

16. Removing the Button Controls

In order to control the game via keyboard keys, the button controls need to be removed. The following code must be deleted from *fly.java*.

These instance variables must be removed,

```
private Button leftButton, rightButton, startButton;
```

along with their declarations and insertions in method *init*.

```
// set up directional buttons
startButton = new Button("Start");
startButton.addActionListener(new myButtonHandler ());
add(startButton);

leftButton = new Button("J - Left");
leftButton.addActionListener(new myButtonHandler ());
add(leftButton);

rightButton = new Button("K - Right");
rightButton.addActionListener(new myButtonHandler ());
add(rightButton);
```

Since the buttons have been removed, the internal class *myButtonHandler* is no longer needed and can be deleted.

```
// ***** internal class to handle button actions *****
public class myButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (leftButton == e.getSource()) {
            p.turnleft();p.turnleft();
        } else if (rightButton == e.getSource()) {
            p.turnright();p.turnright();
        } else if (startButton == e.getSource()) {
            startPressed = true;
        }
    }
}
```

17. Controlling by Keys

In order for keys to be used for controlling the game, the interface *KeyListener* must be implemented by class *fly*.

```
public final class fly extends Applet implements KeyListener, Runnable {
```

Interface *KeyListener* requires that three methods be implemented. These are *keyPressed*, *keyReleased* and *keyTyped*. Each of these must be declared as *public*, are *void* and receive a parameter of class *KeyEvent*. When a key is pressed, a *KeyEvent* object is created and passed to, in turn, *keyPressed* and *keyTyped*. Then, when the key is released, the event is passed to *keyReleased*.

In this program, methods *keyPressed* and *keyReleased* have nothing to do and will be empty. Empty or not, since class *fly* implements interface *KeyListener*, they must be there.

```
    public void keyPressed(KeyEvent e) { };
```

```
    public void keyReleased(KeyEvent e) { };
```

Method *keyTyped* will be used to react to all key presses that are user commands. First, a char variable *c* will receive a value from the *KeyEvent* method *getKeyChar()*:

```
    public void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
```

Then, the value of variable *c* will be tested. If the <S> key has been pressed, the Boolean variable *startPressed* will be set to *true*, which will activate the loop in the *run* method. If the <J> key has been pressed, the two calls are made to the *turnleft* method of the user's *Bug* object. If the <K> key has been pressed, the two calls are made to the *turnright* method of the user's *Bug* object.

```
        if (c == 's' | c == 'S') {
            startPressed = true;
        } else if (c == 'j' | c == 'J') {
            p.turnleft(); p.turnleft();
        } else if (c == 'k' | c == 'K') {
            p.turnright(); p.turnright();
        }
    }; // *** end of keyTyped ***
```

In the *run* method calls to method *addKeyListener* must be made in order to register the necessary functions to detect a key being pressed. Notice that in this example method *addKeyListener* registers these functions to *this*. Also, in each pass of the while loop of method *run*, a call to method *requestFocus* must be made in order to receive any keyboard input events that might have happened.

```
public void run() {
    addKeyListener(this);
    out:while(true) {
        requestFocus();
        if (startPressed) {
            moveBugArray();
            :
            :
        }
    }
}
```

Lastly, since button labels will not be present in the applet, changes to and additional *drawString* output in method *paint* must be added to provide instructions to the user.

```
hiddenG.setColor(Color.yellow);
hiddenG.drawString("S - Start, J - Left, K - Right", l/2-15,15);
hiddenG.setColor(Color.red);
hiddenG.drawString(Integer.toString(maxbugs-deadbugs)
    + "Bugs, " + Integer.toString(maxevilbugs-deadevilbugs)
    + " Evil Bugs, " + Integer.toString(maxbug_zappers)
    + " BugZappers",l/2-40,30);
```

Full Listing of *fly.java*

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public final class fly extends Applet implements KeyListener, Runnable {

    private int maxbugs = (int)(Math.random() * 30 + 10);
    private int maxevilbugs = (int)(Math.random() * 10);
    private int maxbug_zappers = (int)(Math.random() * 10);
    private int w = 500;
    private int l = 400;

    private Thread animate;
    private Image myBuffer;

    private Background b = new Background(w, l);
    private BugZapper rarray[ ];
```

```

private Bug parray[ ];
private evilBug earray[ ];
private Bug p = new Bug(w,l,w/4+(int)((w/2)*Math.random( )),380);

private int deadbugs = 0;
private int deadevilbugs = 0;

private boolean startPressed = false;

public void init() {
    animate = new Thread(this);
    myBuffer = createImage(w,l);

    // setup bug array
    parray = new Bug[maxbugs];
    for (int x=0; x<maxbugs; x++) {
        parray[x] = new Bug( w,l, (int)(w*Math.random( )),
                           (int)(l*Math.random( )));
        parray[x].setBugColor( (int)(100*Math.random( )+50),
                               (int)(200*Math.random( )+50),
                               (int)(200*Math.random( )+50) );
        for(int y=0; y<((int)(16*Math.random( ))); y++)
            parray[x].turnright( );
    }

    // setup evilbug array
    earray = new evilBug[maxevilbugs];
    for (int x=0; x<maxevilbugs; x++) {
        earray[x] = new evilBug( w,l, (int)(w*Math.random( )),
                                (int)(l*Math.random( )), p);
        earray[x].setBugColor(255,255,255);
        for(int y=0; y<((int)(16*Math.random( ))); y++)
            earray[x].turnright( );
    }

    // set up bug zapper array
    rarray = new BugZapper[maxbug_zappers];
    for (int y=0; y<maxbug_zappers; y++)
        rarray[y] = new BugZapper((int)(w*Math.random( )),
                                  (int)(l*Math.random( )),
                                  (int)(50*Math.random( )+20));

    // set up bug
    p.setBugColor(255,0,0);
}

```

```
public synchronized void paint(Graphics g) {
    Graphics hiddenG = myBuffer.getGraphics();
    b.paint(hiddenG);
    paintBugZappers(hiddenG);
    paintBugArray(hiddenG);
    paintEvilBugArray(hiddenG);
    p.paint(hiddenG);
    hiddenG.setColor(Color.yellow);
    hiddenG.drawString("S - Start, J - Left, K - Right", 1/2-15,15);
    hiddenG.setColor(Color.red);
    hiddenG.drawString(Integer.toString(maxbugs-deadbugs) + "
        Bugs, " + Integer.toString(maxevilbugs-deadevilbugs)
        + " Evil Bugs, " + Integer.toString(maxbug_zappers) + "
        BugZappers",1/2-40,30);
    g.drawImage(myBuffer,0,0,this);
}

private void paintBugZappers(Graphics g){
    for (int x=0; x<maxbug_zappers; x++) {
        rarray[x].paint(g);
    }
}

private synchronized void moveBugArray( ){
    int d;
    for (int x=0; x<maxbugs; x++) {
        d = (int)(Math.random()*5);
        if (d < 1) parray[x].turnright( );
        else if (d >3) parray[x].turnleft( );
        parray[x].go( );
    }
}

private void paintBugArray(Graphics g){
    for (int x=0; x<maxbugs; x++)
        if (parray[x].isAlive( )) parray[x].paint(g);
}

private synchronized void moveEvilBugArray(){
    int d;
    for (int x=0; x<maxevilbugs; x++) {
        earray[x].calcDirection( );
        earray[x].go( );
    }
}
```

```

private void paintEvilBugArray(Graphics g){
    for (int x=0; x<maxevilbugs; x++)
        if (earray[x].isAlive( )) earray[x].paint(g);
}

public void start( ){
    animate.start( );
}

public void stop( ){;}

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

public void run( ){
    addKeyListener(this);
    out:while(true) {
        requestFocus( );
        if (startPressed) {
            moveBugArray( );
            moveEvilBugArray( );
            p.go( );
            repaint( );

            // Check for Bug-to-Bug collisions
            for (int a=0; a<maxbugs-1; a++)
                for (int b=a+1; b<maxbugs; b++)
                    if ( parray[a].isAlive( ) &
                        parray[b].isAlive( ) &
                        parray[a].impactBug(parray[b]) ) {
                        parray[a].isDead( );
                        parray[b].isDead( );
                        deadbugs++;
                        deadbugs++;
                    }
                }
            // Check for Bug-to-evilBug collisions
            for (int a=0; a<maxbugs-1; a++)
                for (int b=0; b<maxevilbugs; b++)
                    if ( parray[a].isAlive( ) &
                        earray[b].isAlive( ) &
                        earray[b].impactBug(parray[a]) ) {
                        parray[a].isDead( );
                        earray[b].isDead( );
                        deadbugs++;
                        deadevilbugs++;
                    }
                }
        }
    }
}

```



```

private void pause (int time) {
    try    { Thread.sleep(time); }
    catch (Exception e) {}
}

// ***** functions for interface KeyListener *****
public void keyPressed(KeyEvent e) { };

public void keyReleased(KeyEvent e) { };

public void keyTyped(KeyEvent e) {
    char c = e.getKeyChar( );
    if (c == 's' | c == 'S') {
        startPressed = true;
    } else if (c == 'j' | c == 'J') {
        p.turnleft( );p.turnleft( );
    } else if (c == 'k' | c == 'K') {
        p.turnright( );p.turnright( );
    }
};
}

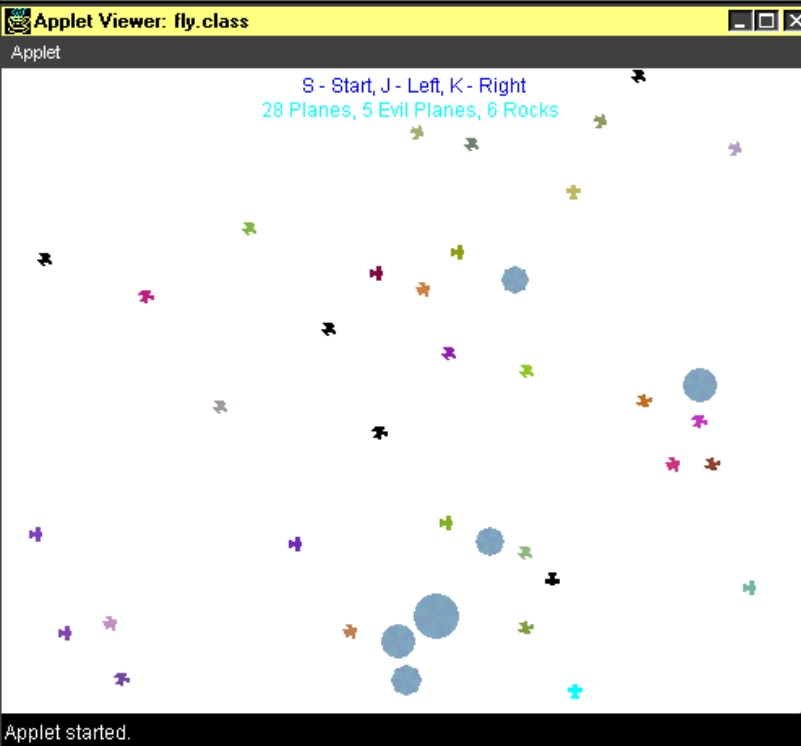
```

Example Compile and Run of Class *fly*

```

C:\jdk1.3\projects\gameexample2>go
C:\jdk1.3\projects\gameexample2>javac fly.java
C:\jdk1.3\projects\gameexample2>appletviewer fly.html

```



Applet started.