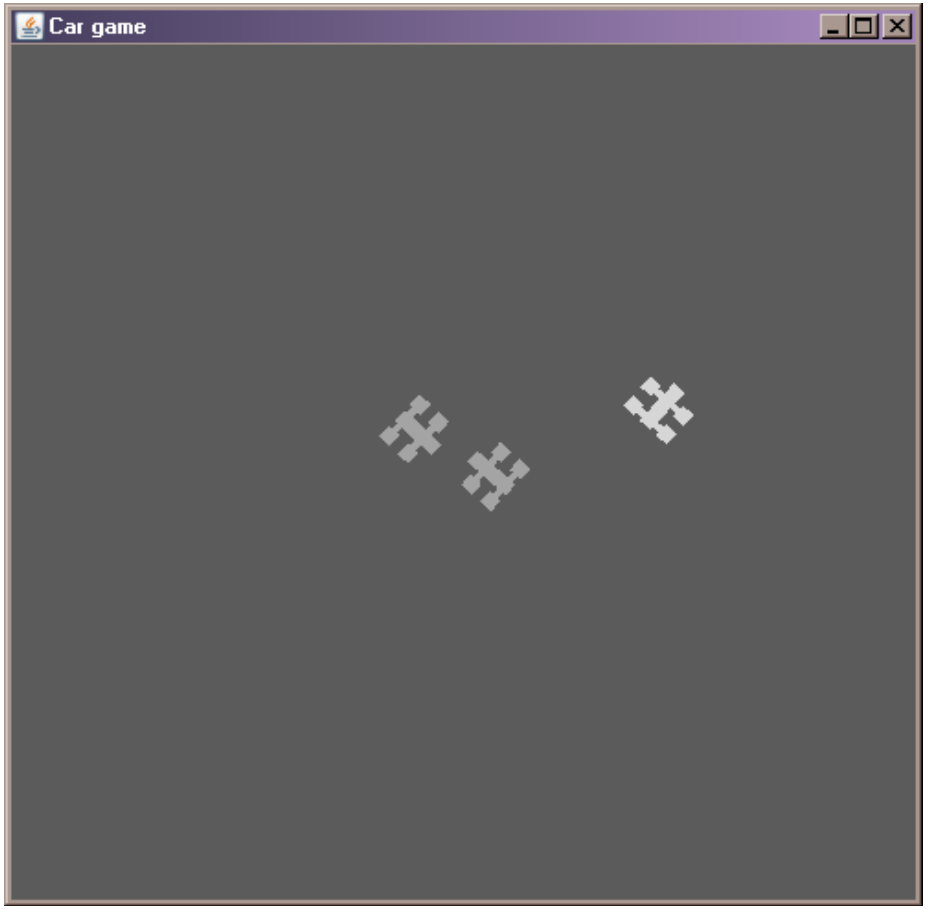
A very simple car game in Java



CarGame.java

```
package cargame;

import javax.swing.*;

public class CarGame {
   public static void main(String[] args) {
     // Create application fram and a game surface

     JFrame frame = new JFrame("Car game");
     frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
     GameSurface s = new GameSurface();

     // Create a few cars and assign behaviours to them

     Car myCar = new Car("car.png");
     myCar.updatePosition(450, 450);
     myCar.setMass(1.0f);
     myCar.setMaxSpeed(100.0f);
     myCar.setMaxSteering(70.0f);
     myCar.addBehaviour(new RoamBehaviour(100, 100, 300, 300));

     Car myCar2 = new Car("car.png");
     myCar2.updatePosition(50, 50);
     myCar2.setMass(1.0f);
     myCar2.setMaxSpeed(120.0f);
     myCar2.setMaxSteering(100.0f);
     myCar2.addBehaviour(new PursuitBehaviour(myCar));
     myCar2.addBehaviour(new BounceOffWallsBehaviour(30, 30, 470, 470));

     Car myCar3 = new Car("playercar.png");
     myCar3.updatePosition(250, 250);
     myCar3.setMass(1.0f);
     myCar3.setMaxSpeed(120.0f);
     myCar3.setMaxSteering(300.0f);
     myCar3.updateVelocity(120.0f, 0.0f);
```

```
    PlayerSteeringBehaviour steering = new PlayerSteeringBehaviour();
    myCar3.addBehaviour(steering);
    myCar3.addBehaviour(new BounceOffWallsBehaviour(30, 30, 470, 470));
    s.addKeyListener(steering);

    // Add the cars to the game surface so that
    // they will be drawn

    s.getVehicles().add(myCar);
    s.getVehicles().add(myCar2);
    s.getVehicles().add(myCar3);

    // Display the game surface in the frame, and
    // make the frame visible

    frame.setContentPane(s);
    frame.setSize(500, 500);
    frame.setVisible(true);

    // Since we want to receive keyboard events,
    // the game surface needs to have the input focus

    s.requestFocusInWindow();

    // Create the animation thread and start it

    AnimationSystem a = new AnimationSystem(s);
    Thread t = new Thread(a);
    t.start();
  }
}
```

Vehicle.java

```
package cargame;

import java.awt.*;
import java.awt.geom.*;
import java.util.*;

public abstract class Vehicle {
  // Member variables

  protected Point2D.Float position = new Point2D.Float();
  protected Point2D.Float orientation = new Point2D.Float();
  protected Point2D.Float side = new Point2D.Float();
  protected Point2D.Float velocity = new Point2D.Float();
  protected Point2D.Float steering = new Point2D.Float();
  protected float mass;
  protected float maxSpeed;
  protected float maxSteering;

  // List of behaviours

  protected ArrayList behaviours = new ArrayList(10);

  // Getters and setters

  public Point2D.Float getPosition() { return position; }
  public void updatePosition(Point2D.Float p) { position.x = p.x; position.y = p.y; }
  public void updatePosition(float x, float y) { position.x = x; position.y = y; }

  public Point2D.Float getOrientation() { return orientation; }
  public void updateOrientation(Point2D.Float o) { orientation.x = o.x; orientation.y = o.y; }
  public void updateOrientation(float x, float y) { orientation.x = x; orientation.y = y; }

  public Point2D.Float getSideVector() { return side; }
```

```
public Point2D.Float getVelocity() { return velocity; }
public void updateVelocity(Point2D.Float v) { velocity.x = v.x; velocity.y = v.y; }
public void updateVelocity(float x, float y) { velocity.x = x; velocity.y = y; }

public Point2D.Float getSteering() { return steering; }
public void updateSteering(Point2D.Float s) { steering.x = s.x; steering.y = s.y; }
public void updateSteering(float x, float y) { steering.x = x; steering.y = y; }

public float getMass() { return mass; }
public void setMass(float m) { mass = m; }

public void setMaxSpeed(float m) { maxSpeed = m; }
public float getMaxSpeed() { return maxSpeed; }

public void setMaxSteering(float f) { maxSteering = f; }
public float getMaxSteering() { return maxSteering; }

public void addBehaviour(Behaviour b) {
  behaviours.add(b);
}


// A few utility methods for working with vectors

static float length(Point2D.Float v) {
  return (float)Math.sqrt((v.x * v.x) + (v.y * v.y));
}

static public void scale(Point2D.Float v, float newLength) {
  float l = length(v);
  v.x *= newLength / l;
  v.y *= newLength / l;
}
```

```
// Update this vehicle

public void update(float dt) {
  for (int i = 0; i < behaviours.size(); i++) {
    ((Behaviour)behaviours.get(i)).update(this, dt);
  }

  // Truncate the length of the desired steering force vector

  Point2D.Float force = new Point2D.Float(steering.x, steering.y);
  float l = length(force);
  if (l > maxSteering) {
    force.x *= maxSteering / l;
    force.y *= maxSteering / l;
  }

  // Newton's second law: steering force = mass * acceleratаion

  Point2D.Float acc = new Point2D.Float(force.x / mass, force.y / mass);

  // Update velocity vector using Euler's method
  // and truncate its length to the maximum allowed

  velocity.x += dt * acc.x;
  velocity.y += dt * acc.y;
  l = length(velocity);
  if (l > maxSpeed) {
    velocity.x *= maxSpeed / l;
    velocity.y *= maxSpeed / l;
  }

  // Update position using Euler's method

  position.x += dt * velocity.x;
  position.y += dt * velocity.y;
```

```
    // Set orientation to equal the velocity vector
    // and set the side vector accordingly

    l = length(velocity);
    if (l > 0.0f) {
      orientation.x = velocity.x / l;
      orientation.y = velocity.y / l;
      side.x = -orientation.y;
      side.y = orientation.x;
    }
  }

  // Abstract methods for drawing and intersection testing

  public abstract void draw(Graphics2D g2);

  public abstract boolean intersects(Vehicle v);
}
```

Car.java

```
package cargame;

import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.swing.*;

public class Car extends Vehicle implements ImageObserver {
  protected Image img;
  protected float w2;
  protected float h2;

  public Car(String imageFileName) {
    ImageIcon iic = new ImageIcon(imageFileName);
    img = Transparency.makeColorTransparent(iic.getImage(), Color.black);
  }

  public void draw(Graphics2D g2) {
    AffineTransform saveXform = g2.getTransform();

    g2.translate(position.x, position.y);
    g2.rotate(Math.atan2(orientation.y, orientation.x));
    g2.drawImage(img,
                 AffineTransform.getTranslateInstance(-img.getWidth(this) / 2.0, -img.getHeight(this) / 2.0),
                 this);

    g2.setTransform(saveXform);

    /*
    g2.setPaint(Color.yellow);
    g2.drawLine((int)Math.floor(position.x), (int)Math.floor(position.y),
         (int)Math.floor(position.x + 50.0f * side.x), (int)Math.floor(position.y + 50.0f * side.y));
    g2.setPaint(Color.blue);
```

```
    g2.drawLine((int)Math.floor(position.x), (int)Math.floor(position.y),
        (int)Math.floor(position.x + velocity.x), (int)Math.floor(position.y + velocity.y));
    g2.setPaint(Color.white);
    g2.drawLine((int)Math.floor(position.x), (int)Math.floor(position.y),
        (int)Math.floor(position.x + steering.x), (int)Math.floor(position.y + steering.y));
    */
  }

  public boolean imageUpdate(Image img, int infoflags, int x, int y, int width, int height) {
    return true;
  }

  public boolean intersects(Vehicle v) {
    if (v instanceof Car) {
      Car c = (Car)v;
      Point2D.Float d = new Point2D.Float(position.x - c.position.x, position.y - c.position.y);
      if (length(d) < 25.0f) {     // Should probably compute the radius from the images instead...
        return true;
      }
    }
    return false;
  }
}
```

GameSurface.java

```
package cargame;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class GameSurface extends JComponent implements MouseListener {
  protected ArrayList vehicles;

  public GameSurface() {
    vehicles = new ArrayList(10);
    addMouseListener(this);        // For requesting input focus
  }

  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    Dimension d = getSize();
    g2.setPaint(new Color(91, 91, 91));
    g2.fillRect(0, 0, d.width, d.height);

    for (int i = 0; i < vehicles.size(); i++) {
      Vehicle v = (Vehicle)vehicles.get(i);
      v.draw(g2);
    }
  }

  ArrayList getVehicles() {
    return vehicles;
  }
```

```
  public void mouseClicked(MouseEvent e) {
    // Custom components need to request the
    // input focus when they are clicked,
    // otherwise they won't receive keyboard events
    requestFocusInWindow();
  }
  public void mouseMoved(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseDragged(MouseEvent e) {}
}
```

AnimationSystem.java

```
package cargame;

import java.util.*;

public class AnimationSystem implements Runnable {
  protected GameSurface game;

  public AnimationSystem(GameSurface gameSurface) {
    game = gameSurface;
  }

  public void run() {
    long time = System.currentTimeMillis();
    for (;;) {
      ArrayList vehicles = game.getVehicles();

      // Update position, velocity etc. of vehicles

      long t = System.currentTimeMillis();
      long dt = t - time;
      float secs = (float)dt / 1000.0f;  // Convert to seconds
      for (int i = 0; i < vehicles.size(); i++) {
        Vehicle v = (Vehicle)vehicles.get(i);
        v.update(secs);
      }

      // Check for collisions

      for (int i = 0; i < vehicles.size(); i++) {
        for (int j = i + 1; j < vehicles.size(); j++) {
          Vehicle vi = (Vehicle)vehicles.get(i);
          Vehicle vj = (Vehicle)vehicles.get(j);
          if (vi.intersects(vj)) {
```

```
                // Collision detected!
                // For now, simply reset the positions of the vehicles
                vi.updatePosition(50, 50);
                vj.updatePosition(450, 450);
              }
            }
          }

          time = System.currentTimeMillis();
          game.repaint();

          // Sleep for a short amount of time to allow the system to catch up.
          // This improves framerate substantially and avoids hiccups

          try {
            Thread.sleep(20);
          } catch (InterruptedException e) {
          }
        }
      }
    }
```

Behaviour.java

```
package cargame;

public abstract interface Behaviour {
   public abstract void update(Vehicle v, float dt);
}
```

PursuitBehaviour.java

```
package cargame;

import java.awt.geom.Point2D;

public class PursuitBehaviour implements Behaviour {
  protected Vehicle target;

  PursuitBehaviour(Vehicle v) {
    target = v;
  }

  public void update(Vehicle v, float dt) {
    // Steer vehicle towards target

    Point2D.Float p = v.getPosition();
    Point2D.Float tp = target.getPosition();
    Point2D.Float desired_velocity = new Point2D.Float(tp.x - p.x , tp.y - p.y);
    Vehicle.scale(desired_velocity, v.getMaxSpeed());
    v.updateSteering(desired_velocity.x, desired_velocity.y);
  }

}
```

RoamBehaviour.java

```
package cargame;

import java.awt.geom.Point2D;

public class RoamBehaviour implements Behaviour {
  protected long lastTargetUpdate;
  protected Point2D.Float target = new Point2D.Float();
  protected float width;
  protected float height;
  protected float x;
  protected float y;

  public RoamBehaviour(float x, float y, float width, float height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
  }

  public void update(Vehicle v, float dt) {
    // Update target if necessary

    long time = System.currentTimeMillis();
    if (time - lastTargetUpdate > 5000) {
      target.x = x + (float)Math.random() * width;
      target.y = y + (float)Math.random() * height;
      lastTargetUpdate = time;
    }

    // Steer vehicle towards target

    Point2D.Float p = v.getPosition();
    Point2D.Float desired_velocity = new Point2D.Float(target.x - p.x , target.y - p.y);
```

```
      Vehicle.scale(desired_velocity, v.getMaxSpeed());
      v.updateSteering(desired_velocity.x, desired_velocity.y);
    }
  }
```

BounceOffWallsBehaviour.java

```
package cargame;

import java.awt.geom.Point2D;

public class BounceOffWallsBehaviour implements Behaviour {
  protected float x1;
  protected float y1;
  protected float x2;
  protected float y2;

  public BounceOffWallsBehaviour(float x1, float y1, float x2, float y2) {
    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2;
    this.y2 = y2;
  }

  public void update(Vehicle v, float dt) {
    Point2D.Float p = v.getPosition();
    Point2D.Float vel = v.getVelocity();
    if (p.x < x1) {
      p.x = x1;
      vel.x = -vel.x;
    }
    if (p.x > x2) {
      p.x = x2;
      vel.x = -vel.x;
    }
    if (p.y < y1) {
      p.y = y1;
      vel.y = -vel.y;
    }
    if (p.y > y2) {
```

```
        p.y = y2;
        vel.y = -vel.y;
      }
    }
  }
```

PlayerSteeringBehaviour.java

```
package cargame;

import java.awt.event.*;
import java.awt.geom.Point2D;

public class PlayerSteeringBehaviour implements Behaviour, KeyListener {

  protected boolean steering = false;
  protected float direction = 1.0f;

  public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == 37) {    // Cursor left
      steering = true;
      direction = -1.0f;
    }
    if (e.getKeyCode() == 39) {    // Cursor right
      steering = true;
      direction = 1.0f;
    }
  }

  public void keyReleased(KeyEvent e) {
    steering = false;
  }

  public void keyTyped(KeyEvent e) {
  }

  public void update(Vehicle v, float dt) {
    if (steering) {
      Point2D.Float side = v.getSideVector();
      Point2D.Float desired_velocity = new Point2D.Float(side.x * direction, side.y * direction);
      Vehicle.scale(desired_velocity, v.getMaxSteering());
      v.updateSteering(desired_velocity.x, desired_velocity.y);
```

```
      } else {
         v.updateSteering(v.getVelocity());
      }
   }
}
```

Transparency.java

```
package cargame;

import java.awt.*;
import java.awt.image.*;

public class Transparency {
  public static Image makeColorTransparent(Image im, final Color color) {
    ImageFilter filter = new RGBImageFilter() {
      public int markerRGB = color.getRGB() | 0xFF000000;
      public final int filterRGB(int x, int y, int rgb) {
        if ( ( rgb | 0xFF000000 ) == markerRGB ) {
          return 0x00FFFFFF & rgb;
        } else {
          return rgb;
        }
      }
    };

    ImageProducer ip = new FilteredImageSource(im.getSource(), filter);
    return Toolkit.getDefaultToolkit().createImage(ip);
  }
}
```