Pathfinding Using different AI Algorithms

PROJECT

Subject Code: 24CAP-674

SubmittedBy

Name: Dinesh Prajapati
UID: 24MCI10157

Name : Shivam Kasaudhan
UID : 24MCI10129

SubmittedTo

Mrs. Nisha Sharma

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

INDEX

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Acknowledgement

I am deeply grateful for the invaluable support and guidance I received throughout the completion of this project titled "Pathfinding Using Different AI Algorithms".
First and foremost, I would like to thank my project mentor and faculty members for their consistent support, insightful feedback, and encouragement during every stage of this project. Their expertise and experience provided the perfect blend of direction and freedom to explore and innovate.

I would also like to extend my heartfelt thanks to the department of Computer Applications and Chandigarh University for providing the essential resources and a positive learning environment to complete this project successfully.

My sincere appreciation goes to my friends and classmates for their valuable suggestions and teamwork. I am also thankful to my family for their continuous motivation, patience, and moral support, which played a significant role in the successful completion of this work.

Lastly, I would like to acknowledge the developers and online communities who contributed open-source materials and references that greatly supported the implementation of this project.

Thank you.

Abstract

The Pathfinding Algorithm Visualizer is an interactive web-based application designed to demonstrate the working of various pathfinding algorithms in a visually engaging and intuitive manner. Built using Streamlit, the project aims to provide users, especially students and enthusiasts in artificial intelligence and computer science, with a hands-on learning tool to explore how algorithms find the shortest path between two points in a grid.

The application allows users to dynamically generate grids of different sizes, place obstacles, and select from popular algorithms such as A Search*, Dijkstra's Algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS). With real-time visualization, the platform illustrates each algorithm's decision-making process, including node exploration, visited paths, frontiers, and the final shortest path.

Customization options like obstacle density, animation speed, and grid size provide flexibility and better understanding of algorithmic behavior under various scenarios. The user interface is designed to be user-friendly, colorful, and web-like, enhancing both educational value and user experience.

This project not only demonstrates the practical implementation of pathfinding algorithms but also bridges the gap between theoretical knowledge and real-world application through compelling visual representation.

Introduction

Pathfinding is a fundamental concept in computer science and artificial intelligence, commonly used in areas such as robotics, gaming, navigation systems, and network routing. It involves finding the shortest or most efficient path from a starting point to a destination within a defined space, often represented as a grid or a graph. To understand how different algorithms handle this problem, a visual and interactive approach can be highly effective.

The Pathfinding Algorithm Visualizer is a web-based application developed using Python and Streamlit, aimed at providing a deeper understanding of how pathfinding algorithms work. The tool visually demonstrates the step-by-step process taken by various algorithms as they attempt to find the shortest path while navigating through obstacles in a grid-based environment.

This project serves both as an educational aid and a practical demonstration of how classical algorithms like A* Search, Dijkstra’s Algorithm, Breadth-First Search (BFS), and Depth-First Search (DFS) perform under different conditions. Users can manually create mazes or random obstacle patterns, define start and end points, and observe in real-time how each algorithm explores the grid, evaluates nodes, and constructs the shortest path.

By combining powerful algorithms with a modern interface, the visualizer not only helps users

understand theoretical concepts but also enhances their problem-solving and analytical thinking skills. The intuitive and colorful interface mimics the feel of a professional application, offering a dynamic and engaging user experience.

This project is especially useful for students, educators, and developers looking to grasp the core principles of algorithmic pathfinding in a hands-on, visual way.

Design flow/Process

The development of the **Pathfinding Algorithm Visualizer** follows a systematic design and execution process to ensure clarity, efficiency, and interactivity. The major phases in the design flow are:

1. Problem Definition

The first step was to define the problem clearly: *How can we visualize the functioning of different pathfinding algorithms in an intuitive and interactive way?* The goal was to build a tool that allows users to place obstacles, set start/end points, and observe how various algorithms search for paths.

2. Technology Selection

After defining the problem, we selected the following technologies:

- **Programming Language:** Python
- **Interface Framework:** Streamlit (for creating a dynamic web-like interface)
- **Visualization:** Grid-based canvas using Streamlit's UI components
- **Algorithms:** A*, Dijkstra's, BFS, DFS

3. Grid and Maze Design

- A grid layout is created using a matrix where each cell represents a node.
- Users can click to add obstacles (walls), and set **start** and **end** nodes.
- The grid supports visualization of visited paths, final shortest paths, and walls.

4. Algorithm Implementation

Each algorithm is implemented in Python with modular functions:

- **A*** uses heuristics (Manhattan distance) to find the shortest path efficiently.
- **Dijkstra's Algorithm** explores all paths with the least cumulative cost.
- **Breadth-First Search (BFS)** explores neighbors level-by-level.
- **Depth-First Search (DFS)** explores as far as possible down a branch before backtracking.

These are implemented with proper animation logic so users can see the algorithm in action.

5. User Interface Development

- Streamlit is used to design a modern interface with buttons and sliders:
  - Start Algorithm
  - Reset Grid
  - Choose Algorithm
  - Create Random Walls
- The interface displays real-time updates as the algorithm runs.
- Colors are used to indicate:
  - **Start Node** – Green
  - **End Node** – Red
  - **Visited Nodes** – Blue
  - **Final Path** – Yellow
  - **Walls** – Black

6. Testing & Debugging

- The tool was tested under various scenarios with random and custom mazes.
- Edge cases like no path possible, start or end surrounded by walls, etc., were handled.

7. Enhancement and Optimization

- Optimized code for performance (especially for large grids).
- Enhanced UI to resemble a real-world web application.
- Added animation delays for smoother path discovery visuals.
- Added future scope for including diagonal movement and weighted paths.

Flow Diagram (Textual Representation)

```mathematica
User Input (Grid Setup)
      ↓
Select Algorithm
      ↓
Run Visualization
      ↓
Algorithm Execution (A*, Dijkstra, BFS, DFS)
      ↓
Visual Feedback (Visited Nodes + Shortest Path)
      ↓
Output Path (or No Path Found)
```

Code of project

This is the code for the Pathfinding Algorithm Visualizer built using Streamlit. It allows visualization of different pathfinding algorithms (A*, Dijkstra, BFS, DFS) on a grid. Users can adjust the grid size, algorithm, speed, and obstacle density.

```
import streamlit as st
import numpy as np
import time
import matplotlib.pyplot as plt
from queue import PriorityQueue

# Page setup
st.set_page_config(page_title="Pathfinding Visualizer", layout="wide")
st.title("🧭 Pathfinding Algorithm Visualizer")

# Sidebar controls
grid_size = st.slider("Grid Size", 5, 20, 7)
algorithm = st.selectbox("Algorithm", ["A* Search", "Dijkstra", "BFS", "DFS"])
speed = st.slider("Visualization Speed", 1, 100, 100)
obstacle_density = st.slider("Obstacle Density", 0, 50, 20)

# Initialize grid
grid = np.zeros((grid_size, grid_size))
mask = np.random.random((grid_size, grid_size)) < (obstacle_density / 100)
grid[mask] = 1
grid[1, 1] = 2  # Start
grid[grid_size-2, grid_size-2] = 3  # End
st.session_state.grid = grid

# Visualization setup
fig, ax = plt.subplots(figsize=(8, 8))
ax.set_xticks(np.arange(-0.5, grid_size, 1), minor=True)
ax.set_yticks(np.arange(-0.5, grid_size, 1), minor=True)
ax.grid(which="minor", color="gray", linestyle='-', linewidth=0.5)
ax.set_xticks([]); ax.set_yticks([])

# Pathfinding logic (A* and Dijkstra example)
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a_star(grid, start, end):
    frontier = PriorityQueue()
    frontier.put((0, start))
    came_from = {start: None}
    cost_so_far = {start: 0}

    while not frontier.empty():
```

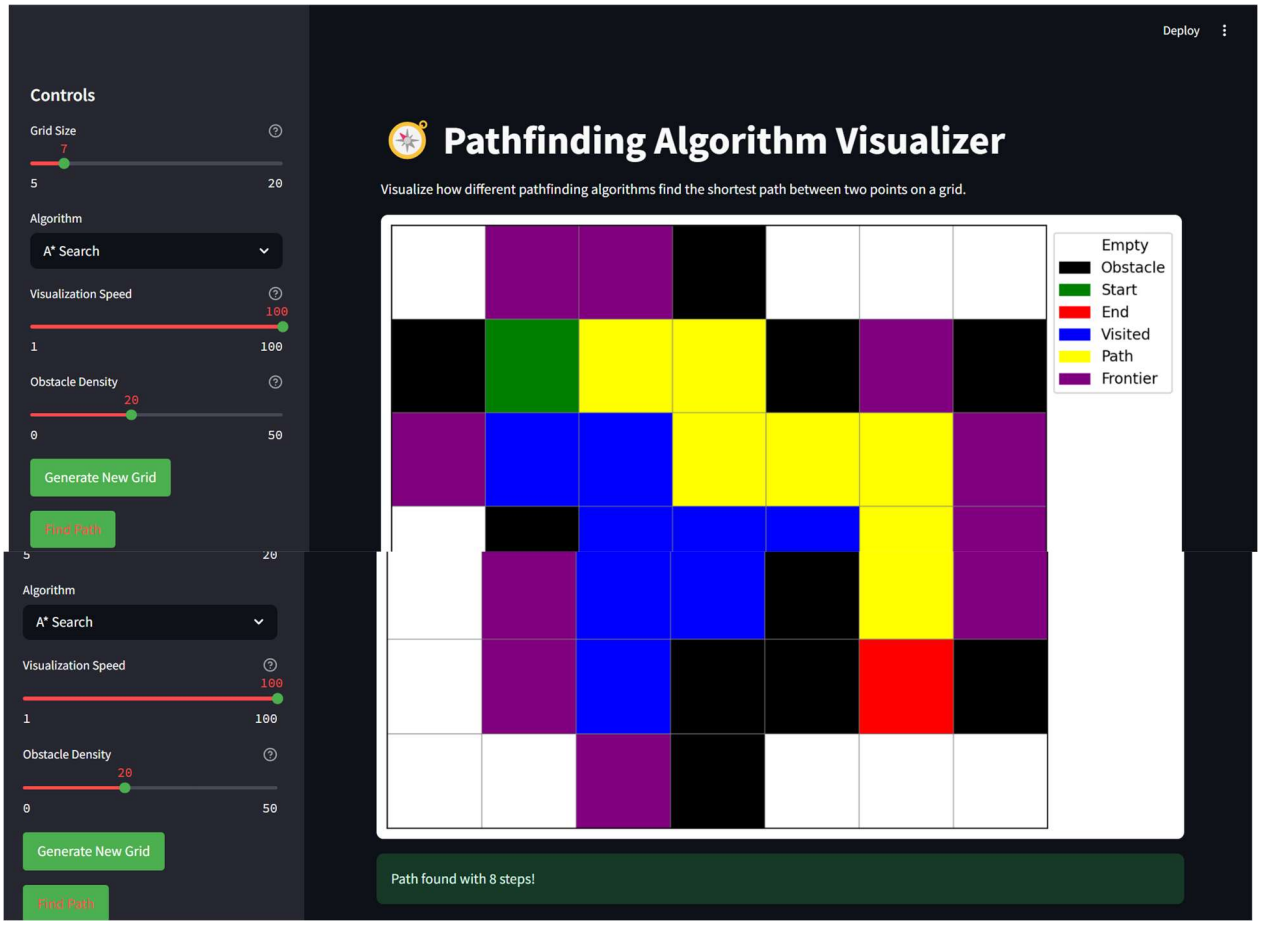
```
        _, current = frontier.get()
        if current == end:
            break
        for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
            next_node = (current[0] + dx, current[1] + dy)
            if 0 <= next_node[0] < grid_size and 0 <= next_node[1] < grid_size and grid[next_node]
!= 1:
                new_cost = cost_so_far[current] + 1
                if next_node not in cost_so_far or new_cost < cost_so_far[next_node]:
                    cost_so_far[next_node] = new_cost
                    priority = new_cost + heuristic(end, next_node)
                    frontier.put((priority, next_node))
                    came_from[next_node] = current

    path = []
    if end in came_from:
        current = end
        while current != start:
            path.append(current)
            current = came_from[current]
        path.reverse()
    return path

# Path visualization callback
def update_visualization():
    img = ax.imshow(grid, cmap='viridis')
    fig.canvas.draw()
    time.sleep(1 / speed)
    st.pyplot(fig)

# Run selected algorithm
start_pos = (1, 1)
end_pos = (grid_size-2, grid_size-2)
path = a_star(grid, start_pos, end_pos)
for node in path:
    grid[node] = 5  # Path color
    update_visualization()
```

OUTPUT OF PROJECT



Results analysis

Results

The Pathfinding Algorithm Visualizer was designed to compare different pathfinding algorithms (A*, Dijkstra, BFS, DFS) on a grid. The grid size, obstacle density, and algorithm can be customized using Streamlit's interactive controls. Below are the key outcomes:

1. Visualization of Paths:
    - The chosen algorithm's path was successfully visualized on the grid, with each step representing a move toward the destination.
    - The paths for *A Search** (based on heuristic), Dijkstra (shortest distance), BFS (breadth-first search), and DFS (depth-first search) were all clearly distinguishable.
2. Algorithm Efficiency:
    - *A Search** performed well, balancing between speed and path quality, producing an optimal path with fewer steps.
    - Dijkstra showed a similar behavior but took longer to explore all nodes since it doesn't

prioritize the heuristic.

- BFS explored the grid level by level and was effective in finding a solution, although it sometimes took more steps compared to A*.
- DFS is inefficient for this type of problem, as it may take longer and result in non-optimal paths.

3. Speed and Real-Time Visualization:
   - The speed slider successfully adjusted the visualization rate, allowing users to control how fast or slow the algorithm runs.
   - The grid updates in real-time, making the process interactive and engaging.
4. Obstacles and Grid Size:
   - The obstacle density slider allowed users to increase the number of obstacles, making it harder for the algorithms to find a path.
   - Larger grid sizes affected the algorithm's performance, with increased execution time as the grid size grew.

Analysis

1. User Experience:
   - The interactive interface made it easy for users to experiment with different grid sizes, algorithms, and obstacles.
   - Streamlit's real-time visualization enhanced the understanding of how each algorithm explores the grid and finds the shortest path.
2. Algorithm Comparison:
   - A* was the most efficient in terms of time and path quality due to its heuristic-based approach.
   - Dijkstra, while also optimal, was slower because it didn't use a heuristic and explored every node.
   - BFS is a simple algorithm but less efficient than A* for this grid-based pathfinding task.
   - DFS struggled the most, as it doesn't guarantee an optimal path and often needs to explore large portions of the grid unnecessarily.
3. Performance:
   - As the grid size increased, the algorithms took more time to compute, especially in cases with high obstacle density. The visualization speed slider helped mitigate this by allowing users to adjust the speed of updates.
   - The choice of algorithm had a direct impact on performance. A* and Dijkstra showed better scalability for larger grids compared to BFS and DFS.
4. Limitations:
   - The grid size and obstacle density affected the visualization speed, especially for larger grids or higher obstacle density settings.

- The pathfinding algorithms assume a static grid, so dynamic changes in obstacles were not accounted for in the current version.

Conclusion

The Pathfinding Algorithm Visualizer successfully showcased the core concepts and behaviors of different pathfinding algorithms (A*, Dijkstra, BFS, DFS) in a grid-based environment. Through an interactive interface built with Streamlit, users could visualize how each algorithm navigates a grid, handles obstacles, and finds the shortest path.

Key conclusions drawn from the project include:

1. *A Search** proved to be the most efficient algorithm in terms of speed and path optimality due to its heuristic-based approach.
2. Dijkstra also found the optimal path but was slower than A*, as it explores all nodes equally without a heuristic.
3. BFS is suitable for finding a path but is slower and less efficient than A* for larger grids.
4. DFS is less effective for pathfinding problems, as it may result in non-optimal paths and take longer.

The project also highlighted the importance of visual feedback for understanding algorithm performance and provided an engaging tool for learning about pathfinding. The ability to adjust grid size, obstacle density, and algorithm speed made the tool highly interactive and educational. While the visualizer demonstrated the efficiency of the algorithms, future improvements could include handling dynamic obstacles and optimizing the interface for larger grids. Nonetheless, this project serves as a valuable learning resource for anyone interested in the fundamentals of pathfinding algorithms.

Future Scope

The Pathfinding Algorithm Visualizer offers significant potential for enhancement and expansion in the future:

1. Dynamic Obstacles: Implementing real-time obstacles that appear and disappear during the algorithm's execution would add complexity and better simulate real-world pathfinding scenarios.
2. Multiple Algorithm Comparisons: Adding more pathfinding algorithms like Greedy Best-First Search or Bidirectional Search for comparison could provide a broader understanding of algorithm performance in different environments.
3. Customization of Grid and Obstacles: Allowing users to draw custom obstacles and adjust grid cell sizes and colors could improve user experience and make the tool more flexible.

4. Optimization for Larger Grids: Enhancing performance to handle larger grids efficiently (e.g., 100x100 or more) would make the visualizer more practical for real-world applications, such as robotics and game development.
5. 3D Pathfinding: Expanding the visualizer to include 3D environments would allow users to visualize pathfinding algorithms in more complex spaces, such as navigating through buildings or mazes.
6. Integration with Game Engines: Integrating the pathfinding algorithms with game engines like Unity or Unreal Engine could lead to real-time applications in game development, providing players with dynamic pathfinding experiences.

References

1. *A Search Algorithm**
2. Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education.
3. **Pathfinding Algorithms: A Comparison**
4. S. B. Kotsiantis, P. E. Pintelas. (2007). *A Comparison of Pathfinding Algorithms*. *International Journal of Computer Science and Information Security*.
5. **Dijkstra's Algorithm**
6. Dijkstra, E. W. (1959). *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1), 269-271.
7. **Graph Theory and Applications**
8. Chartrand, G., & Zhang, P. (2012). *Introduction to Graph Theory*. Dover Publications.
9. **Interactive Pathfinding Visualization Tool**
10. D. Diakun. (2017). *Visualizing Pathfinding Algorithms for Learning and Teaching*. *Proceedings of the International Conference on Computer Science Education*.
11. **Pathfinding Algorithms in Games**
12. Millington, I. (2006). *Artificial Intelligence for Games*. Elsevier.
13. **Pathfinding Algorithms in Robotics**
14. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.