Journal of Computer Science and Artificial Intelligence
ISSN: 3078-8242 (Print), ISSN: 3078-2309 (Online) | Vol. 6, No. 1, 2026

The Design of The Main Class of Flying Saucer-Fighting Weapons

Yunzheng Ding *

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

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

Abstract: Globally, the computer game industry has a history of over three decades. After more than three decades of rapid development, the computer game industry has become one of the entertainment industries that are on par with film and televisi on, music, and others. The emergence of the Internet has provided another fascinating carrier and tool for computer games, injecting fresh blood into the further development of the entire industry. In particular, online games, as an emerging industry, have developed rapidly in recent years. Online games, with their features such as real-time interaction, have become the preferred form of leisure entertainment for many young people. Here, we have developed a simple applet game. By placing the game on a server and utilizing the internet, players can start playing from different locations through a browser, enabling human-computer interaction.

Keywords: Java, Multimedia, Java applet, Java game.

1. Requirement Analysis

1.1. Basic Requirements

This program can be directly run using a browser.

Capable of achieving human-computer interaction through mouse clicks.

The main game class utilizes Java's graphics and image technology, multimedia technology, multithreading technology, and animation technology to call elements within the game.

The game should have a beautiful interface and realistic voice acting.

Implement the function of pausing and resuming the game.

1.2. Environmental Requirements

Operating environment: Windows xp, 2003, Linux.

Software environment: JDK 1.5 or above, Eclipse SDK 3.0 or above.

Hardware environment: CPU above 900MHz, memory above 128MB.

2. Design

The online animation game mini program is a game program that integrates multithreading technology, media tracker technology, and double buffering technology. It is a game program that launches missiles to strike flying objects through mouse operation.

Before designing an animated game program, the first step is to find suitable game images, sound files, and materials, and save these files on the disk. This includes background images, flying object images, missile and launcher images, and flying object explosion images, which are stored in the directory of the HTML document where the game mini program is launched as sub directory images. In the game, a sound function has also been added, with sound files of missle.au, Rocker.au, and exosion.au, which are placed in the subdirectories sound.

The implementation of this game requires six classes:

Launch_Missiles: The main class of the game, mainly implementing the Runable interface to control the game, start the game, pause the game, end the game, score, increase game difficulty, and other functions.

Piece class: The basic class defined in this program, which includes the setting of attributes such as position, color, and velocity of an object, as well as methods for drawing, deleting, and collision detection of the object.

Missile launcher class Launcher: Inherits the Piece class, draws the missile launcher, controls its movement, and redraws it.

Missile class: Inheriting the Piece class, drawing missiles, redrawing when moving missiles, etc.

Flyer: Inheriting the Piece class, drawing flying objects, moving flying objects, etc.

Explosion class: Inheriting the Piece class, drawing explosion scenes and clearing explosions, clearing explosions.

3. Implementation of Main Game Classes

3.1. Applet Related

3.1.1. Game Interface Design

The game interface consists of two panels, the first panel displays the game interface, and the second panel places buttons for various functions. The setLayout method is used to set up a container layout manager.

Panel Status, Control; //Define two panels

Add in init():

Status = new Panel(); //Initialize panel

Control = new Panel();

Status. setLayout (new GridLayout (1, 2)); //Set the layout manager Control. setLayout for Status (new GridLayout (1, 3));

setLayout (new BorderLayout()); //Construct a new border layout with no spacing between components.

Add (Status,BorderLayout. NORTH); //Place Status at the top of the container and add (Control, BorderLayout. OUTH); //Control placed at the bottom of the container

The loading of the background image in the game interface is performed during game initialization. Add the following

code in init():

ShowStatus ("Missile Launch Game"); //Request to display the parameter string in the "Status Window".

tracker = new MediaTracker(this);

bgimg = getImage (this. getCodeBase(), "images/" + "bgimg.gif"); //Loading images

tracker. addImage (bgimg,0);

ufostrip = getImage (this. getCodeBase(), "images/" + "ufostrip.gif");

tracker. addImage (ufostrip,0);

missile = getImage (this. getCodeBase(), "images/" + "missile.gif");

tracker. addImage (missile, 0);

missile_explosion = getImage (this. getCodeBase(), "images/" + "explosionstrip.gif");

racker. addImage (missile_explosion, 0);

In init(), it is also necessary to define fonts and define the colors of backgrounds, flying objects, launchers, and fractions:

font = new Font ("Helvetica", Font. BOLD, 24);

font_s = new Font ("Helvetica", Font. BOLD, 14); //Set the font for displaying characters

bgColor = new Color (0, 0, 112); //Set the desired color

gunColor = new Color (0, 112, 0);

mColor = new Color (255, 255, 255);

ufoColor = new Color (255, 0, 0);

scoreColor = new Color (0, 0, 255);

add button

Button btStart; //Define three buttons

Button btStop;

Button btAbout;

Control. add (btStart); //Add buttons to the control panel

Control. add (btStop);

Control. add (btAbout);

Through the above steps, the initialization of the applet has been completed.

3.1.2. Startup of Applet

After the mini program is initialized, it will be started and run. After this mini program is terminated, it can also be started. For example, if the browser enters a different homepage and stops the current mini program, it will be restarted the next time the user returns to the page containing the mini program. In the lifecycle of a mini program, there can be several startups, but initialization only occurs once. To provide startup behavior for the mini program, you can override the start() method:

public void start() {

window_size = getSize(); //Get the size of the window

buffer = null; //Create buffer zone

buffer = createImage (window_size. width, window_size. height);

backdrop = null;

backdrop = createImage (window_size. width, window_size. height);

buf_g = buffer. getGraphics(); //Fill the buffer with background color

buf_g. setColor (bgColor);

buf_g. fillRect (0, 0, window_size. width, window_size. height);

……. //Display of Text in Games

Graphics g = getGraphics(); //Draw the buffer onto the screen

g.drawImage (buffer, 0, 0, this);

mouse_x = window_size. width/2; //Initialize missile launcher

L = new Launcher (this);

L.set_color (gunColor);

M = new Missile (this); //Initialize missile

M.set_color (mColor);

If (explosion==null)//Load sound file

explosion = getAudioClip (getCodeBase(), "sounds/" + "explosion.au");

if (newufo == null)

newufo = getAudioClip (getCodeBase(), "sounds/" + "sonar.au");

if (missile_launch == null)

missile_launch = getAudioClip (getCodeBase(), "sounds/" + "rocket.au");

game_over = true;

newufo.play(); //Sound playback

missile_launch.play();

explosion.play();

3.1.3. Stop() Method

This method is executed when the user leaves the page where the Applet is located, so it can also be executed multiple times. It allows you to stop some system resource consuming work when users are not paying attention to the Applet, so as not to affect the system's running speed, and there is no need to manually call this method. Add in the stop() method:

If (game != null) game = null;

3.2. Game Thread Processing

Because the game is an Applet program, the main class inherits from the Applet class, and Java does not allow multiple inheritance. To implement thread calling, it can only be done through the Runnable interface. To implement the Runnable interface, a simple method for a class to implement only one run() is as follows:

public void run()

ti = System. currentTimeMillis();

if((UV.size() < NU) && (Math. random() >(UV.size() == 0 ? 0.90:0.98)))

If there is extra space for flying objects, an additional flying object can be added

newufo. play(); //Play alarm sound

U = new Flyer(this);

U.set_color(ufoColor);

if(score > 10 && Math. random() >0.7)

U.vy -= 1;

//Increase the descent speed of the flying object under corresponding conditions

UV.addElement(U); //Add a flying object

}

for(int j = EV.size()-1; j >= 0; --j)

Draw an explosion scene on the background screen and clear it after completion

E =(Explosion) EV.elementAt(j);

if(E.active){E.draw();}

else {

E.erase();

EV.removeElementAt(j);

}

}

L.move(); //Mobile Launch Stand

if (M.active() == true) M.move(); //Mobile missile

for (int i=0; i < UV.size(); ++i)

{//Move each flying object

U = (Flyer) UV.elementAt(i);

```
U.move();
}
for(int i = (UV.size()-1); i >=0; --i)
Monitor the collision between flying objects and missiles
U =(Flyer) UV.elementAt(i);
if(U.active() && M.active() && U.collision(M))
{
++score; //Increase score
explosion.stop();
display_score();
explosion.play();
if ((NU < 5) && (score % 10) == 1) ++NU;
//After shooting down 10 flying objects, increase the maximum number of flying objects until the number reaches 5
M.active(false);
M.erase();
U.active(false);
U.erase();
E = new Explosion(this, U.px, U.py);
EV.addElement(E);
}
if (U.active())
U.draw(); //If a flying object exists, draw it
else
UV.removeElementAt(i); //Remove it from the flying object vector
if (L.has_moved() || ((M.py-M.h) < (L.py+L.h)) || (! M.active()))
L.draw(); //If the missile launcher moves, redraw the launcher
if (M.active() == true) M.draw();
ti = System.currentTimeMillis() - ti;
ti = 20 - ti;
ti = ti > 0? 10 + ti: 10;
Thread.yield();
Try {Thread. sleep (ti);}//Handle exceptions in the thread sleep function
catch (InterruptedException e) { };
if ((count = ++count % 500) == 0) {
//Redraw every 100 cycles
repaint();
Define constants for text display
public static final int CENTER = 1; //Mode: Center
public static final int LEFT = 2; //Ju Zuo
public static final int RIGHT = 3; //Ju Right
public static final int FREE = 0; //Stay at the given (x, y) position
public static final int NORMAL = 0; //Type: Normal
public static final int SHADOW = 1; //With shadows
Define text display variables
private int say_pos_x = 0;
private int say_pos_y = 0;
private int say_mode = -1;
private int say_style = -1;
private int say_margin = 10;
private Font say_font = null;
Text display method
public void say(String s, int x, int y) {
set_say_pos(x, y) ;
say(s);
}
public void say(String s) {
FontMetrics fm = getFontMetrics(say_font); //Get font information
Switch (say_made) {//Calculate x-coordinate
case CENTER:
say_pos_x = (window_size.width - fm.stringWidth(s))/2;
break;
case RIGHT:
say_pos_x = window_size.width - fm.stringWidth(s) - say_margin;
break;
case LEFT:
default:
say_pos_x = say_margin;
break;
}
Graphics bg = buffer.getGraphics();
bg.setFont(say_font);
if (say_style == SHADOW) {
bg.setColor(new Color(150, 150, 150));
bg.drawString(s, say_pos_x+2, say_pos_y+1);
}
bg.setColor(Color.white); //Write a string in the buffer
bg.drawString(s, say_pos_x, say_pos_y);
say_pos_y += (int) (1.2 * fm.getHeight()); //Increase the corresponding y-coordinate value
bg.dispose(); //Release resources
}
Public void set_stay_made (int m) {//Set display mode
say_mode = m;
}
Public void set_stay_style (int s) {//Set display type
say_style = s;
}
Public void set_stay_font (Font f) {//Set the font used for display
say_font = f;
}
Public void set_Say_margin (int margin) {//Set the displayed blank edge
say_margin = margin;
}
Public void set_stay_pos (int x, int y) {//Set the coordinates of the display position
say_pos_x = x;
say_pos_y = y;
}
```

3.3. Display of Text and Scores in the Game

3.3.1. Display of Text

```
Define constants for text display
public static final int CENTER = 1; //Mode: Center
public static final int LEFT = 2; //Ju Zuo
public static final int RIGHT = 3; //Ju Right
public static final int FREE = 0; //Stay at the given (x, y) position
public static final int NORMAL = 0; //Type: Normal
public static final int SHADOW = 1; //With shadows
Define text display variables
private int say_pos_x = 0;
private int say_pos_y = 0;
private int say_mode = -1;
private int say_style = -1;
private int say_margin = 10;
private Font say_font = null;
Text display method
```

```
public void say(String s, int x, int y) {
set_say_pos(x, y);
say(s);
}
public void say(String s) {
FontMetrics fm = getFontMetrics(say_font); //Get font information
Switch (say_made) {//Calculate x-coordinate
case CENTER:
say_pos_x = (window_size.width - fm.stringWidth(s))/2;
break;
case RIGHT:
say_pos_x = window_size.width - fm.stringWidth(s) - say_margin;
break;
case LEFT:
default:
say_pos_x = say_margin;
break;
}
Graphics bg = buffer.getGraphics();
bg.setFont(say_font);
if (say_style == SHADOW) {
bg.setColor(new Color(150, 150, 150));
bg.drawString(s, say_pos_x+2, say_pos_y+1);
}
bg.setColor(Color.white); //Write a string in the buffer
bg.drawString(s, say_pos_x, say_pos_y);
say_pos_y += (int) (1.2 * fm.getHeight()); //Increase the corresponding y-coordinate value
bg.dispose(); //Release resources
}
Public void set_stay_made (int m) {//Set display mode
say_mode = m;
}
Public void set_stay_style (int s) {//Set display type
say_style = s;
}
Public void set_stay_font (Font f) {//Set the font used for display
say_font = f;
}
Public void set_Say_margin (int margin) {//Set the displayed blank edge
say_margin = margin;
}
Public void set_stay_pos (int x, int y) {//Set the coordinates of the display position
say_pos_x = x;
say_pos_y = y;
}
```

3.3.2. Display of Scores

```
Public void display _ score() {//score display
Graphics bkd_g = backdrop.getGraphics();
bkd_g.clipRect(window_size.width/2, 0, window_size.width/2, 40);
//Create display area
bkd_g.drawImage(bgimg, 0, 0, window_size.width, window_size.height, this);
bkd_g.setColor(Color.red); //Set font color
bkd_g.setFont(font); //Set font
String aux = score > 9 ? 0
bkd_g.drawString(aux+score, window_size.width - 60,30); //Display score
bkd_g.dispose();
Graphics bg = buffer.getGraphics();
//Using double buffering to display the image in the buffer on the screen
bg.clipRect(0, 0, window_size.width, 40);
bg.drawImage(backdrop,0,0,this);
bg.dispose();
Graphics g = getGraphics();
g.clipRect(0, 0, window_size.width, 40);
g.drawImage(buffer, 0, 0, this);
g.dispose();
}
```

3.4. Control the Start, End, Pause, And Continue of The Game

3.4.1. The Beginning of the Game

The start of the game can be defined as a thread in the method, and then the thread can start running. It should also be noted to reassign some variables, such as game scores. Specific implementation:

```
NU = 1; //The number of flying objects is 1
score = 0; //Score is 0
game = new Thread(this);
game.setPriority (Thread.MIN_PRIORITY); //Set thread priority
game.start(); //Start the game
```

3.4.2. Ending the Game

Ending the game is also very simple. First, check if the game is currently running. If so, end the thread of the current game. Otherwise, no action will be taken.

```
if (game != null)
game = null;
```

3.4.3. Game Pause

This feature can be achieved by simply hanging up threads. Firstly, we determine whether the game thread is empty, that is, whether the game is running. If so, suspend the thread and mark paused as true.

```
if(game!=null)
{
try {game.suspend();paused = true;}
catch(Exception e){e.printStackTrace();}
}
```

3.4.4. Game Continuation

The continuation of the game is to continue the suspended thread.

```
game.resume();
```

The above functions are mainly implemented in mouse response.

3.5. Mouse Event Handling

3.5.1. Mouse Movement Operation

When the mouse moves on the game interface, it is used to control the direction of the missile's flight, and the direction of the missile's movement is at the X value of the mouse's position. At the same time, the missile launcher should also be moved to the X value where the mouse is located. Therefore, when the mouse moves, only the X value of the mouse position needs to be returned.

```
public boolean mouseMove(Event e, int x, int y)
{
mouse_x = x;
return true;
}
```

3.5.2. Operation When Pressing the Mouse Button

When the mouse is pressed, it is used to control the launch of missiles. If the missile is present, there will be no response. If the missile flies off the screen or collides with the flying object and explodes, a new missile will be launched.

```
public boolean mouseDown(Event e, int x, int y)
{
if (M != null && ! M.active())
{
missile_launch.stop();
missile_launch.play();
M.set_pos(L.px, L.py);
M.active(true);
}
return true;
}
```

3.5.3. Button Response

Java's event handling uses listeners. If dealing with an event, simply install a listener for that event. Firstly, the interface for ActionListener should be implemented in the Applet class. This is a listening interface used to handle ActionEvent events. When an ActionEvent event is generated, the ActionPerformed method defined in the interface will be called.

Code for implementing the interface:

```
public class Launch_Missiles extends Applet implements Runnable, ActionListener
```

The program uses evt. getActionCommand() to obtain information about which button triggered this event. After implementing the listening interface, the component being monitored for this event should be installed with this listener. The method to install a listener is to call the addActionListener() method of the component.

```
btStart.setActionCommand ("start");
btStart.addActionListener (this);
btStop.setActionCommand ("stop");
btStop.addActionListener (this);
btAbout.setActionCommand ("about");
btAbout.addActionListener (this);
```

We also used setActionCommand (String object) here, which is used to distinguish the ActionEvents generated by different components.

```
Handle events related to these buttons.
public void actionPerformed (ActionEvent evt)
{
if(evt.getActionCommand().equals ("start"))
When the 'btStart' button is pressed
If (btStart. getLabel()=="Start")
The label for 'btStart' is' start '
BtStop. setLabel ("pause");
//Start the game
if(game_over||game == null )
{
game_over = false;
NU = 1;
score = 0;
M.active(false);
UV.removeAllElements();
EV.removeAllElements();
game = new Thread(this);
game.setPriority(Thread.MIN_PRIORITY);
game.start();
running = true;
if(game != null)
{
game.stop();
game = null;
buf_g = backdrop.getGraphics();
buf_g.drawImage(bgimg, 0, 0, window_size.width,
window_size.height, this);
buf_g =getGraphics();
buf_g.drawImage(backdrop, 0, 0, this);
buf_g = buffer.getGraphics();
buf_g.drawImage(bgimg, 0, 0, window_size.width,
window_size.height, this); //End the game screen, only display the background
repaint();
display_game_over(); //Display game end text description
BtStart. setLabel ("Start");
if(evt.getActionCommand().equals("stop"))
If 'btStop' is pressed
If (btStop. getLabel()=="Pause"&&&running)
{//The label for 'btStop' is' pause 'and the game is running
if(game!=null)
{//Game paused
try{game.suspend();paused = true;}
catch(Exception e){e.printStackTrace();}
display_game_stop(); //Display game pause prompt
}
BtStop. setLabel ("continue");
}
Else if (btStop. getLabel()=="Continue"&&&running)
{//Continue the game
game.resume();
//Redraw the game screen
buf_g = backdrop.getGraphics();
buf_g.drawImage(bgimg,0,0,window_size.width,
window_size.height,this) ;
buf_g =getGraphics();
buf_g.drawImage(backdrop,0,0,this);
buf_g = buffer.getGraphics();
buf_g.drawImage(bgimg, 0, 0, window_size.width,
window_size.height,this);
repaint();
display_score(); //Display score
L.draw(); //Redraw the missile launcher
paused = false;
BtStop. setLabel ("pause");
}
}
if(evt.getActionCommand().equals("about"))
The 'btAbout' button is pressed
if(game==null)
{//When the game is not running, a "About" window pops up
btAbout.setEnabled(false);
JOptionPane.showmessageDialog (this, "This is a simple animated mini game. After entering the game screen, \ n"+"Click with the mouse to launch missiles and strike flying objects \ n"+"The more flying objects you hit, the higher your score. \The n "+" flying object lands, and the game ends. \Developer: Li Weiwu, Lu Tingting, About, JOptionPane. PLAIN_MESSAGE;
```

3.6. Eliminate flicker

Eliminating flicker requires the use of double buffering technology. This function is very easy to implement. The code is simple: reload the updata() method and directly call the

paint() method in update().

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

Add code to the paint() method:

```
if (buffer != null)
g.drawImage(buffer, 0, 0, this);
```

Other screens are displayed in their respective methods

3.7. Other Code Explanations

3.7.1. Prompt Text Displayed on The Game Screen at The End of The Game

```
public void display_game_over()
{
running = false;
BtStart. setLabel ("Start");
set_say_font(font); //Font of displayed text
set_say_mode(CENTER); //Mode
set_say_style(SHADOW);
set_say_pos(10, 150); //Location
Say ("game over");//Display the content of the text
set_say_font(font_s);
set_say_style(NORMAL);
Say ("Click the start button - start game");
repaint();
try{Thread.sleep(500);}
catch(InterruptedException e){};
}
```

The prompts during game pause are similar to those at the end of the game, and will not be elaborated here.

3.7.2. Change of Mouse Type

Due to the fact that the movement of the missile is controlled by a mouse, the direction of the missile's movement is determined by the X value at the location of the mouse. Therefore, changing the type of mouse to a crosshair is beneficial for aiming at flying objects and increasing scores. But when the mouse moves out of the game area, it returns to the default type.

Firstly, it is necessary to define these two types of mice:

```
Cursor cross = new Cursor (Cursor. CROSSHAIR_CURSOR);
Cursor defaultcursor = new Cursor (Cursor. DEFAULT_CURSOR);
```

When the mouse moves over a button, restore the default mouse:

```
Control.setCursor (defaultcursor);
```

The game starts with setting the mouse as a crosshair:

```
setCursor(cross);
```

References

[1] Yao Baiyun, Zhang Hao, Du Ruiqing, research on the design and implementation of the Java based game "Flying Bird", Information and Computer (theoretical version) 2024, 36 (20), P118-120.

[2] Li Shaofang, Research on the Design of Java based Picture Lianliankan Game, Journal of Xi'an University of Arts and Sciences (Natural Science Edition) 2022, 25 (04), P59-66.

[3] Lin Hengjian, a multi-threaded pinball game program based on Java, in the electronic world 2020 (22), P18-19.

[4] Lv Jiahuan, Jiang Fujinyi, Java based Bomberman game design, computer knowledge and technology 2020, 16 (25), P97-98.

[5] Wang Guangshuai, Practical Java Game Server Architecture, People's Posts and Telecommunications Press, 2020(09).