Rapport sur mon modèle de Jeu 2D en Java

Auteur: Idriss Chadili

December 28, 2025

Abstract

Ce rapport présente l'implémentation d'un jeu 2D où un joueur navigue dans un environnement comportant des obstacles en utilisant un algorithme de recherche de chemin. L'algorithme utilisé, Recherche en Largeur (BFS), permet de déterminer si un chemin existe entre la position du joueur et un objectif en tenant compte des obstacles. Le rapport détaille les classes principales du jeu, l'algorithme de recherche de chemin ainsi que les matrices représentant le jeu, en vue d'assurer une compréhension détaillée de la mécanique du jeu.

Contents

1 Introduction

Ce projet consiste en la création d'un jeu en 2D où un joueur doit se déplacer à travers une grille contenant des obstacles. L'objectif est de déterminer s'il existe un chemin viable entre la position du joueur et l'objectif, en utilisant un algorithme de recherche de chemin comme la Recherche en Largeur (BFS). L'objectif principal du jeu est de simuler le déplacement d'un joueur sur une grille tout en gérant les obstacles et en trouvant un chemin vers un objectif prédéfini.

2 Architecture du Jeu

2.1 Classe Player

La classe `Player` est responsable de la gestion de la position du joueur sur la grille ainsi que des déplacements effectués. Lorsqu'un mouvement est effectué, la classe vérifie la validité de la position du joueur (en s'assurant que la nouvelle position ne contient pas d'obstacle et qu'elle est dans les limites de la grille).

2.2 Classe Level

La classe `Level` génère une grille où le joueur évolue. Elle contient également un algorithme pour vérifier si un chemin existe entre la position actuelle du joueur et l'objectif. Les obstacles sont placés de manière aléatoire dans la grille et l'objectif est situé à une position spécifique.

2.3 Algorithme de Recherche de Chemin (BFS)

L'algorithme de Recherche en Largeur (BFS) est un algorithme classique utilisé pour explorer un graphe de manière systématique. Il explore les voisins d'un point en partant de la position initiale et utilise une file d'attente pour visiter chaque cellule, en vérifiant si un chemin existe jusqu'à l'objectif tout en évitant les obstacles.

3 Modèle Matriciel et Représentation du Jeu

Le jeu est représenté sous forme de matrice. Chaque cellule peut être vide, contenir un obstacle ou être la position de départ ou d'objectif du joueur. Chaque cellule est identifiée par ses coordonnées (x, y), et le modèle est basé sur une matrice carrée, généralement de taille $n \times n$.

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

Ici, S représente la position de départ du joueur, G l'objectif, O un obstacle et 0 une cellule libre.

3.1 Voisinage des Cellules

Les mouvements du joueur sont définis en fonction des voisins immédiats de la cellule courante. Chaque cellule (x, y) possède 4 voisins directs (haut, bas, gauche, droite), dont les coordonnées sont calculées par les relations suivantes :

$$\text{Voisinage de } (x, y) : \begin{cases} (x-1, y) & \text{gauche} \\ (x+1, y) & \text{droite} \\ (x, y-1) & \text{haut} \\ (x, y+1) & \text{bas} \end{cases}$$

L'algorithme BFS vérifie chaque voisin pour déterminer si le joueur peut se déplacer dans cette direction sans rencontrer d'obstacle.

4 Implémentation de l'Algorithme BFS

L'algorithme BFS explore le graphe en commençant par la cellule de départ. Chaque cellule est ajoutée à une file d'attente et est marquée comme visitée. L'algorithme poursuit l'exploration jusqu'à ce que l'objectif soit atteint ou que la file d'attente soit vide, ce qui signifie qu'il n'y a pas de chemin. L'implémentation en Java de cet algorithme est la suivante :

```
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;
    }
}
```

Cet algorithme vérifie systématiquement chaque cellule, en ajoutant chaque cellule valide à la file d'attente. Lorsque l'objectif est atteint, il renvoie 'true', sinon il retourne 'false' si aucun chemin n'est trouvé.

5 Affichage de la Grille et Interface Graphique

Le jeu utilise Java Swing pour l'affichage graphique. La grille est constituée de cellules carrées, et les obstacles sont représentés par des cases rouges. Le joueur est représenté par un carré bleu, et l'objectif par un carré vert.

5.1 Rendu de la Grille

Voici un exemple de code pour dessiner la grille et les obstacles dans la fenêtre du jeu :

```
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

Le jeu 2D développé utilise l'algorithme de Recherche en Largeur (BFS) pour rechercher un chemin entre la position du joueur et l'objectif en tenant compte des obstacles présents sur la grille. L'interface graphique, bien que simple, permet de visualiser les différents états du jeu et de gérer les mouvements du joueur. Ce projet démontre l'application des concepts algorithmiques dans la création de jeux interactifs et peut être étendu pour intégrer de nouvelles fonctionnalités telles que des ennemis, des niveaux plus complexes ou des éléments de jeu supplémentaires.