ROBERT GORDON UNIVERSITY ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

BSc (Hons) Artificial Intelligence and Data Science

Module: CM1602

Data Structures and Algorithms for Artificial Intelligence

Individual Coursework Report

Module Leader: Ms. Malsha Fernando

RGU Student ID : 2330912

IIT Student ID : 20230437

Student Name : Kasthuri Arachchi Gihanga Sandothmi

Acknowledgment

I want to sincerely thank Ms. Malsha Fernando for all of her hard work and assistance in helping my project come to such a remarkable conclusion. Only half of the assignment's credit is awarded for its completion. Lastly, and maybe most importantly, I would want to express my gratitude to my friends and supporters, without whom the task would not have been successfully finished on schedule.

Table of Contents

Table of Figures

Introduction

Autonomous path planning in the context of robotics is a difficult but crucial problem for navigating through settings full of obstacles. This project involves using Java programming to enable an autonomous robot to navigate a grid-based environment and choose the best route to go from its starting point to a predetermined destination while avoiding obstacles. In contrast to traditional methods, is avoided using pre-built data structures in favor of unique implementations, which promotes a better comprehension of the underlying ideas. The goal is to give the robot the cognitive abilities it needs to navigate complex environments with skill and accuracy. Specialized grid representation, careful robot state management, sophisticated path planning algorithms, reliable obstacle detection mechanisms, extensive simulation techniques, and an easy-to-use user interface will all be integrated to achieve this.

Algorithms Selection Justification

The Breadth-First Search (BFS) algorithm is a good option for solving the shortest path issue in a grid-based environment with obstacles. Finding the shortest path in an unweighted network, as this problem is, is guaranteed by BFS, which investigates nodes in the order of their distance from the starting point.

Features

- Before advancing to nodes at the next depth level, BFS investigates every neighbor node at the current depth. In unweighted graphs, it ensures the shortest path.

Benefits

- Completeness:

  If the shortest path is available, BFS promises to locate it.

- Optimality:

  BFS investigates nodes at each level, ensuring the shortest path.

- No Heuristic Needed:

  BFS is easier to use for this problem because it doesn't require any heuristic functions or edge weights, in contrast to A*.

Application to the goal

- In a grid-based system with barriers, the goal is to identify the shortest path from a starting point to an endpoint.
- Grid-based issues with discrete (up, down, left, and right) and uniformly cost movements are a good fit for BFS.
- In order to ensure the quickest way, it effectively investigates every avenue from the beginning to the finish.

Conclusion

- BFS is an appropriate solution for this problem because of its completeness, simplicity, optimality, and suitability for grid-based environments with obstacles. It methodically explores every path that could lead from the starting point to the destination and quickly determines the shortest one.

```
public class ShortestPathFindingRobot {
    1 usage
    private static final int[][] DIRECTIONS = {
            {-1, 0}, {0, -1}, {0, 1}, {1, 0} // Non-diagonal directions only
    };
```

```
public static int[][] findShortestPath(char[][] grid, int startRow, int startCol, int endRow, int endCol) {
    int numRows = grid.length;
    int numCols = grid[0].length;
    boolean[][] visited = new boolean[numRows][numCols];
    int[][] distances = new int[numRows][numCols];
    int[][] parentsRow = new int[numRows][numCols];
    int[][] parentsCol = new int[numRows][numCols];
    Queue queue = new Queue( size: numRows * numCols);

    // Initialize distances to infinity and parents to (-1, -1)
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            distances[i][j] = Integer.MAX_VALUE;
            parentsRow[i][j] = -1;
            parentsCol[i][j] = -1;
        }
    }
```

```java
    // Start point has distance 0
    distances[startRow][startCol] = 0;
    queue.insert( item: startRow * numCols + startCol);

    while (!queue.isEmpty()) {
        int currentCell = queue.remove();
        int row = currentCell / numCols;
        int col = currentCell % numCols;
        visited[row][col] = true;

        if (row == endRow && col == endCol) {
            // Reconstruct path
            return reconstructPath(startRow, startCol, endRow, endCol, parentsRow, parentsCol);
        }

        // Explore neighbors
        for (int[] dir : DIRECTIONS) {
            int newRow = row + dir[0];
            int newCol = col + dir[1];
            if (isValid(grid, visited, newRow, newCol)) {
                queue.insert( item: newRow * numCols + newCol);
                visited[newRow][newCol] = true;
                distances[newRow][newCol] = distances[row][col] + 1;
                parentsRow[newRow][newCol] = row;
                parentsCol[newRow][newCol] = col;
            }
        }
    }

    return null; // No path found
}
```

```java
private static boolean isValid(char[][] grid, boolean[][] visited, int row, int col) {
    int numRows = grid.length;
    int numCols = grid[0].length;
    return row >= 0 && row < numRows && col >= 0 && col < numCols && !visited[row][col] && grid[row][col] != 'X';
}
```

```java
private static int[][] reconstructPath(int startRow, int startCol, int endRow, int endCol, int[][] parentsRow, int[][] parentsCol) {
    int currentRow = endRow;
    int currentCol = endCol;
    int length = 0;

    // Count the length of the path
    while (currentRow != startRow || currentCol != startCol) {
        length++;
        int newRow = parentsRow[currentRow][currentCol];
        int newCol = parentsCol[currentRow][currentCol];
        currentRow = newRow;
        currentCol = newCol;
    }

    // Reconstruct the path
    int[][] path = new int[length + 1][2];
    currentRow = endRow;
    currentCol = endCol;
    for (int i = length; i >= 0; i--) {
        path[i][0] = currentRow;
        path[i][1] = currentCol;
        int newRow = parentsRow[currentRow][currentCol];
        int newCol = parentsCol[currentRow][currentCol];
        currentRow = newRow;
        currentCol = newCol;
    }

    return path;
}
```

Figure 1 Relevant code of BFS.

Data Structure Selection Justification

The following data structures are used to solve the given problem of finding the shortest path in a grid-based environment with obstacles:

- 2D Array (Grid): Describes the robot's movement environment. It effectively stores the grid's layout, complete with obstacles, start point, and finish point.

- Queue: Processes nodes quickly in the order they are found during BFS traversal. BFS relies on the queue data structure to help preserve the exploration order.

Analyzing the relevance and the suitability of each data structure.

2D Array (Grid)

Features

- Effectively maintains a cell grid with a rectangular shape.
- Always gives access to the elements.
- makes it simple to manipulate and see the grid environment.

Benefits

- Fit for depicting a grid-based environment with start point, finish point, and obstacles.
- Offers a straightforward and user-friendly method for updating and accessing cell data.

Application to the Problem

- The environment's structure, including obstacles, the start and finish points, is effectively represented by the grid.
- It makes tasks like updating distances, marking visited cells, and verifying motions that are valid easier.

```java
3 usages
class Grid {
    16 usages
    private char[][] grid;

    1 usage
    public Grid(int rows, int columns) {
        grid = new char[rows][columns];
        initializeGrid();
    }

    1 usage
    private void initializeGrid() {
        // Initialize grid with empty cells
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                grid[i][j] = '_'; // Representing empty cell as '_'
            }
        }
    }
```

```java
public void addObstacles(double obstaclePercentage) {
    Random random = new Random();
    int obstacleCount = (int) (grid.length * grid[0].length * obstaclePercentage);
    for (int i = 0; i < obstacleCount; i++) {
        int row = random.nextInt(grid.length);
        int col = random.nextInt(grid[0].length);
        if (grid[row][col] != 'S' && grid[row][col] != 'G') {
            grid[row][col] = 'X'; // Representing obstacles as 'X'
        } else {
            i--; // Retry if obstacle is placed on start or end point
        }
    }
}

12 usages
public char[][] getGrid() {
    return grid;
}

1 usage
public void printGrid() {
    System.out.print("   ");
    for (int i = 0; i < grid[0].length; i++) {
        System.out.printf("%2d", i);
    }
    System.out.println();
    for (int i = 0; i < grid.length; i++) {
        System.out.printf("%2d ", i);
        for (int j = 0; j < grid[0].length; j++) {
            System.out.print(grid[i][j] + " ");
        }
        System.out.println();
    }
}
```

Figure 2 Relevant code of 2D array.

Queue

Features

- Follows the First-In-First-Out (FIFO) theory.
- Allows for continuous time insertion and removal operations.
- Ideal for traversal algorithms based on BFS.

Benefits

- Effectively preserves the exploration order while traversing BFS.
- Makes certain that nodes are handled in the order that they are found, which is essential for determining the shortest path.

Application to the Problem

- To implement BFS traversal, where nodes representing grid cells are processed in the order that they are found, a queue is necessary.
- It assists in methodically investigating nearby cells while guaranteeing that the shortest way is discovered.

```
class Queue {
    4 usages
    private int front;
    4 usages
    private int rear;
    4 usages
    private int noOfItems;
    4 usages
    private int maxSize;
    3 usages
    private int queueArray[];

    1 usage
    public Queue(int size) {
        maxSize = size;
        front = 0;
        rear = -1;
        noOfItems = 0;
        queueArray = new int[maxSize];
    }
```

```java
    public void insert(int item) {
        if (rear == maxSize - 1) {
            rear = -1;
        }
        queueArray[++rear] = item;
        noOfItems++;
    }

    1 usage
    public int remove() {
        int temp = queueArray[front++];
        if (front == maxSize) {
            front = 0;
        }
        noOfItems--;
        return temp;
    }

    1 usage
    public boolean isEmpty() {
        return (noOfItems == 0);
    }
}
```

Figure 3 Relevant code of Queue.

Conclusion

- The shortest path finding problems in a grid-based environment with obstacles is best solved by combining a 2D array (grid) with a queue data structure. The grid effectively depicts the surroundings, and the queue makes BFS traversal easier and guarantees that the shortest path is found quickly and methodically. These data structures offer the framework required to put the algorithm into practice and handle the grid cells during traversal in an effective manner.

Test Plans

Test Case Number	Input	Expected Output	Actual Output	Status
1	15,15	Print the grid, ask for starting nod	Print the grid, ask for starting nod	Pass
2	15,15,6,0	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for starting nod	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for starting nod	Pass
3	15,15,6,0,0,0	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for starting nod	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for starting nod	Pass
4	15,15,6,0,0,0,6,0	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for goal nod	Print the grid, print" obstacle was found at the starting point. Please choose another point.", ask for goal nod	Pass
5	15,15,6,0,0,0,6,0,10,5	Print the grid, print shortest path points and print the path on grid	Print the grid, print shortest path points and print the path on grid	Pass
6	15,15,2,13,14,14	Print the grid, Print 'No available path'	Print the grid, Print 'No available path'	Pass
7	15,15,1,5,14,0	Print the grid, Print 'No available path'	Print the grid, Print 'No available path'	Pass
8	15,15,20	Print the grid, Print 'Please enter a value between 0 and 14' and ask for row value of starting nod again	Print the grid, Print 'Please enter a value between 0 and 14' and ask for row value of starting nod again	Pass
9	15,15,20,12,25	Print the grid, Print 'Please enter a value between 0 and 14' and ask for column value of starting nod again	Print the grid, Print 'Please enter a value between 0 and 14' and ask for column value of starting nod again	Pass

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ _ X _ X _ _ _ X _ _
 1 _ _ X X _ _ X _ _ _ _ _ _ X _
 2 _ _ _ _ _ _ _ _ X _ _ X _ _ _
 3 _ _ _ _ _ X X _ X _ _ _ X _ X
 4 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 5 _ X _ _ X _ _ _ X X _ _ X _ _
 6 X _ _ X _ X _ X _ _ _ _ X _ _
 7 _ _ _ _ _ X X _ _ X _ _ X _ _
 8 X _ _ _ _ _ _ X X _ _ X X _ _
 9 X _ _ _ _ _ X X _ X X _ _ _ _
10 _ _ X _ _ _ _ X _ _ _ X _ _ _
11 X _ X _ _ _ X _ _ _ _ _ _ X X
12 X _ _ _ _ _ _ _ _ X _ _ _ _ X
13 X _ X _ _ X X X _ _ X _ _ X _
14 _ _ _ _ _ _ X _ _ X _ X _ _ _
Enter the row of the start point:
```

Figure 4 Test plan 1.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ _ X _ X _ _ _ X _ _
 1 _ _ X X _ _ X _ _ _ _ _ _ X _
 2 _ _ _ _ _ _ _ _ X _ _ X _ _ _
 3 _ _ _ _ _ X X _ X _ _ _ X _ X
 4 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 5 _ X _ _ X _ _ _ X X _ _ X _ _
 6 X _ _ X _ X _ X _ _ _ _ X _ _
 7 _ _ _ _ _ X X _ _ X _ _ X _ _
 8 X _ _ _ _ _ _ X X _ _ X X _ _
 9 X _ _ _ _ _ X X _ X X _ _ _ _
10 _ _ X _ _ _ _ X _ _ _ X _ _ _
11 X _ X _ _ _ X _ _ _ _ _ _ X X
12 X _ _ _ _ _ _ _ _ X _ _ _ _ X
13 X _ X _ _ X X X _ _ X _ _ X _
14 _ _ _ _ _ _ X _ _ X _ X _ _ _
Enter the row of the start point: 6
Enter the column of the start point: 0
Obstacle found at the starting point. Please choose another point.
Enter the row of the start point:
```

Figure 5 Test plan 2.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ _ X _ X _ _ _ X _ _
 1 _ _ X X _ _ X _ _ _ _ _ _ X _
 2 _ _ _ _ _ _ _ _ X _ _ X _ _ _
 3 _ _ _ _ _ X X _ X _ _ _ X _ X
 4 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 5 _ X _ _ X _ _ _ X X _ _ X _ _
 6 X _ _ X _ X _ X _ _ _ _ X _ _
 7 _ _ _ _ _ X X _ _ X _ _ X _ _
 8 X _ _ _ _ _ _ X X _ _ X X _ _
 9 X _ _ _ _ _ X X _ X X _ _ _ _
10 _ _ X _ _ _ _ X _ _ _ X _ _ _
11 X _ X _ _ _ X _ _ _ _ _ _ X X
12 X _ _ _ _ _ _ _ _ X _ _ _ _ X
13 X _ X _ _ X X X _ _ X _ _ X _
14 _ _ _ _ _ _ X _ _ X _ X _ _ _
Enter the row of the start point: 6
Enter the column of the start point: 0
Obstacle found at the starting point. Please choose another point.
Enter the row of the start point: 0
Enter the column of the start point: 0
Enter the row of the end point:
```

Figure 6 Test plan 3.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ _ X _ X _ _ _ X _ _
 1 _ _ X X _ _ X _ _ _ _ _ _ X _
 2 _ _ _ _ _ _ _ _ X _ _ X _ _ _
 3 _ _ _ _ _ X X _ X _ _ _ X _ X
 4 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 5 _ X _ _ X _ _ _ X X _ _ X _ _
 6 X _ _ X _ X _ X _ _ _ _ X _ _
 7 _ _ _ _ _ X X _ _ X _ _ X _ _
 8 X _ _ _ _ _ _ X X _ _ X X _ _
 9 X _ _ _ _ _ X X _ X X _ _ _ _
10 _ _ X _ _ _ _ X _ _ _ X _ _ _
11 X _ X _ _ _ X _ _ _ _ _ _ X X
12 X _ _ _ _ _ _ _ _ X _ _ _ _ X
13 X _ X _ _ X X X _ _ X _ _ X _
14 _ _ _ _ _ _ X _ _ X _ X _ _ _
Enter the row of the start point: 6
Enter the column of the start point: 0
Obstacle found at the starting point. Please choose another point.
Enter the row of the start point: 0
Enter the column of the start point: 0
Enter the row of the end point: 6
Enter the column of the end point: 0
Obstacle found at the ending point. Please choose another point.
Enter the row of the end point:
```

Figure 7 Test plan 4.

```
Enter the row of the end point: 10
Enter the column of the end point: 5
Shortest path points:
(0, 0)
(0, 1)
(1, 1)
(2, 1)
(2, 2)
(3, 2)
(4, 2)
(5, 2)
(6, 2)
(7, 2)
(7, 3)
(7, 4)
(8, 4)
(8, 5)
(9, 5)
(10, 5)
```

Figure 8 Test case 5.1.

```
Grid with shortest path:
    0 1 2 3 4 5 6 7 8 91011121314
  0 S * _ _ _ _ X _ X _ _ _ X _ _
  1 _ * X X _ _ X _ _ _ _ _ _ X _
  2 _ * * _ _ _ _ _ X _ _ X _ _ _
  3 _ _ * _ _ X X _ X _ _ _ X _ X
  4 _ _ * _ _ _ _ _ _ _ _ _ _ _ _
  5 _ X * _ X _ _ _ X X _ _ X _ _
  6 X _ * X _ X _ X _ _ _ _ X _ _
  7 _ _ * * * X X _ _ X _ _ X _ _
  8 X _ _ _ * * _ X X _ _ X X _ _
  9 X _ _ _ _ * X X _ X X _ _ _ _
 10 _ _ X _ _ G _ X _ _ _ X _ _ _
 11 X _ X _ _ _ X _ _ _ _ _ _ X X
 12 X _ _ _ _ _ _ _ _ X _ _ _ _ X
 13 X _ X _ _ X X X _ _ X _ _ X _
 14 _ _ _ _ _ _ X _ _ X _ X _ _ _


Process finished with exit code 0
```

Figure 9 Test case 5.2.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ X _ _ _ _ _ _ _ _ X
 1 X _ X _ _ X X X _ _ X _ X X _
 2 _ _ _ _ X _ _ X _ _ _ X _ _ X
 3 _ _ X _ _ _ _ _ _ X _ X X X _
 4 X _ X _ _ _ _ _ X _ _ _ _ _ _
 5 _ _ _ _ _ _ _ X _ X _ X X _ _
 6 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 7 _ _ _ _ _ _ X _ X _ _ _ _ _ _
 8 X _ _ _ _ _ _ X _ _ _ _ X _ _
 9 X _ X _ _ _ _ _ X _ _ X _ _ _
10 _ _ _ _ _ _ X _ _ _ _ _ _ X _
11 _ _ _ X X _ _ X X _ X _ _ _ X
12 _ _ _ X _ _ _ _ _ X _ _ _ _ X
13 _ _ _ _ X _ _ _ _ _ X _ _ _ X
14 X X _ _ _ _ _ _ X X X _ X _ _
Enter the row of the start point: 2
Enter the column of the start point: 13
Enter the row of the end point: 14
Enter the column of the end point: 14
No available path!

Process finished with exit code 0
```

Figure 10 Test case 6.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ _ _ _ _ X X _ _ X _ _
 1 _ _ X _ _ _ _ X X _ X _ _ X X
 2 X _ _ _ _ _ _ X _ _ _ X _ _ _
 3 _ X _ _ _ X _ _ _ X X _ _ _ _
 4 X _ _ X _ _ X _ X X X _ _ X X
 5 _ _ _ _ _ _ _ _ _ X X X _ _ _
 6 _ X _ _ _ _ _ _ _ _ X _ X _ _
 7 _ _ _ _ _ _ X _ _ _ _ X _ _ _
 8 _ _ _ _ _ X _ X _ _ _ _ _ _ X
 9 _ X _ _ _ _ _ X _ _ _ _ _ X _
10 _ _ _ X _ _ X _ X _ _ X X X _
11 _ X X X _ _ X _ _ X _ X _ _ _
12 X _ X X _ _ _ X _ _ _ _ _ X _
13 X _ X _ X _ _ _ _ _ _ _ _ _ _
14 _ _ _ _ X _ _ _ _ X _ X _ _ _
Enter the row of the start point: 1
Enter the column of the start point: 5
Enter the row of the end point: 14
Enter the column of the end point: 0
No available path!
```

Figure 11 Test case 7.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ X _ _ _ X _ X _ _ _ _
 1 _ _ _ _ _ X _ X X _ _ X _ _ _
 2 _ _ X _ _ _ X _ _ _ _ X _ _ X
 3 _ _ X _ _ _ X X _ _ _ _ _ _ _
 4 X _ X _ _ _ _ _ _ _ X _ _ _ X
 5 _ _ _ _ X _ _ _ _ _ X _ X _ _
 6 _ X X _ X _ _ _ _ X X _ X _ _
 7 _ _ _ X _ X _ _ X _ _ _ _ X _
 8 _ _ _ _ _ X _ _ X _ _ _ _ _ _
 9 X _ _ _ _ _ X _ _ _ _ _ _ _ _
10 X _ _ X _ X X _ _ X _ _ _ _ _
11 X _ _ X _ _ _ _ X _ _ _ X X _
12 X X _ X _ X _ _ _ _ _ _ _ _ _
13 X _ _ X X _ _ _ _ X _ _ _ _ _
14 X _ _ _ _ _ _ X X _ X _ _ _ X
Enter the row of the start point: 20
Please enter a value between 0 and 14.
Enter the row of the start point:
```

Figure 12 Test case 8.

```
Enter the number of rows (must be more than 1): 15
Enter the number of columns (must be more than 1): 15
    0 1 2 3 4 5 6 7 8 91011121314
 0 _ _ _ _ X _ _ _ X _ X _ _ _ _
 1 _ _ _ _ _ X _ X X _ _ X _ _ _
 2 _ _ X _ _ _ X _ _ _ _ X _ _ X
 3 _ _ X _ _ _ X X _ _ _ _ _ _ _
 4 X _ X _ _ _ _ _ _ _ X _ _ _ X
 5 _ _ _ _ X _ _ _ _ _ X _ X _ _
 6 _ X X _ X _ _ _ _ X X _ X _ _
 7 _ _ _ X _ X _ _ X _ _ _ _ X _
 8 _ _ _ _ _ X _ _ X _ _ _ _ _ _
 9 X _ _ _ _ _ X _ _ _ _ _ _ _ _
10 X _ _ X _ X X _ _ X _ _ _ _ _
11 X _ _ X _ _ _ _ X _ _ _ X X _
12 X X _ X _ X _ _ _ _ _ _ _ _ _
13 X _ _ X X _ _ _ _ X _ _ _ _ _
14 X _ _ _ _ _ _ X X _ X _ _ _ X
Enter the row of the start point: 20
Please enter a value between 0 and 14.
Enter the row of the start point: 12
Enter the column of the start point: 25
Please enter a value between 0 and 14.
Enter the column of the start point:
```

Figure 13 Test case 9.

Conclusion

A customized solution for autonomous robot path planning in an obstacle-filled grid-based environment is provided by this program. The code successfully navigates the robot from its starting position to the goal while avoiding collisions by incorporating specialized data structures, like the queue, and algorithms, like BFS.

References

- *Java packages*, 2024. [online]. W3schools.com. Available from: https://www.w3schools.com/java/java_packages.asp [Accessed 20 Mar 2024].
- *Java modifiers*, 2024. [online]. W3schools.com. Available from: https://www.w3schools.com/java/java_modifiers.asp [Accessed 19 Mar 2024].
- JENKOV, J., 2020. *Java Queue*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=Gpu52H4yCMM [Accessed 22 Mar 2024].
- BARI, A., 2018. *5.1 graph traversals - BFS & DFS -breadth first search and depth first search*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=pcKY4hjDrxk [Accessed 20 Mar 2024].
- *Breadth first search or BFS for a graph*, 2012. [online]. GeeksforGeeks. Available from: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/ [Accessed 21 Mar 2024].