DESIGN AND ANALYS OF ALGORITHMS

QUIZ 2

KELAS: E

Minesweeper

Kelompok E-07

05111840000013	Clement Prolifel Priyatama
05111840000057	Maisie Chiara Salsabila
05111840000100	Abdur Rohman

Lecturer:

MM Irfan Subakti

Department of Informatics

Faculty of Information and Communication Institut Teknologi Sepuluh Nopember (ITS)

Surabaya 2020

ALGORITHM

1. Graph

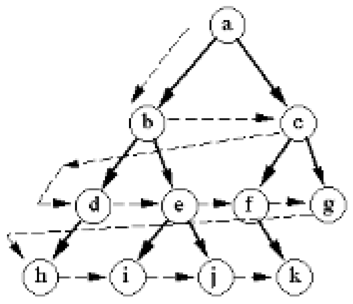
Graphs, one of prime subject in discrete mathematics, have many applications and implementations in computer science world. Graphs can be used to model relation, network, position, adjacency and many else. In game programming, graphs are extensively used, for example to model tiles, object position and image representation. Since video games heavily rely on graphs, graph search algorithms are necessarily needed. There are numerous graph search algorithms; the most basicones are DFS (Depth-First Search) and BFS (Breadth-First Search) algorithm.

2. BFS (Breadth-First Search)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

It uses the opposite strategy as depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes.

For example, depth n, before visit nodes in depth n+1. BFS node-visiting process can be visualized as:



Suppose the process starts from node a. Then, it will respectively visit b, c, d, e, f, g, h, i, j, and k. BFS can be defined as follows:

```
procedure BFS(G, start_v) is
    let Q be a queue
    label start_v as discovered
    Q.enqueue(start_v)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as discovered then
                label w as discovered
                w.parent := v
                Q.enqueue(w)
```

3. Minesweeper

Minesweeper is a single-player puzzle video game. The objective of the game is to clear a rectangular board containing hidden "mines" or bombs without detonating any of them, with help from clues about the number of neighboring mines in each field. The game originates from the 1960s, and has been written for many computing platforms in use today. It has many variations and offshoots.

Minesweeper has a very basic gameplay style. In its original form, mines are scattered throughout a board. This board is divided into cells, which have three states: uncovered, covered and flagged. A covered cell is blank and clickable, while an uncovered cell is exposed, either containing a number (the mines adjacent to it), or a mine. When a cell is uncovered by a player click, and if it bears a mine, the game ends. A flagged cell is similar to a covered one, in the way that mines are not triggered when a cell is flagged, and it is impossible to lose through the action of flagging a cell. However, flagging a cell implies that a player thinks there is a mine underneath, which causes the game to deduct an available mine from the display.

In order to win the game, players must logically deduce where mines exist through the use of the numbers given by uncovered cells. To win, all non-mine cells must be uncovered and all mine cells must be flagged. At this stage, the timer is stopped.

When a player left-clicks on a cell, the game will uncover it. If there are no mines adjacent to that particular cell, the mine will display a blank tile or a "0", and all adjacent cells will automatically be uncovered. Right-clicking on a cell will flag it, causing a flag to appear on it. Note that flagged cells are still covered, and a player can click on it to uncover it, like a normal covered cell.

The main objective of the algorithm is to visit all empty tiles and open it. If it encounters a numbered tile, it opensthe tile but not looks further. Since the main objective is to visit all tiles (or, in graph theory term, nodes), both DFS and BFS can be implemented in this problem. But we are going to use BFS to implement this case.

Implementation

Suppose there are these functions and variables in the game code:

- Tiles (variable)

  A matrix that resembles minesweeper game board. It may has three kind of value : bomb, number and empty. Tiles[i,j] means tiles at column i row j.
- Mark(procedure)

  Marks Tiles[i,j] as visited / unvisited
- Open(procedure)

  Opens Tiles[i,j]

CODE EXPLANATION

Assets

1. Assets.java

```
package com.cma.minesweeper.assets;

import java.awt.Graphics;

public class Assets {
	public static final int spriteWidth=16;
	public static BufferedImage covered;
	public static BufferedImage[] uncovered;
	public static BufferedImage mine,flag;
	public static BufferedImage bomb,wrongFlag;
```

Assets class is a class for initiating all the assets and processing pictures.

```
	public static void init() {
		BufferedImageLoader loader = new BufferedImageLoader();
		SpriteSheets sheet = null;
		try {
			sheet = new 
SpriteSheets(loader.loadImage("res/predatorskin.bmp"));
		} catch (IOException e) {
			e.printStackTrace();
		}

		uncovered= new BufferedImage[9];
		for(int i=0; i<uncovered.length;i++) {
			uncovered[i]= sheet.crop(0,i,spriteWidth);
		}
		covered = sheet.crop(1, 0, spriteWidth);
		mine = sheet.crop(1, 2, spriteWidth);
		flag = sheet.crop(1, 3, spriteWidth);
		wrongFlag = sheet.crop(1, 4, spriteWidth);
		bomb = sheet.crop(1, 5, spriteWidth);
	}
```

This function is a function for iniating pictures. Function will call BufferdImageLoader first to loads the image and then crop the image.

```
	public static void draw(int row, int col, CellStates state,
	Graphics g) {
		BufferedImage img = covered;

		switch (state) {
		case COVERED:
			img = covered;
			break;
		case FLAGGED:
			img = flag;
			break;
		case UNC0:
			img = uncovered[0];
			break;
		case UNC1:
			img = uncovered[1];
			break;
		case UNC2:
			img = uncovered[2];
			break;
		case UNC3:
			img = uncovered[3];
			break;
		case UNC4:
			img = uncovered[4];
			break;
		case UNC5:
			img = uncovered[5];
			break;
		case UNC6:
			img = uncovered[6];
			break;
		case UNC7:
			img = uncovered[7];
			break;
		case UNC8:
			img = uncovered[8];
			break;
		case WRONG_FLAG:
			img = wrongFlag;
			break;
		case MINE:
			img = mine;
			break;
		case FIRED_MINE:
			img = bomb;
			break;
		}

		g.drawImage(img, spriteWidth * col, spriteWidth * row, null);
	}

}
```

This is a function initiating asset and drawing pictures. First of all, function will decide the image based on CellStates, and then draws the image.

2. BufferedImageLoader.java

```java
package com.cma.minesweeper.assets;
import java.awt.image.BufferedImage;

public class BufferedImageLoader {
	private BufferedImage image;
	public BufferedImage loadImage(String path) throws IOException{
		try {
			image = ImageIO.read(getClass().getResource(path));
			return image;
		}catch(IOException e) {
			e.printStackTrace();
			System.exit(1);
		}
		return null;
	}
```

BufferedImageLoader is a function for loading the image from the system through the relative path given and returns a Buffered Image.

3. SpriteSheets.java

```java
package com.cma.minesweeper.assets;
import java.awt.image.BufferedImage;

public class SpriteSheets {


	private BufferedImage sheet;

	public SpriteSheets(BufferedImage sheet) {
		this.sheet = sheet;
	}

	public BufferedImage crop(int row, int col, int width) {
		int y = width * row;
		int x = width * col;
		return sheet.getSubimage(x, y, width, width);
	}
}
```

SpriteSheets is a function for cropping a sprite sheet.

Game

1. Board.java

```
package com.cms.minespweeper.game;

import java.awt.Graphics;

public class Board {
	private GameStates gameStates;
	private int n,nmines,ncovered;
```

n is a variable for n size of cell, nmines is a variable for the nmines number of mines, ncovered is a variable for the number of covered cells.

```
	private final int[] di = new int[] { -1, -1, -1, 0, 1, 1,  1,  0 };
	private final int[] dj = new int[] { -1,  0,  1, 1, 1, 0, -1, -1 };
```

Those are help array to do BFS in 8 directions.

```
	private final CellStates[] uncoveredStates = new CellStates[] {
			CellStates.UNC0, CellStates.UNC1, CellStates.UNC2, 
CellStates.UNC3,
			CellStates.UNC4, CellStates.UNC5, CellStates.UNC6, 
CellStates.UNC7, CellStates.UNC8
	};
```

The uncoveredStates array contains a State based on the number of mines.

```
	private boolean[][] isMine;
	private int[][] mineCnt;
	private CellStates[][] states;
```

isMine is 2D array for checking whether cell is mine. mineCnt is 2D array to store the number of count mines in each cell. states is 2D array to store the states of each cell.

```
	public Board(int ncells, int nmines) {
		this.n=ncells;
		this.nmines=nmines;
		this.ncovered= ncells*ncells;
		isMine = new boolean[n][n];
		mineCnt = new int[n + 2][n + 2];
		states = new CellStates[n][n];
		generatedMines();
		for(int i=0;i<n;i++) {
			Arrays.fill(states[i],CellStates.COVERED);
		}
		gameStates=GameStates.ONGOING;

	}
```

Board function is a function for auto-generating constructor stub. Function will initiate the size of array and then make the mine randomly. After that, cell will be initiated and all the cell states will be covered by looping.

```
	public void generatedMines() {
		Random rand= new Random();
		int mines=nmines;
		while(mines-- >0) {
			int position= rand.nextInt(ncovered);
			int x= position % n;
			int y= position / n;
			//if already filled
			if(isMine[y][x]) {
				mines++;
			}else {
				isMine[y][x]=true;
				for (int d = 0; d < di.length; d++) {
					//increase count adjensi cell
					mineCnt[y + di[d] + 1][x + dj[d] + 1]++;
				}
			}
		}
	}
```

GeneratedMines is a function to place the mines randomly. First, mines will be tracked to know how many mines to placed. Random the place of mines by looping. If there is mines in the position (isMine[y][x]) then add back the number of mines. Otherwise isMine array becomes true and will loop to increase the count cells around the mines.

```
	public  void toogleFlag(int row,int col,Graphics g) {
		if(states[row][col]==CellStates.COVERED) {
			states[row][col]=CellStates.FLAGGED;
		}
		else if(states[row][col]==CellStates.FLAGGED) {
			states[row][col]=CellStates.COVERED;
		}
		Assets.draw(row, col, states[row][col], g);
		g.dispose();
		
	}
```

toogleFlag is function to make flag. If the cell that right-clicked is covered (there are no flags), then add the flag. If there is already a flag then remove the flag.

```
	private void bfs(int row, int col, Graphics g) {
		Queue<Integer> q = new ArrayDeque<>();
		Set<Integer> visited = new HashSet<>();
		
		ncovered++;
		q.add(row * n + col);
		visited.add(row * n + col);
		
		while (!q.isEmpty()) {
			int r = q.peek() / n;
			int c = q.poll() % n;
			
			if (states[r][c] != CellStates.COVERED)
				continue;
			
			states[r][c] = uncoveredStates[mineCnt[r + 1][c + 1]];
			Assets.draw(r, c, states[r][c], g);
			ncovered--;
			
			if (states[r][c] != CellStates.UNC0)
				continue;
			
			for (int i = 0; i < di.length; i++) {
				int _r = r + di[i];
				int _c = c + dj[i];
				int key = _r * n + _c;
				if (_r < 0 || _r >= n || _c < 0 || _c >= n ||
visited.contains(key))
					continue;
				q.add(key);
				visited.add(key);
			}
		}
		
		if (ncovered == nmines)
			gameStates = GameStates.WON;
	}
```

Function above is implementation of BFS by using queue. BFS in minesweeper can be described as a graph where each node has 8 node adjuncts. We use help array di and dj to trace the 8 nodes. In java we use array dequeue to get a better speed and the element of the queue used is an integer which is the cell sequence, add cell to the queue by returning to the initial integer value created in generatedMines function, and mark cell as visited. Then check the generated mines method.

Do looping as long as the queue isn’t empty. Peek (q.peek()) is for getting the top value and poll (q.poll()) is for retrieving and removing. Check if the cell is not covered then skip the cell to continue the next iteration. Get the states cell states which is array unCoveredStates based on the number of mines that around the cell that were stored in mineCnt previously. Check if the cell has bomb around then it skip to the next iteration. If cell doesn’t have bomb around, do a check cells around then store the value into queue. We use the value of array di and array dj (array dj and di represent 8 direction surrounding the cell) to get into the surrounding cell.

```
	public boolean uncoverCell(int row,int col,Graphics g) {
		if(states[row][col]!=CellStates.COVERED) {
			return false;
		}
		if(isMine[row][col]) {
			gameStates=GameStates.LOST;
			uncoverAll(g,false);
			states[row][col]=CellStates.FIRED_MINE;
			Assets.draw(row, col, CellStates.FIRED_MINE, g);
		}
		else {
			ncovered--;

			Assets.draw(row, col, uncoveredStates[mineCnt[row + 
1][col + 1]], g);

			if(nmines==ncovered) {
				gameStates=GameStates.WON;
				uncoverAll(g,true);
			}
			else {
				bfs(row, col, g);
			}

		}
		g.dispose();
		return true;

	}
```

uncoverCell is an execution function to change cell condition when there is a left-clicked input. If the cell is uncovered then return false. If the cell is mines then the game is finished and set the gameStatus as “LOST” and the uncover all cells. Otherwise do BFS.

```
	private void uncoverAll(Graphics g, boolean isWin) {
		// TODO Auto-generated method stub
		for(int i=0;i<n;i++) {
			for(int j=0;j<n;j++) {

				if(states[i][j]==CellStates.COVERED && 
isMine[i][j]) {
					states[i][j] = isWin ? CellStates.FLAGGED 
: CellStates.MINE;
					Assets.draw(i, j, states[i][j], g);
				}else if(states[i][j]==CellStates.FLAGGED && 
!isMine[i][j]) {
					states[i][j] = CellStates.WRONG_FLAG;
					Assets.draw(i, j, states[i][j], g);
				}

			}
		}

	}
	public GameStates getGameStates() {
		return gameStates;
	}


}
```

Function for uncovering all the cells when you hit the bomb or when you are winning the game.

2. CellStates.java

```
package com.cms.minespweeper.game;

public enum CellStates {
	COVERED, FLAGGED, UNC0, UNC1, UNC2, UNC3, UNC4, UNC5, UNC6, UNC7, 
UNC8, WRONG_FLAG, MINE, FIRED_MINE
}
```

3. Frame.java

```java
package com.cms.minespweeper.game;
import java.awt.Canvas;

public class Frame {

	private JFrame frame;
	private Canvas canvas;

	private String title;
	private int width;
	public Frame(JFrame frame, Canvas canvas) {
		super();
		this.frame = frame;
		this.canvas = canvas;
	}

	private int height;
```

```java
	private void setFrame() {
		frame= new JFrame(title);
		canvas= new Canvas();
		frame.setSize(width, height);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setResizable(false);
		frame.setLocationRelativeTo(null);
		frame.setVisible(true);

		canvas.setPreferredSize(new Dimension(width,height));
		canvas.setMaximumSize(new Dimension(width,height));
		canvas.setMinimumSize(new Dimension(width,height));
		canvas.setFocusable(false);

		frame.add(canvas);
		frame.pack();
	}
```

```java
	public Frame (String title,int w,int h) {
		this.title=title;
		this.width=w;
		this.height=h;
		setFrame();
	}

	public JFrame getFrame() {
		return frame;
	}

	public Canvas getCanvas() {
		return canvas;
	}
}
```

4. Game.java

```
package com.cms.minespweeper.game;

import java.awt.Graphics;

public class Game {
	Frame frame;
	private int width,height;
	private String title;
	private int N;
	private Board board;
	private MouseEvents mouseEvent;
	private BufferStrategy bs;

	private boolean ended;
```

```
	public Game(String title, int ncells, int nmines) {
		// TODO Auto-generated constructor stub
		this.N=ncells;
		width=Assets.spriteWidth * ncells;
		height=width;
		this.title=title;
		board= new Board(ncells,nmines);
		mouseEvent= new MouseEvents(this);

		frame = new Frame(title, width, height);
		frame.getFrame().addMouseListener(mouseEvent);
		frame.getCanvas().addMouseListener(mouseEvent);
		frame.getCanvas().createBufferStrategy(2);
		bs= frame.getCanvas().getBufferStrategy();
		Assets.init();

	}
```

```
	public void start() {
		Graphics g =bs.getDrawGraphics();
		for(int i=0;i<N;i++) {
			for(int j=0;j<N;j++) {
				Assets.draw(i, j, CellStates.COVERED, g);
			}
		}
		bs.show();
		g.dispose();
	}
```

Start function is a function to start the game. Loop assets.draw to draw all the cells in covered condition.

```
	public void onClick(boolean isLeft, int x, int y) {
		// TODO Auto-generated method stub
		if(ended)return;
		Graphics g = bs.getDrawGraphics();
		int row=y/Assets.spriteWidth;
		int col=x/Assets.spriteWidth;
		if(isLeft) {
			board.uncoverCell(row, col, g);
		}else {
			board.toogleFlag(row, col, g);
		}
		bs.show();
		GameStates result= board.getGameStates();
		// when game ends
		if (result != GameStates.ONGOING) {
			ended = true;
			System.out.println("Game ended!");
			String msg = (result == GameStates.LOST ? "!!!!! You 
Lose !!!!!" : "!!!!! You Won !!!!!");
			frame.getFrame().setTitle(msg);
		}
		g.dispose();
	}


}
```

First get the row (row) and col (column) by dividing coordinates by sprite size. If the left mouse is released then uncover the cell. if the right mouse is released then flag the cell. Check the game status every click. If the game status is not “ongoing” then the game is finished.

5. GameStates.java

```
package com.cms.minespweeper.game;

public enum GameStates {
	ONGOING, LOST, WON
}
```

GameStates function to determine the game status is “ongoing”, “lost”, or “won”.

6. Main.java

```
package com.cms.minespweeper.game;

public class Main {

	public static void main(String[] args) {
		int ncells=40;
		int nmines=10;

		Game game = new Game("Minesweeper BFS",ncells,nmines);
		game.start();

	}

}
```

This is the main function. ncells is the number of squares or cells each side of the board (ncells x ncells). nmines is the number of mines. In this function, we call the Game function to initiate and play the game.

7. MouseEvents.java

```
package com.cms.minespweeper.game;
import java.awt.event.MouseEvent;

public class MouseEvents implements MouseListener{
	private Game game;

	public MouseEvents(Game game) {
		this.game=game;
	}

	@Override
	public void mouseClicked(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseEntered(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseExited(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mousePressed(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}
```

MouseListener is a function to accept the mouse input.

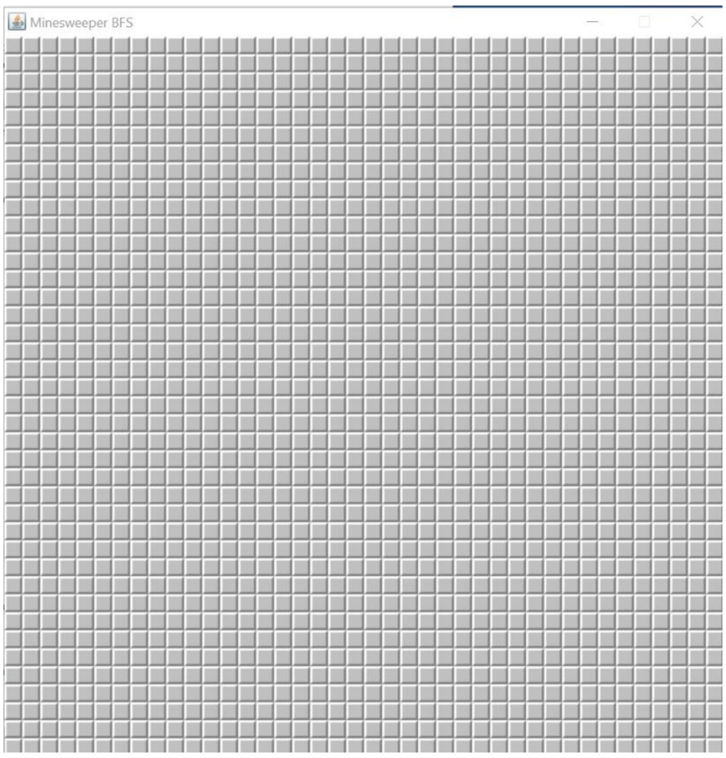
```
        @Override
        public void mouseReleased(MouseEvent e) {
                // TODO Auto-generated method stub
                boolean isLeft = (e.getButton() == MouseEvent.BUTTON1);
                if (game != null)
                        game.onClick(isLeft, e.getX(), e.getY());
        }

}
```

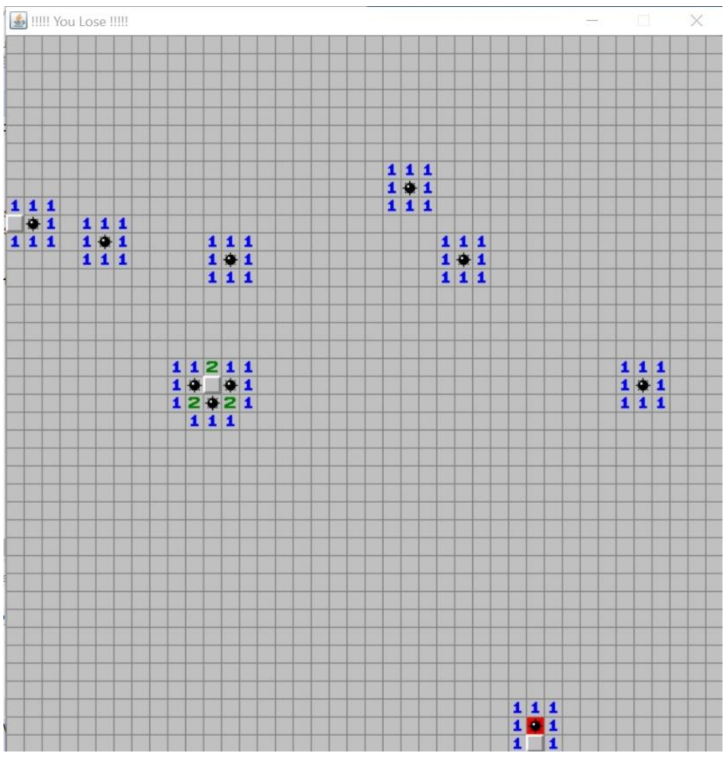
When a mouse is release, check which mouse is, left or right. Then if the game is still running, get the x and y position where the mouse was released then the onClick function in the game will be called.

Output

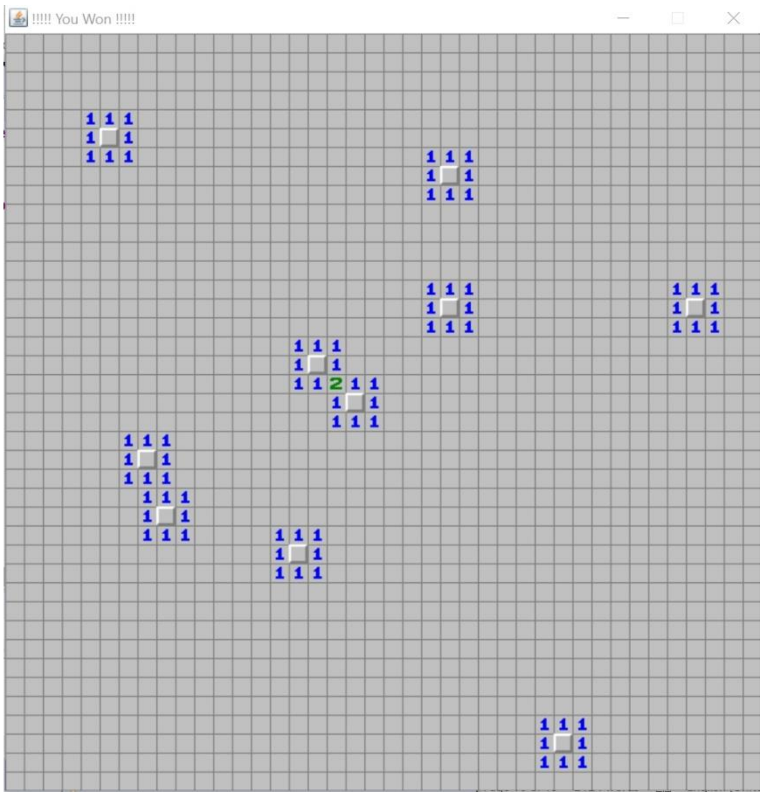
1. **The main view of the game**



2. **If you lose**



3. **If you won**



Source Code

https://github.com/abdurrohman100/DAA-E_QUIZ2.git

DECLARATION

“By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved quiz 2 by myself, didn’t do any cheating by any means, didn’t do any plagiarism, and didn’t accept anybody’s help by any means. I am going to accept all of the consequences by any means if it has proven that I have been done any cheating and/or plagiarism.”

Surabaya, 25 Maret 2020

[illegible]	[illegible]	[illegible]
Clement Prolifel P.	Maisie Chiara Salsabila	Abdur Rohman
05111840000013	05111840000057	05111840000100

REFERENCES

https://en.wikipedia.org/wiki/Breadth-first_search

https://en.wikipedia.org/wiki/Minesweeper_(video_game)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2012-2013/Makalah2012/Makalah-IF2091-2012-069.pdf