**Ho Chi Minh City International University**

School of Computer Science and Engineering

A blue and white logo

Description automatically generated

**Algorithms and Data Structure**

**Pacman – Ghosts Pathfinder**

**Submitted by**

Phạm Nguyễn Tiến Đạt ITDSIU21080

Nguyễn Hải Quân ITITWE21104

Nguyễn Hoàng Minh Khôi ITCSIU22217

Nguyễn Hải Phú ITDSIU22179

Dương Ngọc Phương Anh ITDSIU22135

Instructor: Dr. Tran Thanh Tung

*International University*

*School of Computer Science and Engineering*

[**LIST OF FIGURES**](#list_of_figures) 3

[**CONTRIBUTION TABLE**](#contribution_table)4

[**ABSTRACT**](#abstract)5

[**CHAPTER I: INTRODUCTION**](#introduction) 6

1. [OBJECTIVES](#objectives) 6
2. [THE TOOLS USED](#tools) 6

[**CHAPTER II: METHODOLOGY**](#methodology)8

1. [GAME LOGIC](#logic) 8
2. [DESIGN](#design) 9
3. [DATA STRUCTURES](#data_struct) 10
4. [ALGORITHMS](#algorithm) 13
5. [UML DIAGRAM](#uml) 17

[**CHAPTER III: DEMO – RESULT**](#demo) 18

[**CHAPTER IV: CONCLUSION**](#conclusion) 19

1. [CONCLUSION](#conclusion_1) 19
2. [FUTURE WORK](#future_work) 19
3. [ACKNOWLEGEMENT](#acknowlegdement) 19

[**CHAPTER V: REFERENCES**](#references) 20

**LIST OF FIGURES**

[Figure 1.1. A class in JetBrains IntelliJ IDEA](#figure1_1)

[Figure 1.2. GitHub working environment](#figure1_2)

[Figure 2.1. Original positions of Pacman and Ghosts](#figure2_1)

[Figure 2.2. Pacman.gif](#figure2_2)

[Figure 2.3. Ghosts.png](#figure2_3)

[Figure 3.1. Creation of the Ghosts HashSet](#figure3_1)

[Figure 3.2. Calculate the optimal path in a 2D grid map](#figure3_2)

[Figure 3.3. Find the path to Pacman using Depth-First Search](#figure3_3)

[Figure 3.4. Find the path to Pacman using A-Star](#figure3_4)

[Figure 3.5. Find the path to Pacman using Bidirectional Depth-First Search](#figure3_5)

[Figure 3.6. UML diagram](#figure3_6)

**CONTRIBUTION TABLE**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Full Name | Student ID | Contribution |
| 1 | Phạm Nguyễn Tiến Đạt | ITDSIU21080 | 20% |
| 2 | Nguyễn Hải Quân | ITITWE21104 | 20% |
| 3 | Nguyễn Hoàng Minh Khôi | ITCSIU22217 | 20% |
| 4 | Dương Ngọc Phương Anh | ITDSIU22135 | 20% |
| 5 | Nguyễn Hải Phú | ITDSIU22179 | 20% |

**ABSTRACT**

Based on the classic game Pacman that many people played as children, the project presents a modern implementation, featuring intelligent ghost behavior driven by pathfinding algorithms. The ghosts utilize algorithms such as A\*, Depth-First Search (DFS), or Breadth-First Search (BFS) to dynamically chase Pacman, creating a challenging and engaging gameplay experience.

The system integrates real-time decision-making and collision detection, ensuring smooth interactions between game entities. This implementation highlights the practical application of pathfinding techniques in game AI while maintaining the nostalgic essence of the original Pac-Man.

This report provides a comprehensive overview of the developed game. The first section outlines the project's objectives, and the resources utilized. The second section delves into the rules and design of the game in detail. The subsequent section presents the demo and results. Finally, the report concludes with a discussion of key takeaways and potential future enhancements.

**CHAPTER 1: INTRODUCTION**

1. **Object****ives**

The project's objective is to develop a playable game that incorporates the knowledge acquired in the Algorithms and Data Structures course, with a user-friendly user interface and a structure that can be easily modified and improved in the future by the members. In short, this project aims to:

* Create a game for entertainment.
* Apply algorithms and data structures knowledge.
* Gain experience in game development, code optimization, and project management.

1. **The tools used**

* IDEs for programming and debugging: JetBrains IntelliJ IDEA.
* Code version and project management: GitHub.
* Communication & Weekly Meetings: Google Meet.

A screenshot of a computer

Description automatically generated

*Figure 1.1. A class in JetBrains IntelliJ IDEA*

A screenshot of a computer

Description automatically generated

*Figure 1.2. GitHub working environment*

**CHAPTER 2: METHODOLOGY**

1. **Game Logic:**

Welcome to the exciting game Pac-Man, a thrilling journey into a maze full of twists, turns, and challenges! In this game, you’ll step into the shoes of Pac-Man, navigating a labyrinth while dodging clever and persistent ghosts, all while collecting food to boost your score.

A video game with a maze and pacman

Description automatically generated

*Figure 2.1. Original positions of Pacman and Ghosts*

Game Objective: The primary goal is to guide Pacman through the maze, collect as much food as possible, and stay alive.

* **Collect Food:**

The maze is filled with food pellets that Pac-Man can eat. Each pellet you consume adds 1 point to your score. An ultimate challenge is to clear the entire maze by eating every piece of food.

* **Avoid Ghosts:**

The maze is not just about collecting food; you’ll face a group of colorful, crafty ghosts includes Blinky (Red Ghost), Pinky (Pink Ghost), Inky (Cyan Ghost), Clyde (Orange Ghost); with one mission: to catch Pac-Man. Each ghost has its own unique behavior and strategy to make your task harder.

* **Lives and Game Over:**

Pac-Man starts the game with 3 lives, symbolized by small icons at the bottom of the screen. Each time a ghost catches you, you lose a life. Be cautious—if your lives drop to 0, the game is over, and your adventure in the maze comes to an end.

* **Scoring and Winning:**
  + - 1. Points System: Each food pellet consumed adds 1 point to your score. The more food you eat, the higher your score!
      2. Winning the Game: If you manage to eat all the food in the maze while avoiding the ghosts, you win!
* **Control and Strategy**

Movement Controls: Use the arrow keys (or WASD) to move Pac-Man up, down, left, or right. Each direction corresponds to the pathways in the maze. Plan your moves carefully to avoid dead ends and traps set by the ghosts.

1. **Des****ign**
   1. **UI/UX**
      1. **Pacman animations**



*Figure 2.2. Pacman.gif*

* + 1. **Ghost**



*Figure 2.3. Ghosts.png*

1. **Project algorithms and data structures**
   1. **Data structures**
      1. **HashSet**

* Purpose: Store and manage ghosts in the game. The **ghosts** variable is declared as a HashSet, which is a collection that stores unique elements, and does not guarantee specific orders. It provides faster operations for adding, removing, and checking the presence of elements.
* Time Complexity: The loop iterates over all ghosts in the HashSet exactly once. This makes the time complexity O(1), where n is the number of ghosts.

|  |  |  |
| --- | --- | --- |
| Operation | Time complexity | Space complexity |
| Insert | O (1) (average), O(n) (worst) | O(n) |
| Lookup | O (1) (average), O(n) (worst) | O(n) |
| Delete | O (1) (average), O(n) (worst) | O(n) |

* Space Complexity: The HashSet stores all ghosts and their metadata. This makes the space complexity O(n), as it grows linearly with the number of ghosts.

A computer screen shot of a program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generated

*Figure 3.1. Creation of the Ghosts HashSet*

* + 1. **Priority Queue in Uniform Cost Search:**
* Purpose: UCS always explores the least-cost path first, ensuring that the first time it reaches the destination, it has found the optimal path. The cost is measured as the cumulative steps taken to reach a node (no heuristics involved).
* Steps:

Adding: Neighbors are added to the priority queue with their cumulative cost as the priority. Lower-cost nodes are dequeued first, ensuring the search explores the cheapest paths first.

Processing: The node with the lowest cost is dequeued and processed. If this node is the destination, the algorithm stops and returns the first move.

* + 1. **Stack in Depth-First Search:**
* Purpose: A stack is used to keep track of the nodes to be explored next. Nodes are added and removed in a Last-In-First-Out (LIFO) order.
* Steps:

1. Adding: New nodes are pushed onto the stack after expanding the neighbors of the current node. Neighbors are processed in reverse order (Left, Down, Right, Up) so that when popped, the search goes in the desired order (Up, Right, Down, Left).
2. Processing: The most recently added node (the deepest node) is popped from the stack. If this node is the destination, the algorithm stops and returns the first move.
   * 1. Priority Queue in AStar Search:

* Purpose: A priority queue is used where each node is inserted with a priority based on the sum of the cost to reach the node and the heuristic estimate.
* Steps:
  + - 1. Adding: Neighbors are added to the priority queue with their total cost as the priority. Nodes that are closer to the destination (lower heuristic) are prioritized, allowing A\* to focus on promising paths.
      2. Processing: The node with the lowest total cost is dequeued and processed. If this node is the destination, the algorithm stops and returns the first move.
    1. Stacks in Bidirectional Depth-First Search:
* Purpose: Two stacks are used to keep track of the nodes to be explored next, one for the forward search and one for the backward search.
* Steps:

Adding: Neighbors are added to the respective stack (forward or backward) during each search step. For forward search, nodes are added in the direction order (Up, Right, Down, Left). For backward search, nodes are added in reverse direction order (Down, Left, Up, Right).

Processing: Alternate between processing a node from the forward stack and the backward stack. If a node processed from one stack is found to have been visited by the other, the algorithm stops (a path is found).

* 1. **Algorithms**
     1. **Uniform Cost Search**
* Purpose: Blinky uses UCS to determine the shortest path to Pac-Man’s position based on the number of steps required. It prioritizes expanding nodes (maze cells) with the smallest cumulative cost.
* Time Complexity: O(nlogn)

UCS uses a priority queue. Enqueueing and dequeuing from the priority queue takes O(log n) time. In the worst case, it might explore a significant portion of the map (n represents the number of map cells). The combination of these factors leads to O(nlogn) time.

* Space Complexity: O(n)

The primary contributors to space complexity are the priority queue and the visited array. The priority queue can potentially hold a substantial number of map cells, approaching 'n' in the worst case. The visited array also consumes space proportional to the map size.



*Figure 3.2. Find the path to Pacman using Uniform Cost Search*

* + 1. **Depth First Search**
* Purpose: Pinky uses DFS for exploration. It dives deeply into one potential path until it reaches Pac-Man or is forced to backtrack. This creates unpredictable chasing behavior.
* Time Complexity: O(n)

DFS systematically explores paths, visiting each reachable cell once in the worst case.

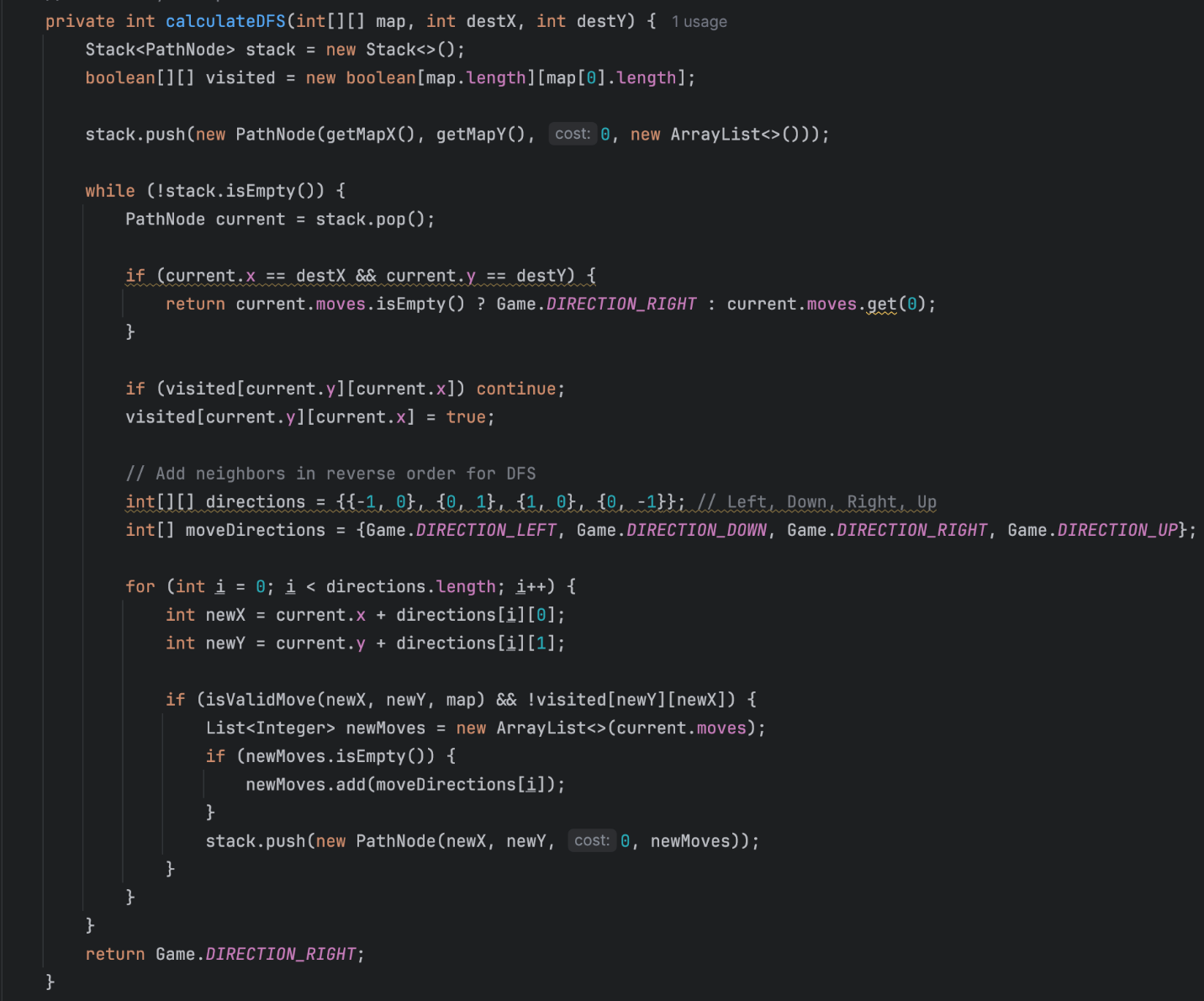
The exploration halts once the target (Pac-Man's location) is found, but in the absence of a shorter route, it may traverse the entire search space.

* Space Complexity: O(n)

The space required is proportional to the depth of the recursion or the size of the stack.

In the worst case, the stack stores up to n cells (e.g., for a single long path).

A visited list may also consume additional space to avoid revisiting cells.



*Figure 3.3. Find the path to Pacman using Depth-First Search*

* + 1. **A\* Search**
* Purpose: Inky uses A\* search for pathfinding, balancing direct chasing and efficiency by incorporating heuristic. The heuristic is the Manhattan distance, estimating the cost to Pac-Man’s position.
* Time Complexity: O(nlogn)

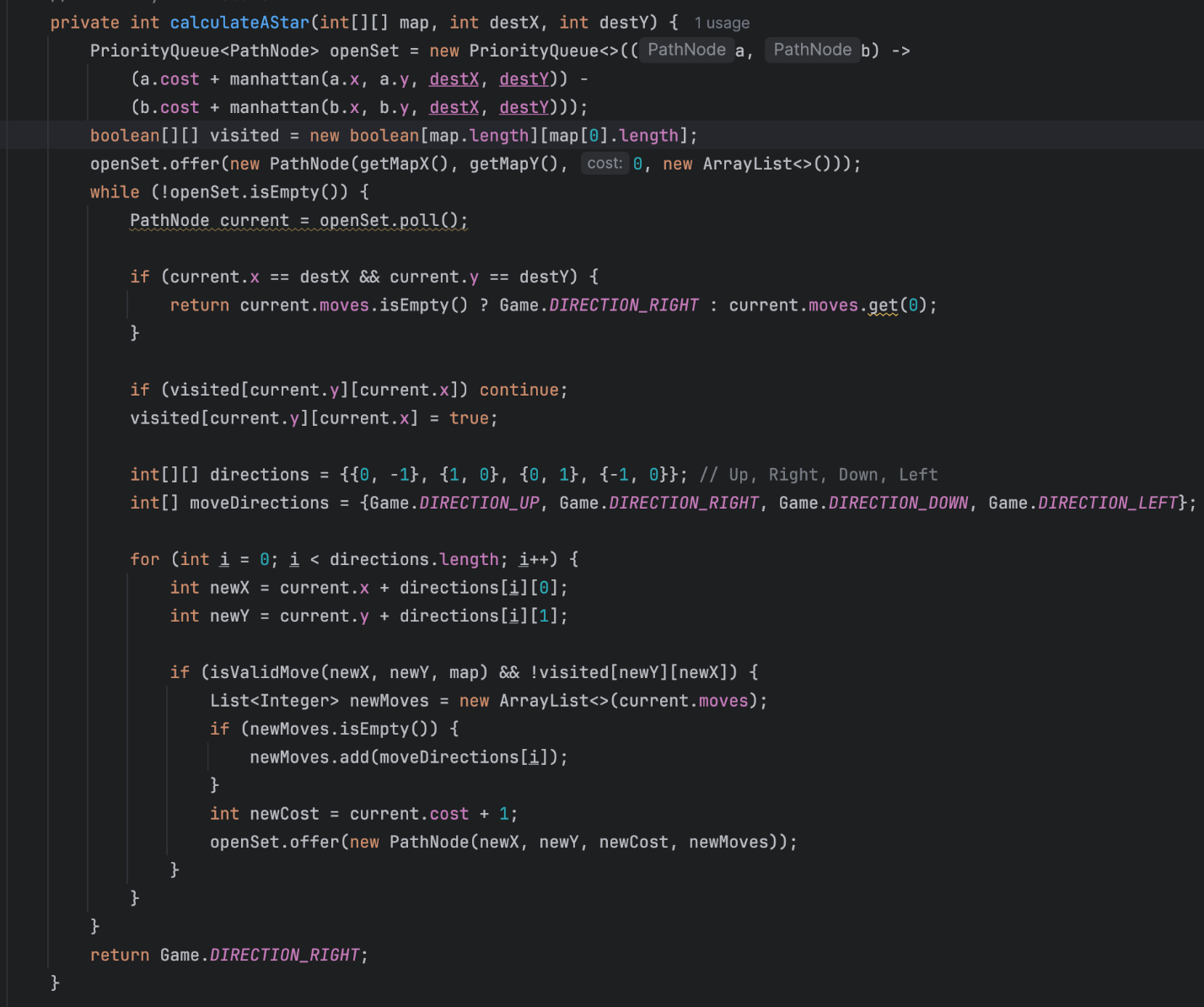
A\* search combines UCS with heuristics to guide the search.

Like UCS, the priority queue operations contribute to a (logn) factor. Exploring all reachable cells adds the n factor, leading to O(nlogn) in total.

* Space Complexity: O(n)

The priority queue store nodes being evaluated, which can grow to n in size in the worst case.

The visited list and data structures for storing path information also require O(n) space.



*Figure 3.4. Find the path to Pacman using A-Star*

* + 1. **Bidirectional Depth-First Search**
* Purpose: Clyde uses BDFS, which involves two simultaneous DFS searches: one starting from Clyde’s position and another from Pac-Man’s position. BDFS uses two stacks to implement DFS in both forward and backward directions simultaneously. The two searches meet in the middle, which Clyde uses to decide its movement.
* Time Complexity: O(n)  
  Bidirectional DFS runs two simultaneous DFS searches:

One starts at Clyde's location and moves toward Pac-Man.

The other starts at Pac-Man's location and moves toward Clyde.

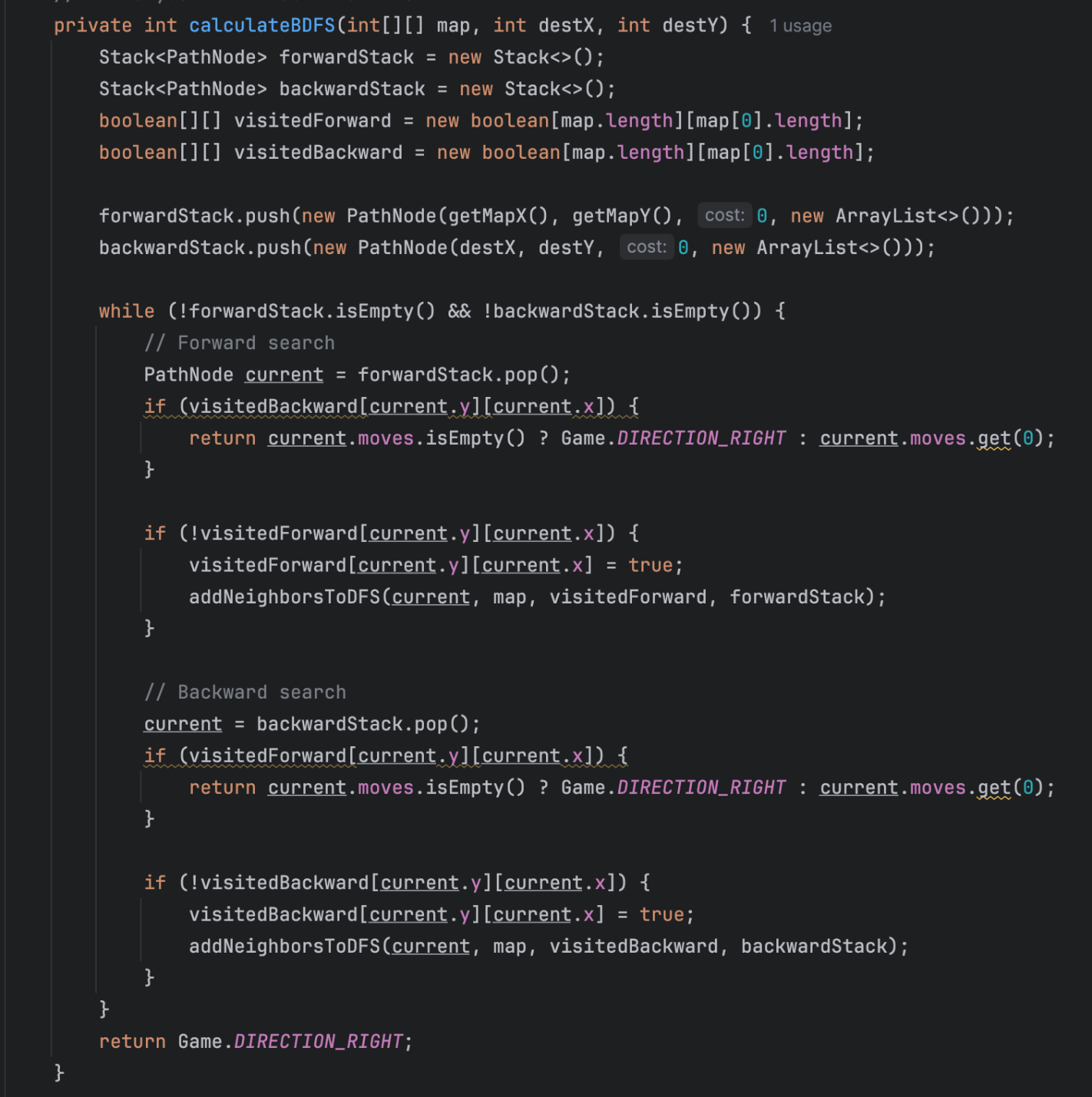
Each search explores approximately half the map in the worst case, resulting in a combined complexity of O(n).

* Space Complexity: O(n)

Each DFS uses its own stack, with a worst-case size of n/2 for each search.

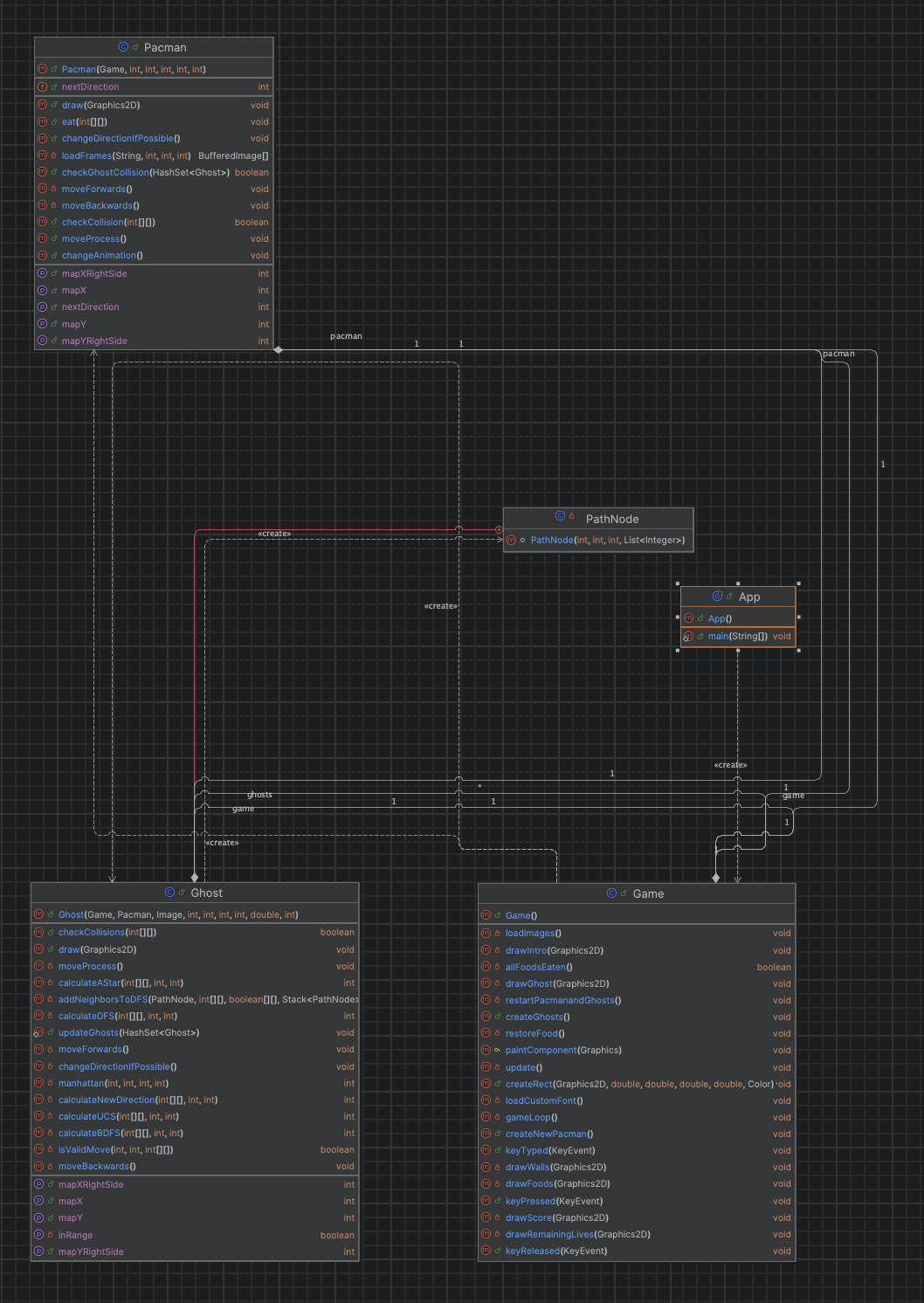
Together, the stacks use O(n) space.

A visited list may also consume O(n) space for tracking explored cells in both searches.



*Figure 3.5. Find the path to Pacman using Bidirectional Depth-First Search*

* 1. **UML Diagram**



*Figure 3.6. UML diagram*

**CHAPTER 3: DEMO – RESULT**

To run the game, users can install this project from GitHub.

To start the game, users need to run the main application. Navigate the project directory. Then, compile and run the application. This will launch the game window with the title “Pac Man”.

Once the game starts, you'll see the game interface with Pac-Man and the ghosts in the maze. The objective is to eat all the dots while avoiding the ghosts.

Pac-Man starts with 3 lives. The number of lives remaining is displayed at the bottom of the screen. The score increases as Pac-Man eats the dots in the maze.

There are four ghosts in the game, each using a different algorithm to chase Pac-Man when he is within their range, if not they will move randomly.

Now you press space or enter to start the game, You can control Pac-Man using the arrow keys on your keyboard

**CHAPTER 4: CONCLUSION AND FUTURE WORK**

Conclusion

Through this project, the team has gained a deeper understanding of algorithms and data structures, particularly in the context of pathfinding. Furthermore, significant improvements have been made in programming and project management skills. This knowledge and experience will be invaluable for future studies and career development.

Future Work

For the game, the team plans to enhance it by introducing essential features such as a timer and advanced player and boss abilities. Additionally, optimizing the code will be a priority to improve the game's overall performance.  
Beyond the game, the team is confident that the knowledge of algorithms and data structures acquired during this project will be effectively applied in future academic and professional endeavors.

Acknowledgment

The team would like to extend heartfelt gratitude to our lecturers, Dr. Tran Thanh Tung and MSc. Thai Trung Tin, for their invaluable guidance and support throughout this project.

**REFERENCES**

Kenny Yip Coding. (2024, October 21). Code Pacman in Java [Video]. YouTube. <https://www.youtube.com/watch?v=lB_J-VNMVpE>.

Servet Gulnaroglu. (2022, November 17). ASMR Programming - Coding Pacman - No Talking [Video]. YouTube. <https://www.youtube.com/watch?v=GXlckaGr0Eo>.

Coder Ph. (2020, November 9). HOW TO MAKE a PACMAN GAME IN JAVA LANGUAGE - PROGRAMMING TUTORIAL FOR BEGINNERS | CS Programmer [Video]. YouTube. <https://www.youtube.com/watch?v=pw-1z5SmXQk>.

Tovmasyan, T. (2021, December 16). AI and Pacman (A story of ghosts’ intelligence) - Tatevik Tovmasyan - Medium. Medium. <https://tateviktome-tovmasyan.medium.com/ai-and-pacman-a-story-of-ghosts-intelligence-d2f296c31675>.