Töölehed Scratchi materjali  
juurde

**Sisu**

[Kraps123 3](#_Toc339451953)

[Veemaailm 5](#_Toc339451954)

[Efektid 8](#_Toc339451955)

[Nimi 9](#_Toc339451956)

[Kraps, pall ja papagoi 10](#_Toc339451957)

[Ristkülik 15](#_Toc339451958)

[Ristkülik ja ring 16](#_Toc339451959)

[Lilleaed 17](#_Toc339451960)

[Juhtimisplokid 18](#_Toc339451961)

[Autod 22](#_Toc339451962)

[Rehmaatika 23](#_Toc339451963)

[Ideaal 25](#_Toc339451964)

[Arvu arvamine 26](#_Toc339451965)

[Labürint 27](#_Toc339451966)

[Tagurpidi 29](#_Toc339451967)

[Autod 2 30](#_Toc339451968)

[Vestlus 31](#_Toc339451969)

[Sõnastik. Keele test 32](#_Toc339451970)

[Sõnastik. Tõlkimine 33](#_Toc339451971)

[Algoritme loenditega 34](#_Toc339451972)

[Loendite kasutamine spraitide asukoha juhuslikul määramisel 35](#_Toc339451973)

# Kraps123

## Harjutus 1

Uue Scratchi rakenduse loomisel on algul laval vaid Kraps – kass kahe **kostüümiga** ja heliga ’näu’. Objekti nime kasutatakse spraidile viitamisel teiste spraitide skriptidest; seda võib muuta igal ajahetkel. Kiisule võiks anda nime: **nimekasti** spraidi **info alas** (akna ülaservas) ikoonikese kõrval kirjutada **Kraps**.

Lihtsa skripti loomiseks lohistada **skriptide alasse** allolevad käsud ja moodustata nendest kõrvalolev **skript**. Seda saab käivitada rohelise lipukesega lava ülaservas.

|  |  |  |
| --- | --- | --- |
| kraps123_kasud | cat1-a  cat1-b | kraps123_skript |

Kraps peaks liikuma (suuna määrab spraidi omadus **suund**), vahetama kostüümi, et jääks mulje astumisest ja tegema seda korduvalt (vt. kordused [**lõputult**] või [**korda …**]). Esialgu võib korraldust [**oota …**] mitte lisada, siis sibab Kraps väga kiiresti. Lavalt välja sprait ei liigu, servani jõudes pöördub Kraps tagasi (korraldus [**kui äärel, põrka**]). Et ta ei liiguks tagasi jalad ülespidi, tuleks spraidi info alas valida pööramise stiil ’**ainult nägu vasakule/paremale**’.

[http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr\_juhend14.html#poor](http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr_juhend14.html" \l "poor)

**NB!** Liikumise kiirus ja sujuvus oleneb sammu pikkusest ja vahepealsest ooteajast.

Sammu pikkus ei pruugi olla konstantne suurus. Selle võib määrata juhuslikult: lohistada väärtuse 10 asemele grupist **Tehted** plokk kraps123_juhuarv ja valida juhusliku väärtuse piirideks näiteks -5 kuni 10. Tekib tantsuline liikumine, Kraps liigub tasapisi siiski edasi.

kraps123_liigujuh

Animatsiooni teeb (peale liikumise vilkumise ja helide) atraktiivseks juhuslikkus ja kasutaja võimalus kuidagi sekkuda.

Sammu pikkuse võib lasta kasutajal valida tegevuse käigus. Üks võimalus selleks on luua **muutuja** ’**samm**’, mille väärtuse võrra astub Kraps. Muutuja väärtust saab kasutaja muuta laval **liuguri** abil.

[http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr\_juhend14.html#muut](http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr_juhend14.html" \l "muut)

Valida käsugrupist **Muutujad** korraldus ’**Tee muutuja**’, panna selle nimeks **samm**. Muutuja **monitor** on näha laval, paremklõps sellel avab menüü, kust valida monitori esitusviisiks ’**liugur**’ ja seejärel ’**määra liuguri min ja maks**’, näiteks 1 kuni 10. Korraldusse [**liigu …**] tuleks lohistada käsugrupist **Muutujad** muutuja **samm**.

kraps123_liigusamm

Valida Krapsule liikumise taustaks ka ilus lavapilt: aktiveerida lava (vajutada lava ikoonikesele lava all), valida vahekaardil **Taustad** nupp **Impordi**.

## Harjutus 2

Huvitava animatsiooni saab ka lihtsalt kostüüme (kuvandeid) vahetades. Lisada uus **sprait** nupuga ’**Vali uus sprait failist**’. Valida kindlasti selline, millele saaks valida mitu kuvandit (kostüümi), näiteks ’ballerina-a’. Teised selle spraidi kostüümid lisada vahekaardilt **Kostüümid** nupuga **Impordi**. „Tantsimiseks“ koostada skript:

[http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr\_juhend14.html#kost](http://www.tud.ttu.ee/~vilip/Scratch/Juhend/Scr_juhend14.html" \l "kost)

## Harjutus 3

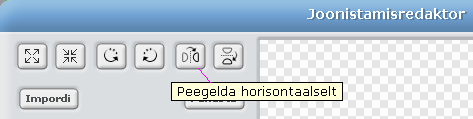
Spraidi kostüümide saamiseks on võimalik kasutada **animeeritud GIF**-i.

Selleks lisada suvaline sprait (’vali uus sprait failist’ või ’Vali üllatussprait’), valida vahekaardilt **Kostüümid** nupp **Impordi** ja viidata **animeeritud GIF-failile**. Iga kaader (kiht) lisandub eraldi kostüümina. Esimene kostüüm, mis tekkis spraiti lisades, tuleb nüüd kustutada.

Lisada spraidid **tiiger** (tiiger.gif) ja **hunt** (hunt.gif). Nende liikumapanemiseks võib mõlemale kopeerida **Krapsu skripti**. Et nendel spraitidel on kostüüme tunduvalt rohkem, tuleb määrata nii samm kui ooteaeg väiksemad ning muidugi pööramise stiil ’ainult nägu vasakule/paremale’.

Kui spraidi liikumapanemiseks soovitakse kasutada korraldust [**liigu … sammu**], tuleb juba kostüümide loomisel arvestada, et liikumissuund on originaalkujutisel alati paremale (90 kraadi).

Tiiger hakkab liikuma ilusasti, hunt aga kulgeb kummaliselt - tagurpidi. Hundi kõiki kostüüme tuleb redigeerida, peegeldades neid horison­taalselt (vahekaart **Kostüümid**, nupp **Redigeeri**, nupp ’**Peegelda horisontaalselt**’).

Spraidist **Hunt** võiks eelnevalt teha koopia. Kui kõikide kostüümide redigeerimine õnnestus, saab ülearuse koopia kustutada.

# Veemaailm

Hajutus Scratchi liikumiskäskudega tutvumiseks.

Veemaailmas on võimalikud väga erinevad liikumised. Iga sprait tuleb panna tegutsema erinevalt.

Kõik kostüümid on leitavad kaustast ***Animals***. Skriptide tööd on mugav kontrollida, kui panna need algul tööle ükshaaval (hiireklõps skriptil). Valmis skripti (või selle osa) saab kopeerida lohistades teisele spraidile, siis ei pea kõiki käske uuesti otsima ja kokku lohistama.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| crab1-a.png crab1-b.png  Krabi nihkub merepõhjal edasi-tagasi.  Algul ’liigu 10 sammu’ ja ’oota 1 sekund’. Sujuvama liikumise saamiseks teha samm ja ooteaeg väiksemaks. Pööramise stiil ’ainult nägu vasakule/paremale’. | | | | | | veemaailm_krabi.gif | |
| fish3.png  Kalake liigub juhuslikult, põrgates servadelt.  Aeg-ajalt muudab kalake ootamatult suunda (teine skript). | veemaailm_kalake_1 veemaailm_kalake_2 | | | | | | |
| veemaailm_hai | | | | | shark1-a  Hai jälitab kalakest.  Kui Hai puudutab kalakest, väljastab tema skript teate ’kala käes’, seepeale võib käivituda mingi teise spraidi skript (vt. Kaheksajalg). | | |
| fish1-a.gif Kala järgneb hiirekursorile. | | | | | veemaailm_kala_1 | | |
| fish2.png Kala2 liigub juhusliku arvu sekunditega juhuslikku kohta ekraanil.  Kasutades käsku [liigu ... sek. x: .. y: ..] ei muutu kahjuks spraidi suund.  veemaailm_kala2 | | | | | | | |
| fish4.png  Kala3 teeb ujudes ringe, kui hoida all klahvi p. | | kala3 | | | | | |
| veemaailm_merikarp | | | | clam1.png  Merikarp teeb paar hüpet, kui kasutaja vajutab tühikule.  Järjest tehtavate hüpete arvu võib määrata juhuslikult (1..5) | | | |
| octopus1-a.pngoctopus1-b.png  Kui Hai on puudutanud Kalakest, ärkab ka Kaheksajalas jahimeheinstinkt. Ta lõbustab ennast Kala2 hirmutamisega hüpates Kala2 asukohta.  Lisada kaheksajalg ja siis tema teine kostüüm, mida näidatakse hetkeks enne hüpet. | | | | | | | veemaailm_kaheksajalg |
| *veemaailm_vaal* | | | *whale1.gif*  Vaal ujub tagaplaanil aeglaselt paremalt vasakule, lava vasakus servas kaob mõneks ajaks ja ilmub uuesti nähtavale paremal (kui tema x-asukoht on väiksem, kui -240, peidetakse ta mõneks sekundiks ja x-asukoha uueks väärtuseks pannakse 240). | | | | |
| *starfish1-a.png*  Meritäht pöörleb ja liigub edasi-tagasi.  Muutuja **s** on liikumise samm, mis määrab liikumissuuna, see muutub servani jõudes vastupidiseks.  Muutuja lisamine: Muutujad – Tee Muutuja. Anda muutujale nimi. Võib olla lokaalne muutuja – ainult selle spraidi jaoks.  Tehted ja tingimuse plokid vt. Tehted. | | | | | veemaailm_taht | | |

**Lisaülesanded:**

* Kuidas saaks meritäht pöörelda selles suunas, kuhu ta liigub?
* Et hiirekursorit jälitav kala kursorit 'kätte saades' kummaliselt käituma ei hakkaks, tohib ta liikuda ainult siis, kui tema kaugus hiirekursorist on suurem, kui liikumissamm.
* Kalakese puutumise järel võiks Hai avada ähvardavalt suu (teine kostüüm).
* Kalakese kiirust võiks muuta (muutuja 'samm' liuguri abil).
* Kalake võiks suunda muuta ka siis, kui tema kaugus Haist on liiga väike (~50 sammu).
* Merikarbi hüppe kõrguse võiks valida juhuslikult.

# Efektid

Harjutus on Scratchi graafikaefektidega tutvumiseks.

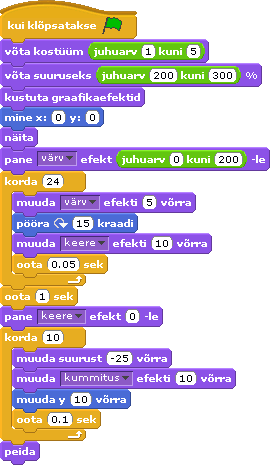
Luua paar suvalist erinevat spraiti ja proovida nende peal korraldusi efektide (värv, kalasilm, keere, pikselda, mosaiik, heledus, kummitus) ning omaduste suurus ja nähtavus muutmiseks. Efektide paremaks mõistmiseks võib koostada lihtsaid skriptikesi:

NB! efektide puhul võib muutus olla ka miinusmärgiga (-10); mõne efekti jaoks (näiteks pikselda) pole muutuse väärtusel tähtsust.

Lisada ühele spraidile mingi hulk kostüüme (pilte, ka tekste) ja koostada skript, kus valitakse spraidi jaoks juhuslik kuvand (kostüümi nr piirides 1 kuni kostüümide arv), määratakse sellele juhuslik suurus ja demonstreeritakse mitmeid õpitud efekte. Lõpuks liigub sprait kaugusse ja kaob.

# Nimi

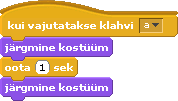
Harjutus on Scratchi liikumiskäskude ja graafikaefektide kordamiseks, skriptide käivitamisvõimaluste proovimiseks.

Luua spraidid tähtedega (kaust Letters) ja moodustada nendest oma nimi. Panna tähed liikuma (liuglema, pöörlema, hüppama) ja muutuma (värvuma, keerduma, kaduma jm). Projekti märkustesse kirjutada, kuidas mingid spraidid millelegi reageerivad.

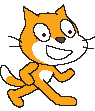
Skripte on võimalik käivitada:

* rohelise lipukesega
* klaviatuuri klahvidega
* hiireklõpsuga mõnel spraidil või laval
* reaktsioonina teatele, mille tekitab mõni teine skript.

Näiteid skriptidest:

Hiireklõps spraidil Kraps käivitab skripti, milles on (muuhulgas) teade ’laiali’, millele võivad reageerida suvaliste spraitide skriptid. Näiteks pärast Krapsu klõpsamist punuvad kõik tähed laiali.

nimi_laiali_T l

|  |  |
| --- | --- |
| m nimi_m_T r nimi__m | Hiireklõpsule tähega spraidil võib reageerida hoopis mingi teine sprait. |

|  |  |
| --- | --- |
| cat1-a  Kusagil võiks kasutajale olla ka info selle kohta, millisele krahvile või spraidile on mõtet vajutada. Näites jalutab Kraps ekraanil ja ütleb aeg-ajalt midagi kasulikku. | nimi_ytle |

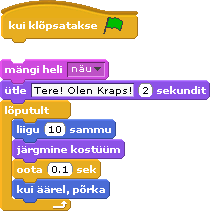
# Kraps, pall ja papagoi

Koostada rakendus (mäng) järgmise stsenaariumi alusel:

Kraps jalutab rahulikult palliga rannas edasi-­tagasi. Järsku ilmub taevasse papagoi, kes lendab ringi ja narrib Krapsu. Kraps üritab mängija abiga tabada palliga papagoid: palli lennutamiseks peab kasutaja vajutama tühiku klahvi. Mängu eesmärgiks on tabada papagoid võimalikult rohkem kordi etteantud aja jooksul (max\_aeg).

Rakendust arendada järk-järgult.

### Krapsu liikumine.

Liikumise ja välimuseplokid; skriptide käivitamine.

Esialgu võib Kraps liikuda ilma pallita.

Krapsul on kaks kostüümi. Liikumiseks koostada skript:

Määrata pööramise stiil ’ainult nägu vasakule/paremale’.

Kasutajal võiks olla võimalik juhtida Krapsu nooleklahvide abil.

Koostada skriptid, mis käivituvad vastavalt klahvidega ’vasak nool’ ja ’parem nool’.



### Papagoi tegevused

Spraitide ja kostüümide lisamine, skriptide kopeerimine ja redigeerimine, tingimused ja valikud, spraitide asukoha määramine, spraitide peitmine ja kuvamine, spraitide „ringliiklus“, juhuslike arvude kasutamine.

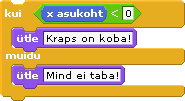
Lisada sprait **papagoi** ja selle teine kostüüm. Kopeerida **Krapsu** skript **papagoile** ja proovida.

Ringliikluse korraldamiseks muuta: kui papagoi jõuab lava parema servani, peita sprait (käsk[**peida**]) korraks (1-3 sek), viia lava vasakusse serva ja tuua nähtavale ([**näita**]).

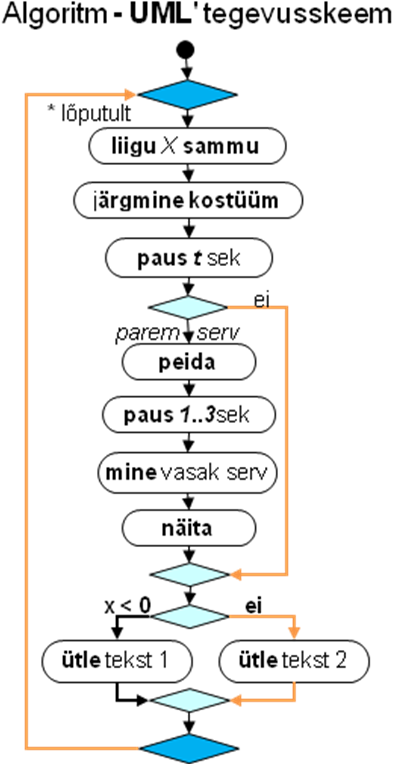
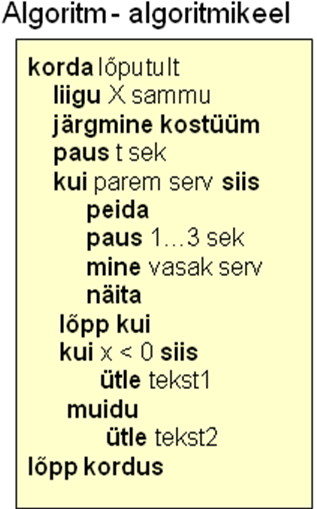
Pausi tekitamiseks kasutada **juhuarve**.

Et teha tabamine raskemas, võiks käsu [**pane x**-240] asemel võtta näiteks käsu [**pane x** juhuarv -240 kuni 120]. Papagoi ilmub nüüd nähtavale juhuslikus kohas antud vahemikus.

Lisada **papagoi** "jutt" (narrimine): **kui** asukoht < 0-st jutt1 **muidu** jutt2



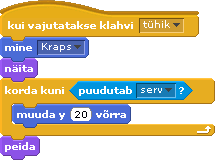
Tegevuste kirjeldamisel on juhendmaterjalis edaspidi kasutusel ka algoritmikeel ja UML tegevus­diagrammid. Võrrelda alltoodud algoritme papagoi skriptiga. Kas on sarnane?

### Palli lisamine ja selle tegevused.

Joonistamisredaktor, sündmus ’puudutab’, objekti viimine teise spraidi asukohta, spraidi kese, tingimuslik kordus, skripti peatamine, teadete saatmine ja vastuvõtmine**.**

Palli jaoks joonistada kujutis joonistamisredaktori abil: ring (shift-klahvi abil ellipsist) ja sellele mõned värvilised laigud. Spraidi (kostüümi) kese on tavaliselt keskel, võimalik määrata ka mujale.

Koostada tühikuga käivitatav skript palli lendamiseks Krapsu juurest üles lava servani.

**mine** Kraps

näita

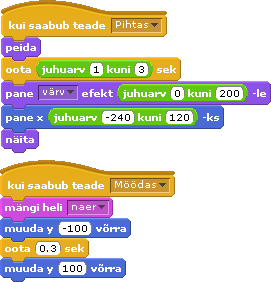
kordus kuni puudutab serv

**muudaY** h võrra

peida

|  |  |
| --- | --- |
| Nüüd tuleb kontrollida, kas pall tabab papagoid. Seda on võimalik teha nii palli kui papagoi skriptis. Otstarbekam lisada palli skripti, sest sel pole mõtet, kui pall ei lenda.  Tabamusest tuleks kindlasti teavitada (teade …), et ka teised spraidid (Kraps ja papagoi) saaks rea­geerida (ühe spraidi skript ei saa määrata tegevusi teiste spraitidega). Teavitama peaks ka mitte­tabamisest. | **mine** Kraps  näita  kordus kuni *serv*  **muudaY** h võrra  kui puudutab papagoi  peida  **teavita** Pihtas  peata skript  lõpp kui  lõpp kordus  peida  teavita Möödas |

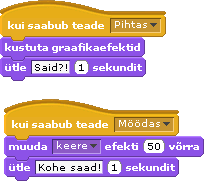
Kassi ja papagoi reaktsioonid:



**Papagoi**

**Pihtas**: papagoi peidetakse korraks, Kraps ütleb midagi

**Möödas**: papagoi naerab, laskub hetkeks allapoole, Kraps - mingid graafikaefektid (keere jm).



**Kraps**

### Visete ja tabamuste loendamine

Muutuja, omistamine

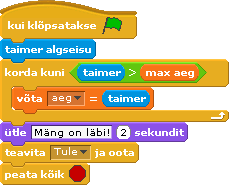
Luua kaks globaalset (nende väärtusi saavad kasutada ja muuta kõigi spraitide skriptid) muutujat loendamiseks: ***viskeid*** ja ***tabas***. Praegu on mõistlik teha nende muutujate väärtuste muutmiseks kas eraldi skriptid või lisada vastavad käsud palli skripti. Visete arv ***viskeid*** peaks muutuma palli lenduminekul ja tabamuste arv ***tabas*** pärast papagoi puudutamist.

**NB!** Muutujate algväärtustamine!

Korduval käivitamisel jätkatakse eelmisest väärtusest. Uue mängu algul tuleks loendurid nullida. Kui neid muutujaid palli skriptide juures juba kasutusel on, võiks ka algväärtustamised olla palli teises skriptis, mis käivitub mängu alustamisel (rohelise lipukesega).

### Aja kuvamine ja mängu lõpetamine.

Taimer, Käsk [**peata kõik**], liugur (kerimisriba) muutujate väärtuste määramiseks,“ teatespraidid“

Luua kaks muutujat: ***aeg*** (jooksev aeg) ja ***max\_aeg*** (mängu aeg). max\_aeg monitorile laval lisada **liugur** (kerimisriba), mille abil saab "käsitsi" muuta muutuja väärtust, ja määrata min ja max väärtused. Valida max\_aeg väärtuseks esialgu 10 (sek).

Skript taimeri nullimiseks, aja näitamiseks ja mängu lõpetamiseks võib olla suvalise spraidi või ka lava oma.

Taimeri plokid on grupis **Andurid.**

Kui klõpsatakse rohelist lippu

taimer algseisu

**korda kuni** *aeg* > *max\_aeg*

*aeg* = taimer

peata kõik

Mängu võib lõpetada ka lubatud visete arvu möödumisel. Luua muutuja **m\_viskeid**, lisada liugur (ka min ja max), valida sobiv väärtus.

Käsk ’oota kuni …’ seab skripti ooteseisu (kaudselt kordus).

Skripti võib panna suvalise spraidi moodulisse: näiteks lava või Kraps



Teadete, selgituste jmt kuvamiseks võib ploki **ütle** asemel kasutada nö "tekstispraite". Spraidil võib olla mitu kostüümi: igaüks neist kujutab omaette teadet. Vastavalt olukorrale võetakse sobiv kostüüm ja tehakse sprait nähtavaks.

Siin on kasutusel kaks teadet:

**Aeg on läbi!** (kostüüm Aeg)

**Visete arv on täis!** (kostüüm Visked)

Sõltuvalt lõpetamise põhjusest valitakse ja kuvatakse vastav teade.

### Palli liikumine koos Krapsuga. Olekumuutuja.

Pall peaks liikuma koos Krapsuga ja võiks ka pöörelda (NB! **Lisaülesanne** kiirematele). Sellise liikumiste sünkroniseerimise saab teha erinevalt. Näiteks võib Kraps iga kord pärast sammu astumist „kutsuda“ palli enda juurde.

Lisada teavitamise käsk [teavita *tule* ja oota] Krapsu põhiskripti.

Koostada palli jaoks skript, mis käivitub teatele **Tule**: viia pall **Krapsu** juurde ja alla (valida sobiv nihe) ning pöörata palli, arvestades Krapsu liikumise suunda.

**Mis juhtub, kui nüüd pall pannakse lendama? Tuleb tagasi!**

Võtta kasutusele nö olekumuutuja (tunnus) ***lendab****:*

* *k*ui **pall** lendab, on muutuja väärtus **1** muidu **0** (kokkulepe)
* muutuja väärtust muudab **palli** põhiskript
* algväärtus 0 (koos teiste muutujate algväärtustamisega).
* lisada Krapsu skripti valikuplokk kui lendab=0 teatega 'tule'.



### Rakenduse kirjeldus (dokumentatsioon)

Rakendusest arusaamiseks, vajadusel muutmiseks ja otstarbekaks kasutamiseks koostatakse alati ka projekti dokumentatsioon:

* kasutusjuhend – eesmärgid, tegevuste käivitamise võimalused
* spetsifikatsioon – selgitused spraitide ja nende skriptide (tegevuste) ning muutujate kohta.

Scratchis võib sellised kirjeldused kirjutada projekti märkustesse (vt menüüs Fail – projekti märkused).

Skriptide juures võivad olla kommentaarid - lühikesed selgitavad tekstid. (Loomiseks paremklõps skriptide alas, ilmuvast menüüst valida "**lisa kommentaar**"; saab muuta asukohta, siduda konkreetse käsuga jm.)

Kasutusjuhend (näiteks eraldi spraidina) võiks olla kättesaadav ka projekti kasutajaliideses.

Suuremate projektide korral võivad taolised märkused olla vormistatud eraldi dokumentidena.

Allpool on näitena toodud vaadeldava projekti (rakenduse) kirjeldus.

#### Kasutusjuhend

Kraps jalutab palliga edasi-tagasi. Papagoi lendab ringiratast ülalpool.

Eesmärk on tabada palliga papagoid võimalikult palju kordi valitud aja jooksul: max\_aeg (saab muuta). Programm näitab aega mängu algusest, tehtud visete ja tabamuste arvu. Visete arvu saab piirata muutuja m\_viskeid väärtusega.

Uue mängu algus: **Roheline Lipp.** Pall läheb lendu Krapsu kohalt tühikuklahvile vajutamisel.

#### Selgitused

**Põhispraidid**: Kraps, papagoi ja pall. Lisaks sprait **Teated**.

**Kraps** (2 kostüümi, 5 skripti). Jalutab palliga edasi-tagasi.

Põhiskript (käivitub rohelise lipuga) juhib Krapsu (ja ka palli) liikumist edasi-tagasi. Kui pall ei lenda, tuuakse see igal sammul Krapsu juurde (palli vastava skripti reaktsioon teatele **Tule** Krapsu skriptis).

Nooleklahvide (parem ja vasak) skriptid võimaldavad Krapsu pöörata ja liigutada vastavas suunas.

Palli teadete Pihtas ja Möödas toimel käivitatavad skriptid määravad reaktsioonid nendele sündmustele.

**Papagoi** (2 kostüümi, 3 skripti).

Põhiskript: juhib papagoi "ringliiklust" ja kuvab tekstid Krapsu narrimiseks.

Palli teadete Pihtas ja Möödas toimel käivitatavad skriptid määravad papagoi reaktsioonid nendele sündmustele.

**Pall** (1 kostüüm, 3 skripti).

Põhiskript: juhib palli lendu, kontrollib puudet papagoi või servaga. Kummagi sündmuse korral lend katkestatakse. Skript muudab ka muutujate viskeid, tabas ja lendab väärtusi. Muutujate algväärtustamine toimub eraldi skriptiga (käivitub rohelise lipuga). Krapsu põhiskriptis oleva teate Tule toimel käivituv skript viib palli Krapsu juurde ja pöörab veidi Krapsu loomise suunas.

**Lava** (1 taust, 2 skripti). Taustaks ranna pilt või ka mingi muu.

Üks skript muudab ja kontrollib taimeri abil aega. Kui aeg ületab muutuja max\_aeg väärtuse, initsialiseerib vastava teate kuvamise ja lõpetab kogu projekti töö. Teine skript kontrollib lõpetamist visete arvu järgi.

**Muutujad:**

***viskeid*** – tehtud visete arv; muudab palli põhiskript

***tabas*** – tabamuste arv; muudab palli põhiskript

***aeg*** – aja jooksev väärtus; muudab lava skript

***max\_aeg*** – lubatud mängu aeg, kasutaja saab valida (muuta) liuguri abil

***m\_viskeid*** – visete maksimaalne arv, valib kasutaja

***lendab*** – palli olekumuutuja: 1 - lendab, 0 - ei lenda; muudab palli skript

# Ristkülik

Koostada rakendus, mis arvutab ristküliku pindala ja ümbermõõdu suhte. Lisada animatsioon ’Matemaatikatunnis’, kus õpetaja küsib kasutajalt külgede pikkused ja seejärel „küsib“ õpilaselt vastused.

Kasutajaliides peaks välja nägema selline:

### Analüüs

Antud on ristküliku küljed (tähistame c ja d). Leida tuleb pindala S, ümbermõõt P, nende suhe ja ristküliku diagonaal. S = c \* d; P = 2(c + d); suhe = S / P; Diag =

### Disain

**Muutujad**: c, d, S, P, suhe, Diag; muutujate monitorid suurte näitudena.

**Spraidid**: Õpetaja, Õpilane, tekst1 (1. külg), tekst2 (2. külg), tekst3 (tulemused).

Skriptide loomisel võiks üldjoontes järgida sellist skeemi:

Õpetaja tegevused

Õpilase tegevused

arvuta pindala

arvuta ümbermõõt

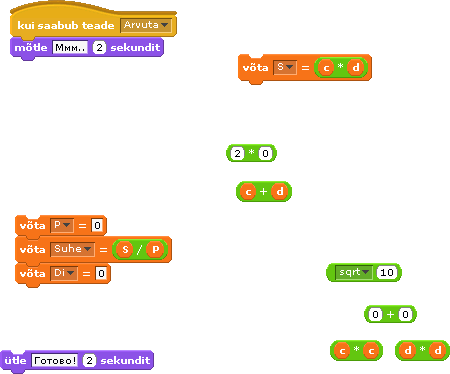
arvuta suhe

arvuta diagonaal

Kui teade *Arvuta*

ütle ***Valmis*** *2 sek*

mõtle ***Hmmm*** 2 sek



kustuta kõik vanad väärtused

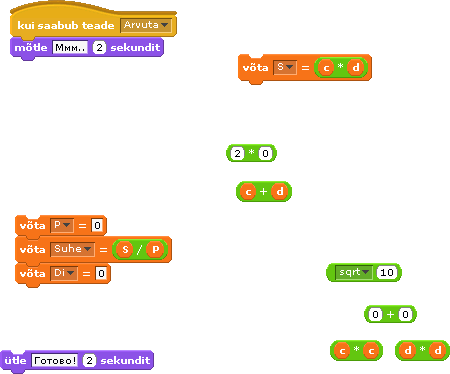
küsi c ja d väärtused ja omista need muutujatele

tervita õpilasi 2 sek

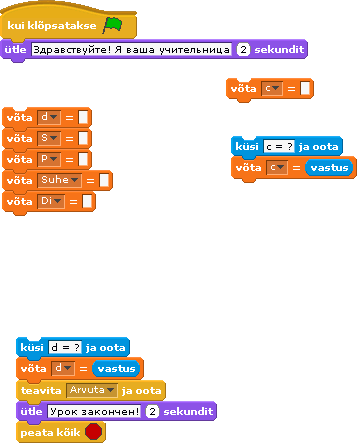
teade ***Arvuta*** ja oota

Ütle ***Tund on läbi*** *2 sek*

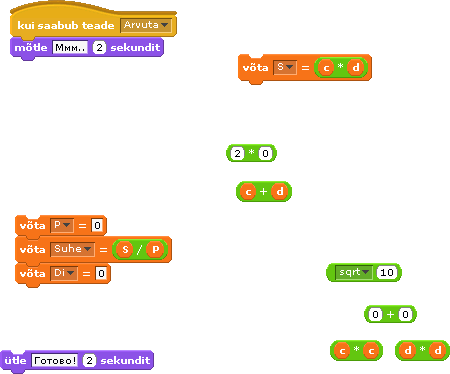
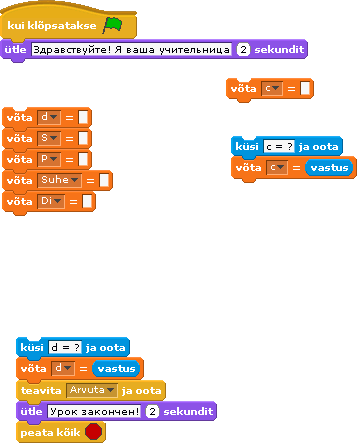
Peata kõik



veel tuleb küsida ***d***



ja kõik ülejäänud



# Ristkülik ja ring

Koostada rakendus, mis leiab etteantud ristküliku külgede abil sama suure pindalaga ringi raadiuse ja joonestab mõlemad kujundid.

### Analüüs:

Antud on ristküliku küljed **a** ja **b**. Leida tuleb ristküliku pindala **S**, mis on ühtlasi ka ringi pindala, ja ringi raadius **r**.

Tuletada meelde valemid ristküliku ja ringi pindalade leidmiseks. Kuidas leida ringi pindala järgi ringi raadius?

Ringi saab joonestada hulknurgana, kui **n** (nurkade arv) on piisavalt suur.

Polaarkoordinaatides on ringjoone iga [punkt](http://et.wikipedia.org/wiki/Punkt_(matemaatika)) üheselt määratud [kaugusega](http://et.wikipedia.org/wiki/Kaugus) fikseeritud punktist ([koordinaatide alguspunktist](http://et.wikipedia.org/w/index.php?title=Koordinaatide_alguspunkt&action=edit&redlink=1)) ning nurgaga fikseeritud suunast (nurk määratakse polaarteljest vastupäeva). Punktide koordinaadid saab leida valemitega:

**x = r · cos(nurk)**

**y = r · sin(nurk)**

Ringjoone joonistamisel anname muutujale **nurk n** väärtust vahemikus 0 … 360.

### Disain:

Vajalikud muutujad: **a**, **b**, **S**, **r**, **nurk**, **n**. Kõikide muutujate monitorid peaks olema näha ka laval. Muutujad **a** ja **b** saavad väärtuse rakenduse kasutamisel, muutuja väärtust saab kasutaja muuta kas liuguri abil või küsitakse need skripti abil (võib realiseerida erinevad variandid). **S** ja **r** tuleb arvutada, kui **a** ja **b** väärtused on teada. **Nurk** ja **n** on vajalikud ringi joonestamisel.

Objektid (spraidid): **rist** ristküliku joonestamiseks ja **ring** ringi joonestamiseks (nii saab moodustada mõlemad kujundid samaaegselt). Rakenduse ilmestamiseks võiks olla veel mingi tegelane, näiteks **Kraps**

**Kraps** küsib kõigepealt kasutajalt ristküliku külgede väärtused ja arvutab väärtused **s** ja **r**. Seejärel käivituvad kujundite joonistamise skriptid.

### Realisatsioon:

Luua muutujad **a**, **b**, **S**, **r**, **nurk**, **n** ja paigutada monitorid sobivatesse kohtadesse laval (muutujat **nurk** pole vaja näidata, **n** optimaalse väärtuse väljaselgitamiseks võiks esialgu võimaldada seda muuta liuguri abil). Luua spraidid **Rist**, **Ring**, **Kraps** (on niigi olemas). Määrata lava taustaks koordinaatteljestik ’xy-grid’.

Koostada skriptid (erinevatele spraitidele):

|  |  |  |
| --- | --- | --- |
| **Kraps**  Kui klõpsatakse rohelist lippu  kustuta varasem joonis  küsi a  küsi b  S = a\*b  r = sqrt(S/3,141592)  teavita joonista | **Ristkülik**  Kui saabub teade joonista  mine[–a/2; -b/2]  osuta suunas 90  pliiats alla  liigu a sammu ja pööra 90 kraadi  liigu b sammu ja pööra 90 kraadi  liigu a sammu ja pööra 90 kraadi  liigu b sammu  pliiats üles | **Ring**  Kui saabub teade joonista  nurk = 0  mine r; 0  pliiats alla  korda n korda  nurk=nurk+360/n  mine [r\*cos(nurk); r\*sin(nurk)]  pliiats üles |

### Lisaülesanded:

* Mõlema kujundi joonestamine võiks toimuda aeglaselt (jälgitavalt) ja lõppeda samal ajal.
* Kuidas saaks ringi ja/või ristküliku joonestada laval mingisse teise kohta?

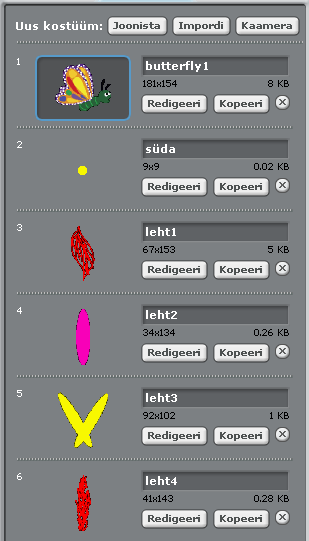
# Lilleaed

Koostada rakendus mis sarnaneks Scratchi demole *GardenSecret* näidete alamkaustas *InterActive Art*.

Hiireklõps laval käivitab tegevuse, mille tulemusena „kasvab“ hiirekursori asukohta lill: kõigepealt joonistatakse vars alates lava allservast kuni hiirekursorini, seejärel moodustub varre otsa õis. Lilled peaksid olema erinevad (juhuslikult valitud suurus, kuju, värv, õielehtede arv), erineda võiks ka nende ilmumisviis.

Objektid: varre joonistaja, õieleht (pöördub ja jätab jäljendi), südamik.

Kui lillede ilmumisviisis pole plaanis väga suuri erinevusi, võivad need olla ka ühe spraidi erinevad kostüümid: üks vars, üks südamik ja mitu õielehte, mille jaoks valitakse iga kord erinev värv ja suurus, ning kogu tegevus on ühes skriptis või on erinevad tegevused igaüks eraldi skriptis (nii on neid mugavam testida) ja skript valitakse juhuslikult.

Keerulisemate erisuste jaoks (kui lilleõis moodustub erinevast hulgast erinevatest kujunditest), on parem luua mitu spraiti (üks iga õiekomplekti jaoks).

Tegevused varre joonistamiseks:

Mine [hiire x; -180]

Võta lillevarrejoonistamiskostüüm

Võta sobiv suurus

Vali pliiatsi värv ja suurus

Pliiats alla

Liigu [hiire x; hiire y]

Pliiats üles

Tüüptegevused õie joonistamiseks:

Vali sobiv kostüüm, suurus ja värv (juhuslikult)

Võta <arv> = juhuarv (mõistlikes piirides)

Korda <arv> korda

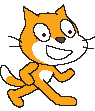
Jäta jälg

Pööra 360/<arv>

# Juhtimisplokid

Harjutus tutvumiseks protsesside juhtimisega (valik, kordus) programmides ning ülevaate saamiseks Scratchi juhtimisplokkidest ja skriptide käivitamisest.

Tegelased: kassike Kraps, kummitus Kaspar, lillevaas ja väike vihane lillehaldjas Piksi.

Kassike Kraps (kaks tavapärast kostüümi) soovib saada kummitus Kaspari sõbraks, jookseb tema suunas ning hüpleb (lihtsam) või teeb kukerpalli (keerukam), kui jõuab piisavalt lähedale.

Kummitus Kasparit võiks juhtida nooleklahvidega. Kui kassike trügib liiga lähedale, muutub ta tasapisi läbipaistvaks ja läheb kiiresti mujale.

Laval on lillevaas, mis läheb ümber, kui Kraps (või Kaspar) seda puudutab.

Lillehaldjas Piksi, kes lendleb mesilasena ümber vaasi, pahandab vaasi ümberajaja peale, muutub pisikeseks vihaseks nõiaks ja hakkab süüdlast jälitama.

Kogu tegevus kestab ettemääratud aja jooksul.

#### Käivitamine:

Skripte on võimalik käivitada rohelise lipukesega lava ülanurgas (Kraps), nooleklahvide abil (Kaspar), hiireklõpsuga spraidil (Lilled) või reaktsioonina teatele, mille käivitab teine skript.

|  |  |  |  |
| --- | --- | --- | --- |
| cat1-aj_lipp_ | ghost1juhtimine_kaspar_nool | flower-vasejuhtimine_lilled_klops | j_teade_ |

Teateid, millele teised skriptid saavad reageerida, on kahesuguseid: j_teavita1 ja j_teavita2.

Käsu [teavita *...***]** korral jätkub antud skripti täitmine kohe, teatega käivituvad skriptid ja teavitaja (välja­kutsuja) töötavad paralleelselt.

Käsk [teavita *...* ja oota**]** peatab antud skripti töö ja see jätkub alles siis, kui on lõpetanud kõik skriptid, mis teatele reageerisid.

#### Valikud:

Hargnevad protsessid.

|  |  |  |  |
| --- | --- | --- | --- |
| j_kui | Sõltuvalt tingimuse väärtusest täidetakse valikuplokis olevad käsud või jäetakse vahele (valik ühest – täidetakse/ei täideta) | j_kui_muidu | Kahest võimalikust käsugrupist valiku­plokis täidetakse üks, teise käsud jäetakse vahele (valik kahest, täidetakse üks või teine käsugrupp). |

|  |  |
| --- | --- |
| j_kui_n | j_kui_muidu_n |

#### Kordused:

Tegevuste kordamist (tsükliline protsess) on selles näites vaja osaliste liikumise tekitamiseks. Kordused on **etteantud korduste arvuga kordus** (Kraps hüppab 3 korda), **lõputu kordus** (Kraps liigub Kaspari suunas) ja **eelkontrolliga kordus**, mis kestab kuni etteantud tingimus on täidetud.

|  |  |  |
| --- | --- | --- |
| j_kordaj_korda_n  j_korda_kuni | j_korda3 | cat1-aj_loputult |
| juhtimine_aeg | Muutuja **aeg** väärtuseks võetakse **taimer**i väärtus kuni ette­antud aeg otsas; seejärel lõpeb kõigi rakenduse skriptide töö. | |

#### Muud juhtimisplokid Scratchis:

|  |  |  |
| --- | --- | --- |
| j_oota | tekitab skripti töös antud pikkusega katkestuse (pausi); sobib kasutada tegevuste (spraitide liikumise/muutumise) kiiruse määramiseks. | |
| j_oota_kuni | Kontrollitakse lõputult tingimust; kui tingimus on tõene, täidetakse järgnevad tegevused. Selle asemel oleks võimalik kasutada lõpmatut kordust, milles on kui-plokk. Kui tingimus on tõene, täidetakse kui-ploki käsud, siis lõpetatakse töö. | **Kordus lõputult**  **Kui** tingimus tõene  ……  **Peata skript**  **Lõpp kui**  **Lõpp kordus** |
| j_korda_kui | Tingimuslik kordus. Plokis olevaid tegevusi täidetakse, kui tingimus on tõene. Tingimust kontrollitakse lõputult. Selle asemel võiks olla lõpmatu kordus, milles on kui‑plokk. | **Kordus lõputult**  **Kui** tingimus tõene  ……  **Lõpp kui**  **Lõpp kordus** |

|  |  |  |
| --- | --- | --- |
| **Kaspar** liigub laval nooleklahvide abil (võib lisada eraldi skripti, et ta liiguks pidevalt aeglaselt ja põrkuks seintelt).  Kõrvalolev skript viib Kaspari juhus­likku kohta ekraanil, kui Kraps jõuab häirivalt lähedale. Kasutusel on korduseplokk [lõputult kui ...], mis võimaldab „jälgida“ olukorda ja tegutseda siis, kui tingimused on sobivad, seejärel „valvab“ edasi, et tegutseda sobivas olukorras uuesti. | ghost1j_Kaspar_alg | |
| Kraps ja Kaspar võivad ümber ajada **lilled**. Lillevaas peaks seatama püsti rakenduse töö alguses. Hiljem võib kasutaja mahaaetud lillevaasi püsti tõsta sellel klõpsates (roheline lipp ja hiireklõps spraidil käivitavad mõlemad sama tegevuse, mis on vormistatud eraldi skriptina).  Mahaaetud lillevaasi veel kord ümber ajada ei saa, seepärast on kasutusel plokk [**oota kuni ...**]  Sprait **lilled** omistab pahategija nime globaalsele muutujale **lõhkuja**. Muutujat, mille väärtuseks on spraidi nimi, saab mitmetes käsuplokkides kasutada spraidile viitamiseks.  Lilled kukuvad õiges suunas, arvestades seda, kuspoolt lõhkuja lähenes.  Teatele [**teavita** appi] peaks reageerima Piksi (võivad ka teised). | | j_lilled_alg j_lilled_klops  flower-vasej_lilled_pysti |

Lillehaldjas Piksi lendleb mesilasekostüümis ümber lillede, kui klaviatuuril on vajutatud tühik (veel üks ploki [**lõputult kui** ...] kasutusviis).

Lillevaasi võib asetada ka mõnele teisele riiulile, Piksi tuleb alati lillede juurde tagasi.

Kui lillevaas ümber lükatakse (teade appi), muutub Piksi vihaseks nõiaks (vahetab kostüümi) ja hakkab pahategijat sõimeldes jälitama nii kaua kui selle kätte saab (**eelkontrolliga kordus**); sajatused lõpevad alles pärast seda.

|  |  |
| --- | --- |
| j_Piksi_alg | j_Piksi_appi1 |

Sama spraidi erinevaid kostüüme saab kasutada objekti muutumiseks. Näiteks auto ja avarii, pomm ja plahvatus jm.

Kui liikumise simuleerimiseks on kasutusel mitu kostüümi, tuleb sama tegelase jaoks formeerida kaks spraiti, mis vahetavad sobival hetkel teisega kohad.

Peategelane Kraps liigub kogu rakenduse töö jooksul Kaspari suunas (vt. skript eespool). Kui ta jõuab piisavalt lähedale, üritab ta Kasparit mängima meelitada hüpates (vt. skript eespool) või kukerpalli tehes (pööreldes). Pöörlemise võiks Krapsu eest ära teha dublant – luua Krapsu koopia Kraps1.

Programmi abil pole võimalik juhtida spraidi omadust, mis määrab, kas sprait saab pöörelda 360° või ’pöördub ainult nägu vasakule/paremale’.

|  |  |
| --- | --- |
| Lahenduseks võib olla kahe sarnase spraidi kasutamine: Kraps ei saa pöörelda, Kraps1 saab. Pöörlemiseks vahetub Kraps mõneks ajaks Kraps1-ga (nii nagu üks kostüüm vahetub teisega).  cat1-a  j_Kraps_teade | cat1-acat1-aj_Kraps1_poorle |

# Autod

Ülesanne: koostada rakendus kolme auto võidusõidu imiteerimiseks. Kohtunik (näiteks Kraps) teatab sõidu alustamisest ja kuulutab lõpuks välja võitja. Autode kiiruse (sammu pikkuse) võiks valida juhuslikult, et tulemus poleks ette määratud. Kasutaja peaks saama ette anda sõidetavate ringide arvu.

Iga auto puhul peab olema teada sõidetava ringi number ja sõiduaeg, need võiks olla auto omadused – lokaalsed muutujad, mida saab muuta ainult selle spraidi skriptides (lugeda lokaalse muutuja väärtust saab ka teistes skriptides).

|  |  |  |
| --- | --- | --- |
| Protseduur **Start**  x = -230  ring = 0  **Kordus** ringe **korda**  **Kordus kuni** x > 240  **Liigu** 10 kuni 15 sammu  **Lõpp kordus**  **Peida**  **Oota** 1 kuni 3 sek  x = -230  **näita**  aeg = taimer  ring = ring + 1  **Lõpp kordus** | Kõikide autode tegevused on ühesugused, seega saab ühest autost pärast skriptide loomist teha vajaliku arvu koopiaid ja seejärel muuta veidi nende välimust.  Algul võiks kõigi muutujate väärtused olla laval näha, et rakenduse tööd oleks parem kontrollida ja võimalikke vigu lihtsam leida. Hiljem võib osa väärtusi peita.  Kõik autod sõidavad nõutud arvu ringe ja peatuvad siis. Lokaalses muutujas **aeg** salvestatakse iga auto lõppuaeg. | |
| Kraps juhatab võidusõitu, tema skriptist tulevad ka teated teiste tegevuste käivitamiseks.  Korraldus [**Teavita** Start **ja oota**] tagab tegevuste jätkamise alles pärast seda, kui kõik autod on sõitmise lõpetanud.  Keerukamad tegevused/arvutused võiks vormistada eraldi skriptidena. | | Protseduur **Kraps**  võitja = ’’  **taimer algseisu**  **Teavita** Start **ja oota**  **Teavita** Kes\_võitis1 **ja oota**  **Mine** võitja  **Ütle** ’Võitja on tubli’ |

Alternatiivne protseduur **kes\_võitis2** kasutab lisaks muutujat **min** väikseima sõiduaja meeles­pidamiseks.

|  |  |  |
| --- | --- | --- |
| Protseduur **kes\_võitis1**  **Kui** (auto1.aeg < auto2.aeg) **ja** (auto1.aeg < auto3.aeg)  võitja = auto1  **Muidu**  **Kui** auto2.aeg < auto3.aeg  võitja = auto2  **muidu**  võitja = auto3  **lõpp kui**  **Lõpp kui** |  | Protseduur **kes\_võitis2**  min = auto1.aeg  võitja = auto1  **Kui** auto2.aeg < min  min = auto2.aeg  võitja = auto2  **Lõpp kui**  **Kui** auto3.aeg < min  min = auto3.aeg  võitja = auto3  **Lõpp kui** |

Lisaks:

* auto kiirus võiks olla spraidi omadus (lokaalne muutuja), mis määratakse (juhuslikult) võistluse alguses
* Kraps võiks rohkem esineda: kamandada kõigepealt võistlejad kohale, lõpus liikuda võitja juurde jmt
* Lisada võistlusele veel üks auto

# Rehmaatika

Koostada rakendus arvutamise harjutamiseks. Kuvatakse tehteid liitmine-lahutamine ja kasutaja peab võimalikult kiiresti sisestama vastuseid.

Arvud (liidetavad) ja tehe tekitatakse juhuarvude abil etteantud piirides min...max. Et rakendus sobiks kasutamiseks algklassiõpilaste jaoks, ei tohiks lahutamistehe tulemus olla negatiivne arv.

Kasutaja peab saama valida taset, millest sõltub tehetes kasutatavate arvude suurus ja punktide arv ühe tehte kohta ning katse aega.

Lisada võib ka sobiva animatsiooni.

Tegevuste üldine skeem on järgnevalt toodud UML tegevusdiagrammidena ja ka algoritmikeeles.

**Muutujad:**

**a** ja **b** – juhuarvud vahemikus min...max,

**c** – a ja b summa (a + b)

**tehe** – juhuarv: 1 - liitmine, 2 - lahutamine

**tulem** – õige vastus: c (kui liitmine) või b (kui lahutamine)

**vastus** – kasutaja vastus

**punkte** – punktide jooksev ja lõplik summa

**tase** – ülesannete tase, võtame 3 taset

**min** ja **max** – min ja maks väärtus, sõltuvad tasemest näiteks: 1. tase: 1...10, 2. tase: 5...20, 3. tase: 10...50

**p** – punkte ühe tehte eest, sõltub tasemest, näiteks: 1. tase: p = 2, 2. tase: p = 5; 3. tase: p = 10

**max\_aeg** – katse maksimaalne aeg, laseme valida kasutajal, näiteks vahemikus 20 ... 60 sek

**aeg** – jooksev aeg katse lõpuni: max\_aeg – taimer

|  |  |
| --- | --- |
|  | **Algoritm** (algoritmikeel)  **protseduur** **Liida\_lahuta**  punkte = 0  **kutsu** **Tee\_tase**  **kordus lõputult**  a = **juhuarv**(min, max)  b = **juhuarv**(min, max)  c = a + b  tehe = **juhuarv**(1, 2)  **kui** tehe = 1 **siis**  **kuva** a & “ + “ & b  tulem = c  **muidu**  **kuva** c & “ - “ & a  tulem = b  **lõpp kui**  **loe** vastus  **kui** vastus = tulem **siis**  punkte = punkte + p  **muidu**  **kuva** “Vale!“  **lõpp kui**  **lõpp kordus** |

Rakendust võiks hakata ehitama järk-järgult:

* tehte koostamine
* tehte vastuse küsimine, vastuse kontroll (esialgu ei pruugi punktiarvestust toimuda)
* raskusastme valik ning miinimumi, maksimumi, ühe tehte eest antavate punktide muutmine vastavalt valitud tasemele.

Realiseerimisel võib kasutada lisaks abimuutujaid näiteks küsimuse (tehte teksti) säilitamise jaoks.

|  |  |
| --- | --- |
| Raskusastme kohandamine vastavalt eelnevalt määratud tasemele | |
| Variant1 | |
| min = 1; max = 10; p = 2  min = 5; max = 20; p = 5  tase =1  tase =2  min = 10; max = 50; p = 10  tase =3  ei  Tegevusskeem - **UM**L    ei  ei | **protseduur Tee\_Tase**  **kui** tase = 1 **siis**  min = 1; max = 10  p = 2  **kui** tase = 2 **siis**  min = 5; max = 20  p = 5  **kui** tase = 3 **siis**  min = 10; max = 50  p = 10  **Algoritmikeel**    Kõiki kolme tingimust kontrollitakse ka siis,  kui tase on 1. Parem on variant 2. |
| Variant2 | |
| tase =1  tase =2  ei  ei  min = 1  max = 10  p = 2  min = 5  max = 20  p = 5  min = 10  max = 50  p = 10 | **protseduur Tee\_Tase**  **kui** tase = 1 **siis**  min = 1; max = 10; p = 2  **muidu**  **kui** tase = 2 **siis**  min = 5; max = 20; p = 5  **muidu**  min = 10; max = 50; p = 10  **lõpp kui**  **lõpp kui** |

Rakenduse kirjeldus lisada projekti märkustesse (Fail – Projekti märkused).

Lisaülesanded:

* Loendada ka tehete ja õigete (ja valede) vastuste arv
* Väikseima ja suurima liidetava asemel võiks olla ette antud väikseim liidetav ja suurim summa.
* Lisada ka korrutamise ja jagamise tehted.
* Vale vastuse puhul võiks sama tehet korrata.
* Raskusaste (tase) võiks punktisumma kasvades suureneda automaatselt.

# Ideaal

Koostada rakendus inimese keha omaduste leidmiseks.

Inimese soo (mees/naine), vanuse *t* (aastates), pikkuse *l* (cm) ja kaalu *m* (kg) alusel tuleb alltoodud valemite abil arvutada ideaalne mass *mid* (kg), rasvaprotsent *r*, kehamassi indeks *kind*, tihedus *ρ* (kg/m3), ruumala *V* (dm3), pindala *S* (m2) ning anda kehamassi indeksi alusel sõnaline *hinnang*.

**NB!** Astendamist Scratch ei tunne, aga on võimalus eksponendi ja logaritmi leidmiseks ning 

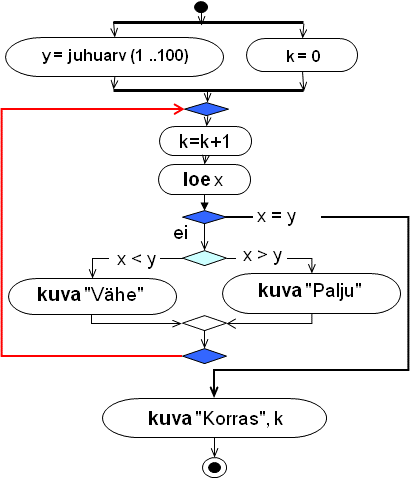
* Keerulisemate avaldiste väärtuste leidmisel on otstarbekas kasutada abimuutujaid.
* Muutujaid võiks näidata laval alles siis, kui neile on saadud väärtused kas kasutajalt või arvutuste teel.
* Lisada sobiv animatsioon.

Saadud tulemust võib võrrelda alloleva tabeli andmetega.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sugu** | N | M | M | N |  |
| **Vanus** | 22 | 20 | 60 | 37 |  |
| **Pikkus** | 165 | 180 | 175 | 160 | cm |
| **Kaal** | 48 | 70 | 90 | 78 | kg |
| **Ideaalne mass** | 55,6 | 72,5 | 78,8 | 55,6 | kg |
| **Rasvaprotsent** | 6,2 | 11,4 | 27,5 | 50,8 | % |
| **Kehamassiindeks** | 17,6 | 21,6 | 29,4 | 30,5 |  |
| **Tihedus** | 1086,9 | 1076,0 | 1042,3 | 993,4 | kg/m3 |
| **Ruumala** | 44,2 | 65,1 | 86,4 | 78,5 | dm3 |
| **Pindala** | 1,5 | 1,9 | 2,1 | 1,9 | m2 |
| **Hinnang** | kõhn | normaalne | ülekaalus | ei või olla! |  |

# Arvu arvamine

Koostada skeemi järgi rakendus, mis võimaldaks kasutajal ära arvata vahemikus 1…100 juhuslikult valitud arvu, loendada tuleb ka katsete arv.



Rakenduse täitmisel tuleb anda kasutajale piisavalt infot (mida ta peaks sisestama, kas varemsisestatud arv oli liiga väike või suur, mitu arvamiskatset on tehtud jm). Rakenduse kirjeldus lisada projekti märkustesse (Fail – Projekti märkused).

### Lisaülesanded:

* Lisada rakendusele mingi asjakohane animatsioon
* Kui vahemikus 1..100 genereeritud arv õnnestub kasutajal ära arvata kuni seitsme katsega, kuvada vastav teade või/ja käivitada mingi lisategevus (animatsioon).
* Lisada võimalus juhusliku arvu genereerimiseks mingis teises etteantud arvuvahemikus. Kasutajat tuleb min ja max väärtusest teavitada kas arvamise alguses või iga küsimuse korral.

# Labürint

Koostada PacMan’i laadne mäng, kus rakenduse peategelane, keda kasutaja saab juhtida noole­klahvidega, korjab labürindis kullakange. Teda jälitab koll. Pacman läbi seinte ei saa, aga koll saab. Rakenduses tuleb leida kogutud kulla maksumus (kullakangide maksumused võivad olla erinevad). Mäng lõpeb, kui kogu kuld on kätte saadud või ületatakse etteantud aeg või koll saab Pacmani kätte.

### Analüüs:

#### Objektid:

**Labürint** – joonistatakse lava taustana; ühte värvi seinad, millest Pacmani läbi ei lasta

**Pacman** – võib luua joonistusredaktoriga, kaks (või rohkem) kostüümi, mida vahetatakse liikumisel. Liigutamine nooleklahvidega (4), ei tohi minna läbi seinte (kui puudutab seina värvi, liikumine peatub).



**Kullakangid** – luuakse joonistamisredaktoriga, omadus ’hind’ võiks olla lokaalne muutuja; koostatakse vajalikud skriptid, paljundatakse ühest valmis spraidist vajalik hulk objekte, paigutatakse labürinti laiali, iga kullakangi omadusele ’hind’ antakse erinevad väärtused. NB! Kulla hulk sõltub arendaja suvast, seetõttu on mõistlik, et kulla skriptis kontrollitakse, kas see sprait puudutab Pacmani

**Koll** – jälitab Pacmani (st. liigub kogu aeg Pacmani poole); kui tabab (puudutab), siis lõpeb mäng.

**Teade** – sprait, mille kostüümideks on teated (’Kogu kuld leitud’, ’Koll sai kätte’, ’Aeg läbi’). Selle (või lava) juurde võib koondada kõik üldised skriptid (muutujate algväärtustamised, ajaarvestus, kullaarvestus)

#### Muutujad:

***hp*** – Pacmani liikumise samm (määratakse ja kasutatakse Pacmani skriptides)

***hk*** – kolli liikumise samm (võib muuta kolli skriptis sõltuvalt tasemest või kogutud kullast)

***max\_kuld*** – otsitavate kullakangide arv. Väärtus omistatakse rakenduse loomise ajal.

***alles* –** alles olevate kullakangide arv (algväärtus ***max\_kuld***), muudavad kullakangide skriptid

***summa*** – kogutud kulla väärtuse jooksev summa (algväärtus 0), muudavad kulla skriptid

***max\_aeg*** – mängu max. aeg, määratakse rakenduse loomise ajal (või määrab kasutaja mängu alguses)

***aeg*** – mängu lõpuni jääv aeg: ***max\_aeg* – taimer** (algväärtus ***max\_aeg*)**

#### Võimalikud lõpuolukorrad: (seejärel peatatakse kõik skriptid)

Tehtud – kui kogu kuld on korjatud (alles = 0). Pacman rõõmustab, kuvatakse teade ’Kogu kuld leitud!’

Käes – teatab koll, kui Pacmani kätte saab. Koll rõõmustab, Pacman kurb, kuvatakse ’Koll sai kätte’

Aeg läbi – kui taimer > max\_aeg. Kuvatakse ’Aeg läbi!’

#### Olulisemad algoritmid:

|  |  |  |
| --- | --- | --- |
| **Kui vajutatakse nooleklahvi**  **Osuta suunas** {90; -90; 0; 180}  **Liigu** hp sammu  **Järgmine kostüüm**  **Kui puudutab** värvi (sein)  **Pööra** 180  **Liigu** hp **sammu**  **Lõpp kui** | **Pacman**  Pacmani nooleklahvi­skriptidel on üsna suur ühisosa, mis oleks mõistlik vormistada eraldi skriptina. | **Kui vajutatakse nooleklahvi** **…**  **Osuta suunas** {90; -90; 0; 180}  **Teata** **tee samm**  **tee samm**  **Liigu** hp sammu  **Järgmine kostüüm**  **Kui puudutab** värvi (sein)  **Pööra** 180  **Liigu** hp **sammu**  **Lõpp kui** |

Kulla ja kolli skriptid ning algseaded ja ajaarvestus (ükskõik millise spraidi juures) käivituvad mängu alguses.

|  |  |
| --- | --- |
| **Protseduur** **ajaarvestus**  **taimer algseisu**  **Kordus** **kuni** taimer > max\_aeg  aeg = taimer  **Lõpp kordus**  **teata** Aeg läbi! **ja oota**  **Peata kõik** | **Protseduur algseaded**  alles = max\_kuld  summa = 0  **oota kuni** alles = 0  **teavita tehtud**  **Peata kõik** |
| **Protseduur** **koll**  **Kordus** **kuni** puudutab Pacman  **osuta** Pacman  **liigu** hk sammu  **oota** …  **Lõpp kordus**  **teata** Käes! **ja oota**  **Peata kõik** | **Protseduur** **kuld**  **näita**  **Oota kuni** puudutab Pacman  summa = summa + hind  alles = alles -1  **peida** |

Rakenduse kirjeldus lisada projekti märkustesse (Fail – Projekti märkused).

Lisaülesanded:

* liikuvad uksed ja muud takistused
* rohkem kolle erineva käitumisega
* mängul võiks olla mitu taset (kolli liikumine muutub kiiremaks)
* Pacman liigub ise, nooleklahvidega saab muuta vaid suunda
* kui mäng ei käi, ei tohi Pacman liikuda.

# Tagurpidi

Koostada rakendus kasutajalt küsitud teksti esitamiseks tagurpidi.

* sümbolid vastupidises järjestuses
* sõnad vastupidises järjestuses.

Tekstist on võimalik eraldada sümboleid t_symbol, tekste saab ühendada t_yhenda, saab teada teksti pikkuse (sümbolite arvu) t_pikkus.

|  |  |
| --- | --- |
| Ühest tekstist tuleb eraldada sümbolid (korrata iga sümboli jaoks, alustada võib nii teksti algusest kui lõpust) ja lisada teise teksti nii, et järjestus oleks vastupidine.  Andmeteks (säilitamist vajavad väärtused) on: kasutajalt küsitud tekst **tekst1**, tulemus (koostatav tekst) **tekst2**, jooksva sümboli järjenumber **nr**. Muutujateks võiks olla ka teksti pikkus **mitu** ja jooksev sümbol **s**. | **Protseduur Tagurpidi1**  mitu = pikkus(tekst1)  tekst2 = ’’  nr = mitu  **Kordus** mitu **korda**  s = tekst1 nr-s täht  tekst2 = ühenda tekst2 ja s  nr = nr – 1  **Lõpp kordus** |

Teksti sõnade kaupa töötlemisel oleks hea eraldada tekstist sõnad (salvestada iga sõna eraldi muutujas). Sõnade eraldajaks on tühik. Sõnade arv tekstis ei ole teada, see on muutuv. Probleemi saab lahendada loendi kasutamisega. Loend – muutujate järjestatud kogum.

|  |  |
| --- | --- |
| Muutujaplokkide juures on nupud ’Tee loend’ ja ’Eemalda loend’. Nii nagu teisedki muutujad, tuleb Scratchis tekitada loendid rakenduse loomisel. Programselt saab väärtusi lisada korraldusega l_lisa ja eemaldada käsuga l_eemaldakoik (kas kõik või ükshaaval järjenumbri abil). Teada saab väärtuste arvu loendis l_pikkus ja etteantud järjenumbriga elemendi väärtuse l_el_nr.  Teksti jaotamine sõnadeks (enne luua loend **sõnad**):  Tekstist **tekst1** eraldatakse sümbol **s** (korrata iga sümboli jaoks). Iga kord kontrollitakse, kas eraldatud sümbol on tühik. Kui tegemist pole tühikuga, lisatakse see **sõna**-le. Kui eraldatud sümbol on tühik, siis seda ei lisata vaid muutuja **sõna** väärtus lisatakse loendile **sõnad** ja **sõna** tühjendatakse. Lõpuks lisatakse loendile veel kord **sõna** väärtus (sümbolid pärast viimast tühikut). | **Protseduur Tagurpidi2**  mitu = pikkus(tekst1)  tekst2 = ’’  sõna = ’’  nr = 1  **Kordus** mitu **korda**  s = tekst1 nr-s täht  **Kui** s = ’ ’  Lisa sõnad-le sõna  sõna = ’’  **Muidu**  sõna = ühenda sõna ja s  **Lõpp kui**  nr = nr + 1  **Lõpp kordus**  tekst2 = sõnad(1)  nr = 1  **Kordus** pikkus(sõnad)-1 **korda**  Nr = nr + 1  Tekst2 = ühenda sõnad(nr) ja (ühenda ’ ’ ja tekst2)  **Lõpp kordus** |

Rakenduse kirjeldus lisada projekti märkustesse (Fail – Projekti märkused).

**NB!** Et mõlemad protsessid võivad töötada paralleelselt, võiks abimuutujate (nr, s) sassiminemise vältimiseks kasutata erinevate spraitide lokaalseid muutujaid.

# Autod 2

Veel kord võidusõit.

Koostada kolme auto võidusõidu sarnane rakendus. Seekord ei ole osalevate autode arv piiratud.

Kohtunik teatab sõidu alustamisest ja kuulutab lõpuks välja võitja.

Enne võistluse algust võiks kasutaja ennustada, kes võidab. Kui võistlus on lõppenud, tuleb teatada, mitmendana lõpetas kasutaja poolt võitjaks ennustatud auto.

|  |  |
| --- | --- |
| **Protseduur** **Start**  x = -230  **Kordus** ringe **korda**  **Kordus kuni** x > 240  **Liigu** 10 kuni 20 sammu  **Lõpp kordus**  **Peida**  **Oota** 1 kuni 3 sek  x = -230  **näit**a  aeg = taimer  **Lõpp kordus**  **Lisa loendisse** | Autode kiirus (sammu pikkus) valitakse juhuslikult igal sammul või määratakse juhuslikult sõidu algul.  Võistluse lõpetanud auto nimi ja aeg kirjutatakse vastavatesse loenditesse.  Võistluse võitja nimi ja aeg on loendites esimesed.  **Andmed:** objektid **autod** ja **kohtunik**, loendid **Ajad** ja **Autod**, muutujad **ringe**, **ennustus**, **j** (järjenumber); iga auto lokaalsed muutujad: **nimi**, **ring**, **kiirus** |
| j = 1  **kordus** Autod.pikkus **korda**  **kui**  ennustus = Autod(j) **siis**  **stopp**  **lõpp kui**  j = j +1  **Lõpp kordus** | Kasutaja poolt võitjaks ennustatud auto leidmine võistluse lõppjärjestuses. Selle võiks vormistada eraldi skriptina (sees on katkestus; Scratch ei võimalda korduse katkestamist, seega peatatakse skripti töö). |

### Lahendamist vajavad:

* Kuidas kasutaja ennustab võitja (kohtunik küsib ja kasutaja vastab; hiireklõps valitud autol; vm)
* Kuidas kuulutatakse välja võitja (kohtunik ütleb võitja nime; kohtunik läheb võitja juurde; võitja teeb auringi, pöörleb või hüpleb; vm)
* Kuidas teatatakse kasutaja võitjaks ennustatu koht ja aeg; mis saab, kui ennustus osutus õigeks.

# Vestlus

Koostada rakendus kahe tegelase dialoogi esitamiseks. Eelnevalt moodustada kaks loendit, kuhu salvestada osaliste tekstid.

### Kirjeldus:

Dialoogi alustab üks osalistest öeldes esimese teksti oma loendist, teine vastab oma loendi esimese repliigiga. Tegevust korratakse nii kaua, kui kõik tekstid on ära öeldud. Kasutada tuleks muutujat järjenumbri meelespidamiseks. Ühe spraidi skriptist algab tegevus, teise spraidi teksti esitamiseks väljastatakse teade ja oodatakse, kuni vastus on näidatud.

**Andmed:** kaks tegelast (spraiti), loendid tekstid1 ja tekstid2, j (loendur).

Esialgu võib kõiki tekste kuvada sama aja jooksul, hiljem proovida teksti kuvamise aeg panna sõltuma teksti pikkusest: aeg = baasaeg + repliigi\_pikkus / k. Sobiva baasaja ja k väärtuse (5 … 20) saab leida proovimise teel.

|  |  |  |  |
| --- | --- | --- | --- |
| girl4-sitting | j = 1  **kordus** tekstid1.pikkus **korda**  **ütle** tekstid1(j)  **teavita Vasta**  j = j + 1  **tagasi** | Skript **Vasta**:  **ütle** tekstid2(j) | girl5 |

### Lisaülesanne:

Kõik tekstid võivad olla ka ühes loendis. Sel juhul on paarituarvuliste järjenumbritega repliigid ühe tegelase jaoks ja paarisarvuliste järjenumbritega repliigid teise omad.

Kas nii saaks koostada ka suurema tegelaskonnaga etenduse?

# Sõnastik. Keele test

Antud on sõnastik – kaks loendit. Mingi hulga sõnu (eesti- ja inglisekeelsed) saab loenditesse importida failidest eesti.txt ja inglise.txt. Koostada rakendus kasutaja teadmiste kontrollimiseks. Loendada tuleb õigete vastuste arv. Koostada kaks skripti: eestikeelsete sõnade inglisekeelsete vastete küsimine ja inglisekeelsete sõnade eestikeelsete vastete küsimine. Rakendust arendada järk-järgult.

NB! õ ja täpitähtedega sõnu ei saa kasutada, seepärast tuleb eestikeelsete sõnadega loend pärast sõnade lisamist üle vaadata ja teha vajalikud parandused.

### Analüüs:

**Tegevuse kirjeldus:** Kasutajalt küsitakse eestikeelsete sõnade hulgast juhuslikult valitud sõna. Kasutaja peab sisestama õige inglisekeelse sõna. Skript peab kontrollima, kas vastus on õige, vajadusel suurendama õigete vastuste arvu.

Kui eelnev toimib, lisada kordus ja otsustada, millal peaks küsimine lõppema.

**Andmed:** loendid **eesti** ja **inglise**, **j** (järjenumber), **õigeid**.

õigeid = 0

**kordus**

j = juhuarv (1 kuni eesti.pikkus)

**küsi** … **loe** vastus

**kui**  vastus = inglise(j) **siis**

õigeid = õigeid + 1

ütle „Õige vastus!“

**muidu**

ütle „Vale vastus!“

**lõpp kui**

**Lõpp kordus**

### Lisaülesandeid:

* realiseerida ka inglisekeelse sõna järgi eestikeelse sõna küsimine
* loendada ka valede vastuste arv
* vale vastuse korral küsitakse sama sõna uuesti
* sama tegevust kasutada küsimustiku juures, loendites küsimused-vastused
* muuta skripti nii, et sõnu (küsimusi) küsitaks järjest – alates esimesest kuni viimaseni.

# Sõnastik. Tõlkimine

Antud on sõnastik – kaks loendit. Sõnad (eesti- ja inglisekeelsed) saab loenditesse importida failidest eesti.txt ja inglise.txt. Koostada rakendus kasutaja antud sõnade tõlkimiseks ühest keelest teise.

NB! õ ja täpitähtedega sõnu ei saa kasutada, seepärast tuleb eestikeelsete sõnadega loend pärast sõnade lisamist üle vaadata ja teha vajalikud parandused.

### Analüüs:

**Tegevuse kirjeldus:** Kasutajalt küsitakse eestikeelne sõna. Seejärel tuleb ükshaaval järjest vaadata iga eestikeelset sõna (kordus ’loendi pikkus’ korda) ja võrrelda antud sõnaga. Kui leitakse, tuleb inglisekeelsete sõnade hulgast võtta sama järjenumbriga sõna (st. muutujat järjenumbri jaoks – loendur) ja öelda see kasutajale. Kui kogu loend on üle vaadatud, aga sõna ei leitud, tuleb ka seda teatada kasutajale.

**Andmed:** loendid **eesti** ja **inglise**, **j** (järjenumber), **sõna** (küsitud sõna).

**NB!** Kui tõlkimised eesti-inglise ja inglise-eesti võivat töötada paralleelselt, võiks järjenumbri ja küsitud sõna jaoks kasutada erinevaid muutujaid või erinevate spraitide lokaalseid muutujaid (sel juhul võivad olla sama nimega).

**loe** sõna

j = 1

**kordus** eesti.pikkus **korda**

**kui**  sõna = eesti(j) **siis**

**kuva** inglise(j)

**stopp**

**lõpp kui**

j = j + 1

**lõpp kordus**

**kuva** "Sellist sõna minu sõnastikus ei ole"

**NB!** Kui väärtused loendis, milles toimub sõna otsimine, oleks järjestatud, saaks kasutada keerukamaid otsimisalgoritme, mis töötaksid kiiremini.

Lisaülesanne: lisada võimalus küsida sõnu korduvalt.

# Algoritme loenditega

|  |  |  |
| --- | --- | --- |
| Luua kaks loendit: Puud ja Hinnad (või Nimi ja Vanus, Sportlane ja Tulemus vm) ning sisestada väärtused.  Koostada skriptid, mis leiavad:   * hindade aritmeetilise keskmise * maksimaalse hinna ja sellele vastava puu nimetuse * etteantud puuliigi hinna   Arvestada võimalusega, et skriptid võivad töötada paralleelselt. | **Puud** | **Hinnad** |
| haab | 1200 |
| kask | 1700 |
| kuusk | 2300 |
| lepp | 1600 |
| mänd | 2100 |
| saar | 3100 |
| tamm | 3500 |
| vaher | 3300 |

|  |  |
| --- | --- |
| Hindade **aritmeetilise keskmise** leidmisel tuleb kõigepealt leida hindade summa, liidetavate arvuks on loendi pikkus (väärtuste arv).  Sisend: loend **Hinnad**, n (väärtuste arv loendis)  Väljund: keskmine  Abi: S (summa), i (järjenumber) | …  S = 0  i = 1  **kordus** n **korda**  S = S + Hinnad(i)  i = i +1  **lõpp kordus**  keskmine = S / n  … |
| **Suurima väärtuse** leidmisel peab muutuja **max** väärtuseks saama suurim loendi väärtustest. Algul eeldame, et esimene väärtustest on suurim, seejärel võrdleme **max** kõigi ülejäänud väärtustega. Kui leidub mõni suurem, omistatakse see muutujale **max**. Meelde tuleb jätta ka suurima asukoht. Muutuja **nr** saab uue väärtuse iga kord, kui muutub **max**.  Sisend: loendid **Puud** ja **Hinnad**, n (väärtuste arv loendis)  Väljund: **max**, **puu**  Abi: **k** (järjenumber), **nr** (suurima väärtuse järjenumber loendis) | …  max = Hinnad(1)  nr = 1  k = 1  **kordus** n - 1 **korda**  k = k +1  **kui**  Hinnad(k) > max **siis**  max = Hinnad(k)  nr = k  **lõpp kui**  **lõpp kordus**  puu = Puud(nr)  … |
| Etteantud puuliigi hinna leidmisel küsitakse kasutajalt puuliigi nimetus **liik**. Seejärel tuleb järjest vaadata kõik puu liigid loendis **Puud** (kordus ’loendi pikkus’ korda) ja võrrelda antud liigiga.  Kui selline leitakse, tuleb loendist **Hinnad** võtta sama järjenumbriga **j** väärtus. Kui kogu puuliikide loend on üle vaadatud, aga antud liiki ei leitud, tuleb ka seda teatada kasutajale.  Sisend: loendid **Puud** ja **Hinnad**, n (väärtuste arv loendis), **liik**  Väljund: **hind**  Abi: **j** (järjenumber) | …  **loe** liik  j = 1  **kordus** n **korda**  **kui**  liik = Puud(j) **siis**  hind = Hinnad(j)  **stopp**  **lõpp kui**  j = j + 1  **lõpp kordus**  hind = 0  **kuva** "Sellist puuliiki ei ole"  … |

# Loendite kasutamine spraitide asukoha juhuslikul määramisel

Kui rakenduses tuleb spraidid paigutada juhuslikult, aga kindlaksmääratud kohtadele, on üks võimalus kasutada loendite abi.

### Andmed:

loendid **sprait**, **X**, **Y**; muutuja **j** (järjenumber)

Loenditesse **X** ja **Y** salvestatakse võimalike asukohtade koordinaadid: iga paar (xi, yi) vastab ühele asukohale.

### I etapp:

Algul tuleb loend **sprait** tühjendada, seejärel peavad kõik spraidid, mis juhuslikult lavale paigutatakse, kirjutama mingi juhusliku ajavahemiku järel sellesse loendisse teatise nime (tunnuse, oma nime vm), millele nad reageerivad.

### II etapp:

Loendist **sprait** võetakse järjekorras (loendur **j** hoolitseb selle eest) teatised ja teatatakse need. Igale teatisele peab reageerima ühe spraidi skript paigutades spraidi järjenumbriga **j** määratud koordinaatidel **X**(j), **Y**(j).

|  |  |
| --- | --- |
| Tegevuste „mootor“ võib olla üks lava skriptidest:  **Tühjenda** loend sprait  **Teade** segamini - *vajalikud teatised kirjutatakse loendisse*  j = 0  **Korda** sprait.pikkus **korda**  j = j + 1  **teade** sprait(j)  **Lõpp kordus** | sun  segamin |