
uses queue
explore all neighbors first then go deeper
guarantee shortest path if all moves equal cost
implemented using deque visited set and parent dict

Depth First Search (DFS):

DFS goes deep in one path before backtracking
uses stack
fast sometimes but not guarantee shortest path
implemented using stack list visited set parent dict

Uniform Cost Search (UCS):

UCS expand node with lowest cost
uses priority queue
in this project cost is same so behaves like BFS but always optimal
implemented using heapq cost etc

Depth Limited Search (DLS):

DFS with depth limit
stop if depth limit reached
may fail if goal far
implemented using stack of (node,depth) visited set and parent

Iterative Deepening DFS (IDDFS):

run DLS repeatedly with increasing depth
combines BFS and DFS advantage
optimal solution with less memory
implemented with loop calling DLS for depth 0 to 20

Bidirectional Search

search from start and goal at same time
stop when both search meet
faster because search space reduced
implemented using two queues two visited sets and parent dict

Pros and Cons

BFS
pros find shortest path complete
cons high memory slow for large grid

DFS
pros low memory simple
cons not shortest path can stuck

UCS
pros always optimal
cons slower and high memory

DLS
pros prevent infinite loops low memory
cons may miss solution hard to choose limit

IDDFS
pros optimal less memory than BFS
cons repeated searching slower

Bidirectional
pros very fast less search space
cons more complex not always work

Test Cases

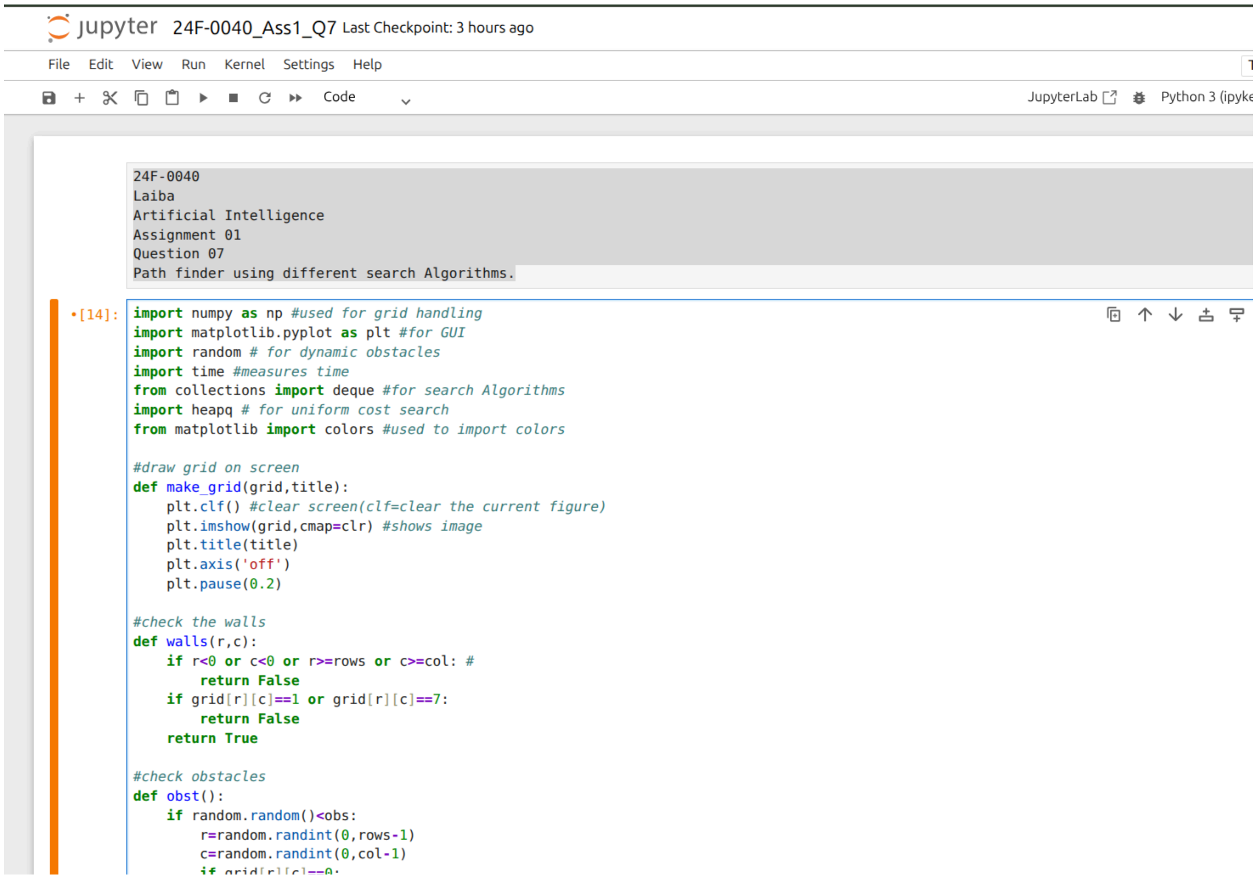
Best Case
few obstacles path mostly free
algorithm reach goal fast
nodes visited less
final path short
GUI shows fast completion

Worst Case
many obstacles appear
algorithm search almost whole grid
search takes long
lots of visited nodes
final path long
GUI shows struggle or failure

Conclusion:

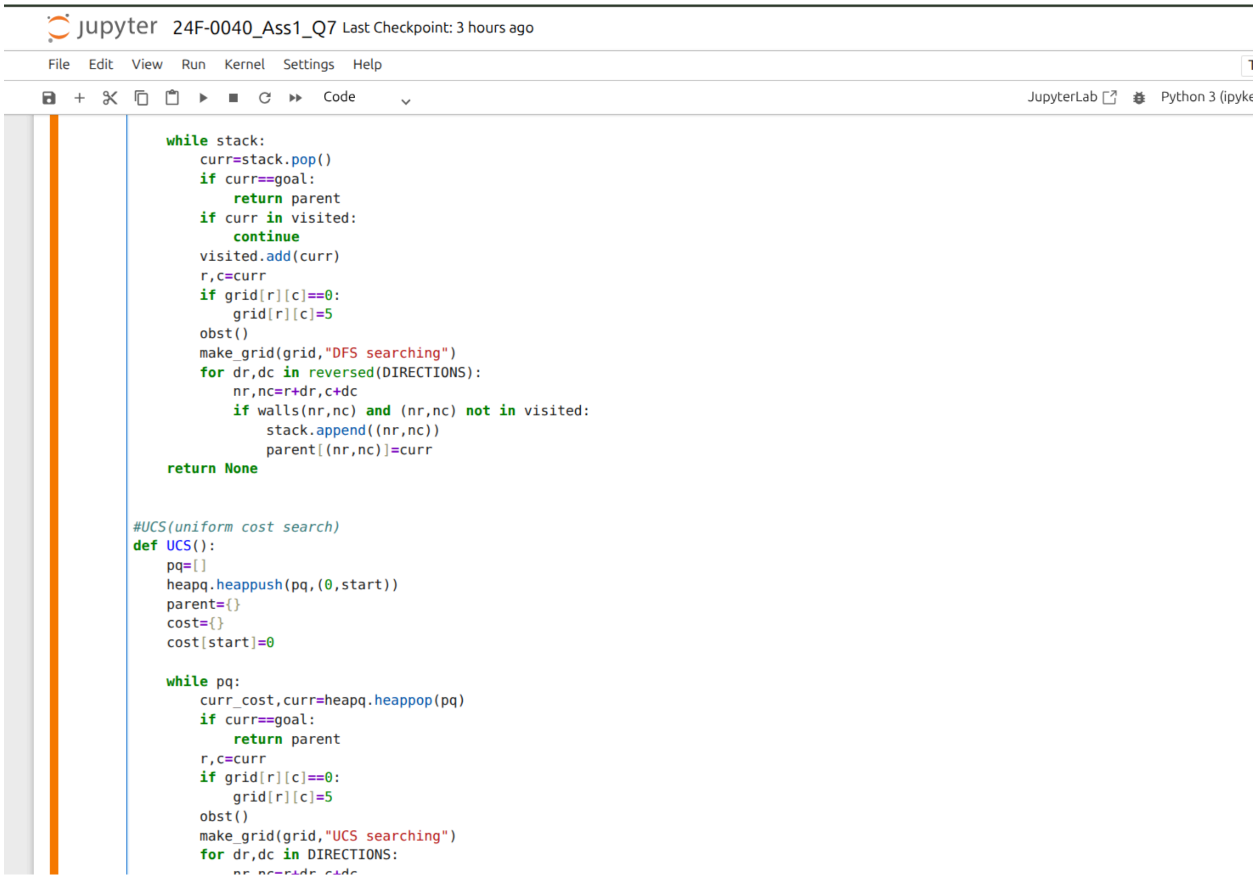
This project shows AI search algorithms and how they explore the grid
Dynamic obstacles make problem real
Visualization help understand how each algorithm works

Screenshots:

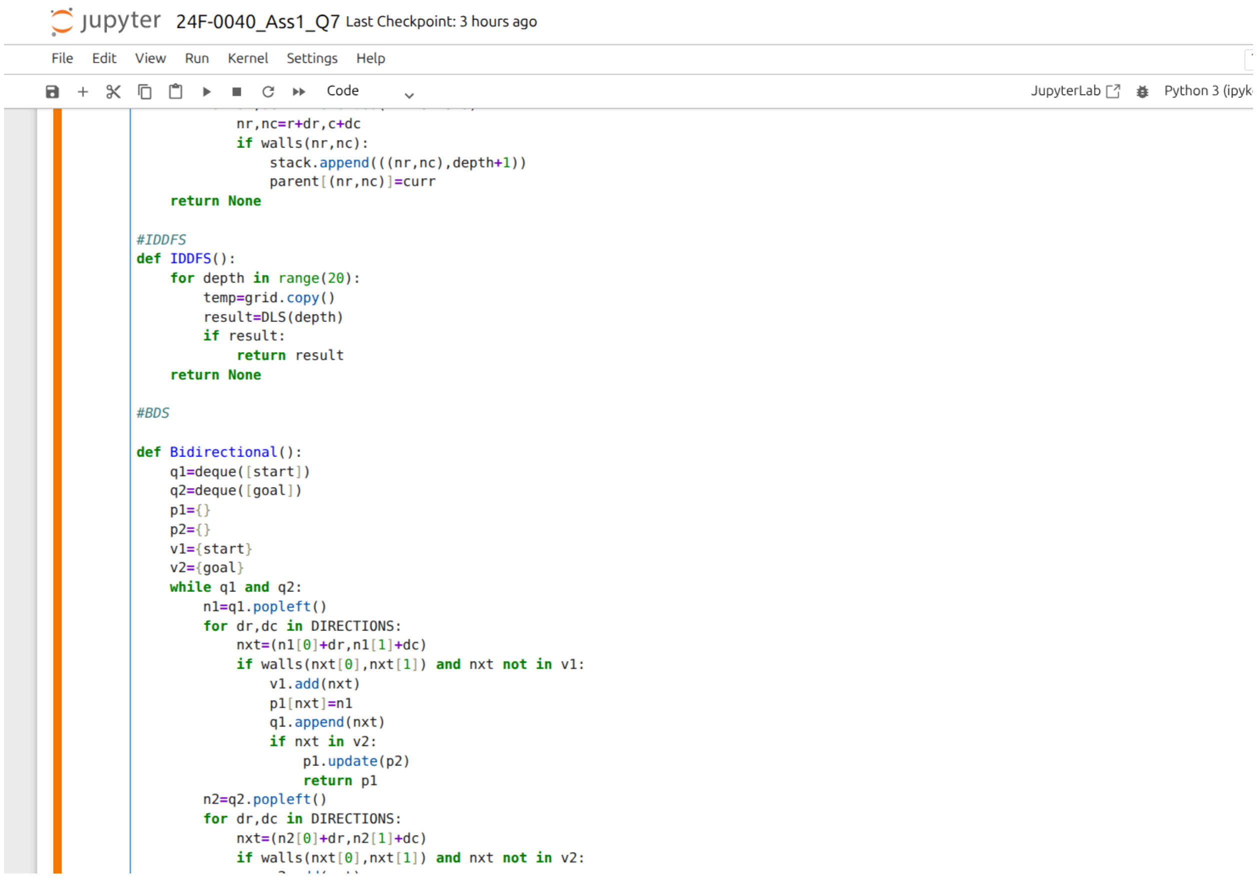

jupyter 24F-0040_Ass1_Q7 Last Checkpoint: 3 hours ago
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipyke
•[14]:

```
24F-0040
Laiba
Artificial Intelligence
Assignment 01
Question 07
Path finder using different search Algorithms.
```

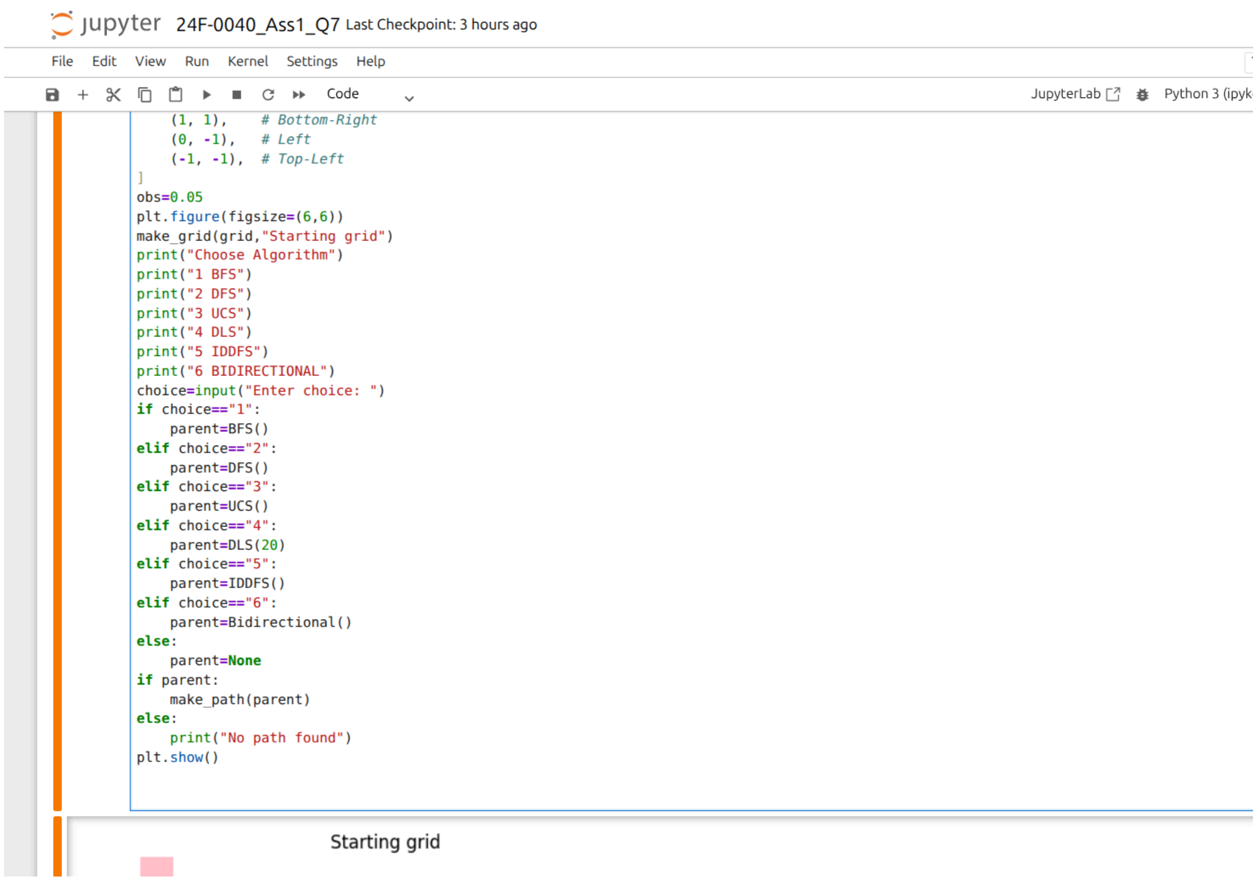
```python
import numpy as np #used for grid handling
import matplotlib.pyplot as plt #for GUI
import random # for dynamic obstacles
import time #measures time
from collections import deque #for search Algorithms
import heapq # for uniform cost search
from matplotlib import colors #used to import colors

#draw grid on screen
def make_grid(grid,title):
    plt.clf() #clear screen(clf=clear the current figure)
    plt.imshow(grid,cmap=clr) #shows image
    plt.title(title)
    plt.axis('off')
    plt.pause(0.2)

#check the walls
def walls(r,c):
    if r<0 or c<0 or r>=rows or c>=col: #
        return False
    if grid[r][c]==1 or grid[r][c]==7:
        return False
    return True

#check obstacles
def obst():
    if random.random()<obs:
        r=random.randint(0,rows-1)
        c=random.randint(0,col-1)
        if grid[r][c]==0:
```


jupyter
24F-0040_Ass1_Q7 Last Checkpoint: 3 hours ago
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipyke

```
    while stack:
        curr=stack.pop()
        if curr==goal:
            return parent
        if curr in visited:
            continue
        visited.add(curr)
        r,c=curr
        if grid[r][c]==0:
            grid[r][c]=5
        obst()
        make_grid(grid,"DFS searching")
        for dr,dc in reversed(DIRECTIONS):
            nr,nc=r+dr,c+dc
            if walls(nr,nc) and (nr,nc) not in visited:
                stack.append((nr,nc))
                parent[(nr,nc)]=curr
    return None


#UCS(uniform cost search)
def UCS():
    pq=[]
    heapq.heappush(pq,(0,start))
    parent={}
    cost={}
    cost[start]=0

    while pq:
        curr_cost,curr=heapq.heappop(pq)
        if curr==goal:
            return parent
        r,c=curr
        if grid[r][c]==0:
            grid[r][c]=5
        obst()
        make_grid(grid,"UCS searching")
        for dr,dc in DIRECTIONS:
```

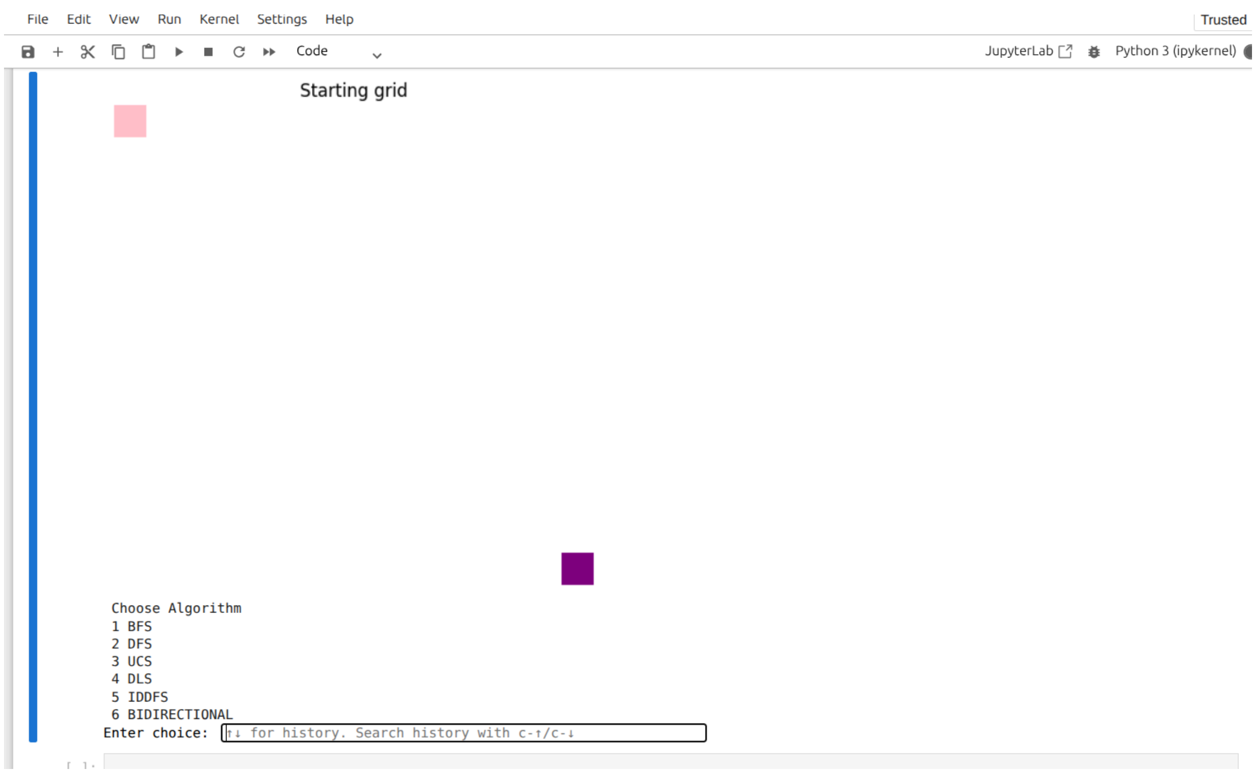

jupyter 24F-0040_Ass1_Q7 Last Checkpoint: 3 hours ago
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipyk

```
            nr,nc=r+dr,c+dc
            if walls(nr,nc):
                stack.append(((nr,nc),depth+1))
                parent[(nr,nc)]=curr
    return None

#IDDFS
def IDDFS():
    for depth in range(20):
        temp=grid.copy()
        result=DLS(depth)
        if result:
            return result
    return None

#BDS

def Bidirectional():
    q1=deque([start])
    q2=deque([goal])
    p1={}
    p2={}
    v1={start}
    v2={goal}
    while q1 and q2:
        n1=q1.popleft()
        for dr,dc in DIRECTIONS:
            nxt=(n1[0]+dr,n1[1]+dc)
            if walls(nxt[0],nxt[1]) and nxt not in v1:
                v1.add(nxt)
                p1[nxt]=n1
                q1.append(nxt)
                if nxt in v2:
                    p1.update(p2)
                    return p1
        n2=q2.popleft()
        for dr,dc in DIRECTIONS:
            nxt=(n2[0]+dr,n2[1]+dc)
            if walls(nxt[0],nxt[1]) and nxt not in v2:
```

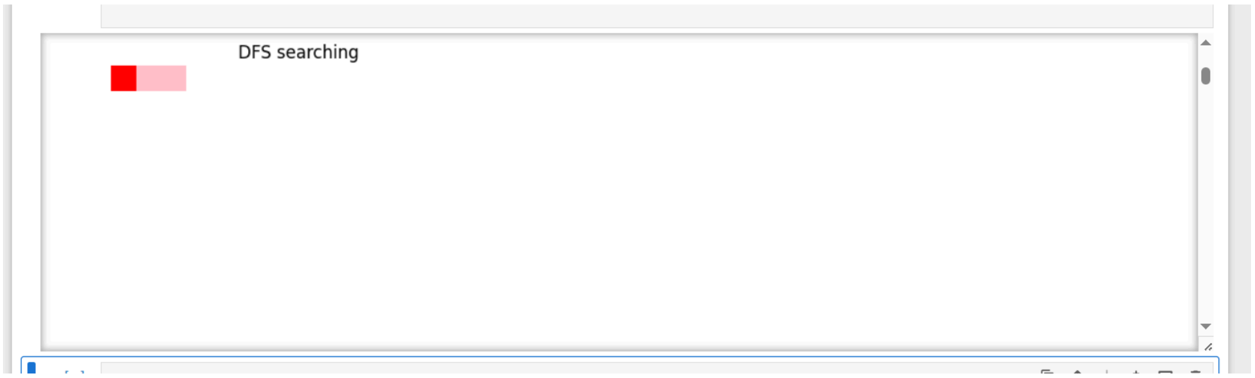

jupyter 24F-0040_Ass1_Q7 Last Checkpoint: 3 hours ago
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipyk

```
    (1, 1),    # Bottom-Right
    (0, -1),   # Left
    (-1, -1),  # Top-Left
]
obs=0.05
plt.figure(figsize=(6,6))
make_grid(grid,"Starting grid")
print("Choose Algorithm")
print("1 BFS")
print("2 DFS")
print("3 UCS")
print("4 DLS")
print("5 IDDFS")
print("6 BIDIRECTIONAL")
choice=input("Enter choice: ")
if choice=="1":
    parent=BFS()
elif choice=="2":
    parent=DFS()
elif choice=="3":
    parent=UCS()
elif choice=="4":
    parent=DLS(20)
elif choice=="5":
    parent=IDDFS()
elif choice=="6":
    parent=Bidirectional()
else:
    parent=None
if parent:
    make_path(parent)
else:
    print("No path found")
plt.show()
```

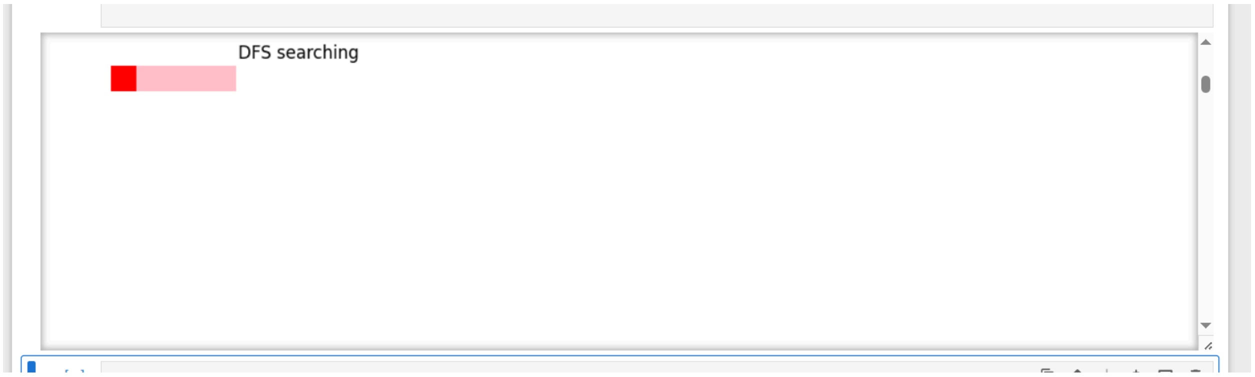
Starting grid

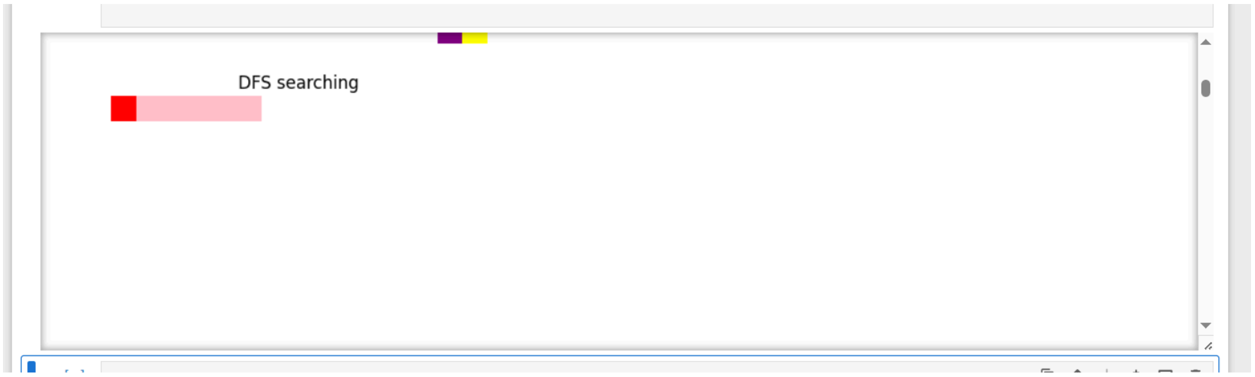


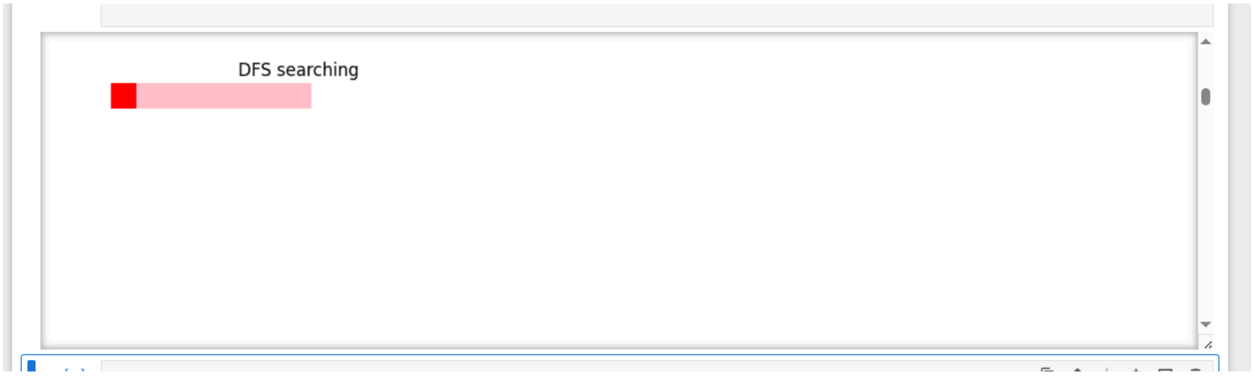
```
Choose Algorithm
1 BFS
2 DFS
3 UCS
4 DLS
5 IDDFS
6 BIDIRECTIONAL
Enter choice: ↑↓ for history. Search history with c-↑/c-↓
```

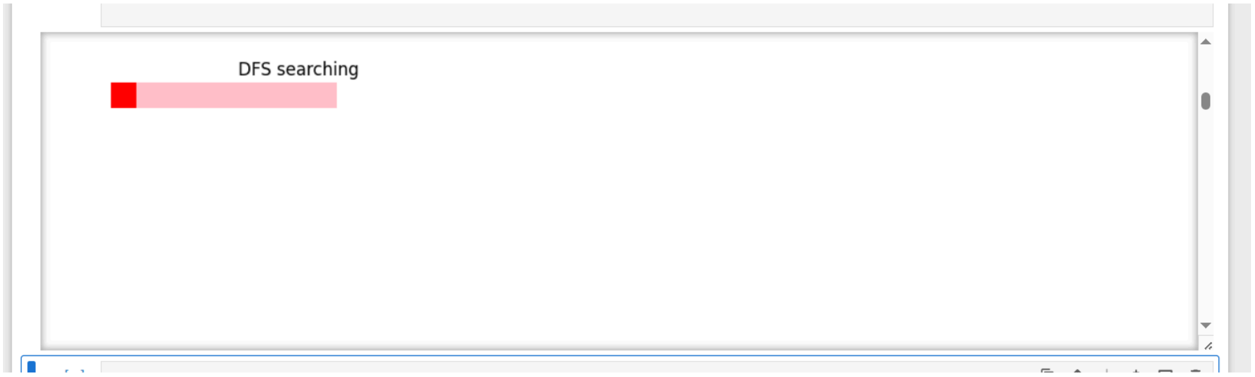

DFS searching

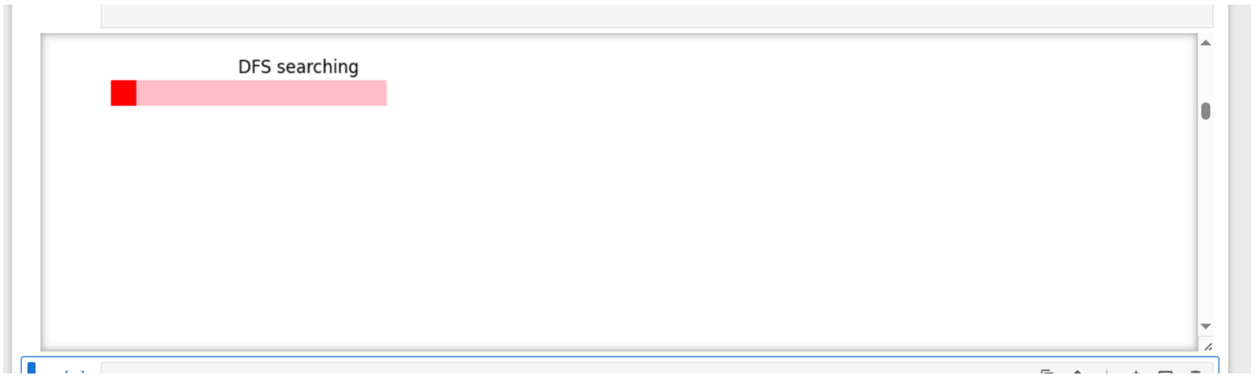
DFS searching

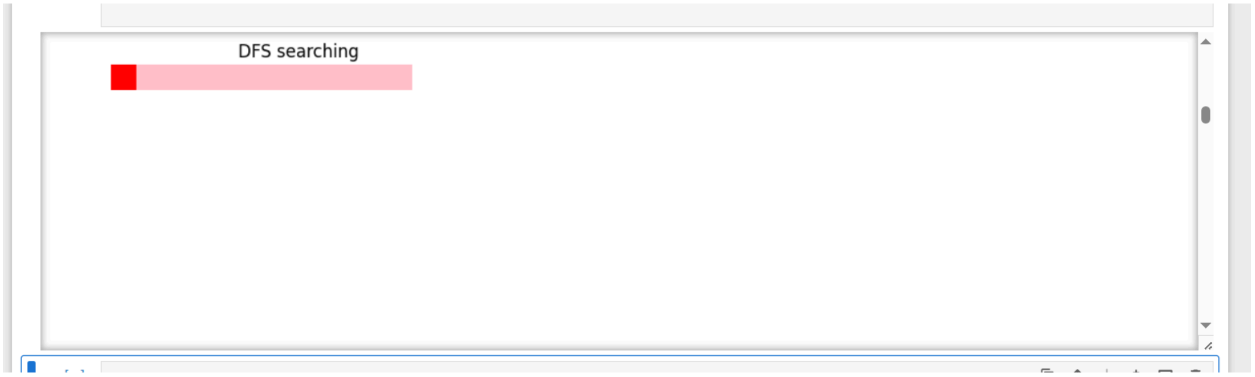

DFS searching

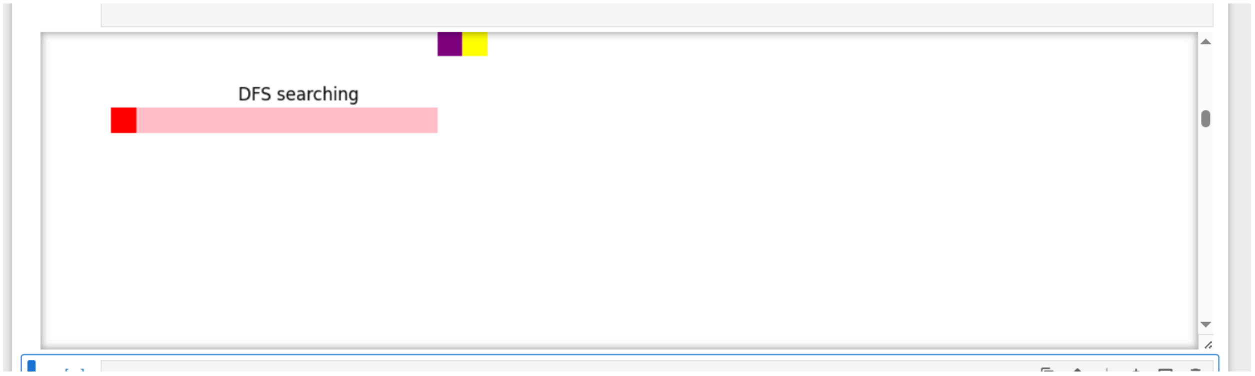

DFS searching

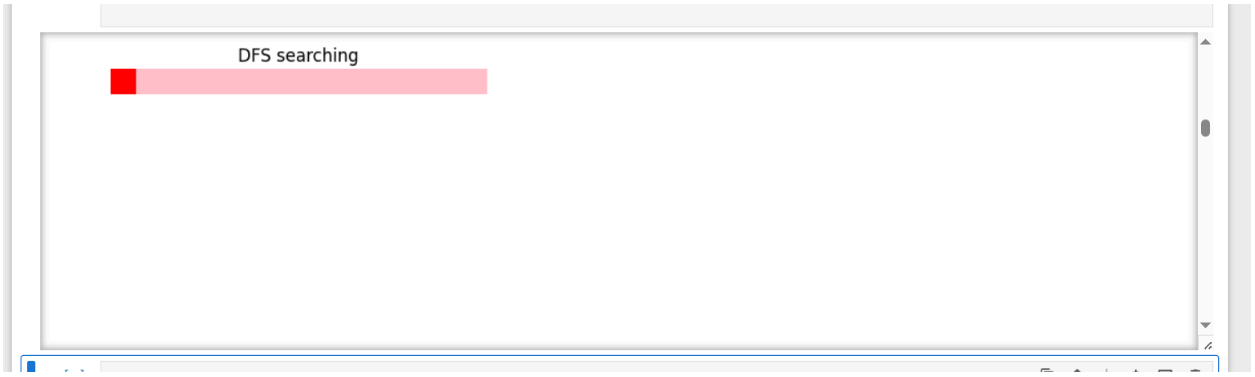

DFS searching

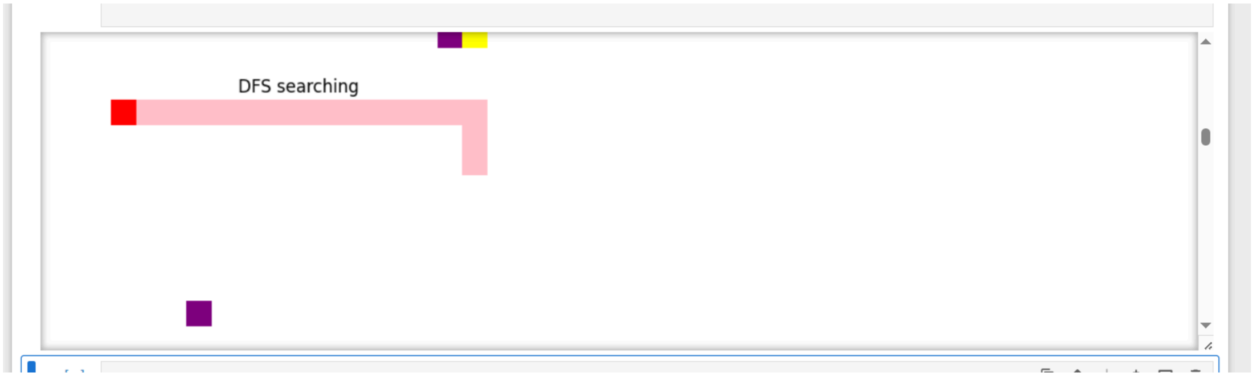

DFS searching

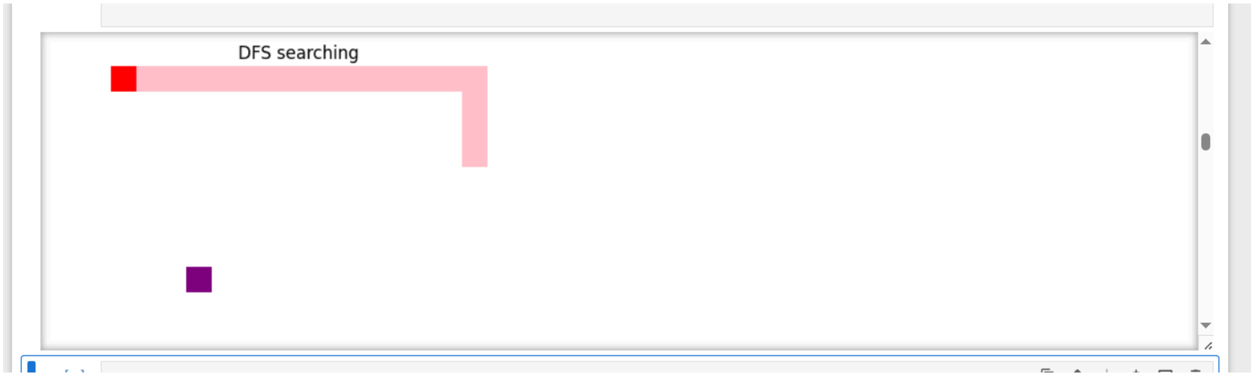

DFS searching

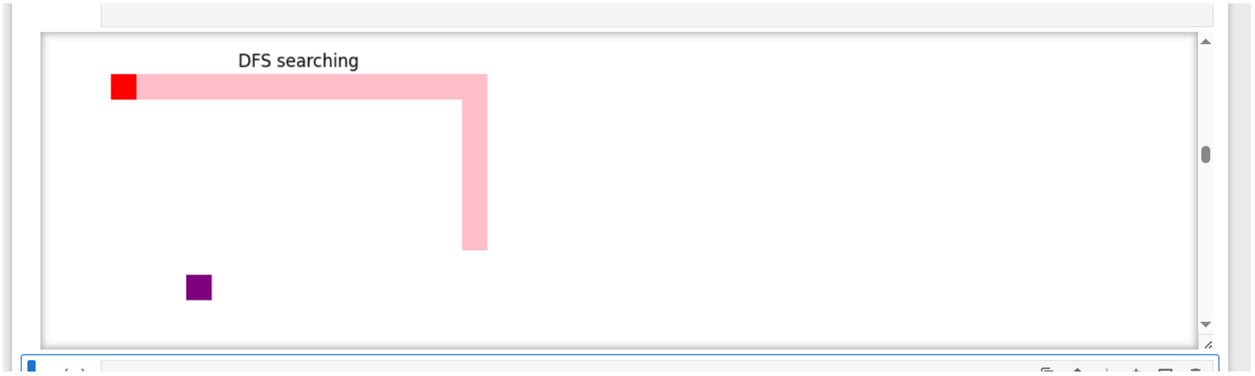
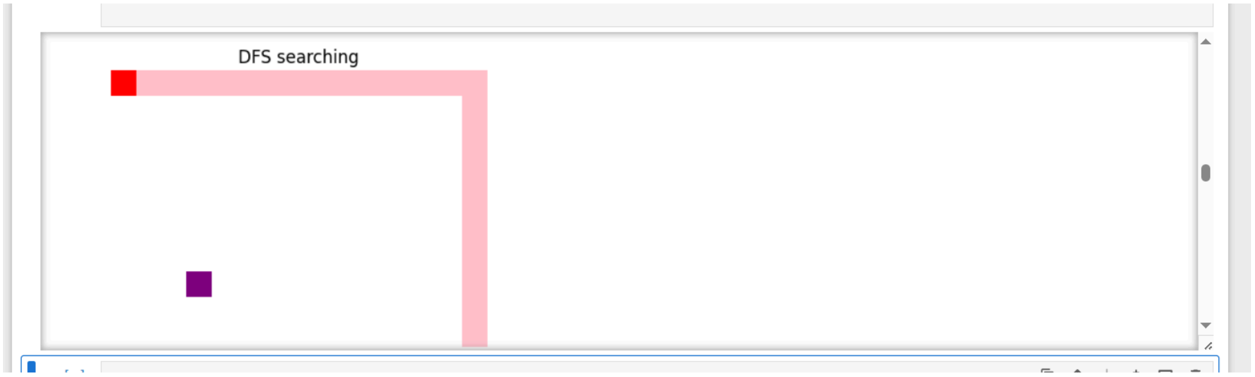
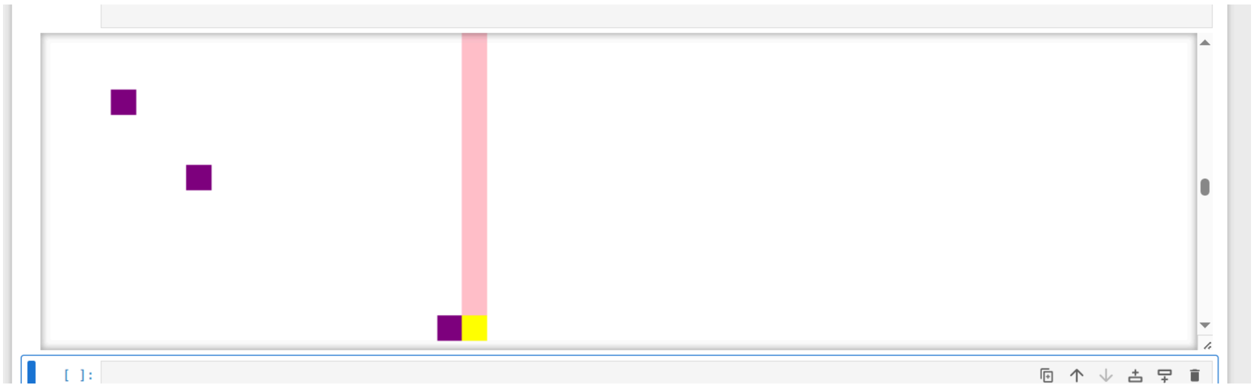
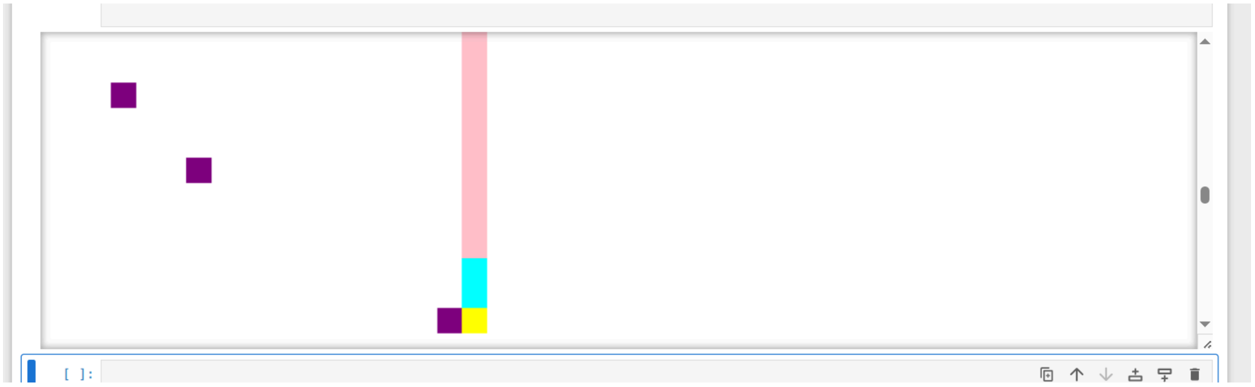

DFS searching

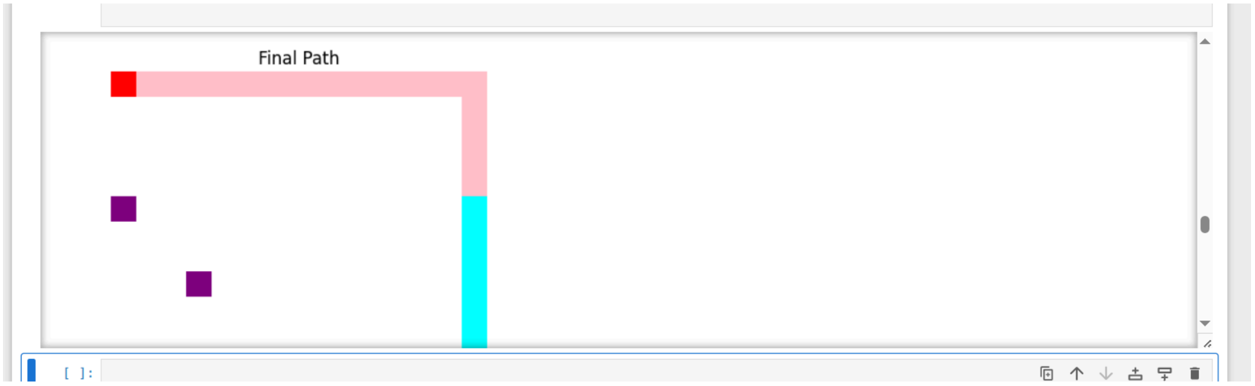

DFS searching

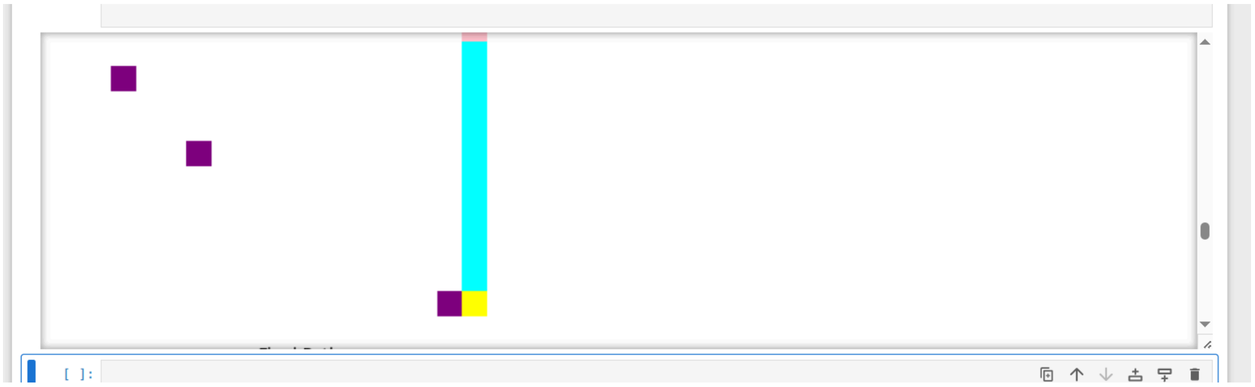

DFS searching

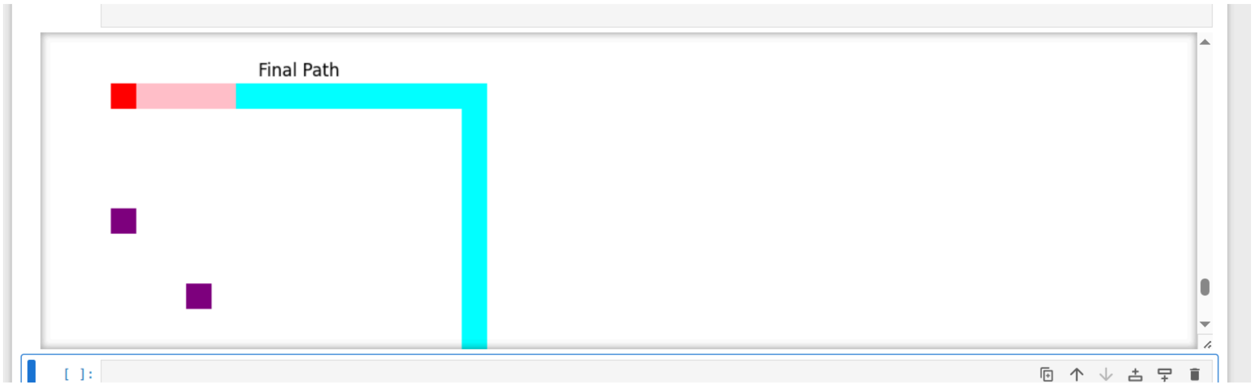

DFS searching

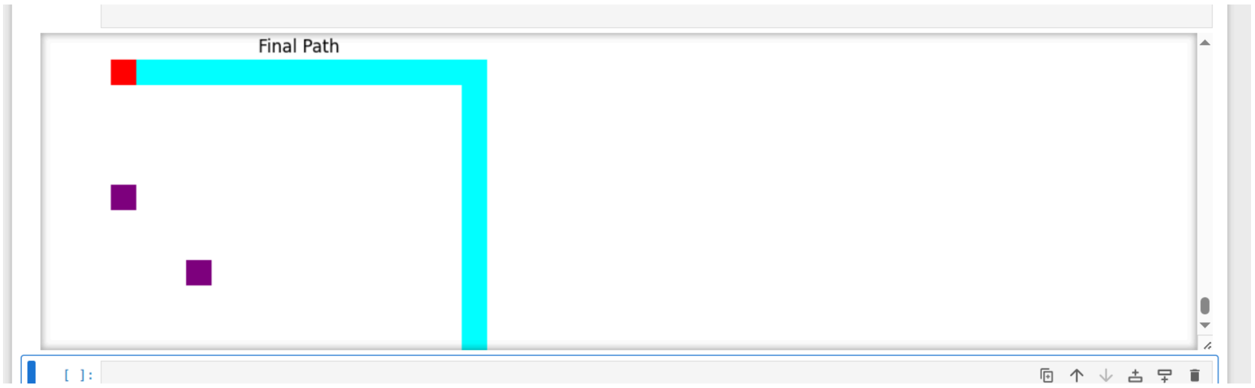

DFS searching

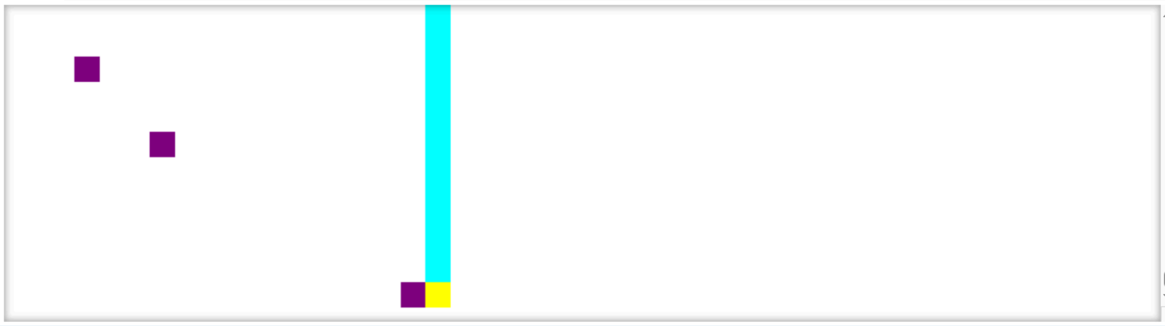

DFS searching

DFS searching

[ ]:

[ ]:


Final Path
[ ]:


[ ]:


Final Path
[ ]:


Final Path
[ ]:



```
[ ]:
```