

20. Preparation for Controlling by Keys

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

These instance variables must be removed,

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

along with their declarations and insertions in the constructor.

```
// 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.

```
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;
        }
    }
}
```

21. Controlling by Keys

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

```
public final class MainPanel extends JPanel
    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 tutorial, methods *keyPressed* and *keyReleased* have nothing to do and will be empty. Empty or not, since class *fly* implements interface *KeyListener*, they must be in the program:

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

Method *keyTyped* will be used to react to all key presses that are user inputs. 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();
        }
    }
```

In the *run* method, a call 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);
    requestFocus();
    out:while(true) {
        if (!startPressed) {
            requestFocus();
        } else {
            requestFocus();
            :
            :
        }
    }
}
```

Lastly, since buttons will not be present, changes to the *drawString* output in method *paintComponent* must be added to provide instructions to the user.

```
hiddenG.setColor(Color.yellow);
hiddenG.drawString(
    "S - Start, J - Left, K - Right",
    WIDTH/2-70,15);
hiddenG.setColor(Color.red);
hiddenG.drawString(
    Integer.toString(maxbugs-deadbugs)
    + " Bugs, "
    + Integer.toString(maxevilbugs-deadevilbugs)
    + " Evil Bugs, "
    + Integer.toString(maxbug_zappers)
    + " BugZappers",WIDTH/2-100,30);
```

22. A Complete (and Final) Listing of *MainPanel.java*

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public final class MainPanel extends JPanel
    implements KeyListener, Runnable {

    private final int WIDTH;
    private final int HEIGHT;

    private Image myBuffer;
    private Thread animate;

    private final int maxbugs = (int)(Math.random()*30+10);
    private final int maxevilbugs = (int)(Math.random()*10);
    private final int maxbug_zappers = (int)(Math.random()*10);

    private BugZapper rarray[];
    private Bug parray[];
    private EvilBug earray[];
    private Bug p;

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

    private boolean startPressed = false;

    public MainPanel(int wInit, int hInit) {
        WIDTH = wInit;
        HEIGHT = hInit;

        p = new Bug(WIDTH,HEIGHT,
            WIDTH/4+(int)((WIDTH/2)*Math.random()),
            HEIGHT-20);

        animate = new Thread(this);

        // setup bug array
        parray = new Bug[maxbugs];
        for (int x=0; x<maxbugs; x++) {
            parray[x] = new Bug(WIDTH,HEIGHT,
                (int)(WIDTH*Math.random()),
                (int)(HEIGHT*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(WIDTH,HEIGHT,
        (int)(WIDTH*Math.random()),
        (int)(HEIGHT*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)(WIDTH*Math.random()),
        (int)(HEIGHT*Math.random()),
        (int)(50*Math.random()+20));

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

public void paintComponent(Graphics g) {
    super.paintComponent(g);

    myBuffer = createImage(WIDTH,HEIGHT);
    Graphics hiddenG = myBuffer.getGraphics();
    hiddenG.setColor(Color.black);
    hiddenG.fillRect(0, 0, WIDTH,HEIGHT);

    paintBugZappers(hiddenG);
    paintBugArray(hiddenG);
    paintEvilBugArray(hiddenG);
    p.paintComponent(hiddenG);

    hiddenG.setColor(Color.yellow);
    hiddenG.drawString("S - Start, J - Left, K - Right",
        WIDTH/2-70,15);
    hiddenG.setColor(Color.red);
    hiddenG.drawString(
        Integer.toString(maxbugs-deadbugs)
        + " Bugs, "
        + Integer.toString(maxevilbugs-deadevilbugs)
        + " Evil Bugs, "
        + Integer.toString(maxbug_zappers)
        + " BugZappers",WIDTH/2-100,30);

    g.drawImage(myBuffer,0,0,this);
}

```

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

private 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].paintComponent(g);
}

private 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].paintComponent(g);
}

public void CollideBugToBug() {
    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++;
            }
}
}
```

```
public void CollideBugToEvilBug() {
    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++;
            }
}

public void CollideEvilBugToEvilBug() {
    for (int a=0; a<maxevilbugs-1; a++)
        for (int b=a+1; b<maxevilbugs; b++)
            if (earray[a].isAlive() &
                earray[b].isAlive() &
                earray[b].impactBug(earray[a])){
                earray[a].isDead();
                earray[b].isDead();
                deadevilbugs++;
                deadevilbugs++;
            }
}

public void CollideBugToBugZapper() {
    for (int a=0; a<maxbugs; a++)
        for (int b=0; b<maxbug_zappers; b++)
            if (parray[a].isAlive() &
                rarray[b].impactBug(parray[a])){
                parray[a].isDead( );
                deadbugs++;
            }
}

public void CollideEvilBugToBugBugZapper() {
    for (int a=0; a<maxevilbugs; a++)
        for (int b=0; b<maxbug_zappers; b++)
            if (earray[a].isAlive() &
                rarray[b].impactBug(earray[a])){
                earray[a].isDead();
                deadevilbugs++;
            }
}

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

```

public void run(){

    addKeyListener(this);
    requestFocus();
    out:while(true) {
        if (!startPressed) {
            requestFocus();
        } else {
            requestFocus();
            moveBugArray();
            moveEvilBugArray();
            p.go();
            repaint();

            CollideBugToBug();
            CollideBugToEvilBug();
            CollideEvilBugToEvilBug();
            CollideBugToBugZapper();
            CollideEvilBugToBugBugZapper();

            // Check for Game Over
            if (maxbugs < deadbugs+1 &&
                maxevilbugs < deadevilbugs+1)
                break out;
            for (int x=0; x<maxbugs; x++)
                if (parray[x].isAlive() &&
                    p.impactBug(parray[x]))
                    break out;
            for (int x=0; x<maxevilbugs; x++)
                if (earray[x].isAlive() &&
                    earray[x].impactBug(p))
                    break out;
            for (int y=0; y<maxbug_zappers; y++)
                if (rarray[y].impactBug(p)) break out;

            pause(50);
        }
    }

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

    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();  
    }  
}  
}
```

23. Running the Game in Its Final Form

Now when the file `MainPanel.java` is compiled:

```
javac MainPanel.java
```

and the file `JavaByBugs.class` is run:

```
java JavaByBugs
```

this is an example of the game that will be produced and can be played by the user:

