Juhitav animatsioon

Joonistuskäsud, lõimed, mälupilt, heli

Taustateave

Joonistamine

Javas on püütud joonistusvahendid teha sihtpinnast sõltumatuks, nii et sama programmilõigu abil võib joonistada nii ekraanile, mällu kui printerisse ning sobiva draiveri abil ka muude vahendite abil. Programmi poolt saadetakse joonistatavale pinnale teateid graafilise konteksti abil, milleks on enamasti klassi Graphics või Graphics2D oskustega isend. Kontekstile antakse üldisi käsklusi joone, ringi, teksti jm. joonistamise kohta, viimase ülesandeks on tulemus kanda üle sihtpinnale. Kas tegemist on mälupildi joonistamisel massiivi väärtuste arvutamisega või põletuspulga liigutamisel plotteris, sellele ei pea programm tähelepanu pöörama. Programmi kirjutaja isegi ei pea teadma, millise sihtseadmeni tema käskude mõju lõpuks ulatub. Et iga lõigu või punkti joonistamisel ei peaks eraldi kaasa lisama joonistamisel vajalikke parameetreid, hoitakse neid graafilise konteksti juures ning koos joonistuskäsklustega edastatakse sihtseadmele. Graphics hoiab eneses joonistusvärvi, -piirkonda ning nihet, Graphics2D juures aga tulevad juurde joone laius, pildi keere, kalle, suurendus, joonte ühendus. Samuti võib joonistusvärviks valida mustri või värviülemineku. Sellisel juhul arvutades mälus asuvale pildile joonistatavat ringi iga punkti puhul vaadatakse, milline koht mustrist vastab joone alla jäävale kohale sihtpinnal.

Välja paistva graafikakomponendi joonistuskäsud koondatakse üldiselt meetodisse nimega paint. Meetodile antakse parameetrina graafiline kontekst, mis oma väljundi suunab pinnale, kus kasutaja arvab komponenti asuvat, üldjuhul ekraanile. Nii õnnestub lihtsustada komponenti näitava kesta tööd. Igal korral kui leitakse, et komponendi sisu on kaduma läinud või vajab lihtsalt uuendamist, kutsutakse taas välja paint, mis komponendi arvates sobiva oleku taastab. Kui komponendi omanik vaatab, et uuendamist vajab vaid osa komponendi pinnast, võidakse ressursside kokkuhoiu huvides saata paint-meetodisse pügatud (vähendatud, clip) joonistusalaga Graphics, mis kannab sihtpinnale edasi vaid nende joonistuskäskude tulemused, mis jäävad konteineri (komponendi omaniku) arvates uuendamist vajavasse piirkonda. Nõnda võib mõnikord veebilehe kerimisel avastada, et konteiner on uuendamist vajavat osa valesti hinnanud ning valgeks jäetakse ka osa komponendi pinnast, kus peaks midagi muud leiduma. Soovides kogu komponendi pinda uuendada, tuleb anda käsklus repaint(). See käsk käivitab pinnauuenduse eraldi lõimes. Ta ootab kõigepealt, et virtuaalmasinal jaguks piisavalt ressursse joonistamiseks. Edasi kutsutakse välja komponendi meetod update parameetrina graafiline kontekst komponendi asukohaseadme pinnale joonistamiseks, mis klassist java.awt.Component päritud vaikimisi oskuste alusel kõigepealt katab komponendi pinna taustavärviga ning seejärel kutsub välja meetodi paint, kus kirjeldatud käskude abil peaks kompomendi arvatav pinnavälimus kasutajani jõudma. Nõndamoodi saab hoolitseda, et vanast välimusest uude juhuslike jupikeste näol rudimente ei jääks, sest kui vana pind ühtlaselt taustavärvi plaadiga katta, siis ei tohi ju põhja midagi alla silma häirima jääda. Samas võib suurema pinna uuendamisel tekkida ekraanile vilgatus, kui platsi tühjendamise ning uue sisu joonistamise vahele piisavalt suur vahe jääb, et inimsilm seda märkab. Lahenduseks leitakse, et tuleb komponendi kohal näidatav pilt eelnevalt valmis arvutada ning siis ühekorraga või rida-realt ekraanile joonistada. Nõnda saab teha, kui pilt koostatakse mällu ning paint-meetodi sisuks oleks vaid selle pildi ekraanile manamine, update aga loobuks taustavärviga katmisest (taustavärv oleks ekraanile joonistatava pildi põhjaks) ning kutsuks kohe välja paint-i. update't annab muuta ülekatte abil: kui kirjutatada

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

siis jääbki taustaga katmine ära ning kogu töö usaldatakse paint'ile. Sellist joonistusviisi nimetatakse topeltpuhverduseks ning Swingi juures võib sama skeemi joonistamise juures ühe lisaparameetriga sisse lülitada.

Ekraanile joonistada saab ka mujal kui paint-meetodis, kuid sellisel juhul konteineri poolt

kutsutud pinnauuenduse korral läheb kõik muu kaduma, mis paint'is kirjas pole. Sellegipoolest on mõnikord mugav kohe hiirevajutuse peale küsida pinna graafiline kontekst ning sealtkaudu soovitud ringike ekraanile manada, mitte asuda andmeid muutujates säilitama ning paint'i kaudu ekraanile tooma. Tõsisemate programmide juures tuleks aga valida viimane tee ning pea alati on siiski vaja ka omal teada, mida ja kus peaks ekraanile ilmuma. Ka liikumise puhul on mõistlik jätta pinna uuendamine paint-meetodi hooleks, kuigi vahel on mugavam otse leitud uute andmete järgi ka pilt ekraanile joonistada.

Taust liikumise ajal mälus.

Järgnevas näites koos kasutaja hiire lohistamisega liigub ekraanil ring. Liikumise taustaks on eelnevalt hiire lahtilaskmise kohas taustapildile joonistatud ringid ning iga uue lahtilaskmisega tekib taustapildile üks ring juurde. Joonistusmeetod paint viib ekraanile vaid taustapildi, liikumismulje jätab mouseDragged, kus alla joonistatakse taust ning peale hiire parajatist asukohta tähistav ring. Et taustapilt oleks kõikjalt Liigu2 isendist kätte saadav, selleks on deklareeritud

```
Image pilt;
```

klassi algusse. Pilt saab enesele väärtuse ning võimaluse andmeid sisestada aga alles joonistusmeetodis

```
public void paint(Graphics g){
    if(pilt==null)looPilt();
    g.drawImage(pilt, 0, 0, this);
  }
```

, sest varem pole kindlustatud, et createImage suudaks Liigu2 komponendi omadustele vastava pildi luua. Koos pildi loomisega küsitakse ka selle graafiline kontekst, et edaspidi oleks hea lihtne mälupildi peale andmeid saata.

```
void looPilt(){
    pilt=createImage(getSize().width, getSize().height);
    piltg=pilt.getGraphics();
  }
```

Java uuemates versioonides on võimalik pilt ka varem luua BufferedImage abil. Hiire sündmustele reageerimiseks on rakendile külge pandud kuular nii hiirevajutuste (MouseListener) kui hiire liikumise (MouseMotionListener) tarvis.

```
  public Liigu2(){
    addMouseMotionListener(this);
    addMouseListener(new Liigu2kuular(this));
  }
```

Liikumise ja vedamise tarvis kaks meetodit on realiseeritud Liigu2 enese sees, vajutuste püüdmiseks on aga loodud eraldi adapterklass, kust hiirenupu lahtilaskmise peale mälupildile ring joonistatakse.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Liigu2 extends Applet
           implements MouseMotionListener{
  Image pilt;
  Graphics piltg;
  public Liigu2(){
    addMouseMotionListener(this);
    addMouseListener(new Liigu2kuular(this));
  }
  public void paint(Graphics g){
    if(pilt==null)looPilt();
    g.drawImage(pilt, 0, 0, this);
  }
  void looPilt(){
    pilt=createImage(getSize().width, getSize().height);
```

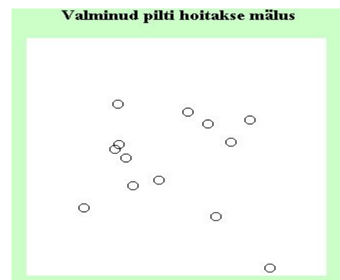
```
    piltg=pilt.getGraphics();
  }
  public void mouseMoved(MouseEvent e){}
  public void mouseDragged(MouseEvent e){
    Graphics g=this.getGraphics();
    g.drawImage(pilt, 0, 0, this);
    g.drawOval(e.getX()-5, e.getY()-5, 10, 10);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame();
    f.add(new Liigu2());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

Et hiire vajutamise teateid püüdev suudaks Liigu2 juurde kuuluvale pildile midagi joonistada, selleks peab siin olema ligipääs ekraanile joonistatava pildi graafilisele kontekstile. Üheks võimaluseks on lisada kuularile osuti, mille kaudu sihtisendile ligi pääseda. Selle muutuja nimeks on siin peremees. Objektorienteeritud programmeerimises on viisakas isendimuutujatele anda väärtusi meetodite kaudu, siin saadakse andmed kohale konstruktori kaudu. Kui üleval kirjutati kuulari lisamiseks

```
addMouseListener(new Liigu2kuular(this));
```

, siis see this Liigu2kuulari loomisel tähendabki osutit Liigu2 eksemplarile.

```
class Liigu2kuular extends MouseAdapter{
  Liigu2 peremees;
  public Liigu2kuular(Liigu2 l2){
    peremees=l2;
  }
  public void mouseReleased(MouseEvent e){
    peremees.piltg.drawOval(
                e.getX()-5, e.getY()-5, 10, 10);
  }
}
```



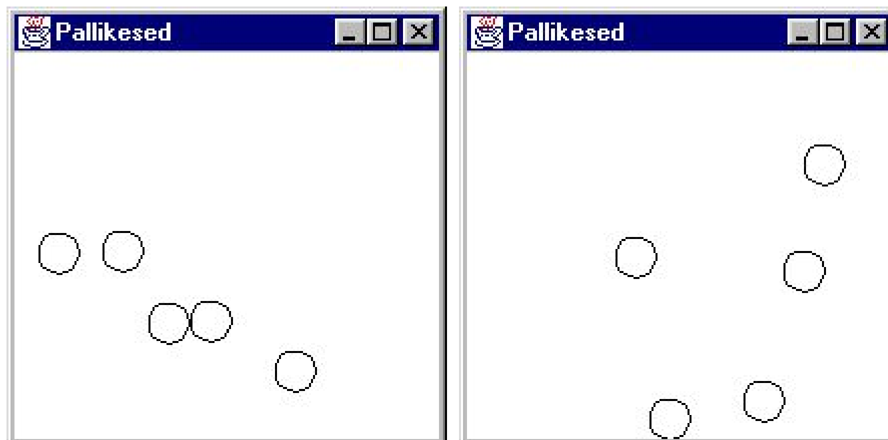
Lõim liikumisel

Järgnevas näites on lõime alamklass Pall2. Igal pallil on koordinaadid x ja y ning iga sammuga muutuvad need dx ja dy võrra. Iga pall liigub omaette lõimes, s.t. programmi muudest osadest sõltumatult. Pall alandab enese prioriteeti, s.t. et protsessori tööaja jagamisel loeb ta end keskmisest vähem tähtsamaks. Prioriteedi alandamine on vajalik selleks, et pallide liigutamine ei hakkaks märkimisväärselt aeglustama teisi lõimi, näiteks ekraanile joonistamist. Muidu võib juhtuda, et suure hulga pallide puhul jäävad ülejäänud tegevused suhteliselt unarusse. Käsk yield tähendab, et lõim annab oma tööjärje üle järgmisele ning asub ise järjekorda uuesti protsessori aega ootama.

Raami alamklass Loim2 loob enesele viis palli isendit ning laseb nad liikuma. Loim2 ise realiseerib liidest Runnable, s.t., et tema run-meetodit on samuti võimalik panna tööle eraldi lõimes. Selle lõime ülesandeks on joonistada iga natukese aja tagant uus ekraanipilt vastavalt pallide uutele koordinaatidele.

```
import java.awt.*;
public class Loim2 extends Frame implements Runnable{
 Pall2[] pallid;
 public Loim2(){
```

```
    pallid=new Pall2[5];
    for(int i=0; i<pallid.length; i++)
      pallid[i]=new Pall2();
    setSize(200, 200);
    setVisible(true);
    new Thread(this).start();
  }
  public void joonista(){
    Image pilt=createImage(200, 200);
    Graphics piltg=pilt.getGraphics();
    for(int i=0; i<pallid.length; i++){
      piltg.drawOval(pallid[i].x()-10, pallid[i].y()-10, 20, 20);
    }
    this.getGraphics().drawImage(pilt, 0, 0, this);
  }
  public void run(){
    while(true){
      joonista();
      Thread.yield();
      try{Thread.sleep(100);}catch(Exception e){}
    }
  }
  public static void main(String argumendid[]){
    new Loim2();
  }
}

class Pall2 extends Thread{
  double x=200, y=200, dx, dy;
  int vasak=20, ulal=50, parem=200, all=200;
  public Pall2(){
    dx=5*Math.random();
    dy=5*Math.random();
    start();
  }
  public int x(){
    return (int)x;
  }
  public int y(){
    return (int)y;
  }
  public void run(){
    setPriority(Thread.NORM_PRIORITY-2);
    while(true){
      if(x>parem)dx=-Math.abs(dx);
      if(y>all)dy=-Math.abs(dy);
      if(x<vasak)dx=Math.abs(dx);
      if(y<ulal)dy=Math.abs(dy);
      x+=dx;
      y+=dy;
      yield();
      try{Thread.sleep(100);}catch(Exception e){}
    }
  }
}
```



Sirelasemäng

Järgnevalt vaadeldalse näidete abil teekond, mida läheb tarvis enamiku liikumisega seotud rakenduste puhul. Alustatakse tausta liigutamisest ning ükshaaval lisatakse elemente. Kas tulemuseks on mäng või juhitav simulaator sõltub rohkem vaatenurgast.

Liikuv taust

Küllalt mugav on tausta tekitada korduva pildiga, kuid see pole sugugi ainuke võimalus. Mustri võib ka pidevalt mõne kavala valemi abil välja arvutada. Mugavaks ümberkäimiseks peaks taustapilt olema nähtavast alast suurem, kuigi mälu kokkuhoiu huvides võib vajadusel proovida ka sama väiksemat pilti mitu korda üksteise kõrvale joonistada. Samuti on nõnda võimalik panna rakendus tausta joonistamisel ka nähtava ala suurust arvestama. Siin näites on taustapildi suuruseks 320x480 punkti, eeldatav vaatamissuurus aga 300x300 punkti. Tausta muster kordub iga 160 punkti tagant. Taustapilti joonistatakse iga kaadri haaval niikaua allapoole, kuni on läbitud kogu mustripikkus (siin 160 punkti). Siis alustatakse järgmisel korral jälle mustri joonistamist ülevalt. Nii tekibki pilt liikumisest, sest hüpe mustri kõrguse jagu üles ei ole kasutajale nähtav.

Muudetavad parameetrid on koondatud programmi algusse, et neid saaks ka siis oma vajaduste järgi paika sättida, kui programmi sisust suuremat ülevaadet pole. Lähem seletus: Taustapildimuutuja on toodud meetoditest välja, et selle abil oleks võimalik pidevalt pildi poole pöörduda ning ei peaks iga joonistamisvajaduse eel pilti asuma failist uuesti välja lugema. Mustripikkus on eraldi muutujas, sest taustapildi vahetamisel teistsuguse vastu on tarvis teada, kui pika hüppe peab üles tegema, et pilt kasutaja silmadele taas samasugune välja näeks.
Samm hoiab meeles, mitme ekraanipunkti jagu iga korraga pilti allapoole nihutatakse, paus teatab joonistamise vahel oodatavate millisekundite arvu. Sammu suurendamisel või ooteaja lühenemisel liikumiskiirus kasvab. Liialt suur samm muudab aga liikumise hüplikuks, liialt väike ooteaeg aga ei pruugi lasta masinal tulemust korralikult välja joonistada.

```
Image taust;
 int mustripikkus=160;
 int samm=3;
 int paus=50;
 int nihe=0;
 boolean veel=false;
```

Plinkimise vältimiseks on üle kaetud meetod update, paludes sel kohe paint välja kutsuda. Muul juhul tahetaks igal korral ekraan enne pildi joonistamist taustavärviga katta ning selle tulemusena hakkaks silmeesine virvendama. Kuna aga taustapilt nagunii kogu nähtava ala katab, siis ei jää vana pilt nagunii näha ning taustavärviga katmine oleks mõttetu.

Pildi laadimiseks on loodud eraldi alamprogramm, kuna rakend ja rakendus saavad pildi kätte erinevalt. See meetod proovib kõigepealt pilti saada rakendi moel. Kui nii ei õnnestu, siis üritatakse Toolkiti abil failist lugeda. Kui fail leidub ja formaadid sobivad, siis üks võimalus nendest võiks õnnestuda.

Pilt laetakse sisse, kui käivitusjärg jõuab esimest korda paint-alamprogammini. Loogiline võiks tunduda pildi laadimine otse muutujate deklareerimise ajal, aga rakendi getImage pole selleks ajaks veel töövõimeline. Nii ongi tehtud paint'i algusesse valik, kus muutuja taust null-väärtuse korral (mis tal algselt on) palutakse pilt sisse laadida.

Iga paint-meetodi väljakutse korral liigutatakse taustapilti sammu jagu allapoole. Nihe näitab, palju ollakse algsest asendist edasi liikunud. Kui pärast arvutust ületab nihe mustripikkust, siis hüpatakse mustri pikkuse jagu ülespoole, et oleks taas võimalik rahumeeles alla liikuda.

Joonistamiskäskluses on x-koordinaadiks 0, sest tausta joonistatakse alates vasakust servast. Y-koordinaadi väärtus nihe-mustripikkus näitab, et igal korral määrab pildi asukoha nihke suurenemine. Samas konstantne mustripikkuse muutuja väärtus hoiab pilti pidevalt niipalju kõrgel, et pildi ülaserv ei hakkaks kasutajale paistma.

Kui kood vaid sellega piirduks, siis püsiks pilt enamiku ajast paigal. Ning vaid juhul, kui mõnd teist akent me rakenduse peale ja sealt ära liigutada või natukese aja tagant meie akna suurust muuta võiksime märgata mingit liikumist. Et aga saaksime silme ees näha pidevalt liikuvat pilti,

peaksid üksikud hüpped toimuma piisavalt sageli.

Sarnase ülejoonistuse nagu toimub akna suuruse muutuse juures, saab esile kutsuda ka käsu repaint() käivitamisega. Püsivat pilti joonistavates programmides paigutati see enamasti tekstivälja või hiiresündmuse töötlusossa. Siin aga soovime, et meetodit kutsutaks välja pidevalt iga natukese aja tagant. Leidub ka klass, mille abil peaks võimalik olema täpselt soovitud ajavahemike tagant etteantud meetodit käivitada, kuid lihtsamal juhul piisab lahendusest, kus pärast iga joonistust peetakse soovitud aja pikkune paus. Selline vaikselt põksuv süda on ka siia rakendusse sisse ehitatud ning võlusõnaks on lõim.

Kuigi traditsioonilise programmeerimise juures on programmi tööjärg kogu aeg kindlas kohas ning ilma eelmist sammu lõpetamata edasi ei liiguta, siis pole see sugugi ainus tava rakendusi töös hoida. Näiteks liigutamise, ehk praegusel juhul repaint'i väljakutsumise saab jätta suhteliselt iseseisvalt töötava lõime hooleks. Selle käivitamise, tööshoidmise ja seiskamise tarvis on näha järgnevas koodis hulk sõnu.

Lõime abil omaette käivitatav kood paigutatakse meetodi run sisse. Selline nimi on määratud liideses Runnable ning selle järgi teab Thread-tüüpi objekt, mida käivitada. Meetodi run ülesandeks selles programmis on repaint-käsklust korduvalt välja kutsuda. Selleks on loodud tsükkel milles käsklus repaint() ning ootamiseks Thread.sleep(paus). Viimane on paigutatud try-catch katsendiplokki, kuna võib mõnes olukorras (lõimede omavahelise suhtlemise korral) heita erindeid. Siin koodis taolist võimalust pole, kuid sellest hoolimata nõuab Java kompilaator potentsiaalselt erindiohtliku käsu ümbritsemist vastava plokiga. Muutuja veel while-tsükli päises võimaldab lõimel oma töö lõpetada, kui too tõeväärtusmuutuja väärtuseks antakse false. See juhtub näiteks käsus stop, mis kutsutakse välja seilur lahkub rakendit sisaldavalt lehelt.

Tööle lükatakse lõim meetodist start. Näites võib märgata kaht käsklust start, mis aga omavahel kuigivõrd seotud pole. Esimene kuulub klassi Pildike1 külge ning katab üle klassi Applet vastavat meetodit. Rakendikäitur (seilur) teatab selle kaudu Pildike1 eksemplarile, et isend on lehel avanenud. Siis on paras aeg liikumislõime käivitamiseks. Käsk new Thread(this).start() teatab, et loodagu uus klassi Thread eksemplar, millele antakse ette this ehk klassi Pildike1 eksemplar. Ning et etteantud eksemplaris pandagu iseseisvana käima run-meetod. Et run just käivitada tuleb, seda teab juba Thread. Iseseisva programmina käsurealt käivitades on näha main-meetodis käsku ap.start () – see käivitab Pildike1 start-käsu samuti nagu rakendikäiturgi veebis. Ning tulemusena õnnestub liikuvat tausta vaadata.

```
import java.applet.Applet;
import java.awt.*;
public class Pildike1 extends Applet implements Runnable{
  Image taust;
  int mustripikkus=160;
  int samm=3;
  int paus=50;
  int nihe=0;
  boolean veel=false;

  public void paint(Graphics g){
    if(taust==null)taust=laePilt("rohetaust320x480.gif");
    nihe=nihe+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    g.drawImage(taust, 0, nihe-mustripikkus, this);
  }
  public void update(Graphics g){
    paint(g);
  }


  public void start(){
    veel=true;
    new Thread(this).start();
  }
  public void run(){
    while(veel){
        repaint();
      try{Thread.sleep(paus); }catch(Exception e){}
    }
  }
  public void stop(){
    veel=false;
  }
```

```
  Image laePilt(String failinimi){
    try{
     return getImage(getCodeBase(), failinimi);
    }catch(Exception e){}
     return Toolkit.getDefaultToolkit().
                           getImage(failinimi);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Pildiraam");
    Pildike1 ap=new Pildike1();
    f.add(ap);
    f.setSize(300, 300);
    f.setVisible(true);
    ap.start();
  }
}
```

Taust koos lilledega

Nagu iga keerukama ülesande kallale tuleb asuda üksikute osade kaupa, nii ka siin on järgmiseks ette võetud lillede paigutamine taustale. Ning hoolitsetud, et lilled suhteliselt ühtlase tihedusega, kuid samas juhuslikult pinnale jaotuksid ning pinna suhtes paigal püsides inimese eest läbi liiguksid. Juurde on tulnud muutujad lilledega seotud andmete hoidmiseks. Lille kõrguse järgi saab arvestada millal ta ekraanile paistma hakkab, millal kaob ning hiljem ka leida, kas lill mõne muu alaga kattuma juhtub. Reaalarvuline lillelisamistõenäosus näitab, mitu lille luuakse keskmiselt iga sammu korral. Et väärtuseks on praegu 0.03*samm, tähendab, et iga kõrguse ekraanipunkti kohta tuleb keskmiselt 0.03 lille, ehk siis ligikaudu üks lill iga 33 punkti kohta. Lilli endeid hoitakse Vector'is; kollektsioonis, mille elemente võib vabalt lisada ja eemaldada ilma, et peaks ise muret tundma mälu eraldamise või vabastamise pärast. Aega arvutatakse ekraanipunktides. Et liikumine on ühtlase kiirusega, siis leidub ka kulunud ajaga võrdeline seos. Samahästi võiks muutujat nimetada kogunihkeks, ehk maapinna liikumise kogu punktide arvuks.

Lillede haldamise tarbeks on loodud mitu alamprogrammi: lisamise, eemaldamise ja joonistamise tarbeks.

Lillelisamistõenäosus näitas, mitu lille tuleb ühe joonistustsükli eel lisada. Kui see väärtus on näiteks 2,7, siis kaks lille lisatakse kindlasti, kolmas aga tõenäosusega 0,7 ehk 70%. Muutuja lt liigub tõenäosuse algsest väärtusest kuni nullini. Kuni lt väärtus on suurem kui 1, lisatakse lill kindlasti, sest Math.random()-i väljastatu on vahemikus 0<=x<1. Jääb aga lt väärtus alla ühe, siis sõltub Math.random'i tegevusest, kas tuleb veel lill või mitte. Lill luuakse uue objektina, millele jäetakse meelde tema x-koordinaat (muutumatu) ning aeg ehk maapinna liikumise teepikkus ekraanipunktides lille lisamise ajaks. Selle järgi on võimalik pärast arvutada, kus lill ekraanil peaks asuma, sest kõik lilled luuakse vaikimisi ülaserva. X-koordinaat leitakse juhuslikult pea kogu nähtava laiuse ulatuses. Kümme punkti võetakse laiusest x-i leidmisel maha, et ei tekiks lille, mis ekraanil sugugi näha poleks. Meeldejäetud x tähistab lille vasakut serva.

```
 void lisaUusiLilli(){
  for(double lt=lillelisamistoenaosus; lt>0; lt=lt-1){
   if(Math.random()<lt){
     lillekesed.addElement(new Lilleke2((int)((laius-10)*Math.random()), aeg));
   }
  }
 }
```

Lille klass näeb välja suhteliselt lihtne. Vaid kaks muutujat ning konstruktor algandmete sisestamiseks. Siin näites on ta paigutatud eraldi klassina samasse faili, kuid selle võiks paigutada ka sootuks omaette faili samasse kataloogi või siis hoopis sisemise klassina Pildike2 sisse. Eraldi failis saaks Lilleke2-te kasutada soovi korral otse ka teised klassid. Sisemise klassina paigutamise puhul aga pole muret et mõni muu samanimeline klass kusagil segadust tekitama hakkaks.

```
class Lilleke2{
  int x, algaeg;
  public Lilleke2(int x1, int algaeg1){
    x=x1; algaeg=algaeg1;
  }
}
```

Iga sammu juures kontrollitakse kõik lilled läbi ning eemaldatakse alt üle ääre sattunud. Käsuga elementAt küsitakse lillekeste vektorist välja järjekorranumbrile vastav lill. Tüübimuundus on tarvilik, kuna Vector hoiab kõiki andmeid ülemklassina Object, meil on aga tarvis lillelt küsida vaid temale omase välja algaeg väärtust.

```
  void eemaldaVanadLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke2 l=(Lilleke2)lillekesed.elementAt(i);
      if(aeg-l.algaeg>korgus+lillekorgus){
        lillekesed.removeElementAt(i);
        i--;
      }
    }
  }
```

Joonistamine läks pikemaks. Ehkki paint näeb välja ilus lühike, on tema ülesannet täitma tulnud hulk abilisi.

```
  public void paint(Graphics g){
    koostaPilt();
    g.drawImage(pilt, 0, 0, this);
  }
```

Lisaks taustale tuleb sisse lugeda ka lille pilt. Samuti ei joonistata tausta enam otse ekraanile, vaid pannakse mälus enne taust ja lilled uue pildi peale kokku ning alles seejärel näidatakse tulemus ekraanile. Mälupildi koostamiseks on createImage, mis jällegi rakendi puhul pole enne võimeline käivituma kui esimese paint-i ajal. Loodud pildi käest küsitud piltg jääb globaalsena üle kogu isendi kättesaadavaks, et ei pea muudele joonistavatele alamprogrammidele seda eraldi kätte jagama. Nagu näha, pärast iga sammu eemaldatakse vanad lilled, lisatakse uusi, joonistatakse taust mälupildi põhjaks ning lilled ükshaaval sellele.

```
  void koostaPilt(){
    if(taust==null)taust=laePilt("rohetaust320x480.gif");
    if(lill==null)lill=laePilt("lill1.gif");
    if(pilt==null){
      pilt=createImage(laius, korgus);
      piltg=pilt.getGraphics();
    }
    nihe=nihe+samm;
    aeg=aeg+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    eemaldaVanadLilled();
    lisaUusiLilli();
    piltg.drawImage(taust, 0, nihe-mustripikkus, this);
    joonistaLilled();
  }
```

Lillede joonistamisel käiakse lihtsalt läbi kõik vektoris olevad lilled ning paigutatakse nende andmete järgi lillepilt taustapildile. Arvutus aeg-lille algaeg näitab lille asukoha ekraanil. Lisaks võetakse maha veel lillekõrgus, et lill asuks nähtavasse alasse sisenema oma alumise poolega ja mitte ei tekiks järsku tervikuna inimese vaatevälja.

```
  void joonistaLilled(){
```

```
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke2 l=(Lilleke2)lillekesed.elementAt(i);
      piltg.drawImage(lill, l.x, aeg-l.algaeg-lillekorgus, this);
    }
  }
```

Ja tervikuna liikuva tausta ja juhuslike lilledega rakenduse kood.

```
import java.applet.Applet;
import java.awt.*;
import java.util.Vector;


public class Pildike2 extends Applet implements Runnable{
  Image taust;
  Image pilt;
  Graphics piltg;
  Image lill;
  int lillekorgus=100;
  int mustripikkus=160;
  int samm=3;
  int paus=50;
  int nihe=0;
  int aeg=0;
  int laius=300, korgus=300;
  Vector lillekesed=new Vector();
  double lillelisamistoenaosus=0.03*samm;
  boolean veel=false;

  public void paint(Graphics g){
    koostaPilt();
    g.drawImage(pilt, 0, 0, this);
  }
  public void update(Graphics g){
    paint(g);
  }


  void koostaPilt(){
    if(taust==null)taust=laePilt("rohetaust320x480.gif");
    if(lill==null)lill=laePilt("lill1.gif");
    if(pilt==null){
      pilt=createImage(laius, korgus);
      piltg=pilt.getGraphics();
    }
    nihe=nihe+samm;
    aeg=aeg+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    eemaldaVanadLilled();
    lisaUusiLilli();
    piltg.drawImage(taust, 0, nihe-mustripikkus, this);
    joonistaLilled();
  }


  void lisaUusiLilli(){
   for(double lt=lillelisamistoenaosus; lt>0; lt=lt-1){
    if(Math.random()<lt){
      lillekesed.addElement(new Lilleke2((int)((laius-10)*Math.random()), aeg));
    }
   }
  }

  void eemaldaVanadLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke2 l=(Lilleke2)lillekesed.elementAt(i);
      if(aeg-l.algaeg>korgus+lillekorgus){
        lillekesed.removeElementAt(i);
        i--;
      }
    }
  }

  void joonistaLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke2 l=(Lilleke2)lillekesed.elementAt(i);
      piltg.drawImage(lill, l.x, aeg-l.algaeg-lillekorgus, this);
    }
  }

  public void start(){
    veel=true;
    new Thread(this).start();
  }
```

```
  public void run(){
    while(veel){
        repaint();
      try{Thread.sleep(paus); }catch(Exception e){}
    }
  }
  public void stop(){
    veel=false;
  }

  Image laePilt(String failinimi){
     try{
      return getImage(getCodeBase(), failinimi);
     }catch(Exception e){}
      return Toolkit.getDefaultToolkit().getImage(failinimi);
  }

public static void main(String argumendid[]){
    Frame f=new Frame("Pildiraam");
    Pildike2 ap=new Pildike2();
    f.add(ap);
    f.setSize(300, 300);
    f.setVisible(true);
    ap.start();
  }
}

class Lilleke2{
  int x, algaeg;
  public Lilleke2(int x1, int algaeg1){
    x=x1; algaeg=algaeg1;
  }
}
```

Liigutatav putukas

Ehkki lõpupoole soovin näha nii lilli kui putukaid koos liiklemas, on hea väiksemaid osi alustuseks eraldi poovida. Järgnevalt saab putukat noolte abil liigutada omasoodu liikuva tausta kohal. Putuka tarvis on juurde tulnud muutujad tema asukoha kohta, samuti sammu pikkus, mõõtmed ja pilt. Joonistatakse sarnaselt lilledega, st., et kõigepealt koostatakse mälus pilt taustast ja putukast ning siis kantakse tulemus tervikuna ekraanile.

Põhiline lisandus on putuka liigutamine klaviatuuri abil. Klahvisündmuste kuulamiseks on realiseeritavaks liideseks lisandunud KeyListener paketist java.awt.event. Liidesega koos kolm kohustuslikult realiseeritavat meetodit: keyPressed, keyReleased ja keyTyped. Et reageerida soovime vaid esimesele, on ülejäänute koodiosa tühi.

Klahvivajutusele reageerimisel küsitakse kõigepealt klahvi kood, et oleks võimalik asuda võrdlema, millisele klahvile vajutati. Vasaku noole puhul kontrollitakse kõigepealt, et putukas oleks vasakust äärest vähemalt oma sammupikkuse kaugusel ning sel puhul vähendatakse putuka x-koordinaadi väärtust sammu jagu. Nõnda võin vasakut noolt vajutades liikuda putukaga vasaku serva lähedale, kuid mitte kaugemale. Pole karta, et putukas ekraani pealt lahkuks. Sarnane kontroll on ka teiste külgede juures, kuid paremal ja all arvestatakse lisaks veel putuka mõõtmetega, et ka parem ega alumine külg üle piiri ei läheks.

```
  public void keyPressed(KeyEvent e){
    int kood=e.getKeyCode();
    if((kood==KeyEvent.VK_LEFT) && (putukax>putukasamm))putukax-=putukasamm;
    if((kood==KeyEvent.VK_RIGHT) && (putukax<laius-putukasamm-putukalaius))
        putukax+=putukasamm;
```

```
    if(kood==KeyEvent.VK_UP && putukay>putukasamm)putukay-=putukasamm;
    if(kood==KeyEvent.VK_DOWN && putukay<korgus-putukasamm-putukakorgus)
        putukay+=putukasamm;
  }
```

Muu kood on küllalt sarnane tausta liikumise näitega. Lõim hoolitseb omasoodu sagedase pildiuuenduse eest, lisaks taustapildile on aga tarvis lisada ka putukas vastavalt oma koordinaatidele.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Pildike3 extends Applet implements Runnable, KeyListener{
  Image taust;
  Image pilt;
  Graphics piltg;
  Image putukas;
  int putukax=100, putukay=150;
  int putukalaius=30, putukakorgus=15;
  int putukasamm=4;
  int mustripikkus=160;
  int samm=3;
  int paus=50;
  int nihe=0;
  int aeg=0;
  int laius=300, korgus=300;
  boolean veel=false;

  public Pildike3(){
    addKeyListener(this);
    requestFocus();
  }
  public void paint(Graphics g){
    koostaPilt();
    g.drawImage(pilt, 0, 0, this);
  }
  public void update(Graphics g){
    paint(g);
  }

  void koostaPilt(){
    if(taust==null)taust=laePilt("rohetaust320x480.gif");
    if(putukas==null)putukas=laePilt("sirelane.gif");
    if(pilt==null){
      pilt=createImage(laius, korgus);
      piltg=pilt.getGraphics();
    }
    nihe=nihe+samm;
    aeg=aeg+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    piltg.drawImage(taust, 0, nihe-mustripikkus, this);
    piltg.drawImage(putukas, putukax, putukay, this);
  }

  public void start(){
    veel=true;
    new Thread(this).start();
  }

  public void run(){
    while(veel){
        repaint();
      try{Thread.sleep(paus); }catch(Exception e){}
    }
  }
  public void stop(){
    veel=false;
  }
  public void keyPressed(KeyEvent e){
    int kood=e.getKeyCode();
    if((kood==KeyEvent.VK_LEFT) && (putukax>putukasamm))putukax-=putukasamm;
    if((kood==KeyEvent.VK_RIGHT) && (putukax<laius-putukasamm-putukalaius))
        putukax+=putukasamm;
    if(kood==KeyEvent.VK_UP && putukay>putukasamm)putukay-=putukasamm;
    if(kood==KeyEvent.VK_DOWN && putukay<korgus-putukasamm-putukakorgus)
        putukay+=putukasamm;
  }
```

```
  public void keyReleased(KeyEvent e){}
  public void keyTyped(KeyEvent e){}
  Image laePilt(String failinimi){
     try{
      return getImage(getCodeBase(), failinimi);
     }catch(Exception e){}
      return Toolkit.getDefaultToolkit().
                          getImage(failinimi);
  }

  public static void main(String argumendid[]){
    Frame f=new Frame("Pildiraam");
    Pildike3 ap=new Pildike3();
    f.add(ap);
    f.setSize(300, 300);
    f.setVisible(true);
    ap.start();
  }
}
```

Nektarit imev putukas

Liikuvad lilled ja liigutatav putukas omaette läbi proovitud, nüüd kannatab nad ühte kesta kokku panna. Juures on omadus, kus putukas lilleni jõudmisel selle nektarist tühjaks imeb, nii et viimane pärast imemist seest tühjemana paistab. Ka lilleklassi sai veidi täiendatud: nüüd on sel lisaks oma x-koordinaadile ja algusajale meeles, kas on ta juba nektarist tühjaks imetud või veel mitte. Ning nii nagu objektorienteeritud programmile kohane, sai tühjuse määramiseks ja küsimiseks koostatud eraldi meetodid. Nii on näiteks võimalik oleku muutumisel kontrollida, kas uus väärtus sobib või soovi korral toimingud kuhugile testiks logida. Samuti võiks praeguse programmi korral olla lubatud tühi vaid tõeseks muuta, sest iseenesest juba tühjaks imetud õied siin enam mesimahla ei tekita.

```
class Lilleke4{
  int x, algaeg;
  boolean tyhi=false;
  public Lilleke4(int x1, int algaeg1){
    x=x1; algaeg=algaeg1;
  }
  void paneTyhi(boolean kasTyhi){
    tyhi=kasTyhi;
  }
  boolean kasTyhi(){
    return tyhi;
  }
}
```

Teadmaks, kas putukas mõne lille pihta satub, kontrollitakse läbi kõikide loetelus olevate lillede kaugused putukast. Kontrollimisel abiks põhikoolist tuttav Pythagorase teoreem, et täisnurkse kolmnurga külgede ruutude summa on võrdne pikima külje ruuduga. Imemiskaugus on

paigutatud eraldi muutujasse, et oleks võimalik määrata, kui kaugelt suudab putukas nektari kätte saada. Esimene mõte imemisvõimaluse leidmisel tõenäoliselt tuleb, et peaks leidma putuka ja lille vahelise kauguse ning siis võrdlema, kas leitud suurus on imemiskaugusest väiksem. Et aga ruutjuure arvutamine nõuab arvutilt küllalt palju protsessoritehteid ning iga sammu ajal on tarvilik leida hulga lillede ja putuka vaheline kaugus, siis annab programmi sujuvamaks teha, kui vaid võrrelda külgede ruutude summat imemiskauguse ruuduga. Viimane ei muutu ning selle saab vähemasti enne sammu algust valmis arvutada.

Kui leitakse, et lill putukale piisavalt lähedale sattus, siis antakse lillele teada, et mingu ta tühjaks. Samuti palutakse Toolkit'il tekitada väikene kõll.

```
  void kontrolliImemisi(){
    int kauguseruut=imemiskaugus*imemiskaugus;
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      int lilley=aeg-l.algaeg-lillekorgus/2;
      int xkaugus=putukax-l.x;
      int ykaugus=putukay-(lilley-lillekorgus/2);
      if(!l.kasTyhi() && (xkaugus*xkaugus+ykaugus*ykaugus)<kauguseruut){
        l.paneTyhi(true);
        Toolkit.getDefaultToolkit().beep();
      }
    }
  }
```

Joonistamise puhul tuleb siis iga lille puhul otsustada, milline pilt lillest välja näidata. Nektarit täis lille pilt on nime all lill, tühjaks imetu oma lill2. Avaldis (l.kasTyhi())?lill2:lill väljastab sulgudes oleva tõese tingimuse puhul küsimärgile kohe järgneva väärtuse, ehk pildi lill2. Kui aga lill pole veel tühjaks imetud, võetakse tulemus pärast koolonit ehk pilt muutujast lill.

```
  void joonistaLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      Image lillepilt=(l.kasTyhi())?lill2:lill;
      piltg.drawImage(lillepilt, l.x, aeg-l.algaeg-lillekorgus, this);
    }
  }
```

Ja taas kogu peaklassi kood tervikuna.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;


public class Pildike4 extends Applet implements Runnable, KeyListener{
  Image taust;
  Image pilt;
  Graphics piltg;
  Image lill, lill2;
  int lillekorgus=100;
  Image putukas;
  int putukax=100, putukay=150;
  int putukalaius=30, putukakorgus=15;
  int putukasamm=4;
  int imemiskaugus=10;
  int mustripikkus=160;
  int samm=1;
  int paus=50;
  int nihe=0;
  int aeg=0;
  int laius=300, korgus=300;
  Vector lillekesed=new Vector();
  double lillelisamistoenaosus=0.03*samm;


  boolean veel=false;

  public Pildike4(){
    addKeyListener(this);
  }
  public void paint(Graphics g){
    koostaPilt();
```

```
    g.drawImage(pilt, 0, 0, this);
  }
  public void update(Graphics g){
    paint(g);
  }


  void koostaPilt(){
    if(taust==null)taust=laePilt("rohetaust320x480.gif");
    if(putukas==null)putukas=laePilt("sirelane.gif");
    if(lill==null)lill=laePilt("lill1a.gif");
    if(lill2==null)lill2=laePilt("lill1.gif");
    if(pilt==null){
      pilt=createImage(laius, korgus);
      piltg=pilt.getGraphics();
    }
    nihe=nihe+samm;
    aeg=aeg+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    eemaldaVanadLilled();
    lisaUusiLilli();
    kontrolliImemisi();
    piltg.drawImage(taust, 0, nihe-mustripikkus, this);
    joonistaLilled();
    piltg.drawImage(putukas, putukax, putukay, this);
  }

  void lisaUusiLilli(){
   for(double lt=lillelisamistoenaosus; lt>0; lt=lt-1){
    if(Math.random()<lt){
      lillekesed.addElement(new Lilleke4((int)((laius-10)*Math.random()), aeg));
    }
   }
  }

  void eemaldaVanadLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      if(aeg-l.algaeg>korgus+lillekorgus){
        lillekesed.removeElementAt(i);
        i--;
      }
    }
  }

  void kontrolliImemisi(){
    int kauguseruut=imemiskaugus*imemiskaugus;
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      int lilley=aeg-l.algaeg-lillekorgus/2;
      int xkaugus=putukax-l.x;
      int ykaugus=putukay-(lilley-lillekorgus/2);
      if(!l.kasTyhi() && (xkaugus*xkaugus+ykaugus*ykaugus)<kauguseruut){
        l.paneTyhi(true);
        Toolkit.getDefaultToolkit().beep();
      }
    }
  }

  void joonistaLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      Image lillepilt=(l.kasTyhi())?lill2:lill;
      piltg.drawImage(lillepilt, l.x, aeg-l.algaeg-lillekorgus, this);
    }
  }



  public void start(){
    veel=true;
    new Thread(this).start();
  }


  public void run(){
    while(veel){
        repaint();
      try{Thread.sleep(paus); }catch(Exception e){}
    }
  }
  public void stop(){
    veel=false;
  }
  public void keyPressed(KeyEvent e){
```

```
    int kood=e.getKeyCode();
    if((kood==KeyEvent.VK_LEFT) && (putukax>putukasamm))putukax-=putukasamm;
    if((kood==KeyEvent.VK_RIGHT) && (putukax<laius-putukasamm-putukalaius))
        putukax+=putukasamm;
    if(kood==KeyEvent.VK_UP && putukay>putukasamm)putukay-=putukasamm;
    if(kood==KeyEvent.VK_DOWN && putukay<korgus-putukasamm-putukakorgus)
        putukay+=putukasamm;
  }
  public void keyReleased(KeyEvent e){}
  public void keyTyped(KeyEvent e){}
  Image laePilt(String failinimi){
     try{
      return getImage(getCodeBase(), failinimi);
     }catch(Exception e){}
      return Toolkit.getDefaultToolkit().getImage(failinimi);
  }


  public static void main(String argumendid[]){
    Frame f=new Frame("Pildiraam");
    Pildike4 ap=new Pildike4();
    f.add(ap);
    f.setSize(300, 300);
    f.setVisible(true);
    ap.start();
  }
}
```

Sirelane mängu alguses

Esimesed lilled

Kaks tühjaks imetud lille

Püüdlus terviku poole

Nähtavalt on juurde tulnud kerimisribad putuka liikumistundlikkuse, maapinna liikumiskiiruse ja lillede tiheduse määramiseks. Näha veel väike statistika ning klahve vajutades tunda, et putukas

mitte ei hüppa klahvivajutuse peale, vaid klahvidega saab pigem putuka suunda ja kiirust määrata. Ühes sellega on muutunud või lisandunud hulga koodi.

Kõigepealt tuleb kerimisribad mälus valmis luua. Sulgude sees näha neil hulga parameetreid. Scrollbar.HORIZONTAL näitab, et riba on pikali. Sooviks püstist riba, oleks vastavaks konstandiks Scrollbar.VERTICAL. Järgnevad arvud: algne väärtus, nupu laius, vähim ja suurim võimalik väärtus. Nagu siit aimata võib, määratakse ja küsitakse kerimisriba väärtusi arvude abil.

```
  Scrollbar sbtaustasamm=new Scrollbar(Scrollbar.HORIZONTAL, 10, 5, 0, 100);
```

Et kerimisriba sündmustele annaks reageerida, selleks on klassi realiseeritavate liideste loendisse lisandunud AdjustmentListener

```
public class Pildike5 extends Applet implements Runnable, KeyListener, AdjustmentListener{
```

Liidesega käib koos meetod adjustmentValueChanged, mis on võimalik kerimisriba liigutamise peale käivitada. Iga käivituse puhul küsitakse kõigist kolmest kerimisribast väärtused ja paigutatakse vastavatesse muutujatesse. Iseenesest oleks võimalik küsida meetodi parameetriks tulnud AdjustmentEvent'i käest käsuga e.getSource(), et millist riba just liigutati. Kuna aga kerimisribast väärtuse küsimine ei nõua kuigivõrd ressurssse, siis pole eristust tehtud. Samuti oleks võimalik läbi ajada sootuks ilma sammupikkuse või mõne muu muutujata, küsides igal vajaminemiskorral väärtuse otse kerimisribast.

Kerimisribast saab väärtuse küsida vaid täisarvuna. Et aga liikumisel saaks asukohti sujuvamalt arvutada, on liikumisega seotud väärtused salvestatud reaalarvudena. Putuka ja maapinna liikumise puhul on kerimisribalt väärtus jagatud kümnega. Lillelisamistõenäosuse puhul aga, kus väärtused väiksemad ning kümnendkohtadel suurem tähtsus - tuhandega. Lõpuks on palutud lõuend fookusesse kutsuda, et kasutaja klahvivajutused taas lõuendilt kinni püütavad oleksid. Lõuendil asuvad taust, lilled ja putukas.

```
  public void adjustmentValueChanged(AdjustmentEvent e){
    samm=sbtaustasamm.getValue()/10.0;
    kiirusesamm=sbputukatundlikkus.getValue()/10.0;
    lillelisamistoenaosus=sblilletoenaosus.getValue()/1000.0*samm;
    louend.requestFocus();
  }
```

Graafikakomponendid paigutatakse konstruktoris. Kui eelmises näites joonistati rakendi enese pinnale, siis nüüd kasutatakse selleks eraldi lõuendit. Nõnda on kergem hoolitseda, et kerimisribad sisestusfookust enesele ei haarakse. Veel taseme jagu viisakam oleks kogu liikumine ja joonistamine jätta eraldi komponendi sisse ning komponendi külge ehitada meetodid, mille abil saab parameetreid muuta. Siis luua kestprogramm, mis paigutaks enese peale nii tolle joonistava komponendi kui kerimisribad ning kas vahendaks kerimisribade teated loodud komponendile või paluks ribadel otse oma teated sinna saata.

Pildike5 paigutushalduriks on valitud BorderLayout. Nii saab paigutada (alla) serva juhtribad ning joonistuskomponendi venitada üle muu pinna, nii et rakend oleks ühtlaselt kaetud.

Alumised kerimisribad koos nende sisu selgitavate siltidega paigutati omaette paneeli. Nii saab valmis ehitatud ploki pärast tervikuna rakendi allserva paigutada. Paneeli paigutushalduriks määrati GridLayout kolme rea ja kahe veeruga. Ning edasi paigutati sinna järgemööda sisse nii kirjeldavad sildid kui ribad ise.

```
  public Pildike5(){
    setLayout(new BorderLayout());
    Panel p1=new Panel(new GridLayout(3, 2));
    p1.add(new Label("Maapinna kiirus")); p1.add(sbtaustasamm);
    p1.add(new Label("Putuka liikumistundlikkus")); p1.add(sbputukatundlikkus);
    p1.add(new Label("Lillede sagedus")); p1.add(sblilletoenaosus);
    add(p1, BorderLayout.SOUTH);
    add(louend, BorderLayout.CENTER);
    louend.addKeyListener(this);
    sbtaustasamm.addAdjustmentListener(this);
```

```
    sbputukatundlikkus.addAdjustmentListener(this);
    sblilletoenaosus.addAdjustmentListener(this);
  }
```

Asukohtade arvutamisel on juurde tulnud putuka liikumise arvutus. Nii nagu maapind, nii ka putukas liigub üldjuhul ühtlase kiirusega. Enne putuka sammu võrra liikumist kontrollitakse, kas uus soovitav asukoht asub liikumiseks sobiliku pinna sees. Meetodi esimesed kaks koordinaati tähistavad uuritavat kohta, viimased kaks lubatud ala laiust ja kõrgust. Eeldatakse, et ala vasak ja ülemine serv hakkavad nullist.

```
  boolean sees(double x, double y, double x2, double y2){
    if(x>=0 && x<x2 && y>=0 && y<y2) return true;
    return false;
  }
```

Kui aga juhtub, et putukas on sattunud lubatud ala serva lähedale ning edasi pole võimalik liikuda, siis seatakse ta liikumiskiirus (sammu pikkus ühe joonistuse jooksul) nulliks.

```
  void arvutaAsukohad(){
    nihe=nihe+samm;
    aeg=aeg+samm;
    if(nihe>mustripikkus)nihe=nihe-mustripikkus;
    if(sees(putukax+putukaxkiirus, putukay+putukaykiirus,
                laius-putukalaius, korgus-putukakorgus)){
      putukax+=putukaxkiirus;
      putukay+=putukaykiirus;
    } else {
      putukaxkiirus=putukaykiirus=0;
    }
  }
```

Reageering klahvivajutusele on mõnevõrra muutunud. Kui ennist muutus vajutusel putuka asukoht, siis nüüd püütakse muuta ta liikumise kiirust. Näitena näha vasak nooleklahv. Kui putukas liikus ennist vasakule (st. putukaxkiirus oli nullist väiksem) sel juhul klahvile vajutades vasakule liikumise kiirus kasvab kiirusesammu võrra, ehk kiirusest lahutatakse kiirusesammu väärtus. Kui aga putukas juhtus enne paigal seisma või paremale liikuma, siis määratakse uueks kiiruseks -kiirusesamm, ehk sama palju, kui muidu iga vajutamisega kiirust juurde tuleb. Nõnda ei pea kiiruse suuna muutmisel liialt aega kulutama pidurdamisele, vaid võib küllalt järsku teises suunas liikuma hakata, nagu putuka lennu puhul ikka näha võib.

```
  public void keyPressed(KeyEvent e){
    int kood=e.getKeyCode();
    if(kood==KeyEvent.VK_LEFT){
      if(putukaxkiirus<0)putukaxkiirus-=kiirusesamm;
      else putukaxkiirus=-kiirusesamm;
    }
    //...
  }
```

Juurde tuli mõningane statistika, et kasutajal oleks näha, kaugele ta mänguga jõudnud on. Kulunud aja arvutamiseks küsitakse käivitamise ajal eraldi muutujasse alghetke väärtus. Tegemist küllalt suure ja väheütleva arvuga: millisekundeid alates aastast 1970.

```
  long alghetk=new Date().getTime();
```

Kui nüüd aga millalgi uuesti süsteemi käest aeg küsida, siis nende kahe väärtuse vahe ütleb, palju mängu algusest aega kulunud on.

```
  void joonistaStatistika(){
    String st="Lilli: "+lilli+" Imetud: "+imetud+
```

```
        " Kadunud: "+kadunud+" Aeg: "+(new Date().getTime()-alghetk)/1000;
    piltg.setColor(Color.white);
    piltg.drawString(st, 30, korgus-10);
  }
```

Rakendusele pandi kaasa heli. Pidevalt korduv linnutaust, et veidigi tekiks mulje niidul lendavast putukast ning märku andev sahin, kui putukas lillest nektari kätte sai. Taustaga on lihtsam. Kui esimest korda joonistama asutakse, siis palutakse muutujasse laadida linnutaust ning käsu loop abil pannakse too korduvalt end ketrama.

```
    if(linnutaust==null){
      linnutaust=laeKlipp("linnutaust.au");
      linnutaust.loop();
    }
```

Lille tabamise sahina saaks ka lihtsamal juhul laadida ning käsuga play mängima panna. Kui aga lilli palju ning sahinaheli vähegi pikem, siis kipuvad järjestikustel tabamustel helid kattuma ning üksikud tabamused ei eristu kõrvale ilusti ja loendatavalt. Sahinahelide eristamiseks loodi omaette klass. Lähem kirjeldus näha juba kommentaarides.

```
/**
* Lõimeklass, mille abil saab meloodiajuppe mängida. Sünkroniseerimise abil hoolitsetakse,
* et mängualguste vahel oleks vähemalt poolesekundiline paus. Luku
* (loa)ga sünkroniseeritud plokki pääseb korraga vaid üks lõim. Ülejäänud
* peavad selle ees ootama, kuni eelmine on plokist väljunud.
*/
class Piiksuja extends Thread{
  static Object lukk=new Object();
  static java.applet.AudioClip sahin;
  public void run(){
   synchronized(lukk){
    sahin.play();
    try{Thread.sleep(500);}catch(Exception e){}
   }
  }
}
```

Nagu näha, on Piiksuja nii lukk kui sahin staatilised muutujad. See tähendab, et neile on võimalik ligi pääseda sõltumata klassi eksemplaride arvust. Lukuobjekt luuakse programmi käivitamisel ning sama isend on kättesaadav kõigile klassi isenditele - nõndamoodi saab selle järgi sünkroniseerimisel otsustada, et korraga ei juhtuks mitu sahinat mängima. Piiksuja sahin laetakse koos muude klippidega ja paigutatakse sinna staatilisse muutujasse.

```
    if(Piiksuja.sahin==null) Piiksuja.sahin=laeKlipp("sahin.au");
```

Kui nüüd käivitamise juures jõutakse niikaugele, et on paras aeg sahistada, siis luuakse lõimeklassist Piiksuja uus eksemplar ning palutakse start abil eraldi lõimes ta run käivitada.

```
      if(!l.kasTyhi() && (xkaugus*xkaugus+ykaugus*ykaugus)<kauguseruut){
        l.paneTyhi(true);
        imetud++;
        new Piiksuja().start();
      }
```

Kui eelmist mängijat pooleli pole, siis asutakse pea kohe mängima. Kui aga eelmine lõime eksemplar peatub luku abil sünkroniseeritud ploki sees, siis jääb uus mängija järjekorda, kuni eelmine on oma töö lõpetanud ning plokist väljunud. Kui nüüd liikuda putukaga läbi tiheda lillesalu, võib järgemööda eristatult kuulda mitut sahinat - iga tabatud lille kohta üht.

```
   synchronized(lukk){
    sahin.play();
    try{Thread.sleep(500);}catch(Exception e){}
   }
```

Nagu koodist näha, pole rakendiklassil Pildike5 enam main-meetodit. Lähem seletus klassi kommentaarides.

```
/**
* Alamklass liikuvate lilledega rakendile Pildike5. Meetod laeKlipp on siin üle kaetud,
* sest rakendil saadakse heliklipi andmed käsust getAudioClip, rakendusel aga samaotstarbeliseks
* käsuks Applet.newAudioClip. Viimane kuulub aga JDK koosseisu alates versioonist 1.2 ning
* varasema versiooni seilurid annavad neile tundmatu meetodi sisse lugemisel klassi kohta veateate
* ning keelduvad vastava klassiga edaspidi tegelemast kartes turvamuresid. Kui aga vastav meetod
* siin üle katta, siis ülemklass on sellest meetodist prii ja võib rahus rakendis töötada,
* käsurealt käivitades aga võetakse siinse alamklassi üle kaetud käsklus ning pannakse
* tööle.
*/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.io.*;
class Pildike5Raam extends Pildike5{
  AudioClip laeKlipp(String failinimi){
    try{return Applet.newAudioClip(new File(failinimi).toURL());}catch(Exception e){}
    return null;
  }

  public static void main(String argumendid[]){
    Frame f=new Frame("Pildiraam");
    Pildike5Raam ap=new Pildike5Raam();
    f.add(ap);
    f.setSize(300, 400);
    f.setVisible(true);
    ap.start();
    f.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      }
    );
  }
}
```

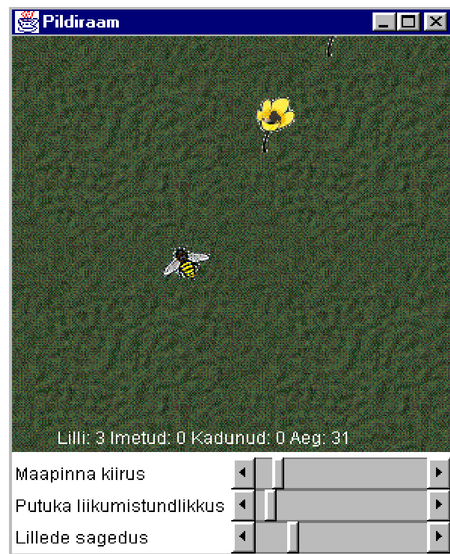
Nõnda on tehtud muudatused/täiendused üle vaadatud ning rakenduse terviku kood loodetavasti arusaadav.

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


public class Pildike5 extends Applet implements Runnable, KeyListener, AdjustmentListener{
  Image taust;
  Image pilt;
  AudioClip linnutaust;
  Graphics piltg;
  Image lill, lill2;
  int lillekorgus=100, lillelaius=50;
  Image putukas;
  double putukax=100, putukay=150;
  double putukaxkiirus=0, putukaykiirus=0;
  double kiirusesamm=0.2;
  int putukalaius=35, putukakorgus=35;
  int imemiskaugus=10;
  int mustripikkus=160;
  double samm=1;
  int paus=50;
  double nihe=0;
  double aeg=0;
  int laius=300, korgus=300;
  int lilli=0, imetud=0, kadunud=0;
  long alghetk=new Date().getTime();
  Vector lillekesed=new Vector();
  double lillelisamistoenaosus=0.02*samm;
  Canvas louend=new Canvas(){
    public boolean isFocusTraversable(){
      return true;
    }
  };
```

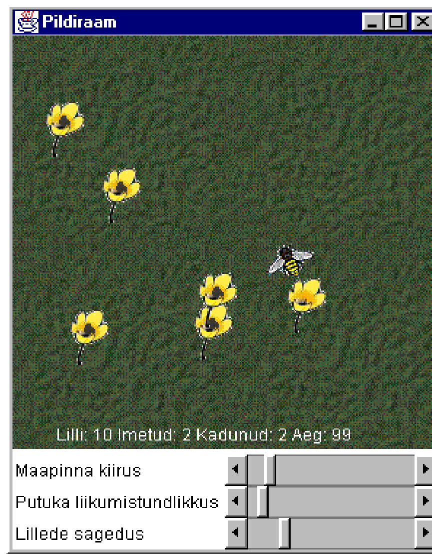
```
Scrollbar sbtaustasamm=new Scrollbar(Scrollbar.HORIZONTAL, 10, 5, 0, 100);
Scrollbar sbputukatundlikkus=new Scrollbar(Scrollbar.HORIZONTAL, 2, 1, 1, 20);
Scrollbar sblilletoenaosus=new Scrollbar(Scrollbar.HORIZONTAL, 20, 1, 0, 100);
boolean veel=false;

public Pildike5(){
  setLayout(new BorderLayout());
  Panel p1=new Panel(new GridLayout(3, 2));
  p1.add(new Label("Maapinna kiirus")); p1.add(sbtaustasamm);
  p1.add(new Label("Putuka liikumistundlikkus")); p1.add(sbputukatundlikkus);
  p1.add(new Label("Lillede sagedus")); p1.add(sblilletoenaosus);
  add(p1, BorderLayout.SOUTH);
  add(louend, BorderLayout.CENTER);
  louend.addKeyListener(this);
  sbtaustasamm.addAdjustmentListener(this);
  sbputukatundlikkus.addAdjustmentListener(this);
  sblilletoenaosus.addAdjustmentListener(this);
}
public void adjustmentValueChanged(AdjustmentEvent e){
  samm=sbtaustasamm.getValue()/10.0;
  kiirusesamm=sbputukatundlikkus.getValue()/10.0;
  lillelisamistoenaosus=sblilletoenaosus.getValue()/1000.0*samm;
  louend.requestFocus();
}

public void joonista(){
  koostaPilt();
  louend.getGraphics().drawImage(pilt, 0, 0, this);
}


void laeKlipid(){
  if(taust==null)taust=laePilt("rohetaust320x480.gif");
  if(putukas==null)putukas=laePilt("sirelane.gif");
  if(lill==null)lill=laePilt("lill1a.gif");
  if(lill2==null)lill2=laePilt("lill1.gif");
  if(pilt==null){
    pilt=createImage(laius, korgus);
    piltg=pilt.getGraphics();
  }
  if(Piiksuja.sahin==null) Piiksuja.sahin=laeKlipp("sahin.au");
  if(linnutaust==null){
    linnutaust=laeKlipp("linnutaust.au");
    linnutaust.loop();
  }
  korgus=louend.getSize().height;
  laius=louend.getSize().width;
  louend.requestFocus();
}

void arvutaAsukohad(){
  nihe=nihe+samm;
  aeg=aeg+samm;
  if(nihe>mustripikkus)nihe=nihe-mustripikkus;
  if(sees(putukax+putukaxkiirus, putukay+putukaykiirus,
               laius-putukalaius, korgus-putukakorgus)){
    putukax+=putukaxkiirus;
    putukay+=putukaykiirus;
  } else {
    putukaxkiirus=putukaykiirus=0;
  }
}

void koostaPilt(){
  if(taust==null)laeKlipid();
  arvutaAsukohad();
  eemaldaVanadLilled();
  lisaUusiLilli();
  kontrolliImemisi();
  piltg.drawImage(taust, 0, (int)nihe-mustripikkus, this);
  joonistaLilled();
  joonistaStatistika();
  piltg.drawImage(putukas, (int)putukax, (int)putukay, this);
}



void lisaUusiLilli(){
 for(double lt=lillelisamistoenaosus; lt>0; lt=lt-1){
  if(Math.random()<lt){
    lillekesed.addElement(new Lilleke4((int)((laius-lillelaius)*Math.random()), (int)aeg));
    lilli++;
  }
 }
}
```

```
  void eemaldaVanadLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      if(aeg-l.algaeg>korgus+lillekorgus){
        if(!l.tyhi)kadunud++;
        lillekesed.removeElementAt(i);
        i--;
      }
    }
  }

  void kontrolliImemisi(){
    int kauguseruut=imemiskaugus*imemiskaugus;
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      int lilley=(int)aeg-l.algaeg-lillekorgus/2;
      int xkaugus=(int)putukax-l.x;
      int ykaugus=(int)putukay-(lilley-lillekorgus/2);
      if(!l.kasTyhi() && (xkaugus*xkaugus+ykaugus*ykaugus)<kauguseruut){
        l.paneTyhi(true);
        imetud++;
        new Piiksuja().start();
      }
    }
  }

  void joonistaLilled(){
    for(int i=0; i<lillekesed.size(); i++){
      Lilleke4 l=(Lilleke4)lillekesed.elementAt(i);
      Image lillepilt=(l.kasTyhi())?lill2:lill;
      piltg.drawImage(lillepilt, l.x, (int)aeg-l.algaeg-lillekorgus, this);
    }
  }


  void joonistaStatistika(){
    String st="Lilli: "+lilli+" Imetud: "+imetud+ " Kadunud: "+kadunud+" Aeg: "+(new Date().getTime
()-alghetk)/1000;
    piltg.setColor(Color.white);
    piltg.drawString(st, 30, korgus-10);
  }

  public void start(){
    veel=true;
    new Thread(this).start();
    if(linnutaust!=null)linnutaust.loop();
  }


  public void run(){
    while(veel){
        joonista();
      try{Thread.sleep(paus); }catch(Exception e){}
    }
  }
  public void stop(){
    veel=false;
    linnutaust.stop();
  }

  boolean sees(double x, double y, double x2, double y2){
    if(x>=0 && x<x2 && y>=0 && y<y2) return true;
    return false;
  }

  public void keyPressed(KeyEvent e){
    int kood=e.getKeyCode();
    if(kood==KeyEvent.VK_LEFT){
      if(putukaxkiirus<0)putukaxkiirus-=kiirusesamm;
      else putukaxkiirus=-kiirusesamm;
    }
    if(kood==KeyEvent.VK_RIGHT){
      if(putukaxkiirus>0)putukaxkiirus+=kiirusesamm;
      else putukaxkiirus=kiirusesamm;
    }
    if(kood==KeyEvent.VK_UP){
      if(putukaykiirus<0)putukaykiirus-=kiirusesamm;
      else putukaykiirus=-kiirusesamm;
    }
    if(kood==KeyEvent.VK_DOWN){
      if(putukaykiirus>0)putukaykiirus+=kiirusesamm;
      else putukaykiirus=kiirusesamm;
    }
  }

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

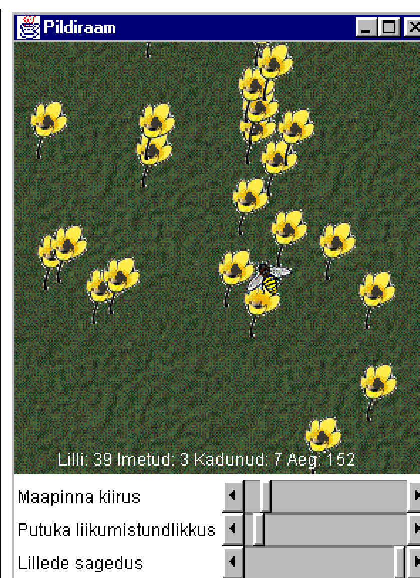
```
  public void keyTyped(KeyEvent e){}

  Image laePilt(String failinimi){
     try{
      return getImage(getCodeBase(), failinimi);
     }catch(Exception e){}
      return Toolkit.getDefaultToolkit().getImage(failinimi);
  }

  AudioClip laeKlipp(String failinimi){
      return getAudioClip(getDocumentBase(), failinimi);
  }
}
```



Esimesed lilled



Kaks imetud, kaks silmist mööda läinud



Tihedaks määratud lilleväli

Edasiarendusvõimalused

Loodud rakendus küll töötab ning on mõningase näpuharjutusena mängitav, kuid pole ta veel ei terviklik mäng ega saa seda kuigivõrd asjalikult ka mõneks muuks otstarbeks kasutada. Olemasolevat koodi vähemalt osalt mõistes on võimalik küllalt vähese vaeva abil koodi toimimist ja rakenduse väljanägemist täiesti märkimisväärselt muuta. Lihtsaimaks asenduseks ehk pildid ja heliklipid. Kui joonistada või leida muru ja lillede abil miskit muud, siis annab kergesti kokku panna koristaja maanteel või elevandi portselanikaupluses. Üks üliõpilane aga kujundas piltide muutmise abil sellest samast näitest aga näiteks küllaltki verise sõjamängu.

Kui näidet mänguks kujundada, siis tõenäoliselt tuleb üsna pea lisada punktidearvutus ja

mängu lõpp, ehk ka tasemed. Punkte kannatab kuvada statistikafunktsiooni veidi muutes. Iseseisva rakenduse puhul annab vahetulemusi faili salvestada. Rakendi puhul on turvapiirangute tõttu see veidi keerulisem, kuid kel PHP või muu veebiserveripoolse programmeerimisvahendiga kokkupuuteid ja kasutusvõimalusi, siis on ka see täiesti tehtav.

Ilus võistlusmoment tekib, kui ühe putuka asemel lendleb kaks, kummagi juhtimiseks omaette klahvid. Ning samuti võib saabuvaid pilte olla mitut tüüpi. Nii magusaid mesikaid, õiteta okaspuid kui muude vahele ära eksinud kärbsepabereid.

Näide ei pea aga mitte ainult mängu aluseks olema. Mõningase muutmise teel saab selle põhjal koostada katseseadme pidurdusteekonna pikkuse või molekulide kaootilise liikumise näitamiseks. Kõige rohkem läheb vaja pealehakkamist.

Ülesandeid

Sirelasemäng

Alustuseks on kasutada mängu põhi

- Tutvu mänguga
- Lae lähtekood, pildid ja helilõigud oma arvutisse, kompileeri ja käivita.
- Otsi Internetist pilte ning kujunda nende abil mäng oma silma järgi.
- Lisa punktidearvutus.
- Loo mängule lõpp.
- Luba samaaegselt liikuda kahel mängijal

Klahvidega liigutamine.

- Nooleklahvidega saab liigutada ekraanil olevat ringi.
- Ekraanil juhuslikus kohas paikneb rist. Jõudes ringiga selleni, hüppab rist uude juhuslikku kohta.
- Liigutatavaid ringe on kaks, kummalgi oma klahvid.
- Risti asemel on pisike pilt. All servas on kirjas, mitu korda kumbki mängija on ristini jõudnud.

Tennis

- Võrguga tenniseväljakul liigub pall (ring) vasakult paremale.
- Samaaegselt ringi liikumisega saab ekraanil liigutada reketit.
- Kummalgi serval on liigutatav reket, mille tabamisel pall tagasi põrkab. Loetakse punkte.

Ussimäng

- 10*10 ruudustikul asub vasakul ülaservas neljalüliline uss, keda saab nooleklahviga paremale liigutada.
- Uss liigub pidevalt, nooleklahvidega saab tema liikumise suunda määrata. Seinani jõudmisel uss seiskub.
- Ussiga püütakse juhuslikus kohas asuvat kuldmuna. Muna kättesaamisel uss pikeneb kahe lüli võrra ning muna tekib uude kohta. Seina või enese hammustamisel uss lüheneb nelja lüli võrra.

Tilgapüüdja

- All servas saab liigutada kaussi
- Ekraanil kukub tilk. Selle kaussi püüdmisel saab punkti.
- Tilku võib korraga kukkuda mitu. Mäng on meeldivalt kujundatud ning töötab märgatavate vigadeta.

Nooltega ralli.

- Noolte abil saab muuta ekraanil liikuva ringi kiirust.
- Külgmiste nooltega saab muuta liikumise suunda, teiste nooltega kiirust.
- Korraga võib üle võrgu sõita kaks kasutajat. Aetakse taga juhuslikus kohas paiknevat aaret, mis selleni jõudmisel hüppab uude kohta.