Aim

To compare the performance and efficiency of different pathfinding algorithms in maze navigation.

Objective

To implement and evaluate Depth-First Search (DFS), Breadth-First Search (BFS), and A* algorithms, and analyze their performance through various metrics.

Summary

This project involves implementing and comparing three fundamental pathfinding algorithms—DFS, BFS, and A*—in a maze environment using the PyAmaze library. The project includes visualizing the pathfinding process, comparing the efficiency of these algorithms in terms of path length and search time, and evaluating the impact of different heuristics used in A* algorithm.

Tools and Libraries Used

- **Python**
- **PyAmaze** (for maze generation and visualization)
- **PriorityQueue** (from queue module for A* algorithm)
- **timeit** (for measuring execution time)

Procedure (Explanation of the Code)

1. **DFS Demo (DFSDemo.py)**:
    - Implements the Depth-First Search algorithm to explore the maze.
    - Maintains a stack (frontier) and a list of explored cells.
    - Uses the direction set ('E', 'W', 'N', 'S') to determine the next cell to visit.
    - Records the path taken and traces it on the maze.

CODE:

```
from pyamaze import maze,agent,textLabel,COLOR


def DFS(m,start=None):
    if start is None:
        start=(m.rows,m.cols)
    explored=[start]
    frontier=[start]
    dfsPath={}
```

```
dSeacrh=[]
while len(frontier)>0:
    currCell=frontier.pop()
    dSeacrh.append(currCell)
    if currCell==m._goal:
        break
    poss=0
    for d in 'ESNW':
        if m.maze_map[currCell][d]==True:
            if d =='E':
                child=(currCell[0],currCell[1]+1)
            if d =='W':
                child=(currCell[0],currCell[1]-1)
            if d =='N':
                child=(currCell[0]-1,currCell[1])
            if d =='S':
                child=(currCell[0]+1,currCell[1])
            if child in explored:
                continue
            poss+=1
            explored.append(child)
            frontier.append(child)
            dfsPath[child]=currCell
    if poss>1:
        m.markCells.append(currCell)
fwdPath={}
cell=m._goal
while cell!=start:
    fwdPath[dfsPath[cell]]=cell
```

```
        cell=dfsPath[cell]
    return dSeacrh,dfsPath,fwdPath


if __name__=='__main__':
    m=maze(10,10) # Change to any size
    m.CreateMaze(2,4) # (2,4) is Goal Cell, Change that to any other valid cell

    dSeacrh,dfsPath,fwdPath=DFS(m,(5,1)) # (5,1) is Start Cell, Change that to any other valid cell


    a=agent(m,5,1,goal=(2,4),footprints=True,shape='square',color=COLOR.green)
    b=agent(m,2,4,goal=(5,1),footprints=True,filled=True)
    c=agent(m,5,1,footprints=True,color=COLOR.yellow)
    m.tracePath({a:dSeacrh},showMarked=True)
    m.tracePath({b:dfsPath})
    m.tracePath({c:fwdPath})
    m.run()
```

2. **BFS Demo (BFSDemo.py)**:
    - Implements the Breadth-First Search algorithm for maze exploration.
    - Uses a queue to explore the maze level by level.
    - Records the path taken and visualizes it.

CODE:

```
from pyamaze import maze,agent,textLabel,COLOR


    m=maze(12,10)
    # m.CreateMaze(5,4,loopPercent=100)
    m.CreateMaze(loopPercent=10,theme='light')
    bSearch,bfsPath,fwdPath=BFS(m)
    a=agent(m,footprints=True,color=COLOR.yellow,shape='square',filled=True)
    b=agent(m,footprints=True,color=COLOR.red,shape='square',filled=False)
    # c=agent(m,5,4,footprints=True,color=COLOR.cyan,shape='square',filled=True,goal=(m.rows,m.cols))
    c=agent(m,1,1,footprints=True,color=COLOR.cyan,shape='square',filled=True,goal=(m.rows,m.cols))
    m.tracePath({a:bSearch},delay=100)
    m.tracePath({c:bfsPath},delay=100)
    m.tracePath({b:fwdPath},delay=100)

    m.run()
```

3. ***A Demo (aStarDemo.py)**:**
    - Implements the A* algorithm with the Manhattan distance heuristic.
    - Utilizes a priority queue to select the next cell based on the lowest estimated cost.
    - Records and visualizes the search path and final path.

CODE:

```
from pyamaze import maze,agent,COLOR,textLabel
from queue import PriorityQueue
def h(cell1, cell2):
    x1, y1 = cell1
    x2, y2 = cell2
    return (abs(x1 - x2) + abs(y1 - y2))

def aStar(m,start=None):
    if start is None:
        start=(m.rows,m.cols)
    open = PriorityQueue()
    open.put((h(start, m._goal), h(start, m._goal), start))
    aPath = {}
    g_score = {row: float("inf") for row in m.grid}
    g_score[start] = 0
    f_score = {row: float("inf") for row in m.grid}
    f_score[start] = h(start, m._goal)
```

```
searchPath=[start]
while not open.empty():
   currCell = open.get()[2]
   searchPath.append(currCell)
   if currCell == m._goal:
      break
   for d in 'ESNW':
      if m.maze_map[currCell][d]==True:
         if d=='E':
            childCell=(currCell[0],currCell[1]+1)
         elif d=='W':
            childCell=(currCell[0],currCell[1]-1)
         elif d=='N':
            childCell=(currCell[0]-1,currCell[1])
         elif d=='S':
            childCell=(currCell[0]+1,currCell[1])

         temp_g_score = g_score[currCell] + 1
         temp_f_score = temp_g_score + h(childCell, m._goal)

         if temp_f_score < f_score[childCell]:
            aPath[childCell] = currCell
            g_score[childCell] = temp_g_score
            f_score[childCell] = temp_g_score + h(childCell, m._goal)
            open.put((f_score[childCell], h(childCell, m._goal), childCell))


fwdPath={}
cell=m._goal
```

```
    while cell!=start:
        fwdPath[aPath[cell]]=cell
        cell=aPath[cell]
    return searchPath,aPath,fwdPath

if __name__=='__main__':
    m=maze(4,4)
    m.CreateMaze(loadMaze='aStardemo.csv')

    searchPath,aPath,fwdPath=aStar(m)
    a=agent(m,footprints=True,color=COLOR.blue,filled=True)
    b=agent(m,1,1,footprints=True,color=COLOR.yellow,filled=True,goal=(m.rows,m.cols))
    c=agent(m,footprints=True,color=COLOR.red)

    m.tracePath({a:searchPath},delay=300)
    m.tracePath({b:aPath},delay=300)
    m.tracePath({c:fwdPath},delay=300)

    l=textLabel(m,'A Star Path Length',len(fwdPath)+1)
    l=textLabel(m,'A Star Search Length',len(searchPath))
    m.run()
```

4. ***A* with Euclidean Heuristic (aStarDemo2.py)**:**
    - Implements the A* algorithm using Euclidean distance heuristic.
    - Compares the performance with the Manhattan heuristic.

CODE:

```
from pyamaze import maze,agent,COLOR,textLabel
from queue import PriorityQueue
from math import sqrt
def h(cell1, cell2):
    x1, y1 = cell1
    x2, y2 = cell2
    # return sqrt((x1-x2)**2+(y1-y2)**2)
    return ((x1-x2)**2+(y1-y2)**2)

def aStar2(m,start=None):
    if start is None:
        start=(m.rows,m.cols)
    open = PriorityQueue()
    open.put((h(start, m._goal), h(start, m._goal), start))
    aPath = {}
    g_score = {row: float("inf") for row in m.grid}
    g_score[start] = 0
    f_score = {row: float("inf") for row in m.grid}
    f_score[start] = h(start, m._goal)
    searchPath=[start]
    while not open.empty():
        currCell = open.get()[2]
        searchPath.append(currCell)
        if currCell == m._goal:
            break
        for d in 'ESNW':
            if m.maze_map[currCell][d]==True:
                if d=='E':
                    childCell=(currCell[0],currCell[1]+1)
                elif d=='W':
                    childCell=(currCell[0],currCell[1]-1)
                elif d=='N':
                    childCell=(currCell[0]-1,currCell[1])
                elif d=='S':
                    childCell=(currCell[0]+1,currCell[1])

                temp_g_score = g_score[currCell] + 1
                temp_f_score = temp_g_score + h(childCell, m._goal)

                if temp_f_score < f_score[childCell]:
                    aPath[childCell] = currCell
                    g_score[childCell] = temp_g_score
                    f_score[childCell] = temp_g_score + h(childCell, m._goal)
                    open.put((f_score[childCell], h(childCell, m._goal), childCell))
    fwdPath={}
    cell=m._goal
    while cell!=start:
        fwdPath[aPath[cell]]=cell
        cell=aPath[cell]
    return searchPath,aPath,fwdPath

if __name__=='__main__':
    m=maze(4,4)
    m.CreateMaze(loadMaze='aStardemo.csv')

    searchPath,aPath,fwdPath=aStar2(m)
    a=agent(m,footprints=True,color=COLOR.blue,filled=True)
    b=agent(m,1,1,footprints=True,color=COLOR.yellow,filled=True,goal=(m.rows,m.cols))
    c=agent(m,footprints=True,color=COLOR.red)

    m.tracePath({a:searchPath},delay=300)
    m.tracePath({b:aPath},delay=300)
    m.tracePath({c:fwdPath},delay=300)


    l=textLabel(m,'A Star Path Length',len(fwdPath)+1)
    l=textLabel(m,'A Star Search Length',len(searchPath))
    m.run()
```

5. **DFS vs BFS (DFSvsBFS.py)**:
    - Compares the performance of DFS and BFS algorithms.
    - Measures and displays the path lengths and search times for both algorithms.

CODE:

```
from BFSDemo import BFS
from DFSDemo import DFS
from pyamaze import maze,agent,COLOR,textLabel
from timeit import timeit


m=maze(20,30)
# m.CreateMaze(loopPercent=100)
m.CreateMaze(1,30,loopPercent=100)
# m.CreateMaze()
# m.CreateMaze(1,30)
searchPath,dfsPath,fwdDFSPath=DFS(m)
bSearch,bfsPath,fwdBFSPath=BFS(m)


textLabel(m,'DFS Path Length',len(fwdDFSPath)+1)
textLabel(m,'BFS Path Length',len(fwdBFSPath)+1)
textLabel(m,'DFS Search Length',len(searchPath)+1)
textLabel(m,'BFS Search Length',len(bSearch)+1)


a=agent(m,footprints=True,color=COLOR.cyan,filled=True)
b=agent(m,footprints=True,color=COLOR.yellow)
```

```
m.tracePath({a:fwdBFSPath},delay=100)

m.tracePath({b:fwdDFSPath},delay=100)



t1=timeit(stmt='DFS(m)',number=1000,globals=globals())

t2=timeit(stmt='BFS(m)',number=1000,globals=globals())


textLabel(m,'DFS Time',t1)

textLabel(m,'BFS Time',t2)



m.run()
```

6. ***BFS vs A* (BFSvsAStar.py)**:**
    - Compares BFS and A* algorithms in terms of path length and search time.
    - Displays and compares the results for both algorithms.

CODE:

```
from BFSDemo import BFS

from aStarDemo import aStar

from pyamaze import maze,agent,COLOR,textLabel

from timeit import timeit

myMaze=maze(50,70)

myMaze.CreateMaze(loopPercent=100)

# myMaze.CreateMaze()

searchPath,aPath,fwdPath=aStar(myMaze)

bSearch,bfsPath,fwdBFSPath=BFS(myMaze)


l=textLabel(myMaze,'A-Star Path Length',len(fwdPath)+1)

l=textLabel(myMaze,'BFS Path Length',len(fwdBFSPath)+1)

l=textLabel(myMaze,'A-Star Search Length',len(searchPath)+1)
```

```
l=textLabel(myMaze,'BFS Search Length',len(bSearch)+1)


a=agent(myMaze,footprints=True,color=COLOR.cyan,filled=True)

b=agent(myMaze,footprints=True,color=COLOR.yellow)

myMaze.tracePath({a:fwdBFSPath},delay=50)

myMaze.tracePath({b:fwdPath},delay=50)


t1=timeit(stmt='aStar(myMaze)',number=10,globals=globals())

t2=timeit(stmt='BFS(myMaze)',number=10,globals=globals())


textLabel(myMaze,'A-Star Time',t1)

textLabel(myMaze,'BFS Time',t2)



myMaze.run()
```

7. ***A* Heuristic Comparison (aStarHeuristicComparison.py)**:**
   - Compares the effectiveness of Manhattan and Euclidean heuristics in A* algorithm.
   - Evaluates which heuristic provides better performance in terms of path length and search time.

CODE:

```
from aStarDemo import aStar

from aStarDemo2 import aStar2

from pyamaze import maze,agent,COLOR,textLabel

from timeit import timeit

f1,f2,f3=0,0,0

s1,s2,s3=0,0,0


for _ in range(100):

   myMaze=maze(20,30)
```

```
    myMaze.CreateMaze(loopPercent=100)
    searchPath,aPath,fwdPath=aStar(myMaze)
    searchPath2,aPath2,fwdPath2=aStar2(myMaze)

    if len(fwdPath)==len(fwdPath2):
        f1+=1
    elif len(fwdPath)<len(fwdPath2):
        f2+=1
    else:
        f3+=1

    if len(searchPath)==len(searchPath2):
        s1+=1
    elif len(searchPath)<len(searchPath2):
        s2+=1
    else:
        s3+=1

print('Final Path Comparison Result')
print(f'Both have same Final Path length for {f1} times.')
print(f'Manhattan has lesser Final Path length for {f2} times.')
print(f'Euclidean has lesser Final Path length for {f3} times.')

print('---------------------------------------------')

print('Search Path Comparison Result')
print(f'Both have same Search Path length for {s1} times.')
print(f'Manhattan has lesser Search Path length for {s2} times.')
print(f'Euclidean has lesser Search Path length for {s3} times.')
```

Highlights

- **DFS**: Provides a pathfinding solution that explores deeply, which can be inefficient for large mazes.
- **BFS**: Ensures finding the shortest path by exploring all possible paths level by level.
- **A***: Utilizes heuristics (Manhattan and Euclidean) to improve efficiency in finding the shortest path.
- **Heuristic Comparison**: Compares the effectiveness of different heuristics in the A* algorithm.

Conclusion

The project successfully demonstrates the comparative analysis of DFS, BFS, and A* algorithms in maze navigation. BFS and A* (with appropriate heuristics) generally perform better than DFS in finding the shortest path. The choice of heuristic in A* significantly impacts its performance, with the Manhattan distance heuristic often providing faster results compared to the Euclidean heuristic.