ISSN: 1813-4890

The Design and Implementation of Tetris

Yunzheng Ding

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

Corresponding author: jciddyz@163.com

Abstract

This design uses Java language design, the entire APPLICATION (Java APPLICATION), the use of the latest Java 2 standard Swing components, making the interface more professional effect. Game design process involves the game frame [1], game interface preparation, menu toolbar event processing, variable definition, algorithm design and preview function. At the same time, it also added other functions for the game, such as: the addition of background music, help dialog box, score display and so on.

Keywords

J2SE, Swing components, Tetris, Java application.

1. Introduction

Tetris is a very classic game. It once caused a sensation and created economic value can be said to be a big event in the history of games. Originally created by Soviet game producer Alex Pajitnov, the game is deceptively simple but infinitely varied. Most users remember a time when they were addicted to Tetris all day long. It has become so popular around the world that many object-oriented programming languages can implement it.

2. Overall design

2.1. Game interface background

The realization of the Chinese block of the game is composed of four small color blocks, the same processing of the game base version is also a color block color value of 0 and non-0 color block number of a panel. In order to correctly determine whether a downward move is out of bounds, the game base is given a border as a limit to the area of movement, by setting the outer value of the array to non-zero (non-empty). The falling block figure also moves only in the space composed of color blocks with a color value of 0; The non-0 number represents that the current floor has been occupied. If the falling box image encounters a non-0 color block, it will stop falling, and mark the coordinate position of the color block at the base plate is not empty, and the color will be set to the color value of the current box. As shown in Fig.1, in the background processing and interface display effect of the backboard picture, the outermost data of the array is 1, which is the border added to the backboard. 0, 1, 3, 4, and 5 respectively represent the colors BLANK, BLUE, PINK, GREEN, and YELLOW. The base of the interface display is composed of a color block, each color block has its own color value to indicate whether it is empty.

ISSN: 1813-4890

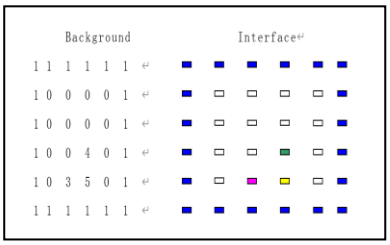

Fig.1 Implement of Game Card and Display of Interface

2.2. Graphical interface display[2] and background connection

Because each standard Tetris is composed of four square blocks, the layout management property of the game board is set to GridLayout type when processing the interface display, the game board JPanel subclasses are divided into n*m parts, Each copy is an instance object of the ChessButton class inherited from the JPanel class, which contains the Color attribute for drawing different colored squares.

2.3. Data structure of standard blocks

Each Tetris is composed of four color blocks of the same color arranged in accordance with certain coordinates, mainly with four point coordinates and color information. The data structure is as follows: (x1,y1,x2,y2,x3,y3,x4,y4, color). For example, the square (1,1,2,1,2,2,3,2,4) represents the coordinate of the first color block as (1,1), the second color block as (2,1), the third color block as (2,2), the fourth color block as (2,4), and the color value of 4 represents GREEN (green). As shown in Fig.2, the background display and interface display of this block in the bottom version.

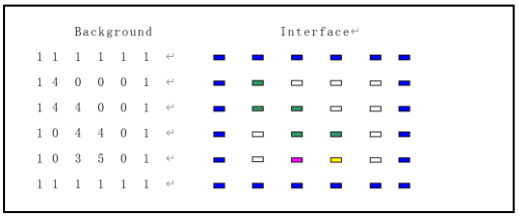

Fig.2 Data Structure of Box

2.4. Realization of automatically falling and receiving key events

The game board is filled with keystroke event monitoring, so when the player pressed the keyboard key is the game's control key, the corresponding method is called to respond to the process. In addition, considering that the box should be able to automatically fall back each certain time when the player has no keys, another thread[3] is opened up to achieve this function, which is to set the player's key value as the function key.

2.5. Implementation of preview diagram

When the game is initialized, a block picture sequence list is defined in advance. After the game starts, the preview area will display the first block picture in the list, which is the next block picture that the game will produce. In this way, whenever the player needs a new block picture, he will get a copy of the first block picture in the list and delete it from the list. And generate a new box image to fill at the end of the list. The preview area takes the first square of the updated list and displays it. This way the preview area always shows the first element in the list, converting it to the current control block as needed, and updating to show the next block image.

ISSN: 1813-4890

2.6. Implementation of row cancellation

When the program detects that none of the bottom lines of the game are empty, it is a full line. First, set the base color block value of this line to BLANK (empty), and search from the previous line, set the base color block information of each line to the base color block information of its previous line, that is, delete all lines above this line and move down one line, and refresh the base in time. When the baseboard value is changed, judge whether the color value of the changed rows is not 0 at all; If more than two full rows exist at the same time, processing starts with the bottom row.

2.7. Implementation of random line addition

In the initial design of the game to consider the two-player attack function, which requires the player's game base to randomly add a few lines of color blocks, to increase the height of the square graphic speed up their death. The process of adding rows is exactly the opposite of eliminating rows. First of all, we must move up a line of all the player's current square graphics, and then randomly add several color blocks in the last line of the panel, and refresh the player's game board in time to achieve the purpose of increasing the interest of the game. Take the first 10 values from the list of points predefined at the beginning of the game (because the width of the game is 10 columns) and draw the base color block based on their values. If more than two lines are added at the same time, the process starts from the bottom line.

2.8. Box falling process demonstration

Fig.3 shows the status of the current block (1, 1, 1, 2, 2, 1, 2, 2).

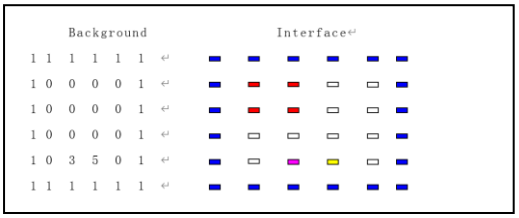

Fig.3 Block-Down1

After a row is lowered, the box is displayed as (2, 1, 2, 2, 3, 1, 3, 2), as shown in Fig.4.

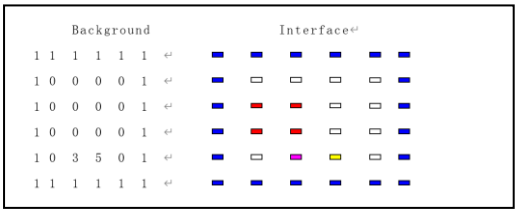

Fig.4 Block-Down2

If the block cannot descend, change the color block value of the baseplate to the color value of the block, and generate the next block graph (1,4,2,4,3,4,4,4,7), as shown in Fig.5.

ISSN: 1813-4890

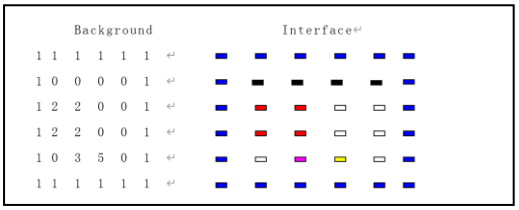

Fig.5 Block-Down3

3. Detailed implementation

3.1. Interface design related classes

The application's main interface, the MainFrame class, inherits from the JFrame class and adds menus and toolbars for the application's controls and other ancillary functions. The layout management setLayout property is set to BorderLayout. An instance object of the SingleDiamonds class for the game board is added. Pre-load the images and music resources used in the program.

The SingleDiamonds class inherits from the PicturePanel class, which is a custom instance of the JPanel class with a background image. The PicturePanel class is designed to make the system look pretty, and its setLayout property is set to null for easy layout management. It adds two players' square panels and their own preview panels, status information display components.

3.2. Public variable

To optimize the code, some variables that are used by many classes are declared static in the Public_variable class.

```
// Tetris picture background color
public static final Color BackColor;
BackColor =new Color(240,192,128);
// Overall background color
public static final Color BackcolorDiamonds;
BackcolorDiamonds=new Color(116,225,236);
// A sequence of randomly generated images
public static final int PICTURE_NUM=300;
// A randomly generated column of points used for shifting
public static final int POINT_NUM=137;
// Define the checkerboard size vertical
public static final int SIZEI=24;
// Define the checkerboard size to be horizontal
public static final int SIZEJ=10;
// Define the game's default block drop speed
public static final int SPEED=800;
```

3.3. The implementation of color blocks

Each shape of standard Tetris is generated by four color blocks in accordance with a certain coordinate position, so a class ChessButton class is defined to describe color block information, which is inherited from the JPanel class.

ISSN: 1813-4890

Its data members are:

```
// 8 colors are predefined, BLANK is the default, and the remaining 7 colors represent different
// shapes.
int LANK=0,BLUE=1,GREEN=2,RED=3,PINK=4;
int YELLOW=5,PURPLE=6,BLACK=7;
// coordinate (subscript) position
public int i=0,j=0;
// Color block color
public int color=0;
```

Its method members are:

```
// When creating such an instance, you must specify its coordinate position and color
public ChessButton(int i,int j,int color){
this.i=i;
this.j=j;
this.color=color;
}
// Overrides the parent paint method. Different implementation parameters can be used to
paint different blocks
public void paint(Graphics g) {
int x1 = 0;
int y1 = 0;
int x2 = getWidth();
int y2 = getHeight();
{
// is the default Blank color
g.setColor(Public_variable.BackcolorDiamonds);
}
```

3.4. The realization of block graphics

Each square is composed of four same color blocks arranged according to certain coordinates, which implies that each square pattern has four groups of coordinate data variables of each color block and color variables representing the pattern. Specific implementation is as follows:

The data members are:

```
// The coordinates of color block one (subscript)
public int x1,y1;
// The coordinates of color block two (subscript)
public int x2,y2;
// Coordinates of color block 3 (subscript)
public int x3,y3;
// Coordinates of color block 4 (subscript)
public int x4,y4;
// The color of the design
public int color;
```

Method members are:

```
// constructor
```

ISSN: 1813-4890

```
// When defining an object of this class, you must specify the coordinates and colors of each color block
public MyImage(int i1,int j1,int i2,int j2,int i3,int j3,int i4,int j4,int color){
x1=i1;
x2=i2;
x3=i3;
x4=i4;
y1=j1;
y2=j2;
y3=j3;
y4=j4;
this.color=color;
}
Since a block's coordinates often change as it falls, moves left and right, and morphs, // there needs to be a common way to reset its properties when called from outside.
public void SetMyImage(int i1,int j1,int i2,int j2,int i3,int j3,int i4,int j4,int color){
x1=i1;
x2=i2;
x3=i3;
x4=i4;
y1=j1;
y2=j2;
y3=j3;
y4=j4;
this.color=color;
}
// In order to test whether the block can fall, move left and right, and transform, get a copy of the block // first, to simulate the action required by the player, rather than using the block directly, // then // need a way to get a copy of the block object. The Clone () block // is defined in the Java.lang library.
public Object clone(){
return new MyImage(x1,y1,x2,y2,x3,y3,x4,y4,color);
}
```

3.5. The realization of the player user

Since this game can implement native dual player user operations, it is a matter of how each player implements this, which data members and method members it has. First of all, each user should have their own game board, their own function keys, but also have their own square pattern, line elimination, score, speed, current state, control of the game thread and other data members, the main method member is the keyboard event processing. The details are as follows:

The data members are:

```
// Represents the current player's start, pause, transform, left, right, and end of the // key
int begin_key,pause_key,up_key,down_key;
int left_key,right_key,bottom_key;
// Chess piece
```

ISSN: 1813-4890

ChessButton cb[][]=new ChessButton[SIZEI+1][SIZEJ+1];
// Checkerboard version
JPanel chessView=new MyJPanel();
// Record whether there is a pause
boolean flag=true;
int speed=Public_variable.SPEED;
// Number of rows deleted by the player
int win_number=0;
// Player score
int mark=0;
// Player game board
int diamonds_panel[][]=new int[SIZEI+2][SIZEJ+2];
Player game event handling thread
Diamonds diamonds_thread;
// The current number of rows
int win_num;
// Whether to request cache
boolean flag_random=false;
// Records information about the image being controlled
tempMyImage=new MyImage(0,0,0,0,0,0,0,0,0,0,0,0);
Method members are:

3.5.1. The realization of clear screen

Clear the screen by setting the color of each piece to the default background color as follows:

```
ublic void ClearChess(){
for(int i=1; i<=SIZEI; i++)
for(int j=1; j<=SIZEJ; j++)
{
cb[i][j].color=0;
}
}
```

3.5.2. Refresh the board[4]

Assign the value of the player's game board to the color value of the chess piece array to complete the refresh of the chess board. The specific method is as follows:

```
public void ReDraw(){
for(int i=1; i<=SIZEI; i++)
for(int j=1; j<=SIZEJ; j++)
cb[i][j].color=diamonds_panel[i][j];
chessView.repaint();
```

3.5.3. Initialization of the baseboard

Setting the perimeter value of the baseboard array to 1 is equivalent to adding a border to the baseboard. The rest of the array members are set to 0 to complete the initialization of the baseboard. The specific method is as follows:

```
void CreatDiamondsPanel(){
for(int i=0; i<=SIZEI+1; i++) diamonds_panel[i][0]=1;
```

ISSN: 1813-4890

```
diamonds_panel[i][SIZEJ+1]=1;
for(int i=0; i<=SIZEJ+1; i++)
diamonds_panel[0][i]=1;
diamonds_panel[SIZEI+1][i]=1;
for(int i=1; i<=SIZEI; i++)
for(int j=1; j<=SIZEJ; j++)
diamonds_panel[i][j]=0;
```

3.5.4. Conflicting judgment

Determine whether the block movement changes, and the bottom plate color block conflict, the specific approach is: the coordinate of each block of the box at the bottom plate value has been marked as zero (can be put value box), as long as there is a place is not empty, return false. The specific methods are as follows:

```
MyImage temp=new MyImage(0,0,0,0,0,0,0,0,0,0);
boolean Clash(MyImage temp){
if(diamonds_panel[temp.x1][temp.y1]! = 0 | |
diamonds_panel[temp.x2][temp.y2]! = 0 | |
diamonds_panel[temp.x3][temp.y3]! = 0 | |
diamonds_panel[temp.x4][temp.y4]! = 0)
return false;
else
return true;
}
```

3.5.5. Can a Test be Moved

Determine whether you can respond to keyboard keystrokes. Perform the following steps:

```
// Check whether it is OK
boolean CanDown(){
// First simulate the coordinates after the box is moved down
temp.SetMyImage(
tempMyImage.x1+1,tempMyImage.y1,
tempMyImage.x2+1,tempMyImage.y2,
tempMyImage.x3+1,tempMyImage.y3,
tempMyImage.x4+1,tempMyImage.y4,
tempMyImage.color);
// Return false if there is a conflict by calling Clash (), true otherwise
return Clash(temp);
}
// Check whether the shift can be left
boolean CanLeft(){
// Simulate the coordinates of the block moved to the left
temp.SetMyImage(
tempMyImage.x1,tempMyImage.y1-1,
tempMyImage.x2,tempMyImage.y2-1,
tempMyImage.x3,tempMyImage.y3-1,
tempMyImage.x4,tempMyImage.y4-1,
```

ISSN: 1813-4890

```
tempMyImage.color);
// Return false if there is a conflict by calling Clash (), true otherwise
return Clash(temp);
}
```

3.5.6. Can deformation test

```
// Determine whether the block can be deformed
boolean CanRound(){
// Field characters, no deformation
if(tempMyImage.color==YELLOW)
return false;
// The third color block coordinate of the box is the center when the box is turned over during the deformation. The rotation changes according to the // ring in order to simulate the change of the box, the rest is calculated first
// The difference between the three color block coordinates and the third color block coordinates
int x1=tempMyImage.x1-tempMyImage.x3;
int x2=tempMyImage.x2-tempMyImage.x3;
int x4=tempMyImage.x4-tempMyImage.x3;
int y1=tempMyImage.y1-tempMyImage.y3;
int y2=tempMyImage.y2-tempMyImage.y3;
int y4=tempMyImage.y4-tempMyImage.y3;
// The rotation of the block can be clockwise or counterclockwise, two kinds of rotation
// The horizontal algorithm for the next block is the same, that is, the horizontal of the third block
// coordinates plus the difference between each color block and the vertical coordinate of the third color block; This department adopts the timing
// The needle rotates, so the vertical coordinate of the remaining three color blocks after rotation is the longitudinal position of the third color block
// Label y3 minus the difference between the horizontal coordinate of this block and the third block, if you want to change to
```

3.5.7. Handling Flickers

In order to eliminate the flicker in the changing process of the block, the strategy of timely local refresh is adopted. First clear the current active graph, the method is to first set the color of the current position of the box as the default background color, that is, set the position of the bottom plate as empty, the specific method is as follows:

```
void ClearTemp(){
cb[tempMyImage.x1][tempMyImage.y1].color=0;
cb[tempMyImage.x2][tempMyImage.y2].color=0;
cb[tempMyImage.x3][tempMyImage.y3].color=0;
cb[tempMyImage.x4][tempMyImage.y4].color=0;
cb[tempMyImage.x1][tempMyImage.y1].repaint();
cb[tempMyImage.x2][tempMyImage.y2].repaint();
cb[tempMyImage.x3][tempMyImage.y3].repaint();
cb[tempMyImage.x4][tempMyImage.y4].repaint();
}
```

ISSN: 1813-4890

3.5.8. Realization of player score and speed

Every time the block falls to the bottom there is a matter of score handling and when and how the speed changes. Encourage players to eliminate as many lines as possible each time, and give certain reward points. The scoring and reward mechanism is: players will get 10 points for each row eliminated, and add 10 points for each additional row eliminated when the number of rows is larger than one. The change in speed is determined by the number of rows eliminated by the player. The initial speed value defined at the beginning of the program is 800ms. The player will speed up when no 5 rows are eliminated. The method code is as follows:

```
void DealWin(){
...
if(win_number%5==0&&(state==0||state==2))
// Speed growth algorithm for traditional mode and two-player self-play mode
speed=speed*9/10;
...
if(win_num>1)
{
// Reward score algorithm
mark+=(win_num-1)*10;
...
// In two-player attack mode, the game speed will not change, but the destroy method will be called
// The number of lines lost by oneself becomes the number of lines added by the opponent
if(state==1)
Destroy(win_num);
}
ReDraw();
}
```

// The realization of the timely display of game scores

To judge whether the player has a full line is to judge the player after the current box in the end, if the player has a full line to do the elimination process, the same player's score will change, then it is necessary to refresh the player's score in time, the specific implementation is as follows:

```
public void DealWithKey(){
...
// to the end
else if(key_num==user2.bottom_key)
{
```

3.5.9. Full Line Test

Every time the block falls to the bottom, it also involves determining whether the row is full or not. In order to optimize the code, I just need to determine whether the row after the block is full or not. The specific method is to pass the X coordinate (the number of rows) of the four small blocks to the HaveWin() method when the box to the bottom, the specific method is as follows:

```
boolean HaveWin(int x1,int x2,int x3,int x4){
// The local variable is used as the flag bit for whether the row is full or not
int i=1;
```

ISSN: 1813-4890

```
// From left to right retrieves whether the bottom of the row where the color block is located is full
for(int j=1; j<=SIZEJ; j++)
// If any color block is empty and the flag bit is 0, the current line is not full. No
// indicates that there are full rows
if(diamonds_panel[x1][j]==0)
i=0;
if(i==1)
return true;
// Reset the value of the flag bit to determine the line where the color block two is located. The method is the same as above.
i=1;
for(int j=1; j<=SIZEJ; j++)
if(diamonds_panel[x2][j]==0)
i=0;
if(i==1)
return true;
// Reset the value of the flag bit to determine the line where the color block is located. The method is the same as above.
i=1;
for(int j=1; j<=SIZEJ; j++)
if(diamonds_panel[x3][j]==0)
i=0;
if(i==1)
return true;
```

3.5.10 Handling Key Events

The realization of automatic box drop, which involves multi-threaded programming, because the game will appear two players, so for each player to define an internal thread class to control their own box automatic drop problem. The thread can not only control the drop of the block, but also in time to determine whether the block has reached the bottom, and do some corresponding processing. For convenience, an additional method, dealwith (), is defined to do this; the thread class simply calls this method.

```
public void DealWith(){
if(CanDown()){
// Each player has a keystroke list to record each keystroke
list_key.addLast(new Integer(down_key));
// User key processing method
DealWithKey();
}
else
{
list_key.addLast(new Integer(bottom_key));
DealWithKey();
}
}
```

ISSN: 1813-4890

```
// Event handler thread
class Diamonds extends Thread{
// When a new class object is created, the game is initialized
Diamonds(){
// Set row elimination to zero
win_number=0;
// Zero the score
mark=0;
// Initialize the baseboard
CreatDiamondsPanel();
// Request an external cache
RandomMyImage();
// Speed is set to the minimum initial value
speed=SPEED;
// Redraw the board
ReDraw();
}
```

3.5.11 Block Request Mechanism

When the player gets to the bottom of a block, it will automatically request the next block, which will request external storage, set the preview block to the current block, delete the top block from the block image sequence, and add a new block image at the end of the sequence.

```
public void RandomMyImage(){
// Indicates that the request cache is true
flag_random=true;
// Request cache, external function
AskRandom();
}
// From the sequence of images generated at the beginning of the game, take out the stored random values and decide
// The next preview box image. picture_key represents the image sequence index,
// The value is incremented each time the cache is requested, that is, each of the images in the sequence
// Square picture. Because the image sequence generation mechanism is randomly generated, so the request
// Blocks have a certain amount of randomness.
void AskRandom(){
// The variable temp is used to obtain the prestored value in the image sequence.
int temp=picture[(picture_key++)%PICTURE_NUM];
// Define an intermediate variable, tt_tempMyImage
// Determine which box image is generated according to the value of temp
MyImage tt_tempMyImage=new MyImage(0,0,0,0,0,0,0,0,0);
if(temp==1)
// Generates a long blue square
1,5,2,5,3,5,4,5 tt_tempMyImage. SetMyImage (BLUE); if(temp==2)
// Generates an orange T-shaped box
```

ISSN: 1813-4890

2,5,4,5,3,5,3,4 tt_tempMyImage. SetMyImage (GREEN);
if(temp==3)
// Generates red type 7 blocks
2,5,4,5,3,5,2,4 tt_tempMyImage. SetMyImage (RED);
if(temp==4)
// Generate pink anti-7 squares
Tt_tempMyImage. SetMyImage (2,4,4,4,3,4,2,5, PINK);
if(temp==5)
// Generates a yellow font box
2,5,3,5,2,4,3,4 tt_tempMyImage. SetMyImage (YELLOW);
What this part of the code does is the change mechanism of the box picture preview area. In fact, every time the player requests a new box picture, it will convert the box picture in the preview area into the current control box picture, that is to say, the preview area always shows the first box picture in the box picture sequence. When the image sequence is updated, update the block image display in the preview area in a timely manner.
// Gets the first square image in the image sequence
MyImage t_temp1=(MyImage)picture1.getFirst();
MyImage t_temp2=(MyImage)picture2.getFirst();
// Refresh the preview area and set the colors of the preview pieces as the background color to clear the previous // block image
for(int i=1; i<=4; i++)
for(int j=1; j<=4; j++)
{
pp_cb1[i][j].color=BLANK;
pp_cb1[i][j].repaint();
pp_cb2[i][j].color=BLANK;
pp_cb2[i][j].repaint();
}
// After the background is drawn, redraw the block image in the preview area
// The color of the corresponding coordinate of the chess piece should be the color of the corresponding coordinate of the first square picture in the picture sequence

References

[1]Yang Hongbo ,Wang Zhishun, “J2SE evolution history”, Programmers,2005(07),P50-52.
[2] Chen Limin ,Twinings nine,” JAVA graphical interface development exploration”, Journal of Southwest University for Nationalities (Natural Science Edition), 2006(02),P405-409.
[3] Hua Weizhong, Zhao Chunyun,” An in-depth look at Java threads”, Computer and Information Technology. 1997(02), P29-30+33.
[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.