```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.*;


/**
 * AlienAttack - 1030 GUI Demonstration.<p>
 *
 * @author William Soukoreff
 * (Based on "DemoTimer" originally written by Scott MacKenzie)
 */
public class AlienAttack extends JFrame implements KeyListener, ActionListener, ImageO
bserver
{
   public static void main(String[] args)
   {
      AlienAttack jframe = new AlienAttack();
      jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jframe.setTitle("AlienAttack");
      jframe.pack();
      jframe.setVisible(true);
   }


   /*
    * here we define some sprites (images) we'll be using
    */
   Sprite spaceship = new Sprite("spaceship.gif");

   Sprite explosion = new Sprite("explosion.gif");

   Sprite missile_with_fire = new Sprite("missile-fire.gif", 1000, 1000);  // no accid
ental explosions
   Sprite missile_no_fire   = new Sprite("missile-nofire.gif");


   /*
    * some constants
    */
   static final int PANEL_WIDTH  = 400;
   static final int PANEL_HEIGHT = 400;

   static final Color GROUND_COLOR = new Color(153, 102, 51);
   static final Color SKY_COLOR    = new Color(204, 236, 255);
   static final Color TEXT_COLOR   = Color.yellow;

   // this is the number of milliseconds between frame redraws
   static final int DELAY = 30;

   // this is the timer we'll use to time the frame redraws
   javax.swing.Timer t;

   // some text fields for output messages
   JTextField countdown;
   JTextField instructions;

   // this is the JPanel where we'll draw the game
   GamePanel gamepanel;
```

```
   /*
    * the constructor for our JFrame object
    */
   public AlienAttack()
   {

      /*
       * construct and configure GUI components
       */

      Dimension d;

      instructions = new JTextField();
      instructions.setEditable(false);
      instructions.setBorder(null);
      instructions.setHorizontalAlignment(JTextField.CENTER);
      instructions.setBackground(GROUND_COLOR);
      instructions.setForeground(TEXT_COLOR);
      d = instructions.getPreferredSize();
      instructions.setPreferredSize(new Dimension(PANEL_WIDTH, d.height));
      instructions.setMaximumSize(new Dimension(PANEL_WIDTH, d.height));
      instructions.setText("INSTRUCTIONS: Cursor Keys and Spacebar!");

      countdown = new JTextField();
      countdown.setEditable(false);
      countdown.setHorizontalAlignment(JTextField.CENTER);
      countdown.setFont(new Font("sansserif", Font.PLAIN, 24));
      countdown.setBorder(null);
      countdown.setBackground(GROUND_COLOR);
      countdown.setForeground(TEXT_COLOR);
      d = countdown.getPreferredSize();
      countdown.setPreferredSize(new Dimension(PANEL_WIDTH, d.height));
      countdown.setMaximumSize(new Dimension(PANEL_WIDTH, d.height));
      countdown.setText("Score: 000");

      gamepanel = new GamePanel(this);
      gamepanel.setBackground(SKY_COLOR);
      gamepanel.setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
      gamepanel.setMaximumSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));


      /*
       * setup the sprites
       */

      // register the images
      gamepanel.registerSprite(new Sprite("clouds.gif",  20,  20, true));
      gamepanel.registerSprite(new Sprite("clouds.gif", 300, 100, true));
      gamepanel.registerSprite(new Sprite("clouds.gif", 100, 150, true));

      gamepanel.registerSprite(explosion);

      missile_no_fire.setXY(
         (PANEL_WIDTH - missile_no_fire.getWidth()) / 2,
         PANEL_HEIGHT - missile_no_fire.getHeight()
      );

      gamepanel.registerSprite(missile_with_fire);
      gamepanel.registerSprite(missile_no_fire);
      gamepanel.registerSprite(spaceship);

      missile_no_fire.setVisible(true);
```

```
      /*
       * startup the timer
       */

      t = new javax.swing.Timer(DELAY, this);
      t.start();


      /*
       * add the key listener
       */

      this.addKeyListener(this);


      /*
       * final GUI setup - layout the components and setup the content pane
       */

      // add components to panels

      JPanel bottomPanel = new JPanel(new BorderLayout());
      bottomPanel.add(countdown, "North");
      bottomPanel.add(instructions, "South");
      bottomPanel.setMaximumSize(bottomPanel.getPreferredSize());

      JPanel contentPane = new JPanel();
      contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
      contentPane.add(gamepanel);
      contentPane.add(bottomPanel);

      // make paint panel this JFrame's content pane

      setContentPane(contentPane);
   }


   /*
    * we need this function to allow us to receive keyboard input
    */

   public boolean isFocusable()
   {
      return true;
   }


   /*
    * implement KeyListener methods
    */

   public void keyTyped(KeyEvent ke) {}

   public void keyPressed(KeyEvent ke)
   {
      // if the user hits the SPACE bar... FIRE
      if(ke.getKeyCode() == KeyEvent.VK_SPACE)
      {
         // hide the launchpad (no-fire) missile
         missile_no_fire.setVisible(false);

         // set the position of the fire missile to be where the
         // no-fire missile was
```

```
         int x = missile_no_fire.getX()
               - (missile_with_fire.getWidth() - missile_no_fire.getWidth())/2;
         missile_with_fire.setXY(x, missile_no_fire.getY());

         // set the upward speed, and make it visible
         missile_with_fire.setSpeedXY(0, -0.8);
         missile_with_fire.setVisible(true);
      }

      // cursor keys move the missile
      if(ke.getKeyCode() == KeyEvent.VK_RIGHT)
      {
         missile_no_fire.setSpeedXY(0.4, 0);
      }

      // cursor keys move the missile
      if(ke.getKeyCode() == KeyEvent.VK_LEFT)
      {
         missile_no_fire.setSpeedXY(-0.4, 0);
      }

      // Escape exits the game
      else if (ke.getKeyCode() == KeyEvent.VK_ESCAPE)
         System.exit(0);
   }

   // if the user is not holding down either of the cursor
   // keys, then we stop the missile from moving
   public void keyReleased(KeyEvent ke)
   {
      missile_no_fire.setSpeedXY(0, 0);
   }


   /*
    * implement ActionListener method
    *
    * the action listener gets called by the timer, every
    * DELAY milliseconds
    *
    * so this is where the main control loop of the game is
    */

   // we need some randomness, so we need a random number generator
   private Random random = new Random();

   // this counts down the millisconds between appearences of the
   // ufo spaceship
   int spaceshipDelay = -1;

   // this tells us that we're doing an explosion right now, which
   // basically means that the game is paused while we wait for the
   // explosion to finish
   boolean isExploding = false;

   // this counts down the 19 frames of the explosion, when it hits
   // zero, the explosion animation is over
   int explosionFrameCounter = 0;


   /*
    * this function is responsible for managing the behavior of the
    * ufo spaceship
    */
```

```
   void manageSpaceShip()
   {
      if(spaceship.isVisible())
      {
         if(spaceship.getX() > PANEL_WIDTH + 10
         || spaceship.getX() < -spaceship.getWidth() - 10)
         {
            // if the spaceship is visible, but has run all the way
            // accross the screen, then we set up a short random
            // delay, before the ship appears again

            spaceship.setVisible(false);
            spaceship.setSpeedXY(0, 0);
            int rand = random.nextInt(100);
            spaceshipDelay = (int)(2000 + 1000 * random.nextGaussian());
         }
      }
      else if(!isExploding)
      {
         // if the spaceship is not exploding, then we wait for the
         // random delay to run-out (we don't want too many spaceships
         // to appear too often)

         spaceshipDelay -= DELAY;
         if(spaceshipDelay < 0)
         {
            // once the delay is over, we're ready to make a new
            // spaceship appear
            //
            // first we need to decide whether the spaceship is going to
            // enter from the left or right?
            //
            // then we need to randomise the height and speed

            if(random.nextBoolean())
            {
               spaceship.setXY(-spaceship.getWidth(), 10 + random.nextInt(150));
               spaceship.setSpeedXY(0.4 + random.nextDouble()/10.0, 0);
            }
            else
            {
               spaceship.setXY(PANEL_WIDTH, 10 + random.nextInt(150));
               spaceship.setSpeedXY(-0.4 - random.nextDouble()/10.0, 0);
            }

            spaceship.setVisible(true);
         }
      }
   }


   /*
    * this function is responsible for managing the missile
    *
    * the missile is easy - once it has been fired, we just have to
    * wait for it to hit the top of the screen, at which point we
    * shut off the fired missile, and turn back on the no-fire missile
    */
   void manageMissile()
   {
      if(missile_with_fire.getY() < -missile_with_fire.getHeight())
      {
         missile_with_fire.setVisible(false);
         missile_with_fire.setSpeedXY(0, 0);
```

```
            missile_no_fire.setVisible(true);
        }
    }


    /*
     * this code checks to see whether the missile and the spaceship
     * have hit each other, and if so, it starts the explosion
     * animation playing
     *
     * it also keeps score
     */
    int score = 0;

    void checkForCollision()
    {
        // if we're already exploding, then don't do anything
        if(isExploding)
            return;

        // let's calculate the deltas - how far away is the centre of
        // the missile from the centre of the spaceship?

        int sx = spaceship.getX() + spaceship.getWidth()/2;
        int sy = spaceship.getY() + spaceship.getHeight()/2;
        int mx = missile_with_fire.getX() + missile_with_fire.getWidth()/2;
        int my = missile_with_fire.getY() + missile_with_fire.getHeight()/2;

        int dx = sx - mx;  dx = (dx < 0 ? -dx : dx);
        int dy = sy - my;  dy = (dy < 0 ? -dy : dy);

        // if the centres are within 50 pixels, then we call it a hit
        if(dx < 50 && dy < 50)
        {
            // setup the explosion

            isExploding = true;
            spaceship.setVisible(false);

            explosion.setXY(sx - explosion.getWidth()/2, sy - explosion.getHeight()/2);

            // these restart the explosion animation, and the frame counter
            explosion.getImage().flush();
            explosionFrameCounter = 19;

            explosion.setVisible(true);

            // keep score
            score += 100;
            countdown.setText("Score: " + score);
        }
    }


    /*
     * This ActionListener function is called every DELAY ms
     * and it calls the various manage*() functions above.
     */
    public void actionPerformed(ActionEvent ae)
    {
        manageSpaceShip();
        manageMissile();
        checkForCollision();
```

```
      Sprite.MoveSprites(DELAY);
      gamepanel.repaint();
   }


   /*
    * this is the explosion animation frame counter
    *
    * every time it gets called with the FRAMEBITS flag set,
    * means that the time between another frame has elapsed
    */
   public boolean imageUpdate( Image img, int flags, int x, int y, int w, int h )
   {
      if(!isExploding)
         return true;

// debugging code:
//      System.out.println("Image update: flags="+flags+" x="+x+" y="+y+" w="+w+" h="+
h);

      if((flags & ImageObserver.FRAMEBITS) != 0 && (explosionFrameCounter-- < 0))
      {
         explosion.setVisible(false);
         isExploding = false;
      }

// normally you'd do this,, because if you don't keep calling
// repaint(), then the animation in the GIF won't work, but
// because our main control loop is regularly and frequently
// calling repaint(), we don't need to also be doing this here
//      repaint();

      return true;
   }
}
```