Eesti Teadusagentuur
Estonian Research Council
TeaMe
Euroopa Liit
Euroopa Sotsiaalfond
Eesti tuleviku heaks
TTÜ
informaatikainstituut

Rakendused

Rakendus (*Application*) on tarkvara kogum antud liiki ülesannete lahendamiseks või tegevuste täitmiseks arvutil. Samas tähenduses kasutatakse ka mõisteid programm, rakendusprogramm, programmipakett.

Sisu

Rakenduse olemus

Rakendus (*Application*) on tarkvara kogum antud liiki ülesannete lahendamiseks või tegevuste täitmiseks arvutil. Samas tähenduses kasutatakse ka lähedasi mõisteid: programm, rakendusprogramm, programmipakett.

Eristada võib rakenduste kahte põhiliiki:

- üldotstarbeline rakendustarkvara (rakendusprogrammid): veebilehitsejad, tekstitöötluse-, tabeli-, graafikaprogrammid, …
- spetsialiseeritud rakendused (erineva otstarbe ja mahuga, realiseeritakse erinevate vahenditega ja erinevates keskkondades): võrrandite lahendamine, palgaarvestus, detailide projekteerimine, mängud, …

Peamised tegevused rakenduste loomisel:

- modelleerimine
- analüüs
- projekteerimine (disain)
- algoritmimine
- programmeerimine (kodeerimine)
- testimine
- dokumenteerimine

Rakenduse põhikomponendid

Rakenduses on üldjuhul omavahel tihedalt seotud:

- programmid
- andmed ja objektid
- kasutajaliides

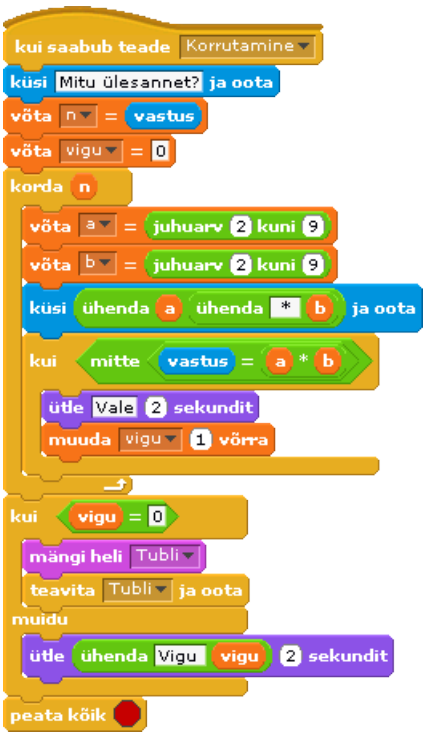
Programm

Programm on käskude (korralduste, instruktsioonide, **lausete**) kogum, mis määrab, milliseid tegevusi peab arvuti täitma **andmete** ja **objektidega**, tagades tavaliselt ka kasutajaliidese töö.

Programmide koostamiseks kasutatakse spetsiaalseid **programmeerimiskeeli** ja -**süsteeme**. Igas keeles on piiratud valik **käske** ehk **lauseid** ning nende esitamiseks ja täitmiseks on kindlad reeglid. Programm võib koosneda mitmest üksusest. Erinevates keeltes kasutatakse nende jaoks erinevaid nimetusi: **protseduur**, **funktsioon**, **skript**.

Tänapäeva arvutid saavad täita programme, mis on esitatud antud arvutitüübi jaoks ettenähtud **masinkeeles**. See koosneb kahendsüsteemis esitatud elementaarsetest käskudest nagu liitmine, lahutamine, võrdlemine jms. Programmide tõlkimiseks masinkeelde kasutatakse spetsiaalseid programme, mida nimetatakse **translaatoriteks**. On kahte liiki translaatoreid – **kompilaatorid** ja **interpretaatorid**. Kompilaator tõlgib (transleerib) terve programmi enne täitmist masinkeelde. Interpretaator tõlgib programmi täitmise ajal. Millist transleerimise viisi kasutatakse, sõltub keelest ja programmeerimissüsteemist. Mitmed süsteemid võimaldavad mõlemat varianti: rakenduse loomise ajal kasutatakse interpretaatorit, valmisprogramm aga transleeritakse masinkeelde, sest selline programm töötab kiiremini ja ei vaja enam programmeerimissüsteemi, millega ta on loodud. Translaator on programmeerimissüsteemi üheks peamiseks osaks. Lisaks translaatorile sisaldab see tavaliselt spetsiaalset **redaktorit**, mille abil sisestatakse ja redigeeritakse programme, vahendeid kasutajaliideste loomiseks, silumise ja testimise vahendeid jms.

Järgnevas näites on toodud sarnase sisuga programmid kolmes programmeerimiskeeles: Scratch, Visual Basic ja Python. Programmid esitavad kasutaja poolt etteantud arvu (**n**) korrutamisülesandeid, hindavad vastuseid ja annavad üldhinnangu tulemustele.




```
Sub Korrutamine()
 Dim a, b, n, vastus, vigu, k
 n = InputBox("Mitu
ülesannet?")
 vigu = 0
 For k = 1 To n
   a = Int(2 + Rnd() * 8)
   b = Int(2 + Rnd() * 8)
   vastus = InputBox(a &" * "&b)
   If Int(vastus) <> a * b Then
     MsgBox "Vale!"
     vigu = vigu + 1
   End If
Next k
If vigu = 0 Then
  MsgBox "Tubli! Kõik oli õige!"
  Call Tubli (Shapes("Juku"),3)
 Else
    MsgBox "Vigu oli " & vigu
End If
End Sub
```

```
import random
def Korrutamine() :
  n = int(input("Mitu ?"))
  vigu = 0
  for k in range(n) :
     a = random.randint(2, 9)
     b = random.randint(2, 9)
     t = str(a) + " * " + str(b)
     vastus = input(t + "=>")
     if int(vastus) != a * b :
        print ("Vale!")
        vigu += 1
  if vigu == 0 :
     print ("Kõik on õige!")
  else :
     print ("Vigu oli ", vigu)

Korrutamine()
```


```
Sub Tubli(kuju, n)
  Dim i
  For i = 1 To n
    kuju.IncrementTop -30
    paus 0.3
    kuju.IncrementTop 30
    paus 0.5
  Next i
End Sub
```

Lisaks põhitegevusele on Scratchis ja VBAs kasutusel protseduur, mis realiseerib lihtsa animatsiooni (graafika-objekt teeb mõned hüpped), kui kasutaja on kõik õigesti vastanud.

Programmid on väga tihedalt seotud **andmetega**, sest kasutatavate andmete liigist ja organisatsioonist sõltuvad otseselt programmi sisu ja kasutatavad vahendid. Järgnevalt esitletakse andmeid põhjalikumalt. Programminäited on toodud vaid Scratchis ja Visual Basicus.

Andmed

Andmed on informatsiooni esitus kujul, mis võimaldab seda säilitada, töödelda ja edastada programm-juhtimisega süsteemide (eeskätt arvutite) abil. Kasutatakse erinevat liiki andmeid ja neil võib olla erinev organisatsioon.

Andmete liigid

Peamised andmete liigid on

- märkandmed: arvud, tekstid, …
- graafikaandmed: pildid, joonised, diagrammid, …
- heliandmed: kõne, muusika, …

Liigist sõltub andmete esitus arvuti seadmetes ning võimalikud tehted (operatsioonid) ja tegevused nendega. Kõik andmed esitatakse arvuti mälus kodeerituna – kahendarvudena.

Märkandmeid kasutatakse praktiliselt kõikides rakendustes. Nende esitamiseks arvutis on esmaseks märkhaaval kodeerimine kahendarvude abil: igale märgile (täht, number jms) vastab kindel kahendarv (näiteks A on 01000001, B on 01000010, 3 on 00110011, + on 00101011). Kaheks peamiseks **kodeerimis-süsteemiks** on ASCII ja UNICODE. Arvud, millega tehakse matemaatilisi operatsioone, võivad olla salvestatud spetsiaalsetes vormingutes: **täisarvud, püsikomaarvud** ja **ujukomaarvud** (reaalarvud).

Graafikaandmete loomiseks ja redigeerimiseks saab kasutada erinevaid programme. On olemas terve hulk spetsiaalseid graafika-, pilditöötluse- ja joonestusprogramme (**Paint**, **GIMP**, **Adobe Photoshop**, **AutoCAD**, jm), **mille töö** tulemusi saab salvestada erinevates vormingutes failidesse. Sõltuvalt vormingust on graafikafailidel kindlad tüübitunnused nagu **PNG**, **JPEG**, **GIF**, **PSDb DWG**, …

Graafikaobjekte saab luua ja kasutada ka paljudes üldotstarbelistes rakendusprogrammides (nt tekstitöötluse-, esitluse- ja tabeliprogrammides) või neid importida ning lisada siis dokumenti.

Loodavates rakendustes saab graafikaandmeid importida olemasolevatest graafikafailidest kas rakenduse loomise või täitmise ajal. Graafikaobjektideks on näiteks ka Scratchi spraitide kostüümid ja lava taustad. Mitmed programmeerimissüsteemid võimaldavad luua graafilisi kujutisi ka vahetult täitmise ajal. Scratchis saab näiteks joonistusi teha grupi **Pliiats** ja **Liikumine** käskudega.

Heliandmed salvestatakse samuti erinevates failivormingutes – **WAV**, **MP3**, **MID**, … **H**eliklippe saab kasutada mitmetes rakendustes. Programme on ka heli salvestamiseks ning muusika loomiseks ja redigeerimiseks.

Andmete organisatsioon

Organisatsiooni järgi võib andmed jagada järgmistesse gruppidesse:

- muutujad ehk mäluväljad,
- andmekogumid,
- objektid.

Muutujad

Muutuja ehk **mäluväli** (öeldakse ka mälupesa jms) on koht arvuti mälus, mis koosneb teatud hulgast baitidest, kuhu programm saab salvestada mingi **väärtuse**: arvu, teksti jms. **Viitamiseks** väljale (muutujale) kasutatakse programmis selle **nime**.

Muutujat võib käsitleda objektina.

Nimetus **muutuja** tekkis programmeerimiskeelte loomise algaastatel, mil arvutitel lahendati peamiselt matemaatilisi ülesandeid ja kajastab seda, et mäluvälju kasutatakse muutuvate suuruste väärtuste salvestamiseks, st välja väärtused võivad olla erinevad.

Rakenduse koostajale ei pruugi olla tähtis, kus asub väli või mis kujul täpselt salvestatakse selles väärtused ja milline on välja pikkus (baitide arv väljas). Olgu öeldud, et viimane omadus võib siiski mõnikord olla oluline ning programmi koostamisel saab seda teatud määral reguleerida.

Muutujaid võib ette kujutada kui nimega tähistatud lahtrit tabelis või kirje väljana.

eesnimi	perenimi	vanus	pikkus	kaal	
Juku	Naaskel	10	153	42	…

eesnimi	perenimi	vanus	pikkus	kaal
Juku	Naaskel	10	153	42

Programmides **Korrutamine** (vt lk 4) on kasutusel kuus muutujat: **a**, **b**, **k**, **n**, **vastus**, **vigu**.

Omadused (atribuudid)

Nimi identifitseerib mälupesa (muutuja). Enamikes programmeerimiskeeltes on nimede esitamiseks üsna täpsed reeglid. Enamasti nõutakse, et nimi võib sisaldada ainult **tähti** ja **numbreid** ning peab algama tähega. Erandina on lubatud kasutada **allkriipsu** tühiku asendajana mitmeosalistes nimedes, sest tühikute kasutamine nimedes on tavaliselt keelatud. Füüsilisel tasemel on muutuja üheks omaduseks ka aadress – välja esimese baidi aadress.

Muutuja
nimi väärtus väärtuste tüüp välja pikkus skoop
loomine() väärtuse omistamine() väärtuse lugemine() eemaldamine()

Väärtus on mäluväljas salvestatud muutuja väärtus (arv, tekst, …). Igal ajahetkel saab muutujal olla ainult üks väärtus.

Väärtus tekib või muutub siis, kui see omistatakse muutujale programmi täitmisel vastava käsu toimel. Suureks erandiks on selles osas Scratch, kus saab muutujale väärtuse omistada ka enne programmi täitmist, kasutades muutuja monitori liugurit.

Iga muutujaga saab programmis seostada tüübi: tekst, täisarv, reaalarv, tõeväärtus jms. Tüübist sõltub ka välja pikkus. Võimalik valik sõltub programmeerimiskeelest. Paljudes keeltes (Java, C, Pascal) on tüübi määramine kohustuslik, kuid mitmes keeles ei ole see vajalik (Visual Basic) või pole isegi võimalik (Scratch, JavaScript, Python). Kui muutuja tüübi määramine ei ole vajalik (tüüp on määramata), valib väärtuste esitusviisi ja väljade pikkused translaator. Programmi koostamisel (näiteks väärtuste leidmist kirjeldavates avaldistes) peab aga arvestama muutujate väärtuste liigiga (arv, tekst, tõeväärtus).

Skoop on muutuja **määramispiirkond**. Enamikes programmeerimissüsteemides on muutujal (aga näiteks ka andmekogumil) vähemalt kaks võimalikku skoopi – terve rakendus või konkreetne protseduur või objekt, ning räägitakse globaalsetest ja lokaalsetest muutujatest. Esimesel juhul saab muuta ning kasutada muutuja väärtusi kõikides programmi protseduurides või skriptides, teisel juhul on muutujale juurdepääs ainult kindlas protseduuris või objektis.

Tegevused

Loomine: muutuja loomine seisneb sellele mäluvälja eraldamises arvuti mälus. Enamikes programmeerimiskeeltes toimub see programmi täitmise ajal. Erandiks on Scratch, kus muutujad luuakse enne programmi täitmist.

Eemaldamine: muutuja eemaldamine seisneb mälu vabastamises. Enamike süsteemide korral toimub see automaatselt programmi või protseduuri töö lõppemisel. Mõned keeled võimaldavad eemaldada muutujaid täitmise ajal. Scratchis saab muutujaid eemaldada nö „käsitsi“.

Väärtuse omistamine ja **väärtuse lugemine** on kaks põhitegevust, mida arvuti saab täita. Väärtuse omistamine seisneb mingi kindla väärtuse salvestamises antud muutuja väljas. Tüüpiliselt eelneb sellele väärtuse tuletamine (leidmine) etteantud avaldise alusel või lugemine väliskeskkonnast. Üheks peamiseks vahendiks väärtuse leidmisel ja salvestamisel on **omistamislause**, millel on enamikes programmeerimis-keeltes järgmine kuju:

muutuja = avaldis

Scratchis kasutatakse **omistamisplokki**: 

Siin on **muutuja** esindatud oma nimega, märki „=“ nimetatakse **omistamissümboliks** (mitte võrdusmärgiks), **avaldis** määrab väärtuse leidmise eeskirja. Avaldise väärtuse leidmisel kasutatakse tavaliselt konstante ja muutujate väärtusi. Avaldise erijuhuks on ka konstant ja muutuja.

S = 0: k = 1: nimi = „Juku Naaskel“

siin omistatakse muutujatele konstantide väärtused, st salvestatakse need vasakus pooles näidatud muutujate väljades

max = a 

loetakse muutuja **a** väärtus ja salvestatakse see muutujas **max** (toimub kopeerimine)

D = b * b – 4 * a * c

loetakse muutujate **a**, **b** ja **c** väärtused,leitakse avaldise väärtus ja omistatakse muutujale **D, st** salvestatakse selle väljas

k = k + 1 

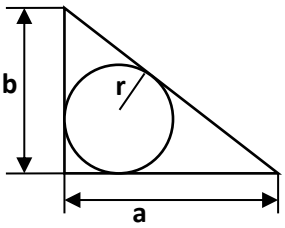
loetakse muutuja **k** väärtus, liidetakse sellele **1** ja tulemus salvestatakse tagasi muutujasse **k**

n = n – 5 

loetakse muutuja **n** väärtus, lahutatakse sellest **5**, tulemus salvestatakse tagasi muutujasse **n**

Siin toodud näidetes esinevad avaldistes lisaks muutujatele ka **konstandid**. Konstandi väärtus kirjutatakse vahetult programmi lausesse (avaldisse).

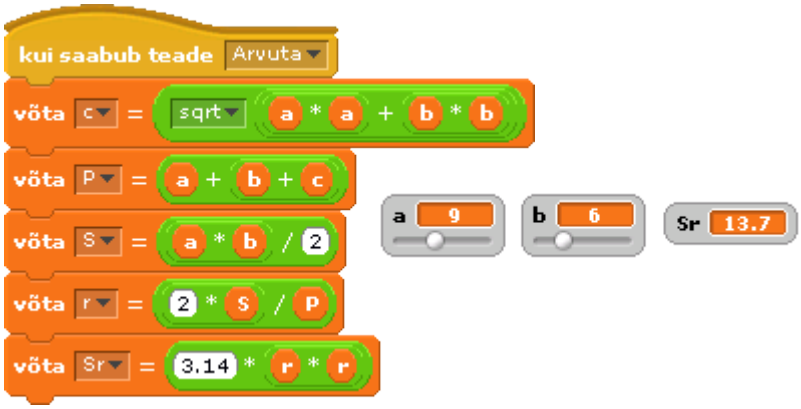
Näide: Kolmnurga siseringi pindala leidmine



Näites on toodud arvutusliku iseloomuga Scratchi ja Visual Basicu programmid. Need leiavad täisnurkse kolmnurga siseringi pindala (**Sr**) antud kaatetite pikkuste (**a, b**) järgi:

$$c = \sqrt{a^2 + b^2}; \quad P = a + b + c; \quad S = a \cdot b / 2; \quad r = 2 \cdot S / P; \quad Sr = \pi \cdot r^2$$

Põhiosa programmides koosneb omistamislausete jadast. Mõlemas programmis on kasutusel muutujad **a**, **b**, **c**, **P**, **S**, **r**, **Sr**. VBA protseduuris on kasutatud muutujate kirjeldamiseks Dim lauset, muutujatele tüüp võetakse automaatselt.




```
Sub Arvuta ()
  Dim a, b, c, p, S, r, Sr
   a = Val(InputBox ("anna a"))
   b = Val(InputBox ("anna b"))
   c = Sqr (a * a + b * b)
   P = a + b + c
   S = a * b / 2
   r = 2 * S / P
   Sr = 3.14 * r * r
   MsgBox "Pindala= " & Sr
End Sub
```

Scratchi skriptis eeldatakse, et muutujate **a** ja **b** väärtusi muudetakse liugurite abil ning tulemus (siseringi pindala) kuvatakse muutuja **Sr** monitoris.

Visual Basic programm loeb sisendboksidest ja salvestab vastavates muutujates kaatetite pikkused (a, b). Järgnevad omistamislaused leiavad ja salvestavad vastavates muutujates hüpotenuusi (c) ümbermõõdu (P), kolmnurga pindala (S), siseringi raadiuse (r) ja siseringi pindala (Sr). Lause **MsgBox** kuvab muutuja Sr väärtuse teateboksis.

Andmekogumid

Andmekogumeid nagu loendid, tabelid, massiivid, kirjed, kollektsioonid, nimistud jms, saab moodustada ja kasutada (sõltuvalt programmeerimissüsteemist) nii märk-, graafika- kui ka heliandmete jaoks. Eriti iseloomulik on andmekogumite kasutamine märkandmete puhul. Siin tutvume kahe lihtsama ja enamkasutatava kogumiliigiga: **loendid** ja **tabelid**.

Loendid

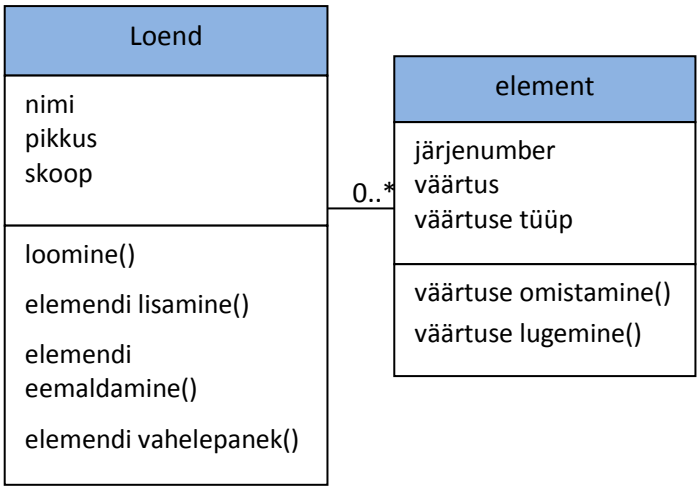
Loend ehk ühemõõtmeline massiiv (öeldakse ka **järjend** või **vektor**) kujutab endast järjestatud elementide (muutujate) kogumit, mis tähistatakse programmis ühe nimega. Viitamiseks elementidele kasutatakse loendi nime koos **indeksiga** (järjenumbriga). Näites on toodud kaks loendit. **T** sisaldab töötajate nimesid, **Palk** nende töötajate palkasid.

T

Kask	**Kuusk**	**Mänd**	**Paju**	**Saar**	**Tamm**	**Vaher**			
1	2	3	4	5	6	7	8	9	10
920	**670**	**1120**	**990**	**1040**	**1230**	**848**			

Palk

Loendi **põhiomadusteks** on **nimi**, **pikkus** ja **skoop**. Nimede jaoks kehtivad enamikes programmeerimiskeeltes samad reeglid nagu muutujate jaoks. Loendi pikkuseks on elementide arv selles. Loendite elementide numeratsioon algab tavaliselt ühest või nullist. Mõnedes programmeerimiskeeltes saab kirjelduses määrata indeksi minimaalse ja maksimaalse väärtuse. Scratchis on indeksi minimaalne väärtus alati 1, maksimaalne väärtus ei ole piiratud. Skoobi tähendus on samasugune nagu muutujal.



Loendi (massiivi) loomine seisneb selle elementidele mäluväljade eraldamises, mis tüüpiliselt tehakse programmis oleva kirjelduse (deklaratsiooni) alusel programmi täitmise ajal. Erandiks on jällegi Scratch, kus loendi loomine toimub „käsitsi". Esialgu elemendid puuduvad. Elementide lisamise, eemaldamise ja vahelepaneku käske ei pruugi igas keeles olla. Sel juhul tuleb taolised operatsioonid eraldi programmeerida. Scratchis on need käsud olemas.

Loendi element vastab oma olemuselt muutujale – tegemist on mäluväljaga. Oluliseks erinevuseks on see, et loendi elementidel ei ole eraldi nimesid, neile vastavad **järjenumbrid**. Elemendi järjenumbrid võivad muutuda, kui elemente lisatakse vahele või neid eemaldatakse. Põhimõtteliselt võivad massiivi elementidel olla erinevad tüübid (liigid), kuid tavaliselt on elementide tüübid siiski ühesugused. Väärtuste omistamine massiivi elementidele ja väärtuste lugemine toimub analoogselt muutujatega, erinevus on vaid viitamise viisis – loendi nimi koos indeksiga:

nimi(*indeks*) või **nimi[*indeks*]** või 

Siin **nimi** on loendi nimi, **indeks** – elemendi järjenumber, mis võib olla kas konstant, muutuja või avaldis. Mõnes keeles paigutatakse indeks ümarsulgudesse, teistes nurksulgudesse. Scratchis kasutatakse viitamiseks loendi elementidele spetsiaalset plokki. Mõned näited:

T(3) = „Känd" 

Loendi **T** kolmandale elemendile omistatakse väärtus Känd.

p = Palk(1) + Palk(k)

Loendi **Palk** esimesele elemendile liidetakse sama loendi element numbriga **k** ja saadud väärtus omistatakse muutujale **p**.

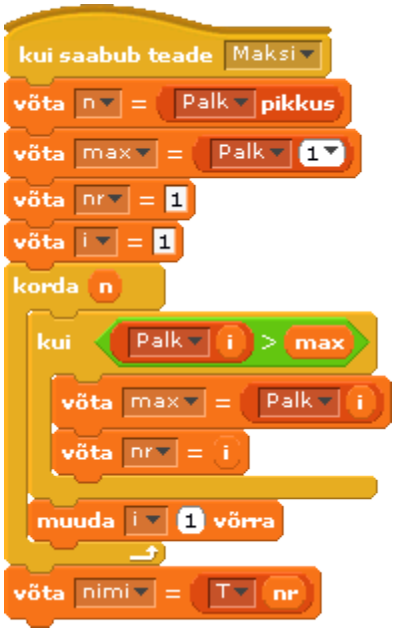
Allpool on toodud Scratchi skript ja Visual Basicu protseduur, mis leiavad loendis (massiivis) **Palk** olevate palkade aritmeetilise keskmise. Esitatud on ka Scratchi loendid **T** ja **Palk**.


```
Sub Keskmine()
  Dim k, n, Sum, kesk
  Dim Palk(1 To 10)
  Loe_Palk Palk, n
  Sum = 0
  For k = 1 To n
    Sum = Sum + Palk(k)
  Next k
  kesk = Sum / n
  MsgBox kesk
End Sub
```

Scratchi skript teeb kõigepealt kindlaks loendite pikkused (elementide arvu loendites). Loendid on ühesuguse pikkusega, seega pole oluline, millise järgi pikkus määratakse. Põhiosa käskudest on seotud loendi **Palk** elementide summa leidmisega. Summa (muutuja **Sum**) algväärtus võetakse alguses null. Korduses liidetakse seejärel summale järjest juurde iga töötaja palk. Viitamiseks loendi erinevatele elementidele kasutatakse muutujat **k**, mille väärtust suurendatakse igal korduse sammul ühe võrra alates ühest kuni töötajate arvuni (muutuja **n** väärtus). Summa jagatakse töötajate arvuga ja saadud väärtus omistatakse muutujale **kesk**.

Visual Basicus toimub kõik analoogselt. Massiivide asukoht ei ole praegu oluline (see võib olla vormil, Exceli töölehel, failides vm). Protseduurid Loe_Palk ja Loe_PT hangivad massivide elementide väärtused.

Allpool on toodud Scratchi skript ja Visual Basicu protseduur, mis leiavad maksimaalse palga ja seda saava töötaja nime. Keskel on toodud protseduuri **algoritm**.




```
protseduur Maksi
  loe T,  Palk
  n = Palk.pikkus
  max =Palk(1)
  nr = 1
  kordus i = 1..n
    kui Palk(i) > max siis
       max = Palk(i)
       nr = i
    lõpp kui
  lõpp kordus
  nimi = T(nr)
  kuva nimi, max
```

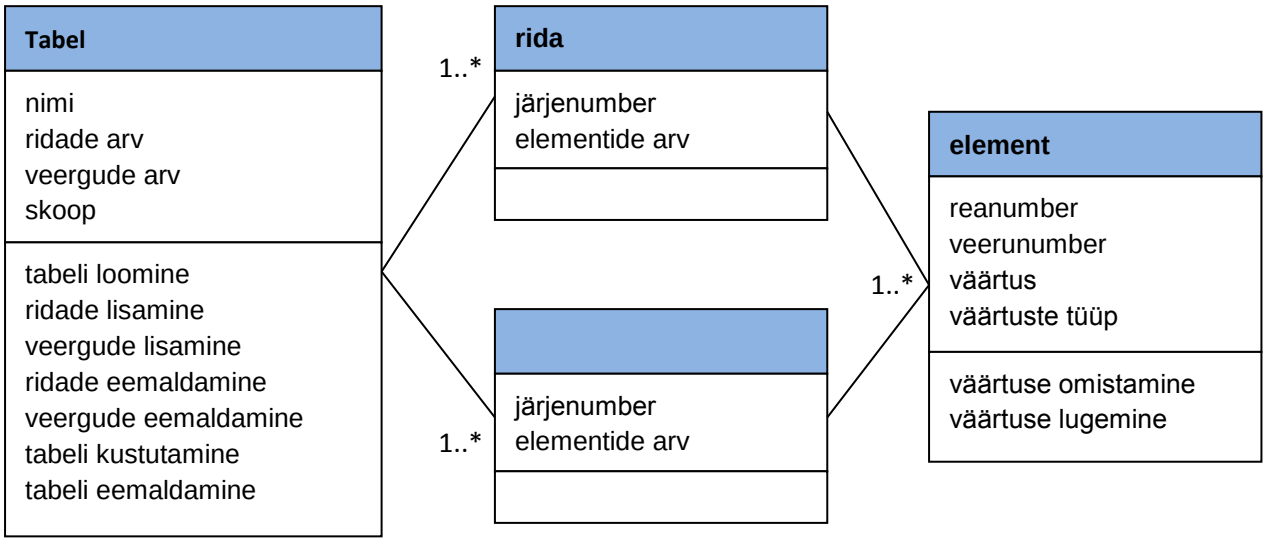
```
Sub Maksi()
   Dim i, n, max, nr, nimi
   Dim T(1 To 10), Palk(1 To 10)
   Loe_TP T, Palk, n
   max = Palk(1)
   nr = 1
   For i = 1 To n
      If Palk(i) > max Then
         max = Palk(i)
         nr = i
      End If
   Next i
   nimi = T(nr)
   MsgBox nimi & ", palk: " & max
End Sub
```

Tabel

Tabel on väga sageli kasutatav andmekogum. Tabeli korral on tegemist kahemõõtmelise massiiviga. Viitamiseks tabeli elementidele kasutatakse nime koos kahe indeksiga:

nimi(rida, veerg)

Näiteks A(1, 1), A(i, j), …



NB! Scratchis tabeleid kasutada ei saa.

Objektid

Objektorienteeritud programmeerimissüsteemides kasutatakse tarkvaraobjekte, mis modelleerivad reaalsete objektide olekut ja käitumist või esindavad teatud tarkvarakomponente. Objektideks võivad olla mitmesugused andmekogumid, dokumendid ja nende osad, vormid, ohjurid jms. Kitsamas tähenduses on objekt lihtsalt teatud andmeüksus, millega programm saab täita mingeid tegevusi. Üheks tüüpiliseks objektiks on näiteks graafiline kujutis ehk graafikaobjekt.

Laiemas tähenduses on objekt tarkvaraüksus, milles on ühendatud andmed ja protseduurid (programmid), mis määravad tegevusi nendega. Iga objektiga on seotud teatud valik **atribuute** ehk **omadusi** (näiteks

graafikaobjekti jaoks on nendeks nimi, mõõtmed, asukoht ekraanil jms) ning **meetodeid,** mille abil määratakse tegevusi (operatsioone) antud tüüpi objektiga (näiteks graafikaobjekti puhul asukoha muutmine ekraanil, pööramine, värvuse muutmine jms). Omadused kujutavad endast objekti sisemuutujaid: igale omadusele vastab mäluväli, kus säilitakse selle jooksvat väärtust. Meetodid on protseduurid, mille abil määrtakse tegevusi objektidega. Tüüpiliselt muutuvad selle tulemusena objekti omadused. Näiteks meetodid graafikaobjekti teisaldamiseks (asukoha muutmiseks) muudavad selle asukohta määravaid koordinaate. Objekti saab panna reageerima teatud **sündmustele** – hiireklõps, vajutus kindlale klahvile, kokkupuude teise objektiga, teate saabumine teiselt objektilt jms. Sündmuse esinemisel käivitub automaatselt vastav protseduur või skript. Kui süsteem toetab paralleelprotsesse (nagu Scratch), siis võib sündmuse esinemisel käivituda ka mitu protsessi.

Ühetüübilised objektid kuuluvad ühte **klassi**. Klass on üldistus ehk abstraktsioon, mis määratleb (kirjeldab) antud klassi kuuluvate objektide olemuse ja kasutamise. See fikseerib objektide **omaduste** ja **sündmuste** valiku, mis on ühesugune kõikidel antud klassi kuuluvatel objektidel, sisaldades protseduure **meetodite** realiseerimiseks ning malli, mille alusel luuakse uus objekt.

Koostades objektorienteeritud keskkonnas oma rakendusi, saab rakenduse looja kasutada oma programmides olemasolevaid objekte või luua uusi objekte klasside määratluste alusel. Paljud süsteemid võimaldavad defineerida ja luua ka oma klasse. Viitamiseks objektide omadustele ja meetoditele kasutatakse enamikes süsteemides taolisi konstruktsioone:

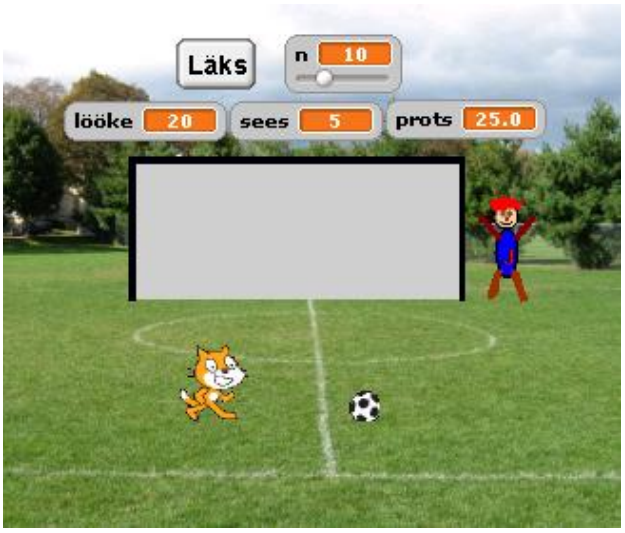
objekt.omadus ja **objekt.meetod** ***argumendid***

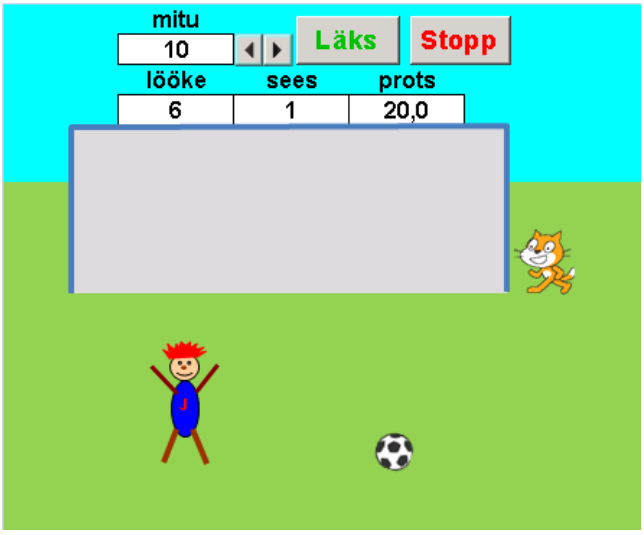
Siin on objekt **viit objektile** (võib olla ka muutuja); omadused ja meetodid esitatakse nende nimede abil. Meetodites kasutatakse sageli argumente, mis määravad omaduste uued väärtused, väärtuste muutuse, tegevuse tüübi, …

Sprait
nimi asukoht: X, Y suund, suurus värvus, nähtavus, …
liigu(), pööra() muudaX(), muudaY() vaheta_kostüümi() muuda_värvi() ütle(), mängi_heli(), …

Scratchis objekti mõistet otseselt ei kasutata, kuid praktiliselt on tegemist objektorienteeritud süsteemiga. Kõrval on toodud klassi **Sprait** kirjeldus, milles on esitatud valik omadusi ja meetodeid. Spraidid, lava ja selle taustad ning muud elemendid (pliiats, heliklipid jms) kujutavad endast objekte. Skriptide loomiseks kasutatavad käsud vastavad oma olemuselt meetoditele. Scratchis on objektide, nende omaduste, meetodite ja sündmuste kasutamine viidud sellisele tasemele, et programmi koostaja ei peagi nendest eriti täpselt teadma. Kuna skriptid tehakse Scratchis konkreetse spraidi jaoks, ei ole käskudes viidet objektile nagu oli näidatud ülalpool.

Näites on toodud Scratchi ja VBA (Excelis) programmid, mis imiteerivad pealelööke väravale. Programmid loendavad löökide ja tabamuste arvu ning leiavad tabavusprotsendi. Mõlemas programmis on neli graafikaobjekti: **Kraps**, **Juku**, **pall** ja **värav**. Scratchi programmis teeb pealelööke **Kraps**, VBAs **Juku**. Vaata demosid: Scratch, VBA (Excel).





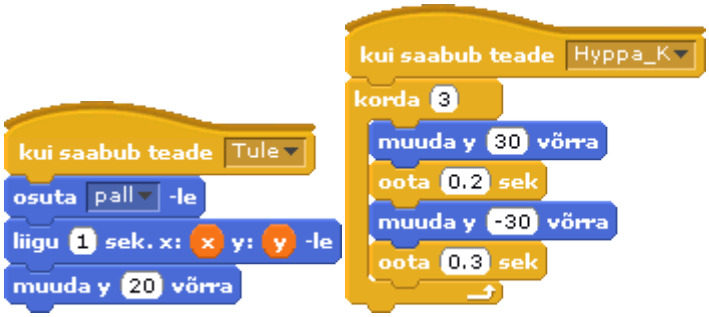
Palli skript

VBA peaprotseduur

```
Sub Trenn()
[lööke] .Value = 0
[sees].Value = 0
Do Until [lööke] = [n]
                                        ' pall platsile
  x = 40 + Rnd() * 400
  y = 160 + Rnd() * 150
  Liigu_XY pall, x, y, 10
                                        ' Juku palli juurde
  Liigu_XY Juku, x, y, 5
  Juku.IncrementTop -50
  paus 1
                                        ' pall värava suunas
  x = 40 + Rnd() * 400
  y = 20 + Rnd() * 200
  Liigu_XY pall, x, y, 20
  [lööke] = [lööke] + 1
  If On_Puude(pall, värav) Then
    [sees] = [sees] + 1
    Hyppa Juku, 3, 30
  Else
    Hyppa Kraps, 4, 20
  End If
  [prots] = [sees] / [lööke] * 100
  paus 0.5
Loop

pall.Left = Juku.Left
pall.Top = Juku.Top + Juku.Height
End Sub
```

Krapsu skriptid



VBA alamprotseduur

```
Sub Hyppa(kuju, n, h)
  Dim i
  For i = 1 To n
    kuju.IncrementTop  -h
    paus 0.2
    kuju.IncrementTop  h
    paus 0.4
  Next  i
End Sub
```

Tegevused on mõlemas programmis analoogsed. Kasutaja saab valida löökide arvu, st muutuja **n väärtuse**.

Tegevusi korratakse **n** korda:

1. **pall** viiakse sujuvalt juhuslikku kohta platsil,
2. lööja (**Kraps** või **Juku**) liigub **palli** juurde,
3. **pall** viiakse juhuslikku kohta **värava** piirkonnas,
4. suurendatakse löökide arvu ühe võrra,
5. kui **pall** puudutab **väravat,**
6. suurendatakse muutuja **sees** väärtust ühe võrra,
7. lööja (**Kraps** või **Juku**) teeb mõned hüpped,
8. vastupidisel juhul teeb hüppeid teine osaleja (**Juku** või **Kraps**),
9. arvutatakse protsent.

Kui löögid tehtud, viiakse **pall** lööja juurde ja lõpetatakse programmi töö.

Scratchis kasutatakse **Krapsu** viimiseks **palli** juurde **Krapsu** skripti **Tule** ning hüpete tegemiseks vastavalt **Krapsu** ja **Juku** skripte **Hyppa_K** ja **Hyppa_J** (ei ole näidatud).

VBAs kasutatakse hüpete juhtimiseks ühte parameetritega protseduuri **Hyppa(kuju, n, h),** mille poole pöördutakse kaks korda erinevate argumentide komplektiga.

Scratchis kuuluvad skriptid kindlale spraidile ja määravad käsuplokkide (meetodite) abil tegevusi ainult antud spraidi jaoks. Sellepärast viiteid objektidele (spraitidele) skriptides ei kasutatagi.

VBAs saab protseduur määrata tegevusi mitme objektiga ning seepärast peab käsus alati näitama objekti. Tegevuste määramiseks kasutatakse **meetodeid** (*IncrementTop*), **omadusi** (Left**,** *Top*) ja spetsiaalseid **protseduure**: **Liigu_XY** (vastab Scratchi käsule [**liigu *t* sek x…, y…**]), **On_Puude** jms. Kõrvaloleval pildil on toodud väike valik klassi **Shape** (kujund ehk graafikaobjekt) omadusi ja meetodeid.

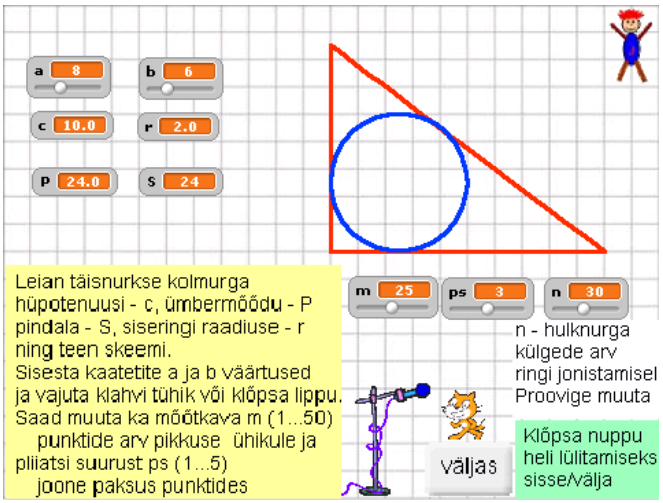
Shape
Name Left, Top, Rotation Width, Height Visible, …
IncrementLeft () IncrementTop () IncrementRotation () Copy (), Cut() Select (), …

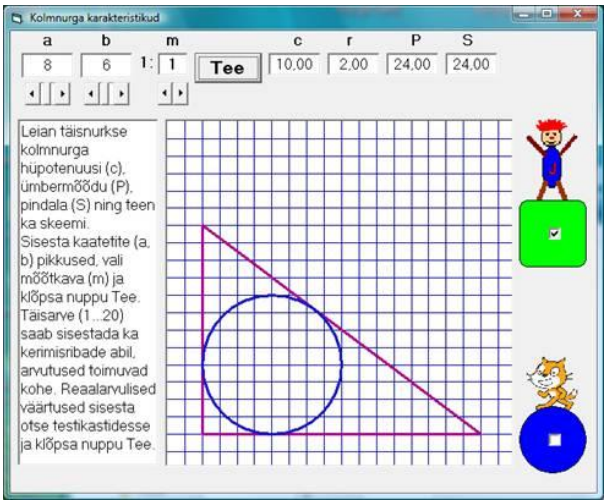
Lisaks graafikaobjektidele on VBA programmis kasutusel neli töölehe lahtrit: **[lööke]**, **[sees]**, **[prots]** ja **[n]**, mille näol on samuti tegemist objektidega. Konstruktsioon [***lahtri_nimi***] on üks võimalikest viitamisviisidest lahtritele VBA programmides. Lahtril on üsna palju omadusi ja meetodeid: **Name**, **Value** (väärtus), **Address** jm. Omadus **Value** on lahtri jaoks vaikimisi võetav väärtus ning selle võib programmis ära jätta.

Siin ei ole näidatud väikest pealisehitust VBA programmile, mis määratleb objekti muutujad ja seob nendega vastavad objektid.

Kasutajaliides

Kasutajaliides sisaldab vahendeid, mille abil kasutaja saab rakendusega suhelda – näha tulemusi, muuta algandmeid, anda vajalikke korraldusi jms. Programmeerimissüsteemides on liideste loomiseks spetsiaalsed vahendid. Tavaliselt tehakse kasutajaliides rakenduse loomise faasis, kuid programmis võivad olla võimalused liidese muutmiseks ka täitmise ajal.





Pildil on näha Scratchis ja Visual Basicus loodud rakenduste kasutajaliidesed. Programm võimaldab leida kaatetite väärtuste järgi täisnurkse kolmnurga hüpotenuusi, ümbermõõdu, pindala ja siseringi raadiuse ning teeb ka vastava joonise. Erinevate andmeliikide kasutamise ning mõningate juhtimisega seotud tegevuste demonstreerimiseks on programmidesse lisatud ülesande lahendamise seisukohast mittevajalikke elemente – animatsioone ning Scratchis lisaks ka heliefekte. Vt programmi täitmist Scratchi näite veebilehelt ja VB rakenduse juurest.

Rakenduste loomise vahendid

Üldotstarbelised programmeerimiskeeled ja -süsteemid

Java, C, **C++**, **C#**, **Visual Basic**, **Pascal**, **Fortran**, … võimaldavad luua **autonoomseid rakendusi**. Programmi saab transleerida masinkeelde ja selle kasutamiseks pole enam vaja programmeerimissüsteemi, millega antud rakendus loodi. Lihtsamal juhul on Windowsis tegemist nn EXE-failiga. Kasutajaliidese realiseerimiseks kasutatakse tüüpiliselt vorme.

Üldotstarbelised rakendusprogrammid

Enim kasutatakse **tarkvarakomplekte** MS Office, OpenOffice, CorelOffice, mis sisaldavad tabeli-, teksti-, esitlus- ja andmebaasirakendusi, ning **graafikaprogramme** nagu AutoCAD, CorelDRAW jt. Enamik kaasaegsetest rakendusprogrammidest sisaldavad ka arendamisvahendeid, millest tuntuimaks on *Visual Basic for Application* (**VBA**). Lisaks Microsofti toodetele kasutatakse VBA-d enam kui 200-s rakendusprogrammis, seal hulgas ka sellistes tuntud süsteemides nagu AutoCAD, CorelOffice ja CorelDRAW.

Üldotstarbelised rakendusprogrammid võimaldavad luua nn dokumendipõhiseid rakendusi, mis luuakse mingi baasrakenduse keskkonnas. Võib eristada kolme põhivarianti:

- **Kasutatakse ainult baasrakenduse vahendeid** (tabelarvutustarkvara, graafikaprogrammid ja andmebaasisüsteemid). Näiteks oleks tabelarvutustarkvara, mis võimaldab tabeleid luua, kasutada valemeid, suurt hulka sisefunktsioone, diagramme jmt; võimaldavad luua efektiivseid ja paindlikke rakendusi ka arendusvahenditeta. Vt näidet: rakendus ruutvõrrandite lahendamiseks.
- **Kasutatakse baasrakenduse vahendeid ja arendusvahendeid** (nt VBA). Baasrakendust kasutatakse eeskätt kasutajaliidese loomiseks, andmete kujundamiseks ja vormindamiseks. Näited Excel+VBA: hunt, kits ja kapsas.
- **Kasutatakse ainult arendusvahendeid**, kusjuures baasrakendus on siis nö konteiner arendusvahendi programmide hoidmiseks. Kasutajaliidesena kasutatakse sellisel juhul tavaliselt vorme. Rakenduse üleviimine ühest rakendusest teise, millel on sama arendusvahend (nt VBA), on võrdlemisi lihtne. Järgnevalt on võimalik vaadata paari väikest näidet, mis on realiseeritud VBA abil Excelis, Wordis ja PowerPointis.

Võrgurakenduste arendusvahendid

Võrgurakendused põhinevad veebi- ehk HTML-dokumentidel. Nende loomiseks on palju erinevaid vahendeid, HTML-redaktoreid nagu **Front Page**, **Dreamweaver**, **HTML Kit** jpt. Veebidokumentide dünaamilisuse suurendamiseks saab kasutada mitmeid skriptikeeli **PHP**, **JavaScript**, **VBScript**, **Python**. Võrgurakenduste loomisel saab kasutada ka üldotstarbelisi programmeerimiskeeli nt **Java**, **C#**, **Visual Basic**, **Net** jt.

Matemaatikaprogrammid

MathCAD, **MathLab**, **Mathematica**, **Maple**, **Wiris** jms, mis sisaldavad rikkaliku valikut vahenditest arvutuslike rakenduste loomiseks. Neis on võimalik luua ka graafikuid ning kasutada ka **sisseehitatud programmeerimisvahendeid**.

Modelleerimiskeeled ja -süsteemid

Rakenduste modelleerimiseks kasutatakse analüüsi- ja disainifaasis üha enam spetsiaalseid keeli, millest üheks enimkasutatavaks on UML (***U**nified **M**odeling **L**anguage*).

Rakenduste loomise põhietapid

Allpool on esitatud rakenduste loomise põhietapid ja nende lühiiseloomustus. Analoogsed etapid esinevad ka muude toodete (ehitised, masinad, …) loomisel.

Ülesande püstitus	põhifunktsioonid, nõuded, tingimused, piirangud, ...
Analüüs	lahendamise üldpõhimõtted, põhiandmed, -objektid ja -tegevused, kontseptuaalsed mudelid, realiseerimisvahendid, ...
Projekteerimine (disain)	kasutajaliidese kavandamine ja loomine, täpsemad mudelid, andmete ja programmide struktuur, algoritmid, dokumenteerimine, …
Realisatsioon	kasutajaliidese loomine, valemite, skriptide ja programmide koostamine, silumine, testimine, dokumenteerimine, ...

Etappide sisu ja mahud sõltuvad rakenduse iseloomust, suurusest, keerukusest, kasutatavast metoodikast ja vahenditest, teostajast või meeskonnast jms. Lihtsate ja väikeste rakenduste korral võivad kõik etapid sulanduda ühte ja taanduda praktiliselt realisatsioonile (programmeerimisele). Suurte ja keeruliste rakenduste korral võivad analüüsi- ja disainietapid olla tunduvalt mahukamad kui realisatsioonietapp. Taoliste rakenduste puhul eristatakse mõnikord kolme osa: ülesande püstitus, projekteerimine ja realiseerimine (ehitamine). Ülesande püstitus võib tulla ühelt firmalt (tellijalt), projekteerimise ja realiseerimise võivad täita erinevad meeskonnad (firmad).

Näites on toodud klassikaline loogikaülesanne ehk mäng „Hunt, kits ja kapsas“, mis on tuntud juba sajandeid erinevate rahvaste folklooris ning sellel on mitmeid variatsioone. Kaasajal on loodud ka palju selle teemaga seotud arvutimänge. Tüüpvariant on järgmine: mees rändab koos hundi, kitse ja kapsaga ning peab saama koos oma seltskonnaga paadiga (parvega) üle jõe, arvestades teatud kitsendusi, mis on esitatud ülesande püstituses (vt ka demo).

Ülesande püstitus

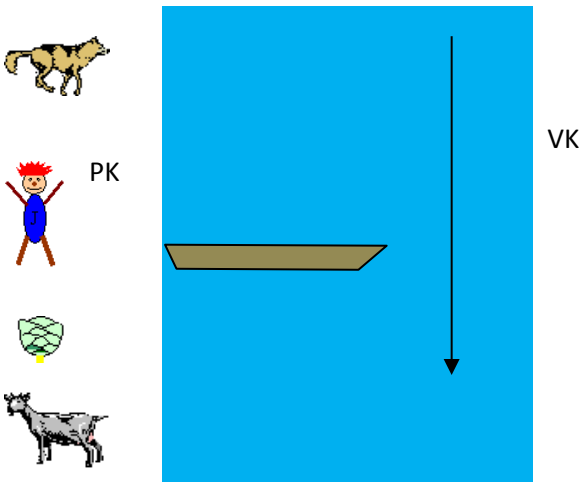
Koostada rakendus, mis võimaldab mängida arvutil mängu „Hunt, kits ja kapsas“.

Juku peab minema üle jõe hundi, kitse ja kapsaga, arvestades järgmist:

- paati mahub lisaks Jukule ainult üks „tegelane“,
- ilma Jukuta paat ei sõida,
- kui kits jääb ilma Jukuta koos kapsaga, sööb kits kapsa ära,
- kui hunt jääb ilma Jukuta kitsega, sööb hunt kitse ära.

Kes ja millal läheb paati või tuleb maha ning millal ja kellega paat ja Juku sõidavad, määrab mängija (kasutaja). Võiks fikseerida ka aja, mis kulub kõigi üle jõe viimiseks.

Analüüs



Põhiobjektideks on **paat, Juku, hunt, kits** ja **kapsas**. Alguses on paat parema kalda (PK) juures ning kõik tegelased on paremal kaldal.

Ülesandel on kaks lahendust:

(1) Juku viib **kitse** vasakule kaldale, tuleb tagasi ja viib üle **hundi** ning toob **kitse** tagasi. Viib üle **kapsa**, tuleb tagasi ja viib vasakule kaldale **kitse**.

(2) Juku viib **kitse** vasakule kaldale, tuleb tagasi ja viib **kapsa** vasakule kaldale ning toob **kitse** tagasi. Siis viib **hundi** üle, tuleb tagasi ja viib vasakule kaldale **kitse**.

Esitame esimese lahenduse stsenaariumi (algoritmi) veidi detailsemalt.

Juku viib kitse vasakule kaldale

Juku tuleb paadiga tagasi.

Juku viib hundi vasakule kaldale.

Juku toob kitse tagasi.

Juku viib kapsa vasakule kaldale.

Juku tuleb tagasi.

Juku viib kitse vasakule kaldale

Toodud stsenaarium annab mängijale üsna täpse tegutsemisjuhendi, kuid jätab rakenduse realiseerimise seisukohalt mitmeid lahtisi otsi. Ei ole selged paatimineku ja mahatuleku reeglid. Näiteks, kas kaks sõitjat lähevad paati või tulevad maha ühekaupa (järjestikku) või korraga? Kui pealeminek või mahatulek toimub järjestikku, võivad tekkida olukorrad, kus kits ja kapsas või hunt ja kits jäävad koos ühte kohta ilma Jukuta.

Näiteks, kui alguses läheb paati kõigepealt Juku, ongi nii hundil kui kitsel vaba voli – Juku on paadis, nemad kaldal. Võib minna lahti suureks söömiseks – kits sööb kapsa ja hunt sööb kitse. Kas arvestada, et Juku on samuti kaldal ja takistab söömist? Et teha mängu raskemaks (huvitamaks) otsustame nii, et pealeminek ja mahatulek toimub alati ühekaupa mängija käsu peale ning kitse ja kapsast või hunti ja kitse ei tohi mingil juhul kusagile jätta ilma Jukuta. Üleviimise detailsem stsenaarium (algoritm) oleks järgmine:

1. Kits paati
2. Juku paati (hunt kapsast ei söö)
3. Paat Juku ja kitsega vasakule kaldale
4. Kits maha vasakule kaldale (Juku jääb paati)
5. Paat Jukuga paremale kaldale
6. Hunt paati (Juku ei pea maha tulema)
7. Paat Juku ja hundiga vasakule kaldale
8. Juku maha vasakule kaldale (enne hunti!)
9. Hunt maha vasakule kaldale (Juku on kohal)
10. Kits paati (enne Jukut, muidu hunt sööb ära)
11. Juku paati
12. Paat Juku ja kitsega paremale kaldale
13. Juku maha paremal kaldal (enne kitse)
14. Kits maha paremale kaldale
15. Kapsas paati (enne Jukut)
16. Juku paati
17. Paat Juku ja kapsaga vasakule kaldale
18. Kapsas maha vasakule kaldale
19. Paat Jukuga paremale kaldale
20. Kits paati
21. Paat Juku ja kitsega vasakule kaldale
22. Juku maha vasakule kaldale
23. Kits maha vasakule kaldale (kõik on üle!)

Nagu näha, on nüüd kirjas üsna palju tegevusi, mis arvestavad, et pealeminek ja mahatulek toimuks ühekaupa ja teatud paare ei jäetaks kunagi kahekesi. Kui mängija tegutseb täpselt selle stsenaariumi järgi,

ei saa tekkida olukorda, kus paati püütakse panna üle kahe tegelase. Kuid kasutaja võib ju mitte reegleid teada või eksida. Kuidas võiks sel juhul toimida? Kaks põhivarianti: rakendus ei luba viia ülearust tegelast paati või paat läheb ülearuse tegelase korral põhja (koos paadisolejatega) ning mängu peab alustama uuesti. Valime teise variandi.

Stsenaariumi alusel saab määrata ka objektide põhitegevused ja -omadused, mida saab arvestada rakenduse kavandamisel ja realiseerimisel.

Paat
nimi koht X, Y mitu ...
algseaded sõit vasakule sõit paremale kas_palju põhjaminek

Paadi omadust **nimi** saab kasutada paadile viitamiseks. Omaduse **koht** abil määratakse paadi asukoht: vasak kallas, parem kallas, jõel, **X, Y** on paadi koordinaadid, mida saab kasutada näiteks sihtkoha määramisel. Omadus **mitu** näitab tegelaste arvu paadis: alguses võetakse selle väärtuseks 0 ning seda muudetakse iga kord, kui mõni läheb paati (+1) või tuleb maha (-1).

Meetod **algseaded** paneb paika mõned algväärtused: asukoht – parem kallas, sõitjate arv (omaduse **mitu** väärtus) – 0 jms. Meetodid **sõit vasakule** ja **sõit paremale** juhivad paadi liikumist vastavalt vasakule või paremale. Meetod **kas palju** kontrollib, kui paati tuleb uus tegelane, kas sõitjate arv ei ole suurem kui kaks. Kui on, käivitatakse meetod **põhjaminek**.

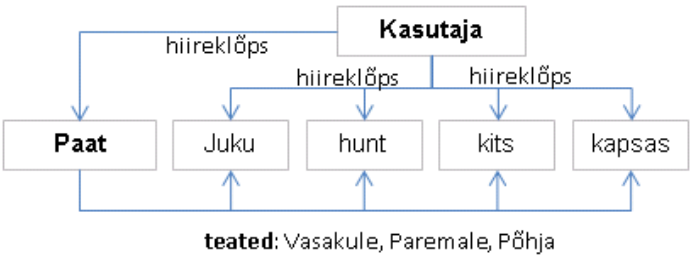
Tegelane
nimi koht X, Y nähtavus
algseaded maha paremal maha vasakul paati sõit vasakule sõit paremale põhjaminek

Neljal objektil **Juku**, **hunt**, **kits** ja **kapsas** (siin kasutatakse nende jaoks ühisnimetust **Tegelane)**, on ühesugune valik omadusi ja meetodeid. Võib rääkida ka klassist **Tegelane**, milles on neli objekti (eksemplari).

Omadust **nimi** kasutatakse viitamiseks konkreetsele objektile, **koht** näitab tegelase asukohta: paremal kaldal, paadis, vasakul kaldal, **X, Y** on objekti koordinaadid, mida kasutatakse liikumiste sihtkohtade määramiseks. Omadust **nähtavus** võib kasutada objekti peitmiseks, kui see läheb koos paadiga põhja.

Meetod **algseaded** viib objekti alguspunkti paremal kaldal ning võtab omadusele **koht** vastava väärtuse jms. Meetodid **maha paremal** ja **maha vasakul** viivad objekti vastavasse punkti paremal või vasakul kaldal, **paati** viib objekti paati. Meetodid **sõit vasakule** ja **sõit paremale** viivad objekti vastavale kaldale, kui objekt on paadis. Meetod **põhjaminek** teeb objekti nähtamatuks, kui paat läheb põhja.

Üheks rakendusega seotud oluliseks tegelaseks on **kasutaja** ehk **mängija**, sest just tema määrab oma tegevusega, mida peab tegema üks või teine objekt ning juhib kogu mängu. Rakenduse töökorraldus võiks olla järgmine. Mängija klõpsab hiirega objekti. Selle vastav protseduur valib hetke seisust lähtudes tegevuse, kasutades objekti omadust **koht**. Kui näiteks klõpsatakse paremal kaldal olevat **paati** ning **Juku** on paadis, liigub **paat** vasakule kaldale. Paadiga koos liigub vasakule ka **Juku.** Kui paadis on veel keegi, liigub ka see. Selle, et sõit toimub, ja sõidu suuna, teatab **paat**. Näiteks, kui klõpsatakse kaldal olevat **hunti** ja **paat** on sama kalda juures, läheb **hunt** paati. Kui paat on teise kalda juures, siis hiireklõps ei mõju.



Sellega on määratletud rakenduse töötamise üldised põhimõtted, fikseeritud põhiobjektid ning nende omadused ja meetodid ning tehtud kindlaks koostöö objektide vahel. Analüüsietapi võib lugeda lõppenuks ja saab asuda disaini (projekteerimise) juurde. Paneme tähele, et seni pole olnud juttu realiseerimisvahendi(te)st. See on üsna tüüpiline analüüsifaasi korral, sest siin vaadeldavad põhimõtted ja mudelid ei peaks olema seotud kindla programmeerimiskeelega.

Sageli valitakse realiseerimisvahendid just analüüsietapi tulemuste alusel, kuid kui realiseerimisvahendid on juba ette teada, võib teatud määral nendega arvestada.

Projekteerimine ehk disain

Disainifaasis peaks juba arvestama kindlate realiseerimisvahenditega, kuid mitmed aspektid võivad olla üldise iseloomuga ja need täpsustatakse realiseerimisfaasis. Siia kuluvad eeskätt kasutajaliidese mitmed elemendid ja algoritmid. Praegu arvestame, et realiseerimine toimub Scratchis.

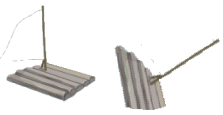
Alustame kasutajaliidesest, mis kujundatakse laval. Lava taustaks on jõe kujutisega pilt. Spraidid (graafikaobjektid) on praegu ka kasutajaliidese elemendid, sest mängija kasutab neid hiirega klõpsates vajalike tegevuste määramiseks.

Laval on mõned teated ja juhendid. Samuti on siin aja (muutuja **aeg**) monitor.

Objektid (spraidid)

Põhimõtteliselt on tegemist samade põhiobjektidega. On tehtud mõningaid mitteolulisi muudatusi ja täiendusi.

- paadi asemel on siin **parv**, lihtsalt pilt on teine. Nimi oleks võinud jääda ka endiseks, kuid praegu on seda siiski muudetud. Spraidil on kaks kostüümi: parv nö normaalasendis ja ümberläinud parv.



- **Juku** – sama, mis enne. Üks kostüüm. Juku kasutamine erineb mõnevõrra teiste tegelaste kasutamisest. Formaalselt viib tema teisi üle: paat ilma Jukuta ei liigu. Muus osas erinevusi pole.

- **hunt** – üks kostüüm. Sööb ära kitse, kui on sellega ilma Jukuta, kapsast ei söö.

- **kits** ja **kapsas** – mõlemal on kaks kostüümi. Teised kostüümid on tekstiteated: „Hunt sõi kitse ära!“ ja „Kits sõi kapsa ära!“

- mõned abispraidid: juhendid, teated jmt.

Muutujad

- **koht_P** – parve asukoht: 1 – paremal kaldal, 2 – vasakul kaldal. Täpsemalt öeldes: parv on parema või vasaku kalda juures. Vastab parve (paadi) omadusele **koht**. Globaalne muutuja. Algväärtus 1. Muudab parve põhiskript.
- **koht** – tegelaste asukohad, neli lokaalset muutujat. Väärtused: 1 – paremal kaldal (maas), 2 – vasakul kaldal, 3 – parvel. Muutujad vastavad tegelaste omadustele **koht**. Algväärtusteks on 1. **NB**! Võivad olla ka globaalsed muutujad, siis peavad neil olema erinevad nimed.

- **mitu** – tegelaste arv parvel. Esindab parve vastavat omadust. Globaalne muutuja, algväärtus 0, suurendatakse ühe võrra iga kord, kui mingi tegelane läheb parvele ning vähendatakse ühe võrra, kui keegi läheb maha. Kui väärtus läheb suuremaks kahest, läheb parv koos sellel olevate tegelastega põhja.
- **aeg** – näitab jooksvat aega. See on globaalne muutuja.
- **tun** – tunnus, mis näitab, kas mäng käib (tun=1) või mitte (tun=0). See on globaalne muutuja. Algväärtus on 1, kui mäng käivitatakse; omistatakse null, kui mäng lõppeb normaalselt: parv läheb põhja, kits sööb kapsa või hunt sööb kitse ära. Kui tun=0, ei järgne klõpsamisele mingeid tegevusi.
- **t** – ülesõidu aeg sekundites on globaalne muutuja, mida saab muuta programmi seadistamisel paadi skriptis algus.

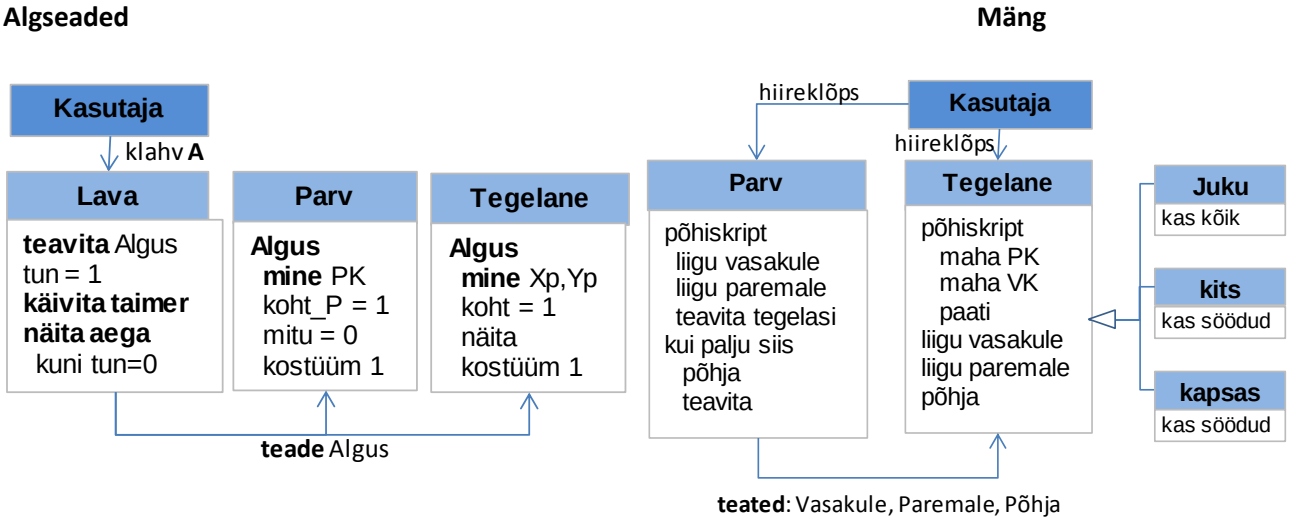
Programm

Scratchi programm koosneb **skriptidest**. Skript kuulub kindlale **spraidile** või **lavale** ning ühe objekti skript ei saa määrata tegevusi teise objektiga, kuid objekt (skript) võib väljastada teate käivitamaks teise skripti, mis saab täita vajalikud tegevused.

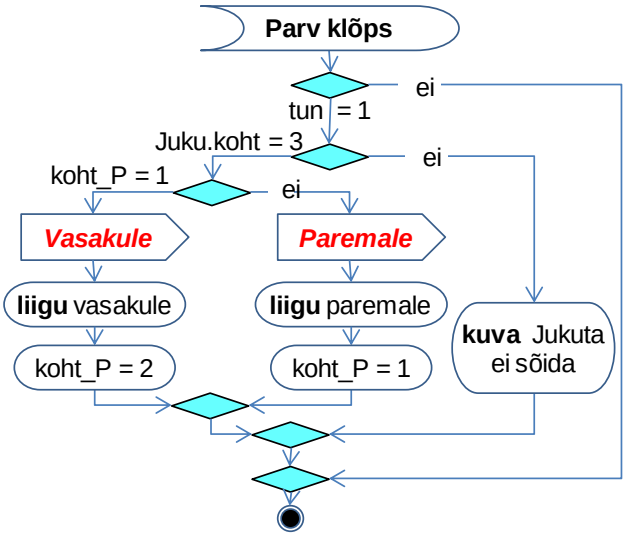
Antud juhul võib programmis eristada kahte osa: **algseaded** ja **mäng**.

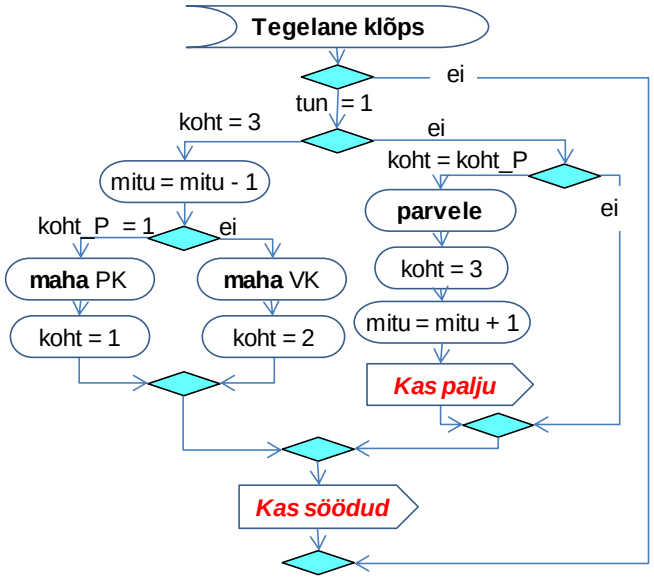
Algseaded tehakse uue mängu algul, kui kasutaja vajutab klahvi **A**. Seda võib teha suvalisel hetkel, katkestades jooksva mängu. Klahvisündmusele reageerib **Lava**. Vastav skript väljastab teate **Algus**. Teate võtavad vastu **parv** ning **Tegelased** (**Juku, hunt, kits** ja **kapsas**). Parve skript **Algus** viib parve parema kalda juurde (**mine PK**), võtab asukohta näitava muutuja **koht_P** väärtuseks on **1** ja muutuja **mitu** väärtuseks **0**.

Kõikide tegelaste tegevused on analoogsed. Skeemil tähendab '**mine Xp, Yp**' objekti viimist parema kalda juurde – punkti koordinaatidega **Xp, Yp**. Erinevatel objektidel on need erinevad, et nad ei kataks üksteist. Konkreetsed väärtused pannakse paika realisatsiooni ajal Scratchis. Käsk **koht = 1** tähendab, et vastava tegelase asukohta määravale muutujale omistatakse väärtus **1**.

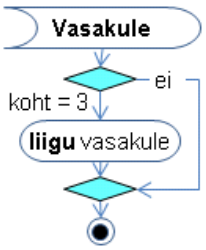


Mängu käigu määrab mängija: arvestades hetkeseisu, vajadust ja reegleid, klõpsab ta vastavat objekti. Allpool on toodud parve ja tegelaste põhiskriptide algoritmid.



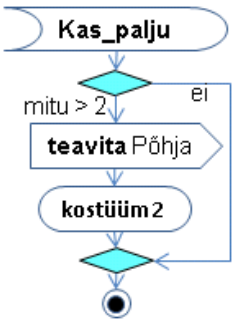


Kui klõpsatakse parve, kontrollitakse, kas mäng käib (**tun = 1**) ning kui **jah**, vaadatakse, kas **Juku** on parvel (**Juku.koht = 3**). Kui on, tehakse kindlaks, kas parv on paremal (**koht_P = 1**) või vasakul kaldal. Vastavalt sellele toimub ka parve liikumine. Enne seda saadab **parv** teate, mille võtavad vastu kõik tegelased, kuid sellele reageerivad ainult parvel olevate tegelaste (**koht = 3**) vastavad skriptid. Näites on toodud skeem liikumiseks vasakule. Käsk **liigu** realiseeritakse Scratchi plokiga [**liigu t sek. x…, y…**] (sujuv liikumine) või **[mine x…, y…]** (kohene asukoha vahetus).



Parve ja sellel olevate tegelaste liikumine peab toimuma sünkroonselt.

Kui klõpsatakse tegelast ja muutja **tun** väärtus on **1** (mäng käib), tehakse kindlaks, kas tegelane on parvel (**koht = 3**) või kaldal. Kui tegelane on parvel, siis vastavalt parve asukohale (näitab muutuja **koht_P** väärtus), läheb tegelane maha paremale (**maha PK**) või vasemale kaldale (**maha VK**); seejuures muudetakse vastavalt ka asukohta näitava muutuja väärtust ja vähendatakse ühe võrra muutuja **mitu** väärtust. Realisatsioonis võib asukoha muutmiseks kasutada Scratchi käsku [**liigu t sek, x…, y…**] (sujuv liikumine) või **[mine x…, y…]** (kohene). Kui tegelane ei ole parvel ja on parvega samal kaldal, läheb ta parvele. Kui tegelane ja parv on erinevatel kallastel, ei tohiks hiireklõps mõjuda.



Kui keegi läheb parvele, suurendatakse muutuja **mitu** väärtust ühe võrra ja saadetakse teade **Kas_palju**. Teate võtab vastu **parv** ja kontrollib muutuja **mitu** väärtust. Kui see on suurem kui 2, läheb parv põhja. Parvele võetakse teine kostüüm, parvel olnud tegelased peidetakse. Mängu tuleb alustada uuesti algusest.

Iga kord, kui toimub kas tegelase minek parvele või mahaminek, saadetakse teade **Kas söödud**. Seda töötlevad **kapsa** ja **kitse** vastavad skriptid, sest neil on oht saada sööduks. Näiteks **kitse** skriptis kontrollitakse, kas **kitsel** ja **hundil** on sama **koht**, kui **jah** ning **Juku koht** ei lange sellega kokku, loetakse **kits** sööduks: võetakse teine kostüüm teatega „Kits on söödud!“ ning muutujale **tun** omistatakse väärtus **0**, mis tähendab mängu lõppu. Seda teeb lava vastav skript. **Kapsaga** toimub kõik analoogselt. Ka **Jukul** on üks lisaskript, mis teeb kindlaks, kas kõik on viidud üle jõe.

```
Kas_söödud
  kui kits.koht = hunt.koht siis
    kui kits.koht <>Juku.koht = 3 siis
      võta kostüüm 2
      tun = 0
```

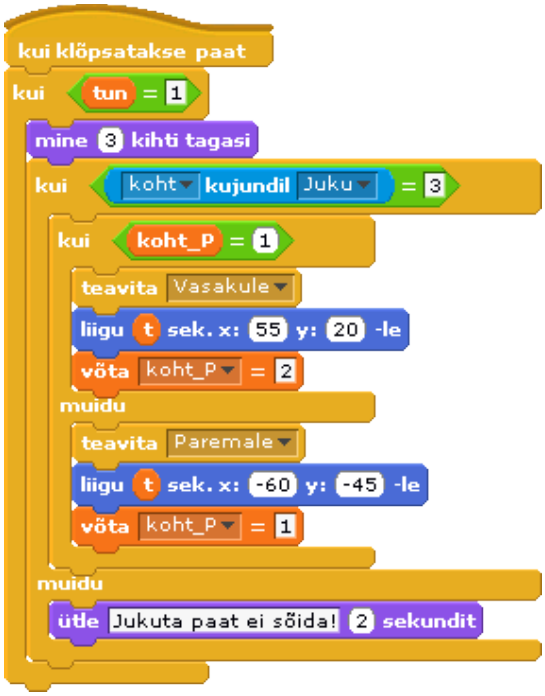
Realisatsioon

Realisatsioonifaasis võidakse teha mõningad täpsustused. Praegu otsustame, et objektide olekute (asukohtade) fikseerimiseks kasutame lokaalseid muutujaid **koht**, ainult parve jaoks kasutame globaalset muutujat **koht_P**, sest sellele tuleb tihti viidata teiste spraitide skriptides. Ülesõitmine toimub sujuvalt, pealeminek ja mahatulek on momentsed. Loomulikult pannakse realiseerimisel paika ka konkreetsed koordinaadid.

Siin on esitatud **Lava** skriptid. Kõik algab skriptist, mis reageerib klahvi **A** vajutusele. See paneb muutuja **tun** väärtuse korraks nulliks, teeb väikese pausi (eelmise mängu peatamiseks) ja väljastab teate **Algus**, mille võtab vastu **Lava** skript **Algus** ja ka tegelaste samanimelised skriptid.

See kõik viib mängu algseisu.

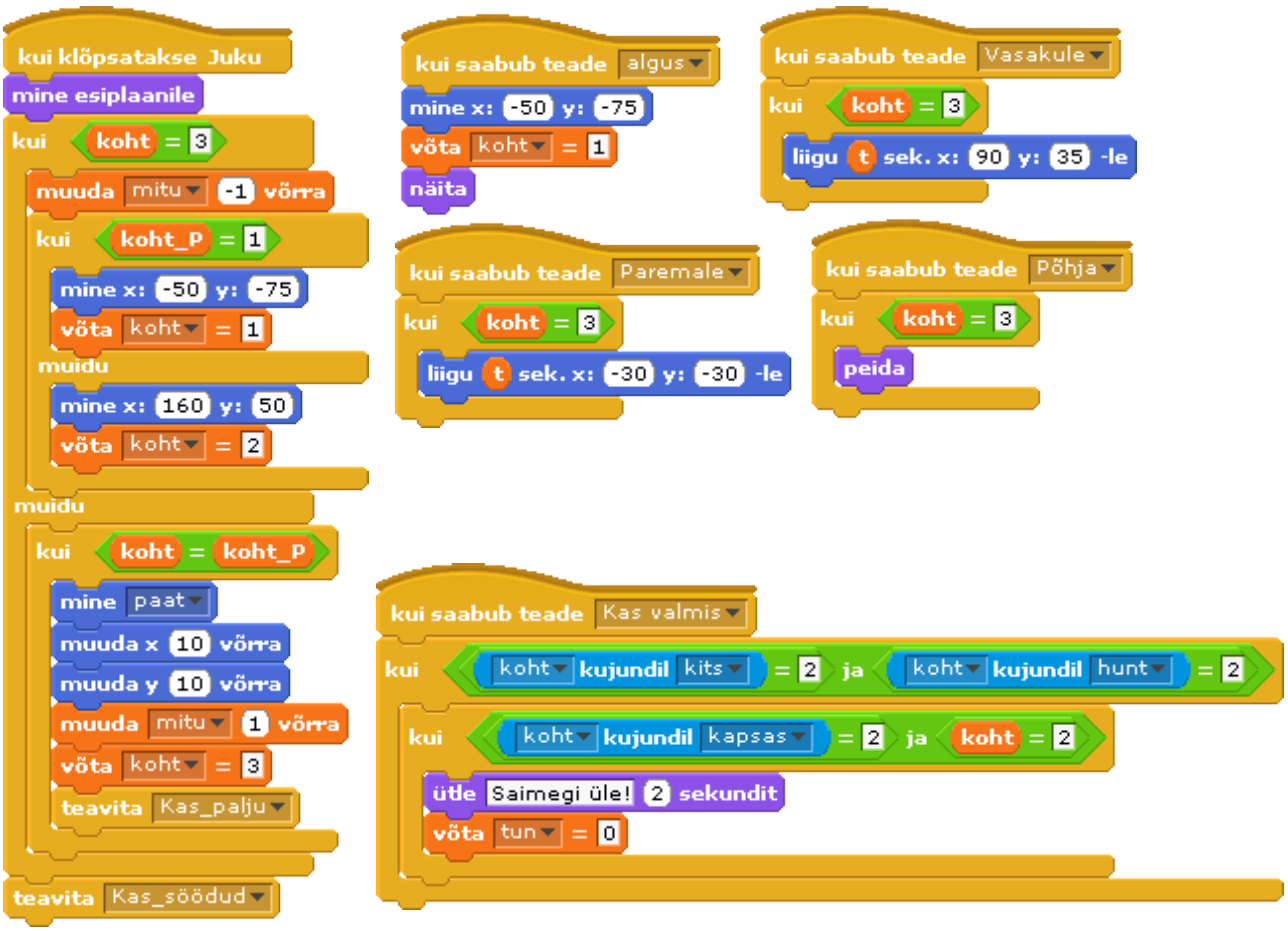
Parve skriptid



Parve koordinaadid vasaku kalda ääres on (-60; -45). Neid kasutatakse skriptis **Algus** ja põhiskriptis liikumisel paremale.

Liikumised toimuvad sujuvalt. See määratakse käskudega [**liigu t sek x…y…**] Siin on muutuja **t** väärtus 1, mis on omistatud skriptis **Algus** ning määrab liikumise aja.

Juku skriptid



Kõikide tegelaste skriptid on väga sarnased ja neid saab luua kopeerimise teel. Koopiates tuleb teha mõningaid muudatusi, mis seisneb peamiselt koordinaatide muutmises.

Juku skriptid erinevad teistest kõige rohkem. Jukut nihutatakse peale parvele minekut käskudega [**muuda x…]** ja [**muuda y…]**, et see ei sattuks kohakuti üleviidava tegelasega. Teistel ei ole see vajalik. Siin on ka üks eriskript: **Kas valmis**, mida teistel ei ole. Skript kontrollib, kas kõik on üle viidud. Selleks peab kõigil olema muutuja **koht** väärtus 2 (vasak kallas).

Ühe spraidi skriptides saab viidata teiste spraitide lokaalsetele muutujatele. Vaadeldav skript kuulub Jukule, kelle lokaalsele muutujale **koht** viidatakse otse nime 

abil, teise spraidi sama nimega muutujale viitamiseks kasutatakse grupi **Andurid** plokki 

.

Kitsel ja kapsal on skriptid, mis kontrollivad võimalikku söömise fakti. Ka neis kasutatakse viitamist teise spraidi muutujatele.

Hundi ja kitse üldised skriptid

Kitse ja kapsa skriptid, mis kontrollivad söömist