Tallinna Ülikool
Digitehnoloogiate instituut

GTK+ raamistiku kasutamine Pythonis PyGl mooduli vahendusel

Seminaritöö

Autor: Sander Peerna
Juhendaja: Inga Petuhhov

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Python on programmeerimiskeel, mis on viimastel aastatel väga kiiresti populaarsust kogunud, seda nii akadeemilises võtmes kui ka töökeskkonnas. Lehekülje PYPL[1], mis paneb keeled järjekorda selle järgi, kui palju programmeerimiskeele õpetusi otsitakse, andmetel on Python viimase viie aasta jooksil enim populaarsust kogunud programmeerimiskeel. Akadeemilise populaarsuse taga võib olla see, et Python-st on suhtelist kerge aru saada, seda ei pea kompileerima ja materjale on palju. Kuna paljudel noorematel programmeerimise õppuritel on just Python esimene keel mida õppitakse ja see on ka Tallinna Ülikoolis ja Tartu Ülikoolis esimene keel mida õpetatakse, siis võiks esimene keel, kus tehakse graafilist kasutajaliidest olla Python-ga seonduv. Seega oleks hea õppida just GTK+ (GIMP Toolkit) raamistiku, kuna see on üks levinumaid Python-i jaoks.

Teema sai valitud, kuna olen ise ka huvitatud Pythoni-ga graafilise kasutajaliidese loomisest ning puudub eestikeelne materjal GTK+ raamistikust. Antud töö on suunatud inimestele, kellel on baasteadmised Python-st.

Käesoleva seminaritöö eesmärgiks on tutvustada GTK+ raamistiku ja selle erinevaid võimalusi ning luua juhend, mis näitaks kuidas kasutada GTK+ raamistiku võimalusi Python-ga. GTK+ kasutatakse graafilise kasutajaliidese loomiseks ja üheks pealmiseks eeliseks on sealjuures see, et seda on lihtne kasutada ning töötab võimalikult paljudel platvormidel.

Töö on jaotatud kahte ossa, esimeses osas tehakse ülevaade GTK+ raamistikust, selle ajaloost ja peamistest osadest. Teine osa on õpetus GTK+-ga loodud programmist.

Üldisem informatsioon Python-i ja GTK+-i, mis on juhendis kasutatud pärineb Python-i veebilehelt (The Python Language Reference, kuupäev puudub) ja GTK+ kohta käiv informatsioon pärineb raamatust “ Foundations of GTK+ Development” (Krause, 2007).

1 http://pypl.github.io/PYPL.html

1. GTK+

GTK+ ehk GIMP Toolkit on mitmeplatvormiline raamistik, millega luuakse graafilisi kasutajaliideseid. GTK+ on kirjutatud C keeles, kuid on disainitud algusest peale nii, et see oleks ühilduv ka teiste keeltega[2]. GTK+ on üles ehitatud vidinate peale, mis teeb raamistiku kasutamise lihtsaks ja kiireks. Kuna GTK+ on vabavara, kõigil võimalus seda kasutada.

1.1. Ajalugu

GTK loodi aastal 1998 Peter Mattise poolt GNU Image Manipulation Programm-i (GIMP) jaoks. Põhjus miks GTK loodi seisneb selles, et Peterile ei meeldinud GIMP-s kasutusel olev Motif raamistik[3]. Kui GTK oli jõudnud versioonini 0.60, vahetati GIMP-s kasutusel olev Motif raamistik GTK vastu välja. Algselt polnud GTK üldse loodud üldkasutatavaks raamistikuks, vaid oligi mõldud kõigest GIMP-i jaoks. Aastal 1997 lasti välja versioon 0.99, selles versioonis sai GTK'st ka GTK+. Põhilised uuendused versioonis 0.99 olid pluginate portimine uuele mäluskeemile ja uus API. Mõningad arendajad olid arvamusel, et GTK+ on hea raamistiks ja seda peaks kasutama igal võimalusel. Seda ideed kasutas ka GNOME, kes tegid GTK+ raamistikuga GNOME'i töölauakeskkonna.

1.2. GTK+ omadused

GTK+-i on arendatud üle kümne aasta, seega on raamistik väga stabiilne ja kiire. GTK+ toetavad suured ettevõtted, näiteks Intel ja Red Hat. Raamistik on multiplatvormne, kuigi algselt loodi see X Window System-i jaoks. Aastate jooksul on seda aga nii palju arendatud, et tänapäeval töötab see ka Windows-i ja MacOS-i peal.

Keele seotised (ingl *bindings*) lubavad GTK+ kasutada ka teistel programmeerimiskeeltel ja ka teiste keelte stiilis. Kuigi GTK+ on algselt kirjutatud C jaoks, siis tänu keele seotistele on seda võimalik kasutada ka näiteks Python-i, C++, JavaScript-i ja Vala-ga[4].

2 https://www.gtk.org/
3 https://www.gimp.org/about/ancient_history.html
4 https://www.gtk.org/features.php

1.3. GTK+ vidinad

GKT+'l on suur hulk vidinaid ja liideseid, mida on võimalik oma projektides kasutada. Järgnevalt kirjeldatakse lühidalt kõiki GTK+ vidinaid.

Boks (Box) on nähtamatu konteiner, mille sisse saab paigutada vidinaid. Vidinad paigutatakse automaatselt horistonaalselt kas vasakult paremale või paremalt vasakule, sõlutavalt kas on kasutatud `Gtk.Box.pack_start()`, mis alustab vidinate paigutamist vasakult või `Gtk.Box.pack_end()`, mis alustab vidinate paigutamist paremalt.

Ruudustik (Grid) on konteiner, mis seab alamvidinad ridadesse ja veergudesse. Ruudustiku puhul ei pea konstruktoris eraldi suurusi spetsifiseerima. Ruudustiku saab kasutada sarnaselt boksile, selleks tuleb kasutada `Gtk.Grid.add()`, mis paigutab boksid üksteise järel vastavalt sellele, millist orientatsiooni on kasutatud (vaikesättena horisontaalne).

Loendiboks (ListBox) on vertikaalne konteiner, mida saab dünaamiliselt sorteerida ja filtreerida. Tihti kasutatakase loendiboksi puuvaate alternatiivina, eriti kui sisu on keerulisem sellest, mida `Gtk.CellRender()` suudab kuvada.

Virn ja virna vahetaja (Stack, StackSwitcher) on `Gtk.Stack` on konteiner, mis näitab ainult ühte alamvidinat üheaegselt. Selleks, et oleks võimalikt alamvidinat vahetada on vaja kasutada vidinat `Gtk.StackSwitcher`. Üleminekuid kahe vidina vahel näidatakse animatsioonidega, näiteks ühe eest ära nihutamine ja teise asemele nihutamine. Samuti on võimalik muuta ka üleminekuaega.

Päiseriba (HeaderBar) on riba, mis on tavaliselt akna ülaosas. Tavaliselt on selles konteineris programmi nimi ja ka kõige tavalisemad nupud, mis sulgevad akna või teevad akna väikseks.

Voogboks (FlowBox) on konteiner mis paigutab alamvidinad järjekorda. Vidinad paigutatakse automaatselt vasakult paremale ja vajadusel alustatakse ka uuelt realt.

Silt (Label) on põhiline meetod mida kasutatakse, kui on vaja lisada aknasse teksti mida ei saa muuta. Teksti laiust muudetakse automaatselt ja mitme realisi tekste saab luua, kui kasutada rea lõpus ("\n") stringi. Teksti on võimalik muuta ka kopeeritavaks kasutades

`Gtk.Label.set_selectable()`. Silt toetab ka lihtsamat vormistamist *Pango Markup* abil, mida saab kasutada `Gtk.Label.set_markup()` abil.

Sisestus (Entry) lubab kasutajal sisestada teksti. Sisu saab lugeda `Gtk.Entry.get_text()` meetodi abil. Ühtlasi saab ka seada tähemärkide piirangu `Gtk.Entry.set_max_length()` meetodi abil. Antud vidinat saab kasutada ka paroolide jaoks, sest on olemas võimalus peita sisestatud teksti. Selleks peab kasutama meetodit `Gtk.Entry.set_visibility(False)`.

Nupp (Button) on vidin, mida tavaliselt kasutatakse funktsiooni sidumiseks graafilise nupuga. Nupu vidin suudab endas hoida enamus standartseid vidinaid, aga kõige rohkem kasutatakse vidinat `Gtk.Label()`. Üldiselt ühendatakse nupuga “clicked” signaal, mis antakse edasi kui nuppu vajutatakse ja lahti lastakse.

Tumblernupp (ToggleButton) on sarnane tavalisele nupule, aga peale vajutades püsib nupp aktiivsena kuni seda vajutatakse uuesti. Kui nupu olek muutub, siis saadetakse “toggled” signaal.

Märkenupp (CheckButton) on sarnane tumblernupule. Erinevus seisneb selles, et märkenupu puhul tekib valiku ette kast, kuhu peale vajutades tekib linnuke.

Raadionupp (RadioButton) on nupud, mis töötavad ainult grupina, kusjuures grupist olevatest nuppudest saab ainult ühe aktiivseks teha.

Linginupp (LinkButton) on nupp, mis sisaldab linki veebilehele ja sellele peale vajutades avatakse veebibrauser antud veebilehega.

Spinnernupp (SpinButton) on nupp, mis lubab sisestusväljas olevat sisu muuta. Näiteks kui sisestusväljas on numbrid, siis saab spinnernupu abil muuta neid ükshaaval kas suuremaks või väiksemaks.

Lüliti (Switch) on vidin millel on kaks olekut- sees või väljas. Kasutaja saab kontrollida aktiivset olekut vajutades tühjale alale või nihutades nuppu.

Edenemisnäidik (ProgressBar) on kasutusel, kui programm suudab tuvastada kui palju tööd on vaja teha, näitamaks kui palju tööd on tehtud ja kui palju on veel vaja teha.

Spinner kuvab ikooni suuruse keerlemisanimatsiooni, mida tavaliselt kasutatakse edenemisenäidik alternatiivina, näiteks juhul kui pole võimalik kindlalt määrata kui kaua võib käesolev töö aega võtta.

Liitboks (ComboBox) on sarnane raadionupule, aga valikud on esitatud rippmenüüna. Üldiselt kasutatakse siis, kui valikus on raadionupu kasutamiseks liiga palju elemente.

Ikoonivaade (IconView) kuvab ikoonid võrestikus, mis muutub automaatselt vastavalt akna suurusele. Vajadusel on võimalik ikoone ka liigutada ja ümber järjestada.

Mitmerealine teksti redaktor (Multiline Text Editor) on vidin, mis kuvab suuremat hulka teksti ja samas võimaldab seda tekst muuta.

Dialoogiboks (Dialogs) on sarnane tavalisele aknale, kuid üldiselt kasutatakse, et anda edasi kindlat informatsiooni.

Lõikelaud (Clipboard) tekitab koha, kus saab mälus hoida näiteks teksti või pilti. Lõikelaud lubab mälus olevaid andmeid kopeerida erinevate programmide vahel.

Pukseerimine (Drag and Drop) lubab programmi siseselt objekte nihutada. Eeldab, et on määratud nihutamis allikas ja koht kuhu objekt nihutatakse.

(The Python GTK+ 3 Tutorial, 2016)

2. Graafilise liidese loomine GTK+ abil

Kuigi varem pidi GTK+ puhul enne alustamist ise raamistiku kompileerima, siis tänu populaarsuse kasvamisele on nüüd raamistiku ülesseadmine palju lihtsam. Nii Windows-i kui ka Linux-i peal on paigaldamine kõigest paari minuti töö.

Juhendi kasutamise eelduseks on Python-i baasteadmised, näiteks kursuse “Programmeerimise Alused” läbimine, samuti võiks olla baasteadmised objektorienteeritud programmeerimises. Autor on juhendis nähtavad näited kirjutanud Linux-i peal, mis tähendab, et joonised võivad erineda Windows-is nähtavatest programmidest. Koodiread on ka nummerdatud ning neid ei tohi ise oma koodi sisse kirjutada. Antud hetkel on numbrid selleks, et teha koodi ja juhendi jälgimist lihtsamaks. Ühtlasi on ka võimalik näha kõikide juhendis valminud programmide koodi autori leheküljelt[5].

2.1. Paigaldamine

GTK+ paigaldamise eelduseks on Python 2.X või 3.X, kusjuures Windows-i peal ei tohi olla uuem kui 3.4 versioon, kuna versiooniga 3.5 ei ole hetkel PyGObject veel ühilduv. Antud juhendis on kasutusel Python 3.4. Kui Python on paigaldatud, siis järgmiseks tuleb allalaadida GTK+ raamistik Python-i jaoks. Selleks kasutame PyGObject (PyGl) nimelist moodulit, mis võimaldab kasutada Pythonil GTK+ teeke.

Windows-i peal installimiseks tuleb kõigepealt kindlaks teha, et oleks paigaldatud Python-i versioon 3.4 ja see järel alla laadida ja installida PyGl-aio faili uusim versioon, mis on kättesaadav Sourceforge lehelt[6] (French, 2014). Paigaldamise alustamisel võib juhtuda, et programm küsib kausta, kuhu Python on paigaldatud, sellisel juhul tuleb see koht üles leida ja valida. Üldiselt on selleks kohaks C:\Python34. Seejärel tuleb järgmine aken valikutega, kus tuleb valida Python-i versioon. Juhul kui näitab ühte versiooni mitmekordselt, tuleb valida esimene valik. Järgmisena antakse valik erinevaid teeke, millest valime *Base pacakges* ja *GTK+3.18.*

5http://www.tlu.ee/~sanderss/Seminar/

6https://sourceforge.net/projects/pygobjectwin32/

Peale seda kuvatakse veel kaks akent, kus antakse võimaluse paigaldada lisateeke, kuid neid pole antud juhendis vaja.

Enamus Linux-i distributsioonide puhul on võimalik PyGObject-i allalaadida distributsiooni ametlikust repositooriumist.

Ubuntu ja Debian:

```
sudo apt install python3-gi
```

Fedora:

```
sudo dnf install pygobject3
```

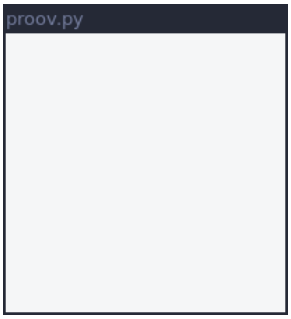
Kui PyGObject on paigaldatud, siis tuleks proovida kas kõik töötab. Selleks tuleb avada näiteks Python-i IDLE või kasutada omale sobivat tekstitöötlusvahendit ja kirjutada sinna järgmise koodi näite (vt koodinäide 1).

```
1  import gi
2  gi.require_version('Gtk', '3.0')
3  from gi.repository import Gtk
4  win = Gtk.Window()
5  win.connect("delete-event", Gtk.main_quit)
6  win.show_all()
7  Gtk.main()
```

Koodinäide 1: Esimene programm

Kui kood on kirjutatud saab seda testida kas IDLE-s selle käivitamisega või kasutades terminali.

Tulemuseks peaks tekkima tühi aken nagu joonisel 1.



Joonis 1: Tühi aken

Arusaamaks, miks aken tekib, tuleb lahti seletada, mida iga rida antud koodi näites teeb.

GTK+ töötamiseks tuleb kõigepealt projekti alguses importida teek `gi`. Kuna on võimalik, et paigaldatud on rohkem kui üks versioon GTK-st, siis tuleb ka kindlasti määrata millist versiooni tahame kasutada, seda saab teha käsuga `gi.require_version('Gtk', '3.0')` ja kolmanda reaga importime gi-st GTK. Neljanda reaga loome tühja akna. Viienda reaga loome sündmuse mis määrab olukorra, kus *X* vajutamisel sulgub aken (Joonisel 1 pole antud funktsionaalsust näha, kuna autor on joonistel kasutanud Linux-t ja töölauakeskkonda, milles puudub klassikaline akende sulgemise meetod). Kuuendal real me teeme akna nähtavaks. Viimase reaga alustame GTK+ protsessi, mida me saame sulgeda viiendal real loodud käsuga.

2.2. Nupud

Antud peatükis kirjeldatakse erinevate nuppude võimalusi ja luuakse programm, mis ka kasutab erinevaid nuppe. Selleks, et programmi hakata üles ehitama võib võtta aluseks eelmises näites tehtud proovi. Üldiselt võib tehtud näite võtta alusmalliks peaaegu alati, kuna tõenäoliselt on programmis alati olemas kinnipanemisnupp ja ülejäänud koodiread peavad alati olemas olema, et GTK töötaks.

Esiteks tuleb taaskasutada kolme esimest rida eelmisest näitest ja seejärel loome klassi, mis on alamklass `Gtk.Window`-st ja anname sellele nime, näiteks Nupud. Järgmisena loome klassi konstruktori, kuhu lisame programmi tiitli ja määrame ka piiride suuruse. Lisame ka ***Boks*** konteineri, kuhu sisse hakkame mahutama nuppe. Lisaks määrame, et konteineri orientatsioon on `VERTICAL`, mis tähendab et nuppud hakkavad järjest tekkima üksteise alla ja spacing 6, mis näitab, et elementide vahel on 6 piksline vahe (vt koodinäide 2).

```
4  class Nupud(Gtk.Window):
5      def __init__(self):
6          Gtk.Window.__init__(self, title="Nupud")
```

```
7           self.set_border_width(10)
8           vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
9           self.add(vbox)
```

Koodinäide 2: Nupuprogrammi algus

Kui konteiner tehtud, on võimalik nuppe lisada. Algatuseks loome kõige lihtsama nupu, mis vajutamise peale kuvab konsooli teksti „”Vajuta” nuppu vajutati”. Selleks teeme kõigepealt ***Nupu*** vidina, seejärel ühendame nupu funktsiooniga ja lõpetuseks lisame nupu konteinerisse (vt koodinäide 3).

```
10 nupp = Gtk.Button.new_with_label("Vajuta")
11 nupp.connect("clicked", self.vajuta_mind)
12 vbox.pack_start(nupp, True, True, 0)
```

Koodinäide 3: Nupu loomine

Et nupp töötaks tuleb juurde teha ka funktsioon, mis pannakse klassi lõppu (vt koodinäide 4).

```
37 def vajuta_mind(self, nupp):
38    print("\"Vajuta\" nuppu vajutati")
```

Koodinäide 4: Nupu funktsioon

Nüüd peaks juba töötama programm, mis kuvab ühe nupu mida saab vajutada ja mida vajutades trükitakse konsooli tekst: „”Vajuta” nuppu vajutati”. Järgmiseks teeme nupu, mis vajutuse peale sulgeb akna. Selleks tuleb kõigepealt teha ***Nupu*** vidinaga samasugune nupp nagu eelmises näites ja see järel teha ka uus funktsioon selle tarvis. Kindlasti tuleb ära vaheta funktsiooni nimi nii uues funktsioonis kui ka uues ***Nupu*** vidinas. Erinevuseks on see, et peale `print`-i kirjutame lõppu juurde veel ka `Gtk.main_quit()` (vt koodinäide 5).

```
39 def sulge(self, nupp):
40    print(“Sulgen programmi”)
41    Gtk.main_quit()
```

Koodinäide 5: Teise nupu funktsioon

Nüüdseks on juba valmis programm, mis suudab nupu vajutuse peale kas trükkida konsooli teksti või vajutuse peale programmi kinni panna. Järgmiseks lisame nupud, millel on kaks olekut, kas sees või väljas. Selleks kasutame eraldi vidinat ***Tumblernupp***, millel on sisseehitatud võimalus kaheks olekuks (vt koodinäide 6).

```
16 nupp = Gtk.ToggleButton("Nupp 1")
17 nupp.connect("toggled", self.lylitus, "1")
18 vbox.pack_start(nupp, True, True, 0)
```

Koodinäide 6: Tumblernupp

Kõigepealt defineerime funktsiooni `lülitus`, mis erineb viimastest funktsioonidest selle poolest, et tal on kolm argumenti kahe asemel. Juurde on tulnud argument `nimi`, mis on nagu argument ütleb, nupu nimi. Nupu nime lisasime 17 real ja nimeks on “1”. Funktsiooni jaoks on vaja kasutada `if` lauset, et muuta seda, mida erinevate olekutega edasi tehakse. Näite jaoks muudame `if` lausega muutujat `olek`, milles hoiame väärtust “sisse” või “välja” (vt koodinäide 7).

```
42 def lylitus(self, nupp, nimi):
43         if nupp.get_active():
44             olek = "sisse"
45         else:
46             olek = "välja"
47         print("Nupp", nimi, "lülitati", olek)
```

Koodinäide 7: Lülituse funktsioon

Kolmandaks teeme raadionupud, mille eripäraks on see, et ainult üks nupp raadionuppude grupist saab olla aktiivne. Raadionuppe luues on oluline, et siduda peale esimest nuppu kõik järgnevad nupud esimese nupuga (vt koodinäide 8).

```
23 nupp1 = Gtk.RadioButton.new_with_label_from_widget(None, "Nupp 1")
24 nupp1.connect("toggled", self.lylitus, "1")
25 vbox.pack_start(nupp1, False, False, 0)
```

Koodinäide 8: Raadionupud

Esimene nupp on sarnane eelnevalt loodud nuppudele, küll aga järgmistes nuppudes tuleb esimesel real `None` asemel määrata grupp, milleks antud näites on `nupp1` (vt koodinäide 9).

```
26 nupp2 = Gtk.RadioButton.new_with_label_from_widget(nupp1,"Nupp 2")
27 nupp2.connect("toggled", self.lylitus, "2")
28 vbox.pack_start(nupp2, False, False, 0)
```

Koodinäide 9: Teine raadionupp

Viimaseks teeme lüliti tüüpi nupu, millel on samuti kaks olekut, küll aga näeb see välja nagu lüliti. Erinevalt raadionupust kasutab ***lüliti*** signaalina `notify::active` ja ***toggled*** *ei ole* soovitatav signaal ***lüliti*** puhul. Lisaks tuleb veel kasutada argumenti `set_active`, mis määrab kas lüliti on aktiveeritud või mitte (vt koodinäide 10).

```
33 lyliti = Gtk.Switch()
34 lyliti.connect("notify::active", self.aktiveeritud)
35 lyliti.set_active(False)
36 vbox.pack_start(lyliti, True, True, 0)
```

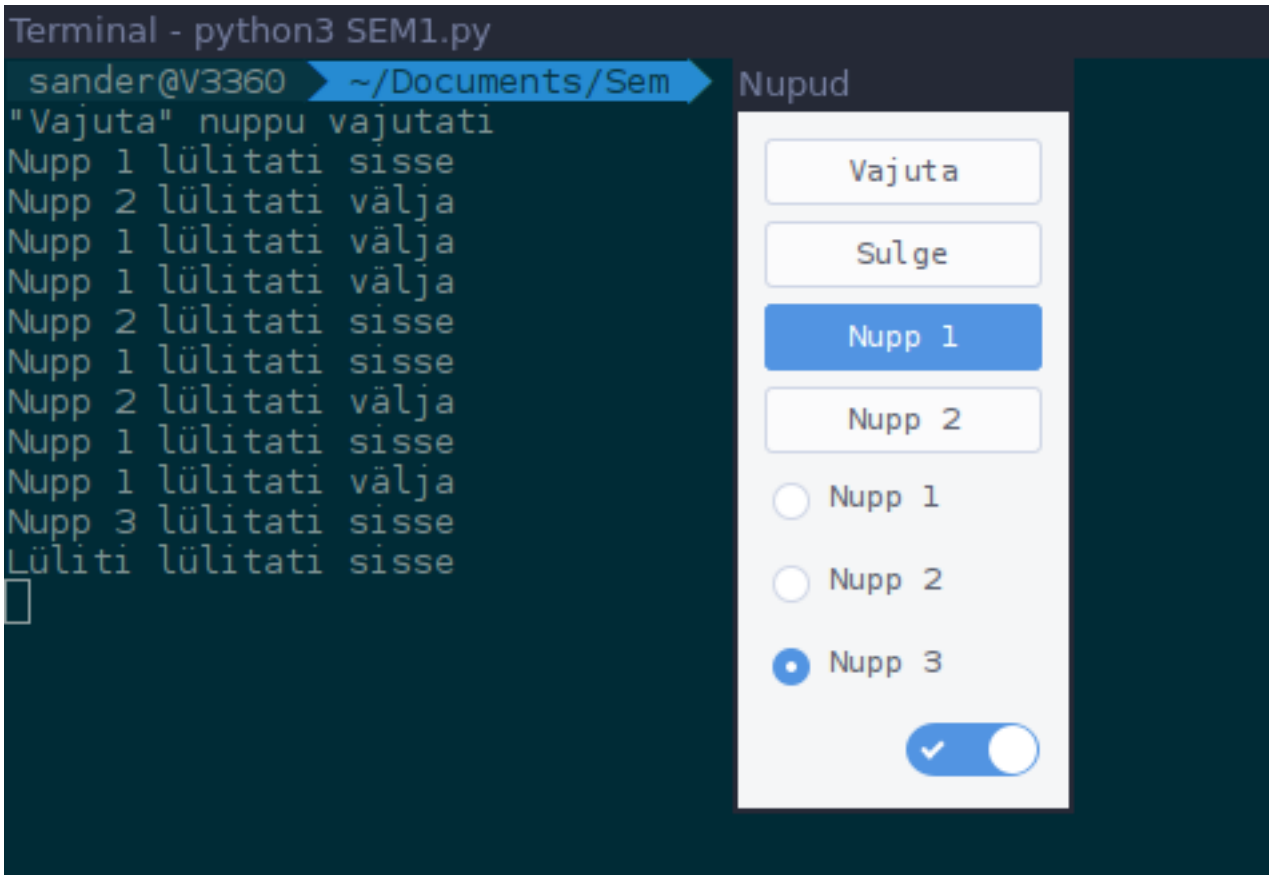
Koodinäide 10: Lüliti tüüpi nupp

Funktsioonina saab uuesti kasutada ***tumblernupu*** tarvis loodud funktsiooni (vt koodinäide 11).

```
48 def aktiveeritud(self, lyliti, gparam):
49         if lyliti.get_active():
50             olek = "sisse"
51         else:
52             olek = "välja"
53         print("Lüliti lülitati", olek)
```

Koodinäide 11: Lüliti tüüpi nupu funktsioon

Kõige selle lõpptulemusena peaks valmis saama programm mitmete erinevate nuppudega, mis peaks olema hea ülevaade erinevatest nupu võimalustest GTK+ raamistikus (vt joonis 2). Lisas 1 on ka välja toodud kogu kood ühe tervikuna.



Joonis 2: Näide nuppude programmi tööst

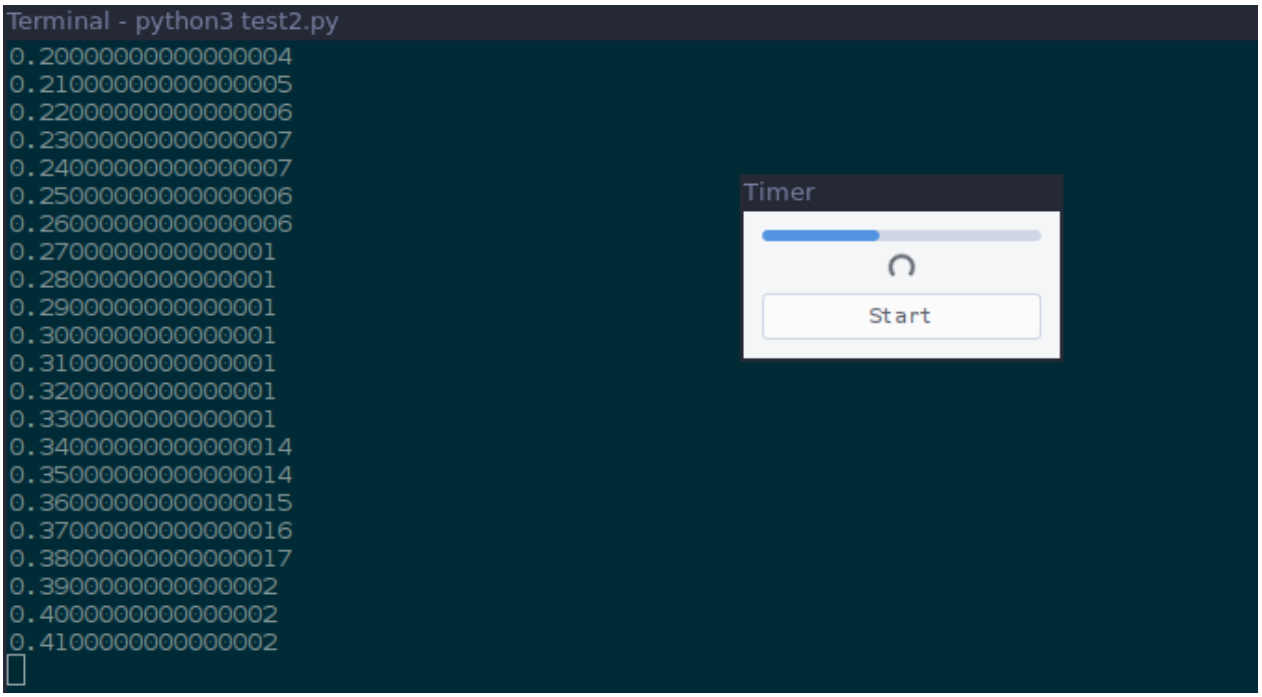
2.3. Protsessi näitamine

Protsessi näitamiseks on GTK+-s paar erinevat vidinat. Esiteks on ***Edenemisnäidik***, mis näitab protsessi joone peal ja mille järgi on võimalik ka hinnata umbkaudu kui kaugel protsess on. Teiseks võimaluseks on ***Spinner***, mis on alternatiiv ***Edenemisnäidikule***, aga erinevalt sellest ei saa ***Spinner*** näidata kui kaugele on protsessiga jõutud (vt joonis 3). Seega üldiselt kasutatakse seda kohtades, kus pole kindlalt võimalik määrata protsessi ajakulu. Programmi malliks võib võtta eelmise näite, eemaldades nupud ja nendega seonduvad funktsioonid. Kõigepealt lisame kaks vidinat, ***Edenemisnäidik*** ja ***Spinner*** ning siis lisame veel ka nupu. Nupule lisame funktsiooni, mis käivitab mõlemad protsessi näitavad vidinad. Lisaks teeme veel

edenemisnäidikule ka tsükkli, mis tekitaks olukorra, kus riba saab täituda ainult ühe korra (vt Lisa 2).

```
17     def on_start(self, button):
18         self.timeout_id = GObject.timeout_add(50, self.on_start, None)
19         self.activity_mode = False
20         self.spinner.start()
21         while (self.progressbar.get_fraction() < 1):
22             new_value = self.progressbar.get_fraction() + 0.01
23             print (self.progressbar.get_fraction())
24             self.progressbar.set_fraction(new_value)
25             return False
```

Koodinäide 12: Edenemisriba funktsioon



Joonis 3: Protsesside näitamine

Real 18 olevat `self.timeout`-i kasutame, et loodud funktsioonalsus antud intervalli tagant korduks. `Self.activity_mode = False` määrab, et edenemisriba liiguks vaskult paremale, kui oleks väärtuseks `True`, siis edenemisriba ei näita protsentuaalselt edenemist, vaid riba liigub edasi

tagasi (vt Koodinäidet 12). Real 24 olev `set_fraction()` annab edenemisribale väärtuse, mis määrab kui kaugele edenemisriba joonistama peab, väärtuse 0 puhul on edenemisriba alguses ja väärtuse 1 puhul on edenmisriba lõpus.

2.4. Puu ja listi vidinad

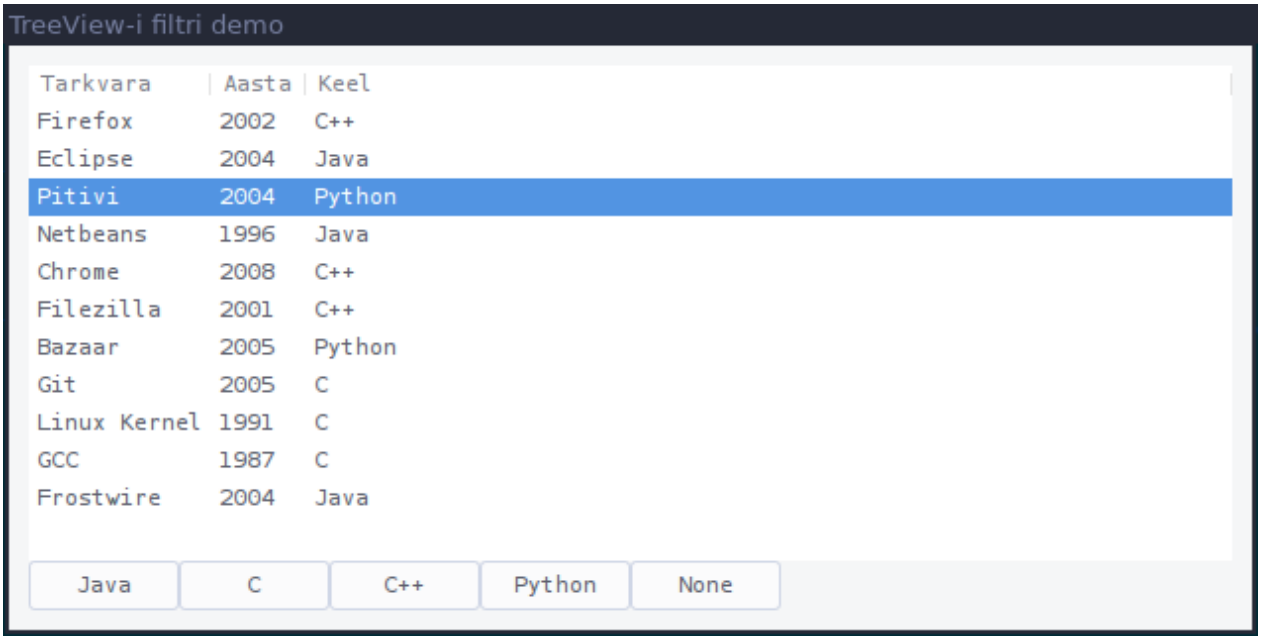
Puuvaate vidinat kasutatakse koos ***Listihoidla*** (ingl *ListStore*) ja ***Puuhoidla*** (ingl *TreeStore*) mudelitega ning nendega koos loodakse võimalus, et kuvada ja töödelda informatsiooni (vt joonis 4). ***Puuvaatel*** on mitu erinevat viisi andmeid kuvada ja manipuleerida, millest olulisemad on:

- Automaatselt uuendab andmeid, kui neid lisatakse, eemaldatakse või muudetakse
- Andmete sorteerimine ja filtreerimine
- Võimalus kasutada koos Puuvaatega märkenupu, edenemisriba ja teisi vidinaid

Listihoidla hoiab endas andmete ridasi ja igal real puudub tütarrida, kuid ***Puuhoidla*** suudab hoida endas andmete ridasi, millel on igal veel omakorda ka tütarread. Koos ***Puuvaatega*** peab kasutama ka ***Puumudelit*** (ingl *TreeModel*), mis kuvab andmed. Igat ***Puumudelit*** saab kasutada rohkem kui üks ***Puuvaate***. See annab võimaluse samu andmeid kahel erineval viisil samaaegselt muuta või näidata samade andemete erinevaid ridu samaaegselt.

Üks tähtsamaid omadusi puudel ja listidel on sorteerimine ja see võimalus on ka olemas. Kõige lihtsam viis teha puu vaade sorteeritavaks, on kasutada funktsiooni `Gtk.TreeViewColumn.set_sort_column_id()`.

Kuigi sorteerimises kasutatakse ***Puuhoidlat*** ja ***Listihoidlat***, siis filtreerides kasutatakse täiesti eraldi mudelit ***Puumudelifiltrit*** (ingl *TreeModelFilter*). Sarnaselt kahele eelnevale mudelile on ka see ***Puumudeliga*** seonduv. See töötab kihina nii-öelda päris mudelites, peites mõned elemendid vaates. Kõige selle lihtsamaks arusaamiseks saab vaadata koodinäidet lisades (vt Lisa 3).



Joonis 4: Puuvaate filtri demo

Puuvaates on oluline, et elemendid oleksid listis, seega tuleb kõigepealt teha üks. Antud näites on list loodud programmi algusesse (vt koodinäide 13).

```
4  software_list = [("Firefox", 2002,  "C++"),
5                   ("Eclipse", 2004, "Java" ),
6                   ("Pitivi", 2004, "Python"),
7                   ("Netbeans", 1996, "Java"),
8                   ("Chrome", 2008, "C++"),
9                   ("Filezilla", 2001, "C++"),
10                  ("Bazaar", 2005, "Python"),
11                  ("Git", 2005, "C"),
12                  ("Linux Kernel", 1991, "C"),
13                  ("GCC", 1987, "C"),
14                  ("Frostwire", 2004, "Java")]
```

Koodinäide 13: List Puuvaate jaoks

Et filtreerimine töötaks on vaja luua ***Listihoidla*** mudel ja sinna sisse lisada filtrid. Samuti on vaja ka teha ***Puuvaate***, kuna ilma selleta poleks midagi näha (vt koodinäide 14).

```
23          self.software_liststore = Gtk.ListStore(str, int, str)
24          for software_ref in software_list:
25              self.software_liststore.append(list(software_ref))
26          self.current_filter_language = None
27          self.language_filter = self.software_liststore.filter_new()
28          self.language_filter.set_visible_func(self.language_filter_func)
29          self.treeview = Gtk.TreeView.new_with_model(self.language_filter)
30          for i, column_title in enumerate(["Tarkvara", "Aasta", "Keel"]):
31              renderer = Gtk.CellRendererText()
32              column = Gtk.TreeViewColumn(column_title, renderer, text=i)
33              self.treeview.append_column(column)
```

Koodinäide 14: Listihoidla ja Puuvaate loomine

Tuleb kirjutada ka funktsioon filtri jaoks. Filtriga saame filtreerida massiivis olevad andmed kasutatud programmeerimiskeele järgi. (vt koodinäide 15).

```
48      def language_filter_func(self, model, iter, data):
49          if self.current_filter_language is None or 
self.current_filter_language == "None":
50              return True
```

Koodinäide 15: Filtreerimis funktsioon

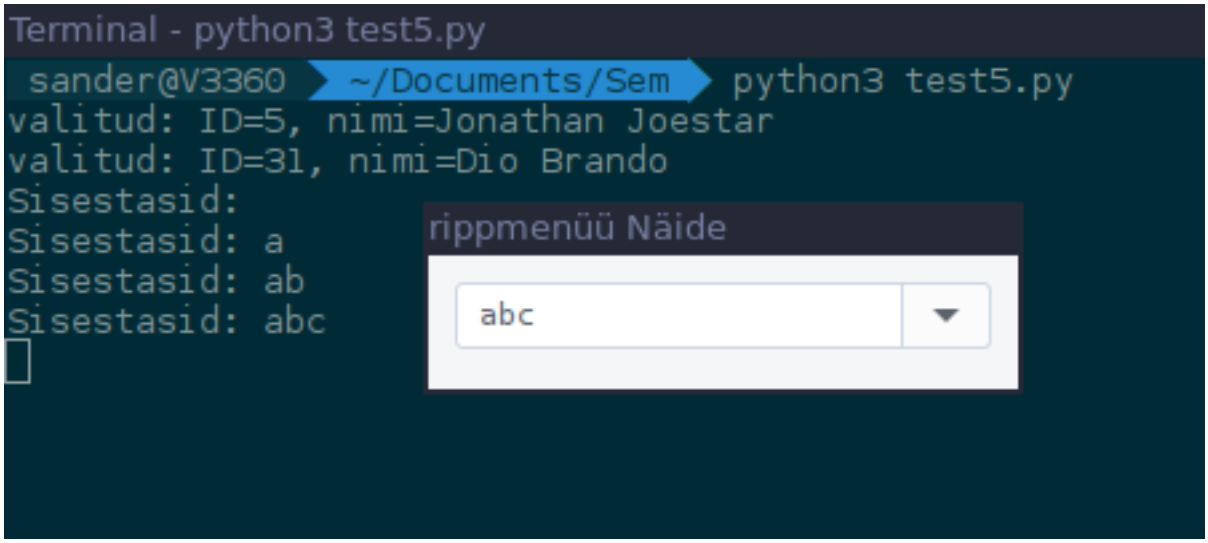
2.5. Liitboks

Liitboksi kasutatakse juhtudel, kus raadionuppe ei saa kasutada, kuna valikuid on liiga palju või tahetakse anda kasutajale võimalus ka ise vastusevariant välja mõelda (vt joonis 5). Vajadusel on võimalik lisada sinna ka pilte ja edenemisriba. Üldiselt piiratakse kasutaja ainult antud valikutele, aga on ka võimalik kasutada `Gtk.entry` funktsiooni, et lubada kasutajal ise kirjutada vastus, kui valikus ei leidu sobivat.

Programmi malliks võetakse ikka need sama seitse rida, mis said alguses kirjutatud ja siis hakkatakse sinna edasi kirjutama. Kõigepealt tuleb luua `Gtk.ListStore` abil andmetehoidla. Selleks kirjutame koodirea: `name_store = Gtk:listStore(int, str)`. Siinkohal kasutame

`int` objekti *ID*-na ja `str` kasutame objekti nimena, milleks selle näite põhjal on inimeste nimed. Seejärel hakkame hoidlasse objekte lisama, seda saab teha kirjutades näiteks `name_store.append([1, “Charmion Simion”])`. Kui on mõned nimed lisatud, tuleb see kõik ***Liitboksiga*** ühendada:

`name_combo = Gtk.ComboBox.new_with_model_and_entry(name_store)`(vt Lisa 4).



Joonis 5: Liitboksi näide

2.6. Ikoonid ja nende pukseerimine

Ikoonide haldamiseks kasutatakse `Gtk.IconView` vidinat, mis kuvab ikoone võrestikus, lubab ikoonide nihutamist, mitme ikooni üheaegselt valimist ja ka nende ümber reastamist. Sarnaselt ***Puuvaatele*** kasutab ka ***Ikoonivaade*** sama mudelit, milleks on ***Listihoidla***. Selleks, et näiteks määrata ikoonide valimise meetodit tuleb kasutada `Gtk.IconView.set_selection_mode()`, mis lubab kas valida mitut ikooni korraga või keelata igasuguse ikoonide valimise.

Enne kui alustada nihutamise implementeerimist, tuleb kindel olla, et paigaldatud PyGObject on vähemalt versioon 3.0.3 või uuem, vastasel juhul ei tööta antud funktsionaalsus. Selleks, et pukseerimise funktsionaalsus töötaks on vaja ikoonide kinnivõtmis allikat ja nende lahtilaskmisala (vt koodinäide 16).

```
14          self.iconview = DragSource()
15          self.drop_area = DropArea()
```

Koodinäide 16: Kinnivõtmis allikas ja lahtilaskmisala

Kõige tavalisem kinnivõtmis ja lahtilaskmis funktsioon vajab, et allikas oleks ühendatud signaaliga “drag-data-get” ja sihtpunkt signaaliga “drag-data-received”. Esiteks teeme klassi DragSource, mis on meie kinnivõtmisala. Sinna sisse teeme mudeli, milles on ***Listihoidla*** kuhu lisame kolm ikooni. Lisame mudelile veel ka funktsiooni selleks, et oleks võimalik ikoonidest kinni võtta (vt koodinäide 17).

```
24 class DragSource(Gtk.IconView):
25     def __init__(self):
26         Gtk.IconView.__init__(self)
27         self.set_pixbuf_column(COLUMN_PIXBUF)
29         model = Gtk.ListStore(str, GdkPixbuf.Pixbuf)
30         self.set_model(model)
31         self.add_item("Ikoon 1", "edit-paste")
32         self.add_item("Ikoon 2", "edit-cut")
33         self.add_item("Ikoon 3", "edit-copy")
34         self.enable_model_drag_source(Gdk.ModifierType.BUTTON1_MASK, [],
35             DRAG_ACTION)
36         self.connect("drag-data-get", self.on_drag_data_get)
37     def on_drag_data_get(self, widget, drag_context, data, info, time):
38         selected_path = self.get_selected_items()[0]
39         selected_iter = self.get_model().get_iter(selected_path)
40         pixbuf = self.get_model().get_value(selected_iter, COLUMN_PIXBUF)
41         data.set_pixbuf(pixbuf)
```

Koodinäide 17: Kinnivõtmisala

Funktsiooniga `add_item` saame kätte kinnivõtmisalal kasutatavad ikoonid (vt koodinäide 18).

```
42     def add_item(self, text, icon_name):
43         pixbuf = Gtk.IconTheme.get_default().load_icon(icon_name, 16, 0)
44         self.get_model().append([text, pixbuf])
```

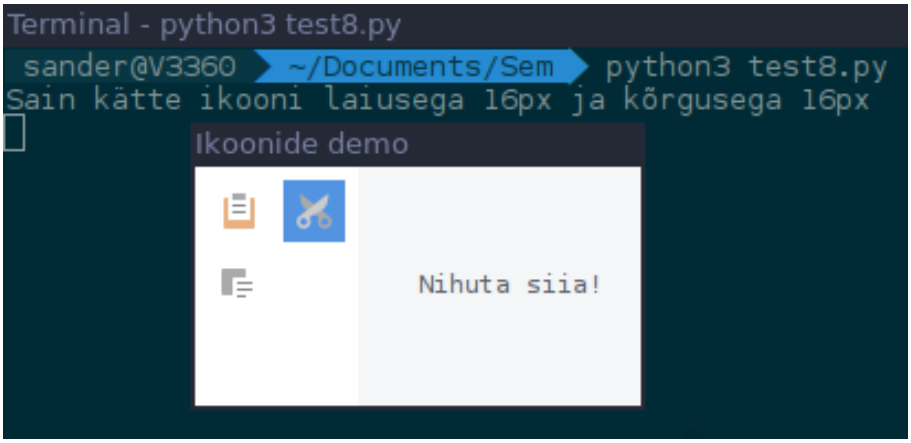
Koodinäide 18: Ikoonide loomine

Lõpetuseks tuleb teha ka lahtilaskmisala, sinna teeme algatuseks ***sildi,*** kuhu on kirjutatud “Nihuta siia!”, et oleks selge kuhu antud programmis ikoone pukseerima peab. Järgmiseks ühendame antud ala signaaliga `drag-data-received` ja seejärel loome talle funktsiooni. Real 51 olevat `data.get_pixbuf()`-i kasutame, et saada kätte ikoonide andmed, antud näites võtame neilt ikooni kõrguse ja laiuse.

```
45 class DropArea(Gtk.Label):
46     def __init__(self):
47         Gtk.Label.__init__(self, "Nihuta siia!")
48         self.drag_dest_set(Gtk.DestDefaults.ALL, [], DRAG_ACTION)
49         self.connect("drag-data-received", self.on_drag_data_received)
50     def on_drag_data_received(self, widget, drag_context, x,y, data,info,
time):
51         pixbuf = data.get_pixbuf()
52         width = pixbuf.get_width()
53         height = pixbuf.get_height()
54         print("Sain kätte ikooni laiusega %spx ja kõrgusega %spx" % (width,
            height))
```

Koodinäide 19: Lahtilaskmisala loomine

Valmis programmi (vt joonis 6) koos kommentaaridega on võimalik leida lisat 5.



Joonis 6: Ikoonide näide

Kokkuvõte

Antud seminaritöö eesmärgiks oli tutvustada üht kõige populaarsemat graafilist raamistiku GTK+, mida on võimalik ka kasutada Python’ga läbi PyGl mooduli ja luua sellele ka juhend koos koodinäidetega. Loodud õpetusi peaks suutma järgida õppurid, kes on läbinud Tallinna Ülikoolis kursuse “Programmeerimise alused” ja ka kõik teised kes on huvitatud Python’st või sellele graafiliseliidese loomisest. Peale juhendi läbitöötamist peaks õpilane omandama oskuse luua lihtsamaid kasutajaliideseid GTK+ abil.

Töö on jaotatud kahte ossa. Esimeses osas on tekst üldiselt teoreetiline. Suuremas osas on juttu GTK+ erinevatest vidinatest ja nende fuktsionaalsusest ning eripäradest. Samas on veel ka juttu sellest, mis on GTK+, raamistiku ajaloost, kes ja milleks seda kasutab.

Teine osa on juba aga praktiline. Antud töö osas on lahti seletatud nii raamistiku paigaldamine ja selle nõuded, kui ka õpetused, kuidas raamistikuga kergemaid asju teha . Koodinäidetes on proovitud seletada lahti, mis toimub koodis, kuid on üritatud mitte teksti liiga keeruliseks teha.

Seminaritöö oli ka väga õpetlik mulle, kuna kõike seda pidin ka mina ise õppima, mis tuleneb sellest, et enne seda tööd puudus mul kokkupuude igasuguste Pythoni graafilisteliidestega. Olles ise need õpetused läbi teinud, olen ma kindel, et suudan vähemalt algelisegi kasutajaliidese luua kui tekib vajadus.

Kasutatud kirjandus

Andrew Krause. (2007). Foundations of GTK+ Development New York City: Apress

Geoffrey French (2014, 29. Jaanuar) Developing GTK 3 apps with Python on Windows. Loetud aadressilt: https://blogs.gnome.org/kittykat/2014/01/29/developing-gtk-3-apps-with-python-on-windows/ (04.11.2016)

The Python GTK+ 3 Tutorial (2016). Loetud aadressilt: https://python-gtk-3-tutorial.readthedocs.io/en/latest/index.html (06.11.2016)

The Python Language Reference (kuupäev puudub). Loetud aadressilt: https://docs.python.org/3/reference/index.html (03.11.2016)

LISAD

Lisa 1

Antud lisas on välja kirjutatud kogu peatükis 2.2 valminud programmi kood.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
class Nupud(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="Nupud")
        self.set_border_width(10)
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)
        nupp = Gtk.Button.new_with_label("Vajuta")
        nupp.connect("clicked", self.vajuta_mind)
        vbox.pack_start(nupp, True, True, 0)
        nupp = Gtk.Button.new_with_mnemonic("_Sulge")
        nupp.connect("clicked", self.sulge)
        vbox.pack_start(nupp, True, True, 0)
        nupp = Gtk.ToggleButton("Nupp 1")
        nupp.connect("toggled", self.lylitus, "1")
        vbox.pack_start(nupp, True, True, 0)
        nupp = Gtk.ToggleButton("Nupp 2", use_underline=True)
        nupp.set_active(True)
        nupp.connect("toggled", self.lylitus, "2")
        vbox.pack_start(nupp, True, True, 0)
        nupp1 = Gtk.RadioButton.new_with_label_from_widget(None, "Nupp 1")
        nupp1.connect("toggled", self.lylitus, "1")
        vbox.pack_start(nupp1, False, False, 0)
        nupp2 = Gtk.RadioButton.new_from_widget(nupp1)
        nupp2.set_label("Nupp 2")
```

```
        nupp2.connect("toggled", self.lylitus, "2")
        vbox.pack_start(nupp2, False, False, 0)
        nupp3 = Gtk.RadioButton.new_with_label_from_widget(nupp1,"Nupp 3")
        nupp3.connect("toggled", self.lylitus, "3")
        vbox.pack_start(nupp3, False, False, 0)
        lyliti = Gtk.Switch()
        lyliti.connect("notify::active", self.aktiveeritud)
        lyliti.set_active(False)
        vbox.pack_start(lyliti, True, True, 0)
    def vajuta_mind(self, nupp):
        print("\"Vajuta\" nuppu vajutati")
    def sulge(self, nupp):
        print("Sulgen programmi")
        Gtk.main_quit()
    def lylitus(self, nupp, nimi):
        if nupp.get_active():
            olek = "sisse"
        else:
            olek = "välja"
        print("Nupp", nimi, "lülitati", olek)
    def aktiveeritud(self, lyliti, gparam):
        if lyliti.get_active():
            olek = "sisse"
        else:
            olek = "välja"
        print("Lüliti lülitati", olek)
win = Nupud()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
```

```
Gtk.main()
```

Lisa 2

Antud lisas on välja toodud kogu kood peatükis 2.3 valminud programmist.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk, GObject
class Protsess(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="Protsess")
        self.set_border_width(10)
#vaate loomine
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)
        self.progressbar = Gtk.ProgressBar()
        vbox.pack_start(self.progressbar, True, True, 0)
        self.spinner = Gtk.Spinner()
        vbox.pack_start(self.spinner, True, True, 0)
#loome nupu mis paneb funktsioonid tööle
        button = Gtk.Button("Start")
        button.connect("clicked", self.on_start)
        vbox.pack_start(button, True, True, 0)
#fuktsioon mis paneb protsessi näitajad tööle
    def on_start(self, button):
        self.timeout_id = GObject.timeout_add(50, self.on_start, None)
        self.activity_mode = False
        self.spinner.start()
#teeme while tsükkli, et protsessiriba saaks ainult ühe korra lõppuni töödata
        while (self.progressbar.get_fraction() < 1):
            new_value = self.progressbar.get_fraction() + 0.01
            print (self.progressbar.get_fraction())
            self.progressbar.set_fraction(new_value)
            return False
win = Protsess()
```

```
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Lisa 3

Lisas on välja kirjutatud kogu programmikood peatükist 2.4.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
#List programmide nimest, nende väljalaske aasta, ja kasutatud keel
software_list = [("Firefox", 2002,  "C++"),
                 ("Eclipse", 2004, "Java" ),
                 ("Pitivi", 2004, "Python"),
                 ("Netbeans", 1996, "Java"),
                 ("Chrome", 2008, "C++"),
                 ("Filezilla", 2001, "C++"),
                 ("Bazaar", 2005, "Python"),
                 ("Git", 2005, "C"),
                 ("Linux Kernel", 1991, "C"),
                 ("GCC", 1987, "C"),
                 ("Frostwire", 2004, "Java")]
class TreeViewFilterWindow(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="TreeView-i filtri demo")
        self.set_border_width(10)
     #Sel.grid, kuhu paigutatakse elemendid
        self.grid = Gtk.Grid()
        self.grid.set_column_homogeneous(True)
        self.grid.set_row_homogeneous(True)
        self.add(self.grid)
     #ListStore mudeli loomine
        self.software_liststore = Gtk.ListStore(str, int, str)
        for software_ref in software_list:
            self.software_liststore.append(list(software_ref))
        self.current_filter_language = None
     #Filtri loomine ja sinna ListStore-i lisamine
```

```
        self.language_filter = self.software_liststore.filter_new()
        self.language_filter.set_visible_func(self.language_filter_func)
    #Loome TreeView-i, lisame filtri sinna mudeliks ja lisame veerud
        self.treeview = Gtk.TreeView.new_with_model(self.language_filter)
        for i, column_title in enumerate(["Tarkvara", "Aasta", "Keel"]):
            renderer = Gtk.CellRendererText()
            column = Gtk.TreeViewColumn(column_title, renderer, text=i)
            self.treeview.append_column(column)
    #Nupud, et filtreerida keeled ja nende funktsioonid
        self.buttons = list()
        for prog_language in ["Java", "C", "C++", "Python", "None"]:
            button = Gtk.Button(prog_language)
            self.buttons.append(button)
            button.connect("clicked", self.on_selection_button_clicked)
    #Vaate loomine
        self.scrollable_treelist = Gtk.ScrolledWindow()
        self.scrollable_treelist.set_vexpand(True)
        self.grid.attach(self.scrollable_treelist, 0, 0, 8, 10)
        self.grid.attach_next_to(self.buttons[0], self.scrollable_treelist,
Gtk.PositionType.BOTTOM, 1, 1)
        for i, button in enumerate(self.buttons[1:]):
            self.grid.attach_next_to(button, self.buttons[i],
Gtk.PositionType.RIGHT, 1, 1)
        self.scrollable_treelist.add(self.treeview)
        self.show_all()
    def language_filter_func(self, model, iter, data):
        if self.current_filter_language is None or
self.current_filter_language == "None":
            return True
        else:
            return model[iter][2] == self.current_filter_language
    def on_selection_button_clicked(self, widget):
        self.current_filter_language = widget.get_label()
        print("%s valitud" % self.current_filter_language)
```

```
        self.language_filter.refilter()
win = TreeViewFilterWindow()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Lisa 4

Käesolevas lisas on välja toodud programmikood peatükis 2.5 valminud programmist.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
class ComboBoxWindow(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="rippmenüü Näide")
        self.set_border_width(10)
#Lisame ListStore-sse andmed
        name_store = Gtk.ListStore(int, str)
        name_store.append([1, "Charmion Simion"])
        name_store.append([101, "Jumaane Milla"])
        name_store.append([44, "Mary Sue"])
        name_store.append([5, "Jonathan Joestar"])
        name_store.append([3, "Garry Stue"])
        name_store.append([31, "Dio Brando"])
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
#loome Gtk.combobox-i millel on nii hoidla kui ka sisestuse funktsionaalsus
        name_combo = Gtk.ComboBox.new_with_model_and_entry(name_store)
        name_combo.connect("changed", self.on_name_combo_changed)
        name_combo.set_entry_text_column(1)
        vbox.pack_start(name_combo, False, False, 0)
        self.add(vbox)
#loome funktsiooni mis näitaks, kas valiti listist element, või kirjutati ise
uus
    def on_name_combo_changed(self, combo):
        tree_iter = combo.get_active_iter()
        if tree_iter != None:
            model = combo.get_model()
            row_id, name = model[tree_iter][:2]
            print("valitud: ID=%d, nimi=%s" % (row_id, name))
```

```
        else:
            entry = combo.get_child()
            print("Sisestasid: %s" % entry.get_text())
win = ComboBoxWindow()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Lisa 5

Lisas 5 on programmikood ikoonide pukseermisest, mis valmist peatükis 2.6.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk, Gdk, GdkPixbuf
#importitakse gdk ja gdkpixbuf ikoonide jaoks
(TARGET_ENTRY_TEXT, TARGET_ENTRY_PIXBUF) = range(2)
(COLUMN_TEXT, COLUMN_PIXBUF) = range(2)
DRAG_ACTION = Gdk.DragAction.COPY
class Window(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="Ikoonide demo")
     #vaate loomine
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)
        hbox = Gtk.Box(spacing=10)
        vbox.pack_start(hbox, True, True, 0)
        self.iconview = DragSource()
        self.drop_area = DropArea()
        hbox.pack_start(self.iconview, True, True, 0)
        hbox.pack_start(self.drop_area, True, True, 0)
        self.add_image_targets()
    def add_image_targets(self, button=None):
        targets = Gtk.TargetList.new([])
        targets.add_image_targets(TARGET_ENTRY_PIXBUF, True)
        self.drop_area.drag_dest_set_target_list(targets)
        self.iconview.drag_source_set_target_list(targets)
#loome eraldi klassi kinnivõtualal ja hiljem ka lahtilaskealale
class DragSource(Gtk.IconView):
 #vaade esimesele alale
    def __init__(self):
        Gtk.IconView.__init__(self)
```

```
        self.set_pixbuf_column(COLUMN_PIXBUF)
        #ikoonide loomine kinnivõtmisalale
        model = Gtk.ListStore(str, GdkPixbuf.Pixbuf)
        self.set_model(model)
        self.add_item("Ikoon 1", "edit-paste")
        self.add_item("Ikoon 2", "edit-cut")
        self.add_item("Ikoon 3", "edit-copy")
        self.enable_model_drag_source(Gdk.ModifierType.BUTTON1_MASK, [],
            DRAG_ACTION)
        self.connect("drag-data-get", self.on_drag_data_get)
 #funktsionaalsus ikooni kinnivõtmisel
    def on_drag_data_get(self, widget, drag_context, data, info, time):
        selected_path = self.get_selected_items()[0]
        selected_iter = self.get_model().get_iter(selected_path)
        pixbuf = self.get_model().get_value(selected_iter, COLUMN_PIXBUF)
        data.set_pixbuf(pixbuf)
 #ikoonide loomis funktsioon
    def add_item(self, text, icon_name):
        pixbuf = Gtk.IconTheme.get_default().load_icon(icon_name, 16, 0)
        self.get_model().append([text, pixbuf])
class DropArea(Gtk.Label):
 #loome vaate sellele alale
    def __init__(self):
        Gtk.Label.__init__(self, "Nihuta siia!")
        self.drag_dest_set(Gtk.DestDefaults.ALL, [], DRAG_ACTION)
        self.connect("drag-data-received", self.on_drag_data_received)
 #funktsionaalsus selleks kui programm saab signaali, et ikoon on antud alale
lahti lastud
    def on_drag_data_received(self, widget, drag_context, x,y, data,info,
time):
        pixbuf = data.get_pixbuf()
        width = pixbuf.get_width()
        height = pixbuf.get_height()
        print("Sain kätte ikooni laiusega %spx ja kõrgusega %spx" % (width,
```

```
          height))
win = Window()
win.connect("delete-event", Gtk.main_quit)
win.show_all()
Gtk.main()
```