Computer Life
ISSN: 1819-4818 | Vol. 11, No. 1, 2023

Jigsaw puzzle game development based on J2SE

Yunzheng Ding *

School of Information Engineering, Jingdezhen Ceramic university, Jingdezhen 333403, P.R.China

* **Corresponding author**: Yunzheng Ding (Email: jciddyz@163.com)

Abstract: This puzzle game is based on the J2SE platform development, it is an Applet, its game rules and Noah's boat puzzle game is the same. The game cuts a large map into nine smaller ones, and then randomly picks eight of them to appear in nine random positions. Move the 8 small pictures that have been cut through the direction keys of the mouse or keyboard to restore them to the original order, the player will win and the game will end. At the end of the game, the game score will be calculated automatically.

Keywords: J2SE; Java language; Applet; HTML.

1. The overall design of the game

The following is a brief description of what the game is trying to do, and where to do it. The game's design is roughly as follows:

Description of the rules of the game. Since the game is embedded in a web page, the rules of the game can be explained on the web page instead of using Java code in the game.

Game interface. As aesthetically pleasing as possible, this part should be refined in the interface design section.

Menus and buttons to start or end the game. You don't need this either, just have the player start playing when the page is open and turn it off when the page is closed. Instead, there should be a button that will restart the game when the user presses it. This should be done in the section dealing with keyboard and mouse events.

The game can play, is to press the mouse and keyboard at home, the game can do the corresponding action, and judge whether the game has ended. This should be done with a mouse and keyboard event handling section.

Show the player's score at the end of the game. This feature is relatively independent, and adding it when the game is ready to play after event processing does not have any effect on the previous code.

The whole plan goes something like this. Here is just a list of the game to achieve the function, and make clear where each function should be completed, the plan, mainly in order to avoid the blindness at the beginning of the design (which is not conducive to the implementation of later functions). It basically forces programmers to design games in the first place, considering whether the design is suitable for the implementation of other features. For complex projects, there should also be a general plan for the implementation of each function.

2. Interface implementation

2.1. HTML code [1-2]

The HTML code for this part is as follows:

```
<Html>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>
jigsaw puzzle
</title>
</HEAD>
<body>
jigsaw puzzle
<hr>
```

It's a jigsaw puzzle. The player should put together a larger picture from a smaller picture. Players can use the mouse and keyboard to move the small map, the number of moves nd spelling The amount of time spent on a large chart is used as a basis for scoring the game.

Score =1000- Time (s) - Number of moves *10

Press F1 to restart the game, press Y key to preview the entire picture

```
<hr>
<applet code="Puzzle.class" width="480" height="360">
<! NumOfImage The tag of param indicates how many images there are
```

The remaining param is labeled with the name of each graph

```
-->
</applet>
</body>
</html>
```

2.2. Applet Main Screen

The existing image file can be displayed in the Applet in a two-step process[3].

Load image file

Use the Applet's getImage method to load the image file and generate a java.awt.Image object. The format is:

Public Image getImage(URL,String filename)

Public Image getImage(url url2)

url1 is the path where the image file is located,filename is the name of the image file,url2 already contains the name of the image file. The image formats supported in Java are gif,jpg,bmp and so on. Such as:

Imageimg = getImage (getDocumentBase (), "/ images/color. GIF");

Image img = getImage (" http://localhost/ java/images /color.jpg ");

(2) Display image objects

Use the drawImage method in the java.awt.Graphics class to display the Image object, which can be jpg,gif, and other formats. The format is as follows:

drawImage(Image img,int x,int y,ImageObserver observer)

drawImage(Image img,int x,int y,int width,int height,ImageObserver obser)

Where,img is the image object,x and y are the display coordinates,observer is the drawing monitor, that is, the object on which the image is drawn, with the Applet as the monitor, this is used as the parameter. The width and height in the second method specify the width and height of the image display, and if the image size does not match this, the image will be scaled according to this size.

2.3. Interface design of puzzle game

(1) Prepare the picture and load it.

Start by finding a nice picture, 360 by 360 pixels, to use as a jigsaw puzzle frame. Use arrays to represent large and small graphs. To load a small image, first create an instance of each small image object with the createImage(int width,int height) method, then get it with the getGraphics method, and finally draw a fixed area of the large image onto each small image using the drawImage method.

Walk through nine small graph objects to load each of them.

myImage[i] = createImage(IMAGE_WIDTH, IMAGE_HEIGHT);

Create an instance to allocate memory space for each image object:

Graphics g = myImage[i].getGraphics();

Get the Graphics object, figure out which area of the larger image the small image should fit into, and draw it onto the small image.

g.drawImage(myImageAll, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, nRow*IMAGE_WIDTH,nCol*IMAGE_HEIGHT,(nRow+1)*IMAGE_WIDTH, (nCol + 1) * IMAGE_HEIGHT, this);

(2) Draw the interface

Leave an area on the left to show some information about the game and draw the puzzle picture on the right. Define a two-dimensional array to identify the arrangement of each small picture, traverse the array, and draw the puzzle area according to the arrangement order of the small picture expressed in the array. First of all, set the current color to white and fill the prompt area on the left. The coordinate of the string is (10,20). The font of the prompt string is set to Song typeface, the size is 15, and the color is blue. The prompt message is "the number of steps is, the existing picture is 4, please press 1-4 keys to change the picture".

3. Event Processing

In this section I will add the handling of mouse and keyboard events so that the game can be played, the game can be judged to be over, and the player's score can be calculated. I still use the Java 2 messaging mechanism, which is closer to the concept of a generic listener.

3.1. Mouse Event Processing

The game should allow the player to move the puzzle to the surrounding space when clicking on the puzzle with the mouse. When the player clicks on the lower-left puzzle, it moves to the surrounding space, as shown in Figure 1:

Figure 1 After clicking on the mouse

Add the mouse Listener, implement its interface.

First import the package containing the MouseListener interface, which JBuilder has already done.

Import java.awt.event.*;

Declare that the MouseListener interface is implemented

Public class Puzzle extends Applet implements MouseListener;

Add a MouseListener to the Applet:

Public void init()

{... addMouseListener }

Add functionality to the mouseClicked() method:

First determine which puzzle the mouse clicks

int nX = e.getX() - DELTAX;

int nY = e.getY();

int nCol = nY / IMAGE_HEIGHT;

int nRow = nX / IMAGE_WIDTH;

Then determine which direction the puzzle can move in.

First determine where the mouse click on the small graph, and then use the directionCanMove(int nCOl,int nRow) method to determine which direction it can move, if it can move, then use the move method to move the small graph.

Determine which direction the map can move:

Constants that define an Applet are as follows:

final int DIRECTION_UP = 1;

// Indicates that the movement direction is up

final int DIRECTION_DOWN = 2;

// Indicates that the movement direction is down

final int DIRECTION_LEFT = 3;

// Indicates that the movement direction is left

final int DIRECTION_RIGHT = 4;

// Indicates that the movement direction is to the right

final int DIRECTION_NONE = -1;

// means cannot move

The code for this method is as follows:

public int directionCanMove(int nCol, int nRow) {

if ((nCol - 1) >= 0)

if (myImageNo[nRow][nCol - 1] == NO_IMAGE)

return DIRECTION_UP;

// Check whether there is a puzzle above, if not, return this direction value.

if ((nCol + 1) <= 2)

if (myImageNo[nRow][nCol + 1] == NO_IMAGE)

return DIRECTION_DOWN;

Overall description of the algorithm :4 directions in turn judge whether there is a puzzle, if not, return this direction value. If the puzzle is present in all four directions, an integer value is returned indicating that it cannot be moved.

If it can be moved, move the puzzle:

public void move(int nCol, int nRow, int nDirection) {

```
//nCol and nRow are the positions of the puzzle to be moved
switch (nDirection) {
case DIRECTION_UP:
myImageNo[nRow][nCol-1]= myImageNo[nRow][nCol];
myImageNo[nRow][nCol] = NO_IMAGE;
break;
// Determine whether the key is "up". If so, move m_nImageNo up and leave the original position empty
case DIRECTION_DOWN:
myImageNo[nRow][nCol+1]=myImageNo[nRow][nCol];
myImageNo[nRow][nCol] = NO_IMAGE;
break;
// Determine whether the key is down. If so, move m_nImageNo down and leave the original position empty
case DIRECTION_LEFT:
myImageNo[nRo -1][nCol] = myImageNo[nRow][nCol];
myImageNo[nRow][nCol] = NO_IMAGE;
break;
// Check whether the key is left. If so, move m_nImageNo to the left and leave the original position empty
case DIRECTION_RIGHT:
myImageNo[nRow+1][nCol]=myImageNo[nRow][nCol];
myImageNo[nRow][nCol] = NO_IMAGE;
break;
// Determine whether the key is "right". If so, move m_nImageNo to the right and leave the original position empty
}
}
```

A brief description of the algorithm: determine which direction is in turn, and then carry out different operations on m_nImageNo according to different directions.

With the above two methods in place, you can add the following code to the mouseClicked() method:

```
int nDirection = directionCanMove(nCol, nRow);
if (nDirection ! = DIRECTION_NONE) {
move(nCol, nRow, nDirection);
nStep++}
```

Code description: First determine which direction can be moved, if it can be moved (i.e. the direction that can be moved is not DIRECTION_NONE), move the puzzle and add 1 to the number of steps taken by the player.

Repaint()

This completes the processing of the mouse, and then you can use the mouse to play the game. Now let's go ahead and add keyboard controls.

3.2. Keyboard Event Processing

Using a keyboard to play games and playing with a mouse is a little different in the control method, with the mouse to play, the manipulation is just clicked on the puzzle. But how do you play with a keyboard? I do this to specify, use the keyboard arrow keys to move the puzzle. Figure 2 shows any state of the game. There is only one puzzle that can move down, the one in the middle of the far right.

When the player presses the Down arrow button, the game changes to the state shown in Figure 3.

Figure 2 Before the direction key

Figure3 After pressing the direction key

Add the keyboard Listener and implement its interface as follows:

Declare to implement the KeyListener interface:

```
public class Puzzle extends Appletimplements MouseListener, KeyListener
{
...
}
```

Add functional code to the keyClicked methods.

First judge which arrow key is pressed:

```
int nDirection = DIRECTION_NONE;
switch (e.getKeyCode()) {
case KeyEvent.VK_DOWN:
nDirection = DIRECTION_DOWN;
break;
// Check whether the pressed key is "down". If so, save the direction to nDirection, otherwise return
case KeyEvent.VK_UP:
nDirection = DIRECTION_UP;
break;
// Determine whether the pressed key is the "up" key. If so, save the direction to nDirection, otherwise return
case KeyEvent.VK_LEFT:
System.out.println("left111..." );
nDirection = DIRECTION_LEFT;
break;
// Check whether the pressed key is "left". If so, save the direction to nDirection, otherwise return
case KeyEvent.VK_RIGHT:
System.out.println("left..." );
nDirection = DIRECTION_RIGHT;
```

```
break;
// Check whether the pressed key is the "right" key. If so, save the direction to nDirection, otherwise return
```

Algorithm description: If the arrow key is pressed, the direction will be stored in nDirection, otherwise returned.

Move the puzzle in a certain direction.

This algorithm is rather tedious. You can define the method move(int nDirection) to do this. Within this method we first determine which puzzle can be moved in the nDirection and then move the puzzle.

```
public boolean move(int nDirection) {
// Above determine which puzzle can move in the direction nDirection
// The position of the puzzle that can be moved is the nNoImageCol column and the nNoImageRow row
switch (nDirection) {
case DIRECTION_UP:
if (nNoImageCol == 3)
return false;
// Determine whether the key is "up". If so, move m_nImageNo up and leave the original position empty
case DIRECTION_DOWN:
if (nNoImageCol == 0)
return false;
myImageNo[nNoImageRow][nNoImageCol]
= myImageNo[nNoImageRow][nNoImageCol - 1];
myImageNo[nNoImageRow][nNoImageCol-1]=NO_IMAGE;
break;
// Check whether the key is left. If so, move m_nImageNo to the left and leave the original position empty
case DIRECTION_RIGHT:
if (nNoImageRow == 0)
return false;
myImageNo[nNoImageRow][nNoImageCol]
= myImageNo[nNoImageRow - 1][nNoImageCol];
myImageNo[nNoImageRow-1][nNoImageCol]=NO_IMAGE;
break; }
// Determine whether the key is "right". If so, move m_nImageNo to the right and leave the original position empty
```

Call the move(int nDirection) method in mouseClick:

```
move(nDirection);
```

Add 1 to the number of steps the player moves:

```
nstep++;
```

Redraw the interface:

```
repaint();
```

After adding the above code, the game is ready to play. But the game has not yet decided when to win, here is to implement this function.

4. Current game status

The following method is used to determine whether the state of the record by the array m_nImageNo has been won.

4.1. Game state judgment method

Just determine if all eight pieces of the puzzle are in the right place. If yes, the game ends and gives the game score, and let the player press any key can restart the game.

Start by defining a member of the Applet, bWantStartNewGame, to mark whether the game is over and a new game needs to be started.

```
boolean bWantStartNewGame = false;
```

Here's how:

```
public void checkStatus() {
boolean bWin = true;
// Define members. The default value is true
int nCorrectNum = 0;
for (int j = 0; j < 3; j++) {
for (int i = 0; i < 3; i++) {
if (myImageNo[i][j] ! = nCorrectNum
&& myImageNo[i][j] ! = NO_IMAGE)
bWin = false;
nCorrectNum++;
}
}
// Compare the pieces to see if they are all in the right place. If one of them is not in the right place, the game cannot be finished.
if (bWin)
bWantStartNewGame = true; }
```

After calling this method, the current state of the game is stored in bWantStartNewGame. By looking at the value of bWantStartNewGame, you can know the current state of the game: whether it is not finished or the player has won the game.

4.2. Invoking the Method to determine the game State

The method for determining the current state of the game is written and should be called after the player has moved a particular puzzle. Since the redraw method is called every time the puzzle is moved, the call to this method should be placed in the paint () method.

Add the following code to paint() so that the game can determine the current state and print a message like "You won" when the game can end:

```
public void paint(Graphics g) {
... // The code omitted here is the code for drawing
... // The code omitted here is the code to draw the game interface
checkStatus();
// Check the current status
if (bWantStartNewGame) {
// If the game is over, the player puts the puzzle in the right order
nScore = 1000 - nStep * 10 - nTime;
g.setColor(Color.blue);
G. rawString(" Please press any key to restart ", 5, 140);
g.setColor(Color.red);
g.setFont(new Font(" Song Typeface ", font-.plain, 40));
G.rawstring (" You win "+ nScore +" score ", 70 + DELTAX, 160);
G. rawString(" Congratulations! , 110 + DELTAX, 210);
// The message that the player has won is displayed on the interface
}}
```

By adding the above code, the game can determine if the puzzle has been put together. After finishing, the corresponding information will be displayed, as shown in Figure 4:

Figure 4 The final victory interface

5. Initialization of the game

Use the random() method in the Math() package to generate a random number, which is then used to initialize the state of the game. This way, every time the player starts playing the game, the state of the game will be different, and the player will enjoy the game.

Random(): Returns a value of type double that is positive, greater than or equal to 0 and less than 1.

6. Flicker elimination[4]

Now that you have a blinking animation problem, use the updata() method to clear the applet window first, then call the paint() method to redraw the drawing. Let's override this method so that instead of clearing the applet window, the paint() method is called to redraw the drawing:

```
public void update(Graphics g) {
paint(g);
}
```

Instead of the Applet's default update() method, repaint() calls the defined update() method. The update() method prevents the screen from being cleared with a background color during repaint, which greatly reduces the flicker effect.

Now that the game is playable, there are a few more features to add to the game to make it more complete.

7. Preview of the game

It's useful to know what the whole picture looks like when you're doing a puzzle. The Y key is used to control the preview function, because Y is the first letter of "pre". You can preview the image the first time you press Y, and cancel the preview the next time you press Y.

(1) Add a member of the Applet that holds whether the current state is displaying the entire graph:

```
Boolean bOnShowAll=false;
// Preview switch
```

(2) In the KeyPressed() method, which handles keyboard time, add the function code for the first time the Y key is pressed. When the user presses Y, the preview flag is set to true and the whole picture is redrawn:

You should preview the entire image when you press the Y key, but you should also let the user hold down any other key (except the Y key) without letting the image change at all, that is, you should mask keyboard and mouse messages at this point.

8. Picture replacement

Above is the use of a picture, this picture is divided into 9 puzzle, let the player to solve, if the player doubts the beauty of the picture, what should be done? Of course is to change the image, the next use HTML tags to achieve the ability to allow the player to change the image.

(1) param tags for HTML

A Param tag is a tag embedded between applet tags.

Param takes two arguments: a name argument followed by the name of param; One is value, followed by the value of this tag.

(2) Get the value of the param tag in the applet

Use the getParameter() method in the Applet to get the value of the param tag:

```
getParameter(String name)
```

Returns the value of the param tag named name, which is of type String.

References

[1] Yang Hongbo ,Wang Zhishun, “J2SE evolution history”, Programmers,2005(07),P50-52

[2] Zhu Guojun, Liu Wenye, “An introduction to HTML, the Hypertext Markup Language”, Computer Knowledge and Technology,1999(07),P39

[3] Sun Yurong, Wu Lenan,“Unique web applet -JavaApplet”, Digital Communications,1996(04),P30-32

[4] Song Weiwei, Chen Shuzhen, Sun Xiao 'an, “ Multithreading and dual buffering in the Java language” ,Electronic Computers and External Equipment, 1998(06),P30-31