

Report on my 2D Java Game template

Author: Idriss Chadili

December 28, 2025

Abstract

This report presents the implementation of a 2D game where a player navigates through an environment containing obstacles using a pathfinding algorithm. The algorithm used, Breadth-First Search (BFS), determines whether a path exists between the player's position and a goal, taking obstacles into account. The report details the main classes of the game, the pathfinding algorithm, and the matrices representing the game, to ensure a detailed understanding of the game's mechanics.

Contents

1	Introduction	2
2	Game Architecture	2
2.1	Player Class	2
2.2	Level Class	2
2.3	Pathfinding Algorithm (BFS)	2
3	Matrix Model and Game Representation	2
3.1	Cell Neighborhood	2
4	BFS Algorithm Implementation	3
5	Grid Display and Graphical Interface	4
5.1	Grid Rendering	4
6	Conclusion	4

1 Introduction

This project involves creating a 2D game where a player must move through a grid containing obstacles. The goal is to determine whether a viable path exists between the player's position and the goal, using a pathfinding algorithm like Breadth-First Search (BFS). The main objective of the game is to simulate the movement of a player on a grid while managing obstacles and finding a path to a predefined goal.

2 Game Architecture

2.1 Player Class

The `Player` class is responsible for managing the player's position on the grid as well as the movements made. When a movement is made, the class checks whether the player's new position is valid (ensuring that the new position does not contain any obstacles and is within the grid boundaries).

2.2 Level Class

The `Level` class generates the grid in which the player moves. It also includes an algorithm to check if a path exists between the player's current position and the goal. Obstacles are randomly placed on the grid, and the goal is located at a specific position.

2.3 Pathfinding Algorithm (BFS)

The Breadth-First Search (BFS) algorithm is a classical algorithm used to systematically explore a graph. It explores the neighbors of a point starting from the initial position and uses a queue to visit each cell, checking if a path exists to the goal while avoiding obstacles.

3 Matrix Model and Game Representation

The game is represented as a matrix. Each cell can be empty, contain an obstacle, or be the player's starting position or goal. Each cell is identified by its coordinates (x, y) , and the model is based on a square matrix, usually of size $n \times n$.

$$\begin{bmatrix} S & 0 & 0 & O & G \\ 0 & O & 0 & O & 0 \\ 0 & O & 0 & O & 0 \\ 0 & 0 & 0 & O & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Here, S represents the player's starting position, G the goal, O an obstacle, and 0 an empty cell.

3.1 Cell Neighborhood

The player's movements are defined based on the immediate neighbors of the current cell. Each cell (x, y) has 4 direct neighbors (up, down, left, right), with coordinates calculated by the following relations:

$$\text{Neighborhood of } (x, y) : \begin{cases} (x - 1, y) & \text{left} \\ (x + 1, y) & \text{right} \\ (x, y - 1) & \text{up} \\ (x, y + 1) & \text{down} \end{cases}$$

The BFS algorithm checks each neighbor to determine if the player can move in that direction without hitting an obstacle.

4 BFS Algorithm Implementation

The BFS algorithm explores the graph starting from the starting cell. Each cell is added to a queue and marked as visited. The algorithm continues exploring until the goal is reached or the queue is empty, meaning no path exists. The Java implementation of this algorithm is as follows:

```

public class Pathfinding {
    public static boolean isPathExists(int startX,
        int startY,
        int goalX,
        int goalY,
        boolean[][] obstacles) {
        boolean[][] visited = new boolean[Level.GRID_SIZE][Level.GRID_SIZE];
        Queue<Point> queue = new LinkedList<>();
        queue.add(new Point(startX, startY));
        visited[startY][startX] = true;

        int[] directions = {-1, 0, 1, 0, 0, -1, 0, 1};

        while (!queue.isEmpty()) {
            Point current = queue.poll();
            int x = current.x;
            int y = current.y;

            if (x == goalX && y == goalY) {
                return true;
            }

            for (int i = 0; i < 4; i++) {
                int nx = x + directions[i * 2];
                int ny = y + directions[i * 2 + 1];

                if (nx >= 0 &&
                    ny >= 0 &&
                    nx < Level.GRID_SIZE
                    && ny < Level.GRID_SIZE
                    && !visited[ny][nx]
                    && !obstacles[ny][nx]) {
                    queue.add(new Point(nx, ny));
                    visited[ny][nx] = true;
                }
            }
        }

        return false;
    }
}

```

This algorithm systematically checks each cell, adding each valid cell to the queue. When the goal is reached, it returns ‘true’; otherwise, it returns ‘false’ if no path is found.

5 Grid Display and Graphical Interface

The game uses Java Swing for graphical display. The grid consists of square cells, with obstacles represented by red squares. The player is represented by a blue square, and the goal by a green square.

5.1 Grid Rendering

Here is an example of code to draw the grid and obstacles in the game window:

```
public class GridGame extends JPanel {
    private Player player;
    private Level level;

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        level.draw(g);
    }

    public void draw(Graphics g) {
        for (int i = 0; i < Level.GRID_SIZE; i++) {
            for (int j = 0; j < Level.GRID_SIZE; j++) {
                if (obstacles[i][j]) {
                    g.setColor(Color.RED);
                } else {
                    g.setColor(Color.WHITE);
                }
                g.fillRect(j * cellSize, i * cellSize, cellSize, cellSize);
            }
        }
    }
}
```

6 Conclusion

The 2D game developed uses the Breadth-First Search (BFS) algorithm to find a path between the player's position and the goal, taking obstacles on the grid into account. The graphical interface, while simple, allows the visualization of different game states and manages the player's movements. This project demonstrates the application of algorithmic concepts in the creation of interactive games and can be extended to include additional features such as enemies, more complex levels, or extra game elements.