World Classes

StartScreen

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import greenfoot.Color;
import java.io.*;
import java.awt.*;
/**
 * Start Screen of game.
 */
public class StartScreen extends World
{

    /**
     * Constructor for objects of class StartScreen.
     *
     */
    public StartScreen()
    {
        super(600, 600, 1);
        GreenfootImage background = new GreenfootImage("start1.png");
        background.setColor(Color.BLACK);
        background.fill();


        createStars(706);

    }

    /**
     * Calling the method checkKey.
     */
    public void act()
    {
        checkKey();
    }

    /**
     * Creating the space background by adding stars.
     */
    private void createStars(int number)
    {
        GreenfootImage background = getBackground();
        for(int i=0;i<number;i++)
        {
            int x = Greenfoot.getRandomNumber(getWidth());
            int y = Greenfoot.getRandomNumber(getHeight());
            int color = 120 - Greenfoot.getRandomNumber(100);
            background.setColor(new Color(color, color, color));
            background.fillOval(x,y,2,2);
        }
    }
    /**
     * Method which takes user to level 1.
     */
    private void checkKey()
    {
        if (Greenfoot.isKeyDown("space"))
        {
            Greenfoot.delay(25);

            Greenfoot.setWorld(new Level1());
        }

        if (Greenfoot.isKeyDown("F1"))
        {
            try {Desktop.getDesktop().open(new File("UserGuide.pdf"));}
            catch (IOException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

Level 1

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)


/**
 * The Search for Home
 */
public class Level1 extends World
{
    public Counter scoreCounter;
    private int startAsteroids = 6;
    public int numberOfObjects;

    /**
    * Constructor
    */
    public Level1()
     {
         super(600, 600, 1);
         GreenfootImage background = getBackground();
         background.setColor(Color.BLACK);
         background.fill();

         createStars(706);

         StarFighter1 rocket = new StarFighter1();
         addObject(rocket, getWidth()/2 + 100, getHeight()/2);

         addAsteroids(startAsteroids);

         scoreCounter = new Counter("Score: ");
         addObject(scoreCounter, 60, 580);

         Explosion.initializeImages();
         ProtonWave.initializeImages();

         prepare();
     }
     /**
     * background stars
     */
    private void createStars(int number)
     {
         GreenfootImage background = getBackground();
         for(int i=0;i<number;i++)
         {
             int x = Greenfoot.getRandomNumber(getWidth());
             int y = Greenfoot.getRandomNumber(getHeight());
             int color = 120 - Greenfoot.getRandomNumber(100);
             background.setColor(new Color(color, color, color));
             background.fillOval(x,y,2,2);
         }
```

```
    }
    /**
     * This method gets called in the constructor to spawn asteroids.
     */
    private void addAsteroids(int count)
    {
        for(int i = 0; i < count; i++)
        {
            int x = Greenfoot.getRandomNumber(getWidth()/2);
            int y = Greenfoot.getRandomNumber(getHeight()/2);
            addObject(new Asteroid(), x, y);
        }
    }

    /**
     * This method is called when the game is over to display the final score.
     */
    public void gameOver()
    {
        addObject(new ScoreBoard(scoreCounter.getValue()), 300, 300);
    }
    public void gameOverWithMessage()
    {
        addObject(new ScoreBoard("You Win!",scoreCounter.getValue()), 300, 300);
    }
    /**
     * Prepares all obstacles.
     * This method gets called in order to prepare all obstacles.
     */
    private void prepare()
    {
        Asteroid asteroid = new Asteroid();
        addObject(asteroid, 417, 242);
        Asteroid asteroid2 = new Asteroid();
        addObject(asteroid2, 368, 283);
        Asteroid asteroid3 = new Asteroid();
        addObject(asteroid3, 341, 383);
        Asteroid asteroid4 = new Asteroid();
        addObject(asteroid4, 120, 317);
        Asteroid asteroid5 = new Asteroid();
        addObject(asteroid5, 458, 134);
        Asteroid asteroid6 = new Asteroid();
        addObject(asteroid6, 481, 332);
        Asteroid asteroid7 = new Asteroid();
        addObject(asteroid7, 57, 185);
        asteroid.setLocation(413, 342);
        removeObject(asteroid6);
        removeObject(asteroid);
        removeObject(asteroid3);
        removeObject(asteroid2);
        removeObject(asteroid5);
        removeObject(asteroid4);
        removeObject(asteroid7);
    }

    /**
     * Score counter method used to keep score and add +1 for every enemy
     * destroyed.
     */
    public void countScore()
    {
        scoreCounter.add(1);
    }
}
```

Level 2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Level2 extends World
{
    private Counter scoreCounter;
    private int startAsteroids2 = 3;
    private int startAlien = 5;
    public int numberOfObjects;

    /**
     * Constructor for level.
     */
    public Level2()
    {
        super(600, 600, 1);
        GreenfootImage background = getBackground();
        background.setColor(Color.BLACK);
        background.fill();

        createStars(706);

        StarFighter2 starfighter2 = new StarFighter2();
        addObject(starfighter2, getWidth()/2 + 100, getHeight()/2);

        addAsteroids2(startAsteroids2);
        addAliens(startAlien);

        scoreCounter = new Counter("Score: ");
        addObject(scoreCounter, 60, 580);

        Explosion.initializeImages();
        ProtonWave.initializeImages();

        prepare();
    }
     /**
     * background stars.
     */
    private void createStars(int number)
    {
        GreenfootImage background = getBackground();
        for(int i=0;i<number;i++)
        {
            int x = Greenfoot.getRandomNumber(getWidth());
            int y = Greenfoot.getRandomNumber(getHeight());
            int color = 120 - Greenfoot.getRandomNumber(100);
            background.setColor(new Color(color, color, color));
            background.fillOval(x,y,2,2);
        }
    }
    /**
     * This method is called in the constructor to spawn asteroids.
     */
    private void addAsteroids2(int count)
    {
        for(int i = 0; i < count; i++)
        {
            int x = Greenfoot.getRandomNumber(getWidth()/2);
            int y = Greenfoot.getRandomNumber(getHeight()/2);
            addObject(new Asteroid2(), x, y);
        }
    }
    private void addAliens(int count)
    {
        for(int i = 0; i < count; i++)
        {
            int x = Greenfoot.getRandomNumber(getWidth()/2);
            int y = Greenfoot.getRandomNumber(getHeight()/2);
            addObject(new Alien(), x, y);
        }
    }
```

```
    /**
     * This method is called when the game is over to display the final score.
     */
    public void gameOver()
    {
        addObject(new ScoreBoard(scoreCounter.getValue()), 300, 300);
    }
    public void gameOverWithMessage()
    {
        addObject(new ScoreBoard("You Win!",scoreCounter.getValue()+42), 300, 300);
    }
    /**
     * Prepare the world for the start of the program. That is: create the initial
     * objects and add them to the world.
     */
    private void prepare()
    {
        Asteroid2 asteroid = new Asteroid2();
        addObject(asteroid, 417, 242);
        Asteroid2 asteroid2 = new Asteroid2();
        addObject(asteroid2, 368, 283);
        Asteroid2 asteroid3 = new Asteroid2();
        addObject(asteroid3, 341, 383);
        Asteroid2 asteroid4 = new Asteroid2();
        addObject(asteroid4, 120, 317);
        Asteroid2 asteroid5 = new Asteroid2();
        addObject(asteroid5, 458, 134);
        Asteroid2 asteroid6 = new Asteroid2();
        addObject(asteroid6, 481, 332);
        Asteroid2 asteroid7 = new Asteroid2();
        addObject(asteroid7, 57, 185);
        asteroid.setLocation(413, 342);
        removeObject(asteroid6);
        removeObject(asteroid);
        removeObject(asteroid3);
        removeObject(asteroid2);
        removeObject(asteroid5);
        removeObject(asteroid4);
        removeObject(asteroid7);

        Alien alien = new Alien();
        addObject(alien, 200, 200);
        Alien alien1 = new Alien();
        addObject(alien, 220, 220);
        Alien alien2 = new Alien();
        addObject(alien, 250, 245);
        Alien alien3 = new Alien();
        addObject(alien, 300, 290);
        Alien alien4 = new Alien();
        addObject(alien, 100, 400);


    }
    public void countScore()
    {

        scoreCounter.add(1);
    }
}
```

Actor Classes

Counter

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.awt.Graphics;

/**
 * Counter that displays a text and number.
 */
public class Counter extends Actor
{
    private static final Color textColor = new Color(255, 180, 150);
    private int value = 0;
    private int target = 0;
    private String text;
    private int stringLength;

    public Counter()
    {
        this("");
    }

    public Counter(String prefix)
    {
        text = prefix;
        stringLength = (text.length() + 2) * 10;

        setImage(new GreenfootImage(stringLength, 16));
        GreenfootImage image = getImage();
        image.setColor(textColor);

        updateImage();
    }

    public void act() {
        if(value < target) {
            value++;
            updateImage();
        }
        else if(value > target) {
            value--;
            updateImage();
        }

        /**
         * If statement that decides when score = 42 it will take player from
         * Level1 to Level2
         */
        World myLevel1 = getWorld();
        if (value == 42)
        {
           Greenfoot.delay(100);

           Greenfoot.setWorld(new Level2());
        }
    }

    public void add(int score)
    {
        target += score;
    }

    public int getValue()
    {
        return value;
    }

    /**
     * Make the image of counter
     */
    private void updateImage()
    {
        GreenfootImage image = getImage();
        image.clear();
        image.drawString(text + value, 1, 12);
    }
}
```

Explosion

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * An explosion. It starts by expanding and then collapsing.
 * The explosion will explode other obejcts that the explosion intersects.
 */
public class Explosion extends Actor
{
    /** How many images should be used in the animation of the explostion */
    private final static int IMAGE_COUNT= 12;

    /**
     * The images in the explosion. This is static so the images are not
     * recreated for every object (improves performance significantly).
     */
    private static GreenfootImage[] images;

    /** Current size of the explosion */
    private int imageNo = 0;

    /** How much do we increment the index in the explosion animation. */
    private int increment=1;

    /**
     * Create a new explosion.
     */
    public Explosion()
    {
        initializeImages();
        setImage(images[0]);
        Greenfoot.playSound("MetalExplosion.wav");
    }

    /**
     * Create the images for explosion.
     */
    public synchronized static void initializeImages()
    {
        if(images == null) {
            GreenfootImage baseImage = new GreenfootImage("explosion-big.png");
            images = new GreenfootImage[IMAGE_COUNT];
            for (int i = 0; i < IMAGE_COUNT; i++)
            {
                int size = (i+1) * ( baseImage.getWidth() / IMAGE_COUNT );
                images[i] = new GreenfootImage(baseImage);
                images[i].scale(size, size);
            }
        }
    }

    /**
     * Explode!
     */
    public void act()
    {
        setImage(images[imageNo]);

        imageNo += increment;
        if(imageNo >= IMAGE_COUNT) {
            increment = -increment;
            imageNo += increment;
        }

        if(imageNo < 0) {
            getWorld().removeObject(this);
        }
    }
}
```

ProtonWave

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.util.List;
/**
 * A proton wave that expands and destroys things in its path.
 */
public class ProtonWave extends Actor
{
    /** The damage this wave will deal */
    private static final int DAMAGE = 30;

    /** How many images should be used in the animation of the wave */
    private static final int NUMBER_IMAGES= 30;

    /**
     * The images of the wave. This is static so the images are not
     * recreated for every object (improves performance significantly).
     */
    private static GreenfootImage[] images;
    int imageCount = 0;
    /**
     * Create a new proton wave.
     */
    public ProtonWave()
    {
        initializeImages();
        Greenfoot.playSound("proton.wav");
    }

    /**
     * Create the images for expanding the wave.
     */
    public static void initializeImages()
    {
        if(images == null)
        {
            GreenfootImage baseImage = new GreenfootImage("wave.png");
            images = new GreenfootImage[NUMBER_IMAGES];
            int i = 0;
            while (i < NUMBER_IMAGES)
            {
                int size = (i+1) * ( baseImage.getWidth() / NUMBER_IMAGES );
                images[i] = new GreenfootImage(baseImage);
                images[i].scale(size, size);
                i++;
            }
        }
    }
    /**
     * Act for the proton wave is: grow and check whether we hit anything.
     */
    public void act()
    {
        checkCollision();
         grow();
    }
    private void grow()
    {

       if (imageCount < NUMBER_IMAGES)
        {
            setImage (images [imageCount]);
            imageCount++;
        }
        else
        {
            imageCount = 0;
            getWorld().removeObject(this);
        }
    }
```

```
    /**
     * Method to detect anything that touches ProtonWave and destroy it.
     */
    private void checkCollision()
    {
        int range = getImage().getWidth()/2;
        List<Asteroid2> asteroids2 = getObjectsInRange(range, Asteroid2.class);
        for(Asteroid2 asteroid2 : asteroids2)
        {
            asteroid2.hit2(DAMAGE);
        }

        List<Alien> aliens = getObjectsInRange(range, Alien.class);
        for(Alien alien : aliens)
        {
            alien.hit3(DAMAGE);
        }
    }
}
```

ScoreBoard

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.util.Calendar;
/**
 * The scoreboard pop-up at the end depending on results and will display
 * appropriate message with score.
 */
public class ScoreBoard extends Actor
{
    public static final float FONT_SIZE = 48.0f;
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;


    public ScoreBoard()
    {
        this(100);
    }

    /**
     * 2 possible outcomes for scoreboard messages.
     */
    public ScoreBoard(int score)
    {
        makeImage("Game Over", "Score: ", score);
    }
    public ScoreBoard(String message,int score)
    {
        makeImage(message, "Score: ", score);
    }
    /**
     * Scoreboard image design
     */
    private void makeImage(String title, String prefix, int score)
    {
        GreenfootImage image = new GreenfootImage(WIDTH, HEIGHT);

        image.setColor(new Color(255,255,255, 128));
        image.fillRect(0, 0, WIDTH, HEIGHT);
        image.setColor(new Color(0, 0, 0, 128));
        image.fillRect(5, 5, WIDTH-10, HEIGHT-10);
        Font font = image.getFont();
        font = font.deriveFont(FONT_SIZE);
        image.setFont(font);
        image.setColor(Color.WHITE);
        image.drawString(title, 60, 100);
        image.drawString(prefix + score, 60, 200);
        setImage(image);
    }
}
```

SmoothMover
(This class is a parent class for all objects that move; all actors from here down)

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

public abstract class SmoothMover extends Actor
{
    private Vector movement;
    private double exactX;
    private double exactY;

    public SmoothMover()
    {
        this(new Vector());
    }

    /**
     * Create new thing initialised with given speed.
     */
    public SmoothMover(Vector movement)
    {
        this.movement = movement;
    }

    /**
     * Move in the current movement direction. Wrap around to the opposite edge of the
     * screen if moving out of the world.
     */
    public void move()
    {
        exactX = exactX + movement.getX();
        exactY = exactY + movement.getY();
        if(exactX >= getWorld().getWidth()) {
            exactX = 0;
        }
        if(exactX < 0) {
            exactX = getWorld().getWidth() - 1;
        }
        if(exactY >= getWorld().getHeight()) {
            exactY = 0;
        }
        if(exactY < 0) {
            exactY = getWorld().getHeight() - 1;
        }
        super.setLocation((int) exactX, (int) exactY);
    }
    /**
     * Set the location from exact coordinates.
     */
    public void setLocation(double x, double y)
    {
        exactX = x;
        exactY = y;
        super.setLocation((int) x, (int) y);
    }

    /**
     * Set the location from int coordinates.
     */
    public void setLocation(int x, int y)
    {
        exactX = x;
        exactY = y;
        super.setLocation(x, y);
    }

    /**
     * Return the exact x-coordinate (as a double).
     */
    public double getExactX()
    {
        return exactX;
    }
```

```
    /**
     * Increase the speed with the given vector.
     */
    public void addForce(Vector force)
    {
        movement.add(force);
    }

    /**
     * Accelerate the speed of this mover by the given factor. (Factors < 1 will
     * decelerate.)
     */
    public void accelerate(double factor)
    {
        movement.scale(factor);
        if (movement.getLength() < 0.15) {
            movement.setNeutral();
        }
    }

    /**
     * Return the speed of this actor.
     */
    public double getSpeed()
    {
        return movement.getLength();
    }

    /**
     * Stop movement of this actor.
     */
    public void stop()
    {
        movement.setNeutral();
    }

    /**
     * Return the current speed.
     */
    public Vector getMovement()
    {
        return movement;
    }
}
```

Alien (space pirate)

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Alien enemy ship that only appears in Level2.
 */
public class Alien extends SmoothMover
{
    private int size;


    /** When the stability reaches 0 the asteroid will explode */
    private int stability;


    public Alien()
    {
        stability = 100;
        GreenfootImage image = getImage();
        image.scale(70,45);
    }

    public Alien(int size)
    {
        super(new Vector(Greenfoot.getRandomNumber(500), 2));
        setSize(size);
    }

    public Alien(int size, Vector speed)
    {
        super(speed);
        setSize(size);
    }

    /**
    * Alien ship will come toward the player's character.
    */
    public void act()
    {
        move(1);
        if (getWorld().getObjects(StarFighter2.class).isEmpty()) return; // skips following if the tank is not in world
        Actor StarFighter2 = (Actor)getWorld().getObjects(StarFighter2.class).get(0); // gets reference to tank
        turnTowards(StarFighter2.getX(), StarFighter2.getY()); // turn toward tank
    }
    /**
     * Set the size of this Alien ship.
     */
    public void setSize(int size)
    {
        stability = size;
        this.size = size;
        GreenfootImage image = getImage();
        image.scale(size, size);
    }

    /**
     * Return the current stability of alien ship. (If it goes down to
     * zero, it breaks up.)
     */
    public int getStability()
    {
        return stability;
    }

    /**
     * Hit this ship dealing the given amount of damage.
     */
    public void hit3(int damage) {
        stability = stability - damage;
        if(stability <= 0)
            breakUp3 ();
    }
```

```
    /**
    * Method that destroyes alien ship when player destroys it.
     */
     private void breakUp3()
    {
        Greenfoot.playSound("Explosion.wav");
        Level2 space = (Level2) getWorld();

        space.countScore();
        if(size <= 0)
        {
            getWorld().removeObject(this);
        }
    }
}
```

Asteroid

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A rock in space
 */
public class Asteroid extends SmoothMover
{
    /** Size of this asteroid */
    private int size;

    /** When the stability reaches 0 the asteroid will explode */
    private int stability;


    public Asteroid()
    {
        this(50);
    }

    public Asteroid(int size)
    {
        super(new Vector(Greenfoot.getRandomNumber(360), 2));
        setSize(size);
    }

    public Asteroid(int size, Vector speed)
    {
        super(speed);
        setSize(size);
    }

    public void act()
    {
        move();
    }

    /**
     * Set the size of this asteroid. Note that stability is directly
     * related to size. Smaller asteroids are less stable.
     */
    public void setSize(int size)
    {
        stability = size;
        this.size = size;
        GreenfootImage image = getImage();
        image.scale(size, size);
    }
    /**
     * Return the current stability of this asteroid. (If it goes down to
     * zero, it breaks up.)
     */
    public int getStability()
    {
        return stability;
    }

    /**
     * Hit this asteroid dealing the given amount of damage.
     */
    public void hit(int damage) {
        stability = stability - damage;
        if(stability <= 0)
            breakUp ();
    }
```

```
    /**
     * Break up this asteroid. If we are still big enough, this will create two
     * smaller asteroids. If we are small already, just disappear.
     */
    private void breakUp()
    {
        Greenfoot.playSound("Explosion.wav");
        Level1 space = (Level1) getWorld();

        space.countScore();
        if(size <= 16)
        {
            getWorld().removeObject(this);
        }
        else
        {
            int r = getMovement().getDirection() + Greenfoot.getRandomNumber(45);
            double l = getMovement().getLength();
            Vector speed1 = new Vector(r + 60, l * 1.2);
            Vector speed2 = new Vector(r - 60, l * 1.2);
            Asteroid a1 = new Asteroid(size/2, speed1);
            Asteroid a2 = new Asteroid(size/2, speed2);
            getWorld().addObject(a1, getX(), getY());
            getWorld().addObject(a2, getX(), getY());
            a1.move();
            a2.move();

            getWorld().removeObject(this);
        }
    }
}
```

Asteroid2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
/**
 * A rock in space
 */
public class Asteroid2 extends SmoothMover
{
    /** Size of this asteroid */
    private int size;

    /** When the stability reaches 0 the asteroid will explode */
    private int stability;


    public Asteroid2()
    {
        this(50);
    }

    public Asteroid2(int size)
    {
        super(new Vector(Greenfoot.getRandomNumber(360), 2));
        setSize(size);
    }

    public Asteroid2(int size, Vector speed)
    {
        super(speed);
        setSize(size);
    }

    public void act()
    {
        move();
    }

    /**
     * Set the size of this asteroid. Note that stability is directly
     * related to size. Smaller asteroids are less stable.
     */
    public void setSize(int size)
    {
        stability = size;
        this.size = size;
        GreenfootImage image = getImage();
        image.scale(size, size);
    }
```

```
    /**
     * Return the current stability of this asteroid. (If it goes down to
     * zero, it breaks up.)
     */
    public int getStability()
    {
        return stability;
    }

    /**
     * Hit this asteroid dealing the given amount of damage.
     */
    public void hit2(int damage) {
        stability = stability - damage;
        if(stability <= 0)
            breakUp2 ();
    }

    /**
     * Break up this asteroid. If we are still big enough, this will create two
     * smaller asteroids. If we are small already, just disappear.
     */
    private void breakUp2()
    {
        Greenfoot.playSound("Explosion.wav");
        Level2 space = (Level2) getWorld();

        space.countScore();
        if(size <= 16)
        {
            getWorld().removeObject(this);
        }
        else
        {
            int r = getMovement().getDirection() + Greenfoot.getRandomNumber(45);
            double l = getMovement().getLength();
            Vector speed1 = new Vector(r + 60, l * 1.2);
            Vector speed2 = new Vector(r - 60, l * 1.2);
            Asteroid2 a1 = new Asteroid2(size/2, speed1);
            Asteroid2 a2 = new Asteroid2(size/2, speed2);
            getWorld().addObject(a1, getX(), getY());
            getWorld().addObject(a2, getX(), getY());
            a1.move();
            a2.move();

            getWorld().removeObject(this);
        }
    }
}
```

Bullet

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A bullet that can hit asteroids.
 */
public class Bullet extends SmoothMover
{
    /** The damage this bullet will deal */
    private static final int damage = 16;

    /** A bullet looses one life each act, and will disappear when life = 0 */
    private int life = 40;

    public Bullet()
    {
    }

    public Bullet(Vector speed, int rotation)
    {
        super(speed);
        setRotation(rotation);
        addForce(new Vector(rotation, 15));
        Greenfoot.playSound("EnergyGun.wav");
    }
```

```
    /**
     * The bullet will damage asteroids if it hits them.
     */
    public void act()
    {
        if(life <= 0) {
            getWorld().removeObject(this);
        }
        else {
            life--;
            move();
            checkAsteroidHit();

        }
    }

    /**
     * Check whether we have hit an asteroid.
     */
    private void checkAsteroidHit()
    {
        Asteroid asteroid = (Asteroid) getOneIntersectingObject(Asteroid.class);
        if (asteroid != null){
            getWorld().removeObject(this);
            asteroid.hit(damage);
        }
    }

}
```

Bullet2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * A bullet that can hit asteroids.
 */
public class Bullet2 extends SmoothMover
{
    /** The damage this bullet will deal */
    private static final int damage = 16;

    /** A bullet looses one life each act, and will disappear when life = 0 */
    private int life = 40;

    public Bullet2()
    {
    }

    public Bullet2(Vector speed, int rotation)
    {
        super(speed);
        setRotation(rotation);
        addForce(new Vector(rotation, 15));
        Greenfoot.playSound("EnergyGun.wav");
    }

    /**
     * The bullet will damage asteroids if it hits them.
     */
    public void act()
    {
        if(life <= 0) {
            getWorld().removeObject(this);
        }
        else {
            life--;
            move();

            checkAsteroidHit2();
        }
    }
```

```
    /**
     * Check whether we have hit an asteroid.
     */
    private void checkAsteroidHit2()
    {
        Asteroid2 asteroid2 = (Asteroid2) getOneIntersectingObject(Asteroid2.class);
        if (asteroid2 != null){
            getWorld().removeObject(this);
            asteroid2.hit2(damage);
        }

        if (getWorld() == null) return;

        Alien alien = (Alien) getOneIntersectingObject(Alien.class);
        if (alien !=null)
        {
            getWorld().removeObject(this);
            alien.hit3(damage);
        }
    }

}
```

StarFighter1 (player's spaceship)

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.util.List;
/**
 * Player's character. Cna move and shoot bullets.
 */
public class StarFighter1 extends SmoothMover
{
    private static final int gunReloadTime = 7;//minimum delay between firing bullets.

    private int reloadDelayCount;   //How long ago we fired the gun the last time.

    private GreenfootImage rocket = new GreenfootImage("rocket.png");
    private GreenfootImage rocketWithThrust = new GreenfootImage("rocketWithThrust.png");

    /**
     * Initilise this rocket.
     */
    public StarFighter1()
    {
        reloadDelayCount = 40;

    }

    /**
     * Calls all the methods for rocket.
     */
    public void act()
    {
        checkKeys();
        move();
        checkForAsteroids();
        checkCollision();
        reloadDelayCount++;
    }

    /**
     * Check whether there are any key pressed and react to them.
     */
    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("space"))
        {
            fire();
        }
        if (Greenfoot.isKeyDown("left"))
        {
            turn(-7);
        }
```

```
        if (Greenfoot.isKeyDown("right"))
        {
            turn(7);
        }
        ignite(Greenfoot.isKeyDown("up"));
        if (Greenfoot.isKeyDown("up"))
        {
            addForce(new Vector(getRotation(), .2));
        }

        if (Greenfoot.isKeyDown("A"))
        {
            turn(-7);
        }
        if (Greenfoot.isKeyDown("D"))
        {
            turn(7);
        }
        ignite(Greenfoot.isKeyDown("W"));
        if (Greenfoot.isKeyDown("W"))
        {
            addForce(new Vector(getRotation(), .2));
        }
    }

    /**
    * Method checks if rocket hits anything and explodes if it does.
    */
    private void checkCollision()
    {
        Asteroid asteroid = (Asteroid) getOneIntersectingObject(Asteroid.class);
        if (asteroid != null){
            World world = getWorld();
            world.addObject(new Explosion(), getX(), getY());
            Level1 space = (Level1) getWorld();
            space.gameOver();
            getWorld().removeObject(this);
        }

    }
    /**
     * Fire a bullet if the gun is ready.
     */
    private void fire()
    {
        if (reloadDelayCount >= gunReloadTime)
        {
            Bullet bullet = new Bullet (getMovement().copy(), getRotation());
            getWorld().addObject (bullet, getX(), getY());
            bullet.move ();
            reloadDelayCount = 0;
        }
    }
    /**
    * Makes rocket thrust turn on.
    */
    private void ignite(boolean boosterOn)
    {
       if(boosterOn)
       {
           setImage(rocketWithThrust);
       }
       else
       {
           setImage(rocket);
       }
    }
    /**
     * Brings up game over message if rocket hits asteroid.
    */
    private void checkForAsteroids()
    {
       Level1 space = (Level1) getWorld();
       if(space.numberOfObjects() < 3)
       {
           Greenfoot.playSound("fanfare.wav");
           space.gameOverWithMessage();
       }
    }
}
```

StarFighter2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
import java.util.List;
/**
 * Player's character. Cna move and shoot bullets.
 */
public class StarFighter2 extends SmoothMover
{
    private static final int gunReloadTime = 7;         // The minimum delay between firing the gun.
    private static final int protonReloadTime = 450;
    private int reloadDelayCount;               // How long ago we fired the gun the last time.
    private int protonReloadDelay;
    private GreenfootImage rocket = new GreenfootImage("rocket.png");
    private GreenfootImage rocketWithThrust = new GreenfootImage("rocketWithThrust.png");

    /**
     * Initilise this rocket.
     */
    public StarFighter2()
    {
        reloadDelayCount = 40;
        protonReloadDelay = 1000;
    }

    /**
     * Do what a rocket's gotta do. (Which is: mostly flying about, and turning,
     * accelerating and shooting when the right keys are pressed.)
     */
    public void act()
    {
        checkKeys();
        move();
        checkForAsteroids2();
        checkCollision();
        reloadDelayCount++;
        protonReloadDelay++;


    }

    /**
     * Check whether there are any key pressed and react to them.
     */
    private void checkKeys()
    {
        if (Greenfoot.isKeyDown("space"))
        {
            fire();
        }
        if (Greenfoot.isKeyDown("left"))
        {
            turn(-7);
        }
        if (Greenfoot.isKeyDown("right"))
        {
            turn(7);
        }
        ignite(Greenfoot.isKeyDown("up"));
        if (Greenfoot.isKeyDown("up"))
        {
            addForce(new Vector(getRotation(), .2));
        }
        if (Greenfoot.isKeyDown ("Shift"))
        {
            startProtonWave();
        }
        if (Greenfoot.isKeyDown("A"))
        {
            turn(-7);
        }
        if (Greenfoot.isKeyDown("D"))
        {
            turn(7);
        }
        ignite(Greenfoot.isKeyDown("W"));
        if (Greenfoot.isKeyDown("W"))
        {
            addForce(new Vector(getRotation(), .2));
        }
    }
```

```
    /**
    * Method that stars up the ProtonWave.
    */
    private void startProtonWave()
    {
        if(protonReloadDelay >= protonReloadTime)
        {
            World world = getWorld();
            world.addObject(new ProtonWave(), getX(), getY());
            protonReloadDelay = 0;
        }
    }


    /**
    * Method checks if rocket hits anything and explodes if it does.
    */
    public void checkCollision()
    {
        Asteroid2 asteroid2 = (Asteroid2) getOneIntersectingObject(Asteroid2.class);
        Alien alien = (Alien) getOneIntersectingObject(Alien.class);
        if (asteroid2 != null){
            World world = getWorld();
            world.addObject(new Explosion(), getX(), getY());
            Level2 space = (Level2) getWorld();
            space.gameOver();
            getWorld().removeObject(this);
        }
        else if (alien !=null) {
            World world = getWorld();
            world.addObject(new Explosion(), getX(), getY());
            Level2 space = (Level2) getWorld();
            space.gameOver();
            getWorld().removeObject(this);
        }
    }

    /**
     * Fire a bullet if the gun is ready.
     */
    private void fire()
    {
        if (reloadDelayCount >= gunReloadTime)
        {
            Bullet2 bullet2 = new Bullet2 (getMovement().copy(), getRotation());
            getWorld().addObject (bullet2, getX(), getY());
            bullet2.move ();
            reloadDelayCount = 0;
        }
    }

    /**
    * Makes rocket thrust turn on.
    */
    private void ignite(boolean boosterOn)
    {
       if(boosterOn)
       {
           setImage(rocketWithThrust);
       }
       else
       {
           setImage(rocket);
       }
    }
    /**
     * Brings up game over message if rocket hits asteroid.
    */
    private void checkForAsteroids2()
    {
       Level2 space = (Level2) getWorld();
       if(space.numberOfObjects() < 3)
       {
           Greenfoot.playSound("fanfare.wav");
           space.gameOverWithMessage();
       }
    }
}
```