#sinh mê cung 1.1

int cols = 16, rows = 16;

int cellSize = 40;

Cell[][] grid;

ArrayList<Cell> stack = new ArrayList<Cell>();

Cell current;

void setup() {

size(16\*40, 16\*40);

frameRate(30);

grid = new Cell[cols][rows];

// Khởi tạo lưới mê cung

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

grid[i][j] = new Cell(i, j);

}

}

// Bắt đầu từ ô (0,0)

current = grid[0][0];

current.visited = true;

stack.add(current);

}

void draw() {

background(255);

// Vẽ mê cung

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

grid[i][j].show();

}

}

// Thuật toán tạo mê cung (DFS Backtracking)

if (stack.size() > 0) {

Cell next = current.checkNeighbors();

if (next != null) {

next.visited = true;

stack.add(next);

removeWalls(current, next);

current = next;

} else {

current = stack.remove(stack.size() - 1);

}

}

// Đánh dấu vị trí xuất phát

fill(0, 255, 0);

rect(0, 0, cellSize, cellSize);

// Đánh dấu đích (4 ô giữa)

fill(255, 0, 0);

rect(7 \* cellSize, 7 \* cellSize, cellSize, cellSize);

rect(8 \* cellSize, 7 \* cellSize, cellSize, cellSize);

rect(7 \* cellSize, 8 \* cellSize, cellSize, cellSize);

rect(8 \* cellSize, 8 \* cellSize, cellSize, cellSize);

}

void removeWalls(Cell a, Cell b) {

int x = a.i - b.i;

int y = a.j - b.j;

if (x == 1) {

a.walls[3] = false;

b.walls[1] = false;

} else if (x == -1) {

a.walls[1] = false;

b.walls[3] = false;

}

if (y == 1) {

a.walls[0] = false;

b.walls[2] = false;

} else if (y == -1) {

a.walls[2] = false;

b.walls[0] = false;

}

}

class Cell {

int i, j;

boolean[] walls = {true, true, true, true}; // top, right, bottom, left

boolean visited = false;

Cell(int i, int j) {

this.i = i;

this.j = j;

}

void show() {

int x = i \* cellSize;

int y = j \* cellSize;

stroke(0);

if (walls[0]) line(x, y, x + cellSize, y);

if (walls[1]) line(x + cellSize, y, x + cellSize, y + cellSize);

if (walls[2]) line(x, y + cellSize, x + cellSize, y + cellSize);

if (walls[3]) line(x, y, x, y + cellSize);

if (walls[0] && walls[1] && walls[2] && walls[3]) {

fill(0); // Bôi màu đen nếu có tường ở cả 4 phía

} else {

noFill(); // Để trống nếu không có tường

}

noStroke();

rect(x, y, cellSize, cellSize);

}

Cell checkNeighbors() {

ArrayList<Cell> neighbors = new ArrayList<Cell>();

if (i > 0 && !grid[i-1][j].visited) neighbors.add(grid[i-1][j]);

if (i < cols-1 && !grid[i+1][j].visited) neighbors.add(grid[i+1][j]);

if (j > 0 && !grid[i][j-1].visited) neighbors.add(grid[i][j-1]);

if (j < rows-1 && !grid[i][j+1].visited) neighbors.add(grid[i][j+1]);

if (neighbors.size() > 0) {

return neighbors.get(int(random(neighbors.size())));

} else {

return null;

}

}

}

#1.2 sinh mê cung tự động và BFS vẽ đường

import java.util.\*;

int cols = 16, rows = 16;

int cellSize = 40;

Cell[][] grid;

ArrayList<Cell> stack = new ArrayList<Cell>();

Cell current;

boolean solving = false;

ArrayList<Cell> path = new ArrayList<Cell>();

void setup() {

size(640, 640);

frameRate(30);

grid = new Cell[cols][rows];

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

grid[i][j] = new Cell(i, j);

}

}

current = grid[0][0];

current.visited = true;

stack.add(current);

}

void draw() {

background(255);

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

grid[i][j].show();

}

}

if (stack.size() > 0) {

Cell next = current.checkNeighbors();

if (next != null) {

next.visited = true;

stack.add(next);

removeWalls(current, next);

current = next;

} else {

current = stack.remove(stack.size() - 1);

}

} else if (!solving) {

solveMazeBFS();

solving = true;

}

fill(0, 255, 0);

rect(0, 0, cellSize, cellSize);

fill(255, 0, 0);

rect(7 \* cellSize, 7 \* cellSize, cellSize, cellSize);

rect(8 \* cellSize, 7 \* cellSize, cellSize, cellSize);

rect(7 \* cellSize, 8 \* cellSize, cellSize, cellSize);

rect(8 \* cellSize, 8 \* cellSize, cellSize, cellSize);

for (Cell c : path) {

fill(0, 0, 255);

rect(c.i \* cellSize + cellSize / 4, c.j \* cellSize + cellSize / 4, cellSize / 2, cellSize / 2);

}

}

void removeWalls(Cell a, Cell b) {

int x = a.i - b.i;

int y = a.j - b.j;

if (x == 1) {

a.walls[3] = false;

b.walls[1] = false;

} else if (x == -1) {

a.walls[1] = false;

b.walls[3] = false;

}

if (y == 1) {

a.walls[0] = false;

b.walls[2] = false;

} else if (y == -1) {

a.walls[2] = false;

b.walls[0] = false;

}

}

void solveMazeBFS() {

Queue<Cell> queue = new LinkedList<>();

HashMap<Cell, Cell> cameFrom = new HashMap<>();

Cell start = grid[0][0];

Cell end = grid[7][7];

queue.add(start);

cameFrom.put(start, null);

while (!queue.isEmpty()) {

Cell current = queue.poll();

if (current == end) break;

for (Cell neighbor : current.getValidNeighbors()) {

if (!cameFrom.containsKey(neighbor)) {

queue.add(neighbor);

cameFrom.put(neighbor, current);

}

}

}

Cell step = end;

while (step != null) {

path.add(step);

step = cameFrom.get(step);

}

}

class Cell {

int i, j;

boolean[] walls = {true, true, true, true};

boolean visited = false;

Cell(int i, int j) {

this.i = i;

this.j = j;

}

void show() {

int x = i \* cellSize;

int y = j \* cellSize;

stroke(0);

if (walls[0]) line(x, y, x + cellSize, y);

if (walls[1]) line(x + cellSize, y, x + cellSize, y + cellSize);

if (walls[2]) line(x, y + cellSize, x + cellSize, y + cellSize);

if (walls[3]) line(x, y, x, y + cellSize);

if (walls[0] && walls[1] && walls[2] && walls[3]) {

fill(0);

} else {

noFill();

}

noStroke();

rect(x, y, cellSize, cellSize);

}

Cell checkNeighbors() {

ArrayList<Cell> neighbors = new ArrayList<Cell>();

if (i > 0 && !grid[i-1][j].visited) neighbors.add(grid[i-1][j]);

if (i < cols-1 && !grid[i+1][j].visited) neighbors.add(grid[i+1][j]);

if (j > 0 && !grid[i][j-1].visited) neighbors.add(grid[i][j-1]);

if (j < rows-1 && !grid[i][j+1].visited) neighbors.add(grid[i][j+1]);

if (neighbors.size() > 0) {

return neighbors.get(int(random(neighbors.size())));

} else {

return null;

}

}

ArrayList<Cell> getValidNeighbors() {

ArrayList<Cell> neighbors = new ArrayList<>();

if (i > 0 && !walls[3]) neighbors.add(grid[i-1][j]);

if (i < cols-1 && !walls[1]) neighbors.add(grid[i+1][j]);

if (j > 0 && !walls[0]) neighbors.add(grid[i][j-1]);

if (j < rows-1 && !walls[2]) neighbors.add(grid[i][j+1]);

return neighbors;

}

}

#1.3 giải mã map từ file .txt, vẽ chuẩn. Nhưng đang gặp vấn đề vẽ đường đi

import java.util.\*;

import java.io.\*;

int cols = 16, rows = 16;

int cellSize = 40;

Cell[][] grid;

ArrayList<Cell> stack = new ArrayList<Cell>();

Cell current;

boolean solving = false;

ArrayList<Cell> path = new ArrayList<Cell>();

Cell start, end;

void setup() {

size(640, 640);

frameRate(30);

grid = new Cell[cols][rows];

selectMazeFile();

}

void draw() {

background(255);

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

if (grid[i][j] != null) {

grid[i][j].show();

}

}

}

if (!solving && start != null && end != null) {

solveMazeBFS();

solving = true;

}

if (start != null) {

fill(0, 255, 0);

rect(start.i \* cellSize, start.j \* cellSize, cellSize, cellSize);

}

if (end != null) {

fill(255, 0, 0);

rect(end.i \* cellSize, end.j \* cellSize, cellSize, cellSize);

}

for (Cell c : path) {

fill(0, 0, 255);

rect(c.i \* cellSize + cellSize / 4, c.j \* cellSize + cellSize / 4, cellSize / 2, cellSize / 2);

}

}

void selectMazeFile() {

selectInput("Select a maze file:", "fileSelected");

}

void fileSelected(File selection) {

if (selection != null) {

loadMaze(selection.getAbsolutePath());

} else {

println("No file selected.");

}

}

void loadMaze(String filename) {

try {

BufferedReader br = new BufferedReader(new FileReader(filename));

String line;

int row = 0;

while ((line = br.readLine()) != null && row < rows \* 2 + 1) {

if (row % 2 == 0) {

// Dòng chứa các góc và tường ngang

for (int col = 0; col < cols; col++) {

grid[col][row / 2] = new Cell(col, row / 2);

if (line.charAt(col \* 4 + 2) == '-') {

grid[col][row / 2].walls[0] = true; // Tường trên

}

}

} else {

// Dòng chứa các tường dọc và đường đi

for (int col = 0; col < cols; col++) {

if (grid[col][row / 2] == null) {

grid[col][row / 2] = new Cell(col, row / 2);

}

char c = line.charAt(col \* 4);

if (c == '|') {

grid[col][row / 2].walls[3] = true; // Tường trái

}

c = line.charAt(col \* 4 + 2);

if (c == 'S') start = grid[col][row / 2];

if (c == 'G') end = grid[col][row / 2];

}

}

row++;

}

br.close();

} catch (Exception e) {

println("Error loading maze: " + e.getMessage());

}

}

void solveMazeBFS() {

if (start == null || end == null) {

println("Start or End position not set correctly.");

return;

}

Queue<Cell> queue = new LinkedList<>();

HashMap<Cell, Cell> cameFrom = new HashMap<>();

queue.add(start);

cameFrom.put(start, null);

while (!queue.isEmpty()) {

Cell current = queue.poll();

if (current == end) break;

for (Cell neighbor : current.getValidNeighbors()) {

if (!cameFrom.containsKey(neighbor)) {

queue.add(neighbor);

cameFrom.put(neighbor, current);

}

}

}

Cell step = end;

while (step != null) {

path.add(step);

step = cameFrom.get(step);

}

}

class Cell {

int i, j;

boolean[] walls = {false, false, false, false}; // Top, Right, Bottom, Left

Cell(int i, int j) {

this.i = i;

this.j = j;

}

void show() {

int x = i \* cellSize;

int y = j \* cellSize;

stroke(0);

if (walls[0]) line(x, y, x + cellSize, y);

if (walls[1]) line(x + cellSize, y, x + cellSize, y + cellSize);

if (walls[2]) line(x, y + cellSize, x + cellSize, y + cellSize);

if (walls[3]) line(x, y, x, y + cellSize);

}

ArrayList<Cell> getValidNeighbors() {

ArrayList<Cell> neighbors = new ArrayList<>();

if (i > 0 && !walls[3]) neighbors.add(grid[i-1][j]);

if (i < cols-1 && !walls[1]) neighbors.add(grid[i+1][j]);

if (j > 0 && !walls[0]) neighbors.add(grid[i][j-1]);

if (j < rows-1 && !walls[2]) neighbors.add(grid[i][j+1]);

return neighbors;

}

}

#4 floodfill ok, load maze ok, chưa có animation

import java.util.\*;

import java.io.\*;

int cols = 16, rows = 16;

int cellSize = 40;

Cell[][] grid;

ArrayList<Cell> stack = new ArrayList<Cell>();

Cell current;

boolean solving = false;

ArrayList<Cell> path = new ArrayList<Cell>();

Cell start, end;

void setup() {

size(640, 640);

frameRate(30);

grid = new Cell[cols][rows];

selectMazeFile();

}

void draw() {

background(255);

for (int i = 0; i < cols; i++) {

for (int j = 0; j < rows; j++) {

if (grid[i][j] != null) {

grid[i][j].show();

}

}

}

if (!solving && start != null && end != null) {

floodFill();

solving = true;

}

if (start != null) {

fill(0, 255, 0);

rect(start.i \* cellSize, start.j \* cellSize, cellSize, cellSize);

}

if (end != null) {

fill(255, 0, 0);

rect(end.i \* cellSize, end.j \* cellSize, cellSize, cellSize);

}

for (Cell c : path) {

fill(0, 0, 255);

rect(c.i \* cellSize + cellSize / 4, c.j \* cellSize + cellSize / 4, cellSize / 2, cellSize / 2);

}

}

void selectMazeFile() {

selectInput("Select a maze file:", "fileSelected");

}

void fileSelected(File selection) {

if (selection != null) {

loadMaze(selection.getAbsolutePath());

} else {

println("No file selected.");

}

}

void loadMaze(String filename) {

try {

BufferedReader br = new BufferedReader(new FileReader(filename));

String line;

int row = 0;

while ((line = br.readLine()) != null && row < rows \* 2 + 1) {

if (row % 2 == 0) {

// Dòng chứa các góc và tường ngang

for (int col = 0; col < cols; col++) {

grid[col][row / 2] = new Cell(col, row / 2);

if (line.charAt(col \* 4 + 2) == '-') {

grid[col][row / 2].walls[0] = true; // Tường trên

}

}

} else {

// Dòng chứa các tường dọc và đường đi

for (int col = 0; col < cols; col++) {

if (grid[col][row / 2] == null) {

grid[col][row / 2] = new Cell(col, row / 2);

}

char c = line.charAt(col \* 4);

if (c == '|') {

grid[col][row / 2].walls[3] = true; // Tường trái

}

c = line.charAt(col \* 4 + 2);

if (c == 'S') start = grid[col][row / 2];

if (c == 'G') end = grid[col][row / 2];

}

}

row++;

}

br.close();

} catch (Exception e) {

println("Error loading maze: " + e.getMessage());

}

}

void floodFill() {

if (end == null) return;

Queue<Cell> queue = new LinkedList<>();

end.value = 0;

queue.add(end);

while (!queue.isEmpty()) {

Cell current = queue.poll();

for (Cell neighbor : current.getValidNeighbors()) {

if (neighbor.value == -1) {

neighbor.value = current.value + 1;

queue.add(neighbor);

}

}

}

}

class Cell {

int i, j;

boolean[] walls = {false, false, false, false}; // Top, Right, Bottom, Left

int value = -1;

Cell(int i, int j) {

this.i = i;

this.j = j;

}

void show() {

int x = i \* cellSize;

int y = j \* cellSize;

stroke(0);

if (walls[0]) line(x, y, x + cellSize, y);

if (walls[1]) line(x + cellSize, y, x + cellSize, y + cellSize);

if (walls[2]) line(x, y + cellSize, x + cellSize, y + cellSize);

if (walls[3]) line(x, y, x, y + cellSize);

if (value >= 0) {

fill(255, 255, 0);

textSize(12);

text(value, x + cellSize / 3, y + cellSize / 2);

}

}

ArrayList<Cell> getValidNeighbors() {

ArrayList<Cell> neighbors = new ArrayList<>();

if (i > 0 && !walls[3] && !grid[i-1][j].walls[1]) neighbors.add(grid[i-1][j]); // Left

if (i < cols-1 && !walls[1] && !grid[i+1][j].walls[3]) neighbors.add(grid[i+1][j]); // Right

if (j > 0 && !walls[0] && !grid[i][j-1].walls[2]) neighbors.add(grid[i][j-1]); // Top

if (j < rows-1 && !walls[2] && !grid[i][j+1].walls[0]) neighbors.add(grid[i][j+1]); // Bottom

return neighbors;

}

}