Outcome 2 Report

Class: 16SD

Name: Zhao Jichen

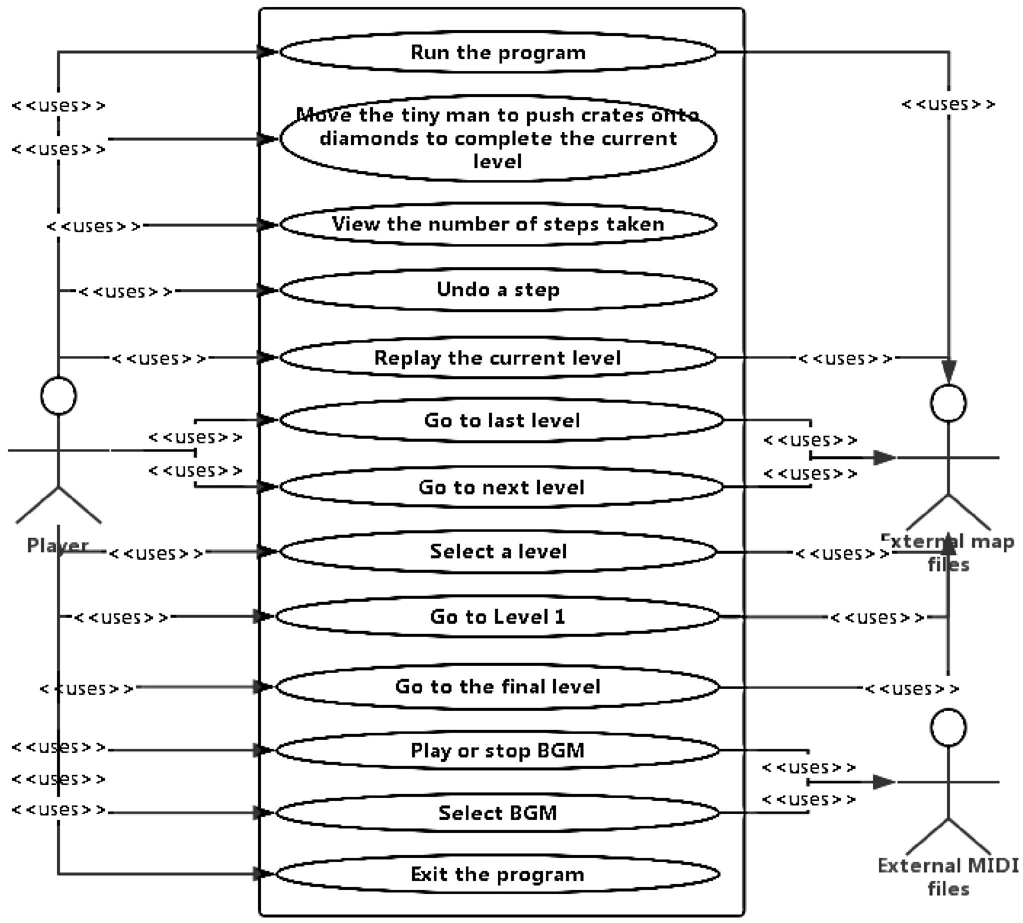
SCN: 187115469

Contents

1. UML

1.1. Use Case Diagram

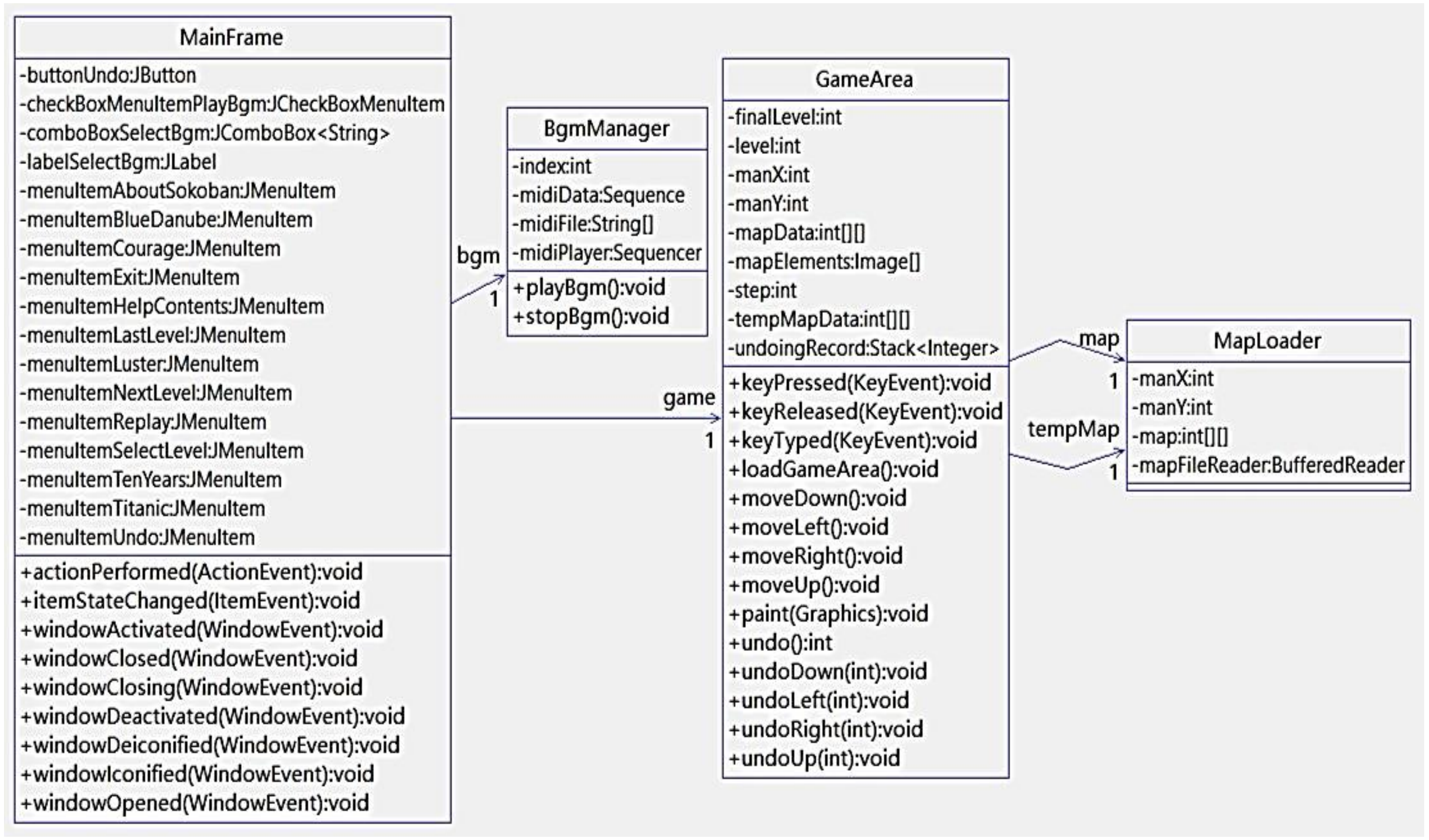
The following use case diagram (Picture 1.1 – 1) indicates use cases of the game Sokoban.



Picture 1.1 – 1

1.2. Class Diagram

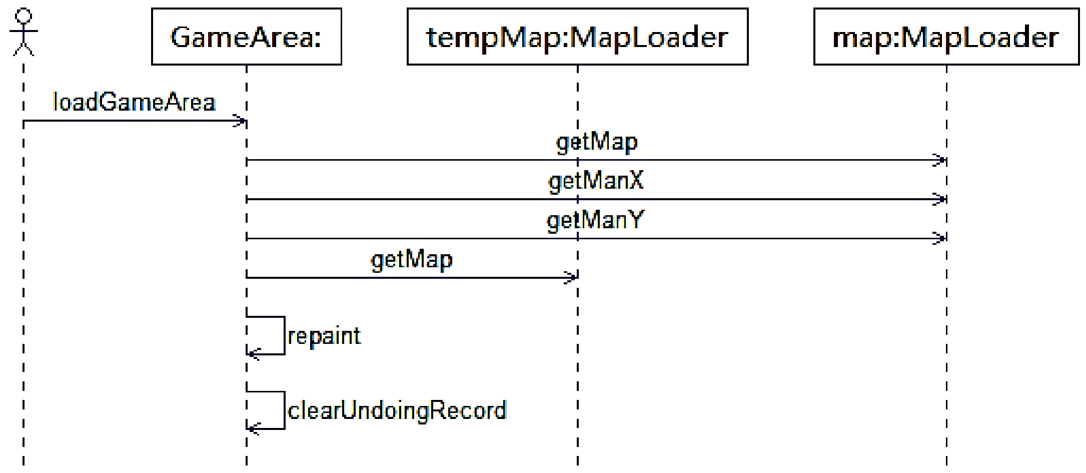
It is planned to build several classes to implement the functions of controlling background music, selecting a level, moving crates, etc. The following class diagram (Picture 1.2 – 1) indicates the static relationships among these classes.


MainFrame
-buttonUndo:JButton
-checkBoxMenuItemPlayBgm:JCheckBoxMenuItem
-comboBoxSelectBgm:JComboBox<String>
-labelSelectBgm:JLabel
-menuItemAboutSokoban:JMenuItem
-menuItemBlueDanube:JMenuItem
-menuItemCourage:JMenuItem
-menuItemExit:JMenuItem
-menuItemHelpContents:JMenuItem
-menuItemLastLevel:JMenuItem
-menuItemLuster:JMenuItem
-menuItemNextLevel:JMenuItem
-menuItemReplay:JMenuItem
-menuItemSelectLevel:JMenuItem
-menuItemTenYears:JMenuItem
-menuItemTitanic:JMenuItem
-menuItemUndo:JMenuItem
+actionPerformed(ActionEvent):void
+itemStateChanged(ItemEvent):void
+windowActivated(WindowEvent):void
+windowClosed(WindowEvent):void
+windowClosing(WindowEvent):void
+windowDeactivated(WindowEvent):void
+windowDeiconified(WindowEvent):void
+windowIconified(WindowEvent):void
+windowOpened(WindowEvent):void
bgm
1
BgmManager
-index:int
-midiData:Sequence
-midiFile:String[]
-midiPlayer:Sequencer
+playBgm():void
+stopBgm():void
game
1
GameArea
-finalLevel:int
-level:int
-manX:int
-manY:int
-mapData:int[][]
-mapElements:Image[]
-step:int
-tempMapData:int[][]
-undoingRecord:Stack<Integer>
+keyPressed(KeyEvent):void
+keyReleased(KeyEvent):void
+keyTyped(KeyEvent):void
+loadGameArea():void
+moveDown():void
+moveLeft():void
+moveRight():void
+moveUp():void
+paint(Graphics):void
+undo():int
+undoDown(int):void
+undoLeft(int):void
+undoRight(int):void
+undoUp(int):void
map
1
tempMap
1
MapLoader
-manX:int
-manY:int
-map:int[][]
-mapFileReader:BufferedReader

Picture 1.2 – 1

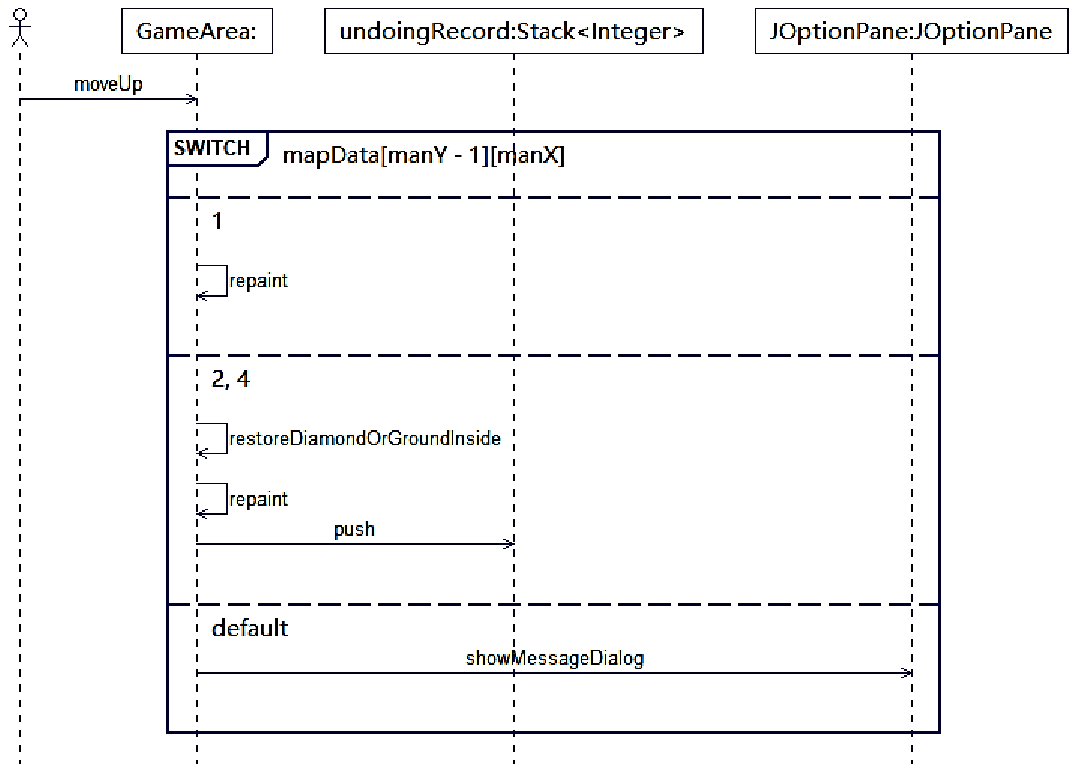
1.3. Sequence Diagrams

To load the game area, the sequence is shown in Picture 1.3 – 1.

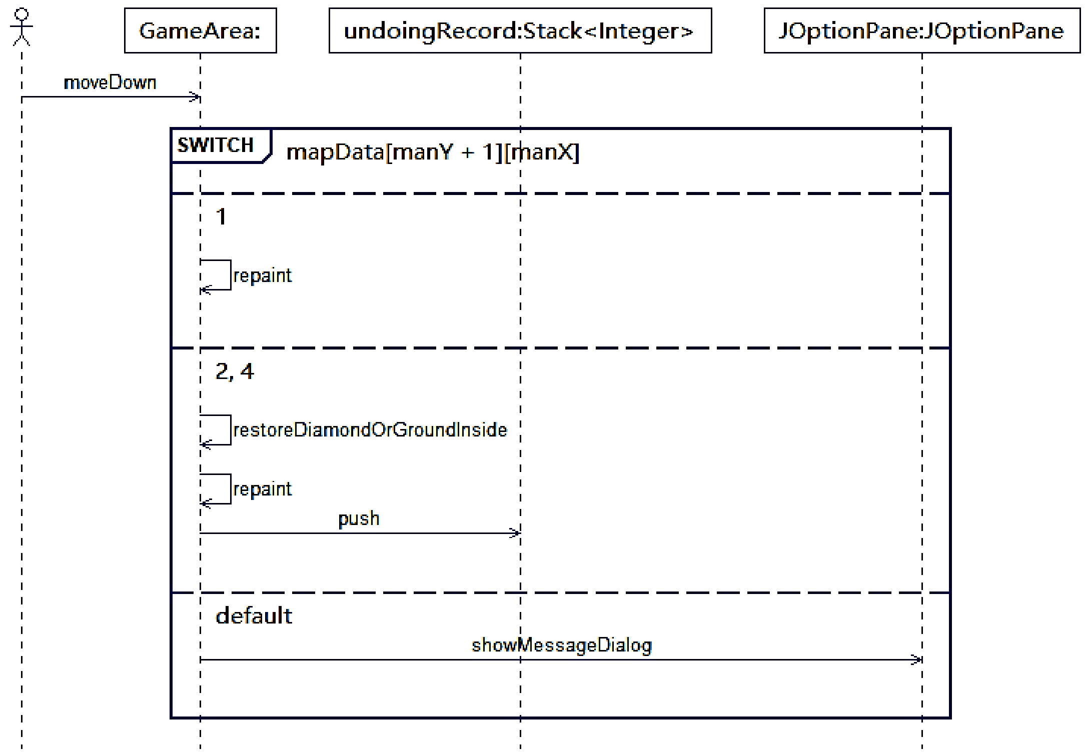


Picture 1.3 – 1

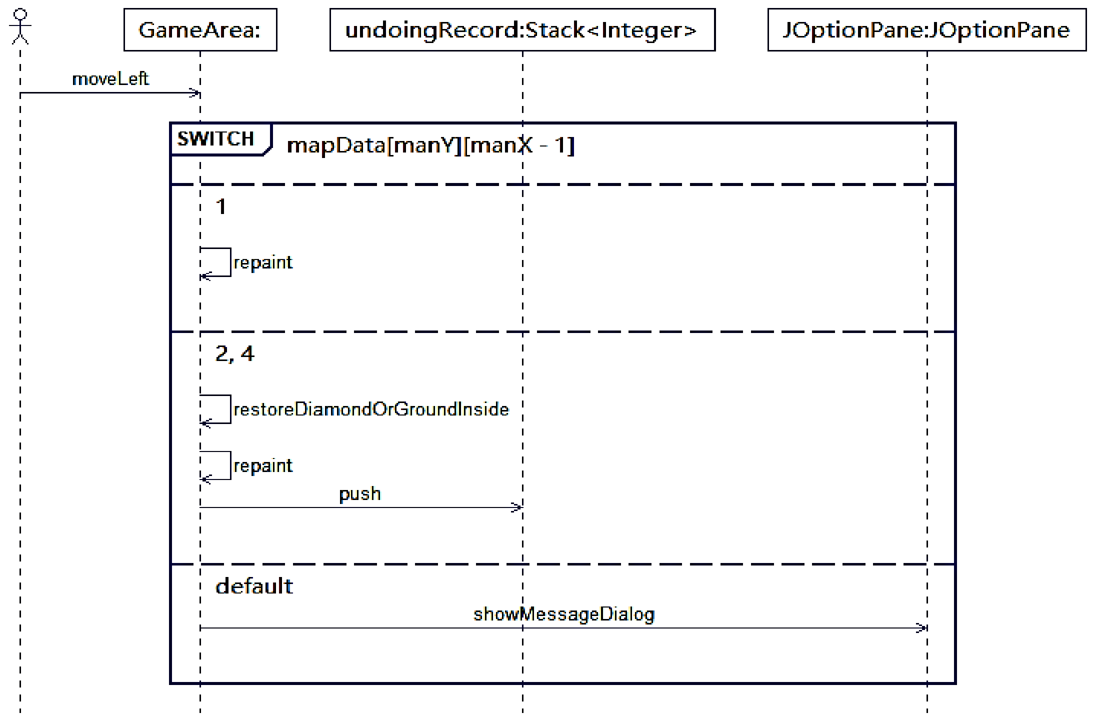
To control the tiny man to move up / down / left / right, the sequences are respectively shown in Pictures 1.3 – 2, 1.3 – 3, 1.3 – 4, and 1.3 – 5.



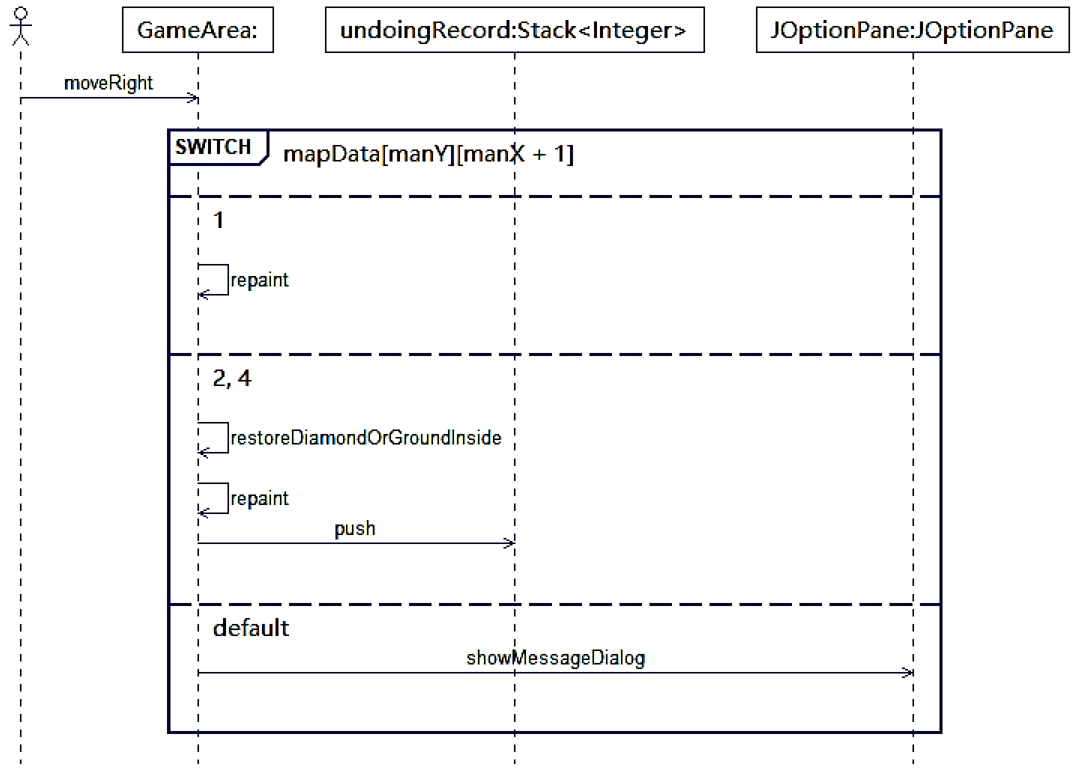
Picture 1.3 – 2



Picture 1.3 – 3

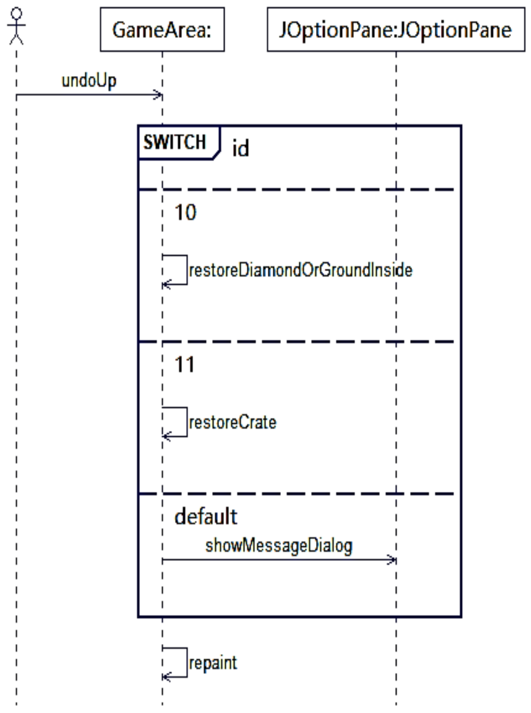


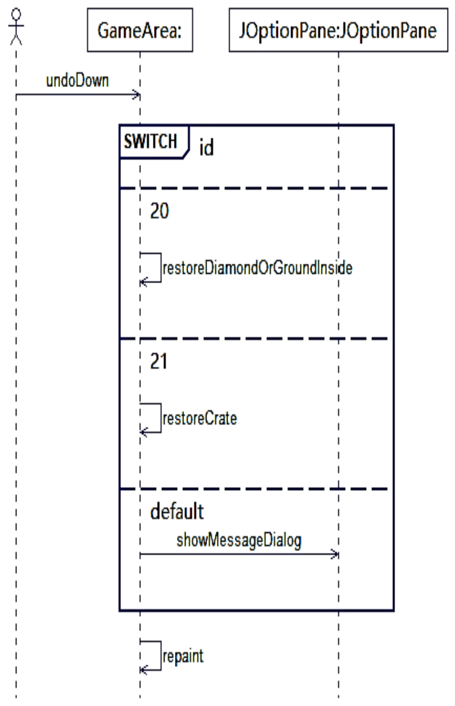
Picture 1.3 – 4



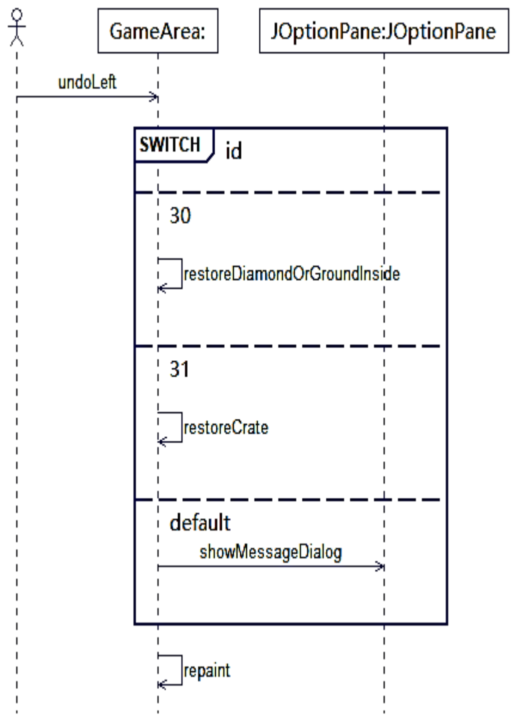
Picture 1.3 – 5

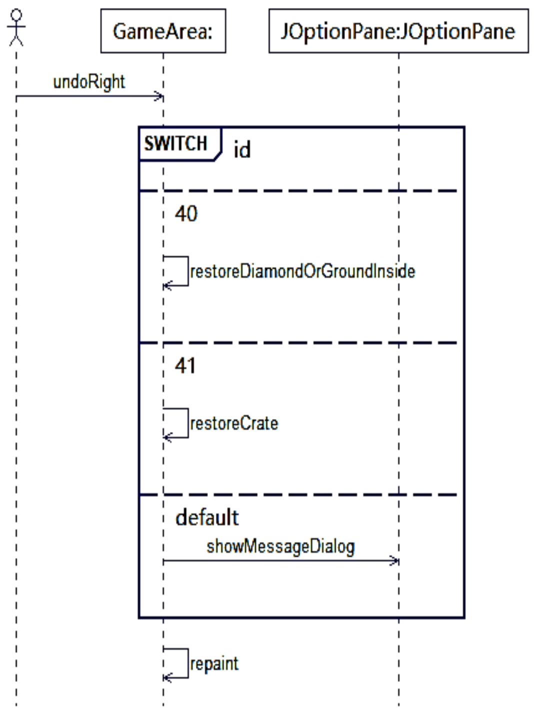
To undo a step, there are also 4 sequences as shown in Pictures 1.3 – 6, 1.3 – 7, 1.3 – 8, and 1.3 – 9.





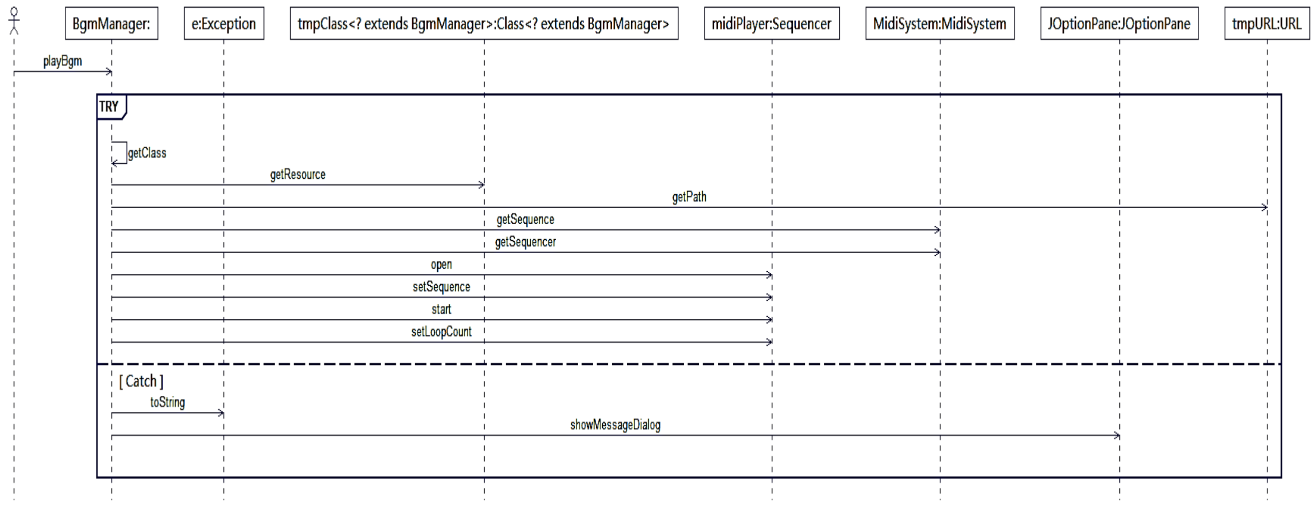
Picture 1.3 – 6 Picture 1.3 – 7



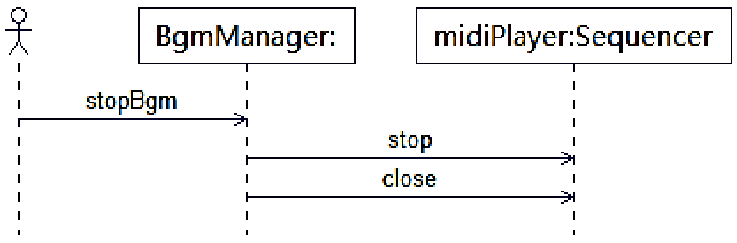


Picture 1.3 – 8 Picture 1.3 – 9

To play or stop BGM, the sequences are shown in Pictures 1.3 – 10 and 1.3 – 11.



Picture 1.3 – 10



Picture 1.3 – 11

2. Requirement Specification

2.1. Functional Requirements

- Controlling the tiny man by pressing directive keys (↑ / ↓ / ← / →)

  The game Sokoban should allow the player to control the tiny man with directive keys on the keyboard. For example, when the player presses the UP key, the tiny man may move up a step.

- Displaying the current step

  The current step should be displayed in real time.

- Undoing steps

  The game should provide the function of undoing steps. When the current step is 0, the player cannot use the function.

- Harder and harder levels from Level 1 to Level 5

  The game provides 5 levels, and the difficulty increases gradually by generating a more and more complex map.

- Displaying the current level

  The current level should be displayed in real time.

- Logic for passing a level

The game has the ability of judging if the player passes the current level. If it is passed, a message box should be displayed to allow the player to select whether to go to next level or not. Specially, if the final level is passed, game over, or stay at the current level according to the player's selection.

- Going to last level

  The player can request to go to last level. When the player is at Level 1, this function is disabled.

- Going to next level

  The player can request to go to next level. When the player is at the final level, this function is disabled.

- Going to Level 1

  The player can request to go to Level 1.

- Going to the final level

  The player can request to directly go to the final level.

- Selecting a level

  The player can enter an integer from 1 to 5 each of which represents a level to go to the level selected. If the player enters illegal input (for example, special characters) or just nothing after trimming, the game stays at the current level.

- Replaying the current level

  The player can request to replay the current level.

- Playing or stopping BGM

  In a default situation, BGM named Blue Danube is played as long as the program is running. The player can stop it.

- Selecting BGM

  The player can select BGM from a BGM list.

- Help contents

The game provides simple help contents for the player.

- About Sokoban

  The game should have its version, author, copyright (Retro Games), etc.

- Confirmation for exiting the program

  As long as the player tries to close the main frame, confirmation should be displayed.

2.2. Non-functional Requirements

- Reliability

  In order to minimise troubleshooting and enhance safety, the logical design of the game must be mature and the code must be high-quality.

- Easy-going

  A concise interface and an easy-understanding set of operations of the game are required so that the game is easy for the player to learn to use.

- System requirements

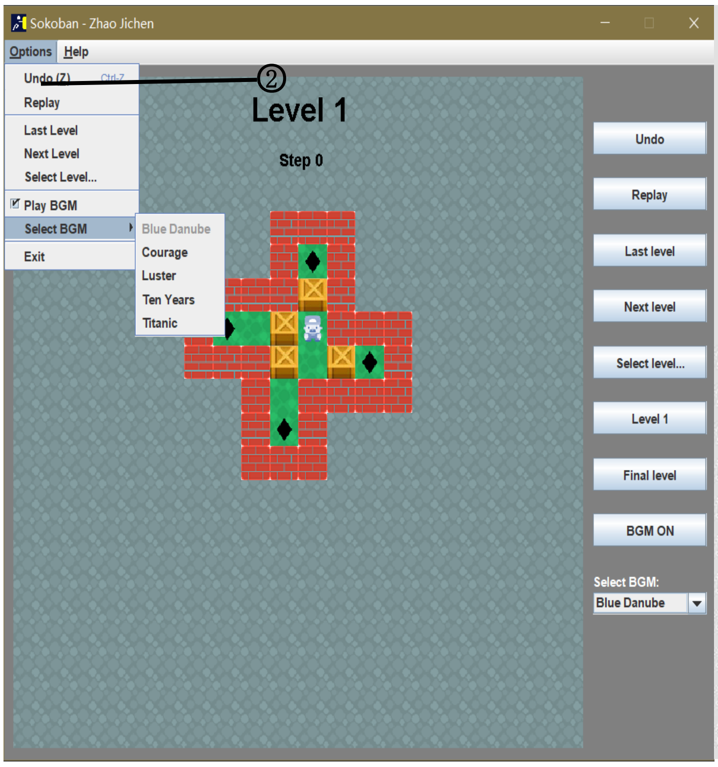
  Windows XP / 7 / Vista / 8 / 8.1 / 10 are recommended.

  Since this is a small Sokoban and personal computers commonly have a good performance nowadays, there is no strict requirement for a processor, the memory, the graphics, and the storage.

  What is really important is that a JVM is required and the version of JVM System Library is at least JavaSE-1.8. JavaSE-11 is strongly recommended because the program is developed using Eclipse 2018-09 with JavaSE-11.

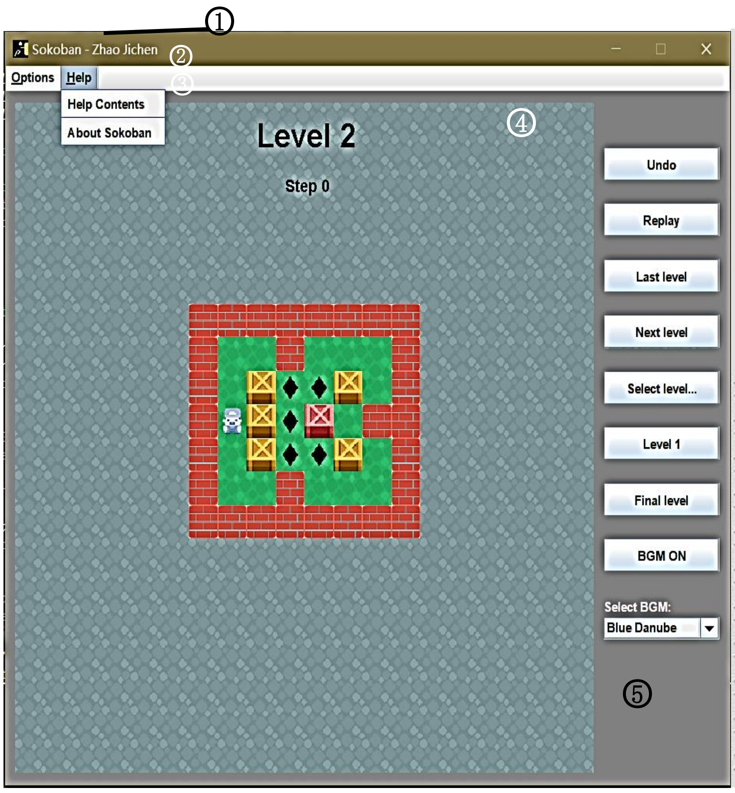
3. UI Introduction

The game has 5 levels, and there is certainly a map for each level. The user interfaces of 5 levels are shown in Pictures 3.0 – 1, 3.0 – 2, 3.0 – 3, 3.0 – 4, and 3.0 – 5.



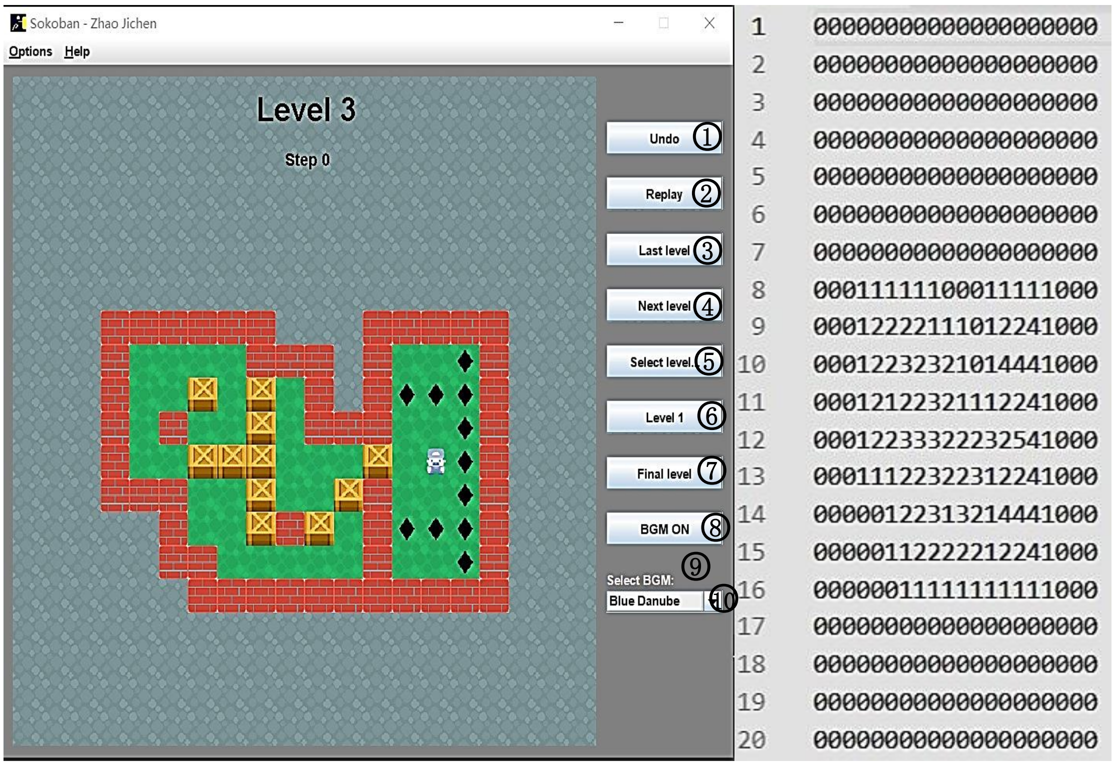
```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000011100000000
00000000014100000000
00000001111310000000
00000001423511100000
00000001113234100000
00000000012111100000
00000000014100000000
00000000011100000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Picture 3.0 – 1



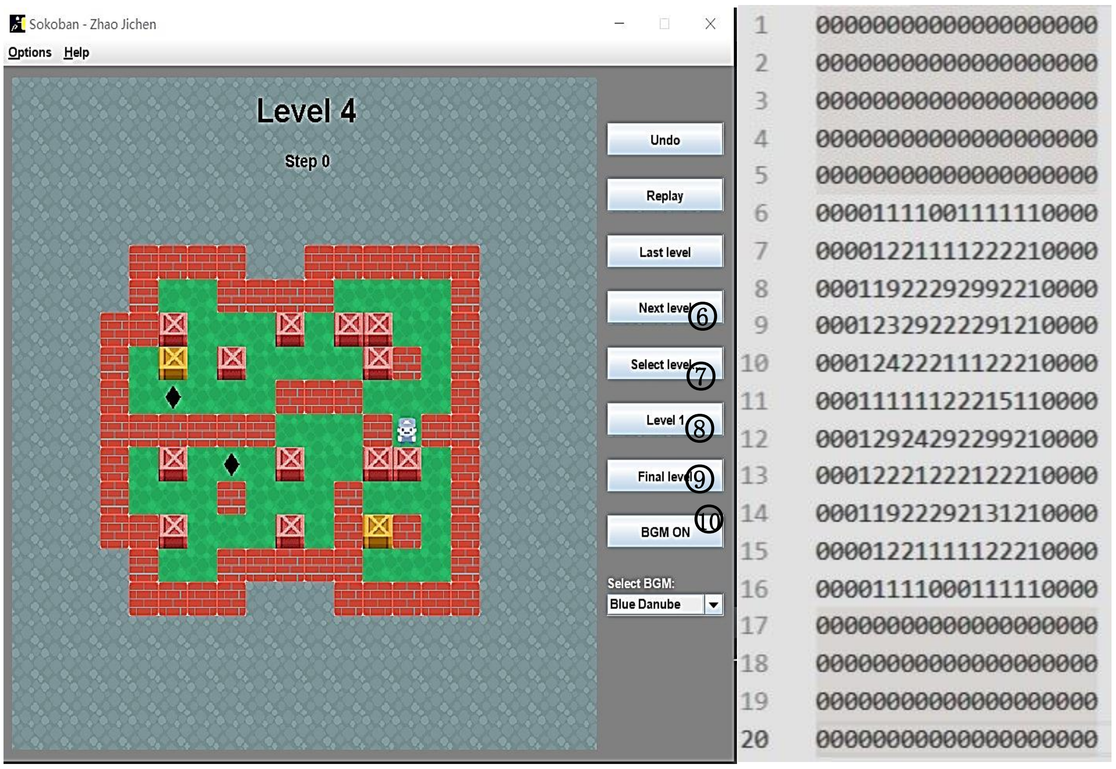
```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000001111111100000
00000001221222100000
00000001234432100000
00000001534921100000
00000001234432100000
00000001221222100000
00000001111111100000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Picture 3.0 – 2



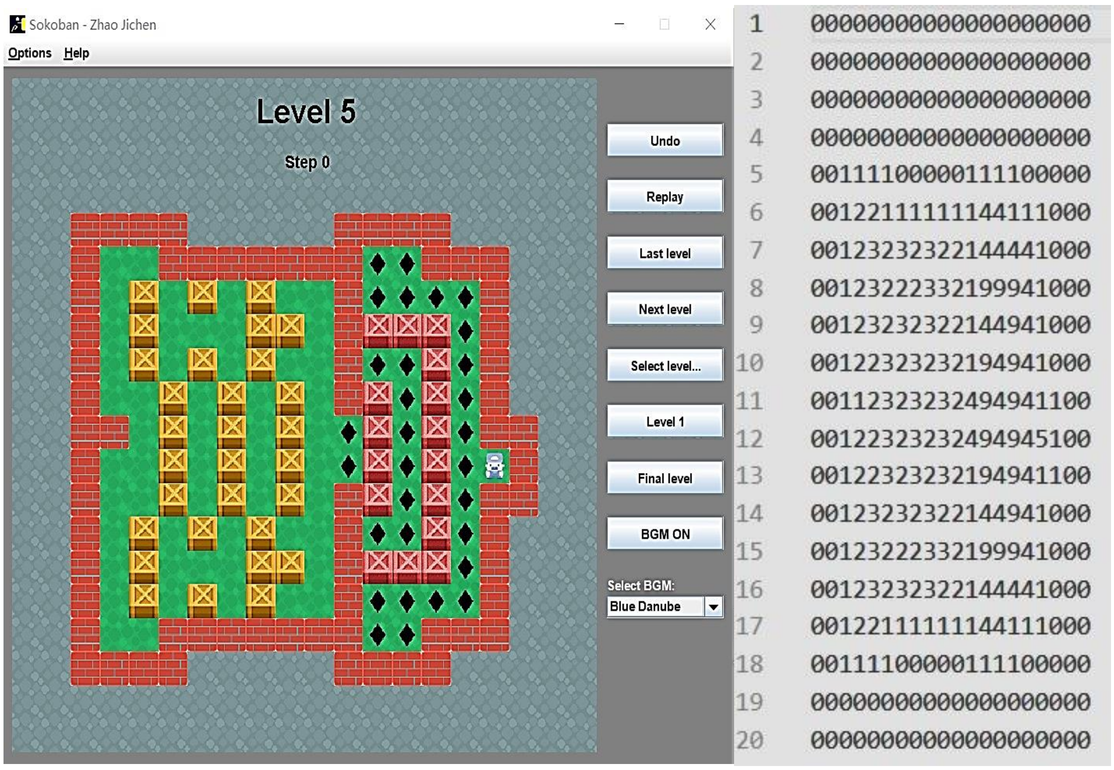
```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00011111100011111000
00012222111012241000
00012232321014441000
00012122321112241000
00012233322232541000
00011122322312241000
00000122313214441000
00000112222212241000
00000011111111111000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Picture 3.0 – 3



```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00001111001111110000
00001221111222210000
00011922292992210000
00012329222291210000
00012422211122210000
00011111122215110000
00012924292299210000
00012221222122210000
00011922292131210000
00001221111122210000
00001111000111110000
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
```

Picture 3.0 – 4



```
00000000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
00111100000111100000
00122111111144111000
00123232322144441000
00123222332199941000
00123232322144941000
00122323232194941000
00112323232494941100
00122323232494945100
00122323232194941100
00123232322144941000
00123222332199941000
00123232322144441000
00122111111144111000
00111100000111100000
00000000000000000000
00000000000000000000
```

Picture 3.0 – 5

The name of components used, their instance names, and their instructions are listed as follows.

Picture 3.0 – 1 (map file: 1.map)			
Index	Component	Instance name	Instruction
1	JMenuBar	menuBarMainFrame	The menu bar on the main frame including the menus Options and Help.
2	JMenu	menuOptions	The menu Options.
3	JMenuItem	menuItemUndo	A menu item in the menu Options. Undo a step.
4		menuItemReplay	A menu item in the menu Options. Replay the current level.
5		menuItemLastLevel	A menu item in the menu Options. Go

			to last level.
6		menuItemNextLevel	A menu item in the menu Options. Go to next level.
7		menuItemSelectLevel	A menu item in the menu Options. Select a level from Level 1 to Level 5.
8	JCheckBoxMenuItem	checkBoxMenuItemPlayBgm	A check box menu item in the menu Options. Play or stop BGM.
9	JMenu	subMenuSelectBgm	The sub menu Select BGM in the menu Options. Select BGM from the BGM list.
10	JMenuItem	menuItemBlueDanube	A menu item in the sub menu Select BGM of the menu Options. Default BGM – Blue Danube.
11		menuItemCourage	A menu item in the sub menu Select BGM of the menu Options. Play BGM selected – Courage.
12		menuItemLuster	A menu item in the sub menu Select BGM of the menu Options. Play BGM selected – Luster.
13		menuItemTenYears	A menu item in the sub menu Select BGM of the menu Options. Play BGM selected – Ten Years.
14		menuItemTitanic	A menu item in the sub menu Select BGM of the menu

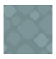
			Options. Play BGM selected – Titanic.
15		menuItemExit	A menu item in the menu Options. Show the confirmation whether to exit the program or not.
Picture 3.0 – 2 (map file: 2.map)			
1	JMenu	menuHelp	The menu Help.
2	JMenuItem	menuItemHelpContents	A menu item in the menu Help. Show brief help contents.
3		menuItemAboutSokoban	A menu item in the menu Help. Show info about Sokoban, such as its version, author, copyright, etc.
4	JPanel	game (class GameArea extends JPanel)	A panel for loading a map to generate the game area.
5	JFrame	- (class MainFrame extends JFrame)	The main frame displayed to the player.
	Container	containerMainFrame	The container on the main frame containing all components.
Picture 3.0 – 3 (map file: 3.map)			
1	JButton	buttonUndo	Undo a step.
2		buttonReplay	Replay the current level.
3		buttonLastLevel	Go to last level.
4		buttonNextLevel	Go to next level.
5		buttonSelectLevel	Select a level from Level 1 to Level 5.
6		buttonFirstLevel	Go to Level 1.
7		buttonFinalLevel	Go to the final level.
8		buttonBgmOnOrOff	Play or stop BGM.

9	JLabel	labelSelectBgm	A notice for the following combo box.
10	JComboBox	comboBoxSelectBgm	A combo box containing the BGM list which allows the player to select BGM.
Picture 3.0 – 4 (map file: 4.map)			
Picture 3.0 – 5 (map file: 5.map)			

4. Map Instruction

- Map elements

There are at most 10 kinds of elements in a map. Each image represents an element, and they are listed as follows. Both the height and the width of an image are 30 pixels.

Image	Index	Element
	0	The ground outside.
	1	The wall.
	2	The ground inside.
	3	A crate.
	4	A diamond.
	5	The front of the tiny man.
	6	The left of the tiny man.
	7	The right of the tiny man.
	8	The back of the tiny man.
	9	A crate on a diamond.

- Map size

Both the height and the width of a map are 20 elements. That is, there are 20 elements in each row or line.

5. Partial Code

5.1. Main Functions for the Menu Bar and the Option Area

The following code is a part of the java class file “MainFrame.java”.

```
    @Override
    // it is the method which must be implemented to respond to the user click on a
specified button or menu item that is in the interface ActionListener
    public void actionPerformed(ActionEvent e)
    {
        int level = game.getLevel(); // call the specified method in GameArea to get
the current level
        int finalLevel = game.getFinalLevel(); // call the specified method in
GameArea to get the final level

        // undo a step
        if (e.getSource() == buttonUndo || e.getSource() == menuItemUndo)
        {
            // call the specified method to check if the undoing record is empty
            if (game.isUndoingRecordEmpty())
                JOptionPane.showMessageDialog(this, "No step to undo! The tiny
man does not move.", "Sokoban - Zhao Jichen", JOptionPane.WARNING_MESSAGE);
            else
            {
                // call the specified method in class GameArea to undo a step
represented by ID
                switch (game.undo())
                {
                case 10:
                    game.undoUp(10); // call the specified method in class
GameArea to undo a move-up step
                    break;

                case 11:
                    game.undoUp(11); // call the specified method in class
GameArea to undo a move-up step
```

```
                    break;

                case 20:
                    game.undoDown(20); // call the specified method in class
GameArea to undo a move-down step
                    break;

                case 21:
                    game.undoDown(21); // call the specified method in class
GameArea to undo a move-down step
                    break;

                case 30:
                    game.undoLeft(30); // call the specified method in class
GameArea to undo a move-left step
                    break;

                case 31:
                    game.undoLeft(31); // call the specified method in class
GameArea to undo a move-left step
                    break;

                case 40:
                    game.undoRight(40); // call the specified method in class
GameArea to undo a move-right step
                    break;

                case 41:
                    game.undoRight(41); // call the specified method in class
GameArea to undo a move-right step
                    break;

                default:
                    JOptionPane.showMessageDialog(this, "Error! Something
wrong during the undoing process.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
                } // end switch-case
            } // end if...else

            game.requestFocus();
        }
        // replay the current level
        else if (e.getSource() == buttonReplay || e.getSource()== menuItemReplay)
        {
```

```
        game.loadGameArea(); // call the specified method in GameArea to load
the game area
        game.requestFocus();
    }
    // go to last level
    else if (e.getSource() == buttonLastLevel || e.getSource() ==
menuItemLastLevel)
    {
        if (level - 1 < 1)
        {
            JOptionPane.showMessageDialog(this, "No last level! You are now
at Level 1.", "Sokoban - Zhao Jichen", JOptionPane.WARNING_MESSAGE);
            game.requestFocus();
        }
        else
        {
            game.setLevel(--level); // call the specified method in class
GameArea to set the current level
            game.loadGameArea(); // call the specified method in class
GameArea to load the game area
            game.requestFocus();
        } // end if...else
    }
    // go to next level
    else if (e.getSource() == buttonNextLevel || e.getSource() ==
menuItemNextLevel)
    {
        if (level + 1 > finalLevel )
        {
            JOptionPane.showMessageDialog(this, "No next level! You are now
at the final level.", "Sokoban - Zhao Jichen", JOptionPane.WARNING_MESSAGE);
            game.requestFocus();
        }
        else
        {
            game.setLevel(++level); // call the specified method in class
GameArea to set the current level
            game.loadGameArea(); // call the specified method in class
GameArea to load the game area
            game.requestFocus();
        } // end if...else
    }
    // select a level
    else if (e.getSource() == buttonSelectLevel || e.getSource() ==
menuItemSelectLevel)
    {
        String selection = JOptionPane.showInputDialog(this, "Please enter the level you want to select (1 - " + finalLevel + "):", "Sokoban - Zhao Jichen", JOptionPane.PLAIN_MESSAGE);

        if (selection != null)
        {
            try
            {
                int levelSelected = Integer.parseInt(selection.trim());

                if (levelSelected > finalLevel || levelSelected < 1)
                {
                    JOptionPane.showMessageDialog(this, "No such level! Not an integer from 1 to " + finalLevel + "?", "Sokoban - Zhao Jichen", JOptionPane.WARNING_MESSAGE);
                    game.requestFocus();
                }
                else
                {
                    game.setLevel(levelSelected); // call the specified method in class GameArea to set the current level
                    game.loadGameArea(); // call the specified method in class GameArea to load the game area
                    game.requestFocus();
                } // end if...else
            }
            catch (NumberFormatException exception)
            {
                JOptionPane.showMessageDialog(this, "Illegal input! Please enter an integer from 1 to " + finalLevel + ".\n(" + exception.toString() + ")", "Sokoban - Zhao Jichen", JOptionPane.ERROR_MESSAGE);
                game.requestFocus();
            } // end try...catch
        }
        else
            game.requestFocus();
    }
    // go to Level 1
    else if (e.getSource() == buttonFirstLevel)
    {
        game.setLevel(1); // call the specified method in class GameArea to set the current level
```

```
                game.loadGameArea(); // call the specified method in class GameArea to
load the game area
                game.requestFocus();
            }
            // go to the final level
            else if (e.getSource() == buttonFinalLevel)
            {
                game.setLevel(finalLevel); // call the specified method in class GameArea
to set the current level
                game.loadGameArea(); // call the specified method in class GameArea to
load the game area
                game.requestFocus();
            }
            // play or stop BGM
            else if (e.getSource() == buttonBgmOnOrOff || e.getSource() ==
checkBoxMenuItemPlayBgm)
            {
                // call the specified method in class BgmManager to check if BGM is
played
                if (bgm.getPlayingStatus())
                {
                    bgm.stopBgm(); // call the specified method in class BgmManager to
stop BGM
                    buttonBgmOnOrOff.setText("BGM OFF");
                    buttonBgmOnOrOff.setToolTipText("Click on me to play BGM.");
                }
                else
                {
                    bgm.playBgm(); // call the specified method in class BgmManager to
load and play BGM
                    buttonBgmOnOrOff.setText("BGM ON");
                    buttonBgmOnOrOff.setToolTipText("Click on me to stop BGM.");
                } // end if...else

                game.requestFocus();
            }
            // select BGM of Index 0
            else if (e.getSource() == menuItemBlueDanube)
                comboBoxSelectBgm.setSelectedIndex(0);
            // select BGM of Index 1
            else if (e.getSource() == menuItemCourage)
                comboBoxSelectBgm.setSelectedIndex(1);
            // select BGM of Index 2
            else if (e.getSource() == menuItemLuster)
                comboBoxSelectBgm.setSelectedIndex(2);
            // select BGM of Index 3
            else if (e.getSource() == menuItemTenYears)
                comboBoxSelectBgm.setSelectedIndex(3);
            // select BGM of Index 4
            else if (e.getSource() == menuItemTitanic)
                comboBoxSelectBgm.setSelectedIndex(4);
            // exit the program
            else if (e.getSource() == menuItemExit)
            {
                actionBeforeExiting(); // call the specified method to ask the user to
confirm whether to exit the program or not
            }
            // help contents
            else if (e.getSource() == menuItemHelpContents)
                JOptionPane.showMessageDialog(this, "1. A menu bar and the option
area are provided for you to\n    undo a step, replay the current level, go to last level,
go\n    to next level, select a level, go to Level 1, go to the final\n    level, play or
stop BGM, and select BGM.\n2. Press ↑ / ↓ / ← / → to move the tiny man.\n3. To
expand the menu Options conveniently, press Alt + O.\n    To expand the menu Help
conveniently, press Alt + H.\n    To undo a step conveniently, press Ctrl + Z.", "Help
Contents", JOptionPane.PLAIN_MESSAGE);
            // about Sokoban
            else if (e.getSource() == menuItemAboutSokoban)
                JOptionPane.showMessageDialog(this,          "Sokoban\n\nVersion:
4.0\nAuthor: Zhao Jichen (SCN: 187115469)\n\nCopyright ©  Retro Games 2018",
"About Sokoban", JOptionPane.PLAIN_MESSAGE);
    } // end method actionPerformed

    @Override
    // it is the method which must be implemented to respond to the change of the item
selected in the combo box for selecting BGM that is in the interface ItemListener
    public void itemStateChanged(ItemEvent e)
    {
            switch (comboBoxSelectBgm.getSelectedIndex())
            {
            case 0:
                changeBgmInComboBox(0); // call the specified method to change BGM
played according to the item selected in the combo box
                menuItemBlueDanube.setEnabled(false);
                menuItemCourage.setEnabled(true);
                menuItemLuster.setEnabled(true);
                menuItemTenYears.setEnabled(true);
                menuItemTitanic.setEnabled(true);
```

```
                game.requestFocus();
                break;

            case 1:
                changeBgmInComboBox(1); // call the specified method to change BGM
played according to the item selected in the combo box
                menuItemBlueDanube.setEnabled(true);
                menuItemCourage.setEnabled(false);
                menuItemLuster.setEnabled(true);
                menuItemTenYears.setEnabled(true);
                menuItemTitanic.setEnabled(true);
                game.requestFocus();
                break;

            case 2:
                changeBgmInComboBox(2); // call the specified method to change BGM
played according to the item selected in the combo box
                menuItemBlueDanube.setEnabled(true);
                menuItemCourage.setEnabled(true);
                menuItemLuster.setEnabled(false);
                menuItemTenYears.setEnabled(true);
                menuItemTitanic.setEnabled(true);
                game.requestFocus();
                break;

            case 3:
                changeBgmInComboBox(3); // call the specified method to change BGM
played according to the item selected in the combo box
                menuItemBlueDanube.setEnabled(true);
                menuItemCourage.setEnabled(true);
                menuItemLuster.setEnabled(true);
                menuItemTenYears.setEnabled(false);
                menuItemTitanic.setEnabled(true);
                game.requestFocus();
                break;

            case 4:
                changeBgmInComboBox(4); // call the specified method to change BGM
played according to the item selected in the combo box
                menuItemBlueDanube.setEnabled(true);
                menuItemCourage.setEnabled(true);
                menuItemLuster.setEnabled(true);
                menuItemTenYears.setEnabled(true);
                menuItemTitanic.setEnabled(false);
                game.requestFocus();
                break;

            default:
                JOptionPane.showMessageDialog(this, "Error! Something wrong during the process of responding to the change of the item selected in the combo box for selecting BGM.\n(Unknown error)", "Sokoban - Zhao Jichen", JOptionPane.ERROR_MESSAGE);
        } // end switch-case
    } // end method itemStateChanged
```

5.2. Confirmation for Exiting the Program

The following code is a part of the java class file “MainFrame.java”.

```
    /**
     * Ask the user to confirm whether to exit the program or not.
     */
    private void actionBeforeExiting()
    {
        if (JOptionPane.showConfirmDialog(this, "Are you sure to stop playing Sokoban?", "Sokoban - Zhao Jichen", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
            System.exit(0); // completely exit the program
        else
            game.requestFocus();
    } // end method actionBeforeExiting

    @Override
    // it is the method which must be implemented to respond to the main frame to be closed that is in the interface WindowListener
    public void windowClosing(WindowEvent e)
    {
        actionBeforeExiting(); // call the specified method to ask the user to confirm whether to exit the program or not
    } // end method windowClosing
```

5.3. Main Functions for the Game Area

The following code is a part of the java class file “GameArea.java”.

```
    /**
```

```
	 * Non-parameter constructor that initialises the game area.
	 */
	public GameArea()
	{
		setVisible(true);
		setBounds(10, 10, 600, 600);
		setBackground(Color.WHITE);
		addKeyListener(this);

		mapElements = new Image[10];

		for (int counter = 0; counter < 10; counter++)
			mapElements[counter]                                    =
Toolkit.getDefaultToolkit().getImage(getClass().getResource("").getPath() + "/img/" +
counter + ".png");
	} // end constructor GameArea

	/**
	 * Clear undoing records.
	 */
	private void clearUndoingRecord()
	{
		undoingRecord.removeAllElements();
		step = 0;
	} // end method clearUndoingRecord

	/**
	 * Do specified actions if the current level is passed.
	 */
	private void actionForLevelPassed()
	{
		// call the specified method to check if the current level is passed
		if (isPassed())
		{
			if (level == finalLevel)
			{
				JOptionPane.showMessageDialog(this, "Congratulations! The final
level     passed!\nGame     over!",     "Sokoban     -     Zhao     Jichen",
JOptionPane.PLAIN_MESSAGE);

				if (JOptionPane.showConfirmDialog(this, "Are you sure to stop
playing Sokoban?", "Sokoban - Zhao Jichen", JOptionPane.YES_NO_OPTION) ==
JOptionPane.YES_OPTION)
					System.exit(0); // completely exit the program
```

```
            }
            else
            {
                if (JOptionPane.showConfirmDialog(this, "Congratulations! Level " + level + " passed!\nGo to next level?", "Sokoban - Zhao Jichen", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
                {
                    level++;
                    loadGameArea(); // call the specified method to load the game area
                } // end if
            } // end if...else
        } // end if
    } // end method actionForLevelPassed

    /**
     * Restore a diamond or the ground inside after the tiny man moves.
     */
    private void restoreDiamondOrGroundInside()
    {
        if (tempMapData[manY][manX] == 4 || tempMapData[manY][manX] == 9 )
            mapData[manY][manX] = 4;
        else
            mapData[manY][manX] = 2;
    } // end method restoreDiamondOrGroundInside

    /**
     * Restore a crate on a diamond or just a crate after the tiny man moves.
     */
    private void restoreCrate()
    {
        if (tempMapData[manY][manX] == 4 || tempMapData[manY][manX] == 9 )
            mapData[manY][manX] = 9;
        else
            mapData[manY][manX] = 3;
    } // end method restoreCrate

    // it is this method which can be called by method repaint that is invoked by Swing to draw components
    public void paint(Graphics g)
    {
        for (int i = 0; i < 20; i++)
            for (int j = 0; j < 20; j++)
                 g.drawImage(mapElements[mapData[j][i]], i * 30, j * 30, this);
        g.setColor(Color.BLACK);
        g.setFont(new Font("Arial", Font.BOLD, 30));
          g.drawString("Level " + level, 250, 40); // display the level info
          g.setFont(new Font("Arial", Font.BOLD, 15));
          g.drawString("Step " + step, 280, 80); // display the step info
    } // end method paint

    /**
      * Load the game area.
      */
    public void loadGameArea()
    {
        map = new MapLoader(level);
        tempMap = new MapLoader(level);
        mapData = map.getMap(); // call the specified method in class MapLoader to
get the map of a specified level
        manX = map.getManX(); // call the specified method in class MapLoader to
get X of the tiny man in the map
        manY = map.getManY(); // call the specified method in class MapLoader to
get Y of the tiny man in the map
        tempMapData = tempMap.getMap(); // call the specified method in class
MapLoader to get the map of a specified level
        repaint();
        clearUndoingRecord(); // call the specified method to clear undoing records
    } // end method loadGameArea

    @Override
    // it is the method which must be implemented to respond to the user press on a
specified key that is in the interface KeyListener
    public void keyPressed(KeyEvent e)
    {
        if (e.getKeyCode() == KeyEvent.VK_UP)
        {
            moveUp(); // call the specified method to try to move up a step
            actionForLevelPassed(); // call the specified method to do specified
actions if the current level is passed
        } // end if

        if (e.getKeyCode() == KeyEvent.VK_DOWN)
        {
            moveDown(); // call the specified method to try to move down a step
            actionForLevelPassed(); // call the specified method to do specified
actions if the current level is passed
```

```
        } // end if

        if (e.getKeyCode() == KeyEvent.VK_LEFT)
        {
            moveLeft(); // call the specified method to try to move left a step
            actionForLevelPassed(); // call the specified method to do specified
actions if the current level is passed
        } // end if

        if (e.getKeyCode() == KeyEvent.VK_RIGHT)
        {
            moveRight(); // call the specified method to try to move right a step
            actionForLevelPassed(); // call the specified method to do specified
actions if the current level is passed
        } // end if
    } // end method keyPressed

    /**
     * Check if the undoing record is empty.
     * @return The boolean value true if the undoing object is empty, or false if it is
full.
     */
    public boolean isUndoingRecordEmpty()
    {
        return undoingRecord.isEmpty();
    } // end method isUndoingRecordEmpty

    /**
     * Undo a step represented by ID.
     * @return ID which represents a kind of step.
     */
    public int undo()
    {
        return undoingRecord.pop();
    } // end method undo

    /**
     * Try to move up a step.
     */
    public void moveUp()
    {
        // the element above the tiny man before moving
        switch (mapData[manY - 1][manX])
        {
```

```
        case 1:
            mapData[manY][manX] = 8;
            repaint();
            break;

        case 2:
        case 4:
            restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
            mapData[manY - 1][manX] = 8;
            repaint();
            manY--;
            step++;
            undoingRecord.push(10);
            break;

        case 3:
        case 9:
            // the element above the element above the tiny man before moving
            if (mapData[manY - 2][manX] == 4)
            {
                restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                mapData[manY - 1][manX] = 8;
                mapData[manY - 2][manX] = 9;
                repaint();
                manY--;
                step++;
                undoingRecord.push(11);
            }
            else if (mapData[manY - 2][manX] == 2)
            {
                restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                mapData[manY - 1][manX] = 8;
                mapData[manY - 2][manX] = 3;
                repaint();
                manY--;
                step++;
                undoingRecord.push(11);
            }
            else
            {
                mapData[manY][manX] = 8;
```

```
                  repaint();
              } // end nested if...else
              break;

          default:
              JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of trying to move up a step.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
          } // end switch-case
      } // end method moveUp

      /**
        * Undo a move-up step.
        * @param id ID which represents a move-up step.
        */
      public void undoUp(int id)
      {
          switch (id)
          {
          case 10:
              restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
              break;

          case 11:
              restoreCrate(); // call the specified method to restore a crate on a diamond
or just a crate after the tiny man moves
              // the element above the tiny man before moving
              if (tempMapData[manY - 1][manX] == 4 || tempMapData[manY -
1][manX] == 9)
                  mapData[manY - 1][manX] = 4;
              else
                  mapData[manY - 1][manX] = 2;
              break;

          default:
              JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of undo moving up.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
          } // end switch-case

          mapData[manY + 1][manX] = 8; // the element below the tiny man
          repaint();
          manY++;
          step--;
      } // end method undoUp

      /**
        * Try to move down a step.
        */
      public void moveDown()
      {
          // the element below the tiny man before moving
          switch (mapData[manY + 1][manX])
          {
          case 1:
              mapData[manY][manX] = 5;
              repaint();
              break;

          case 2:
          case 4:
              restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
              mapData[manY + 1][manX] = 5;
              repaint();
              manY++;
              step++;
              undoingRecord.push(20);
              break;

          case 3:
          case 9:
              // the element below the element below the tiny man before moving
              if (mapData[manY + 2][manX] == 4)
              {
                  restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                  mapData[manY + 1][manX] = 5;
                  mapData[manY + 2][manX] = 9;
                  repaint();
                  manY++;
                  step++;
                  undoingRecord.push(21);
              }
              else if (mapData[manY + 2][manX] == 2)
              {
                  restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                  mapData[manY + 1][manX] = 5;
                  mapData[manY + 2][manX] = 3;
                  repaint();
                  manY++;
                  step++;
                  undoingRecord.push(21);
              }
              else
              {
                  mapData[manY][manX] = 5;
                  repaint();
              } // end nested if...else
              break;

          default:
              JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of trying to move down a step.\n(Unknown error)", "Sokoban - Zhao
Jichen", JOptionPane.ERROR_MESSAGE);
          } // end switch-case
      } // end method moveDown

      /**
        * Undo a move-down step.
        * @param id ID which represents a move-down step.
        */
      public void undoDown(int id)
      {
          switch (id)
          {
          case 20:
              restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
              break;

          case 21:
              restoreCrate(); // call the specified method to restore a crate on a diamond
or just a crate after the tiny man moves
              // the element below the tiny man before moving
              if (tempMapData[manY + 1][manX] == 4 || tempMapData[manY +
1][manX] == 9)
                  mapData[manY + 1][manX] = 4;
              else
                  mapData[manY + 1][manX] = 2;
              break;

          default:
              JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of undo moving down.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
          } // end switch-case

          mapData[manY - 1][manX] = 5;    // the element above the tiny man
          repaint();
          manY--;
          step--;
      } // end method undoDown

      /**
        * Try to move left a step.
        */
      public void moveLeft()
      {
          // the element on the left side of the tiny man before moving
          switch (mapData[manY][manX - 1])
          {
          case 1:
              mapData[manY][manX] = 6;
              repaint();
              break;

          case 2:
          case 4:
              restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
              mapData[manY][manX - 1] = 6;
              repaint();
              manX--;
              step++;
              undoingRecord.push(30);
              break;

          case 3:
          case 9:
              // the element on the left side of the element on the left side of the tiny
man before moving
              if (mapData[manY][manX - 2] == 4)
              {
```

```
                restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                mapData[manY][manX - 1] = 6;
                mapData[manY][manX - 2] = 9;
                repaint();
                manX--;
                step++;
                undoingRecord.push(31);
            }
            else if (mapData[manY][manX - 2] == 2)
            {
                restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                mapData[manY][manX - 1] = 6;
                mapData[manY][manX - 2] = 3;
                repaint();
                manX--;
                step++;
                undoingRecord.push(31);
            }
            else
            {
                mapData[manY][manX] = 6;
                repaint();
            } // end nested if...else
            break;

        default:
            JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of trying to move left a step.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
        } // end switch-case
    } // end method moveLeft

    /**
     * Undo a move-left step.
     * @param id ID which represents a move-left step.
     */
    public void undoLeft(int id)
    {
        switch (id)
        {
        case 30:
            restoreDiamondOrGroundInside(); // call the specified method to restore
```

```
a diamond or the ground inside after the tiny man moves
                break;

            case 31:
                restoreCrate(); // call the specified method to restore a crate on a diamond
or just a crate after the tiny man moves
                // the element on the left side of the tiny man before moving
                if (tempMapData[manY][manX - 1] == 4 || tempMapData[manY][manX
- 1] == 9)
                    mapData[manY][manX - 1] = 4;
                else
                    mapData[manY][manX - 1] = 2;
                break;

            default:
                JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of undo moving left.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
            } // end switch-case

            mapData[manY][manX + 1] = 6; // the element on the right side of the tiny
man
            repaint();
            manX++;
            step--;
        } // end method undoLeft

        /**
         * Try to move right a step.
         */
        public void moveRight()
        {
            // the element on the right side of the tiny man before moving
            switch (mapData[manY][manX + 1])
            {
            case 1:
                mapData[manY][manX] = 7;
                repaint();
                break;

            case 2:
            case 4:
                restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
```

```
                mapData[manY][manX + 1] = 7;
                repaint();
                manX++;
                step++;
                undoingRecord.push(40);
                break;

            case 3:
            case 9:
                // the element on the right side of the element on the right side of the tiny
man before moving
                if (mapData[manY][manX + 2] == 4)
                {
                    restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                    mapData[manY][manX + 1] = 7;
                    mapData[manY][manX + 2] = 9;
                    repaint();
                    manX++;
                    step++;
                    undoingRecord.push(41);
                }
                else if (mapData[manY][manX + 2] == 2)
                {
                    restoreDiamondOrGroundInside(); // call the specified method to
restore a diamond or the ground inside after the tiny man moves
                    mapData[manY][manX + 1] = 7;
                    mapData[manY][manX + 2] = 3;
                    repaint();
                    manX++;
                    step++;
                    undoingRecord.push(41);
                }
                else
                {
                    mapData[manY][manX] = 7;
                    repaint();
                } // end nested if...else
                break;

            default:
                JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of trying to move right a step.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
```

```
		} // end switch-case
	} // end method moveRight

	/**
	 * Undo a move-right step.
	 * @param id ID which represents a move-right step.
	 */
	public void undoRight(int id)
	{
		switch (id)
		{
		case 40:
			restoreDiamondOrGroundInside(); // call the specified method to restore
a diamond or the ground inside after the tiny man moves
			break;

		case 41:
			restoreCrate(); // call the specified method to restore a crate on a diamond
or just a crate after the tiny man moves
			// the element on the right side of the tiny man before moving
			if (tempMapData[manY][manX + 1] == 4 || tempMapData[manY][manX
+ 1] == 9)
				mapData[manY][manX + 1] = 4;
			else
				mapData[manY][manX + 1] = 2;
			break;

		default:
			JOptionPane.showMessageDialog(this, "Error! Something wrong during
the process of undo moving right.\n(Unknown error)", "Sokoban - Zhao Jichen",
JOptionPane.ERROR_MESSAGE);
		} // end switch-case

		mapData[manY][manX - 1] = 7; // the element on the left side of the tiny man
		repaint();
		manX--;
		step--;
	} // end method undoRight

	/**
	 * Check if the current level is passed.
	 * @return The boolean value true if the current level is passed, or false if it is not
passed.
	 */
```

```
    public boolean isPassed()
    {
        boolean isPassed = false;

        for (int i = 0; i < 20; i++)
            for (int j = 0; j < 20; j++)
            {
                if (tempMapData[i][j] == 4 || tempMapData[i][j] == 9)
                    if (mapData[i][j] == 9)
                        isPassed = true;
                    else
                    {
                        isPassed = false;
                        return isPassed;
                    } // end if...else
            } // end for

        return isPassed;
    } // end method isPassed
```

5.4. Map Loader

The following code is a part of the java class file “MapLoader.java”.

```
    /**
     * 1-parameter constructor MapLoader that generates the basis for painting a 20 *
20 map.
     * @param level A specified level for getting a map file.
     */
    public MapLoader(int level)
    {
        String mapData = ""; // store map data read from a map file
        String eachLine = ""; // store data in each line of a map file

        try
        {
            File mapFile = new File(getClass().getResource("").getPath() + "/maps/"
+ level + ".map"); // load the map file of the specified level
            mapFileReader = new BufferedReader(new FileReader(mapFile));
        }
        catch (IOException e)
        {
            JOptionPane.showMessageDialog(null, "Error! Something wrong during
the process of loading a map file.\n(" + e.toString() + ")", "Sokoban - Zhao Jichen", JOptionPane.ERROR_MESSAGE);
        } // end try...catch

        try
        {
            while ((eachLine = mapFileReader.readLine()) != null)
                mapData += eachLine;
        }
        catch (IOException e)
        {
            JOptionPane.showMessageDialog(null, "Error! Something wrong during
the process of reading a map file.\n(" + e.toString() + ")", "Sokoban - Zhao Jichen", JOptionPane.ERROR_MESSAGE);
        } // end try...catch

        byte[] mapDataToByte = mapData.getBytes(); // get map data as type byte
        int length = mapData.length(); // the length of map data
        int[] mapDataToInt = new int[length]; // store map data as type int
        int index = 0; // the index of map data of type int in the array mapDataToInt

        for (int counter = 0; counter < length; counter++)
            mapDataToInt[counter] = mapDataToByte[counter] - 48;

        for (int i = 0; i < 20; i++ )
            for ( int j = 0; j < 20; j++ )
            {
                map[i][j] = mapDataToInt[index++]; // a line of a map file is a row of
a map

                // 5 represents the starting position of the tiny man
                 if (map[i][j] == 5)
                 {
                     manX = j;
                     manY = i;
                 } // end if
            } // end for
    } // end constructor MapLoader
```

5.5. BGM Manager

The following code is a part of the java class file “BgmManager.java”.

```
        try
        {
            midiData = MidiSystem.getSequence(new
File(getClass().getResource("").getPath() + "/bgm/" + midiFile[index])); // get musical
information of BGM
              midiPlayer = MidiSystem.getSequencer(); // obtain the default
sequencer for playing back the MIDI sequence for BGM
              midiPlayer.open();
              midiPlayer.setSequence(midiData);
            midiPlayer.start();
            midiPlayer.setLoopCount(Sequencer.LOOP_CONTINUOUSLY); // loop
until the player selects another BGM
         }
         catch (Exception e)
        {
            JOptionPane.showMessageDialog(null, "Error! Something wrong during
the process of loading and playing BGM.\n(" + e.toString() + ")", "Sokoban - Zhao
Jichen", JOptionPane.ERROR_MESSAGE);
         } // end try...catch
```