Assessment checklist

Unit assessment: candidate's assessment record

HP2L 48 — Software Development: Object Oriented Programming

Class 20SD1 Candidate's name Zhao Shirui

Group Candidate's ID 227006951

Outcome 2

Assessment task 1

Record of Performance

Evidence Requirements

Candidates will need to provide evidence to demonstrate their Knowledge and/or Skills by showing that they can investigate object oriented programming techniques and apply them appropriately to a design.

- Declaring and initialising variables.
- Using operators.
- Implementing control structures.
- Defining data structures.
- Accessing and manipulating data structures.
- Using parameter passing.
- Creating classes.
- Creating instances of classes.
- Creating relationships between classes.
- Creating constructor methods.
- Use of exceptions.
- Use of standard object libraries.
- Documenting code.

Satisfactory/Unsatisfactory Comments

The comment column can be used to highlight any re-assessment that may be needed.

Overall comments

Assessor's signature _____ **Date** _____

Beihang University (BUAA) Own Work Declaration

Candidate's name	*Zhao Shirui*	Class	***20SD1***
SCN Number	*227006951*	Group	

I declare that all submitted work will be my own.

I understand that any false claim for my work will be penalised in accordance with University regulations and relevant SQA requirements.

Signature of candidate: Zhao Shirui

Date（M/D/Y）: 12/30/2022

Content

Object-Oriented Programming Outcome 2

1. Functional Requirements

1. Help information (indicating which keys to use when moving).
2. Display the current off.
3. Display the current step number (effective steps).
4. Music can be opened and closed (other music can be selected from the drop-down list).
5. Optional OFF.
6. You can enter the previous level, the next level, the first level and the last level.
7. You can play the current game again.
8. After the current level of the game is displayed successfully, it will automatically enter the next level.
9. Fallback function (the number of steps for fallback is also reduced).
10. The game can end halfway.

2. Non-Functional Requirements

1. At least 5 levels. The latter level is much more difficult than the previous one.
2. Place each related map in the file and create a class reading file.
3. The box pusher pushes the box up, down, left and right, using different pictures.
4. The game map is rectangular, and the difference between rows and columns is more than 3.
5. The current closing and moving steps cannot be displayed on the map.
6. The buttons and other controls on the interface are placed around the map area.
7. The menu bar is required.
8. Two dimensional array must be used in the game.
9. At least 4 different classes.
10. At least member methods have formal parameters.
11. All class attributes are private (except for some special purpose attributes).
12. The control name must be standard btn, lbl, txt, lst, cbo, etc.
13. The window must have a panel at the top, bottom, left and right, with the box pushing game diagram in the middle.
14.5 There must be a pass in Guanzhong. The map man just appeared in the pit.
15.5 There must be a level in the level. When it first appeared, the box was in the pit (already
But we need to push it to another pit to succeed.
16. There is a title on the top, Sokoban Version X also has the student's name&student number.

3. MainFrame Code

This class is the main class, which supports all other classes. It displays all other main panels and game tips that need to be used on this panel. Sokoban's game main body is placed in the middle of this main page, and players can control the warehouse keeper to push boxes. The top (north) of the main page displays the current level. At the bottom (south) of the main page are four buttons, btnFirst, btnPrev, btnNext, and btnLast. The user can select a level according to these keys. When the level is the first level, btnFirst and btnPrev can only return to the first level. When the level is at the fifth level, btnLast and btnNext can only return to the fifth level. On the left (west) side of the main interface are music selection, play and cancel buttons. Users can select and control the playing of music according to their preferences. If the user makes an error when completing sokoban, you can select undo to return to the state before the operation. The right side (east side) of the main interface shows how many steps the current user has taken to play Sokoban. The user can evaluate the level according to how many steps he has taken.

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;


public class MainFrame    extends JFrame implements ActionListener {

    //Declare two panel objects, named pSouth,

    // pNorth respectively,

    // for the two positions of the panel.

    private JPanel pSouth,pNorth;

    //Declare four button keys,

    // which are used to let the user select the first, previous, next and last level.

    private JButton btnFirst, btnPrev, btnNext, btnLast;

    //The mainpanel class is where users play their main games.

    private MainPanel mainPanel;
```

```
//The playmusic class is where users choose music.
private PlayMusic playMusic;
//Menubar, which stores three options.
private JMenuBar jmb;
//Declare three options for users to choose,
// and the user can choose help or exit the game.
private JMenu file,edit,exit;
//The lower level can select help information.
private JMenuItem openfile,closefile,help;
//Declare a variable of type int to control the level of the level.
private int gate=1;
//Constructor to initialize the declared variables
// and add some space to the main page.
public    MainFrame() throws IOException {
//Initialize the mainpanel class to display
// the main game implementation in the center of the main form.
mainPanel=new MainPanel();
//Initialize the playlist class to
// enable the function of playing music to
// be realized on the left (west) side of the main window.
playMusic=new PlayMusic();
/*
Instantiate the north and south panel and
  initialize them in grid layout.
  The four buttons, menu bar and
  the number of current levels are displayed respectively.
*/
```

```
pSouth=new JPanel(new GridLayout(1,6,5,5));
pNorth=new JPanel(new GridLayout(1,2,5,5));
btnFirst=new JButton("First");
btnPrev=new JButton("Prev");
btnNext=new JButton("Next");
btnLast=new JButton("Last");
//Add four buttons to the action monitoring queue.
// The system allows them to trigger corresponding actions after being clicked.
btnFirst.addActionListener(this);
btnPrev.addActionListener(this);
btnNext.addActionListener(this);
btnLast.addActionListener(this);



jmb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
exit=new JMenu("Exit");
openfile=new JMenuItem("Open");
closefile=new JMenuItem("Close");
help=new JMenuItem("Help");
//Add menu options to the menu bar.
jmb.add(file);
jmb.add(edit);
jmb.add(exit);
//Add secondary options under the file option.
file.add(openfile);
```

```
file.addSeparator();
file.add(closefile);
file.add(help);
//Monitor the actions of sub menu options.
// When these options are clicked,
// corresponding events will be triggered respectively.
closefile.addActionListener(this);
openfile.addActionListener(this);
help.addActionListener(this);
//Add the btnUndo button
// to the playback panel and place it below cbomusic and jplay.
playMusic.add(mainPanel.btnUndo);
playMusic.jplay.addActionListener(this);
//Add event listeners to the cbomusic and btnUndo keys.
// After clicking or adjusting the options in the drop-down dialog box,
// the system executes their functions respectively.
playMusic.cbomusic.addActionListener(this);
mainPanel.btnUndo.addActionListener(this);
//Add the required buttons on pSouth panel.
pSouth.add(btnFirst);
pSouth.add(btnPrev);
pSouth.add(btnNext);
pSouth.add(btnLast);
//Add the required buttons on pNorth panel.
pNorth.add(jmb);
pNorth.add(mainPanel.jgate);
```

```
        //Use the border layout layout to
        // place pSouth in the south of the main interface,
        // pNorth in the north of the main interface,
        // the main body of the game in the middle of the page,
        // and music playback and cancellation operations in the west of the interface.
        this.add(pSouth,"South");
        this.add(pNorth,"North");
        this.add(mainPanel,"Center");
        this.add(playMusic,"West");
        //Set the parameters of the main interface.
            // Set the main interface to be visible,
            // and set its width to 700 and height to 550.
            // Set the default closing method to
            // click the upper right corner of the game page to exit.
            // Set the game interface to open in the middle of the screen by default.
            //Set the name of the game window to Sokoban Version 3 Zhao Shirui-25
        this.setVisible(true);
        this.setSize(700,550);
        this.setDefaultCloseOperation(3);
        this.setLocationRelativeTo(null);
        this.setTitle("Sokoban Version 3 Zhao Shirui-25");
        //Set the game to open automatically and focus on the mainpanel.
        mainPanel.requestFocus();
        mainPanel.requestFocus();
    }
        //Main function to instantiate a MainFrame object.
        public static void main(String[] args) throws IOException {
```

```
        new MainFrame();
    }

    //Listening function, whose return type is void,
    //is used to listen to each key.
    @Override
    public void actionPerformed(ActionEvent e) {
        //When the btnfirst button is pressed,
        // the game jumps to the first level.
        if(e.getSource()==btnFirst)
        {
            //Set the current level to 1
            gate=1;
            //Set the label jgate of the mainpanel to display as 1
            mainPanel.jgate.setText("Gate: "+gate);
            //Reset the pedometer of the mainpanel to start from zero.
            mainPanel.step=0;
            //Clear the stack.
            mainPanel.undo.clear();
            try {
                //Execute the initialization function in mainpanel.
                mainPanel.init(gate);
                //Re focus on the mainpanel.
                mainPanel.requestFocus();
                mainPanel.requestFocus();
            } catch (IOException ex) {
                ex.printStackTrace();
```

```
        }
    }
    //When the btnPrev button is pressed,
    // the game jumps to the previous level.
    // If the current level is already the first level,
    // it only returns to the first level.
    else if(e.getSource()==btnPrev)
    {
        //Set the current level minus 1
        gate--;
        //Reset the pedometer of the mainpanel to start from zero.
        mainPanel.step=0;
        //Clear the stack.
        mainPanel.undo.clear();
        //If the current level is less than or equal to 1, set the level to 1
        if(gate<=1)
            gate=1;
        //The current level number is displayed at the top of the interface.
        mainPanel.jgate.setText("Gate: "+gate);
        try {
            //Execute the initialization function in mainpanel.
            mainPanel.init(gate);
            //Re focus on the mainpanel.
            mainPanel.requestFocus();
            mainPanel.requestFocus();
        } catch (IOException ex) {
            ex.printStackTrace();
```

```
        }
    }
    //When the btnNext button is pressed,
    // the game jumps to the next level.
    // If the current level is already the last level,
    // it only returns to the last level.
    else if(e.getSource()==btnNext)
    {
        //Set level number plus 1
        gate++;
        //Reset the pedometer of the mainpanel to start from zero.
        mainPanel.step=0;
        //Clear the stack.
        mainPanel.undo.clear();
        //If the current level is greater than or equal to 5, set the level to 5
        if(gate>=5)
            gate=5;
        //The current level number is displayed at the top of the interface.
        mainPanel.jgate.setText("Gate: "+gate);
        try {
            //Execute the initialization function in mainpanel.
            mainPanel.init(gate);
            //Re focus on the mainpanel.
            mainPanel.requestFocus();
            mainPanel.requestFocus();
        } catch (IOException ex) {
            ex.printStackTrace();
```

```
        }
    }
    //When the btnLast button is pressed,
    // the game jumps to the last level.
    else if(e.getSource()==btnLast)
    {
        //Set the number of levels to 5
        gate=5;
        //Reset the pedometer of the mainpanel to start from zero.
        mainPanel.step=0;
        //The current level number is displayed at the top of the interface.
        mainPanel.jgate.setText("Gate: "+gate);
        //Clear the stack.
        mainPanel.undo.clear();
        try {
            //Execute the initialization function in mainpanel.
            mainPanel.init(gate);
            //Re focus on the mainpanel.
            mainPanel.requestFocus();
            mainPanel.requestFocus();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    //Click the music selection control and refocus to the mainpanel.
    else if(e.getSource()==playMusic.jplay)
            mainPanel.requestFocus();
```

```
        // Click the control music playback selection control and refocus to the
mainpanel.
        else if(e.getSource()==playMusic.cbomusic)
                mainPanel.requestFocus();
        //Click Undo to execute the undo function and focus on the mainpanel.
        else if(e.getSource()==mainPanel.btnUndo)
        {
            //Execute the undo function of mainpanel
            mainPanel.undoProcess();
            mainPanel.requestFocus();
        }
        //After clicking these menu controls,
        //close the game. If help is clicked,
        // the help dialog box will pop up.
        if(e.getSource()==closefile)
        {
            System.exit(0);
        }
        if (e.getSource()==openfile)
        {
            System.exit(0);
        }
        if (e.getSource()==help)
        {
            JOptionPane.showMessageDialog(null,"The up, down, left and right
buttons " +"\n"+
                    "on the keyboard can control the warehouse keeper to push
the box. " +"\n"+
```

```
                    "Click first to return to the first level, click prev to return to
the previous level, " +"\n"+
                    "next to the next level, and last to the last level. You can
select three pieces of music " +"\n"+
                    "to play or pause. The operation can be rolled back.");
            }
            //Re focus on the mainpanel.
            mainPanel.requestFocus();
        }
}
```

4. ReadMaps Code

This class is used to read external map files and is called by the main game classes of mainpanel. At the beginning of the game, the system automatically loads and reads the external file into a two-dimensional array, and passes this array to the mainpanel class, where a two-dimensional array will accept the map data.

```
import java.io.*;

public class ReadMaps {
    //Variable of type int,
    // used to control the rows and columns of the map.
    private int row=10,col=13;
    //String type variable, used to control the path of the map.
```

```
    private String path="221116\\src\\a\\maps\\";

    //Constructor.
    // The parameters are the number of maps
    // to be read and the two-dimensional array of maps to be read.
    public ReadMaps(int gate,int [][]rawMap) throws IOException {
        //String type variable, used to read the map.
        String filepath=path+"map"+gate+".txt";
        File file =new File(filepath);
        FileReader fr=null;
        BufferedReader br = null;

        fr=new FileReader(file);
        br=new BufferedReader(fr);
        //Read the data in the map.
        for (int i=0;i<row;i++)
        {
            //Read the data of the external map by row.
            String line=br.readLine();
            //The data of the external map is recorded in the array in bytes.
            byte[]bits=line.getBytes();
            for (int j = 0; j < col; j++) {
                //The data by row will be stored in a two-dimensional array in the
form of numbers.
                rawMap[i][j]=bits[j]-48;
            }
        }

    }
}
```

5. MainPanel code

This class is the implementation class of the game. The main body of the game will be displayed in the middle of the main interface. The file data read from the ReadMaps class will be saved by multiple two-dimensional arrays in this class, and then the player will be moved and the original data will be reproduced. The up, down, left, right and undo operations of the game are implemented in this class through keyboard press and pop events, and there is a prompt message in this class to prompt the player whether the game is over. For different difficulties of the map, for example, the initial position of the warehouse keeper is in a pit, or one of the positions is already full, but the player needs to push again.

```
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;
import java.util.Stack;

public class MainPanel extends JPanel implements KeyListener {
    //Variables of type int,
    // used to record the number of rows,
    // columns, the number of holes on the map,
    // and the full hole position of the fifth map.
    private int row=10,col=13,hole,x,y;
    //An array of type int is used to receive a
    // two-dimensional array of map data from the ReadMaps class.
    private int [][]rawMap=new int [row][col];
    //An array of type int, used to record the original map style.
    // When players move the warehouse keeper,
    // the system can rely on this array to restore the moved map to its original state.
    private int [][]data;
    //An array of int types,
    // used to record the status of players after moving.
    // After the player operates on the warehouse keeper,
    // the map will change to the shape after the player moves according to this
array.
    private int [][]temp;
    //Variables of type int, used to record the x and y coordinates of the warehouse
keeper.
    private int WX,WY;
    //The label class is used to display the current level number.
    public JLabel jgate;
    //Button class, click it to recall the previous step.
    public JButton btnUndo;
    //Read the file class, and store the read external map into the array.
    private ReadMaps readmaps;
    //The stack class is used to store the operation number of each key press.
    public Stack<Integer> undo=new Stack<>();
    //String type variable, used to record the path of the map image.
    private String path="221116\\src\\OOPPicture\\";
    //The int type attribute is used to match
    // whether the number of boxes and bits is the same.
    // When the number is different,
    // the default warehouse keeper is standing on a pit.
    private int box=0,pit=0;
    //Variables of type int, used to display the 30level and the number of moves.
```

```
    public int gate=1,step=0;
    //Set the Boolean variable judge. When the judge is true, the hole increases by
one. That is, a pit is full at the beginning of the game. You must push it again to clear
customs.
    private boolean judge=false;
    //Image type array, used to record map pictures.
    Image[] pics={
            new ImageIcon(path+"img1.png").getImage(),//floor0
            new ImageIcon(path+"img2.png").getImage(),//wall1
            new ImageIcon(path+"img6.png").getImage(),//top2
            new ImageIcon(path+"img7.png").getImage(),//down3
            new ImageIcon(path+"img8.png").getImage(),//left4
            new ImageIcon(path+"img9.png").getImage(),//right5
            new ImageIcon(path+"img3.png").getImage(),//hole6
            new ImageIcon(path+"img5.png").getImage(),//full7
            new ImageIcon(path+"img4.png").getImage(),//box8
            new ImageIcon(path+"img5.png").getImage()//full9
    };
    //Constructor to instantiate the declared class
    // and focus on this class when calling it.
    public MainPanel() throws IOException {
        data=new int[row][col];
        temp=new int[row][col];
        //The current level number is displayed at the top of the interface.
        jgate=new JLabel("Gate: "+gate);
        //Initialize the button, and the displayed name is Undo
        btnUndo=new JButton("Undo");
        //Call the init function to
        // initialize the map by passing the data of the current level.
        init(gate);
        //Add a key listener.
        // When the user operates
        // the warehouse administrator on the keyboard from top to bottom,
        // left to right, the system can receive the key parameters and perform the
next operation.
        addKeyListener(this);
        //Make the game focus on the mainpanel by default at the beginning.
        requestFocus();
    }
    //The initialization function
    // will assign a value to the level
    // and display the map data read on the main interface.
    public void init(int gate) throws IOException {
        //The parameter is passed to
```

```
            // the attribute gate to set the current level.
            this.gate=gate;
            //Each time the init function is called, the initial values of pit and box are 0.
            pit=0;
            box=0;
            //Read the corresponding map information according to
            // the passed level parameters, and put the read data
            // into the passed parameter array.
            readmaps=new ReadMaps(this.gate,rawMap);
            judge=false;
            copyData(rawMap);
            //Execute the paint function again.
            repaint();
        }
        //Transfer the read map information to the other two two-dimensional arrays.
        // One is used to save the original map information,
        // and the other is used to display the map information after the player's
operation.
        //If the number of boxes read does not match the number of pits,
        // set the player's position to the pit's position in the array where the map
information is saved.
        // If the number read from the transfer data array is 9,
        // the array where the map information is saved will be set to the pit state.
        // If the player does not push the box again, the game cannot be judged as over.
        private void copyData(int[][]rawdata)
        {
            for(int i=0;i<row;i++)
            {
                for (int j=0;j<col;j++)
                {
                    //Assign map information to the array that players can operate.
                    temp[i][j]=rawdata[i][j];
                    //If the number on the map is 6 or 8,
                    // the pit or box attribute increases by 1.
                    if(rawdata[i][j]==6)
                        pit++;
                    if (rawdata[i][j]==8)
                        box++;
                    //If the location of the warehouse keeper is read,
                    // record its location in WX and WY, and classify
                    // the location of the warehouse keeper as the floor
                    // in the original array where the map information is saved.

if(rawdata[i][j]==2||rawdata[i][j]==3||rawdata[i][j]==4||rawdata[i][j]==5)
```

```
                    {
                        WX=i;
                        WY=j;
                        data[i][j]=0;
                    }
                    else{
                        //If the box position is read, it is saved as the floor in the data
array.
                        if(rawdata[i][j]==8)
                            data[i][j]=0;
                        //In other cases, the data stored in data is consistent with the
map information.
                        else
                            data[i][j]=rawdata[i][j];
                    }
                    //If it is the fifth map,
                     // the position of the small full hole
                     // in the data array will be pushed again
                     // by default to pass,
                     // and the current position will be recorded.
                    if(rawdata[i][j]==9)
                    {
                        data[i][j]=6;
                        judge=true;
                        x=i;
                        y=j;
                    }
                }
            }
            //If the number of boxes read does not match the number of pits,
            // the location of the warehouse keeper in the data array is the location of
the pits.
            if(pit!=box)
                data[WX][WY]=6;
        }
        //Output the obtained map data in the form of pictures,
        // and display the steps taken
        // by the warehouse keeper on the right side of the map.
        public void paint(Graphics g)
        {
            super.paint(g);
            g.drawString("Steps: "+String.valueOf(step),550,200);
            for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
```

```
                    g.drawImage(pics[temp[i][j]],j*42,i*42,40,40,this);
                }
                System.out.println();
            }
            //If there is no data in the stack,
            // the user are not allowed to click btnundo.
            if (!undo.empty())
                btnUndo.setEnabled(true);
            else
                btnUndo.setEnabled(false);
        }
        //Undo function. The return type is void.
        // The undo function performs rollback
        // according to the number pushed in and displays the result on the main
interface.
        public void undoProcess()
        {
            if(undo.empty())
                btnUndo.setEnabled(false);
            int num=undo.pop();
            switch(num)
            {
                //When the pop-up number is 0,
                // it is determined that it has moved to the left,
                // and then it is withdrawn to the right.
                case 0:
                    temp[WX][WY+1]=4;
                    temp[WX][WY]=data[WX][WY];
                    WY++;
                    step--;
                    break;
                //When the pop-up number is 2,
                // it is determined to move to the left
                // and the box has been transported.
                // At this time, it is withdrawn to the right.
                case 2:
                    temp[WX][WY]=8;
                    temp[WX][WY+1]=4;
                    temp[WX][WY-1]=data[WX][WY-1];
                    WY++;
                    step--;
                    break;
                //When the pop-up number is 4,
                // it is determined that the box
```

```
// that moves to the left
// and has been pushed into the pit is withdrawn to the right.
case 4:
        temp[WX][WY]=7;
        temp[WX][WY+1]=4;
        temp[WX][WY-1]=data[WX][WY-1];
        WY++;
        step--;
        break;
//When the pop-up number is 10,
// it is determined that it has moved to the top,
// and then it is withdrawn to the downward.
case 10:
        temp[WX+1][WY]=2;
        temp[WX][WY]=data[WX][WY];
        WX++;
        step--;
        break;
//When the pop-up number is 12,
// it is determined to move to the top
// and the box has been transported.
// At this time, it is withdrawn to the downward.
case 12:
        temp[WX][WY]=8;
        temp[WX+1][WY]=2;
        temp[WX-1][WY]=data[WX-1][WY];
        WX++;
        step--;
        break;
//When the pop-up number is 14,
// it is determined that the box
// that moves to the top
// and has been pushed into the pit is withdrawn to the downward.
case 14:
        temp[WX][WY]=7;
        temp[WX+1][WY]=2;
        temp[WX-1][WY]=data[WX-1][WY];
        WX++;
        step--;
        break;
//When the pop-up number is 20,
// it is determined that it has moved to the right,
// and then it is withdrawn to the left.
case 20:
```

```
            temp[WX][WY-1]=5;
            temp[WX][WY]=data[WX][WY];
            WY--;
            step--;
            break;
        //When the pop-up number is 22,
        // it is determined that the box
        // that moves to the right and
        // has been pushed into the pit is withdrawn to the left.
        case 22:
            temp[WX][WY]=8;
            temp[WX][WY-1]=5;
            temp[WX][WY+1]=data[WX][WY+1];
            WY--;
            step++;
            break;
        //When the pop-up number is 24,
        // it is determined that the box
        // that moves to the right
        // and has been pushed into the pit is withdrawn to the left.
        case 24:
            temp[WX][WY]=7;
            temp[WX][WY-1]=4;
            temp[WX][WY+1]=data[WX][WY+1];
            WY--;
            step--;
            break;
        //When the pop-up number is 30,
        // it is determined that it has moved downward,
        // and then it is withdrawn upward.
        case 30:
            temp[WX-1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX--;
            step--;
            break;
        //When the pop-up number is 32,
        // it is determined that the box moves downward
        // and has been transported through the pit,
        // and then it is withdrawn upward.
        case 32:
            temp[WX][WY]=8;
            temp[WX-1][WY]=2;
            temp[WX+1][WY]=data[WX+1][WY];
```

```
                WX--;
                step--;
                break;
            //When the pop-up number is 34,
            // it is determined that the box
            // that moves to the downward
            // and has been pushed into the pit is withdrawn upward.
            case 34:
                temp[WX][WY]=7;
                temp[WX-1][WY]=2;
                temp[WX+1][WY]=data[WX+1][WY];
                WX--;
                step--;
                break;
        }
        repaint();
}
//The goleft function controls the warehouse administrator
// to move to the left at the allowed position.
private    void goleft()
{
        //If the warehouse keeper walks to the left
        // and the floor is in front of him, he can move.
        if(temp[WX][WY-1]==0)
        {
            temp[WX][WY-1]=4;
            temp[WX][WY]=data[WX][WY];
            WY--;
            step++;
            undo.push(0);
        }
        //If the warehouse keeper walks to the left
        // and there is a pit in front of him, he can move.
        else if(temp[WX][WY-1]==6)
        {
            temp[WX][WY-1]=4;
            temp[WX][WY]=data[WX][WY];
            WY--;
            step++;
            undo.push(0);
        }
        //If the warehouse keeper walks to the left,
        // and the box is in front,
        // and the floor is in front of the box,
```

```
// it can be moved.
else if(temp[WX][WY-1]==8&&temp[WX][WY-2]==0)
{
	temp[WX][WY-2]=8;
	temp[WX][WY-1]=4;
	temp[WX][WY]=data[WX][WY];
	WY--;
	step++;
	undo.push(2);
}
//The warehouse keeper can move
// if he walks to the left with a box in front and a pit in front.
else if(temp[WX][WY-1]==8&&temp[WX][WY-2]==6)
{
	temp[WX][WY-2]=7;
	temp[WX][WY-1]=4;
	temp[WX][WY]=data[WX][WY];
	WY--;
	step++;
	undo.push(2);
}
//The warehouse keeper can move the boxes
// that have been pushed into the pit on the left.
else if(temp[WX][WY-1]==7&&temp[WX][WY-2]==0)
{
	temp[WX][WY-2]=8;
	temp[WX][WY-1]=4;
	temp[WX][WY]=data[WX][WY];
	WY--;
	step++;
	undo.push(4);
}
//The warehouse keeper can move the boxes
// that have been pushed into the pit on the left.
else if(temp[WX][WY-1]==7&&temp[WX][WY-2]==6)
{
	temp[WX][WY-2]=7;
	temp[WX][WY-1]=4;
	temp[WX][WY]=data[WX][WY];
	WY--;
	step++;
	undo.push(4);
}
//The warehouse keeper can move the boxes
```

```
        // that have been pushed into the pit on the left.
        else if(temp[WX][WY-1]==9&&temp[WX][WY-2]==0)
        {
            temp[WX][WY-2]=8;
            temp[WX][WY-1]=4;
            temp[WX][WY]=data[WX][WY];
            WY--;
            step++;
            undo.push(4);
        }
        //The warehouse keeper can move the boxes
        // that have been pushed into the pit on the left.
        else if(temp[WX][WY-1]==9&&temp[WX][WY-2]==6)
        {
            temp[WX][WY-2]=7;
            temp[WX][WY-1]=4;
            temp[WX][WY]=data[WX][WY];
            WY--;
            step++;
            undo.push(4);
        }
    }
    //The gotop function controls the warehouse administrator
    // to move to the top at the allowed position.
    private    void gotop()
    {
        //If the warehouse keeper walks to the top
        // and the floor is in front of him, he can move.
        if(temp[WX-1][WY]==0)
        {
            temp[WX-1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX--;
            step++;
            undo.push(10);
        }
        //If the warehouse keeper walks to the top
        // and there is a pit in front of him, he can move.
        else if(temp[WX-1][WY]==6)
        {
            temp[WX-1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX--;
            step++;
```

```
        undo.push(10);
    }
    //If the warehouse keeper walks to the top,
    // and the box is in front,
    // and the floor is in front of the box,
    // it can be moved.
    else if(temp[WX-1][WY]==8&&temp[WX-2][WY]==0)
    {
        temp[WX-2][WY]=8;
        temp[WX-1][WY]=2;
        temp[WX][WY]=data[WX][WY];
        WX--;
        step++;
        undo.push(12);
    }
    //The warehouse keeper can move
    // if he walks to the top with a box in front and a pit in front.
    else if(temp[WX-1][WY]==8&&temp[WX-2][WY]==6)
    {
        temp[WX-2][WY]=7;
        temp[WX-1][WY]=2;
        temp[WX][WY]=data[WX][WY];
        WX--;
        step++;
        undo.push(12);
    }
    //The warehouse keeper can move
    // if he walks to the top with a box in front and a pit in front.
    else if(temp[WX-1][WY]==7&&temp[WX-2][WY]==0)
    {
        temp[WX-2][WY]=8;
        temp[WX-1][WY]=2;
        temp[WX][WY]=data[WX][WY];
        WX--;
        step++;
        undo.push(14);
    }
    //The warehouse keeper can move
    // if he walks to the top with a box in front and a pit in front.
    else if(temp[WX-1][WY]==7&&temp[WX-2][WY]==6)
    {
        temp[WX-2][WY]=7;
        temp[WX-1][WY]=2;
        temp[WX][WY]=data[WX][WY];
```

```
            WX--;
            step++;
            undo.push(14);
        }
        //The warehouse keeper can move
        // if he walks to the top with a box in front and a pit in front.
        else if(temp[WX-1][WY]==9&&temp[WX-2][WY]==0)
        {
            temp[WX-2][WY]=8;
            temp[WX-1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX--;
            step++;
            undo.push(14);
        }
        //The warehouse keeper can move
        // if he walks to the top with a box in front and a pit in front.
        else if(temp[WX-1][WY]==9&&temp[WX-2][WY]==6)
        {
            temp[WX-2][WY]=7;
            temp[WX-1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX--;
            step++;
            undo.push(14);
        }

    }
    //The goright function controls the warehouse administrator
    // to move to the right at the allowed position.
    private    void goright()
    {
        //If the warehouse keeper walks to the right
        // and the floor is in front of him, he can move.
        if(temp[WX][WY+1]==0)
        {
            temp[WX][WY+1]=5;
            temp[WX][WY]=data[WX][WY];
            WY++;
            step++;
            undo.push(20);
        }
        //If the warehouse keeper walks to the right
        // and there is a pit in front of him, he can move.
```

```
else if(temp[WX][WY+1]==6)
{
	temp[WX][WY+1]=5;
	temp[WX][WY]=data[WX][WY];
	WY++;
	step++;
	undo.push(20);
}
//If the warehouse keeper walks to the right,
// and the box is in front,
// and the floor is in front of the box,
// it can be moved.
else if(temp[WX][WY+1]==8&&temp[WX][WY+2]==0)
{
	temp[WX][WY+2]=8;
	temp[WX][WY+1]=5;
	temp[WX][WY]=data[WX][WY];
	WY++;
	step++;
	undo.push(22);
}
//The warehouse keeper can move
// if he walks to the right with a box in front and a pit in front.
else if(temp[WX][WY+1]==8&&temp[WX][WY+2]==6)
{
	temp[WX][WY+2]=7;
	temp[WX][WY+1]=5;
	temp[WX][WY]=data[WX][WY];
	WY++;
	step++;
	undo.push(22);
}
//The warehouse keeper can move
// if he walks to the right with a box in front and a pit in front.
else if(temp[WX][WY+1]==7&&temp[WX][WY+2]==0)
{
	temp[WX][WY+2]=8;
	temp[WX][WY+1]=5;
	temp[WX][WY]=data[WX][WY];
	WY++;
	step++;
	undo.push(24);
}
//The warehouse keeper can move
```

```
        // if he walks to the right with a box in front and a pit in front.
        else if(temp[WX][WY+1]==7&&temp[WX][WY+2]==6)
        {
            temp[WX][WY+2]=7;
            temp[WX][WY+1]=5;
            temp[WX][WY]=data[WX][WY];
            WY++;
            step++;
            undo.push(24);
        }
        //The warehouse keeper can move
        // if he walks to the right with a box in front and a pit in front.
        else if(temp[WX][WY+1]==9&&temp[WX][WY+2]==0)
        {
            temp[WX][WY+2]=8;
            temp[WX][WY+1]=5;
            temp[WX][WY]=data[WX][WY];
            WY++;
            step++;
            undo.push(24);
        }
        //The warehouse keeper can move
        // if he walks to the right with a box in front and a pit in front.
        else if(temp[WX][WY+1]==9&&temp[WX][WY+2]==6)
        {
            temp[WX][WY+2]=7;
            temp[WX][WY+1]=5;
            temp[WX][WY]=data[WX][WY];
            WY++;
            step++;
            undo.push(24);
        }
    }
    //The godown function controls the warehouse administrator
    // to move to the down at the allowed position.
    private    void godown()
    {
        //If the warehouse keeper walks to the downward
        // and the floor is in front of him, he can move.
        if(temp[WX+1][WY]==0)
        {
            temp[WX+1][WY]=3;
            temp[WX][WY]=data[WX][WY];
            WX++;
```

```
        step++;
        undo.push(30);
}
//If the warehouse keeper walks to the downward
// and there is a pit in front of him, he can move.
else if(temp[WX+1][WY]==6)
{
        temp[WX+1][WY]=3;
        temp[WX][WY]=data[WX][WY];
        WX++;
        step++;
        undo.push(31);
}
//If the warehouse keeper walks to the downward
// and the box is in front,
// and the floor is in front of the box,
// it can be moved.
else if(temp[WX+1][WY]==8&&temp[WX+2][WY]==0)
{
        temp[WX+2][WY]=8;
        temp[WX+1][WY]=3;
        temp[WX][WY]=data[WX][WY];
        WX++;
        step++;
        undo.push(32);
}
//The warehouse keeper can move
// if he walks to the downward with a box in front and a pit in front.
else if(temp[WX+1][WY]==8&&temp[WX+2][WY]==6)
{
        temp[WX+2][WY]=7;
        temp[WX+1][WY]=3;
        temp[WX][WY]=data[WX][WY];
        WX++;
        step++;
        undo.push(33);
}
//The warehouse keeper can move
// if he walks to the downward with a box in front and a pit in front.
else if(temp[WX+1][WY]==7&&temp[WX+2][WY]==0)
{
        temp[WX+2][WY]=8;
        temp[WX+1][WY]=3;
        temp[WX][WY]=data[WX][WY];
```

```
                WX++;
                step++;
                undo.push(34);
            }
            //The warehouse keeper can move
            // if he walks to the downward with a box in front and a pit in front.
            else if(temp[WX+1][WY]==7&&temp[WX+2][WY]==6)
            {
                temp[WX+2][WY]=7;
                temp[WX+1][WY]=3;
                temp[WX][WY]=data[WX][WY];
                WX++;
                step++;
                undo.push(35);
            }
            //The warehouse keeper can move
            // if he walks to the downward with a box in front and a pit in front.
            else if(temp[WX+1][WY]==9&&temp[WX+2][WY]==0)
            {
                temp[WX+2][WY]=8;
                temp[WX+1][WY]=3;
                temp[WX][WY]=data[WX][WY];
                WX++;
                step++;
                undo.push(36);
            }
            //The warehouse keeper can move
            // if he walks to the downward with a box in front and a pit in front.
            else if(temp[WX+1][WY]==9&&temp[WX+2][WY]==6)
            {
                temp[WX+2][WY]=7;
                temp[WX+1][WY]=3;
                temp[WX][WY]=data[WX][WY];
                WX++;
                step++;
                undo.push(37);
            }
        }

        @Override
        public void keyTyped(KeyEvent e) {

        }
        //Press the key to press the monitoring function.
```

```
// when the pressed key is up, down, left
// and right, the warehouse keeper moves in the corresponding direction.
@Override
public void keyPressed(KeyEvent e) {
        switch (e.getKeyCode())
        {
                case 37:
                        goleft();
                        break;
                case 38:
                        gotop();
                        break;
                case 39:
                        goright();
                        break;
                case 40:
                        godown();
                        break;
        }
        repaint();
}
//Press the key to lift the monitoring function.
// When the pressed key is lifted,
// judge whether there is any box on the map that has been pushed into the pit.
// If there is, the game will not be ended.
// If not, the user will be prompted to complete
// the game and can choose whether to enter the next level.
@Override
public void keyReleased(KeyEvent e) {
//Set the value of the hole to 0.
        hole=0;
        //
        if(judge)
                hole++;
        if(gate==5&&temp[x][y]==7)
                hole=0;
        for(int i=0;i<row;i++)
        {
                for (int j=0;j<col;j++)
                {
                        if(temp[i][j]==8)
                                hole++;
                }
        }
```

```
            if(hole==0)
            {
                int result=JOptionPane.showInternalConfirmDialog(null,
                                "Do you want to challenge the next level?", "Great!",
                                JOptionPane.YES_NO_CANCEL_OPTION,
                                JOptionPane.INFORMATION_MESSAGE);
                if(result==JOptionPane.OK_OPTION)
                {
                    gate++;
                    jgate.setText("Gate: "+gate);
                    if(gate>5)
                    {
                        gate=5;
                        jgate.setText("Gate: "+gate);
                        JOptionPane.showInternalConfirmDialog(null,"You have
completed the last level!",
                                        "Well Done!",JOptionPane.YES_OPTION,
JOptionPane.INFORMATION_MESSAGE);
                    }
                    try {
                         init(gate);
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                }
                else if(result==JOptionPane.NO_OPTION)
                {
                    try {
                        init(gate);
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                }
                undo.clear();
                step=0;
            }

        }
}
```

6. PlayMusic Code

This class mainly realizes the function of playing music. This page will be displayed on the left (west) side of the main interface. After starting the game, the music is paused, and the first music is played by default. Players can select the three imported music and control its playback and pause. When switching songs, the music can be played automatically.

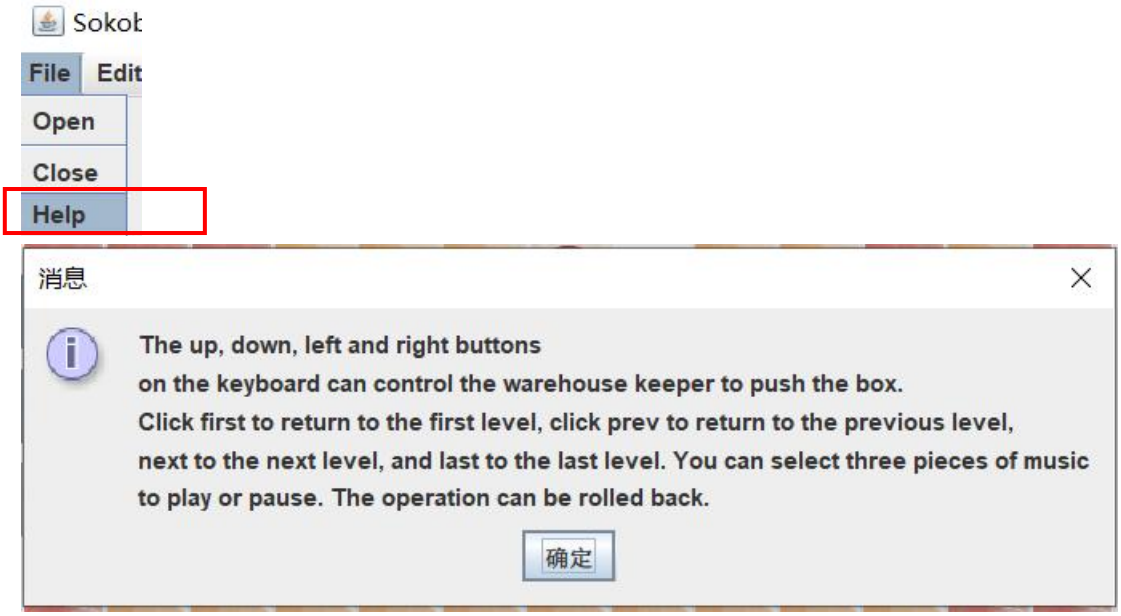
```
import javafx.scene.media.AudioClip;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.File;

class PlayMusic extends JPanel implements ActionListener, ItemListener {

    //Button control, used to control the playing state of music.
    public JButton jplay;
    //Music playing class, used to play music.
    private AudioClip ac;
    //String type variable, used to control the path of music playing.
    private String path="221116\\src\\music\\";
    //String type variable array, used to control the name of music.
    private String[] musicFile={"clock.MP3","fog.mp3","lazy.mp3"};
    //String type variable array, used to control which music is played.
    private String[] musicDisplay={"First","Second","Third"};
    //The label control array is used to occupy space in the grid layout.
    private JLabel []jlabel=new JLabel[3];
    //The combobox class is used to select music.
    public JComboBox cbomusic;
    //Constructor, which is used to instantiate the previously declared class.
    public PlayMusic()
    {
        this.setLayout(new GridLayout(10,1,5,10));
        cbomusic=new JComboBox(musicDisplay);
        jplay=new JButton("play");
        for(int i=0;i<3;i++)
        {
            jlabel[i]=new JLabel();
            this.add(jlabel[i]);
```

```
        }

        this.add(cbomusic);
        this.add(jplay);

        jplay.addActionListener(this);
        cbomusic.addItemListener(this);
        ac=new AudioClip(new File(path+"\\clock.MP3").toURI().toString());
        ac.stop();
    }
    @Override
    //Monitor function. When the current music is playing,
    // jplay displays stop. When the current music is paused,
    // jplay displays play.
    public void actionPerformed(ActionEvent e) {
        if(jplay.getText()=="play")
        {
            ac.play();
            jplay.setText("stop");
        }
        else {
            ac.stop();
            jplay.setText("play");
        }
    }
    //The function that controls
    // the change of the content displayed on the combobox.
    // When switching music, the music will play automatically.
    @Override
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==cbomusic)
        {
            String str=musicFile[cbomusic.getSelectedIndex()];
            ac.stop();
            jplay.setText("play");
            ac=new AudioClip(new File(path+str).toURI().toString());
            ac.play();
            jplay.setText("stop");
        }
    }


}
```

7.Basic operation.

7.1 Help

Click help under the file prompt box to pop up the help dialog information.

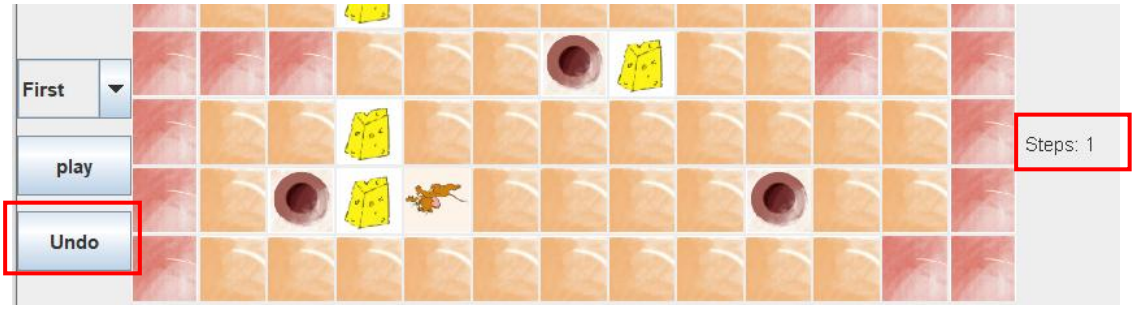


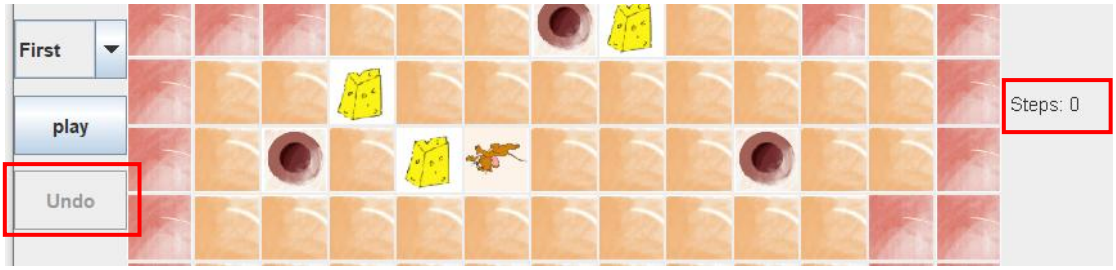

```
//If the key clicked is help in the menu bar, a help dialog box will pop up.
if (e.getSource()==help)
{
    JOptionPane.showMessageDialog(null,"The up, down, left and right buttons "
+"\n"+
                "on the keyboard can control the warehouse keeper to push the box. "
+"\n"+
                "Click first to return to the first level, click prev to return to the
previous level, " +"\n"+
                "next to the next level, and last to the last level. You can select three
pieces of music " +"\n"+
                "to play or pause. The operation can be rolled back.");
}
```

7.2 Undo

If the warehouse administrator has not been moved, the undo button is disabled

When the warehouse keeper moves in any direction, click Recall to return to the previous step.






```
//Undo function. The return type is void.
// The undo function performs rollback
// according to the number pushed in and displays the result on the main interface.
public void undoProcess()
{
    if(undo.empty())
        btnUndo.setEnabled(false);
    int num=undo.pop();
    switch(num)
    {
        //When the pop-up number is 0,
        // it is determined that it has moved to the left,
        // and then it is withdrawn to the right.
        case 0:
            temp[WX][WY+1]=4;
```

```
            temp[WX][WY]=data[WX][WY];
            WY++;
            step--;
            break;
        //When the pop-up number is 2,
        // it is determined to move to the left
        // and the box has been transported.
        // At this time, it is withdrawn to the right.
        case 2:
            temp[WX][WY]=8;
            temp[WX][WY+1]=4;
            temp[WX][WY-1]=data[WX][WY-1];
            WY++;
            step--;
            break;
        //When the pop-up number is 4,
        // it is determined that the box
        // that moves to the left
        // and has been pushed into the pit is withdrawn to the right.
        case 4:
            temp[WX][WY]=7;
            temp[WX][WY+1]=4;
            temp[WX][WY-1]=data[WX][WY-1];
            WY++;
            step--;
            break;
        //When the pop-up number is 10,
        // it is determined that it has moved to the top,
        // and then it is withdrawn to the downward.
        case 10:
            temp[WX+1][WY]=2;
            temp[WX][WY]=data[WX][WY];
            WX++;
            step--;
            break;
        //When the pop-up number is 12,
        // it is determined to move to the top
        // and the box has been transported.
        // At this time, it is withdrawn to the downward.
        case 12:
            temp[WX][WY]=8;
            temp[WX+1][WY]=2;
            temp[WX-1][WY]=data[WX-1][WY];
            WX++;
```

```
        step--;
        break;
//When the pop-up number is 14,
// it is determined that the box
// that moves to the top
// and has been pushed into the pit is withdrawn to the downward.
case 14:
        temp[WX][WY]=7;
        temp[WX+1][WY]=2;
        temp[WX-1][WY]=data[WX-1][WY];
        WX++;
        step--;
        break;
//When the pop-up number is 20,
// it is determined that it has moved to the right,
// and then it is withdrawn to the left.
case 20:
        temp[WX][WY-1]=5;
        temp[WX][WY]=data[WX][WY];
        WY--;
        step--;
        break;
//When the pop-up number is 22,
// it is determined that the box
// that moves to the right and
// has been pushed into the pit is withdrawn to the left.
case 22:
        temp[WX][WY]=8;
        temp[WX][WY-1]=5;
        temp[WX][WY+1]=data[WX][WY+1];
        WY--;
        step++;
        break;
//When the pop-up number is 24,
// it is determined that the box
// that moves to the right
// and has been pushed into the pit is withdrawn to the left.
case 24:
        temp[WX][WY]=7;
        temp[WX][WY-1]=4;
        temp[WX][WY+1]=data[WX][WY+1];
        WY--;
        step--;
        break;
```

```
            //When the pop-up number is 30,
            // it is determined that it has moved downward,
            // and then it is withdrawn upward.
            case 30:
                temp[WX-1][WY]=2;
                temp[WX][WY]=data[WX][WY];
                WX--;
                step--;
                break;
            //When the pop-up number is 32,
            // it is determined that the box moves downward
            // and has been transported through the pit,
            // and then it is withdrawn upward.
            case 32:
                temp[WX][WY]=8;
                temp[WX-1][WY]=2;
                temp[WX+1][WY]=data[WX+1][WY];
                WX--;
                step--;
                break;
            //When the pop-up number is 34,
            // it is determined that the box
            // that moves to the downward
            // and has been pushed into the pit is withdrawn upward.
            case 34:
                temp[WX][WY]=7;
                temp[WX-1][WY]=2;
                temp[WX+1][WY]=data[WX+1][WY];
                WX--;
                step--;
                break;
        }
        //Execute the paint function again.
        repaint();
    }
```

7.3 Play Music

When entering the game, the music is played in the pause mode. Click play to start playing music.


```
//Monitor function. When the current music is playing,
// jplay displays stop. When the current music is paused,
// jplay displays play.
public void actionPerformed(ActionEvent e) {
	if(jplay.getText()=="play")
	{
		ac.play();
		jplay.setText("stop");
	}
	else {
		ac.stop();
		jplay.setText("play");
	}
}
```

When switching music, the system automatically starts playing music.


```
//The function that controls
// the change of the content displayed on the combobox.
// When switching music, the music will play automatically.
@Override
public void itemStateChanged(ItemEvent e) {
	if(e.getSource()==cbomusic)
	{
		//If the contents of the combobox change, read the corresponding serial
number of the combobox first.
```

```
            String str=musicFile[cbomusic.getSelectedIndex()];
            //Pause the currently playing music.
            ac.stop();
            //Switch the content of jplay to play.
            jplay.setText("play");
            //Retrieve files for the selected music.
            ac=new AudioClip(new File(path+str).toURI().toString());
            //play the music
            ac.play();
            //Switch the content of jplay to stop.
            jplay.setText("stop");
        }
}
```

7.4 The birth place is a pit.

According to the judgment of the system, when the number of boxes and the number of pits do not match, the system defaults that the warehouse keeper is standing on the box.

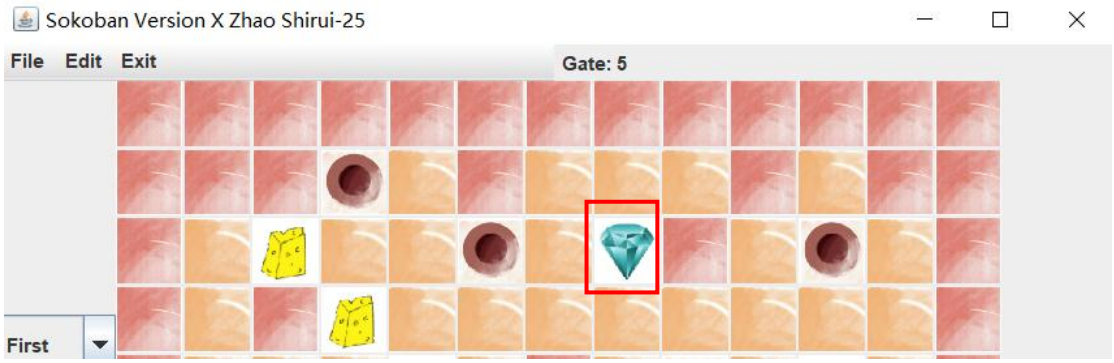

```
//Transfer the read map information to the other two two-dimensional arrays.
// One is used to save the original map information,
// and the other is used to display the map information after the player's operation.
//If the number of boxes read does not match the number of pits,
// set the player's position to the pit's position in the array where the map information is saved.
// If the number read from the transfer data array is 9,
// the array where the map information is saved will be set to the pit state.
```

```
// If the player does not push the box again, the game cannot be judged as over.
private void copyData(int[][]rawdata)
{
    for(int i=0;i<row;i++)
    {
        for (int j=0;j<col;j++)
        {
            //Assign map information to the array that players can operate.
            temp[i][j]=rawdata[i][j];
            //If the number on the map is 6 or 8,
            // the pit or box attribute increases by 1.
            if(rawdata[i][j]==6)
                pit++;
            if (rawdata[i][j]==8)
                box++;
            //If the location of the warehouse keeper is read,
            // record its location in WX and WY, and classify
            // the location of the warehouse keeper as the floor
            // in the original array where the map information is saved.

if(rawdata[i][j]==2||rawdata[i][j]==3||rawdata[i][j]==4||rawdata[i][j]==5)
            {
                WX=i;
                WY=j;
                data[i][j]=0;
            }
            else{
                //If the box position is read, it is saved as the floor in the data
array.
                if(rawdata[i][j]==8)
                    data[i][j]=0;
                //In other cases, the data stored in data is consistent with the map
information.
                else
                    data[i][j]=rawdata[i][j];
            }
            //If it is the fifth map,
            // the position of the small full hole
            // in the data array will be pushed again
            // by default to pass,
            // and the current position will be recorded.
            if(rawdata[i][j]==9)
            {
                data[i][j]=6;
```

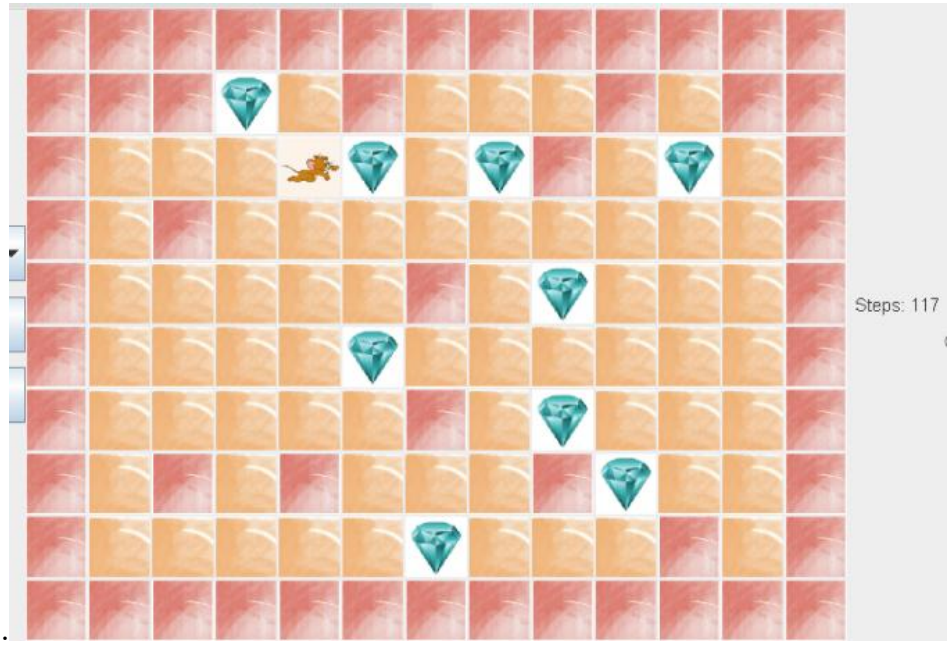
```
                    judge=true;
                    x=i;
                    y=j;
                }
            }
        }
        //If the number of boxes read does not match the number of pits,
        // the location of the warehouse keeper in the data array is the location of the
pits.
        if(pit!=box)
            data[WX][WY]=6;
}
```

7.5 Full Pit

At the beginning of a certain level, there is a full pit state. At this time, you need to push the box into the pit again, otherwise the game cannot be determined to be completed.

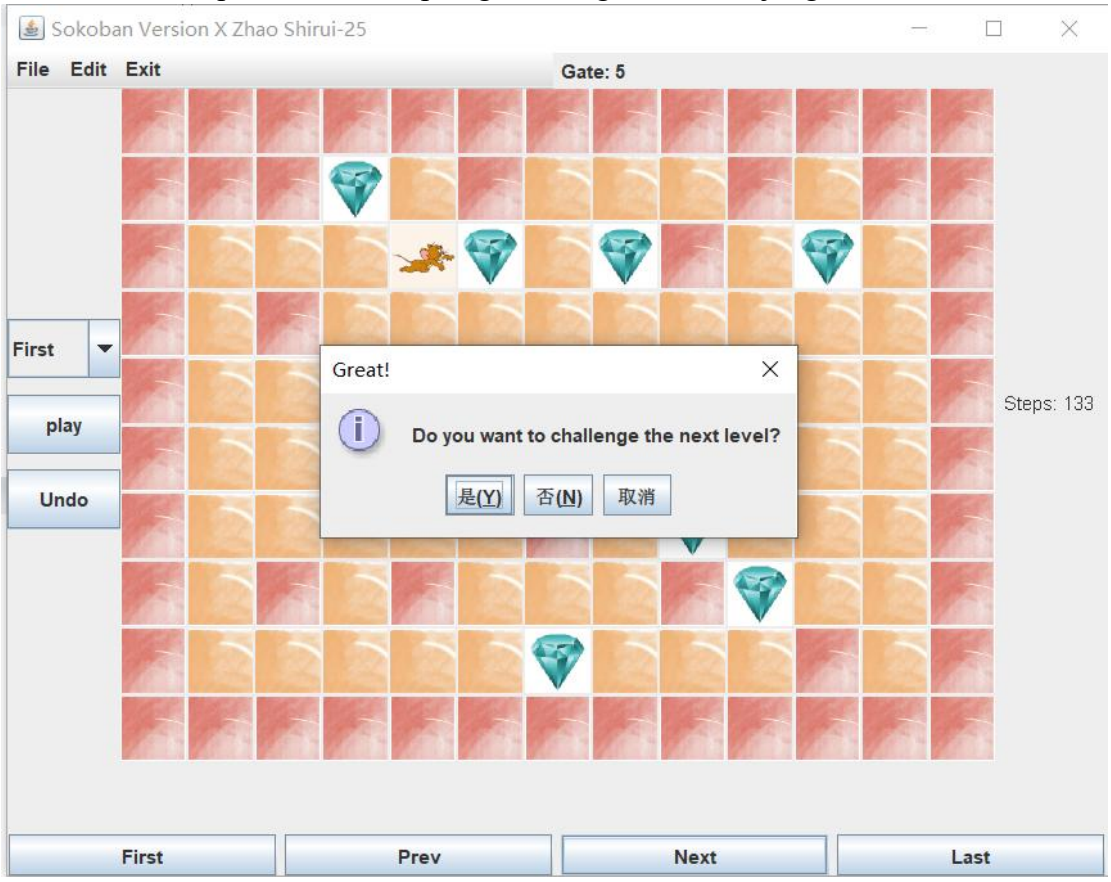


Even if all the other boxes are pushed into the pit, the game will not end



.

When the box is pushed into the pit again, the game can be judged as finished.



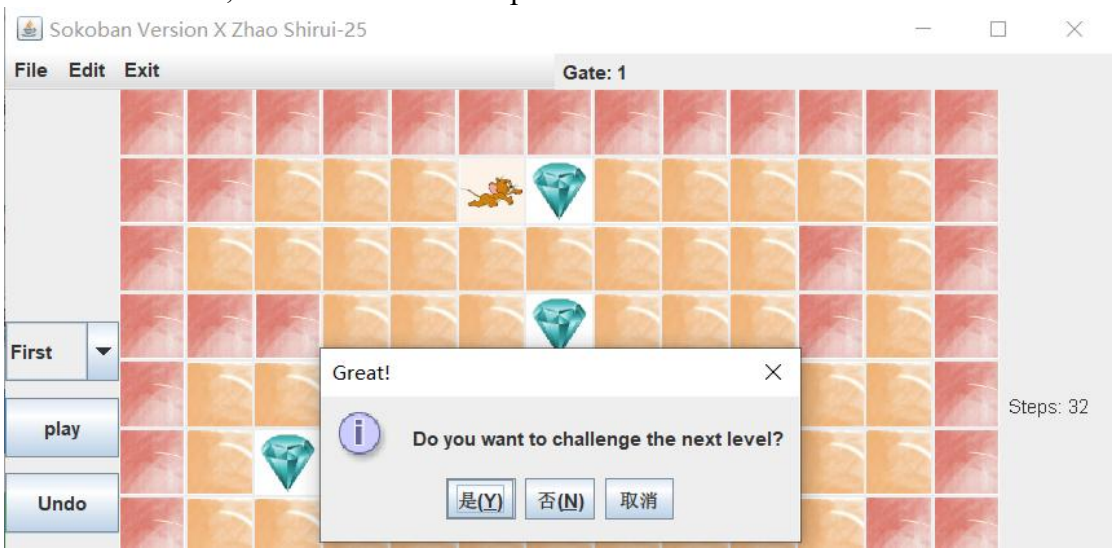

```
//Transfer the read map information to the other two two-dimensional arrays.
// One is used to save the original map information,
// and the other is used to display the map information after the player's operation.
//If the number of boxes read does not match the number of pits,
// set the player's position to the pit's position in the array where the map information is saved.
// If the number read from the transfer data array is 9,
// the array where the map information is saved will be set to the pit state.
// If the player does not push the box again, the game cannot be judged as over.
private void copyData(int[][]rawdata)
{
    for(int i=0;i<row;i++)
    {
        for (int j=0;j<col;j++)
        {
            //Assign map information to the array that players can operate.
            temp[i][j]=rawdata[i][j];
            //If the number on the map is 6 or 8,
            // the pit or box attribute increases by 1.
            if(rawdata[i][j]==6)
                pit++;
            if (rawdata[i][j]==8)
```

```
                box++;
            //If the location of the warehouse keeper is read,
            // record its location in WX and WY, and classify
            // the location of the warehouse keeper as the floor
            // in the original array where the map information is saved.

if(rawdata[i][j]==2||rawdata[i][j]==3||rawdata[i][j]==4||rawdata[i][j]==5)
            {
                WX=i;
                WY=j;
                data[i][j]=0;
            }
            else{
                //If the box position is read, it is saved as the floor in the data
array.
                if(rawdata[i][j]==8)
                    data[i][j]=0;
                //In other cases, the data stored in data is consistent with the map
information.
                else
                    data[i][j]=rawdata[i][j];
            }
            //If it is the fifth map,
            // the position of the small full hole
            // in the data array will be pushed again
            // by default to pass,
            // and the current position will be recorded.
            if(rawdata[i][j]==9)
            {
                data[i][j]=6;
                judge=true;
                x=i;
                y=j;
            }
        }
    }
    //If the number of boxes read does not match the number of pits,
    // the location of the warehouse keeper in the data array is the location of the
pits.
    if(pit!=box)
        data[WX][WY]=6;
}
//Press the key to lift the monitoring function.
// When the pressed key is lifted,
```

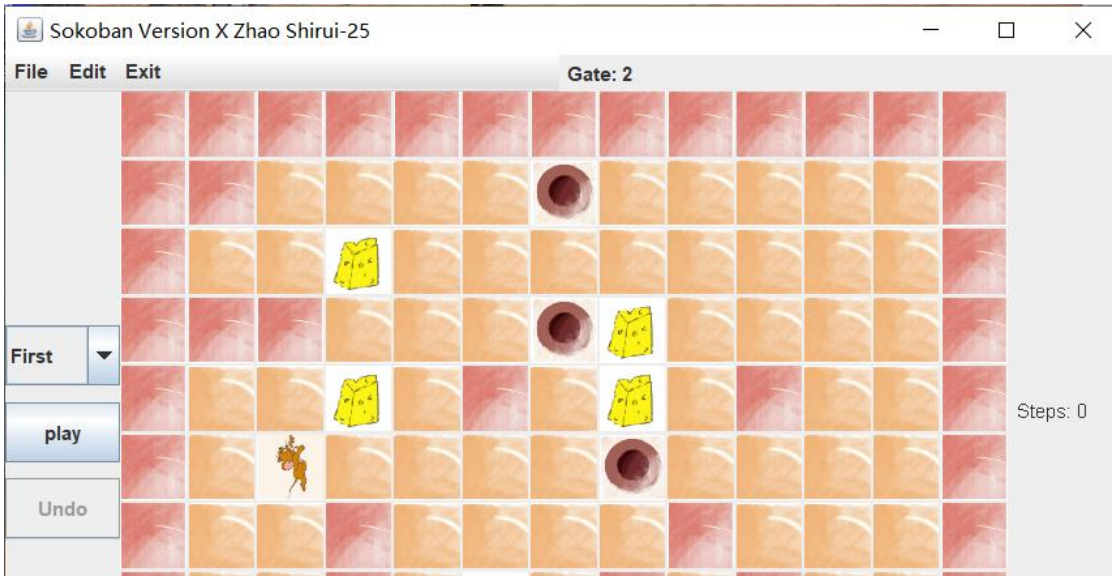
```
// judge whether there is any box on the map that has been pushed into the pit.
// If there is, the game will not be ended.
// If not, the user will be prompted to complete
// the game and can choose whether to enter the next level.
@Override
public void keyReleased(KeyEvent e) {
//Set the value of the hole to 0.
    hole=0;
    //If it is determined to be true, hole first increases by 1, that is, there are still
boxes not pushed into the pit.
    if(judge)
        hole++;
    //If it is determined as true, and the full pit position of the map operated by the
player is pushed into a box again, hole will be 0.
    if(judge&&temp[x][y]==7)
        hole=0;
    //Every time the keyboard keys are raised, it is necessary to determine whether
there are pits on the map. If there are pits, the game is not over.
    for(int i=0;i<row;i++)
    {
        for (int j=0;j<col;j++)
        {
            if(temp[i][j]==8)
                hole++;
        }
    }
```

7.6 Game over

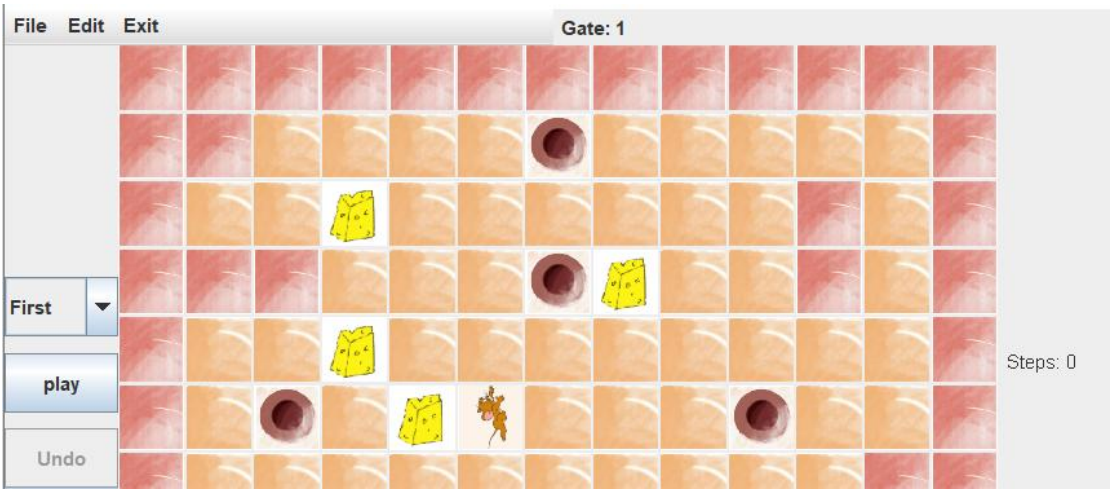
When the system judges that the game is over, if the player clicks OK, he or she will enter the next level. If he or she clicks OK, he or she will play this level again. If he or she clicks Cancel, he or she will not be processed.



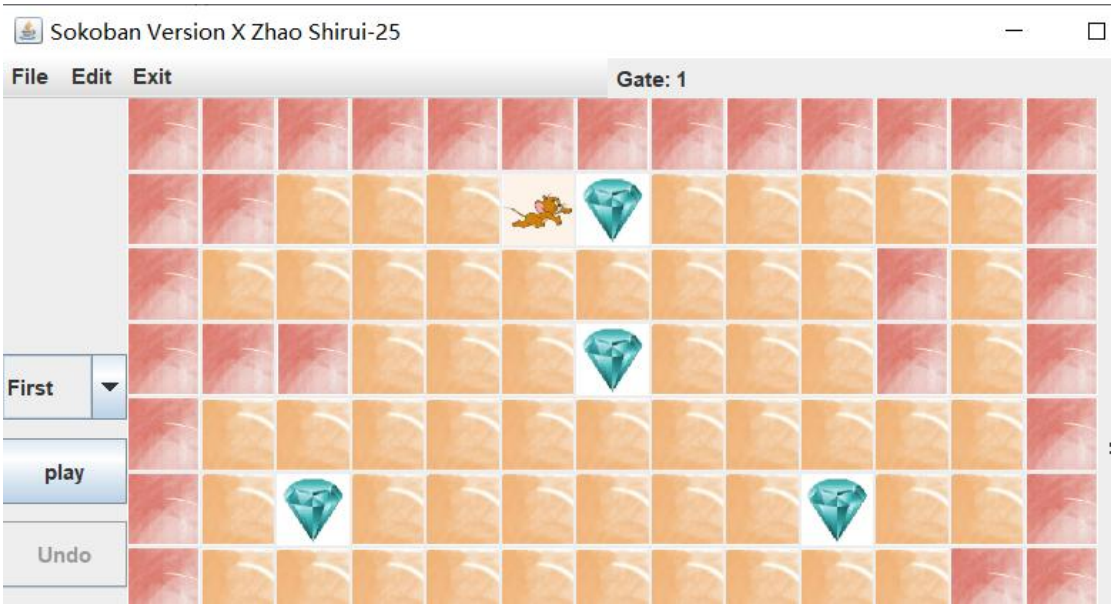
If the player click Yes, the user will enter the next level.



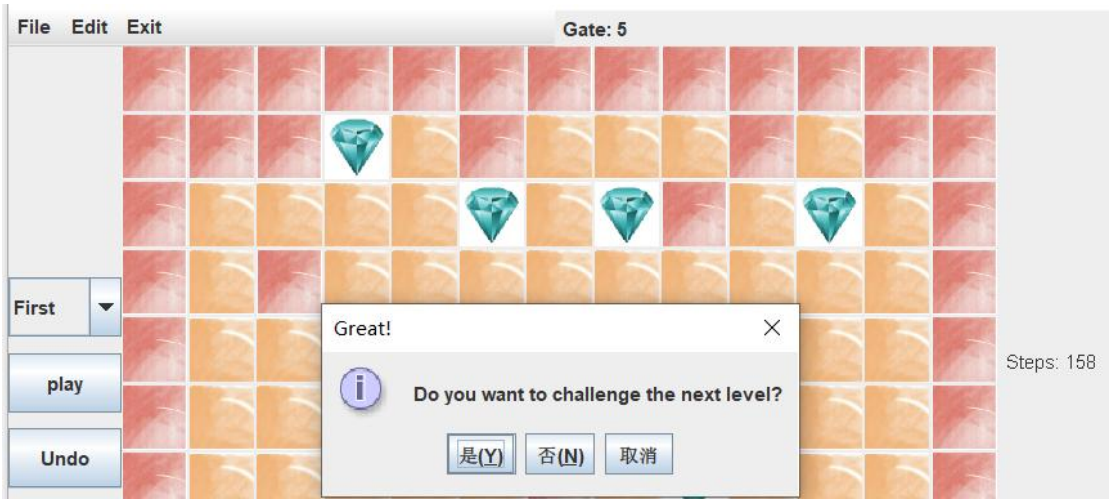
If player click No, this level will be played again.

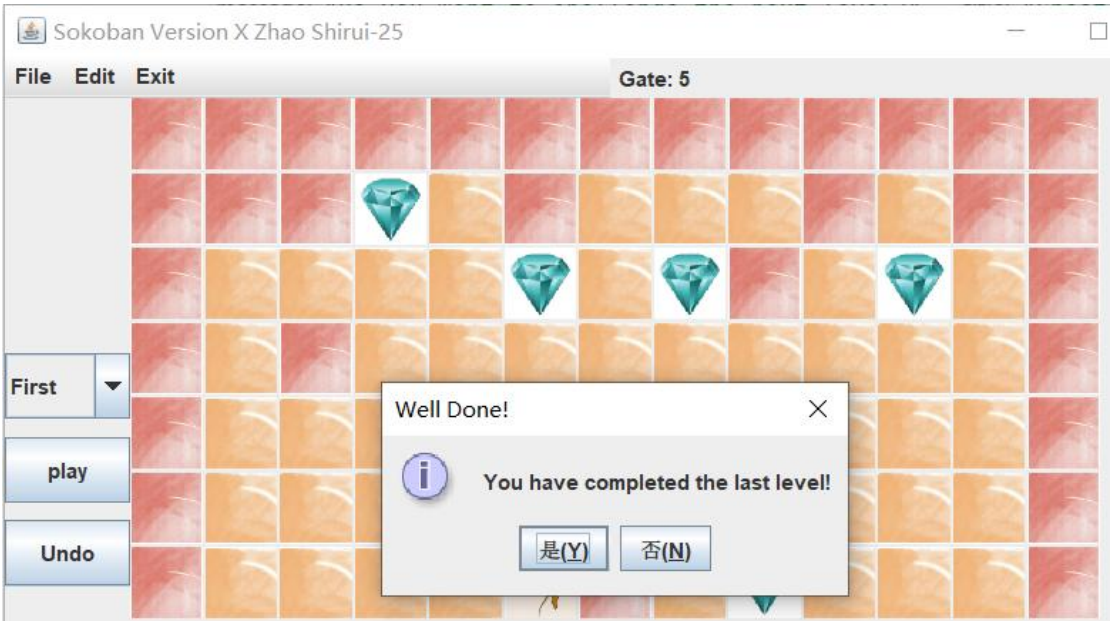


If player click Cancel, the user will stay in this interface.



If the fifth level has been completed, click Yes again to display clearance.






```
//Press the key to lift the monitoring function.
// When the pressed key is lifted,
// judge whether there is any box on the map that has been pushed into the pit.
// If there is, the game will not be ended.
// If not, the user will be prompted to complete
// the game and can choose whether to enter the next level.
@Override
public void keyReleased(KeyEvent e) {
//Set the value of the hole to 0.
    hole=0;
    //If it is determined to be true, hole first increases by 1, that is, there are still boxes not pushed into the pit.
    if(judge)
        hole++;
    //If it is determined as true, and the full pit position of the map operated by the player is pushed into a box again, hole will be 0.
    if(judge&&temp[x][y]==7)
        hole=0;
    //Every time the keyboard keys are raised, it is necessary to determine whether there are pits on the map. If there are pits, the game is not over.
    for(int i=0;i<row;i++)
    {
        for (int j=0;j<col;j++)
        {
            if(temp[i][j]==8)
                hole++;
        }
    }
```

```
        //If all the pits on the map are full, it will be judged as clearance and the
clearance information will be sent to the player.
        if(hole==0)
        {
            int result=JOptionPane.showInternalConfirmDialog(null,

                    "Do you want to challenge the next level?", "Great!",

                    JOptionPane.YES_NO_CANCEL_OPTION,

                    JOptionPane.INFORMATION_MESSAGE);
                //If the player clicks OK, it will enter the next level. If this level is the
last one, you will be prompted that all the games have passed.
            if(result==JOptionPane.OK_OPTION)
            {
                gate++;
                jgate.setText("Gate: "+gate);
                if(gate>5)
                {
                    gate=5;
                    jgate.setText("Gate: "+gate);
                    JOptionPane.showInternalConfirmDialog(null,"You have
completed the last level!",
                                    "Well Done!",JOptionPane.YES_OPTION,
JOptionPane.INFORMATION_MESSAGE);
                }
                try {
                     init(gate);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }//If you click No, this level will be played again. If you click Cancel,
nothing will be done.
            else if(result==JOptionPane.NO_OPTION)
            {
                try {
                    init(gate);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
            undo.clear();
            step=0;
        }
```

}

8. History Version

Filename	File Version	Date	Type	Change
Outcome2v1.doc	Version 1	2022/12/20	doc	The implementation of the basic functions of sokoban is recorded.
Outcome2v2.doc	Version 2	2022/12/21	doc	On the basis of v1, the setting of the fourth and fifth special maps has been added.
Outcome2v3.doc	Version 3	2022/12/22	doc	Added code for Undo function.
Outcome2v4.doc	Version 4	2022/12/22	pdf	Convert the written file to pdf format.

9. References

Old, S, (2020) *How to use Jframe* ,CSDN [online]. Available from: https://blog.csdn.net/qq_45867029/article/details/105372888 (Accessed: 2022/9/19).

Awak, N (2022) *Copy code to word to remove black background in IDEA, Pycharm and other environments* ,CSDN[online]. Available from: https://blog.csdn.net/weixin_43334838/article/details/125496897 (Accessed: 2022/9/19).

Jun, B (2022) *Confirmation dialog in JOptionPane*, CSDN[online]. Available from: https://blog.csdn.net/qq_57907966/article/details/124712979 (Accessed: 2022/9/20).

Guo, Z (2008) *Common keyboard ASC II code*, CSDN[online]. Available from: https://blog.csdn.net/guozi_upc/article/details/3523159?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-1-3523159-blog-103969632.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-1-3523159-blog-103969632.pc_relevant_default&utm_relevant_index=2 (Accessed: 2022/10/20).

Lzn, C(2012) *Java AudioClip Plays Audio* ,CSDN[online]. Available from: https://blog.csdn.net/china_lzn/article/details/7694701 (Accessed: 2022/10/20).

Wan, X(2019), *Writing relative paths in java* ,CSDN[online]. Available from: https://blog.csdn.net/qq_41856633/article/details/89715920 (Accessed: 2022/11/21).

Yu, Z(2021), *Set grid layout in java_ JAVA Grid Layout Manager* ,CSDN[online]. Available from: https://blog.csdn.net/weixin_35508482/article/details/114186140 (Accessed: 2022/11/21).

Xiao, L(2016) *Java JFrame grid layout,* CSDN[online]. Available from: https://blog.csdn.net/qq_21478795/article/details/51547188 (Accessed: 2022/12/20).

Pcuty, P(2018) *Java [22] JFrame three layouts and Jpanel,* CSDN[online]. Available from: https://blog.csdn.net/qq_38125626/article/details/81282687?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2~default~CTRLIST~Rate-1-81282687-blog-51547188.pc_relevant_multi_platform_whitelistv3&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2~default~CTRLIST~Rate-1-81282687-blog-51547188.pc_relevant_multi_platform_whitelistv3&utm_relevant_index=1 (Accessed: 2022/12/20).