import tapplet.\*;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.util.\*;

import javax.swing.\*;

public class main extends TApplet{

// Program

static Random *rng* = new Random();

// Settings

static final int ***maxMapWidth*** = 15, ***maxMapLength*** = 15;

static final int ***cellSize*** = 50;

static final int ***edgeBufferX*** = 30, ***edgeBufferY*** = 60;

static final int ***newCellFreq*** = 45;// x\*100%

static int *minMapLen*;// Manhattan distance

static boolean *displayMap* = false;

static final double ***moveRate*** = 0.04;

static final int ***screenX*** = 400, ***screenY*** = 400;

static final int ***screenBounds*** = 30;

static final double ***mouseReduction*** = 6.5;

static final double ***FOV*** = 90;

static final double ***renderQuality*** = 30.0;

static final int ***fps*** = 60;

static int *maxBrightness*;

static final Color ***wallColor*** = Color.***white***;

static final Color ***winColor*** = Color.***green***;

// Data

static int *mapWidth*, *mapLen*;

static char[][] *grid* = new char[***maxMapLength***+5][***maxMapWidth***+5];

static int *sx*, *sy*, *ex*, *ey*;

static int[] *mx* = {1, -1, 0, 0}, *my* = {0, 0, 1, -1};

// Variable

static Coord *player* = new Coord(0, 0, null);

static double *pRot* = 0;// r%360+360

static ArrayList<Coord> *blocks* = new ArrayList<Coord>();

static long *startTime*;

static Coord *lastMouse* = new Coord(0, 0, null);

public static void main(String[] args) {

// Player Setup

JOptionPane.*showMessageDialog*(null, "In this 3D maze game, try to get to the end (green block) as soon as possible\nControls: WASD for movement, mouse for direction.");

*mapWidth* = Integer.*parseInt*(JOptionPane.*showInputDialog*("Map Width (max-"+***maxMapWidth***+"): ", 8));

*mapWidth* = Math.*min*(***maxMapWidth***, *mapWidth*);

*mapLen* = Integer.*parseInt*(JOptionPane.*showInputDialog*("Map Length (max-"+***maxMapLength***+"): ", 8));

*mapLen* = Math.*min*(***maxMapLength***, *mapLen*);

*minMapLen* = Math.*max*(*mapLen*, *mapWidth*);

// Map Setup

*generateNewMap*();

new main();

}

public void init() {

// System.out.println(winWidth+" "+winLen);

setCursor(Cursor.*getPredefinedCursor*(Cursor.***CROSSHAIR\_CURSOR***));

*maxBrightness* = 200;

setSize(***screenX***, ***screenY***);

Graphics g = getScreenBuffer();

// Maze Setup

setFPS(***fps***);

double incW = 1.0/***renderQuality***;

for (double idx = 0; idx <= 1; idx += incW) {

*blocks*.add(new Coord(*ex*, *ey*+idx, ***winColor***));

*blocks*.add(new Coord(*ex*+1, *ey*+idx, ***winColor***));

*blocks*.add(new Coord(*ex*+idx, *ey*, ***winColor***));

*blocks*.add(new Coord(*ex*+idx, *ey*+1, ***winColor***));

}

for (int i = 0; i < *mapLen*; i++) {

for (int j = 0; j < *mapWidth*; j++) {

if (*grid*[i][j] == '\*') {

double inc = 1.0/***renderQuality***;

if (i+1 >= *mapLen* || *grid*[i+1][j] != '\*') {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+1, j+idx, ***wallColor***));

}

}

if (i-1 <= 0 || *grid*[i-1][j] != '\*') {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i, j+idx, ***wallColor***));

}

}

if (j+1 >= *mapWidth* || *grid*[i][j+1] != '\*') {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+idx, j+1, ***wallColor***));

}

}

if (j-1 <= 0 || *grid*[i][j-1] != '\*') {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+idx, j, ***wallColor***));

}

}

} else {

double inc = 1.0/***renderQuality***;

if (i == 0) {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i, j+idx, ***wallColor***));

}

}

if (i == *mapLen*-1) {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+1, j+idx, ***wallColor***));

}

}

if (j == 0) {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+idx, j, ***wallColor***));

}

}

if (j == *mapWidth*-1) {

for (double idx = 0; idx <= 1; idx += inc) {

*blocks*.add(new Coord(i+idx, j+1, ***wallColor***));

}

}

}

}

}

*startTime* = System.*currentTimeMillis*();

// repaint();

}

public void movie(Graphics g) {

// Display

g.setColor(Color.***black***);

g.fillRect(0, 0, ***screenX***, ***screenY***);

// Rotation

try {

*mouseRot*();

} catch (Exception e) {};

// render

Comparator<Coord> cmp = (a, b) -> Double.*compare*(*dist*(*player*, b), *dist*(*player*, a));

Collections.*sort*(*blocks*, cmp);

drawSprites(*blocks*);

repaint();

// DEBUG

// System.out.println(mouseX());

}

/\*

\* Personal Methods

\*/

public void drawSprites(ArrayList<Coord> arr) {

Graphics g = getScreenBuffer();

for (int i = 0; i < arr.size(); i++) {

Coord t = arr.get(i);

// calc

double bRot = *pRot*+***FOV***/2;

double bD = *dist*(*player*, t);// radius of view circle

double bX = *player*.x-Math.*sin*(Math.*toRadians*(bRot))\*bD, bY = *player*.y-Math.*cos*(Math.*toRadians*(bRot))\*bD;

Coord b = new Coord(bX, bY, null);

double bA = *dist*(*player*, t), bB = *dist*(*player*, b), bC = *dist*(t, b);

double bDeg = Math.*acos*(-(bC\*bC-bA\*bA-bB\*bB)/(2.0\*bA\*bB));

double bZRot = 45-*pRot*;

double bOX = *player*.x-Math.*sin*(Math.*toRadians*(bZRot))\*bD, bOY = *player*.y+Math.*cos*(Math.*toRadians*(bZRot))\*bD;

Coord bO = new Coord(bOX, bOY, null);

double bZA = *dist*(*player*, t), bZB = *dist*(*player*, bO), bZC = *dist*(t, bO);

double bZDeg = Math.*acos*(-(bZC\*bZC-bZA\*bZA-bZB\*bZB)/(2.0\*bZA\*bZB));

double tmpDist = *dist*(*player*, t);

// Display

if (bDeg <= Math.*toRadians*(90) && bZDeg <= Math.*toRadians*(90)) {

int bCC = Math.*min*(*maxBrightness*, (int)(*maxBrightness*/bD/2.5));

Color tmpC = new Color(t.color.getRed()\*bCC/255, t.color.getGreen()\*bCC/255, t.color.getBlue()\*bCC/255);

drawRect(bDeg\*250, ***screenY***/2, 50/tmpDist\*6/***renderQuality***, 200/tmpDist, tmpC, true);

}

}

}

public void keyDown(KeyEvent e) {

char k = e.getKeyChar();

double px = *player*.x, py= *player*.y;

switch(k) {

case 'a':

px = *player*.x - Math.*sin*(Math.*toRadians*(*pRot*))\****moveRate***;

py = *player*.y - Math.*cos*(Math.*toRadians*(*pRot*))\****moveRate***;

break;

case 'd':

px = *player*.x + Math.*sin*(Math.*toRadians*(*pRot*))\****moveRate***;

py = *player*.y + Math.*cos*(Math.*toRadians*(*pRot*))\****moveRate***;

break;

case 'w':

px = *player*.x - Math.*cos*(Math.*toRadians*(*pRot*))\****moveRate***;

py = *player*.y + Math.*sin*(Math.*toRadians*(*pRot*))\****moveRate***;

break;

case 's':

px = *player*.x + Math.*cos*(Math.*toRadians*(*pRot*))\****moveRate***;

py = *player*.y - Math.*sin*(Math.*toRadians*(*pRot*))\****moveRate***;

break;

case ' ':

System.*exit*(0);

}

*player*.x = px;

*player*.y = py;

*player*.x = Math.*max*(0.0, *player*.x);

*player*.x = Math.*min*(*mapLen*, *player*.x);

*player*.y = Math.*max*(0.0, *player*.y);

*player*.y = Math.*min*(*mapWidth*, *player*.y);

// Win

if ((int)*player*.x == *ex* && (int)*player*.y == *ey*) {

JOptionPane.*showMessageDialog*(null, "Win!\nTime (s): "+(double)((System.*currentTimeMillis*()-*startTime*)/10)/100);

System.*exit*(0);

}

}

static void mouseRot() throws Exception {

Robot robot = new Robot();

*pRot* += (*mouseX*()-*lastMouse*.x)/***mouseReduction***;

if (*mouseX*() < ***screenBounds***) robot.mouseMove(***screenBounds***, *mouseY*());

else if (*mouseX*() > ***screenX***-***screenBounds***) robot.mouseMove(***screenX***-***screenBounds***, *mouseY*());

if (*mouseY*() < ***screenBounds***) robot.mouseMove(*mouseX*(), ***screenBounds***);

else if (*mouseY*() > ***screenY***-***screenBounds***) robot.mouseMove(*mouseX*(), ***screenY***-***screenBounds***);

*lastMouse* = new Coord(*mouseX*(), *mouseY*(), null);

}

static double rotConv(double r) {

return r%360+360;

}

static double dist(Coord a, Coord b) {

double xDist = Math.*abs*(a.x-b.x), yDist = Math.*abs*(a.y-b.y);

return Math.*sqrt*(xDist\*xDist + yDist\*yDist);

}

void drawRect(double cx, double cy, double w, double h, Color c, boolean fill) {

// Setup

Graphics g = getScreenBuffer();

g.setColor(c);

int dx = (int)Math.*round*(cx-w/2), dy = (int)Math.*round*(cy-h/2);

// Draw

if (fill) {// fillRect

g.fillRect(dx, dy, (int)w, (int)h);

} else {// drawRect (hollow)

g.drawRect(dx, dy, (int)w, (int)h);

}

}

static void generateNewMap() {

// select start & end cell

do {

do {

*sx* = *rng*.nextInt(*mapLen*);

*sy* = *rng*.nextInt(*mapWidth*);

} while ((*sx* != 0 && *sx* != *mapWidth*-1) && (*sy* != 0 && *sy* != *mapLen*-1));

do {

*ex* = *rng*.nextInt(*mapLen*);

*ey* = *rng*.nextInt(*mapWidth*);

} while ((*ex* != 0 && *ex* != *mapWidth*-1) && (*ey* != 0 && *ey* != *mapLen*-1));

} while (*manhattanDist*(*sx*, *sy*, *ex*, *ey*) < *minMapLen*);

// generate

do {

// empty grid

for (char[] row : *grid*) {

Arrays.*fill*(row, '\0');

}

*grid*[*sx*][*sy*] = 's'; *grid*[*ex*][*ey*] = 'e';

for (int i = 0; i < *mapLen*; i++) {

for (int j = 0; j < *mapWidth*; j++) {

if (*grid*[i][j] == '\0') {// unfilled

if (*rng*.nextInt(100) < ***newCellFreq***) {// empty

*grid*[i][j] = ' ';

} else {// wall

*grid*[i][j] = '\*';

}

}

}

}

} while (!*bfs*());

*player* = new Coord((double)*sx*+0.5, (double)*sy*+0.5, null);

// DEBUG

if (*displayMap*) {

for (int i = 0; i < *mapLen*; i++) {

for (int j = 0; j < *mapWidth*; j++) {

System.***out***.print(*grid*[i][j]);

}

System.***out***.println();

}

}

}

static boolean bfs() {

boolean[][] vis = new boolean[***maxMapLength***+5][***maxMapWidth***+5];

Queue<int[]> q = new ArrayDeque<int[]>();

q.add(new int[] {*sx*, *sy*}); vis[*sx*][*sy*] = true;

while (!q.isEmpty()) {

int[] cur = q.poll();

for (int i = 0; i < 4; i++) {

int dx = cur[0]+*mx*[i], dy = cur[1]+*my*[i];

if (dx == *ex* && dy == *ey*) return true;

if (dx >= 0 && dx < *mapLen* && dy >= 0 && dy < *mapWidth* && vis[dx][dy] == false && *grid*[dx][dy] == ' ') {

vis[dx][dy] = true;

q.add(new int[] {dx, dy});

}

}

}

return false;

}

static int manhattanDist(int a, int b, int c, int d) {

return Math.*abs*(a-c)+Math.*abs*(b-d);

}

static int mouseX() {

return (int)MouseInfo.*getPointerInfo*().getLocation().getX();

}

static int mouseY() {

return (int)MouseInfo.*getPointerInfo*().getLocation().getY();

}

}

public class Coord {

double x, y;

Color color;

public Coord(double x, double y, Color color) {

this.x = x;

this.y = y;

this.color = color;

}

}