**Content**

[1.0 UML diagrams 2](#_Toc533551471)

[1.1 Use case diagrams 2](#_Toc533551472)

[1.2 Class diagram 3](#_Toc533551473)

[1.3Sequence diagram 3](#_Toc533551474)

[2.0 Functional and Non-functional Requirements 4](#_Toc533551475)

[2.1 Functional requirements 4](#_Toc533551476)

[2.2 Non-Functional requirements 4](#_Toc533551477)

[3.0 UI design 5](#_Toc533551478)

[3.1 Main panel design 5](#_Toc533551479)

[3.1.1 First level map 5](#_Toc533551480)

[3.1.2 Second level map 6](#_Toc533551481)

[3.1.3 Third gate map 7](#_Toc533551482)

[3.1.4 Fourth gate map 8](#_Toc533551483)

[3.1.5 Fifth gate map 9](#_Toc533551484)

[3.2 Map pictures description 10](#_Toc533551485)

[3.2.1 Grass 10](#_Toc533551486)

[3.2.2 Wall 10](#_Toc533551487)

[3.2.3 Floor 10](#_Toc533551488)

[3.2.4 Hole 10](#_Toc533551489)

[3.2.5 Box (normal and in the hole) 10](#_Toc533551490)

[3.2.6 Warehouse keeper(up/down/left/right) 11](#_Toc533551491)

[4.0 Code 12](#_Toc533551492)

[4.1 mainmenu 12](#_Toc533551493)

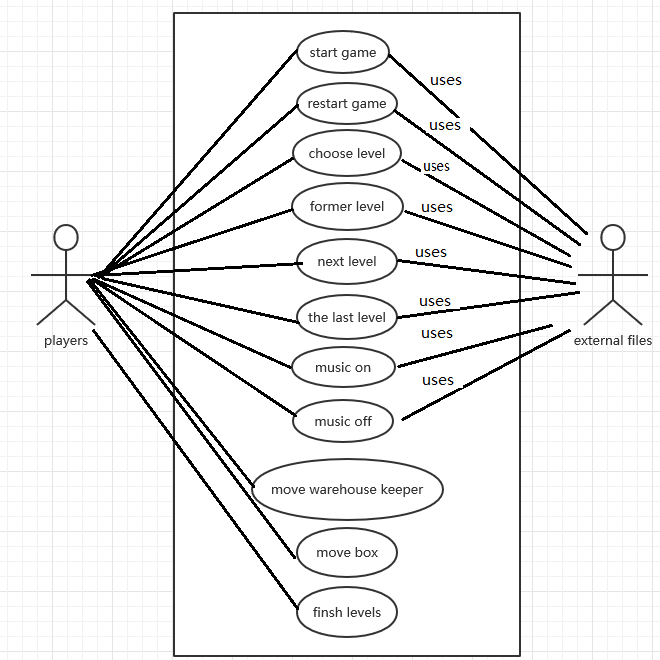
[4.2 MainPanel 18](#_Toc533551494)

[4.3 Music 29](#_Toc533551495)

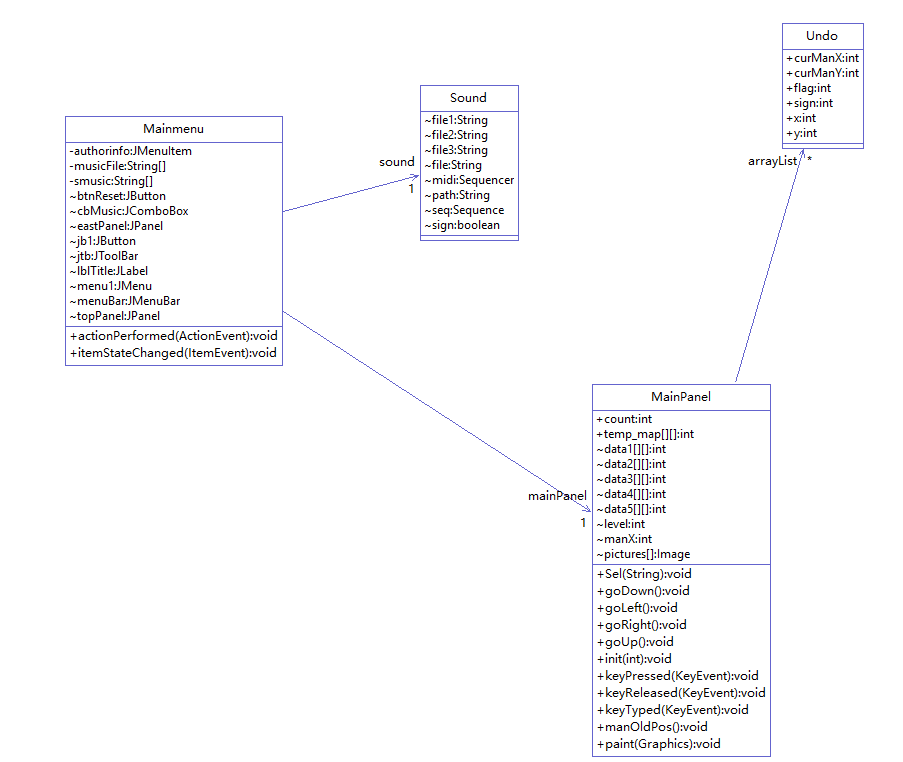
[4.4 Undo 30](#_Toc533551496)

# 1.0 UML diagrams

## Use case diagrams

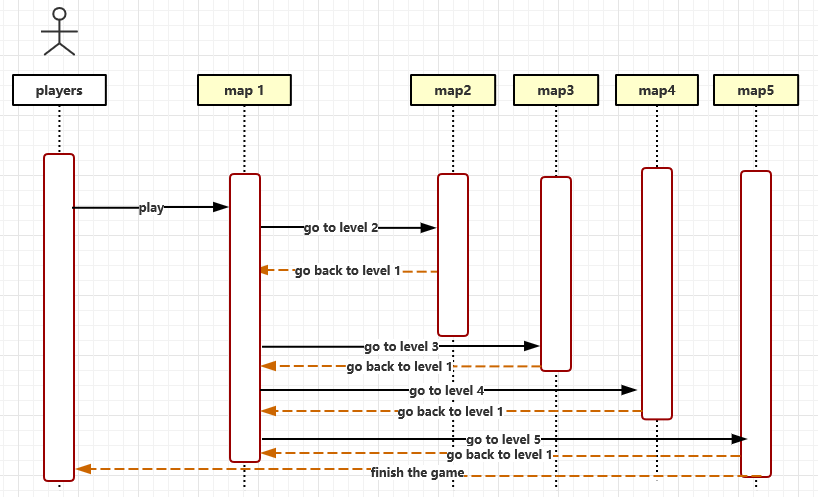


## 1.2 Class diagram



## 1.3Sequence diagram

Choose map sequence diagram:



# 2.0 Functional and Non-functional Requirements

## 2.1 Functional requirements

**2.1.1 Start the game**

With the restart button, Click to reload the current map and start the game With the return button, Click to go back to the previous step

There is a statistic of the number of steps the current game has taken, step backward will also be shown at the same time step backward. When there is a wall behind it, the number of steps will not increase.

2.1.2 Gate Settings

The current number of checkpoints is marked.

You can choose the level freely. You can enter the level you want to enter by clicking the level selection button and entering the number of existing levels.

You can go straight to the last and first levels.

There is a button to return to the previous level. Click to restart the previous level. When the player successfully clears the customs, he can enter the next level automatically. After clearing the customs, he will not because there is no next level. When the player does not play the current level, you can click help to get help information.

**2.1.3 music**

Music can be turned on and off.

Players can also choose the music they want to listen to in the catalog through the drop-down box control.

## 2.2 Non-Functional requirements

**2.2.1 Hardware Minimum Requirements**

It's a computer.

The operating system is Windows Win7, 8, 10.

CPU at least i5

Memory 1G

**2.2.2 Other requirements**

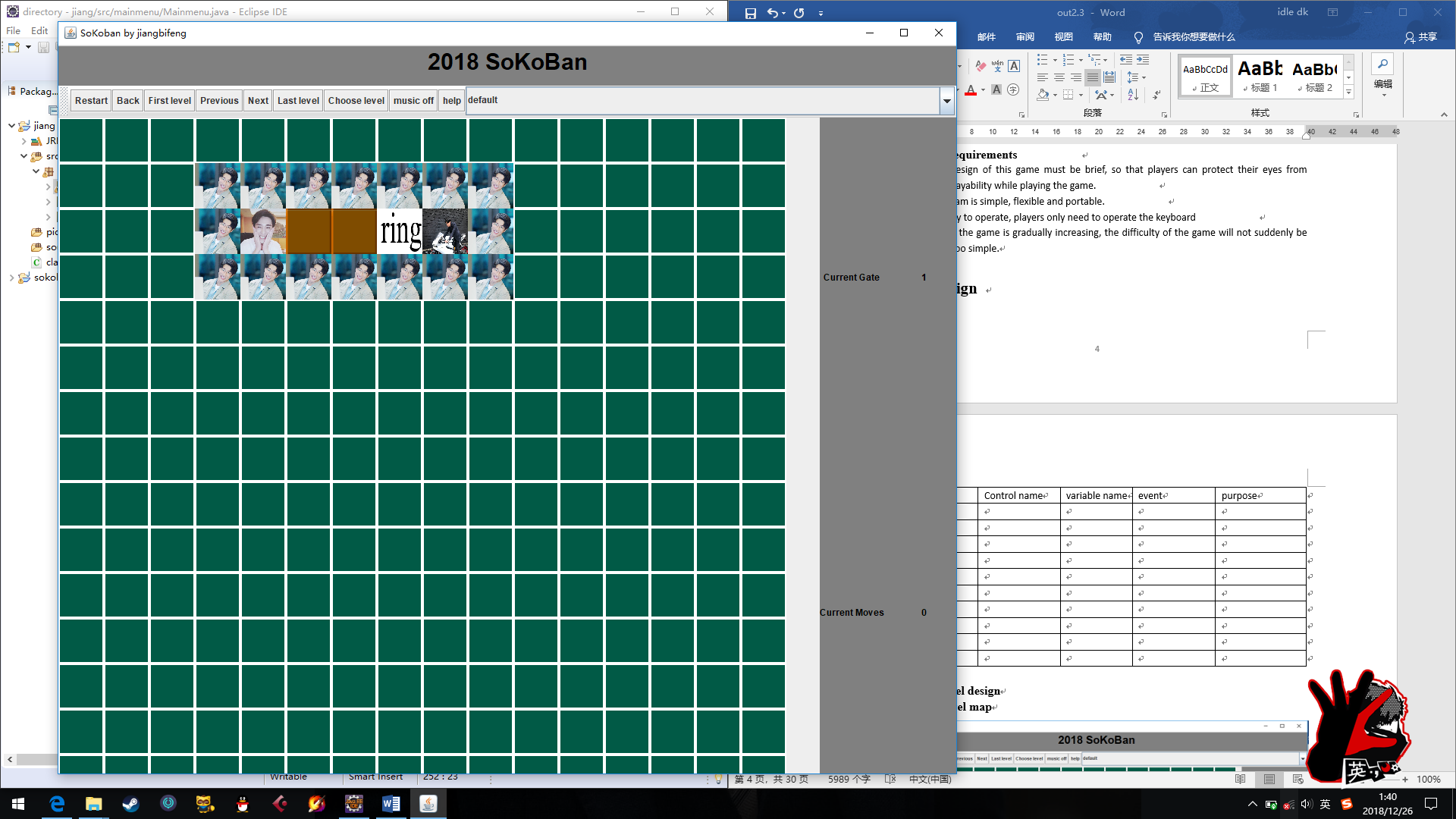
The interface design of this game must be brief, so that players can protect their eyes from tiredness and playability while playing the game.

The game program is simple, flexible and portable.

This game is easy to operate, players only need to operate the keyboard

The difficulty of the game is gradually increasing, the difficulty of the game will not suddenly be too difficult or too simple.

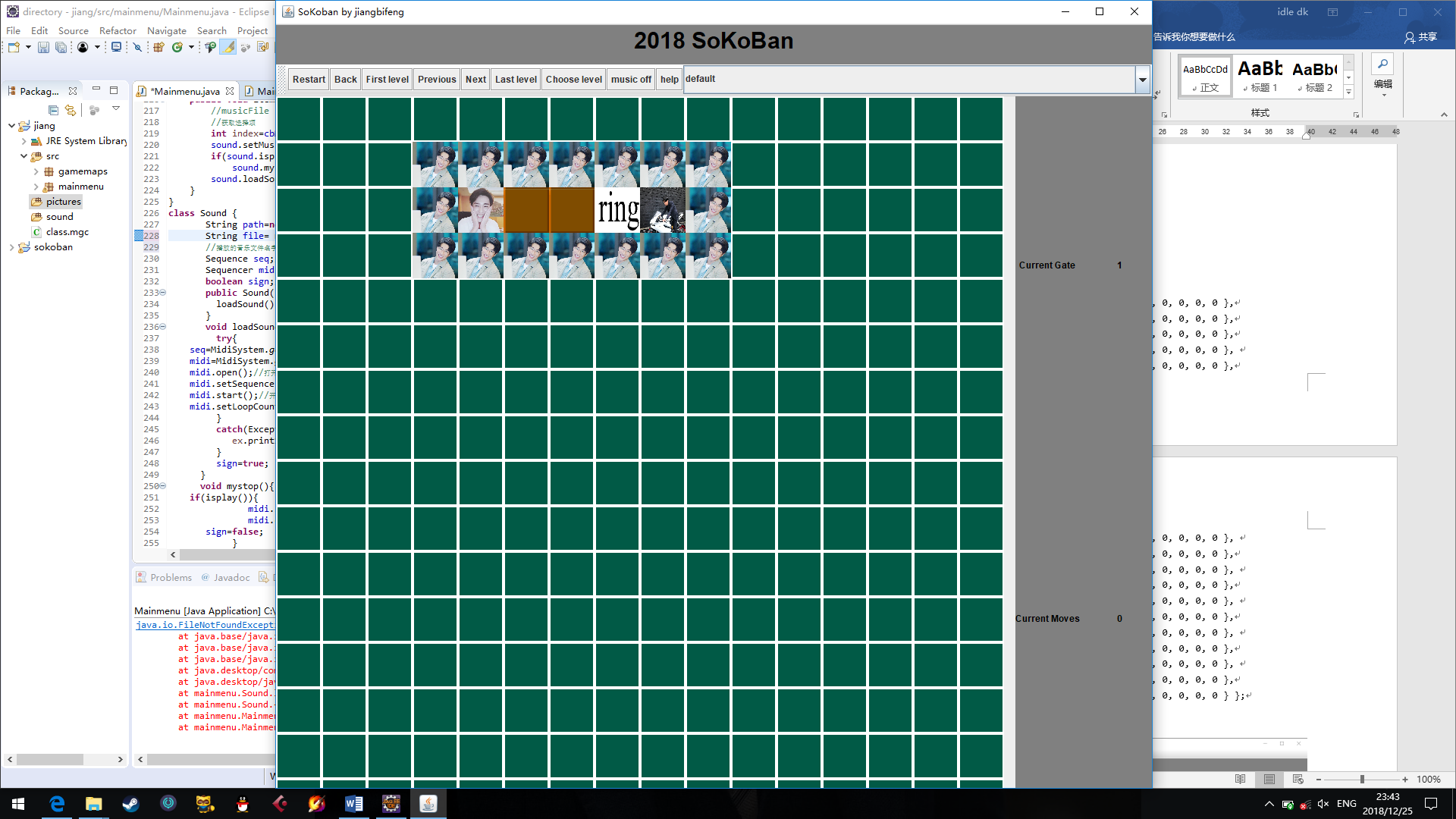
# 3.0 UI design



|  |  |  |  |
| --- | --- | --- | --- |
| Control name | variable name | event | purpose |
| Restart | btnReset | button | Restart current level |
| Back | btnBack | button | Back to last step |
| First level | btnFirst | button | Go to first level |
| Pervious | btnPrev | button | Go to former level |
| Next | btnNext | button | Go to next level |
| Last level | btnLast | button | Go to last level |
| Choose level | btnSelect | button | Choose level |
| music off/on | btnMusic | button | Turn on/off music |
| help | btnHelp | button | Give players help information |
| default | cbMusic | Drop down | Choose music from menu |
| Current Moves | lblStepTitle | label | Show current move steps |
| Current Gate | lblGateTitle | label | Show current level |

## 3.1 Main panel design

### 3.1.1 First level map



array:

**final** **int** data1[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 3, 2, 2, 5, 6, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

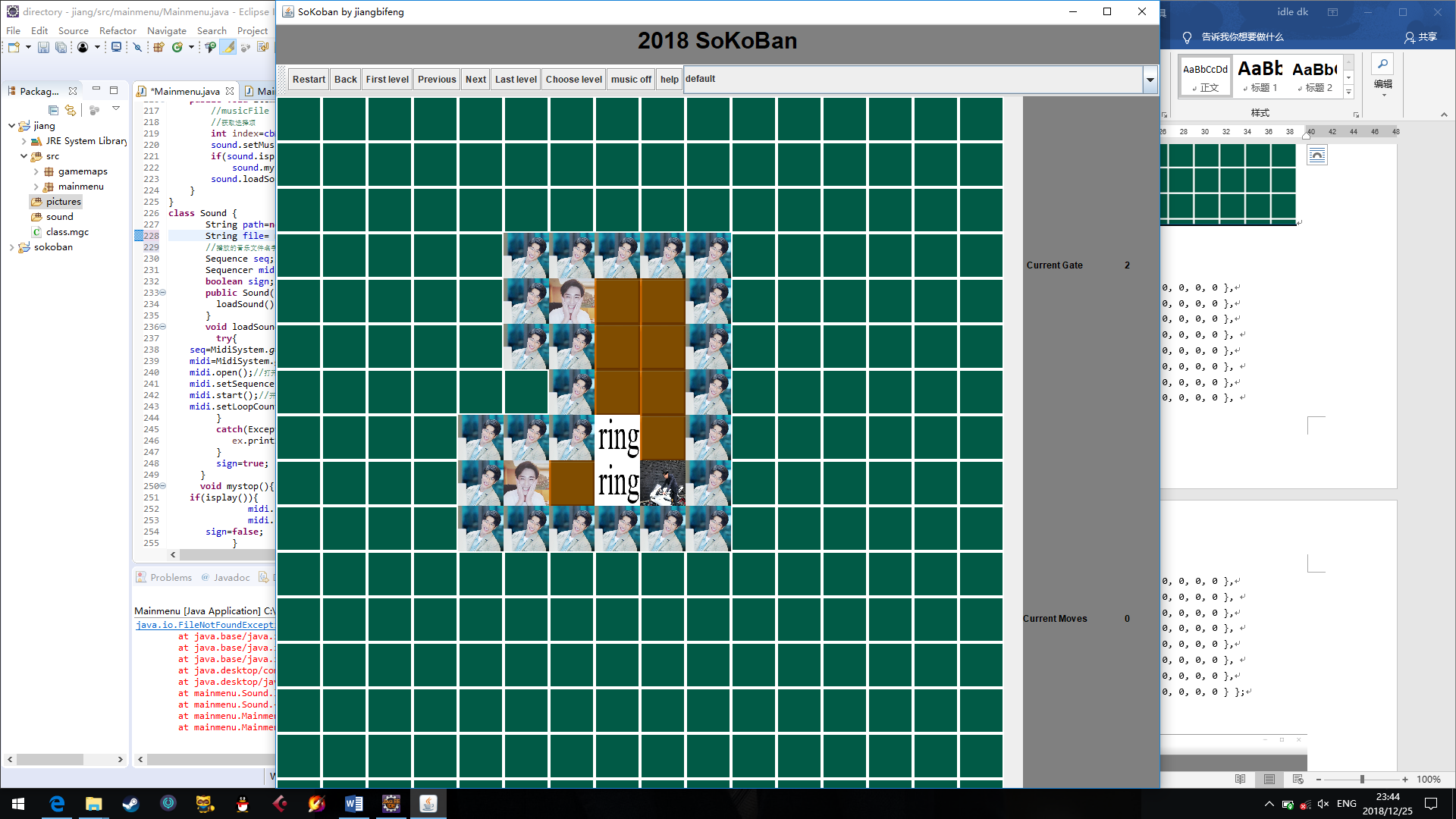
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

### 3.1.2 Second level map



array:

**final** **int** data2[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 3, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 5, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 5, 6, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

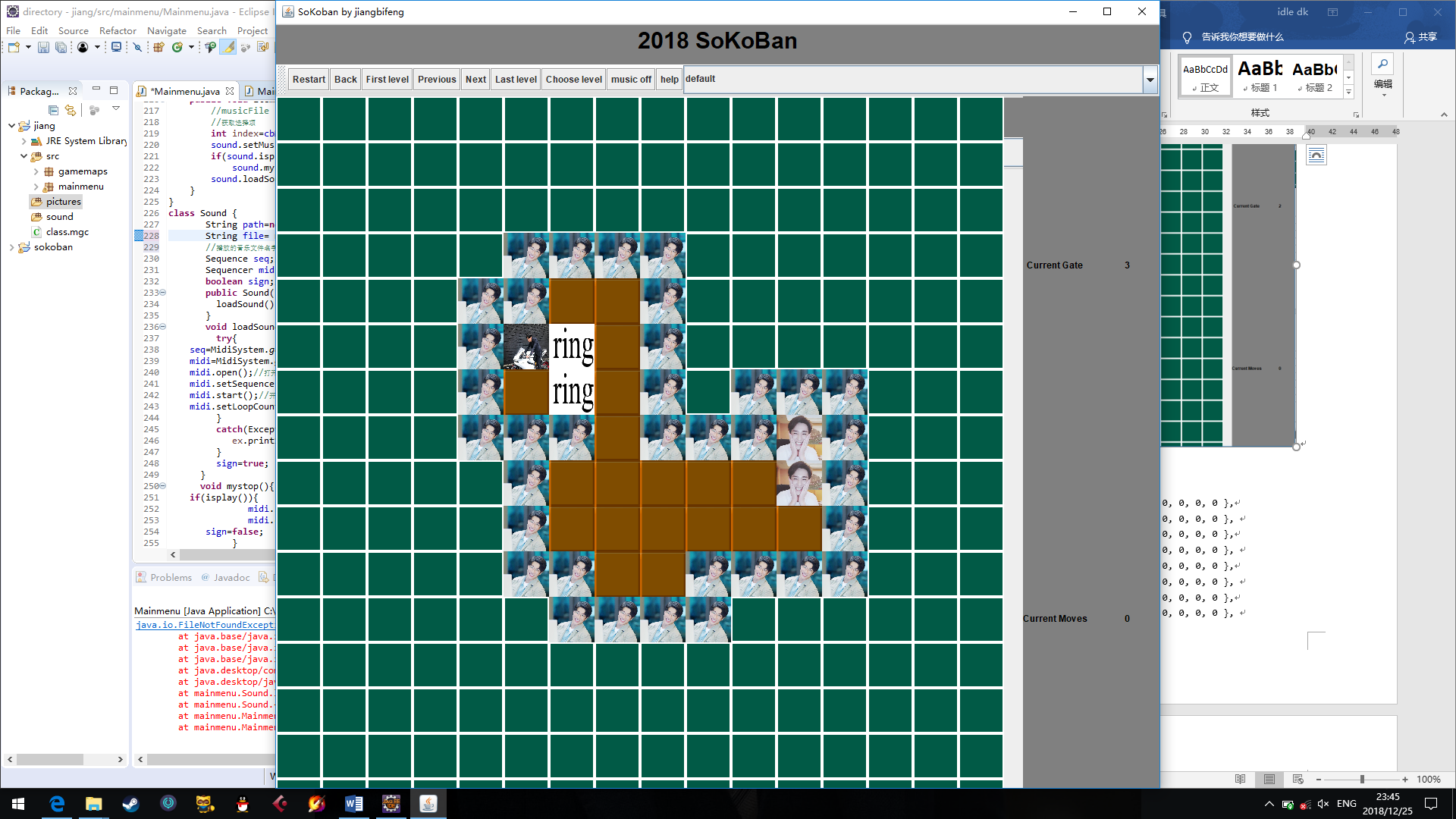
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

### 3.1.3 Third gate map



array:

**final** **int** data3[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 6, 5, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 5, 2, 1, 0, 1, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 2, 1, 1, 1, 3, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 3, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

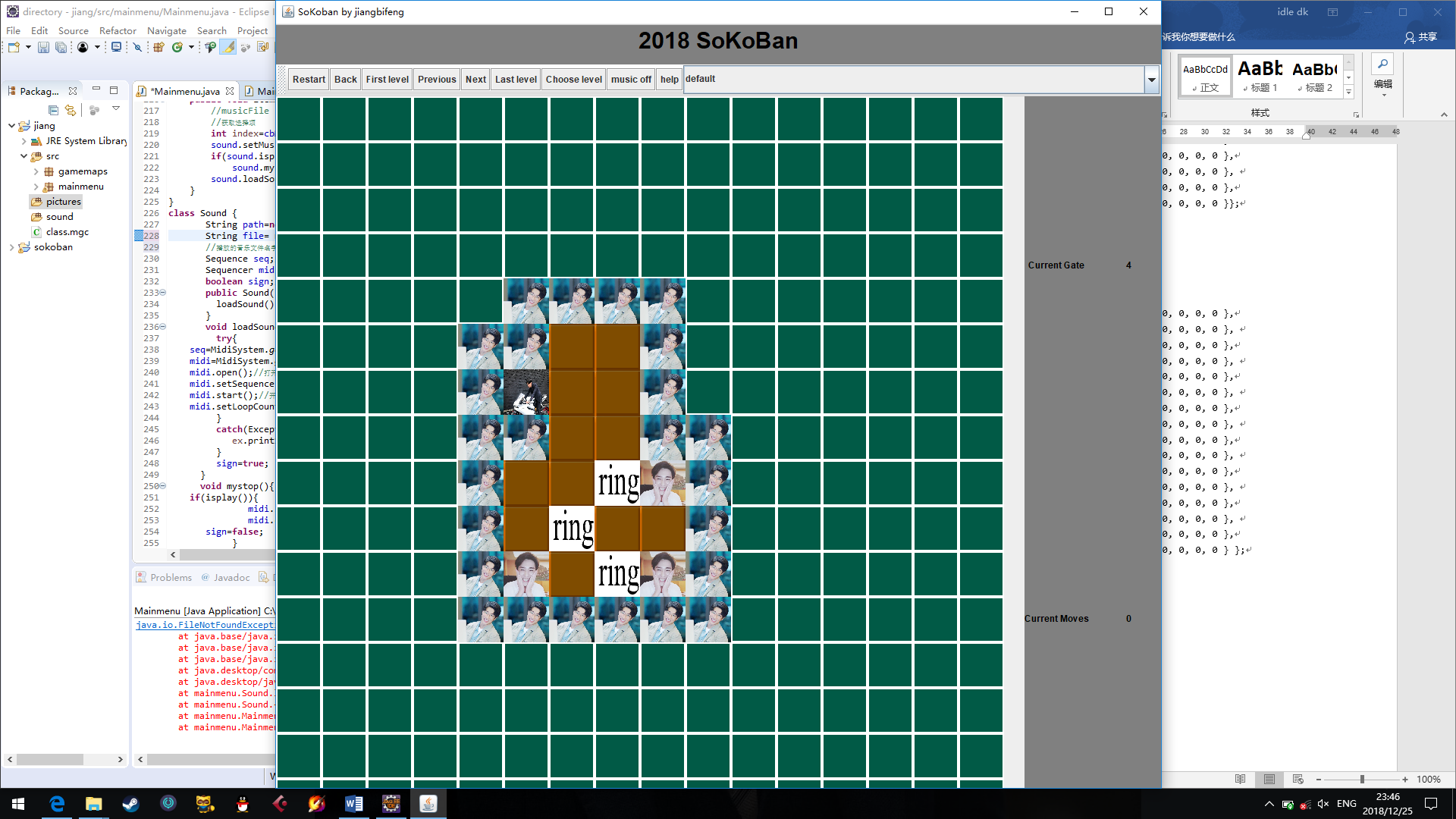
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

### 3.1.4 Fourth gate map



array:

**final** **int** data4[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 6, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 2, 5, 3, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 5, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 5, 3, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

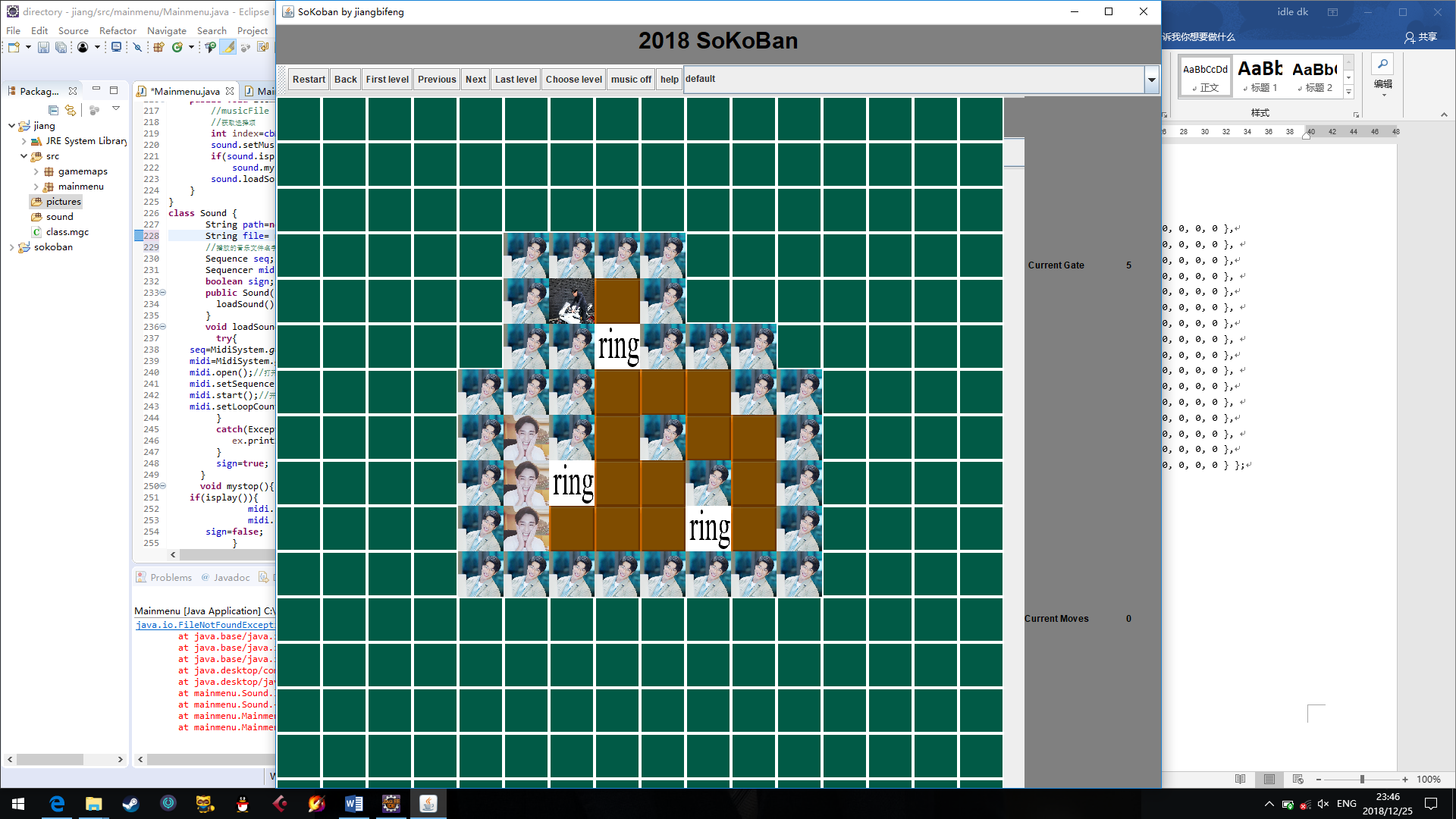
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

### 3.1.5 Fifth gate map



array:

**final** **int** data5[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 6, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 5, 1, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 1, 2, 1, 2, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 5, 2, 2, 1, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 2, 2, 5, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

## 3.2 Map pictures description

### 3.2.1 Grass

60x60



3.2.2 Wall

60x60



### 3.2.3 Floor

60X60



### 3.2.4 Hole

60X60



### 3.2.5 Box (normal and in the hole)

Normal: 60X60



In the hole :60X60



### 3.2.6 Warehouse keeper(up/down/left/right)

Face Up:60X60



Face Down:60X60



Face Left :60X60



Face Right :60X60



# 4.0 Code

## 4.1 mainmenu

package mainmenu;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.io.File;

import javax.sound.midi.MidiSystem;

import javax.sound.midi.Sequence;

import javax.sound.midi.Sequencer;

import javax.swing.\*;

@SuppressWarnings("serial")

public class Mainmenu extends JFrame implements ItemListener, ActionListener{

//drop-down combo box event with ItemListener

JLabel lblTitle,lblGateTitle,lblStepTitle;

static JLabel lblGate;

static JLabel lblStep;

JButton btnReset, btnBack, btnFirst, btnPrev, btnNext, btnLast, btnMusic, btnSelect, btnHelp;

@SuppressWarnings("rawtypes")

JComboBox cbMusic;

// Defining Music

JPanel topPanel;

// Define the top panel

JPanel eastPanel;

// Define the right panel

MainPanel mainPanel;

// Define the main interface panel

private String[] musicFile = {"healthy.mid", "talkself.mid", "willing.mid", "rain.mid"};

//Played music files

private String[] smusic = { "default", "music1", "music2", "music3"};

// Name of Music Directory Played

Sound sound;

//Definition Music Playing

//new JMenubar

JMenuBar menuBar=new JMenuBar();

//add item into JMenubar

private JMenuItem authorinfo =new JMenuItem("Developer info");

//new JToolbar to sotre buttons

JToolBar jtb;

JButton jb1, jb2, jb3, jb4, jb5, jb6, jb7,jb8,jb9;

@SuppressWarnings({ "unchecked", "rawtypes" })

public Mainmenu() {

lblTitle = new JLabel("2018 SoKoBan", JLabel.CENTER);

lblTitle.setFont(new Font("subject", Font.BOLD, 30));

lblTitle.setForeground(Color.black);

topPanel=new JPanel(new GridLayout(0, 1, 10, 10));

topPanel.setBackground(Color.gray);

jtb= new JToolBar();

eastPanel=new JPanel(new GridLayout(2, 1, 10, 20));

topPanel.setBackground(Color.gray);

//set text for buttons

btnReset = new JButton("Restart");

btnBack = new JButton("Back");

btnFirst = new JButton("First level");

btnPrev = new JButton("Previous");

btnNext = new JButton("Next");

btnLast = new JButton("Last level");

btnSelect = new JButton("Choose level");

btnMusic = new JButton("music off");

btnHelp = new JButton("help");

cbMusic = new JComboBox(smusic);

//add buttons into JToolBar

jtb.add(btnReset);

jtb.add(btnBack);

jtb.add(btnFirst);

jtb.add(btnPrev);

jtb.add(btnNext);

jtb.add(btnLast);

jtb.add(btnSelect);

jtb.add(btnMusic);

jtb.add(btnHelp);

jtb.add(cbMusic);

//set layout of toppanel

topPanel.add(jtb, BorderLayout.NORTH);

topPanel.add(lblTitle);

//set new JMenubar

setJMenuBar(menuBar);

//menuBar.add(menu1);

//add item to menu

//menu1.add(authorinfo);

//set title layout

Image icon = new ImageIcon("pic/img0.png").getImage();

setIconImage(icon);

setTitle("SoKoban by jiangbifeng");

//set lblgatetitle layout

lblGateTitle = new JLabel("Current Gate", JLabel.CENTER);

lblGateTitle.setFont(new Font("subject ", Font.BOLD, 12));

lblGateTitle.setForeground(Color.black);

//set lblGate layout

lblGate = new JLabel("1", JLabel.CENTER);

lblGate.setFont(new Font("subject ", Font.BOLD, 12));

lblGate.setForeground(Color.black);

//set lblStepTitle layout

lblStepTitle = new JLabel("Current Moves", JLabel.CENTER);

lblStepTitle.setFont(new Font("subject ", Font.BOLD, 12));

lblStepTitle.setForeground(Color.black);

//set lblStep layout

lblStep = new JLabel("0", JLabel.CENTER);

lblStep.setFont(new Font("subject", Font.BOLD, 12));

lblStep.setForeground(Color.black);

eastPanel.setBackground(Color.gray);

eastPanel.add(lblGateTitle);

eastPanel.add(lblGate);

eastPanel.add(lblStepTitle);

eastPanel.add(lblStep);

//registrate events

btnReset.addActionListener(this);

btnBack.addActionListener(this);

btnFirst.addActionListener(this);

btnPrev.addActionListener(this);

btnNext.addActionListener(this);

btnLast.addActionListener(this);

btnSelect.addActionListener(this);

btnMusic.addActionListener(this);

btnHelp.addActionListener(this);

cbMusic.addItemListener(this);

mainPanel = new MainPanel();

//add item to top panel

topPanel.add(lblTitle);

topPanel.add(jtb);

//set panels location

add(topPanel, BorderLayout.NORTH);

add(eastPanel,BorderLayout.EAST);

add(mainPanel);

//set game page layout

setSize(1200, 1000);// Width-height

setVisible(true);

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

sound=new Sound();

//get focus of the game

mainPanel.requestFocus();

}

public static void main(String[] args) {

new Mainmenu();

}

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

if (e.getSource() == btnReset) {

mainPanel.arrayList.clear();

// Clean back containers

mainPanel.init(mainPanel.level);

//Reread the map

} else if (e.getSource() == btnBack) {

Undo u = new Undo();

//initialization

for(int i=mainPanel.arrayList.size()-1;i>=0;i--) {

u=mainPanel.arrayList.get(i);

mainPanel.temp\_map[u.x][u.y] = u.flag;

mainPanel.manX=u.curManX;

mainPanel.manY=u.curManY;

mainPanel.arrayList.remove(mainPanel.arrayList.size()-1);

// Remove the last element

i=mainPanel.arrayList.size();

repaint();

// After pressing any key, judge whether to clear customs or not.

if(u.sign==1) break;

}

} else if (e.getSource() == btnLast) {

mainPanel.Sel("Last");

lblGate.setText(mainPanel.level+"");

} else if (e.getSource() == btnFirst) {

mainPanel.Sel("First");

lblGate.setText("1");

} else if (e.getSource() == btnPrev) {

mainPanel.Sel("Prev");

lblGate.setText(mainPanel.level+"");

} else if (e.getSource() == btnNext) {

mainPanel.Sel("Next");

lblGate.setText(mainPanel.level+"");

} else if (e.getSource() == btnSelect) {

String str="choose level（1-"+MainPanel.MAXLEVEL+")，now is level"+mainPanel.level;

String slevel=JOptionPane.showInputDialog(null, str,"choose level",JOptionPane.OK\_CANCEL\_OPTION);

int currentLevel=mainPanel.level;

try{

mainPanel.level=Integer.parseInt(slevel);

lblGate.setText(mainPanel.level+"");

}catch(Exception ex) {

mainPanel.level=currentLevel;

}

mainPanel.init(mainPanel.level);

} else if (e.getSource() == btnMusic) {

String title=btnMusic.getText();

if(title.equals("Music off")) {

sound.mystop();

//Turn off music

btnMusic.setText("Music on");

}

else {

sound.loadSound();

// Turn on the music

btnMusic.setText("Muisc off");

}

} else if (e.getSource() == btnHelp) {

JOptionPane.showMessageDialog(null, "Help message\n\n1. Restart\n2. Back\n3. Ask for baidu\n");

}

// Add monitor on it

mainPanel.requestFocus();

// Last step, next step, wait for the button. You must add this to listen.

}

public void itemStateChanged(ItemEvent e) {

//musicFile

//Get options

int index=cbMusic.getSelectedIndex();

sound.setMusic(musicFile[index]);

// Set the file name to the Sound class

if(sound.isplay())

// Judging whether music is playing

sound.mystop();

//Stop playing

sound.loadSound();

}

}

## 4.2 MainPanel

package mainmenu;

import java.awt.Image;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.ArrayList;

import javax.swing.\*;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

@SuppressWarnings({ "serial", "unused" })

public class MainPanel extends JPanel implements KeyListener {

private static Object lblStep;

//Define variable selection

int level = 1;

//The initial level is 1

public final static int MAXLEVEL = 5;

// Max

int manX, manY;

//Current coordinates of the depositary

public int count;

ArrayList<Undo> arrayList = new ArrayList<Undo>();

//Undo undo = new Undo();// initialization

// Define an array to load images

final Image pictures[] = { new ImageIcon("pictures/img0.jpg").getImage(), new ImageIcon("pictures/img1.jpg").getImage(),

new ImageIcon("pictures/img2.png").getImage(), new ImageIcon("pictures/img3.jpg").getImage(),

new ImageIcon("pictures/img4.jpg").getImage(), new ImageIcon("pictures/img5.png").getImage(),

new ImageIcon("pictures/img6.jpg").getImage(), new ImageIcon("pictures/img7.jpg").getImage(),

new ImageIcon("pictures/img8.jpg").getImage(), new ImageIcon("pictures/img9.jpg").getImage() };

// Define map array, people are 6, 7, 8, 9// level 1, 2, 3, 4, 5

final int data1[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 3, 2, 2, 5, 6, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

final int data2[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 3, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 5, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 5, 6, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

final int data3[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 6, 5, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 5, 2, 1, 0, 1, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 2, 1, 1, 1, 3, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 3, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};

final int data4[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 6, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 2, 5, 3, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 2, 5, 2, 2, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 5, 3, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

final int data5[][] = {

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 6, 2, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 5, 1, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 1, 2, 1, 2, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 5, 2, 2, 1, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 3, 2, 2, 2, 5, 2, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

// Define a temporary value

public int temp\_map[][], map[][];

// Define a constructor

public MainPanel() {

level = 1;

init(level);

//Registration Events

addKeyListener(this);

//Getting Focus

requestFocus();

//Getting Focus

}

public void init(int level) {

count = 0;//Variables defining statistical steps

Mainmenu.lblStep.setText(String.valueOf(count));

this.level = level;

switch (level) {

case 1:

copydata(data1);

break;

case 2:

copydata(data2);

break;

case 3:

copydata(data3);

break;

case 4:

copydata(data4);

break;

case 5:

copydata(data5);

break;

}

repaint();

// After pressing any key, judge whether to clear customs or not.

}

public void Sel(String opr) {

// First Prev Next Last

if (opr.equals("First")) {

level = 1;

} else if (opr.equals("Last")) {

level = MAXLEVEL;

} else if (opr.equals("Prev")) {

if (level > 1)

level--;

else

level = 1;

} else if (opr.equals("Next")) {

if (level >= MAXLEVEL)

level = MAXLEVEL;

else

level++;

}

init(level);

}

// copy array

void copydata(int data[][]) {

map = data;

// Keep the original array unchanged, but if it's a box or a person, it's the floor (if the box is empty, then the person's position is empty).

temp\_map = new int[16][16];

manX = manY = 0;

// initialization

for (int i = 0; i < 16; i++)

for (int j = 0; j < 16; j++) {

if (data[i][j] == 6 || data[i][j] == 7 || data[i][j] == 8 || data[i][j] == 9) {

// Human Initial Position

manX = i;

manY = j;

// map[i][j]=2;

}

temp\_map[i][j] = data[i][j];

}

}

@Override

// Display picture

// Picture sitting angle X coordinates

// Picture sitting angle y coordinates

// Picture width // Picture height

public void paint(Graphics g) {

int left, top;

for (int i = 0; i < 16; i++) {

for (int j = 0; j < 16; j++) {

left = j \* 60;

top = i \* 60;

g.drawImage(pictures[temp\_map[i][j]], left, top, 60, 60, this);

// To display the control this means to display Panel on the current panel

}

}

}

// Judging whether customs clearance is completed and completing the current customs clearance

// There were no empty pits in the original map (maybe people were standing on the empty pit (not right). No box to win

public boolean isWin() {

for (int i = 0; i < 16; i++) {

for (int j = 0; j < 16; j++) {

if (temp\_map[i][j] == 5)

// If there are empty pits, there will be no win.

return false;

}

}

return true;

}

public void keyPressed(KeyEvent e) {

switch (e.getKeyCode()) {

// Return the value of the key

case KeyEvent.VK\_LEFT:

// Press the left key

goLeft();

break;

case KeyEvent.VK\_UP:

// Press the up direction key

goUp();

break;

case KeyEvent.VK\_RIGHT:

// Press the right key

goRight();

break;

case KeyEvent.VK\_DOWN:

// Press down the direction key

goDown();

break;

}

Mainmenu.lblStep.setText(String.valueOf(count));

repaint();

// To Register Events

if (isWin()) {

JOptionPane.showMessageDialog(null, "congratulation to you");

Sel("Next");

// Enter the Next Pass

}

}

//Processing of the original map person, box, empty pit, full pit, floor in map

public void manOldPos() {

// After moving, the current person's position is changed to execute this code

if (map[manX][manY] == 5 || map[manX][manY] >= 6)

temp\_map[manX][manY] = 2;

else if (map[manX][manY] == 4)

// It turned out to be full pit 4 and empty Pit 3

temp\_map[manX][manY] = 3;

else

temp\_map[manX][manY] = map[manX][manY];

// The original person's place is the original picture

}

// 1: wall, 2: floor, 3: empty pit, 4: full pit, 5: box, 6: up, 7: down, 8: left, 9: right

public void goLeft() {

// The first position on the left is box 5, or box 4 on the empty pit.

if (temp\_map[manX][manY - 1] == 5 || temp\_map[manX][manY - 1] == 4) {

// The second position on the left is floor 2

if (temp\_map[manX][manY - 2] == 2) {

temp\_map[manX][manY - 2] = 5;

// The second position on the left is box 5.

temp\_map[manX][manY - 1] = 8;

// The first position on the left is 8:left

manOldPos();

// After moving, the current person's position is changed to execute this code

manY--;

count++;

} else if (temp\_map[manX][manY - 2] == 3) {

// The second position on the left is empty Pit 3

temp\_map[manX][manY - 2] = 4;

// The second position on the left is the box pushed into the empty pit to fill the pit 4.

temp\_map[manX][manY - 1] = 8;

// The first position on the left is 8:left

manOldPos();

// After moving, the current person's position is changed to execute this code

manY--;

count++;

}

} else if (temp\_map[manX][manY - 1] == 2 || temp\_map[manX][manY - 1] == 3) {

// The first position on the left is the floor, or the empty pit.

temp\_map[manX][manY - 1] = 8;

// The first position on the left is 8:left.

manOldPos();

// After moving, the current person's position is changed to execute this code

manY--;

count++;

}

}

public void goRight() {

// The first position on the left is box 5, or box 4 on the empty pit.

if (temp\_map[manX][manY + 1] == 5 || temp\_map[manX][manY + 1] == 4) {

// The second position on the left is floor 2

if (temp\_map[manX][manY + 2] == 2) {

temp\_map[manX][manY + 2] = 5;

// The second position on the left is box 5.

temp\_map[manX][manY + 1] = 9;

// The first position on the left is human 9:right.

manOldPos();

// After moving, the current person's position is changed to execute this code

manY++;

count++;

} else if (temp\_map[manX][manY + 2] == 3) {

// The second position on the left is empty Pit 3

temp\_map[manX][manY + 2] = 4;

// The second position on the left is the box pushed into the empty pit to fill the pit 4.

temp\_map[manX][manY + 1] = 9;

// The first position on the left is human 9:right.

manOldPos();

// After moving, the current person's position is changed to execute this code

manY++;

count++;

}

} else if (temp\_map[manX][manY + 1] == 2 || temp\_map[manX][manY + 1] == 3) {

// The first position on the left is the floor, or the empty pit.

temp\_map[manX][manY + 1] = 9;

// The first position on the left is human 9:right.

manOldPos();

// After moving, the current person's position is changed to execute this code

manY++;

count++;

}

}

public void goUp() {

// The first position on the left is box 5, or box 4 on the empty pit.

if (temp\_map[manX - 1][manY] == 5 || temp\_map[manX - 1][manY] == 4) {

// The second position on the left is floor 2

if (temp\_map[manX - 2][manY] == 2) {

temp\_map[manX - 2][manY] = 5;

// The second position on the left is box 5.

temp\_map[manX - 1][manY] = 6;

// The first position on the left is person 6: up

manOldPos();

// After moving, the current person's position is changed to execute this code

manX--;

count++;

} else if (temp\_map[manX - 2][manY] == 3) {

// The second position on the left is empty Pit 3

temp\_map[manX - 2][manY] = 4;

// The second position on the left is the box pushed into the empty pit to fill the pit 4.

temp\_map[manX - 1][manY] = 6;

// The first position on the left is person 6: up

manOldPos();

// After moving, the current person's position is changed to execute this code

manX--;

count++;

}

} else if (temp\_map[manX - 1][manY] == 2 || temp\_map[manX - 1][manY] == 3) {

// The first position on the left is the floor, or the empty pit.

temp\_map[manX - 1][manY] = 6;

// The first position on the left is person 6: up

manOldPos();

// After moving, the current person's position is changed to execute this code

manX--;

count++;

}

}

public void goDown() {

// The first position on the left is box 5, or box 4 on the empty pit.

if (temp\_map[manX + 1][manY] == 5 || temp\_map[manX + 1][manY] == 4) {

// The second position on the left is floor 2

if (temp\_map[manX + 2][manY] == 2) {

temp\_map[manX + 2][manY] = 5;

// The second position on the left is box 5.

temp\_map[manX + 1][manY] = 7;

// The first position on the left is person 7: down

manOldPos();

// After moving, the current person's position is changed to execute this code

manX++;

count++;

} else if (temp\_map[manX + 2][manY] == 3) {

// The second position on the left is empty Pit 3

temp\_map[manX + 2][manY] = 4;

// The second position on the left is the box pushed into the empty pit to fill the pit 4.

temp\_map[manX + 1][manY] = 7;

// The first position on the left is person 7: down

manOldPos();

// After moving, the current person's position is changed to execute this code

manX++;

count++;

}

} else if (temp\_map[manX + 1][manY] == 2 || temp\_map[manX + 1][manY] == 3) {

// The first position on the left is the floor, or the empty pit.

temp\_map[manX + 1][manY] = 7;

// The first position on the left is person 7: down

manOldPos();

// After moving, the current person's position is changed to execute this code

manX++;

count++;

}

}

@Override

public void keyReleased(KeyEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void keyTyped(KeyEvent arg0) {

// TODO Auto-generated method stub

}

}

## 4.3 Music

class Sound {

String path=new String("sound/");

// Music File Catalogue

String file= new String("healthy.mid") ;

// Name of playing music file

Sequence seq;

// sequence

Sequencer midi;

// Define a Sequencer-type variable

boolean sign;

// sign, true is playing, otherwise not playing

public Sound(){

loadSound();

}

void loadSound(){

//Play music

try{

seq=MidiSystem.getSequence(new File(path+file));

//Read music files

midi=MidiSystem.getSequencer();

// Create a music player and initialize parameters

midi.open();

//Turn on the music player

midi.setSequence(seq);

// Pass the file to the music player

midi.start();

// Start playing

midi.setLoopCount(Sequencer.LOOP\_CONTINUOUSLY);

//Infinite Loop Play

}

catch(Exception ex){

ex.printStackTrace();

}

sign=true;

}

void mystop(){

if(isplay()){

midi.stop();

//Stop playing

midi.close();

// stop playing

sign=false;

}

}

boolean isplay(){

// Judge sign's value. True means it's playing, otherwise it's not playing.

return sign;

}

void setMusic(String e){

//Get the name of the music file

file=e;

}

}

## 4.4 Undo

public class Undo {

public int sign;

//1: for a new start 0: follow the new one

public int x;

//X coordinates

public int y;

//Y coordinates

public int flag;

// Icon values on maps

//1: wall, 2: floor, 3: empty pit, 4: full pit, 5: box, 6: up, 7: down, 8: left, 9: right

public int curManX;

//X coordinates of the current person

public int curManY;

//Y coordinates of the current person

}