Vivekanand Education Society's Institute Of Technology
Department Of Information Technology

DSA mini Project

A.Y. 2025-26

Title: Data Structures & Algorithms

Domain: Data Structures & Algorithms

Member:Suraj Prakash

Mentor Name: Kajal Jewani

Content

1. **Introduction to the Project**
2. **Problem Statement**
3. **Scope of the Project**
4. **Requirements of the System (Hardware, Software)**
5. **Code and Video**
6. **Data Structure & Concepts Used**
7. **Time and Space Complexity**
8. **Future Scope & Enhancements**
9. **Conclusion**
10. **References (in IEEE Format)**

Introduction to Project

- Built using **javascript, css** and **bfs**
- Built with **React** to demonstrate a practical application of the **Breadth-First Search (BFS)** algorithm
- Inspired by the **Arcade Original**
- Focuses on **DSA implementatio**

Problem Statement

- **Main Goal:** To create a Pac-Man game with **intelligent ghost AI**, not just enemies that move randomly.
- **The Challenge:** To program the ghosts to efficiently **hunt the player** by always finding the shortest possible path through the maze.
- **The Solution:** To use a classic computer science algorithm (**Breadth-First Search**) as the "brain" for the ghosts' movement.

Scope of The Project

- A complete, **single-level** Pac-Man game.
- A player-controlled Pac-Man using **keyboard inputs**.
- Three **intelligent ghosts** that use the BFS algorithm to chase the player.
- A full game loop, including scoring, lives, **win/loss panels**, and a pause feature.
- A **high score** system that saves to the browser.

Objectives of the project

- **Apply a Pathfinding Algorithm:** To implement Breadth-First Search (BFS) to create intelligent ghost AI.
- **Build a Complete Game:** To develop a full, playable Pac-Man game with all essential features.
- **Gain React Experience:** To build the project using the React.js library to manage the game's state and user interface.

Requirements of the system (Hardware, software)

- **Hardware:** Any standard PC/Laptop (2 GB RAM minimum)
- **Software:** To build or modify the project, **Node.js** and **npm** are required.
- **OS:** Windows/Linux/Mac
- **Web Browser:** An up-to-date version of a web browser like Google Chrome, Mozilla Firefox, or Microsoft Edge.

Data Structures & Concepts Used

Array for making Map

```
const layout = [
  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
  1,2,2,2,2,2,2,2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,1,
  1,2,1,1,1,1,2,1,1,1,1,1,2,1,1,2,1,1,1,1,1,2,1,1,1,1,2,1,
  1,3,1,1,1,1,2,1,1,1,1,1,2,1,1,2,1,1,1,1,1,2,1,1,1,1,3,1,
  1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,
  1,2,1,1,1,1,2,1,1,2,1,1,1,1,1,1,1,1,2,1,1,2,1,1,1,1,2,1,
  1,2,2,2,2,2,2,2,1,1,2,2,2,2,1,1,2,2,2,2,1,1,2,2,2,2,2,2,1,
  1,1,1,1,1,1,2,1,1,1,1,1,0,1,1,0,1,1,1,1,1,2,1,1,1,1,1,1,
  0,0,0,0,0,0,1,2,1,1,0,0,0,0,0,0,0,0,0,0,1,1,2,1,0,0,0,0,0,
  1,1,1,1,1,1,2,1,1,0,1,1,1,4,4,1,1,1,0,1,1,2,1,1,1,1,1,1,
  1,0,0,0,0,0,0,0,0,0,1,4,4,4,4,4,4,1,0,0,0,0,0,0,0,0,0,1,
  1,1,1,1,1,1,2,1,1,0,1,4,4,4,4,4,4,1,0,1,1,2,1,1,1,1,1,1,
  0,0,0,0,0,0,1,2,1,1,0,1,1,1,1,1,1,1,1,0,1,1,2,1,0,0,0,0,0,
  1,1,1,1,1,1,2,1,1,0,0,0,0,0,0,0,0,0,0,1,1,2,1,1,1,1,1,1,
  1,2,2,2,2,2,2,2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,2,2,2,1,
  1,2,1,1,1,1,2,1,1,1,1,1,2,1,1,2,1,1,1,1,1,2,1,1,1,1,2,1,
  1,3,2,2,1,1,2,2,2,2,2,2,2,0,0,2,2,2,2,2,2,2,1,1,2,2,3,1,
  1,1,1,2,1,1,2,1,1,2,1,1,1,1,1,1,1,1,2,1,1,2,1,1,2,1,1,1,
  1,2,2,2,2,2,2,2,1,1,2,2,2,2,1,1,2,2,2,2,1,1,2,2,2,2,2,2,1,
  1,2,1,1,1,1,1,1,1,1,1,1,2,1,1,2,1,1,1,1,1,1,1,1,1,1,2,1,
  1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,
  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
];
```

Array (as 2D Grid) :-
The entire game map is represented by a single array. Numbers define walls (1), dots (2), and paths (0).

Data Structures & Concepts Used

The Breadth-First Search (BFS) for for Pathfinding

```
export function bfs(start, goal, grid, width) {
  const queue = [[start]];
  const visited = new Set([`${start}`]);
  while (queue.length > 0) {
    const path = queue.shift();
    const current = path[path.length - 1];
    if (current === goal) {
      return path;
    }
    const directions = [-width, width, -1, 1];
    const neighbors = directions.map(dir => current + dir);
    for (const neighbor of neighbors) {
      if (current === 28 * 14 && neighbor === 28 * 14 - 1) {
      } else if (current === 28 * 14 + 27 && neighbor === 28 * 14
      }
      if (
        neighbor >= 0 &&
        neighbor < grid.length &&
        !visited.has(`${neighbor}`) &&
        grid[neighbor] !== 1 // Not a wall
      ) {
        visited.add(`${neighbor}`);
        const newPath = [...path, neighbor];
        queue.push(newPath);
      }
    }
```

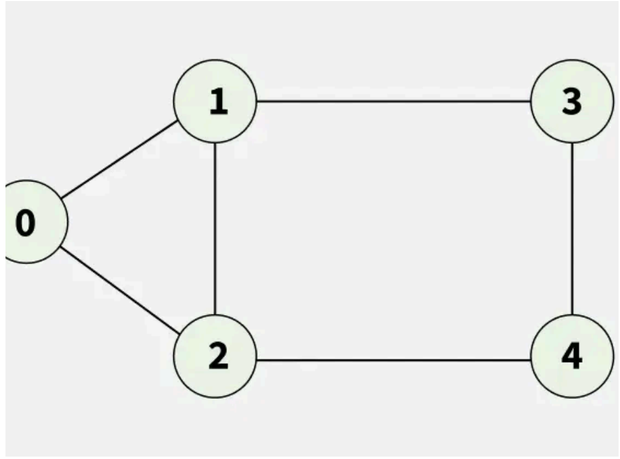
Queue (for Pathfinding)

The Breadth-First Search (BFS) algorithm uses a **Queue** to explore the maze and find the shortest path. In JavaScript, an array's push and shift methods simulate a queue.

Set (for Pathfinding)

A **Set** is used by the BFS algorithm to keep track of tiles that have already been visited. This is extremely fast and prevents the ghost from getting stuck in a loop.

Data Structure Info



Output: [0, 1, 2, 3, 4]
Explanation: Starting from 0, the BFS traversal will follow these steps:
Visit 0 → Output: [0]
Visit 1 (first neighbor of 0) → Output: [0, 1]
Visit 2 (next neighbor of 0) → Output: [0, 1, 2]
Visit 3 (next neighbor of 1) → Output: [0, 1, 2, 3]
Visit 4 (neighbor of 2) → Final Output: [0, 1, 2, 3, 4]
Input: adj[][] = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]

Breadth First Search (BFS) *is a fundamental* ***graph traversal algorithm.*** *It begins with a node, then first traverses all its adjacent nodes. Once all adjacent are visited, then their adjacent are traversed.* Follow the below given approach:

Initialization: Enqueue the given source vertex into a queue and mark it as visited.

Exploration: While the queue is not empty:

Dequeue a node from the queue and visit it (e.g., print its value).

For each unvisited neighbor of the dequeued node:

Enqueue the neighbor into the queue.

Mark the neighbor as visited.

Termination: Repeat step 2 until the queue is empty

CODE And OUTPUT

https://github.com/surajprakash2642006-code/Dsa-Project/tree/main

Time & Space Complexity

Here are the time and space complexity details for your project, in short points.

Time & Space Complexity

This analysis focuses on the most complex part of the project: the **Breadth-First Search (BFS)** algorithm used for the ghost's pathfinding AI.

Time Complexity (Speed)

Complexity: O(V+E)

V = Vertices

E = Edges

Space Complexity (Memory)

Complexity: O(V)

Future Scope & Enhancements

Implement Unique Ghost AI:
Give each ghost its classic "personality"—like one that ambushes (Pinky) and one that wanders (Clyde)—instead of all three just chasing.

Activate Power Pellets:
Add the "frightened mode" where ghosts turn blue after Pac-Man eats a power pellet, allowing him to eat them for extra points.

Add Multiple Levels:
Introduce new maze layouts and increase the game's difficulty after the player clears the board.

Game Polish:
Incorporate the iconic Pac-Man **sound effects** and music.
Add the **bonus fruit** that appears periodically for extra points.

Conclusion

- Successfully developed a complete, playable Pac-Man game using React.js.
- Achieved the primary objective of implementing a DSA-powered enemy AI.
- The Breadth-First Search (BFS) algorithm was used to create intelligent ghosts.
- This project is a successful demonstration of applying CS theory to a practical problem.
- It proves that foundational algorithms are essential for building modern, interactive applications.

References

- ***Breadth First Search (BFS)***(GeekforGeeks): https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/
- Agnihotri, A. (2025, April 28). cJSON - JSON file write/read/modify in C. GeeksforGeeks. Retrieved October 7, 2025, from https://www.geeksforgeeks.org/c/cjson-json-file-write-read-modify-in-c/
- GeeksforGeeks. (2025, September 1). Trie | (insert and search). GeeksforGeeks. Retrieved October 7, 2025, from https://www.geeksforgeeks.org/dsa/trie-insert-and-search/