```java
package asteroids2053;

import javax.swing.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.util.*;
//import java.io.*;
//import javax.imageio.*;
//import java.net.*;

import net.minimumrage.util.Debug;

/**
 * Universe.java
 *
 * Universe is the 2d space that contains all the game objects. Ship, asteroids,
 * bullets, etc. It houses the main game thread that controls action and display
 * of these objects. It's a panel and can be placed in containers like any other
 * panel. Behaviour when given sizes outside of the expected 1024x768 is
 * currently undefined. I think it'll be fine, but I can't be sure.
 *
 * The universe wraps around itself, so that if you go off the top of the screen
 * you appear at the bottom, etc. This is just like real life, as per my
 * understanding of real life, which is not really a very robust understanding.
 *
 * The universe itself is actually bigger than you think. It extends beyond the
 * visible range by 100x100 pixels. This is so there's an invisible zone outside
 * in which ships and asteroids and such can go. The reasons are twofold: first,
 * so that the universe appears bigger and is not so claustrophobic. Also, so we
 * don't have to draw asteroids half on one side and half on the other. That
 * would be a lot of work, and so it's just better if it makes that transition
 * in invisible land.
 *
 * @author Richard Banister, Will Matthaeus
 * @version 1.0
 */
public class Universe extends JComponent implements CollisionListener, ImageObserver {

    private static final long serialVersionUID = -6054799563430853995L;

    // in debug mode, some info is displayed in top right corner
    public static boolean debug = false;
```

```
/*
 * input acceptors booleans to maintain status of direction keys (on/off),
 * the fire key and thruster key. Turn on when a key pressed, off when
 * released. Then check for them at every game cycle. This is better than
 * just performing an action every time you detect a key press, since who
 * knows how many presses will be accurately detected in a second?
 */
private boolean up = false;
private boolean down = false;
private boolean left = false;
private boolean right = false;
private boolean fire = false;
// private boolean killThrust = false; //TODO what was this for?

// boolean that maintains the run status of the game. If it's not running,
// don't bother updating the graphics
private boolean running = false;

// prefered screen width and height. TODO: selectable resolutions?
public static final int PREF_WIDTH = 1024;
public static final int PREF_HEIGHT = 768;

// SPACE OBJECTS
// a ship, some asteroids
private Ship ship;
private ArrayList<Asteroid> asteroids = new ArrayList<Asteroid>();

// HUD
private HealthIndicator healthIndicator;
private ScoreIndicator scoreIndicator;

// private boolean GODMODE = false; //maintain god mode in the ship now

// allows for customization of ship controls
// TODO: customize controls screen
private Controls controls = new Controls();

// THREAD CONTROLS
// thread for the game, initial last time an update was done,
// how long the update took to complete
// and an int to store and display the frames per second
private GameThread gameThread;
```

```
private long lastUpdate = -1;
private long updateTime;
private int fps;

// DRAWING
// buffer object and associated image for switching to display, using
// double buffering. We'll draw everything to the buffer then swap it in.
private Image offscreenImage;
private Graphics2D buffer;

// images object storing most of the game images
private Images images;

// imageicons for messages and explosion
private ImageIcon explosion, gameover, gameoverMessage;

// long storing time of death, if the player died. Used to determine how
// long to display game over messages, then return to title screen
private long timeofdeath;

// EXPLOSION
/*
 * x,y coords of the ship explosion. This is something of a hack, i guess.
 * The ship becomes a null object when its exploded and gets sent out into
 * deep space. Somehow the explosion needs to stay behind, and not affect
 * any other space objects. Of course, this makes me wonder what it will
 * take to show explosions for asteroids and stuff.
 */
private double ex, ey;
int animCount = 0;
boolean animDone = false;

GamePanel game; // we need this so we can make callbacks when games are over

/* returns the top of the universe (ie x coord) */
public int getUpperBound() { return 0; }

/* returns the leftmost coord of the universe */
public int getLeftBound() { return 0; }

/*
 * returns the bottomost universe boundary. Note that the universe is
 * actually bigger than it appears
```

```
 */
public int getLowerBound() { return getHeight() + 100; }

/*
 * returns the rightmost universe boundary. Note that the universe is
 * actually bigger than it appears
 */
public int getRightBound() { return getWidth() + 100; }

// detector for space object collisions.
private ImpactDetector impactDetector;

// score storage
private long score;

/**
 * Create a new universe (whooooooo!)
 *
 * @param game
 *            The parent GameFrame. This will be called when the game is
 *            over. Must not be null.
 */
public Universe(GamePanel game) {
    setOpaque(false);
    this.game = game;
}

/**
 * initialize the universe; does everything necessary prior to the running
 * of the game: create ship at starting point; gets and preload images;
 *
 * tries to load images for explosion, gameover and gameovermessage. Note
 * then that images are loaded in separate places (class Image and here).
 * This should probably be changed.
 *
 * This must prepare for the beginning of a game, whether or not a game has
 * been played already. We can't count on the universe being constructed
 * directly beforehand.
 */
public void init() {

    spawnShip();
```

```
        images = Images.getImages(this);
        images.preload();

        try {
            // File f = new File("resources/shipExplosion.gif");
            // URL url = f.toURI().toURL();
            // explosion = new ImageIcon(url);
            explosion = images.getImage(Images.SHIP_EXPLOSION);

            // File f1 = new File("resources/gameover.gif");
            // URL url1 = f1.toURI().toURL();
            // gameover = new ImageIcon(url1);
            gameover = images.getImage(Images.GAME_OVER);

            // File f2 = new File("resources/gameoverMessage.gif");
            // URL url2 = f2.toURI().toURL();
            // gameoverMessage = new ImageIcon(url2);
            gameoverMessage = images.getImage(Images.GAME_OVER_MESSAGE);

        } catch (Exception e) {
            // TODO no exceptions should be thrown here now, i think
            Debug.out("Failed to load a Universe image: " + e.getMessage());
        }

        healthIndicator = new HealthIndicator(getWidth() - 40, getHeight() - 160, 60, 180);
        // i don't know how that makes sense

        score = 0;
        scoreIndicator = new ScoreIndicator(80, 80, 300, 72);
        scoreIndicator.setScore(score);

        initAsteroids();

        this.addKeyListener(new Keyser());

        impactDetector = new ImpactDetector();
        impactDetector.addCollisionListener(this);

        // check validity!!!
        // getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_UP, 0), "up");
        // getActionMap().put("up", new PlayerAction());
    }
```

```java
/**
 * Place some asteroids on the screen. This could be modified if we wanted
 * to implement leveling at some point.
 *
 * @see createAsteroid
 */
public void initAsteroids() {
    asteroids = new ArrayList<Asteroid>();

    createAsteroid();
    createAsteroid();
    createAsteroid();
    createAsteroid();

}

/** return the score */
public long getScore() {
    return score;
}

/**
 * Initialize the ship in the starting place. Right now this makes a fighter
 * ship. Initialize explosion and stuff also. TODO different ship types
 *
 * @see FighterShip
 */
public void spawnShip() {
    ship = new FighterShip(this, PREF_WIDTH / 2 + 50, PREF_HEIGHT / 2 + 50);
    animCount = 0;
    animDone = false;
}

/*
 * handle collision event. universe receives a CollisionEvent, tells the
 * space objects what to do. Should this be handled by individual objects,
 * or does it make more sense to control everything in the universe? think
 * about this, maybe TODO
 *
 * @param ce CollisionEvent
 *
 * @see SpaceObject
 *
```

```
     * @see scoreIndicator
     *
     * @see Bullet
     */
    public void collision(CollisionEvent ce) {
        // Debug.out(ce.c1 + " collided with "+ce.c2);

        // update score?
        double points = ce.distance;
        SpaceObject obj1 = (SpaceObject) ce.c1;
        points *= obj1.getScoreMultiplier();
        SpaceObject obj2 = (SpaceObject) ce.c2;
        points *= obj2.getScoreMultiplier();
        score += (int) points;
        scoreIndicator.setScore(score);

        double damage = 1.0;
        if (!(obj1 instanceof Bullet)) damage = Math.abs(getImpulse(obj1, obj2));

        /*
         * if (!(obj1 instanceof Ship) || !GODMODE) if (obj1.damage(damage))
         * obj1.destroy();
         */
        if (obj1.damage(damage)) {
            obj1.destroy();
            if (obj1 == ship) {
                ex = ship.getX();
                ey = ship.getY();
                ship = ship.expire();
                timeofdeath = System.currentTimeMillis();
                Debug.out("TIME OF DEATH: " + timeofdeath);
            }
        }
        if (obj2.damage(damage)) {
            obj2.destroy();
        }

        // ce.c1.destroy(); ce.c2.destroy();
    }

    /*
     * Uses the velocity and mass of two space objects that have collided to
     * calculate their changes in momentum and thereby their resulting motion.
```

```
     *
     * @param a first space object (the ship)
     *
     * @param b the second space object (the asteroid)
     *
     * @see Momentum
     *
     * @see SpaceObject
     */
    @SuppressWarnings("unused")
    public double getImpulse(SpaceObject a, SpaceObject b) {
        double v2b, v2a, j, vax, vbx, vay, vby, ma, mb, nx, ny, A, B, ax, ay, bx, by;
        Momentum moma, momb, momn;

        // x velocities
        vax = a.momentum.getVX();
        vbx = b.momentum.getVX();

        // y velocities
        vay = a.momentum.getVY();
        vby = b.momentum.getVY();
        // normal vector
        momn = new Momentum(a.getCX(), a.getCY(), b.getCX(), b.getCY());
        double angle = momn.getAngle();

        nx = momn.x1 + Math.sin(angle);
        ny = momn.y1 + Math.cos(angle);

        // nx = b.getX()-a.getX();
        // ny = b.getY()-a.getY();
        momn.x2 = nx;// momn.x1 + Math.sin(angle);
        momn.y2 = ny;// momn.y1 + Math.cos(angle);

        // masses
        ma = a.getMass();
        mb = b.getMass();

        // IMPULSE!!!!
        // A = relative velocity dot product normal vector
        A = ((vax - vbx) * Math.sin(angle)) + ((vay - vby) * Math.cos(angle));
        // B = inverse masses added
        B = (1 / ma) + (1 / mb);
        // j = IMPULSE or change in momentum
```

```
        j = -2.0 * (A) / B;
        Debug.out(/*
                              * "vax:"+vax+"; vab:"+vbx+"; angle:"+
                              * angle+"; vay:"
                              * +vay+"; vby:"+vby+"; A:"+A+"; B:"+B+
                              */"; j:" + j);
        // each vector after collision = vector before collision
        // normal vector. may have to reverse a and b.

        // momn = new Momentum(a.getX(), a.getY(), b.getX(), b.getY());
        // double angle = momn.getAngle();

        // set magnitude to 1.0

        ax = a.getCX();
        ay = a.getCY();
        bx = b.getCX();
        by = b.getCY();

        //Check initial and final kinetic energies to make sure they match:
        //(removed for efficiency)
        //double k1, k2;
        //k1 = (.5 * ma * Math.pow(a.momentum.getMagnitude(), 2)) + (.5 * mb *
Math.pow(b.momentum.getMagnitude(), 2));

        v2a = a.momentum.push(ax, ay, ax + (j * Math.sin(angle) / ma), ay + (j * Math.cos(angle) / ma), j, ma);

        v2b = b.momentum.push(bx, by, bx + (j * Math.sin(angle) / mb), by + (j * Math.cos(angle) / mb), j, mb);

        // check final kinetic energies:
        //k2 = (.5 * ma * Math.pow((v2a), 2)) + (.5 * mb * Math.pow((v2b), 2));
        // Debug.out("k1 = "+k1+"; k2 = "+k2 );

        return j;
    }

    /*
     * protocol for destruction of asteroid as a result of collision and for
     * creation of its two daughter asteroids (createAsteroid())
     *
     * @param ast Asteroid to be destroyed
     */
    public void asteroidDestroyed(Asteroid ast) {
```

```
    for (int i = 0; i < asteroids.size(); i++) {
        if (ast == asteroids.get(i)) {
            asteroids.remove(i);
            // make two new ones, add them to the array
            createAsteroid();
            createAsteroid();
        }
    }
}

/**
 * Create an asteroid randomly. It will be attached to one of the edges of
 * the screen, so it looks like it's coming in from outer space or
 * something.
 */
public void createAsteroid() {
    double x = getLeftBound(), y = getUpperBound();
    double randX = Math.random() * (PREF_WIDTH + 100);
    double randY = Math.random() * (PREF_HEIGHT + 100);
    int select = (int) randX % 4;
    Debug.out(randX + "," + randY + " selected " + select);
    switch (select) {
        case 0:
            x = randX;
            break; // attached to top
        case 1:
            x = PREF_WIDTH + 100;
            y = randY;
            break; // attached to left
        case 2:
            x = randX;
            y = PREF_HEIGHT + 100;
            break; // attached to bottom
        case 3:
            y = randY;
            break;
    }
    asteroids.add(new Asteroid(this, x, y));
}


/**
 * Does all the init needed to begin a game, then starts a new GameThread.
```

```
 */
public void start() {

    offscreenImage = createImage(getWidth() + 100, getHeight() + 100);
    buffer = (Graphics2D) offscreenImage.getGraphics();

    running = true;

    gameThread = new GameThread();
    gameThread.start();
}

/*
 * Flags the universe for destruction. This lets the GameThread know that
 * we're done with life and at the next opportunity it can return us to
 * the title screen.
 */
public void stop() {
    gameThread.killMe = true;
}

/*
 * returns the Dimension object for the prefered size of the panel
 */
public Dimension getPreferredSize() {
    return new Dimension(PREF_WIDTH, PREF_HEIGHT);
}

/*
 * returns the Dimension object for the minimum size of the panel
 */
public Dimension getMinimumSize() {
    return new Dimension(PREF_WIDTH, PREF_HEIGHT);
}

/*
 * returns the Dimension object for the maximum size of the panel
 */
public Dimension getMaximumSize() {
    return new Dimension(PREF_WIDTH, PREF_HEIGHT);
}

/**
```

```
     * Determine if an animated image should update itself or not. In particular,
     * this is for the ship explosion image. The animation should run for 10
     * frames if the ship has been destroyed, then let us know that it has
     * finished by setting the animDone bool.
     * Yeah, it's a hack.
     */
    public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
        // Debug.out("flags: "+flags);
        if (flags == 16) animCount++;
        // repaint();
        if (animCount >= 10) {
            animDone = true;
            return false;
        }
        return true;
    }

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

    /*
     * Draws a frame of the game. First write everything to a buffer (using
     * the offscreenImage object init'd in start()), then write that real quick
     * to the passed in Graphics.
     *
     * Handles some logic in the sense of determining whether to draw a dying
     * ship or a living one, e.g., but in general there shouldn't be any side
     * effects in here.
     *
     * @param gMoney Graphics object for universe
     *
     */
    public void paint(Graphics gMoney) {
        if (!running) return;

        Graphics2D g = buffer;

        // set rendering options
        // this may slow things down too much...
        g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
        g.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
```

```
g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
g.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);

// /// BACKGROUND /////
// load a beautiful image
g.setColor(Color.BLACK);
// g.fillRect(0,0,getWidth()+100,getHeight()+100);
// g.drawImage(images.getSpaceImage(), 50,50,null);
// g.drawImage(images.getImage(Images.SPACE), 50, 50, null);
images.getImage(Images.SPACE).paintIcon(this, g, 50, 50);

// ship
ship.draw(g);

if (ship.getHealth() <= 0 && !animDone) {

    // g.drawImage(images.getShipExplosionImage(), (int)ship.getX(),
    // (int)ship.getY(), null, this);
    explosion.paintIcon(this, g, (int) ex, (int) ey);
    // images.getShipExplosion().paintIcon(this, g, (int)ship.getX(),
    // (int)ship.getY());

}

// asteroids
for (int i = 0; i < asteroids.size(); i++) {
    asteroids.get(i).draw(g);
}

g.setFont(new Font("Verdana", Font.BOLD, 48));
g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
g.setColor(new Color(100, 180, 160));
// g.drawString(score+"", 60, 96);

scoreIndicator.draw(g);

// HEALTH
// g.drawString((int)ship.getHealth()+"", 60, getHeight()-24);
double healthPercent = ship.getHealth() / ship.getMaxHealth();
healthIndicator.setHealth(healthPercent);
healthIndicator.draw(g);

// game over?
```

```
    int postmortem = (int) (System.currentTimeMillis() - timeofdeath);
    if (ship.getHealth() <= 0 && postmortem >= 1000) {
        if (postmortem >= 3000) {
            // then either display bad message if their score sucksx
            // or quit to title
            if (score > 100000 || postmortem > 7000) {
                // QUITUITDUID

            } else {
                // display message
                gameoverMessage.paintIcon(null, g, 112, 184);
            }
        } else {
            // g.drawImage(gameover, 112, 184, null, null);
            gameover.paintIcon(null, g, 212, 284);
        }
    }// else Debug.out(postmortem);

    // write debug info?
    if (debug) {
        int rightside = getWidth() - 100;

        g.setColor(Color.CYAN);
        g.setFont(new Font("Verdana", Font.PLAIN, 12));
        if (updateTime == 0) updateTime = 1;
        g.drawString("Update Time: " + updateTime, rightside, 65);
        g.drawString("FPS: " + fps, rightside, 77);
        g.drawString(frameCount + "", rightside + 60, 77);
        if (up) g.drawString("U", rightside + 12, 89);
        if (left) g.drawString("L", rightside, 101);
        if (down) g.drawString("D", rightside + 12, 101);
        if (right) g.drawString("R", rightside + 24, 101);
        if (fire) g.drawString("*", rightside + 24, 89);

    }

    gMoney.drawImage(offscreenImage, -50, -50, this);

    super.paint(g);

    frameCount++;
}
```

```
    int frameCount = 0;

    /*
     * This thread represents a game of Asteroids2053
     */
    class GameThread extends Thread {
        public boolean killMe = false; //whether we should exit the thread and return to title

        public void start() {
            // Debug.out("game thread starting");
            Debug.out("game thread starting");

            // run();
            super.start();
        }


        /**
        * The main game loop. Moves and updates all game objects, handles game
        * logic, requests a repaint. We aim for 60fps; but if we dip below that
        * the flow of time should remain constant.
        *
        * This will terminate some seconds after the ship explodes, or when
        * it is specifically requested (e.g. by the user quitting).
        */
        public void run() {
            long nextFPS = System.currentTimeMillis();
            frameCount = 0;
            Debug.out("Running");

            lastUpdate = nextFPS;

            while (true) {
                if (killMe) return;

                if (ship.getHealth() <= 0 && System.currentTimeMillis() - timeofdeath >= 7000)
game.stopper(true);

                long cTime = System.currentTimeMillis();
                if (cTime > nextFPS) {
                    fps = frameCount;
                    frameCount = 0;
                    nextFPS = nextFPS + 1000;
```

```
    // Debug.out("resetting fps; got: "+fps);
}

// move everything
long timeSince = Math.abs(lastUpdate - cTime);
// move everything to new positions
for (int i = 0; i < asteroids.size(); i++) {
    asteroids.get(i).move(timeSince);
}
ship.move(timeSince);

// process controls
// TODO: should we do timeSince here, also?
if (up) ship.forward();
else ship.stopThrust();
if (right) ship.rotateRight();
if (left) ship.rotateLeft();
if (down) ship.back();

if (fire) ship.fire();

// ZE BUYETS!
ship.moveBullets(timeSince);

// collision detection here
// impactDetector imp = new impactDetector(ship, asteroids,
// ship.magazine);
impactDetector.checkForImpact(ship, asteroids, ship.magazine);

// redraw everything
repaint();
// frameCount++;

// how long did that take?
lastUpdate = System.currentTimeMillis();
updateTime = lastUpdate - cTime; // time it took to run that
                                 // update

long sleepTime = 16 - updateTime; // how many millis? 16 ~= 60
                                  // frames/sec

// sleep?
if (sleepTime > 0) try {
```

```
                sleep(sleepTime);
            } catch (Exception te) {
                te.printStackTrace();
            }
        }
    }
}

/*
 * processes key input event
 */
public void key(KeyEvent ke) {
    processKeyEvent(ke);
}

/*
 * public void processKeyEvent(KeyEvent ke){ //do nothing TESTTESTTESTTEST
 * //MWAJAJAAJAJAJAAHAHAHAHAAHAHAAAAA
 *
 *
 *
 * }
 */

/*
 * class to detect and process player action:
 *
 *
 * THIS IS A TEST! Testing a different way of grabbing user input, instead
 * ofthrough listeners. It works, I think. But benefits over listeners?
 * DUnno.
 */
class PlayerAction extends AbstractAction {

    private static final long serialVersionUID = -2326264552369068333L;

    public void actionPerformed(ActionEvent e) {
        Debug.out("cello?");
    }
}

/**
 * Handle key events. Ship movement controls are defined in Controls; they're
```

```
 * handled here. Also keep track of special keys like godmode, debugging
 * mode, ship respawn, etc.
 * @see Controls
 */
public class Keyser implements KeyListener {
    /**
     * Handle keys that should produce action until they are released here.
     * Thrust, turning, firing. The user holds these keys down until they
     * want to stop; so we set a boolean here that we unset upon release.
     * We set these flags instead of calling the actions directly because
     * we can't be sure how often we'll get these events. It certainly
     * won't correspond to the game engine. So by setting flags we tie
     * ship movement to the game engine.
     */
    public void keyPressed(KeyEvent e) {

        // TODO: Problem with three keys pressed? e.g. UP + LEFT + FIRE
        // doesn't work

        int dir = controls.getAction(e.getKeyCode());

        if (dir == Controls.UP) up = true;
        // making down a momentary switch
        // else if (dir == Controls.DOWN) down = true;
        else if (dir == Controls.LEFT) left = true;
        else if (dir == Controls.RIGHT) right = true;
        else if (dir == Controls.FIRE) fire = true;

        // e.consume();
        // Debug.out("pressed: "+dir);
    }

    /**
     * When the user releases a key we deactivate the corresponding action.
     *
     * Also handles momentary controls: emergency teleport, dev toggles, quit, etc.
     * That means these actions take place upon key release, not upon key
     * press, but that shouldn't matter to the user, right?
     */
    public void keyReleased(KeyEvent e) {
        int dir = controls.getAction(e.getKeyCode());
        switch (dir) {
            case Controls.UP:
```

```
                up = false;
                break;
            // case Controls.DOWN: down = false; break;
            case Controls.DOWN:
                ship.back();
                break;
            case Controls.LEFT:
                left = false;
                break;
            case Controls.RIGHT:
                right = false;
                break;
            case Controls.FIRE:
                fire = false;
                break;
            default:
                switch (e.getKeyCode()) {
                    case KeyEvent.VK_7:
                        ship.GODMODE = !ship.GODMODE;
                        Debug.out("GODMODE " + ship.GODMODE);
                        break;
                    case KeyEvent.VK_8:
                        debug = !debug;
                        if (debug) Debug.on();
                        else Debug.off();
                        Debug.out("DEBUG " + debug);
                        break;
                    case KeyEvent.VK_0:
                        spawnShip();
                        Debug.out("SPAWNING");
                        break;
                    case KeyEvent.VK_ESCAPE:
                        // break out to uiPanel
                        game.stopper(false);
                }
        }
        // Debug.out("released: "+dir);
    }

    public void keyTyped(KeyEvent e) {
        // Debug.out("typed");
    }
}
```

```
}
```