## Sisukord




# Kursuse "Teeme ise arvutimänge - algus"

# 5. RAAMAT

# GRAAFIKA JA ANIMATSIOON

**Tiina Kull**

**Tartu Ülikool**

**2012**

## Graafika

Lõpuks ometi GRAAFIKA juurde! Usu, kui ma ise kunagi ammu programmeerimisega alustasin, siis tundusid kõik need harjutused alguses kohutavalt tüütud ja kuidagi magedad, sest algsetel programmidel ei tundunud mitte mingit pistmist olevat nende programmidega, mida saab internetis mängida ja mida saab poest osta. Nüüd tagant järele targutades, saan aru küll, et kui pole põhikonstruktsioonidest ja algoritmilisest mõtlemisest mingit aimu, siis ei tee ka palja graafikagagi suurt midagi. Mis mu jutu moraal on:), ikka see, et loodan, et sa oled siiani kenasti vastu pidanud ja endale korraliku vundamendi ladunud.

Graafika loomiseks võtame appi juba valmis paketi programme (***Pygame***), mis aitavad meil asju ekraanile panna, neid liikuma panna, joonistada maastikke, panna juurde heli ja kõike muud mängude juurde kuuluvat.

## Miks kasutada Pygame?

Ilma Pygame'ta oleks graafika tööle saamine üsna keeruline töö isegi tavaprogrammeerija jaoks, sest siin tuleb arvestada juba väga paljude erinevate arvuti osadega, mitte lihtsalt arvuti mäluga. Tuleb arvestada operatsioonisüsteemiga, sest kõikides op-süsteemides on heli ja pildi lahendused veidi erinevad, tuleb arvestada erinevate heli- ja graafikakaartidega, protsessorite kiirusega jpm sellisega. Põhimõtteliselt peaksime tundma õppima kõigi erinevate jubinate hingeelu, nende eripära, kuidas nendega suheldakse jne, mis on üks väga suur ettevõtmine ja võtab tavaliselt aastaid, kui sellel alal päris spetsiks saada.

Pygame on aga paljude programmeeriate poolt koostatud ja õpetatud suhtlema erinevate kaartide ja op-süsteemidega, nii et meie eest on väga suur töö ära tehtud ning meil ei jäägi midagi muud üle kui nautida eelmiste programmeerijate töö vilju. Pealegi on see tasuta!

Pygame on sul juba arvutisse installeritud ja ka esimesed katsetused eelmisel ja üle-eelmisel nädalal tehtud. Sellel nädalal aga uurime ükispulki järele, kuidas ikkagi neid komponente koostatakse, akna moodustamisest kuni asjade liikuma panemiseni ekraanil ja nende juhtimiseni klaviatuuri või hiirega.

## Akna loomine

Esimene asi mängu realiseerimisel on akna loomine - piirkond, kus mäng toimuma hakkab. Selleks on vaja kirjutada kolm rida:


```
import pygame
pygame.init()
ekraan = pygame.display.set_mode([640, 480]) # nö klassikalised mõõtmed
```

Selle koodi tööle panemisel võisid näha väikest vilksatust, kuid mitte midagi muud. Mis toimub? Pygame on loodud mängude tegemiseks ja mängude juures on väga oluline see, et midagi toimuks. Mängudes ei juhtu mitte midagi ilma mängija käskudeta. Pygame'i käima panemisel hakkab vaikimisi tööle nn **sündmuste kontrolli tsükkel**, mis pidevalt kontrollib, kas midagi mängus tehakse või mitte. Kas vajutatakse mingit klahvi või liigutatakse hiirt? Niipea kui sündmuste kontrollija lõpetab töö, lõpetatakse automaatselt ka kogu programmi töö. Meie oma kolme reaga, ei võtnud kontrolli sündmuste kontrolli tsükli üle, seetõttu ta ka nii ruttu asjad kokku pakkis.

Mängu töö tagamiseks tuleb alati kirjutada programmi sisse selline tsükkel, mis kontrollib seda, millal kasutaja mängu kinni paneb. Samuti tasub while tsüklisse enne sys.exit()-t lisada rida **pygame.quit()** käsu, mis aitab vähendada Idle ja Pygame omavahelist konflikti.

Teeme seda - kirjutame sellise tsükli, mis on alati tõene:

```
import pygame, sys
pygame.init()
ekraan = pygame.display.set_mode([640, 480]) # nö klassikalised mõõtmed

while True:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
#Tsükkel kestab lõputult, sest True on alati True
#Tsükli sees aga kontrollitakse teise tsükliga
#pygame-i sündmuste jada, kui sündmuse tüüp on võrdne
# sellega, et vajutati akna sulegmise nupul (QUIT)
# siis pannakse kogu süsteem kinni, ka lõputu tsükkel:)
```

Kui sa paned antud programmilõigu käima läbi Idle, siis tavaliselt juhtub nii, et Pygame hangub ja sa pead ta jõuga pärast kasutamist kinni panema. See on tingitud Idle ja Pygame omavahelisest konfliktist. Miks see juhtub?
Sest nii Idles kui ka Pygames on sama sündmuste kontrolli tsükkel ja omavahel neid sünkroonis tööle saada on osutunud ka suurtele programmerijatele esialgu üle jõu käivaks.

Teine asi, mida sa kindlasti märkasid, ekraan mis ilmus, on sama taustaga nagu sinu arvutiekraan. See tuleb sellest, et me pole veel lihtsalt midagi käskinud aknasse joonistada, et see ei oleks vaid paljas raam:)

# Joonistamine aknasse

Joonistamist Pygames kasutatakse tavaliselt maastiku loomiseks ja mitte tegelaste loomiseks. Tegelased, kes mängudes ringi jooksevad või asjad, mida tegelased korjavad, on tavaliselt juba valmis pildid, mis on eelnevalt valmis joonistatud mõnes joonistusprogrammis. (Oluline on joonistada erinevate tegelaste erinevad asendid - seismine, paremale jooksmine, vasakule jooksmine, ronimine vms.). Joonistamise alla pygames kuuluvad värvide, kujundite, joonte jm taolise kasutamine. Vaatame, kuidas seda tehakse?

## Esimesed katsetused joonistamiseks

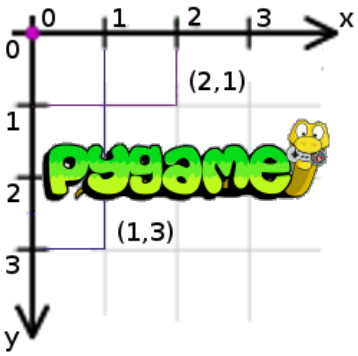

Muudame kõigepealt ekraani tausta valgeks ja seejärel lisame ekraanile punase ringi, rohelise ristküliku ja sinise joone. Kujundite ükshaaval testimiseks peab kindlasti kirjutama ka juurde rea **pygame.display.flip()**, mis uuendab ekraani pilti.

```
import pygame, sys
pygame.init()
ekraan = pygame.display.set_mode([640, 480]) # nö klassikalised mõõtmed pics
ekraan.fill([255, 255, 255]) #valge värvi kood

pygame.draw.circle(ekraan, [255, 0, 0], [50, 50], 25, 0)
#joonistab ringi, ekraanile, punase, asukoht koorinaatteljestikus 50x 50y,
#raadiusega 25, raamiga 0

pygame.draw.rect(ekraan, [0, 225, 0], [100, 50, 150, 80], 0)
#joonistab ristküliku, ekraanile, rohelise, asukoht koorinaatteljestikus 100x 50y,
#laiusega 150, kõrgusega 80 ja raamiga 0

pygame.draw.line(ekraan, [0, 0, 225], (200, 200),(400, 400), 3)
#joonistab joone, ekraanile, sinise, joone alguspunkt 200x 200y,
#joone lõpppunkt 400x 400y ja joone paksusega 3

pygame.display.flip()
# Pygame teeb ekraanipildi enne valmis ja flip() käsuga vahetab
#kogu ekraani korraga, et ei tekiks joonistamise vibratsioone

while True:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
#Tsükkel kestab lõputult, sest True on alati True
#Tsükli sees aga kontrollitakse teise tsükliga
#pygame-i sündmuste jada, kui sündmuse tüüp on võrdne
# sellega, et vajutati akna sulegmise nupul (QUIT)
# siis pannakse kogu süsteem kinni, ka lõputu tsükkel:)
```

Kõiki **pygame.draw** võimalusi ei ole mõtet siin selle väikese lehe peal välja tuua. Hea ülevaate kõikidest **pygame.draw** võimalustest leiad Pythoni pygame dokumentatsiooni lehelt, kus on iga funktsiooni juurde kirjutatud, mida ta teha oskab ja millised argumendid, millises järjekorras tuleb sulgudesse kirjutada:

http://www.pygame.org/docs/ref/draw.html

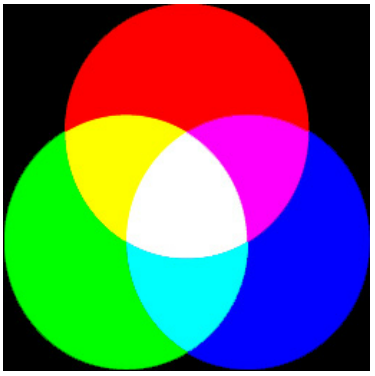
## Akna koordinaadid




Haa, või arvad, et tead, kuidas koordinaatide kaudu ekraanile kujundeid lisada? Arvad, et tead, mis asi on koordinaatteljestik?

Asi pole sugugi nii lihtne vaid tegelikult palju lihtsam:) kui koolimatemaatika koordinaatteljestik. Pygame koordinaatteljestik ei ole traditsiooniline rist *x*- ja *y*-teljest, mis jaotavad kogu ala neljaks veerandiks. Ei, pygame **(0, 0)** punkt asub mitte ekraani keskel vaid ALATI **üleval vasakus nurgas**. *X*-telg liigub paremale ja *y*-telg liigub alla. Kujundite või piltide positsioneerimiseks ekraanil tuleb arvestada alati loendamisest ülevalt vasakust nurgast. Järjekord punkti koordinaatides on aga ikka samasugune - kõigepealt *x*-telje koordinaat ja siis *y*-telje koordinaat.

## Värvid



Värvide defineerimiseks kasutatakse programmeerimises ja nii ka pygame-s kolmest arvust koosnevat kombinatsiooni, mida nimetatakse **RGB**-süsteemiks. Esimene arv näitab punase (**R**ed) doseeringut, teine arv näitab rohelise (**G**reen) doseeringut ja viimane arv näitab sinise (**B**lue) kontsentratsiooni. Iga värvi saab muuta 0-st 255-ni. Mida suurem on arv, seda rohkem on seda värvi lõpptulemuses. Kui kõik arvud keerata 0-i, siis saame musta värvi ehk värvitu tulemuse ja kui kõik värvid keerame maksimumi peale ehk 255-le, siis saame valge värvi. Kõik ülejäänud värvid saame erinevatest kombinatsioonidest. Kui keerata kõik arvud ühesuguseks, näiteks [100, 100, 100], siis saame halli tooni. Mida suuremad on ühesugused arvud, seda heledama halli saame, mida väiksemad, seda tumedama.

Tegelikult on pygame-s ka nimekiri üle 600 värvinimetusega, kui sa ei peaks tahtma kasutada RGB-süsteemi. Kõik nimed koos värvi näidisega leiad siit:

https://sites.google.com/site/meticulosslacker/pygame-thecolors

## Kujundite omavaheline ühendamine

Nüüd räägin sulle **pygame.draw**-s ühest trikitamise võimalusest, mis aitab kujundeid omavahel paiknemise suhtes siduda. Mida see tähendab?

Oletame, et ma olen keset ekraani joonistanud ristküliku või ruudu ja ma tahan täpselt selle ruudu keskele joonistada ringi, siis on väga tülikas hakata välja arvutama xy-koordinaatsüsteemis ringi keskpunkti kooridinaate. Siin tulebki appi üks **pygme.draw** objekt nimega **Rect.** Põhimõtteliselt on see sama, mis ristküliku loomise käsk, kuid teda saab kasutada ka eraldi, andes talle parameetriteks ainult koordinaadi vasakult, koorinaadi ülevalt, laiuse ja kõrguse. Kui selline objekt on defineeritud, saab käskida tema suhtes paiknema panna teisi kujundeid järgmiste käskudega:

- külgede suhtes: **top, left, bottom, right**
- nurkade suhtes: **topleft, bottomleft, topright, bottomright**
- keskel äärtes: **midtop, midleft, midbottom, midright**
- keskel: **center, centerx, centery**
- suurus: **size, width, height**

## Uuri näidet ja proovi järgi:

```
import pygame, sys
pygame.init()
ekraan = pygame.display.set_mode([640, 480]) # nö klassikalised mõõtmed pics
ekraan.fill([255, 255, 255]) #valge värvi kood

pygame.draw.circle(ekraan, [255, 0, 0], [50, 50], 25, 0)
#joonistab ringi, ekraanile, punase, asukoht koorinaatteljestikus 50x 50y,
#raadiusega 25, raamiga 0

kast = pygame.Rect(100, 50, 150, 150)
#defineerin kasti, mis on rect tüüpi

pygame.draw.rect(ekraan, [0, 225, 0], kast, 0)
#joonistan selle kasti (ruudu) reaalselt ekraanile

pygame.draw.circle(ekraan, [100, 100, 100], kast.center, 50, 0)
#joonistan halli ringi ruudu keskele raadiusega 50

pygame.draw.line(ekraan, [0, 0, 225], kast.midleft, ekraan.get_bounding_rect().center, 3)
#joonistab joone, ekraanile, sinise, joone alguspunkt on ruudu vasaku serva keskpunkt,
#joone lõpppunkt asub ekraani keskpunktis ja joone paksus on 3
# pane tähele, kuna ekraan on teist tüüpi kui tavaline ristkülik, siis
# rect-i kasutamiseks peab eelnevalt ekraani rect-ga ühendama get_boundin_rect käsuga

pygame.display.flip()
# Pygame teeb ekraanipildi enne valmis ja flip() käsuga vahetab
#kogu ekraani korraga, et ei tekiks joonistamise vibratsioone

while True:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
#Tsükkel kestab lõputult, sest True on alati True
#Tsükli sees aga kontrollitakse teise tsükliga
#pygame-i sündmuste jada, kui sündmuse tüüp on võrdne
# sellega, et vajutati akna sulegmise nupul (QUIT)
# siis pannakse kogu süsteem kinni, ka lõputu tsükkel:)
```

Oluline on mõista seda, et **Rect**'i võib defineerida mistahes kohta ja mistahes suurusega, kuid **ei pea** selle defineeritud ristküliku reaalselt ekraanile välja joonistama. Nii võin üle kogu ekraani defineerida mitmeid kaste ja kastikesi, millede kaudu saan hiljem teisi kujundeid kergemini positsioneerida ja üksteise suhtes paigutada. Sellepärast ma kutsungi seda trikitamiseks.

## Kujundite raamid

Erinevate kujundite joonistamise viimaseks parameetriks on raami paksus. Kõikides eelnevates näidetes kasutasime raami paksuseks 0-i. See tähendab seda, et kujund ei joonistata üldse mitte raamiga vaid täidetakse tervenisti värviga. Kui muuta see arv nullist suuremaks, siis joonistatakse ainult selle kujundi välispiir ja seest jäetakse tühjaks. Piiri paksus oleneb arvu suurusest. Tee katseid erinevate raami suurustega!

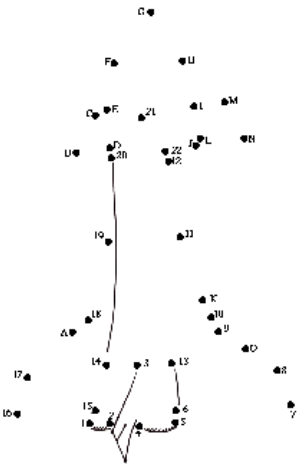
## Mullid!


```
import pygame, sys, random

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

for i in range(100):
    x = random.randint(0, 600)
    y = random.randint(0, 450)
    raadius = random.randint(5, 60)
    raam = random.randint(1, 5)
    R = random.randint(0, 255)
    G = random.randint(0, 255)
    B = random.randint(0, 255)
    pygame.draw.circle(lava, [R, G, B], (x, y), raadius, raam)
pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

Et raamidega jändamine oleks huvitavam, katseta antud näidet - igal käivitamisel tuleb erinev tulemus.


Eriti ilusa tulemuse korral salvesta:)

## Vaba käega joon



Mäletate neid mänge paberil, kus on terve rida punkte numbritega ja need tuleb nubrite järgi ühendada, mille tulemusel tekib mingi vahva pilt. Vabakäelise joone tegemine pygame-s käib täpselt sama moodi. Kõigepealt peab eelnevalt tekitama punktide joru, mida tahetakse joonega ühendada ja siis kasutatakse **pygame.draw** käsku **lines**. Punktide jada pannakse kirja tavaliselt listi, sest selles on punktide ühendamise järjekord juba määratud. Mida tihedamalt on punkte listi kirjutatud, seda sujuvam joon tuleb.

## Teeme paar näidet:


```
import pygame, sys, random

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

punktid = [(300, 50), (350, 100), (400, 50), (400, 150), (300, 250)]

pygame.draw.lines(lava, [0, 0, 0], True, punktid, 2)
#lines ühendab musta värvi joonega paksusega 2 kõik punktid listis
# punktid, True tähendab seda, et ühendatakse ka algus ja lõpppunkt

pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

Sujuva joone tegemiseks tuleks punktid panna üsna tihedalt, kuid seda on käsitsi teha üsna aeganõudev. Seega mõne tuntud sujuva joone tegemiseks kasutatakse tavaliselt matemaatiliste funktsioonide abi punktipaaride konstrueerimisel. Proovime joonistada näiteks laineid sinusoidi abil:

```
import pygame, sys, math

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

punktid = [] # def tühja listi

for x in range(0, 640): # võtame ükshaaval kõik x-telje koordinaadid
    y = int(math.sin(x/640.0*4*math.pi)*100 + 250)
    # y-koordinaadi arvutame siinusfunktsiooni abil, et laine tuleks
    #piisavalt suur, korrutame ja liidame arve juurde.
    punktid.append((x,y)) #kokku pannakse siia listi 640 punktipaari

pygame.draw.lines(lava, [0, 0, 0], False, punktid, 1)
# kuna me ei taha, et punktide jada alguspunkt ja lõpppunkt
#ühendataks, siis kasutame False

pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

## Tekst

Teksti ekraanile trükkimiseks ei pea tegema mitte midagi üleloomulikku. Kui oled ka ussimängu eelnevalt täpselt uurinud, siis ei tohiks siin olla mitte midagi uut. Teksti saab ekraanile nii:

```
# -*- coding: utf-8 -*-

from pygame import *

init()
aken = display.set_mode([640,480])

aken.blit(font.Font(None,70).render('tekst 1 asub (0,0)',1,[255,0,0]),(0,0))
# pannakse teskti pilt vaikimisi fondiga (None), suurusega 70
# asukohta 0x,0y
#käsk render teeb jutumärkide vahel oleva teksti pildiks,
# 1 tähendab, et tähtedel on veidi ümarad servad ja [255, 0, 0] on punane värv

# sama asi, aga veidi rohkem lahti kirjutatud ja muutujaid kasutades:
teksti_font = font.Font(None,50)
tekst2_pildina = teksti_font.render('tekst 2 asub (100,400)',1,[0,255,0])
aken.blit(tekst2_pildina,(100,400))

tekst3_pildina = teksti_font.render('tekst 3 asub keskel',1,[0,0,255])
#paneb pildi akna keskele
tekst3_raam = tekst3_pildina.get_rect()
tekst3_raam.center = aken.get_rect().center
aken.blit(tekst3_pildina,tekst3_raam)

display.flip()
while not event.get(QUIT):
 time.delay(10)

quit()
```

Erinevaid fonte, mida saab sõna None asemel koodis kasutada saad sa näha käsuga **pygame.font.get_fonts().** Kirjuta see käsk käsureale või print() käsuga koodi, siis näed erinevate fontide nimede listi.

## Pildid

Piltide lisamine ekraanile on väga lihtne, vaid kaks rida koodi. Esiteks tuleb tekitada muutuja, millele pilt omistatakse ning seejärel tuleb tekitatud muutuja väärtus ekraanile kuvada.


Proovin selle linnu siin oma tulevase mängu aknasse tekitada:

```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

lind = pygame.image.load("sinine_lind.png")
lava.blit(lind, (50,50))
#laen pildi üles
#pilt peab olema samas kataloogis, kus programmgi
#panen linnu "lavale", koordinaatidega 50x 50y

pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

# Paneme pildi liikuma!

Pildi liigutamine arvutis ei tähenda mitte midagi muud kui pildi uude kohta joonistamist ja vanalt kohalt eelmise pildi ära "kustutamist". Kustutamine aga tähendab arvuti keeles üle värvimist, nii et tekib tunne, et pilt on kustutatud. Kui taust on ühtlase värviga, siis on "kustutamine" kerge, lihtsalt värvime vana pildi tausta värvi ristkülikuga üle. Keeruliseks läheb aga siis, kui taustaks on meil näiteks mõni keeruline maastik või lausa foto, kuidas siis "kustutustöid" teha?

Alustame aga lihtsamast variandist, kus taust on näiteks valge ja paneme siis objekti ekraanil liikuma.


```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

lind = pygame.image.load("sinine_lind.png")
lava.blit(lind, (50,50))
pygame.display.flip()

#liigutame esimene kord
pygame.time.delay(1000)
lava.blit(lind, (150,50))
pygame.draw.rect(lava, [255, 255, 255], [50, 50, 50, 50], 0)
pygame.display.flip()

#liigutame teine kord
pygame.time.delay(1000)
lava.blit(lind, (250,50))
pygame.draw.rect(lava, [255, 255, 255], [150, 50, 50, 50], 0)
pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

## Sujuvam animatsioon

Eelmise leheküljе näites liikus meie lind jonks-jonks ühest punktist teise. Kes enne näinud on, et linnud nii liiguvad? Sujuvuse saavutamiseks kasutatakse sama meetodit nagu joonte ühendamise korral. Punktide arv, kuhu pilt joonistatakse on palju tihedam. Kuna aga sama protseduuri - joonista ja värvi eelmine pilt üle, tehakse kümneid kui mitte sadu kordi, siis võetakse lisaks kasutusele tsükkel.

Uurime linnu lendamist veelkord, kuid nüüd tsükliga:

```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

lind = pygame.image.load("sinine_lind.png")
x, y = 50, 50
# x ja y koordinaat saavad mõlemad algväärtuse 50

lava.blit(lind, (x,y))
pygame.display.flip()

for i in range(100): # liigutame lindu 100x
    pygame.time.delay(20)
    pygame.draw.rect(lava, [255, 255, 255], [x, y, 50, 50], 0)
    x=x+5 # suurendame x-koordinaati 5 võrra
    lava.blit(lind, (x,y))
    pygame.display.flip()


while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

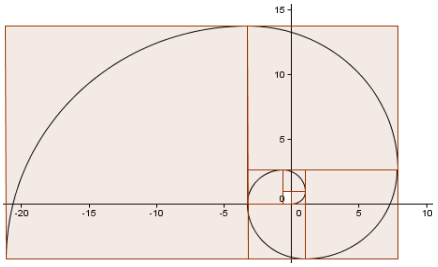
Palju huvitavam oleks ju kui lind ei liiguks sirgjooneliselt vaid hoopis mõne kõverjoone järgi.

Paneme oma linnu liikuma sinusoidi mööda, mille me mõned peatükid tagasi koostasime.

Keera lehte....

## Liikumine mööda kõverjoont

Mööda kõverjoont objektide liikuma panemiseks ei ole vaja midagi muud teha kui valida välja näiteks üks matemaatiline funktsioon, mille järgi objekt peaks ekraanil liikuma. Arvutama selle funktsiooni kaudu piisava tihedusega punktipaarid, unustamata iga kord ühte koordinaatidest suurendada, ja siis tuleb juba pildi kuvamise mehhanismile arvutatud koordinaadid sisse sööta.



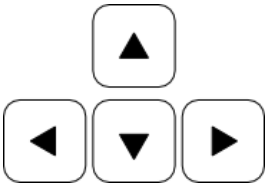

## Paneme linnu lendama mööda sinusoidi:

```
import pygame, sys, math

pygame.init()
lava = pygame.display.set_mode([640, 480])
lava.fill([255, 255, 255])

lind = pygame.image.load("sinine_lind.png")
x, y = 0, 240
# x ja y koordinaat saavad mõlemad algväärtuse 50

lava.blit(lind, (x,y))
pygame.display.flip()

for i in range(110): # liigutame lindu 100x
    pygame.time.delay(40)
    pygame.draw.rect(lava, [255, 255, 255], [x,y, 50, 50], 0)
    x=x+5
    y = int(math.sin(x/640.0*4*math.pi)*100 + 250)
    lava.blit(lind, (x,y))
    pygame.display.flip()

while True:
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
```

## Klaviatuurilt juhtimine

Praeguseks osakme objekti liigutada nii mööda sirgjoont kui mööda kõverjoont, kuid tahaks ikkagi saada kontrolli ka endale. Kuidas ise liigutada objekti ekraanil?

Selleks kasutame ära pygame sündmuste toimumise kontrollijat **pygame.event.get()** ja küsime tema käest, millist klahvi vajutati (**pygame.KEYDOWN**) ning vastavalt millisele nooleklahvile (**K_UP**, **K_DOWN**, **K_LEFT**, **K_RIGHT**) vajutati, muudame koordinaate pildi joonistamiseks.



```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640, 480])

lind = pygame.image.load("sinine_lind.png")
x, y, samm = 0, 240, 10
# x ja y koordinaat ja samm saavad algväärtuse

while True:
    lava.fill([255, 255, 255])
    lava.blit(lind, (x,y))
    pygame.display.flip()
    pygame.time.delay(40)
    for i in pygame.event.get():
        if i.type ==pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_UP:
                y = y - samm
            elif i.key == pygame.K_DOWN:
                y = y + samm
            elif i.key == pygame.K_LEFT:
                x = x - samm
            elif i.key == pygame.K_RIGHT:
                x = x + samm
```

Lisaks nooleklahvidele saab loomulikult sama moodi kontrollida mistahes klahvi vajutust. Kõikide klahvide vastavad käsud leiab **pygame.KEYDOWN dokumentatsioonist**:

http://www.pygame.org/docs/ref/key.html

Koodi käima pannes ja klahvide tööd katsetades paned kindlasti tähele, et kuitahes kaua ka ei hoiaks nooleklahvi all, siis liigutab antud programm ikka vaid ühe korra linnukest. Selleks, et objekt liiguks ka siis, kui ma klahvi all hoian, tuleb **while** tsükli ette kirjutada veel kolm rida:

```
viivitus = 100
intervall = 50
pygame.key.set_repeat(viivitus, intervall)
```

## Sündmused, mida saab kontrollida

Eelmisel lehel vaatasime põhjalikumalt, kuidas enda kontrolli alla võtta klahvivajutused ja kuidas vastavalt vajutusele saab koodis midagi muuta. Tegelikult on Pygames palju selliseid sündmuseid mida saab enda kontrolli alla võtta ja oma soovide järgi kohandada. Kõige tuntumad nendest on

- KEYDOWN - vaatasime eelmisel lehel
- KEYUP
- QUIT
- MOUSEMOTION
- MOUSEBUTTONUP
- MOUSEBUTTONDOWN
- jne

Kogu nimekirja kõikvõimalikest kontrollsündmustest, näiteks kas või mängukonsoolidega ühinemise jaoks või videote kasutamiseks, leiab aadressilt: http://www.pygame.org/docs/ref/event.html

Kõikide nende nn **pygame.event**'te kasutamine käib täpselt sama moodi nagu vaatasime KEYDOWN sündmuste kontrolli juures.

## Hiirega liigutamine

Klaviatuurilt objektide liigutamine on lihtne. Kuulad muudkui KEYDOWN'i käskusid ja kirjutad selle järele if-lausete abil, mis siis juhtuma peaks. Analoogse süsteemi järgi toimub ka hiirega objektide juhtimine ekraanil. Pidevalt tuleb kuulata **MOUSEMOTION**'it ja vastavalt hiire liikumisele kirjutada oma koodi lõik, mis ütleb, mida siis tegema peab kui hiir liigub.

Järgnevalt vaatame kahte näidet hiirega objektide liigutamise kohta. Esimeses näites liigub objekt alati kaasa kui hiir liigub ja teises vaid siis, kui hiirega objektist nö kinni hoida.

Esimene variant on kasulik näiteks ussimängu analoogia programmeerimisel, kui tahta ussi juhtida hiire kaudu, teine on lihtsalt igas muus olukorras kasulik:)


## 1. näide: objekt liigub siis kui hiir liigub:

```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640,480])
lind = pygame.image.load('sinine_lind.png')

lava.fill([255,255,255])
lava.blit(lind,(0,0))
pygame.display.flip()

while True:
        for i in pygame.event.get():
                if i.type == pygame.QUIT:
                        sys.exit()
                elif i.type == pygame.MOUSEMOTION:
                        lava.fill([255,255,255])
                        lava.blit(lind,i.pos)
                        pygame.display.flip()
```

## 2. näide: objekt liigub siis kui hiirega temast kinni võetakse:

```
import pygame, sys

pygame.init()
lava = pygame.display.set_mode([640,480])
lind = pygame.image.load('sinine_lind.png')

x,y = 0,0 #linnu algsed koordinaadid
lava.fill([255,255,255])
lava.blit(lind,(x,y))
pygame.display.flip()
lohista = False

while True:
        for i in pygame.event.get():
                if i.type == pygame.QUIT:
                        sys.exit()
                elif i.type == pygame.MOUSEBUTTONDOWN:
                        mx,my = i.pos
                        if abs(mx-x)<50 and abs(my-y)<50:
                                lohista = True
                        # nende kahe reaga kontrollitakse
                        #kas hiir on linnu objekti peal või mitte
                        # kui on, antakse lohistamiseks luba
                        # mõtle, miks on see nii kirja pandud
                        # proovi paberil jooistada üks näide

                elif i.type == pygame.MOUSEBUTTONUP:
                        lohista = False
                elif lohista and i.type == pygame.MOUSEMOTION:
                        x,y = i.pos # annab hiire koordinaadid
                        lava.fill([255,255,255])
                        lava.blit(lind,(x,y))
                        pygame.display.flip()
                # selles elif-s on luba lohistamiseks, kontrollitakse
                # kas ka hiir liigub ja vastavalt hiire asukohale
                # uuendatakse pidevalt ekraani pilti
```

## Ussimängu finaal

Selle nädalaga saame ussimängu lõpuks valmis. Kindlasti võiks lõpptulemusse veel palju lisavidinaid juurde panna, kuid kas see on eesmärk omaette?

Aga alusta - vaata ja uuri, mis siis seekord on koodis muutunud ja katseta seda ka oma arvutis.

```
# -*- coding: utf-8 -*-
#(teavitame, et kasutame selles Pythoni failis
#8-bitilist Unicode kodeeringut - vajalik täpitähtede jaoks)

# Muutused võrreldes eelmise versiooniga:
# Akna nurgas näidatakse ära söödud hiirte arvu.

# loeme sisse kasutatavad moodulid:
from pygame import *
from random import *

init()    # paneme PyGame'i tööle
mixer.init()    # paneme hääle tekitamise tööle
aken = display.set_mode([640,480])    # loome PyGame'i akna

# loeme failidest sisse kasutatavad pildid ja leiame
#nende piltide nähtamatud raamid:
pilt_ussipea = image.load('ussipea.png')
pilt_ussipea_raam = pilt_ussipea.get_rect()
pilt_kokkuporge = image.load('kokkuporge.png')
pilt_kokkuporge_raam = pilt_kokkuporge.get_rect()
pilt_hiir = image.load('hiir.png')
pilt_hiir_raam = pilt_hiir.get_rect()
pilt_soodud = image.load('soodud.png')
pilt_soodud_raam = pilt_soodud.get_rect()

# loeme sisse helifailid:
heli_soodud = mixer.Sound('soodud.wav')
heli_kokkuporge = mixer.Sound('kokkuporge.wav')

# need muutujad määravad joonistamise stiili:
varv_taust = (255,255,255)    # valge
varv_skoor = (0,0,0)    # must
# määrame skoori kirjutamiseks vaikimisi fondi suurusega 70
font_skoor = font.Font(None,70)
asukoht_skoor = (5,5)    # koordinaadid, kuhu skoori hakkame kirjutama

# need muutujad määravad mängu kiiruse
ussisamm = 5    #nii palju pikseleid liigub uss igal sammul
tsykli_viivitus = 20    #nii palju millisekundeid oodata kahe sammu vahel
hiire_viivitus = 1000    #nii palju millisekundeid pärast hiire söömist tekib uus hiir

# edasi algväärtustame mängu olulised muutujad
hiir_soodud = False    # kas hiir on ära söödud, algul mitte
mitu_soodud = 0    # selle muutujaga loeme skoori, algul 0
(hiir_x,hiir_y) = (300,300)    # hiire asukoha algväärtus
(ussipea_x,ussipea_y) = (50,50)    # ussi algne asukoht
suund = K_RIGHT    # ussi algne suund
```

```
# funktsioon, mis joonistab hetkeseisu ekraanile:
def joonista():
        aken.fill(varv_taust)   # värvime akna taustavärviga üle
        if hiir_soodud:
                pilt_soodud_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_soodud,pilt_soodud_raam)
        else:
                pilt_hiir_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_hiir,pilt_hiir_raam)
        # paneme pildiraami keskpunkti nendele koordinaatidele
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)
        aken.blit(pilt_ussipea,pilt_ussipea_raam)   # paneme pildi raami sisse
        skoor = font_skoor.render(str(mitu_soodud),1,varv_skoor)   # teeme skoorist pildi
        aken.blit(skoor,asukoht_skoor)   # ja joonistame selle pildi
# siiani joonistasime nähtamatusse aknasse, nüüd kopeerime saadud tulemuse nähtavaks
        display.flip()

# funktsioon, mis liigutab ussi ühe ussisammu võrra:
def liiguta_ussi():
        global ussipea_x,ussipea_y
 # tahame muuta globaalseid muutujaid, et fun. sees tehtavad muutused väljaspool fun. kajastuks
        if suund == K_UP:
                ussipea_y = ussipea_y - ussisamm
        elif suund == K_DOWN:
                ussipea_y = ussipea_y + ussisamm
        elif suund == K_LEFT:
                ussipea_x = ussipea_x - ussisamm
        elif suund == K_RIGHT:
                ussipea_x = ussipea_x + ussisamm

# funktsioon, mis kontrollib, kas uss on kokku põrganud seinaga
def kas_on_kokkuporge():
        # paneme ussipea pildiraami õige koha peale
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)
        # tagastab True, kui ussi pea pildi raam ei mahu enam akna sisse, vastasel juhul False
        return not aken.get_rect().contains(pilt_ussipea_raam)

while True: # lõpmatu tsükkel
        joonista()
        time.delay(tsykli_viivitus)
        liiguta_ussi()
        if kas_on_kokkuporge():
                break # katkestame tsükli
        # kui hiir pole söödud ja ussipea on ülekattes hiirega
        if not hiir_soodud and pilt_ussipea_raam.colliderect(pilt_hiir_raam):
                hiir_soodud = True
                mitu_soodud = mitu_soodud + 1
                heli_soodud.play()
                # käivitame taimeri, uus hiir tekib pärast viivitust
                time.set_timer(USEREVENT,hiire_viivitus)
        for e in event.get(): # vaatame üle kõik vahepeal toimunud sündmused
                if e.type == KEYDOWN: # kasutaja vajutas nuppu
                        if e.key in (K_UP,K_DOWN,K_LEFT,K_RIGHT): # kui vajutati nooleklahvi
                                suund = e.key   # uss peab roomama sinna suunas
                # taimeri aeg otsas, aeg tekitada uus hiir juhuslikku kohta aknas
                if e.type == USEREVENT:
                        hiir_x = randint(aken.get_rect().left,aken.get_rect().right)
                        hiir_y = randint(aken.get_rect().top,aken.get_rect().bottom)
                        hiir_soodud = False   # uus hiir, seda pole veel söödud
                        time.set_timer(USEREVENT,0)   # paneme stopperi kinni
                if e.type == QUIT: # kasutaja sulges akna, lõpetame
                        exit()

# kui programmi täitmine siia jõuab, siis järelikult oli kokkupõrge
heli_kokkuporge.play()
pilt_kokkuporge_raam.center = (ussipea_x,ussipea_y)
aken.blit(pilt_kokkuporge,pilt_kokkuporge_raam)
display.flip()
while not event.get(QUIT):   # kuni kasutaja pole sulgenud akent
        pass   # ei tee midagi
```

## Vol8

```
# -*- coding: utf-8 -*-
#(teavitame, et kasutame selles Pythoni failis
#8-bitilist Unicode kodeeringut - vajalik täpitähtede jaoks)

# Muutused võrreldes eelmise versiooniga:
# Ussile lisatud saba, mis pikeneb iga hiire ära söömisel.

# loeme sisse kasutatavad moodulid:
from pygame import *
from random import *

init()    # paneme PyGame'i tööle
mixer.init()    # paneme hääle tekitamise tööle
aken = display.set_mode([640,480])    # loome PyGame'i akna

# loeme failidest sisse kasutatavad pildid ja leiame
#nende piltide nähtamatud raamid:
pilt_ussipea = image.load('ussipea.png')
pilt_ussipea_raam = pilt_ussipea.get_rect()
pilt_kokkuporge = image.load('kokkuporge.png')
pilt_kokkuporge_raam = pilt_kokkuporge.get_rect()
pilt_hiir = image.load('hiir.png')
pilt_hiir_raam = pilt_hiir.get_rect()
pilt_soodud = image.load('soodud.png')
pilt_soodud_raam = pilt_soodud.get_rect()

# loeme sisse helifailid:
heli_soodud = mixer.Sound('soodud.wav')
heli_kokkuporge = mixer.Sound('kokkuporge.wav')

# need muutujad määravad joonistamise stiili:
varv_taust = (255,255,255)    # valge
varv_saba = (108,171,36)    # ussipea pildis esinev roheline (RGB-kood)
varv_skoor = (0,0,0)    # must
# määrame skoori kirjutamiseks vaikimisi fondi suurusega 70
font_skoor = font.Font(None,70)
asukoht_skoor = (5,5)    # koordinaadid, kuhu skoori hakkame kirjutama
ussi_paksus = 30    # ussi paksus (pikselites)

# need muutujad määravad mängu kiiruse
ussisamm = 5    # nii palju pikseleid liigub uss igal sammul
# saba kasvab hiire söömisel kasvamise_kiirus*ussisamm pikseli võrra
kasvamise_kiirus = 15
# nii palju millisekundeid oodata kahe sammu vahel
tsykli_viivitus = 20
# nii palju millisekundeid pärast hiire söömist tekib uus hiir
hiire_viivitus = 1000

# edasi algväärtustame mängu olulised muutujad
saba_list = []    # saba punktide nimekiri, algul tühi
hiir_soodud = False    # kas hiir on ära söödud, algul mitte
mitu_soodud = 0    # selle muutujaga loeme skoori, algul 0
(hiir_x,hiir_y) = (300,300)    # hiire asukoha algväärtus
(ussipea_x,ussipea_y) = (50,50)    # ussi algne asukoht
suund = K_RIGHT    # ussi algne suund
```

```
# funktsioon, mis joonistab hetkeseisu ekraanile:
def joonista():
        aken.fill(varv_taust)   # värvime akna taustavärviga üle
        if hiir_soodud:
                pilt_soodud_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_soodud,pilt_soodud_raam)
        else:
                pilt_hiir_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_hiir,pilt_hiir_raam)
        # joonistame iga sabatüki koha peale värvilise palli
        for sabatyki_asukoht in saba_list:
                draw.circle(aken,varv_saba,sabatyki_asukoht,ussi_paksus//2)
        # paneme pildiraami keskpunkti nendele koordinaatidele
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)
        aken.blit(pilt_ussipea,pilt_ussipea_raam)   # paneme pildi raami sisse
        skoor = font_skoor.render(str(mitu_soodud),1,varv_skoor)   #teeme skoorist pildi
        aken.blit(skoor,asukoht_skoor)   # ja joonistame selle pildi
 # siiani joonistasime nähtamatusse aknasse, nüüd kopeerime saadud tulemuse nähtavaks
        display.flip()

# funktsioon, mis liigutab ussi ühe ussisammu võrra:
def liiguta_ussi():
    # vajalik selleks, et funktsiooni sees tehtavad muutused väljaspool funktsiooni kajastuks
        global saba_list,ussipea_x,ussipea_y
        saba_list = [(ussipea_x,ussipea_y)] + saba_list   #lisame ussi pea asukoha saba etteotsa
        saba_list.pop()   # ja kustutame saba viimase otsa
        if suund == K_UP:
                ussipea_y = ussipea_y - ussisamm
        elif suund == K_DOWN:
                ussipea_y = ussipea_y + ussisamm
        elif suund == K_LEFT:
                ussipea_x = ussipea_x - ussisamm
        elif suund == K_RIGHT:
                ussipea_x = ussipea_x + ussisamm

# funktsioon, mis kontrollib, kas uss on kokku põrganud seina või oma sabaga:
def kas_on_kokkuporge():
        for sabatyki_asukoht in saba_list:   # kontrollime eraldi iga saba punkti jaoks
                # teeme ussisammu suuruse ruudukujulise raami
                sabatyki_raam = Rect(0,0,ussisamm,ussisamm)
                # paneme raami vaadeldava saba punkti ümber
                sabatyki_raam.center = sabatyki_asukoht
                # kontrollime, ega pea keskpunkt ei kattu selle raamiga
                if sabatyki_raam.collidepoint(ussipea_x,ussipea_y):
                        return True   # kui kattub, siis on kokkupõrge
        # paneme ussipea pildiraami õige koha peale
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)
# tagastab True, kui ussi pea pildi raam ei mahu enam akna sisse, vastasel juhul False
        return not aken.get_rect().contains(pilt_ussipea_raam)

while True: # lõpmatu tsükkel
        joonista()
        time.delay(tsykli_viivitus)
        liiguta_ussi()
        if kas_on_kokkuporge():
                break # katkestame tsükli
```

```
            # kui hiir pole söödud ja ussipea on ülekattes hiirega
        if not hiir_soodud and pilt_ussipea_raam.colliderect(pilt_hiir_raam):
                hiir_soodud = True
                mitu_soodud = mitu_soodud + 1
# paneme saba punktide hulka ussipea kohale nii palju punkte juurde, kui määrab kasvamise kiirus
                saba_list = [(ussipea_x,ussipea_y)]*kasvamise_kiirus + saba_list
                heli_soodud.play()
                # käivitame taimeri, uus hiir tekib pärast viivitust
                time.set_timer(USEREVENT,hiire_viivitus)
        for e in event.get(): # vaatame üle kõik vahepeal toimunud sündmused
                if e.type == KEYDOWN: # kasutaja vajutas nuppu
                        if e.key in (K_UP,K_DOWN,K_LEFT,K_RIGHT): # kui vajutati nooleklahvi
                                suund = e.key   # uss peab roomama sinna suunas
                # taimeri aeg otsas, aeg tekitada uus hiir juhuslikku kohta aknas
                if e.type == USEREVENT:
                        hiir_x = randint(aken.get_rect().left,aken.get_rect().right)
                        hiir_y = randint(aken.get_rect().top,aken.get_rect().bottom)
                        hiir_soodud = False   # uus hiir, seda pole veel söödud
                        time.set_timer(USEREVENT,0)   # paneme stopperi kinni
                if e.type == QUIT: # kasutaja sulges akna, lõpetame
                        exit()

# kui programmi täitmine siia jõuab, siis järelikult oli kokkupõrge
heli_kokkuporge.play()
pilt_kokkuporge_raam.center = (ussipea_x,ussipea_y)
aken.blit(pilt_kokkuporge,pilt_kokkuporge_raam)
display.flip()
while not event.get(QUIT):   # kuni kasutaja pole sulgenud akent
        pass   # ei tee midagi
```

## Vol9

```
# -*- coding: utf-8 -*-
#(teavitame, et kasutame selles Pythoni failis
#8-bitilist Unicode kodeeringut - vajalik täpitähtede jaoks)

# Muutused võrreldes eelmise versiooniga:
# Ussi saba on pea juures paksem ja saba otsas peenem.

# loeme sisse kasutatavad moodulid:
from pygame import *
from random import *

init()   # paneme PyGame'i tööle
mixer.init()   # paneme hääle tekitamise tööle
aken = display.set_mode([640,480])   # loome PyGame'i akna

# loeme failidest sisse kasutatavad pildid ja leiame
#nende piltide nähtamatud raamid:
pilt_ussipea = image.load('ussipea.png')
pilt_ussipea_raam = pilt_ussipea.get_rect()
pilt_kokkuporge = image.load('kokkuporge.png')
pilt_kokkuporge_raam = pilt_kokkuporge.get_rect()
pilt_hiir = image.load('hiir.png')
pilt_hiir_raam = pilt_hiir.get_rect()
pilt_soodud = image.load('soodud.png')
pilt_soodud_raam = pilt_soodud.get_rect()

# loeme sisse helifailid:
heli_soodud = mixer.Sound('soodud.wav')
heli_kokkuporge = mixer.Sound('kokkuporge.wav')

# need muutujad määravad joonistamise stiili:
varv_taust = (255,255,255)   # valge
# ussipea pildis esinev roheline (RGB-kood)
varv_saba = (108,171,36)
varv_saba_rongad = (0,0,0)   # must
varv_skoor = (0,0,0)   # must
# määrame skoori kirjutamiseks vaikimisi fondi suurusega 70
font_skoor = font.Font(None,70)
asukoht_skoor = (5,5)   # koordinaadid, kuhu skoori hakkame kirjutama
paksus_pea_juures = 30   # ussi paksus pea juures (pikselites)
paksus_saba_otsas = 10   # ussi paksus saba otsas (pikselites)

# need muutujad määravad mängu kiiruse
ussisamm = 5   # nii palju pikseleid liigub uss igal sammul
# saba kasvab hiire söömisel kasvamise_kiirus*ussisamm pikseli võrra
kasvamise_kiirus = 15
# nii palju millisekundeid oodata kahe sammu vahel
tsykli_viivitus = 20
# nii palju millisekundeid pärast hiire söömist tekib uus hiir
hiire_viivitus = 1000

# edasi algväärtustame mängu olulised muutujad
saba_list = []   # saba punktide nimekiri, algul tühi
hiir_soodud = False   # kas hiir on ära söödud, algul mitte
mitu_soodud = 0   # selle muutujaga loeme skoori, algul 0
(hiir_x,hiir_y) = (300,300)   # hiire asukoha algväärtus
(ussipea_x,ussipea_y) = (50,50)   # ussi algne asukoht
```

```
suund = K_RIGHT   # ussi algne suund

# funktsioon, mis joonistab hetkeseisu ekraanile:
def joonista():
        aken.fill(varv_taust)   # värvime akna taustavärviga üle
        if hiir_soodud:
                pilt_soodud_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_soodud,pilt_soodud_raam)
        else:
                pilt_hiir_raam.center = (hiir_x,hiir_y)
                aken.blit(pilt_hiir,pilt_hiir_raam)
        for i in range(len(saba_list)):
            # kaalutud keskmine paksusest pea juures ja saba otsas, kaalud on määratud asukohaga saba listis
                paksus = ( i*paksus_saba_otsas + (len(saba_list)-i)*paksus_pea_juures ) // len(saba_list)
                draw.circle(aken,varv_saba,saba_list[i],paksus//2)   # värviline pall
                draw.circle(aken,varv_saba_rongad,saba_list[i],paksus//2,1)   # rõngas palli ümber
        # paneme pildiraami keskpunkti nendele koordinaatidele
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)
        aken.blit(pilt_ussipea,pilt_ussipea_raam)   # paneme pildi raami sisse
        skoor = font_skoor.render(str(mitu_soodud),1,varv_skoor)   # teeme skoorist pildi
        aken.blit(skoor,asukoht_skoor)   # ja joonistame selle pildi
        # siiani joonistasime nähtamatusse aknasse, nüüd kopeerime saadud tulemuse nähtavaks
        display.flip()

# funktsioon, mis liigutab ussi ühe ussisammu võrra:
def liiguta_ussi():
    # vajalik selleks, et funktsiooni sees tehtavad muutused väljaspool funktsiooni kajastuks
        global saba_list,ussipea_x,ussipea_y
        saba_list = [(ussipea_x,ussipea_y)] + saba_list   # lisame ussi pea asukoha saba etteotsa
        saba_list.pop()   # ja kustutame saba viimase otsa
        if suund == K_UP:
                ussipea_y = ussipea_y - ussisamm
        elif suund == K_DOWN:
                ussipea_y = ussipea_y + ussisamm
        elif suund == K_LEFT:
                ussipea_x = ussipea_x - ussisamm
        elif suund == K_RIGHT:
                ussipea_x = ussipea_x + ussisamm

# funktsioon, mis kontrollib, kas uss on kokku põrganud seina või oma sabaga:
def kas_on_kokkuporge():
        for sabatyki_asukoht in saba_list:   # kontrollime eraldi iga saba punkti jaoks
                # teeme ussisammu suuruse ruudukujulise raami
                sabatyki_raam = Rect(0,0,ussisamm,ussisamm)
                # paneme raami vaadeldava saba punkti ümber
                sabatyki_raam.center = sabatyki_asukoht
                # kontrollime, ega pea keskpunkt ei kattu selle raamiga
                if sabatyki_raam.collidepoint(ussipea_x,ussipea_y):
                        return True   # kui kattub, siis on kokkupõrge
        pilt_ussipea_raam.center = (ussipea_x,ussipea_y)   # paneme ussipea pildiraami õige koha peale
        # tagastab True, kui ussi pea pildi raam ei mahu enam akna sisse, vastasel juhul False
        return not aken.get_rect().contains(pilt_ussipea_raam)

while True: # lõpmatu tsükkel
        joonista()
        time.delay(tsykli viivitus)
```

```
        liiguta_ussi()
        if kas_on_kokkuporge():
                break # katkestame tsükli
        # kui hiir pole söödud ja ussipea on ülekattes hiirega
        if not hiir_soodud and pilt_ussipea_raam.colliderect(pilt_hiir_raam):
                hiir_soodud = True
                mitu_soodud = mitu_soodud + 1
                # paneme saba punktide hulka ussipea kohale nii palju punkte juurde, kui määrab kasvamise kiirus
                saba_list = [(ussipea_x,ussipea_y)]*kasvamise_kiirus + saba_list
                heli_soodud.play()
                time.set_timer(USEREVENT,hiire_viivitus) # käivitame taimeri, uus hiir tekib pärast viivitust
        for e in event.get(): # vaatame üle kõik vahepeal toimunud sündmused
                if e.type == KEYDOWN: # kasutaja vajutas nuppu
                        if e.key in (K_UP,K_DOWN,K_LEFT,K_RIGHT): # kui vajutati nooleklahvi
                                suund = e.key   # uss peab roomama sinna suunas
                if e.type == USEREVENT: # taimeri aeg otsas, aeg tekitada uus hiir juhuslikku kohta aknas
                        hiir_x = randint(aken.get_rect().left,aken.get_rect().right)
                        hiir_y = randint(aken.get_rect().top,aken.get_rect().bottom)
                        hiir_soodud = False   # uus hiir, seda pole veel söödud
                        time.set_timer(USEREVENT,0)   # paneme stopperi kinni
                if e.type == QUIT: # kasutaja sulges akna, lõpetame
                        exit()

# kui programmi täitmine siia jõuab, siis järelikult oli kokkupõrge
heli_kokkuporge.play()
pilt_kokkuporge_raam.center = (ussipea_x,ussipea_y)
aken.blit(pilt_kokkuporge,pilt_kokkuporge_raam)
display.flip()
while not event.get(QUIT):   # kuni kasutaja pole sulgenud akent
    time.delay(100) # aitab vähendada idle ja pygame vahelist konflikti
quit()
```

## Mida õppisid?

Jälle on möödunud üks tegus ja teadmiste rohke nädal. Lisandunud on palju erinevaid koodilõike, mida saab oma tulevastes mängudes ja programmides ära kasutada. Sellel nädalal õppisid:

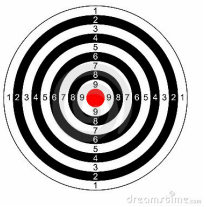
- Pygame kasutama
- kuidas luua oma mängu akent?
- kuidas sellesse aknasse joonistada ja lsiada kujundeid?
- mis asjad on kujundite raamid?
- kuidas kasutatakse värve Pythonis?
- tegema kunsti
- kuidas vabakäe joont teha arvutis?
- kuidas panna erinevate kujundite asukoht üksteisest sõltuma?
- kuidas lisada pilte ekraanile?
- kuidas neid pilte või kujundeid liikuma panna?
- kuidas uuendatakse ekraani pilti?
- kuidas objektid panna liikuma mööda kõverjoont?
- mis asjad on flip() ja blit()?
- kuidas juhtida objekte ekraanil klaviatuuri abil?
- mis asi on sündmuste kontrollija ja kuidas seda ära kasutada?
- kuidas panna objektid liikuma hiire abil?
- said ülevaate teistest kontrollitavatest sündmustest Pygame's, mida oma programmids ära kasutada

## TEE ISE!


**1. ülesanne: Maja.** Et harjutada, erinevate kujundite ja koordinaatide arvutamist pygamega, siis tuleb esimese üelsandena sul kirjutada programm, mis joonistab pygamega maja. Majal peab olema kolmnurkne (või trapetsi kujuline) katus, uks, kandiline aken ja ümmargune aken. Taust peab olema midagi muud kui valge.

**2. ülesanne: Kandiline kunst.** Kujundite raamide peatüki all on programm "Mullid". Muuda antud programmi koodi nii, et tekiks programm "Kastid". Kuna kood on näites kommenteerimata, siis sinu ülesanne on ka see lünk täita. Antud juhul ootan ma kommentaare pea igale reale.

**3. ülesanne: Kõverjoon.** Pane üks enda valitud pilt, kujund või objekt liikuma mööda kõverjoont. Soovitav oleks kasutada mõnda matemaatilist funktsiooni, mille abil sa saaksid koordinaate välja arvutada. Kel huvi põnevamate joonte ja nende valemite järele, siis need leiad siit: http://www.art.tartu.ee/~illi/kunstigeomeetria/koverad/joonpind.htm, sama asi inglise keeles: http://turnbull.mcs.st-and.ac.uk/history/Curves/Curves.html



**4. ülesanne**: **Märklaud.** Kirjuta Pygame programm, mis joonistab aknasse märklaua (Kujuta endale ette Noolemängu). Kõige keskel on punane ring, siis edasi veidi suurem ring 9 punkti jaoks, siis veel veidi suurem ring 8 punkti jaoks jne. Kindlasti tuleks selle ülesande lahendamisel kasutada tsüklit, mis korraldab igal tiirul (10 tiiru) ette antud sammu võrra suurema ringi ja õiget värvi. Märklaud peab asuma täpselt loodud akna keskel. Kes soovib, siis võiks ka numbrid märkaluale kirjutada.

**5. ülesanne**: **Lõputöö I osa.** Alusta oma lõputöö programmeerimisega. Nädala lõpuks lae üles oma poolik töö koos kommentaaridega, mida see lõik juba teha oskab ja kui palju on veel jäänud teha. Ära unusta, et lõputöö peab valmis olema juba **29. mai**. See on 7. nädala teine päev!