TARTU ÜLIKOOL

Arvutiteaduste instituut

Informaatika õppekava

Keili Pedel

E-kursuse "Programmeerimise alused" logifailide analüüs

Bakalaureusetöö (9 EAP)

Juhendaja: Eno Tõnisson

Tartu 2016

E-kursuse "Programmeerimise alused" logifailide analüüs

Lühikokkuvõte:

Programmeerimise õpetamisel saab õppejõud tavaliselt õppijate arengust ülevaate valmis programmide põhjal. Seega puudub neil sageli informatsioon õppijate lahendusprotsessi kohta. Seda eelkõige e-kursustel, kus õpilased lahendavad oma ülesandeid ilma kohalkäimiseta. Lahendusprotsessi uurimine võimaldab õppejõududel saada paremat informatsiooni õppijate kohta ja seega anda neile paremat tagasisidet. Samuti on see abiks ka õppematerjalide ja ülesannete parandamisel.

Üheks lahendusprotsessi uurimise mooduseks on Thonny kasutajate logifailide uurimine. Antud bakalaureusetöö käigus uuritakse Thonny logifailide sisu ning luuakse programm, mis koostab iga logifaili kohta kokkuvõtte.

Võtmesõnad:

Programmeerimise e-kursus, Thonny, logifailid

CERCS: P175

Analyse of e-course “Introduction to Programming” log files

Abstract:

Lectors often lack information on students’ methods of programming. It is even more relevant on e-courses, where students solve their exercises without attending a lecture. Therefore lecturers would need a way to help them understand students’ exercise solving.

One of these ways is to explore logfiles of Thonny’s users. This Bachelor’s thesis focuses on analysing Thonny’s log files content and making a program, which compiles a synopsis of each log file.

Keywords:

Programming e-course, Thonny, log files

CERCS: P175

Sisukord

1. Sissejuhatus 5

2. Programmeerimise algõpe 7

2.1 Programmeerimise algkursused 7

2.1 „Programmeerimise alused“ 7

2.2 Programmeerimiskeskkond Thonny 8

3. Õppijate tegevuste jälgimine ülesannete lahendamise jooksul 12

3.1 Sarnased tööd 12

3.2 Õppijate tegevuste jälgimine Thonny logimise funktsionaalsust kasutades 13

4. Thonny logifailide analüüsimine 15

4.1 Thonny kirjete tüübid 15

4.1.1 Thonny programmiaknaga seotud kirjed 15

4.1.2 Programmikoodiga seotud kirjed 16

4.1.3 Failidega seotud kirjed 19

4.1.4 Programmi käivitamisega seotud kirjed 21

4.1.5 Käskudega seotud kirjed 22

4.2 Logifaili kokkuvõte 22

4.2.1 Õppija kood 23

4.2.2 Kirjete arv logifailis 23

4.2.3 Sessiooni alustamise ja lõpetamise aeg 24

4.2.4 Teksti sisestamiste ja kustutamiste arv redaktoris 24

4.2.5 Teksti kleepimiste ja kopeerimiste arv 25

4.2.6 Faili salvestamiste arv 25

4.2.7 Programmi käivitamiste arv 26

4.2.8 Veateadete arv 26

4.2.9 Failide avamiste arv 26

4.2.11 Sessiooni ajal kasutatud failide nimed 27
5. Thonny logifailide kokkuvõtte kasutamise näidis 28
5.1 Andmed 28
5.2 Thonny logifaili kokkuvõtte kasutamise näide programmi käivitamiste näitel 29
5.2.1 Käivitamiste arv 29
5.2.2 Salvestamiste ja käivitamiste arvude vahe 29
6. Kokkuvõte 31
Kasutatud kirjandus 32
Lisad 33
I Programmi „Tere maailm!“ loomise ja selle ühekordse käivitamise kajastamine Thonny logifailis 33
II Thonny logifailide esitamise juhendid 36
JUHEND 1 36
JUHEND 2 36
Videojuhend 36
III Logifailide kokkuvõtte tabel 37
IV Logifailide kokkuvõtet koostav programm 38
V Litsents 39

1. Sissejuhatus

Tartu Ülikool pakub nii osalejate arvu kui ka mahu poolest suuremaid ja väiksemaid programmeerimise teemalisi e-kursuseid. Hetkel on kõige suurem Tartu Ülikooli programmeerimise teemaline e-kursus „Programmeerimise alused“. Tegemist on programmeerimise algkursusega, kus omandatakse esmased teadmised programmeerimise kohta, kasutades selleks programmeerimiskeelt Python.

Kui tavakursuste raames toimuvad praktikumid, kus õppejõududel on võimalik õppijate lahendamisprotsessi mingil määral otseselt jälgida, siis e-kursuste puhul selline võimalus puudub. Ülesannete lahendamise kohta saavad õppejõud informatsiooni ainult esitatud lahenduste näol. Seetõttu oleks õppejõududel kasulik, kui leiduks moodus, kuidas õppijate käekäiguga kursuse jooksul rohkem kursis püsida. Üheks selliseks võimaluseks on õppijate poolt esitavate logifailide uurimine.

„Programmeerimise aluste“ kursusel kasutatakse ülesannete lahendamiseks Tartu ülikooli õppejõu Aivar Annamaa loodud programmeerimiskeskkonda Thonny. Thonny on programmeerimiskeskkond Pythoni jaoks, mis on mõeldud eelkõige õppimiseks ja õpetamiseks. Thonny üheks oluliseks funktsionaalsuseks on programmis tehtavate toimingute talletamine logifaili. Thonny logifailidest on võimalik kätte saada palju informatsiooni, mis annaksid õppejõududele teavet õppija tegevuste kohta ülesannete lahendamisel. Saadav informatsioon peaks aitama leida õppijaid, kes on ülesannete lahendamisega hädas ja vajavad abi kui ka neid õppijaid, kes on lahendamisel olnud väga leidlikud ja kelle töökäigu uurimine oleks huvitav. Samuti oleks see informatsioon abiks õppejõududele õppematerjalide ja ülesannete paremaks muutmisel.

Thonny logifailid sisaldavad endas palju kirjeid ja seega on nad ka mahult suured. Kirjed sisaldavad omakorda palju andmeid, mis võivad logifailidest arusaamise muuta keerukaks. Seega oleks logifailide käsitsi läbivaatamine õppejõududele suur lisatöö. Selleks, et logifailidest võimalikult palju infot kätte saada, on tuleb nende ülevaatamine automatiseerida.

Antud bakalaureusetöö käigus uuritakse Thonny logifailide sisu ja luuakse programm, mis automatiseerib Thonny logifailide ülevaatamist ning genereerib iga logifaili kohta kokkuvõtte, mis lisatakse tabelisse.

Töö esimeses peatükis tutvustatakse „Programmeerimise aluste“ e-kursust ja selles kasutatavat programmeerimiskeskkonda Thonny. Teises peatükis kirjeldatakse, millistel moodustel on õpilaste tegevusi varem jälgitud, ning kuidas seda teha kasutades Thonny logifaile. Kolmandas

peatükis kirjeldatakse, milliseid andmed on Thonny logifailis. Lisaks sellele, kirjeldatakse antud peatükis ka, milliseid andmeid kasutades genereerib programm logifaili kokkuvõtte. Neljandas peatükis tuuakse konkreetne näide, kuidas logifaili kokkuvõtet on võimalik kasutada.

2. Programmeerimise algõpe

Ülikoolid pakuvad erinevaid kursuseid, mis annavad õppijale esmased teadmised programmeerimise kohta. Osad kursused on tavakursused ning osad e-kursused. Mõlema puhul saavad õppejõud õppijate lahendusprotsessi kohta suhteliselt vähe informatsiooni.

2.1 Programmeerimise algkursused

Tavakursuste puhul toimuvad praktikumid ning loengud, millele järgnevad kodused ülesanded. Antud kursuste juures on positiivne, et õppejõududel on võimalik praktikumides õppija tööd mingil määral jälgida ning selle põhjal vajadusel personaalset tagasisidet anda. Seega on raskustes olevaid õppijaid kergem tuvastada ja aidata. Sellegipoolest hinnatakse üldiselt kodutöid ainult lõpplahenduse järgi ning nende lahendamisekäigu kohta kodus puudub õppejõududel täpsem informatsioon.

Lisaks tavakursustele korraldatakse ülikoolides ka mitmeid suuremaid ja väiksemaid e-kursuseid. Üldiselt sellised kursused ei nõua kohalkäimist ja tänu sellele on võimalik suurendada kursusel osalejate arvu mitmekordseks. Antud kursustel on üldiselt iga nädala kohta konkreetsed teemad, mille kohta tuleb õppijal iseseisvalt lugeda ja seejärel lahendada teemakohased ülesanded. Sealjuures on suur ka saadetavate lahenduste arv, mida peaksid õppejõud kontrollima ja hindama. Seega kasutatakse e-kursustel ülesannete kontrollimiseks eelkõige automaatkontrolli. Automaatkontrolli kasutamine on sellistel kursustel kasulik, kuna see kontrollib õppija töö korrektsust ning funktsionaalsusi ning sealjuures vähendab olulisel määral õppejõudude tööhulka [1]. Samas pole õppejõud kursis ka õppijate käekäiguga ning seega saavad õppijad enamasti tagasisidet ainult automaatkontrollilt.

Kuna õppejõud saavad vähe infot õppija lahendusprotsessi kohta, siis oleks vaja mingisugust moodust, mis aitaks õppejõududel jälgida õppijate lahendusprotsessi. Selle kaudu saaks õppijad ka rohkem personaalset tagasisidet. Eriti oluline on see algkursuste puhul, kus enamik õppijatest puutuvad programmeerimisega esimest korda kokku. Sellises olukorras teevad õppijad kõige rohkem vigu ja jäävad hätta. Paljud hättajäänutest õppijatest ei lõpeta antud kursusi edukalt, kuna saavad vähe tagasisidet ning ei tea täpselt, kus neil vead tekkisid.

2.1 „Programmeerimise alused“

„Programmeerimise alused“ on üks Tartu ülikooli poolt pakutavatest programmeerimise algkursustest. Antud kursus on suunatud neile, kellel puuduvad eelnevad kokkupuuted program-

meerimisega, mis annab neile võimaluse saada selle kohta algteadmisi. Programmeerimiskeelena kasutatakse antud kursuse raames Pythonit. Antud kursus on nii tavakursuse kui ka e-kursuse vormis.

„Programmerimise aluste“ pilootkursus e-kursusena toimus 11. jaanuarist kuni 6. märtsini 2016. Antud kursusel osales 295 õppijat, kellest 145 lõpetasid. 28. märtsil algas ka uus kursus, kus alustas 1770 õppijat. Antud kursus kestab 22. maini 2016. Seega on käimasolev kursus siiamaani osalejate arvu poolest suurim programmerimise e-kursus, mis on Tartu ülikooli poolt korraldatud [2].

Õppijate arv on „Programmerimise aluste“ e-kursusel suur. Seega on ka esitatavate kodutööde arv suur ning nende kontrollimiseks on kõige mõistlikum kasutada automaatkontrolli. Seetõttu saavad õppijad tagasisidet üldiselt ainult automaatkontrollilt. Samuti on antud kursusel kasutusel ka murelahendaja (*troubleshooter*), mis annab õppijatele samm-sammulisi juhendeid, et leida oma vigu [3]. See peaks vähendab õppejõududelt küsitavate küsimuste hulka, kuna annab õppijatele palju erinevaid vihjeid, mis peaksid neid aitama. Niisiis saavad õppejõud informatsiooni õppijate automaatkontrollile esitatud lõpplahenduse kaudu. Samuti saavad nad näha ka, milliseid vihjeid otsitakse murelahendajast, kuid seda vaid üldiselt ning mitte konkreetse õppija kohta. Sellegipoolest puudub ülevaade õppija lahendusprotsessi kohta.

Põhiliseks „Programmeerimise aluste“ kursusel kasutatavaks programmeerimiskeskkonnaks on Tartu ülikoolis arendatud õppimiseks ja õpetamiseks mõeldud tarkvara Thonny [4]. Thonny loob õppija iga kasutussessiooni kohta logifaili. Antud logifailides on palju informatsiooni õppija tegevuste kohta programmi loomise ajal, seega võiks antud informatsioon huvitada ka õppejõude pakkumaks informatsiooni õppija lahendusprotsessi kohta. Kuna Thonny lisab oma logifailidesse palju infot, siis nende käsitsi läbivaatamine pole õppejõudude jaoks mõistlik. Seega on vaja mingisugust abivahendit, mis koostaks logifailidest kokkuvõtte, millest oleks õppejõul võimalik välja lugeda erinevaid näitajaid, mille põhjal oleks võimalik kergesti teha järeldusi õppija tööprotsessi kohta. Näiteks on võimalik Thonny logifailides eristada ükshaaval sisestatud sümboleid ja kleepimise funktsionaalsust. Kleepimised võivad viidata ka osadel juhtudel plagiaadile. Seega on võimalik kahtluse korral logifaili täpsemalt uurida ja samuti kasutada Thonny logifaili taasesitamise funktsionaalsust.

2.2 Programmeerimiskeskkond Thonny

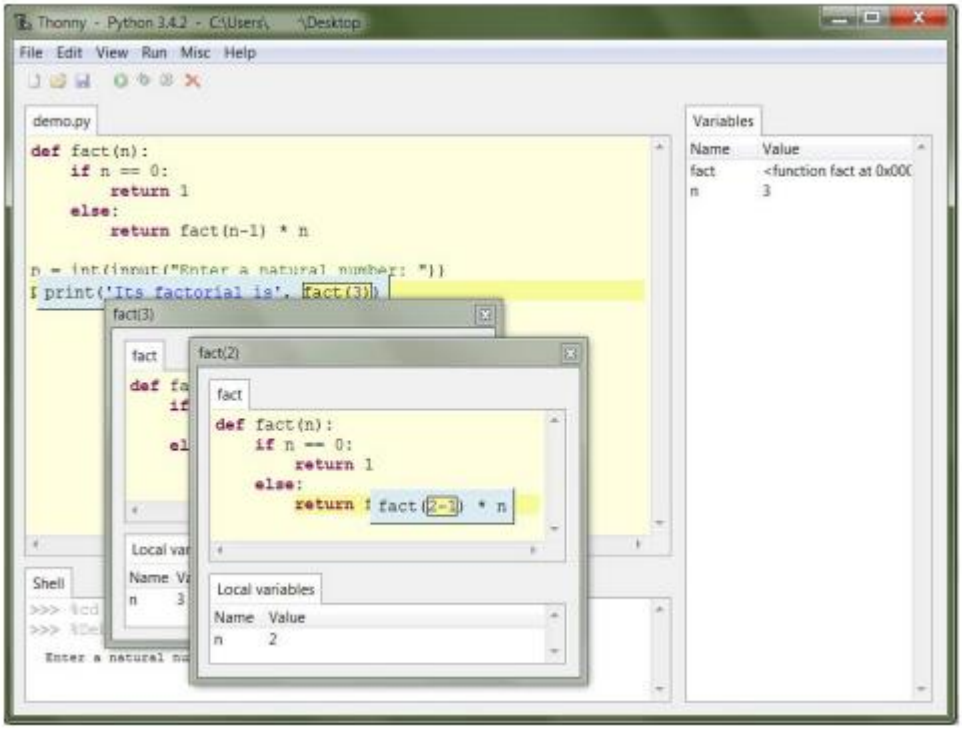
Thonny on Aivar Annamaa poolt arendatud vabavaraline programmeerimiskeskkond, mis on loodud programmeerimiskeele Python jaoks. Antud keskkond on mõeldud eelkõige program-

meerimise õppimiseks ja õpetamiseks. Thonny loomisel oli oluline, et see sisaldaks endas võimalikult palju funktsionaalsusi, mis on eelkõige esimesel programmeerimise kursusel õppijatele vajalikud. Seejuures peab Thonny kasutajaliides jääma võimalikult lihtsaks ja kasutajale mugavaks [5].

Thonny on heaks vahendiks, mille abil saaks lisada programmi visualiseerimise algaja õppija tavapärasesse programmeerimise töövoogu [6]. Visualiseerimine tähendab Thonnys, et kasutajale kuvatakse programmi töökäik sammhaaval. Programmi visualiseerimine peaks aitama õppijal programmi käitumisest paremini aru saada. Thonny sai loodud eelkõige aitamaks õppijaid, kes on programmeerimise õppimise juures hädas ja vajavad vahendit, mis neid ülesannete lahendamisel aitaks [6]. Programmi visualiseerimine on õppijale kasulik, kuna see aitab õppijal mõista, kuidas programm täpsemalt töötab. Seega on õppijal võimalus leida programmis vigane koht ja seda parandada. Sammhaaval programmi käitumise uurimine pole ainult vigade leidmiseks, vaid see on kasulik ka üldises plaanis, ning näitab kasutajale täpsemalt programmi käitumist.

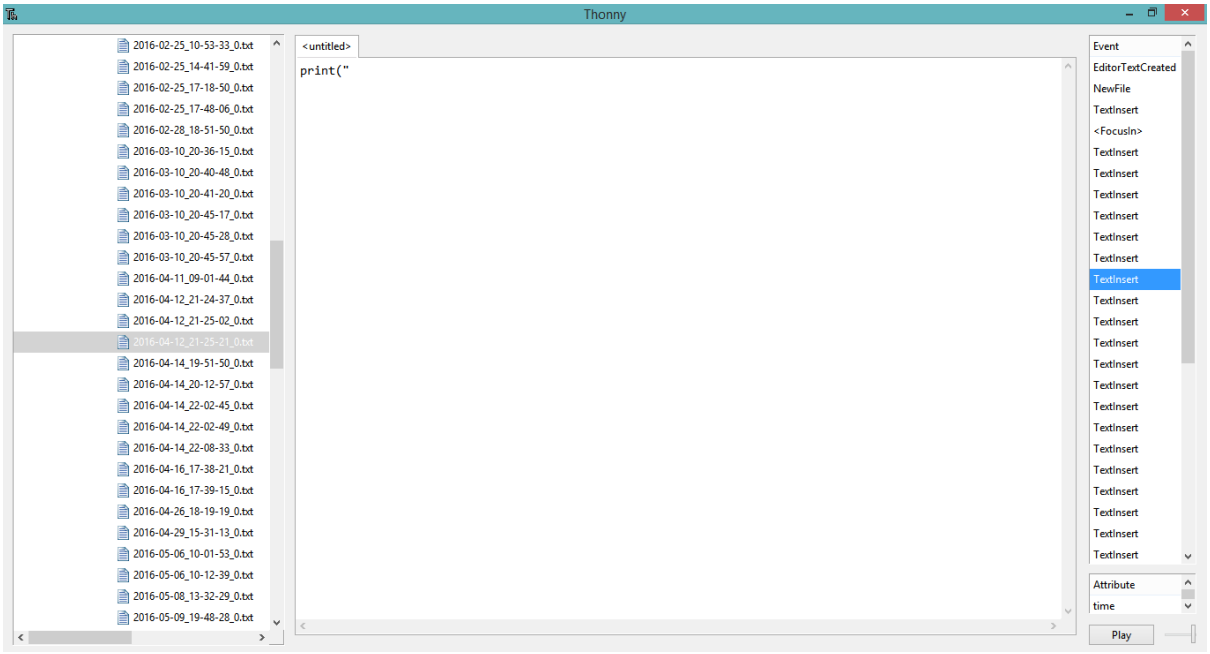
Pythoni põhiline programmeerimiskeskkond (IDLE) on suhteliselt minimaalsete võimalustega. Thonny katab kõik Pythoni standardse IDLE funktsionaalsused, kuid lisaks sellele on Thonnyl ka selliseid funktsionaalsusi, mida IDLE'il pole. Näiteks asub Thonny käsurida koodiredaktoriga samas vaates ning seega on see kasutaja jaoks kergemini jälgitav. Erinevalt Pythoni standardsele programmeerimiskeskkonnale IDLE, kus iga avatud programmikood asub uues aknas, on Thonnys sama olukord lahendatud vahekaartide abil, mis lihtsustab kasutajal programmikoodide vahel opereerimise. Lisaks sellele on olemas ka failide brauserivaade (*browser-view*), mille abil on kerge faile otsida ja avada [5].

Thonnyl on funktsionaalsus, mis näitab kasutajale programmi tööd sammhaaval (*debug*). Kõik programmis esinevad muutujad asuvad ühes vaates (joonis 1), kus on nähtav kõik, mis nendega programmi töötamise jooksul toimub. See aitab kasutajal programmiga paremini kaasa mõelda, ning analüüsida selle õigsust ja saada paremini aru selle programmi töö põhimõttest.



Joonis 1. Thonny programmi silumise (debug) vaade, kus kasutajale kuvatakse programmi käitumine sammhaaval.

Antud bakalaureusetöö jaoks on Thonny kõige olulisem funktsionaalsus kasutaja tegevuste kohta logifaili kirjutamine. Antud logifailidest on võimalik kätte saada erinevaid andmeid kasutaja tegevuste kohta, mis annavad head informatsiooni õppija lahendusprotsessi kohta. Thonny abil on võimalik ka kasutussessiooni taasesitada (joonis 2), mille käigus on võimalik logifaili põhjal terve kasutussessioon läbi mängida valitud kiirusega.



Joonis 2. Thonny logifaili taasesitamine vaade.

Thonnyt on kõige kasulikum kasutada programmeerimise algkursustel, seega on Thonnyt kasutatud Tartu ülikoolis mõndade programmeerimise algkursuste käigus. Õppijad on selle hästi vastu võtnud ja enamik kursuse läbijatest leidis, et Thonnyst oli eelkõige palju abi programmist vigade otsimisel ja nende parandamisel. Õppijatele meeldib ka see, et käsurida on redaktoriga samas vaates ning tänu sellele on see paremini jälgitav [6].

3. Õppijate tegevuste jälgimine ülesannete lahendamise jooksul

Õppejõududele jõuavad üldiselt ainult ülesannete lõplikud lahendused, millest pole võimalik kätte saada informatsiooni õppija tegevuste kohta programmi loomisel. Samas võib õppija programmi loomise protsessi uurides välja tulla huvitavat informatsiooni selle õppija harjumuste kohta programmikoodi kirjutamisel.

3.1 Sarnased tööd

Kõige levinum viis õppijate programmi loomise protsessiga kursis püsimiseks on esitatav lahendus [7]. Kui õppejõududele saabub ainult üks fail, siis selle järgi on võimalik ainult õppijale hinnet panna, õppija lahendusprotsessi pole täpsemalt võimalik selle järgi uurida.

Täpsemalt on võimalik õppija töökäiku uurida, kui koguda õppija ülesannete lahendamise käigus hetkevõtteid (*snapshots*) [7]. Helsingi ülikoolis on mitmetel kursustel kasutatud programmeerimiskeskkonda NetBeans koos automaatse kontrollimise teenusega Test My Code (TMC). TMC sisaldab NetBeans'i pistikprogrammi, mis annab õppijale juhiseid ülesannete lahendamise kohta ning testib lahenduste õigsust. Samal ajal teeb TMC ka hetkevõtteid õppija programmikoodi kohta igal programmikoodi salvestamisel. Hetkevõtetes on kajastatud aeg ja koodis tehtud muutused võrreldes eelneva hetkevõttega. Antud hetkevõtteid on võimalik kasutada õppijate lahendusprotsessi uurimiseks ning selle abil anda õpilastele paremat tagasisidet. Seega esitatakse koos lõplike ülesannete lahendustega ka hetkevõtted [1].

2012. aasta sügisel kasutati Helsingi ülikoolis programmeerimise algkursuse jooksul TMC abil kogutud hetkevõtteid uurimaks õppijate programmeerimise töökäiku. Perkins'i järgi jagunevad algajad programmerimise õppijad kolmeks: peatujad (*stoppers*), nikerdajad (*tinkers*) ja edasiliikujad (*movers*) [1]. Antud kursusest tuli välja, et enamik kursusel osalejatest olid niinimetatud ehitajad (*builders*), kes on Perkins'i järgi edasiliikujad. Nad lahendavad ülesandeid järjest edasi ja läbivad järjest rohkem teste [1]. 2013. aastal kasutati samuti hetkevõtteid ning uurimise käigus eraldati head lahendamise tavad kehvadest. Saadud andmete põhjal suudeti peale teist õppenädalat 64% täpsusega kindlaks teha, kas antud õpilane läbib kursuse edukalt või kukub selle läbi [8].

Samuti on võimalik õppijate ülesannete lahendamisi uurida nupuvajutuste (*key-stroke*'ide) abil, mis kajastavad kõiki programmikoodis tehtud muudatusi lahendamise ajal. Antud võimalus kogub täpsemat infot ja seega on võimalik selle abil õppijaid ka täpsemalt uurida [7]. 2014. aasta sügisel kasutati Helsingi ülikoolis Java kursusel süsteemi, mis tuvastas kõik nupuvajutused, mis tehti NetBeans'is. Selle kaudu uuriti, kas õppijat on võimalik tuvastada nupuvajutuste

järgi. See uuring on kasulik eelkõige plagiaadi küsimuste tõttu. Uuringu kohaselt on see võimalik, kuid selleks on vaja väga suurt hulka andmeid [7].

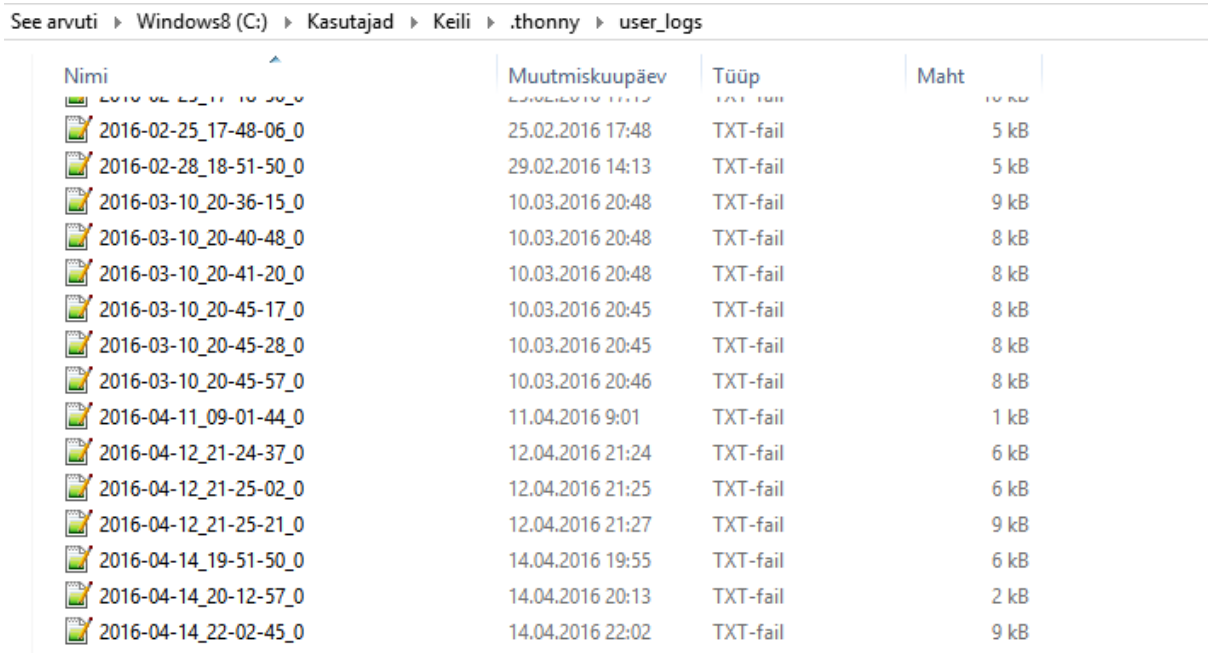
Lisaks neile võimalustele on võimalik kasutada ka programmeerimiskeskkondade poolt salvestatud kirjeid programmi käitumise kohta või jälgida õppijat mingil muul võimalusel [7].

3.2 Õppijate tegevuste jälgimine Thonny logimise funktsionaalsust kasutades

Selleks, et jälgida Thonny kasutaja tegevust ülesannete lahendamisel, on üheks võimaluseks uurida Thonny logifaile.

Programmeerimiskeskkond Thonny loob iga kasutussessiooni kohta tekstiformaadis (.txt) logifaili, mis sisaldab kirjeid kasutaja poolt sooritatud tegevuste kohta [5]. Thonny kasutussessioon algab Thonny avamisega ning lõppeb Thonny sulgemisega. Thonny logifailid on JavaScript Object Notation (JSON) formaadis. JSON on tekstipõhine andmete vahetuse formaat, mis hoiab andmeid hästi organiseeritult ja seega on inimesele hästi loetav [9].

Logifailid salvestatakse vaikimisi *.thonny* kaustas asuvasse alamkausta *user_logs* (joonis 3). Logifaili nimeks on kasutussessiooni algusaja ajatempel formaadis YYYY-MM-DD_hh-mm-ss.



Joonis 3. Thonny kausta user_logs sisu.

Logifailid sisaldavad kirjeid kasutaja erinevate tegevuste kohta Thonnys. Logifailis on kajastatud näiteks failide avamised ja salvestamised, programmikoodi muutmised ja programmi käivitamised [5]. Antud lõputöö raames on jagatud kirjed erinevateks tüüpideks. Näiteks esinevad

logifailides programmikoodiga seotud kirjed, käsukirjed, failidega seotud kirjed, Thonny programmiaknaga seotud kirjed ja programmi käivitamisega seotud kirjed. Kuigi kirjete sisu on mõningate erinevustega, siis kõik kirjed sisaldavad endas toimumise ajatemplit.

Thonny loob logifaile peamiselt oma taasesitamise funktsionaalsuse jaoks, mille puhul saab kasutaja valida endale logifaili, mida taasesitatakse ning samuti ka kiiruse, millega seda kasutajale kuvatakse [5]. Samuti on võimalik logifaile uurides saada informatsiooni kasutaja tegevuste kohta kasutussessiooni käigus. Erinevalt sarnastest töödest uuritakse Thonny logide puhul lisaks programmikoodi muutustele ka õppija harjumusi programmikoodi käivitamisel ja failide vahel opereerimist.

4. Thonny logifailide analüüsimine

Thonny genereerib iga kasutussessiooni kohta tekstiformaadis (.txt) logifaili, mis kirjeldab kasutaja sooritatud toiminuid Thonnys [5]. Üheks kasutussessiooniks loetakse Thonny puhul perioodi, mis algab Thonny avamisega ning lõppeb Thonny sulgemisega. Paljudel inimestel on kombeks programme pikaks ajaks lahti jätta. Seega võivad Thonny kasutussessioonid olla väga pikad, isegi kui selle aja jooksul on olnud kasutamises pikki pause.

4.1 Thonny kirjete tüübid

Thonny logifail on JSON formaadis. Thonny logifail koosneb kirjetest, kus kirje asub loogeliste sulgude sees. Kirjed on üksteisest eraldatud komadega. Kirjete arv oleneb toimingute arvust, näiteks teksti sisestamistest ja kustutamistest antud kasutussessiooni jooksul.

Kirje koosneb omakorda võti-väärtus paaridest, kus esimesel kohal on võti ning teisel kohal selle väärtus (joonis 4). Sarnaselt kirjetele on ka võti-väärtus paarid eraldatud üksteisest komadega. Igal kirjetüübil on oma kindel arv, mitut võti-väärtus paari see sisaldab. Näiteks koosneb teksti sisestamise kirje 7 võti-väärtus paarist. Võti-väärtus paarid on kirjete lõikes erinevad, kuid iga kirje sisaldab endas kindlasti ajatemplit, mil antud toiming toimus.

```
[
{
<võtmesõna>: <väärtus>,
...,
<võtmesõna>: <väärtus>,
},
...,
{
<võtmesõna>: <väärtus>,
...,
<võtmesõna>: <väärtus>,
}

]
```

Joonis 4. Logifaili üldine struktuur.

4.1.1 Thonny programmiaknaga seotud kirjed

Lisaks programmikoodiga seotud kirjetele salvestab Thonny oma logifaili mitmeid kirjeid Thonny programmiaknaga seotud toimingute kohta. Sellised kirjed on näiteks tekstiredaktori loomine Thonny programmiaknasse, kus hakatakse programmikoodi looma (joonis 5) ja Thonny programmiaknas erinevate nuppude kasutamised (joonis 6). Lisaks sellele kajastab Thonny oma logifailides ka ajad, millal kasutaja Thonnyt aktiivsena hoidis ja samuti ka need hetked, mil kasutaja Thonnyt minimeeris (joonis 7).

```
{
        "sequence": "EditorTextCreated",
        "text_widget_id": 58089360,
        "time": "2016-03-17T17:34:58.752747",
        "text_widget_class": "Text",
        "editor_id": 33576304,
        "editor_class": "Editor"
    }
```


Joonis 5. Logifaili kirje koodiredaktori loomise kohta Thonny programmiaknasse.


```
{
        "widget_id": 47486096,
        "time": "2016-03-12T19:25:11.734425",
        "sequence": "<Button-1>",
        "widget_class": "Scrollbar"
    }
```


Joonis 6. Thonny logifaili kirje nupu kasutamise kohta.


```
{
        "widget_id": 9514960,
        "time": "2016-03-12T19:16:28.548739",
        "sequence": "<FocusOut>",
        "widget_class": "Workbench"
    },
{
        "widget_id": 9514960,
        "time": "2016-03-12T19:16:31.804802",
        "sequence": "<FocusIn>",
        "widget_class": "Workbench"
    }
```


Joonis 7. Thonny logifaili kirjed Thonny akna minimeerimise ja Thonny akna aktiviseerimise kohta.

4.1.2 Programmikoodiga seotud kirjed

Thonnys on võimalik programmikoodiga sooritada erinevaid toiminguid. Programmikoodi on võimalik redaktorisse sisestada ja kustutada. Samuti on võimalik koodi ka redaktorist kopeerida.

Programmikoodi sisestamiseks redaktorisse on kaks võimalust: käsitsi trükkimine kasutades klaviatuuri või olemasoleva teksti kleepimine kasutades funktsionaalsust. Thonny logifaili kirjete abil on võimalik neid omavahel eristada. Logifailidesse lisatakse igasuguse programmikoodi sisestamise puhul teksti sisestamise kirje (joonis 8).

```
{
    "text_widget_id": 47485072,
    "text_widget_class": "Text",
    "index": "12.0",
    "sequence": "TextInsert",
    "time": "2016-03-12T19:25:20.996768",
    "tags": "None",
    "text": "p"
}
```


Joonis 8. Logifaili kirje programmikoodi sisestamise kohta.

Antud kirje koosneb 7 võti-väärtus paarist.

- *"sequence": "TextInsert"* näitab, et tegemist on programmikoodi sisestamisega.
- *"text_widget_class": "Text"* näitab, millisesse Thonny akna elementi on antud tekst sisestatud ehk antud juhul on tekst sisestatud redaktorisse.
- *"time": <aeg>* näitab, millal antud programmikood redaktorisse sisestati.
- *"index": <reanumber . mitmes element reas>*, näitab positsiooni, kuhu antud programmikood sisestati. Arv enne punkti näitab, millisele reale sümbol lisati ning arv peale punkti näitab, mitmendaks sümboliks see antud reas on. Positsiooni loendamine algab nullist.
- *"text": <sisestatud tekst>* näitab, milline sümbol(id) sisestati.

Logikirje sisaldab ka teisi elemente, kuid need on olulised Thonnyle taasesitamisel, ning täpsemat lahti seletamist ei vaja.

Kleepimise funktsionaalsust kasutades tekib logifaili kaks kirjet. Logifaili tekib teksti lisamise kirje (joonis 8), erinevalt ühe tähe sisestamisest on võti-väärtus paari *"text": <sisestatud tekst>* pikkus pikem kui üks sümbol. Näiteks võib olla võti-väärtus paar *"text": „Lisatakse pikem tekst."* . Lisaks teksti sisestamise kirjele tekib logifaili kleepimise kirje (joonis 9).


```
{
    "widget_id": 47485072,
    "text_widget_id": 47485072,
    "text_widget_class": "Text",
    "sequence": "<<Paste>>",
    "widget_class": "Text",
    "time": "2016-03-12T19:50:57.069035"
}
```


Joonis 9. Thonny logifaili kirje kleepimise funktsionaalsuse kasutamise kohta..

Kleepimise kirje koosneb 6 võti-väärtus paarist.

- *"sequence": "<<Paste>>"* näitab, et tegemist on kleepimise funktsionaalsuse kasutamisega.
- *"text_widget_class": "Text"* näitab , et kleepimine toimus redaktoris.
- *"time“: <aeg>* näitab, millal kleepimise funktsionaalsust kasutati.

Teksti sisestamisel kehtib ka üks erand, mis on seotud juba olemasoleva programmikoodi sisestamisega redaktorisse. Antud toiming toimub olukorras, kus avatakse fail. Erinevalt kleepimisele ei eelne mingit eristavat kirjet, mis näitab, et tegemist on esmase teksti lisamisega redaktorisse.

Tegemist on tavalise programmikoodi sisestamise käsuga (joonis 8), kuid seda on võimalik eristada selle võti-väärtus paari *"text": <sisestatud tekst>* sisestatud teksti pikkuse järgi. Kui see on pikem kui üks sümbol ja sellele ei eelne kleepimise kirjet (joonis 9), siis on tegemist avatud faili sisu sisestamisega redaktorisse, pärast seda on võimalik faili sisu redaktorit kasutades muuta.

Programmikoodi kustutamisel redaktorist luuakse logifaili teksti kustutamise kirje (joonis 10). Programmikoodi on võimalik kustutada kahte moodi, kas ühe või mitme sümboli kaupa korraga.


```
{
    "index2": "",
    "index1": "13.11",
    "text_widget_class": "Text",
    "sequence": "TextDelete",
    "time": "2016-03-12T19:57:49.262767",
    "text_widget_id": 47485072
}
```


Joonis 10. Thonny logifaili kirje programmikoodist kustutamise kohta.

Programmikoodi kustutamise kirje koosneb 6 võti-väärtus paarist.

- *"sequence": "TextDelete"* näitab, et tegemist on programmikoodi kustutamisega.
- *"text_widget_class": "Text"* näitab, millisesse Thonny programmiakna elemendist on antud tekst kustutatud. Antud juhul on tekst kustutatud redaktorist.
- *"time“: <aeg>* näitab, millal antud programmikood redaktorist kustutati.
- *"index1": <reanumber . mitmes element reas>* näitab positsiooni, millest alates programmikoodi kustutati. Arv enne punkti näitab, millisest reast sümbol(eid) kustutati ning arv peale punkti näitab, mitmendaks sümboliks see antud reas on. Positsioonide loendamine algab nullist.

- *"index2": <reanumber . mitmes element reas>* näitab positsiooni, milleni programmikoodi kustutati. Kui antud võti-väärtus paari väärtus on tühi sõne, siis kustutati ainult üks sümbol. Arv enne punkti näitab, millises reas viimane kustutatud sümbol asus ning arv peale punkti näitab mitmendaks sümboliks see antud reas on. Positsioonide loendamine algab nullist.

Erinevalt programmikoodi sisestamisest pole võimalik logifailist täpselt välja lugeda, milline oli kustutatud tekst.

Programmikoodiga seotud kirjete hulka kuulub ka programmikoodi kopeerimise kirje (joonis 11). Antud kirje tekib, kui kasutaja kopeerib redaktorist mingi osa programmikoodi.


```
{
    "widget_id": 47485072,
    "text_widget_id": 47485072,
    "text_widget_class": "Text",
    "sequence": "<<Copy>>",
    "widget_class": "Text",
    "time": "2016-03-12T20:05:23.914190"
}
```


Joonis 11. Thonny logifaili kirje kopeerimise funktsionaalsuse kasutamise kohta.

Programmikoodi kopeerimise kirje koosneb 6 võti-väärtus paarist.

- *"sequence": "<<Copy>>"* näitab, et tegemist on kopeerimise funktsionaalsuse kasutamisega.
- *"text_widget_class": "Text“* näitab, et kopeerimine toimus redaktoris.
- *"time“: <aeg>* näitab, millal kopeerimise funktsionaalsust kasutati.

Analoogsed teksti muutmise kirjed kehtivad ka käsurea puhul, kuid võti-väärtus paari *"text_widget_class": "Text"* asemel on neis kasutatud *"text_widget_class": "Shell"* .

4.1.3 Failidega seotud kirjed

Thonnys on võimalik sooritada erinevaid tegevusi seoses failidega: failide avamine ja salvestamine.

Avades Thonnys faili oma arvutist tekib faili avamise kirje (joonis 12).

```
{
    "text_widget_id": 33603632,
    "text_widget_class": "Text",
    "filename": "C:/Users/Raili/PROG kodutööd/paaritute arvude summa.py",
    "sequence": "Open",
    "editor_id": 15163856,
    "time": "2016-03-12T19:56:20.707252",
    "editor_class": "Editor"
}
```


Joonis 12. Thonny logifaili kirje faili avamise kohta.

Faili avamise kirje koosneb 7 võti-väärtus paarist.

- *"sequence": "Open"* näitab, et tegemist on faili avamisega.
- *"text_widget_class": "Text"* näitab, et fail avatakse redaktorisse.
- *"time": <aeg>* näitab, millal fail avati.
- *"filename": "C:/Users/Raili/PROG kodutööd/paaritute arvude summa.py*" näitab, milline fail avati.

Thonnys on võimalik faile salvestada kahel moel. Kui faili salvestatakse esimest korda või uue failina, siis kasutatakse „Salvesta nimega“ funktsionaalsust ning logifaili tekib „Salvesta nimega“ kirje (joonis 13).

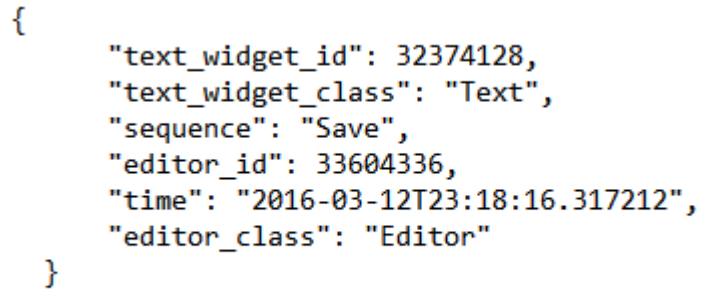

```
{
     "text_widget_id": 32374128,
     "text_widget_class": "Text",
     "filename": "C:/Users/Raili/PROG kodutööd/arvestuse proov.py",
     "sequence": "SaveAs",
     "editor_id": 33604336,
     "time": "2016-03-12T20:05:45.749269",
     "editor_class": "Editor"
 }
```


Joonis 13. Thonny logifaili kirje "Salvesta nimega" funktsionaalsuse kohta.

„Salvesta nimega“ kirje koosneb 7 võti-väärtus paarist.

- *"sequence": "SaveAs"* näitab, et tegemist on „Salvesta nimega“ funktsionaalsusega.
- *"text_widget_class": "Text"* näitab, et fail salvestati kasutades redaktoris olevat programmikoodi.
- *"time": <aeg>* näitab, millal fail avati.
- *"filename": "C:/Users/Raili/PROG kodutööd/paaritute arvude summa.py*" näitab, milline failinimi antud programmikoodile anti.

Samuti on võimalik salvestada juba olemasolevat faili. Salvestamise käigus tekib salvestamise kirje (joonis 14).



Joonis 14. Thonny logifaili kirje faili salvestamise kohta.

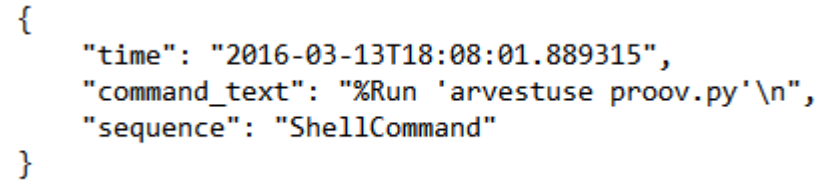
Salvestamise kirje koosneb 6 võti-väärtus paarist.

- *"sequence": "Save"* näitab, et tegemist on „Salvesta nimega“ funktsionaalsusega.
- *"text_widget_class": "Text"* näitab, et fail salvestati kasutades redaktoris olevat programmikoodi.
- *"time“: <aeg>* näitab, millal fail avati.

Thonnys on võimalik lisaks Pythoni koodifailidele, mis on lõpuga .py, avada ka muid faile. Siinjuures tuleb arvestada, et nende puhul saab kasutada Thonnyt kui tekstiredaktorit ehk neid on võimalik muuta ja salvestada, kuid neid pole võimalik käivitada. Sellegipoolest on nendega seotud kirjed kajastatud logifailides.

4.1.4 Programmi käivitamisega seotud kirjed

Programmikoodi käivitamisel Thonnys tekivad erinevad kirjed, mis on seotud programmi käivitamisega. Sellised kirjed on käivitamise käsk, veateated ja käsureale tulemuste kuvamine.
Programmikoodi käivitamisel tekivad mitmed kirjed, mille abil kuvatakse käivitamist käsureal. Käivitamise korral tekib logifaili „TextInsert“ kirje (joonis 8), kuid erinevalt joonisel olevale kirjele, on selle võti-väärtus paari *"text_widget_class": "Text"* väärtus *„Shell“*.
Käsureale sisestatav tekst on kirjes kujul *„%Run <failinimi>“*. Käivitamisel on kõige olulisem kirje käivitamise käsu kirje (joonis 15).



Joonis 15. Thonny logifaili kirje programmi käivitamise kohta.

Käivitamise käsukirje koosneb 3 võti-väärtus paarist.

- *"sequence": "ShellCommand"* näitab, et tegemist on käsurea käsuga.
- *"command_text": "%Run '8.1 arvestus.py'\n"* näitab, et käivitati fail nimega *'8.1 arvestus.py'*.
- *"time": <aeg>* näitab, millal fail käivitati.

4.1.5 Käskudega seotud kirjed

Thonnys tekivad erinevaid käske kasutates käskudega seotud kirjed (joonis 16).

```
{
    "command_id": "open_user_dir",
    "time": "2016-03-13T21:51:35.195981",
    "sequence": "Command",
    "denied": "False"
}
```

Joonis 16. Thonny logifaili kirje käsu kasutamise kohta.

Käsukirje koosnevad 4 võti-väärtus paarist.

- *"sequence": "Command"* näitab, millise käsuga on tegemist. Kui väärtus on *„ShellCommand“,* siis on tegemist käsurea käsuga. Kui väärtus on *„Command“* siis on tegemist tavalise käsuga, mis on seotud mõne teise Thonny elemendiga, kui käsurida.
- *"command_id": "open_user_dir"* näitab, millist käsku kasutati.
- *"time": <aeg>* näitab, millal käsku kasutati.
- *"denied": „False“* näitab, kas antud käsk lükati tagasi või ei. Kui antud väärtus on *„False“,* siis antud käsu sooritamine õnnestus. Kui antud väärtus on *„True“,* siis käsk lükati tagasi.

4.2 Logifaili kokkuvõte

Selleks, et näha täieliku logifaili, mida Thonny ühe kasutuskorra ajal genereerib, toome näite enim kasutatavast esimesest Pythoni programmist, milleks on teksti „Tere maailm!“ kuvamine. Antud programmi loomise ja ühekordse käivitamise käigus tekitati kokku 37 logikirjet, mis tähendab 297-realist logifaili. Logifail on pikk vaatamata sellele, et antud programmikood sisaldab vaid ühte rida, milles on 21 sümbolit. Antud logifail on lisatud ka antud lõputöö lisadesse (Lisa I). Antud näide näitab, et logifailid on pikad ning nende manuaalne läbivaatamine poleks mõislik. Selleks, et logifailidest andmeid kätte saada, on vaja programmi, mis läbiks

logifaile ja genereeriks iga logifaili kohta kokkuvõtte, mida oleks võimalik edaspidisel läbivaatusel analüüsida.

Antud lõputöö käigus loodi Pythoni programm (Lisa IV), mis võttis sisendiks kokkupakitud Thonny logifailid ning genereeris neid läbides CSV faili (Lisa III), milles on info iga kasutussessiooni kohta. Tabelis on kajastatud osad näitajad, mida on võimalik logifailidest kätte saada ja, mis võiksid olla kasutaja tegevuste analüüsimisel kasulikud. Antud informatsioon lisatakse tabelisse ainult juhul, kui läbitav logifail vastab etteantud tingimustele. Kõiki logifaile pole mõtet läbida, kuna logifailide saatja võib esitada palju logifaile, millest osad võivad olla vanad ning ei ole antud uurimise puhul olulised.

Genereeritud CSV faili on kerge muuta kergesti loetavaks tabeliks, milles olevate andmete põhjal on võimalik välja arvutada erinevaid näitajaid ning luua graafikuid.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Õpilase kood	Kirjete arv logifailis	Sessiooni algusaeg	Sessiooni lõpuaeg	Sisestamiste arv redaktoris	Kustutamiste arv redaktoris	Kopeermiste arv	Kleepimiste arv	Faili salvestamiste arv	Programmi käivitamiste arv	Veateadete kuvamise arv	Failide avamiste arv	Kasutatud failide nimed	Salvestamiste ja käivitamiste arvu vahe	Thonny sessiooni pikkus tundides
2	1	1830	2.03.2016 10:34	4.03.2016 21:54	715	202	2	3	20	18	2	3	['tkinterproov7.2ahendusekx.p	2	59,3
3	1	1085	5.03.2016 10:55	5.03.2016 11:44	14	189	2	2	11	7	1	2	['tkinter_kasutajaliides.py', 'ki	4	0,8
4	2	2090	3.03.2016 7:54	3.03.2016 14:51	4	362	6	12	21	20	1	0	['u0308Semakutagastamine.p	1	7,0
5	3	726	29.02.2016 11:52	29.02.2016 12:24	147	63	8	21	16	15	0	2	['read1.py', 'u00ecLESANNE8.2	1	0,5
6	3	708	29.02.2016 12:37	29.02.2016 14:43	121	48	9	8	19	18	1	1	['u00ecLESANNE8.2a.py', 'u00e	1	2,1
7	4	33	17.03.2016 17:36	17.03.2016 17:37	13	18	0	0	1	1	0	1	['ipp.py']	0	0,0
8	4	32	17.03.2016 17:39	17.03.2016 17:40	0	1	1	1	1	1	0	1	['ipp.py', 'Moldovalipp.py']	0	0,0
9	5	3630	4.03.2016 20:10	5.03.2016 0:07	873	283	4	10	52	52	1	2	['kontroll8.1', 'hinnad.txt', 'pro	0	4,0
10	5	237	5.03.2016 0:18	5.03.2016 0:34	0	20	0	0	4	4	0	2	['hinnad.txt', 'kontroll8.1']	0	-1	0,1
11	5	4300	5.03.2016 19:45	6.03.2016 0:29	733	453	7	14	67	68	2	6	['kontroll8.1', 'kontroll8.2', 'hir	0	-1	4,7
12	6	199	2.03.2016 13:41	2.03.2016 14:00	107	18	0	0	3	3	0	1	['yleisanne_4.3_konto.py', 'u00	0	0,3
13	6	1721	2.03.2016 14:01	2.03.2016 15:58	649	109	3	4	29	30	26	1	['yl81.py']	0	-1	2,0
14	6	1702	2.03.2016 19:06	3.03.2016 20:08	0	134	4	7	44	42	37	1	['yl81.py', 'u00e4da(yl81_A.py	2	1,0
15	6	116	2.03.2016 19:53	2.03.2016 20:08	2	11	1	0	3	4	0	1	['yl81.py']	0	-1	0,2
16	6	45	3.03.2016 11:36	3.03.2016 11:36	0	1	0	0	0	1	1	1	['yl81_A.py']	0	-1	0,0
17	6	1343	3.03.2016 13:20	3.03.2016 14:26	380	166	1	2	20	21	8	1	['yl81_A.py']	0	-1	1,1
18	6	113	3.03.2016 14:18	3.03.2016 14:22	12	15	0	0	2	2	0	1	['yl81_A.py']	0	0	0,1
19	7	5731	29.02.2016 12:32	2.03.2016 18:51	1496	491	17	29	86	96	27	23	['7-ülkontroll__lesanne.py', 'u0	0	-10	44,3
20	7	4089	1.03.2016 11:31	1.03.2016 16:32	6	375	39	54	59	61	1	4	['8-üakontrolluleslesnneteiner.	0	-2	5,0
21	8	1292	29.02.2016 13:47	29.02.2016 14:06	518	80	4	10	42	19	1	0	['8.1']	0	23	0,3
22	9	2054	2.03.2016 10:48	2.03.2016 12:34	1175	282	2	4	27	24	0	2	['tekkulastor.py', 'lalaaa.py', 'l	0	3	1,8
23	10	1018	6.03.2016 17:04	6.03.2016 18:06	432	47	0	0	16	8	3	2	['readio.py', 'hinnad.txt', '8.1.p	0	8	1,0
24	10	57	6.03.2016 19:27	6.03.2016 19:42	7	1	0	0	1	1	1	1	['8.1.py']	0	0	0,3
25	10	1336	7.03.2016 18:48	7.03.2016 21:31	328	109	0	0	14	10	7	1	['8.1.py']	0	4	2,7
26	11	33	29.02.2016 12:03	29.02.2016 12:15	0	1	0	0	0	0	0	1	['yl81e.py']	0	0	0,2
27	12	8588	1.03.2016 8:20	2.03.2016 8:41	3017	1369	21	39	564	197	73	4	['sunnikuupaevad.txt', 'paevik.	367	24,3

Joonis 17. Kokkuvõtte tabel logifailides sisaldunud info kohta.

4.2.1 Õppija kood

Paljud õppijad ei lahenda ülesandeid ainult ühe kasutussessiooni jooksul ning seega tekib õppijal mitmeid logifaile, mille esitamiseks peab ta need failid kokku pakkima. Selleks, et eristada erinevate õppijate logifaile, on igal tabelireal ka veerg „Õppija kood“, mis kajastab ühe logifaili informatsiooni. „Õppija koodi“ järgi ongi võimalik õppijaid omavahel eristada. Antud veerg aitab logifailide informatsiooni omavahel ühendada ning selle abil on võimalik kergesti teha kokkuvõte ühe õppija tegevuste kohta mitme logifaili põhjal.

4.2.2 Kirjete arv logifailis

Antud veerg näitab, mitu kirjet on antud logifailis. Kirjete arv on oluline, et eraldada sisukaid ja peaaegu tühje logifaile. Uuritud näidisiandmete puhul tuli välja, et paljud esitatud logifailid sisaldasid kuni 30 kirjet, ning nende sisu täpsemalt uurides ei sisaldanud need analüüsi jaoks piisavalt informatsiooni. Üldiselt sisaldasid need ainult redaktori loomist ning mõne faili avamist, misjärel Thonny suleti. Antud informatsioon pole aga analüüsi jaoks piisav ning neid pole vaja kokkuvõtte failis kuvada.

4.2.3 Sessiooni alustamise ja lõpetamise aeg

Antud veerud näitavad vastavalt, kuna antud kasutussessiooni alustati ning, kuna sessioon lõpetati. Sessiooni alustamise ajaks on kasutatud logifaili pealkirja ning modifitseeritud selle formaati paremasse vormingusse, mida oleks võimalik kasutada hiljem ka erinevate arvutuste ja parema kuvamise jaoks. Sessiooni lõpetamise ajaks on kasutatud logifaili viimase kirje ajatemplit, mida on sarnaselt alustamise ajaga modifitseeritud paremasse formaati.

Alustamise ja lõpetamise kirjete abil on võimalik kindlaks teha antud sessiooni pikkust. Sessiooni pikkus peaks olema uurijale huvitavaks näitajaks, mis näitab, kui kaua õppija antud kodutöö või praktikumiga tegeles. Siin tuleb arvestada, et pole võimalik kindlaks teha, kas ühe sessiooni ajal tegeleti ainult ühe kodutöö või praktikumi ülesannetega. Seda eelkõige, kuna osad sessioonid on väga pikad. Pikad sessioonid tekivad seetõttu, et sageli inimesed ei sulge programme ja need jäävad arvutisse tööle väga pikaks ajaks. Thonny logifail tehakse aga Thonny käivitamisel ning lõpetatakse sulgemisel, seega võib sellisel juhul alustamise ja sulgemise vahele jääda pikk periood, mille sees võivad olla mitme praktikumi teemad. Antud olukorras tuleks selliseid logifaile süvitsi uurida.

4.2.4 Teksti sisestamiste ja kustutamiste arv redaktoris

Veerg „Teksti sisestamiste arv redaktoris“ näitab, mitu sümbolit on üksikult sessiooni ajal redaktorisse sisestatud. Selle jaoks tuleb logifailis loendada sobivad kirjed, mis tähendavad üksiku sümboli sisestamist. Märksõna *„TextInsert“* viitab üldiselt Thonnysse sümbolite sisestamisele. Lisaks redaktorile on Thonnys võimalik teksti sisestada ka käsureale. Seega tuleks eristada need kirjed, mis sobivad (joonis 8).

Selleks, et teksti sisestamist teistest kirjetest eristada, on olulised read *"sequence": "TextInsert"*, mis näitab, et tegemist on teksti sisestamisega. Lisaks sellele on oluline, et antud kirje ei sisaldaks rida *"text_widget_context": "shell"*, mille abil saame eraldada redaktoris tehtud sisestamised käsureal tehtud sisestamistest. Samuti on üksiku sümboli sisestamise tuvastamisel oluline ka võtme *"text":* väärtuse pikkus. Selle väärtus peab olema ühe sümboli pikkune.

Analoogselt näitab veerg „Teksti kustutamiste arv redaktoris“, mitu sümbolit on kasutussessiooni ajal redaktorist kustutatud. Selle jaoks tuleb loendada kirjed, mis tähendavad sümboli kustutamist redaktoris. Märksõna *„TextDelete“* viitab üldiselt sümbolite kustutamisele Thonnys. Lisaks redaktorile saab sümboleid kustutada ka käsurealt. Seega tuleks eristada need kirjed, mis sobivad (joonis 10).

Selleks, et neid kirjeid leida, peab kirje sisaldama rida *"sequence": "TextDelete"*, mis viitab sellele, et tegemist on teksti kustutamisega. Samuti on oluline on, et kirje ei sisaldaks rida *"text_widget_context": "shell"*, mille esinemise korral toimuks kustutamine käsureal.

4.2.5 Teksti kleepimiste ja kopeerimiste arv

Teksti kleepimiste arv näitab, mitu korda on sessiooni jooksul redaktorisse teksti kleebitud. Kleepimise käigus võib õppija kleepida võõra programmikoodi või mingi muu teksti ning selle enda koodi lisada. Seega võiks antud veerg olla uurijale huvitav just plagiaadi küsimuse poolest. Selleks on võimalik kleepimise kirjetest kätte saada ka täpne tekst, mis kleebiti ning uurida seda täpsemalt. Teksti kleepides tekib kolm kirjet. Esimene neist on teksti sisestamise kirje (joonis 8), teine kleepimise kirje (joonis 9) ja kolmas käsu tüüpi kirje (joonis 16). Kuna on teada, et iga kleepimise korral tekib kolm kirjet, siis kleepimise kirjete loendamiseks kasutame ainult ühte neist. Selleks kirjeks on kleepimise kirje. Selle kasutamine on kõige mõistlikum, kuna tegemist on unikaalse kirjega. Teised kirjed on sellised, millel on ka teisi kasutamiskohti ning sellega väldime eksimusi.

Selleks, et neid kirjeid leida, peab kirje sisaldama rida *"sequence": "<<Paste>>"*, mis viitab kleepimisele.

Kopeerimiste arv näitab, mitu korda on sessiooni jooksul kasutatud kopeerimist. Kopeerimise käigus tekib alati kaks kirjet, kopeerimise kirje (joonis 11) ja käsu tüüpi kirje (joonis 16). Selleks, et loendada kopeerimiste arvu, kasutame ainult kopeerimise kirjet.

Selleks, et neid kirjeid leida, peab kirje sisaldama rida *"sequence": "<<Copy>>"*, mis viitab kopeerimisele.

4.2.6 Faili salvestamiste arv

Failide salvestamise veerg näitab, mitu korda on kasutussessiooni jooksul koodifaile salvestatud. Antud näitaja abil saab informatsiooni kasutaja harjumustest failide salvestamise puhul.

Salvestamiste arvu sisse on loendatud nii „Salvesta nimega“ salvestamisi kui ka tavalisi salvestamisi. „Salvesta nimega“ kirje on logifailist leitav märksõna *„SaveAs“* järgi. Selline salvestamine toimub juhul, kui antud failil pole veel nime või fail salvestatakse uue nimega. „SaveAs“ toimingu puhul lisatakse logifaili kaks kirjet, üks neist on „Salvesta nimega“ kirje (joonis 13) ja teine käsu tüüpi kirje (joonis 16).

Tavalisi salvestamisi saab logifailist leida märksõna *„Save“* kaudu. Tavaline salvestamine kutsutakse välja kui kasutaja vajutab „Salvesta“ nuppu. Samas ka juhul, kui kasutaja soovib programmi käivitada, kuid on peale eelmist käivitamist failis muudatusi teinud. Tavalise salvestamise puhul tekib ainult üks salvestamise kirje (joonis 14).

Kuna „salvesta nimega“ puhul tekivad kaks kirjet, siis loendamiseks kasutame ainult „Salvesta nimega“ kirjet (joonis 13). Tavalise salvestamise puhul loendame salvesta kirjeid (joonis 14).
„Salvesta nimega“ kirjete leidmisel on oluline, et see sisaldaks kirjes rida *"sequence": "SaveAs"*, mis viitab salvestamisele nimega. Samuti on sellest kirjest kättesaadav ka salvestatava faili nimi.
Tavalise salvestamise kirje leidmisel on oluline, et see sisaldaks rida *"sequence": "Save"*, mis viitab salvestamisele. Antud kirjest pole võimalik salvestatud faili nime kätte saada.

4.2.7 Programmi käivitamiste arv

Failide käivitamiste arv näitab, mitu korda on kasutussessiooni ajal programme käivitatud. Faile saab kasutaja Thonnys käivitada, kas käivitamise nupust või kiirklahviga F5. Samas pole kirjetest võimalik eristada, kas käivitatud on nupust või kiirklahviga.
Programmi käivitamise korral lisatakse logifaili mitu kirjet. Esimene neist on teksti sisestamise kirje käsureale, millega kuvatakse käsureale, et käivitatakse antud programmi. Teine on faili salvestamise kirje (joonis 14), mis salvestab faili koos muudatustega. Antud kiri tekib vaid juhul, kui kahe käivitamise vahel on programmikoodis toimunud muudatused. Kolmas on käivitamise käsu tüüpi kirje (joonis 15), millega antakse käsk käsureale. Nii teksti sisestamise kui ka käsurea käsu kirje sisaldavad märksõna *„%Run“*, mis viitab käivitamisele. Käivitamiste loendamiseks kasutame käsurea käsu kirjeid (joonis 15).
Loendamiseks kasutavate käivitamise kirjete kindlaks tegemiseks peab kirje sisaldama märksõna *„%Run“*. Samuti peab antud kirje sisaldama ka rida *"sequence": "ShellCommand"* , mis viitab sellele, et tegemist on käivitamise käsuga.

4.2.8 Veateadete arv

Veateadete arv näitab, mitu korda kuvati kasutajale kasutussessiooni ajal veateateid. Veateadete puhul tekivad logifaili mitmed käsureale teksti sisestamise kirjed. Selleks, et veateadete arvu saada tuleks loendada kõik teksti sisestamise kirjed, mis sisaldavad sõna „Error“.

4.2.9 Failide avamiste arv

Failide avamiste arv näitab, mitu korda on kasutussessiooni jooksul Thonnys faile avatud. Faili avamisel tekib logifaili faili avamise kirje. Faili avamiste arvu saamiseks loendatakse kõik faili avamise kirjed (joonis 12).
Faili avamise kirje leidmiseks on oluline, et kirje sisaldaks rida "sequence": "Open", mis viitab faili avamisele. Antud kirjes on olemas ka avatava faili nimi ning selle täpne asukoht kasutaja arvutis.

4.2.11 Sessiooni ajal kasutatud failide nimed

Kasutussessiooni jooksul kasutatud failide nimed esinevad erinevates kirjetes. Sellised kirjed on „Salvesta nimega“ tüüpi kirjed, faili avamise kirjed ja programmi käivitamise kirjed. Kahest esimesest on võimalik kätte saada avatud ja salvestatud failide nimed märksõna „Filename“ kaudu. Programmi käivitamise kirjetes on failinimeks *„%Run“*'ile järnev .py laiendiga faili nimi.

Kuna üks faili nimi võib esineda mitmes kirjes, siis kasutatud failide nimedesse lisame iga nime vaid ühe korra. Samuti eemaldatakse failinimest selle asukoht õppija arvutis.

Antud näitaja annab uurijale aimu, milliseid faili nimesid kasutajad kasutavad.

5. Thonny logifailide kokkuvõtte kasutamise näidis

Thonny logifailide kokkuvõttes on palju informatsiooni õppijate kasutussessioonide kohta. Selleks, et antud informatsiooni kasutada on palju võimalusi ning antud bakalaureusetöö raames tuuakse ka üks näide. Näide tuuakse seoses programmi käivitamistega. Antud näites tuuakse välja, mida on võimalik logifailidest seoses programmi käivitamistega kätte saada. Kuna kokkuvõttes on ka palju muud informatsiooni, siis nende puhul täpsemat uurimist antud bakalaureusetöö raames ei toimu, kuid ka muud informatsiooni on võimalik analoogselt uurida.

5.1 Andmed

Analüüsiks kasutatavad näidisandmed on pärit „Programmeerimise aluste“ e-kursuselt. Kursuse lõputööks oli õppijatel valikus kaks ülesannet – kindel ülesanne ning omaloominguline ülesanne. Mõlema valiku puhul oli Thonnyt kasutanud õppijatel kohustuslik ka ülesande lahendamise jooksul tekkinud logifailide esitamine. Neile, kes kasutasid mingit muud keskkonda oli analoogne ülesanne, kus tuli kirjeldada oma lahendamisprotsessi raskeid ja kergeid kohti ning anda võimalikult täpne hinnang, kui kaua antud ülesande lahendamisele aega kulus [10]. Selleks, et õppijad oskaksid logifaile esitada, koostas antud lõputöö autor kirjaliku juhendi ja videoõpetuse, kuidas logifaile esitada. Kasutatud juhendid on lisatud ka antud lõputöö lisadesse (Lisa II).

Juhendis oli palutud, et õppijad esitaksid oma logifailid kokkupakituna. Antud nõue oli oluline, kuna osad õppijad kasutasid ülesande lahendamiseks rohkem kui ühte kasutussessiooni ning seega oli neil vaja esitada mitu logifaili. Logifailid tuli esitada Moodle kaudu koos oma ülesande lahendusega.

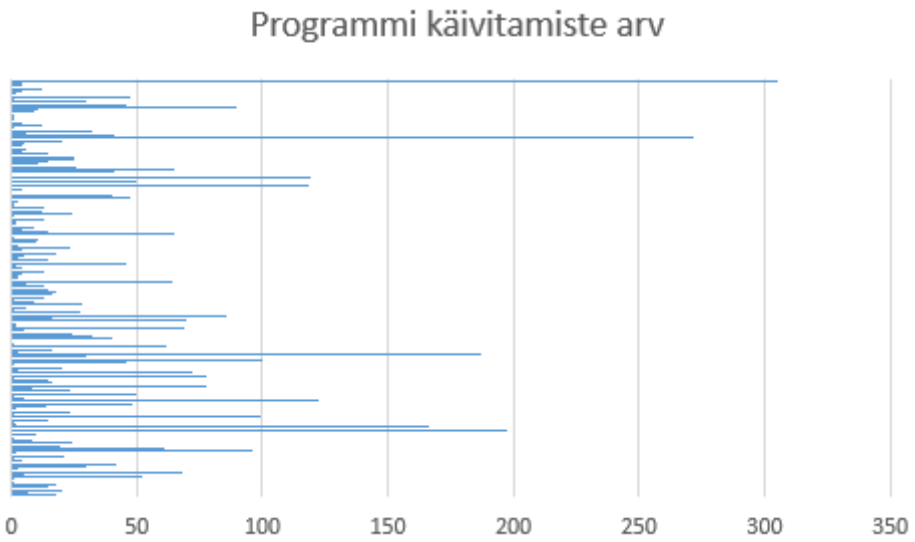
Oma logifailid esitasid 128 kursusel osalenut. Suurem osa neist esitasid oma logifailid nagu juhendis nõutud oli ehk kokkupakitult. Samuti esines ka neid, kes esitasid ainult ühe faili või esitasid kogu Thonny logide kausta. Lisaks sellele oli nendes kaustades ka muid faile, mis polnud logifailid. Seega on vajalik saadetud failidest eraldada need, mis on logifailid ja need, mis on loodud antud ülesande lahendamise perioodil ning mida tuleks eraldi edaspidi uurida.

78 õppijat esitasid oma logifailid zip-formaadis. Antud lõputöö raames uuritakse ainult zip-formaadis esitatud logifaile. Nendest esitatud failidest olid sobivad 139 logifaili, millest koostati punktis 4.2 kirjeldatud kokkuvõtted.

5.2 Thonny logifaili kokkuvõtte kasutamise näide programmi käivitamiste näitel

Thonny logifaili kokkuvõtte kasutamise kasulikkuse näitamiseks töötatakse läbi programmi käivitamiste kirjed. Vaadeldakse 160 logifaili.

5.2.1 Käivitamiste arv



Joonis 18. Käivitamiste arvu diagramm.

Käivitamiste arv oli kasutussessioonide lõikes väga erinev. Kõige väiksem käivitamiste arv oli 0 ja kõige kõrgem 305. 0 käivitamisega logifaile oli valimis 6 ning nende kasutussessioonide pikkus oli 3-18 minutit ehk tegemist oli antud valimi puhul suhteliselt lühikeste sessioonidega. 305 käivitamisega oli üks logifail, mille kasutussessiooni pikkus oli 171 tundi ja 36 minutit ehk 7 ööpäeva 3 tundi ja 36 minutit. Antud kasutussessioon oli ühtlasi ka kõige pikem kasutussessioon valimis.

Keskmine käivitamiste arv oli 28 ja mediaan 13. Seal juures keskmine kasutussessiooni pikkus oli 6 tundi ja mediaan 42 minutit. Seega oli keskmine käivitamiste arv tunnis 4,7.

5.2.2 Salvestamiste ja käivitamiste arvude vahe

Salvestamiste ja käivitamiste arvude vahe on hea moodus tuvastamaks näiteks, kas inimene käivitab oma programmi mitmeid kordi järjest programmikoodi muutmata.

Kui antud vahe on 0, siis enne igat käivitamist on programmikoodi salvestatud ning seda eelkõige, kuna salvestamine toimub juhul, kui programmikoodis on muudatusi automaatselt enne käivitamist.

Kui antud vahe on positiivne, siis on kasutaja salvestanud oma koodi ka ilma programmi käivitamata. Seega antud isikul on kombeks oma faile sagedasti salvestada.

Kui antud vahe on negatiivne, siis on kasutaja käivitanud oma koodi mitmeid kordi järjest. Seejuures pole mitme käivitamise vahel muudatusi tehtud. Selline käitumine võib olla omane programmidele, kus võivad erinevatel käivitamistel olla erinevad väljundid. Näiteks kasutab programm suvalise arvu genereerimist. Samuti võib põhjuseks olla ka kasutaja arvamus, et sama kood võib järgmine kord ehk töötada või mõni veateade, mis ehmatab kasutajat.

Kõige väiksem salvestamiste ja käivitamiste arvude vahe oli -54. See tähendab, et kasutaja üritas mitmeid kordi programmi ilma muudatusteta uuesti käivitada. Kõige suurem salvestamiste ja käivitamiste arvude vahe oli 367.

Keskmine salvestamiste ja käivitamiste arvude vahe oli 8 ja mediaan 0. Seega kasutajad pigem salvestavad oma programmikoodi ka käivitamiste vahepeal või lasevad sellel juhtuda automaatselt käivitamise käigus.

6. Kokkuvõte

Programmeerimist õpetavate e-kursuste juures on õppejõududel piiratud võimalused õppija jälgimiseks ülesannete lahendamisel. Reeglina see informatsioon, mis õppijatelt õppejõududele ülesannete kohta saabub, on juba lõpplahendus. Lõpplahenduse puhul saab kontrollida ainult selle õigsust ning vahepeal tehtud toimetuste kohta informatsioon puudub. Seega oleks õppejõududele kasulik selline vahend, mille abil oleks võimalik õppijate töökäiku ülesannete lahendamisel uurida.

Varasemalt on uuritud õpilaste töökäiku lisaks lõpptulemusele ka kasutades hetkevõtteid (*snapshots*). Nende puhul on puuduseks kätte saadava info täpsus. Hetkevõtteid tehakse iga salvestamise korral ning seega pole salvestamiste vahepealsed tegevused täpselt salvestatud. Lisaks hetkevõtetele on kasutatud ka nupuvajutuste salvestamist. Antud moodus on täpsem kui hetkevõtted, kuna kõik programmikoodi muudatused on salvestatud eraldi kirjeteks, mida on võimalik eraldi analüüsida. Samas annab see võimaluse üldiselt ainult tekstiredaktoris toimunud tegevusi analüüsida.

Programmeerimiskeskkond Thonny, mis on mõeldud eelkõige õppimiseks ja õpetamiseks, logib kõik selles toimuvad toimingud logifaili. Erinevalt nupuvajutuste salvestamisele logib Thonny ka muid sündmusi, mis toimuvad Thonny programmi aknas ning võiksid olla uurimise juures huvitavad.

Antud bakalaureusetöö raames uuriti, milliseid kirjeid sisaldab Thonny logifaili. Kuna Thonny logifailid on mahult suured siis nende käsitsi läbi vaatamine poleks mõistlik. Logifailidest info saamiseks loodi programm, mis koostab iga logifaili kohta kokkuvõtte. Kokkuvõtte koostamiseks tuli valida kirjete seast välja sellised kirjed, mis pakuksid õppejõududele õppija tööd uurides huvi. Sellised kirjed on näiteks programmikoodi muutmisega seotud kirjed (sisestamised, kustutamised ja kleepimised). Samuti ka failidega seotud kirjed (salvestamised, avamised, käivitamised). Kokkuvõte sisaldab kirjete arvu logifailis ning samuti ka failide nimesid, mis antud kasutussessiooni ajal kasutuses olid.

Antud programm aitab õppejõududel koguda informatsiooni õpilaste töökäigu kohta, mille abil on võimalik uurida õpilasi kas üksikult või kõiki õpilasi korraga, koostades nende põhjal statistikat.

Antud lõputöö viimases osas toodi näide programmikoodi käivitamiste kohta. Näide näitab, millist informatsiooni on võimalik vähese vaevaga programmi poolt koostatavast kokkuvõttest kätte saada.

Kasutatud kirjandus

[1] R.Hosseini, A.Vihavainen, P.Brusilovsky Exploring Problem Solving Paths in a Java Programming Course http://d-scholarship.pitt.edu/21832/1/Paper_Pages_from_PPIGproceedings.pdf

[2] E-kursustee „Programmeerimise alused“ ja „Programmeerimisest maalähedaselt“ võrdlus https://courses.cs.ut.ee/2016/progmaa/spring/Main/Maalahealusedvordlus (6.05.2016)

[3] V. Vaherpuu, „Murelahendajate loomise keskkond“ (2016)

[4] Thonny koduleht http://thonny.cs.ut.ee/ (6.05.2016)

[5] Aivar Annamaa, „Introducing Thonny, a Python IDE for Learning Programming“, November 2015 http://dl.acm.org.ezproxy.utlib.ee/citation.cfm?id=2828969&CFID=772017839&CFTOKEN=17231624

[6] A.Annamaa, „Thonny, a Python IDE for Learning Programming“, Juuni 2015 http://dl.acm.org.ezproxy.utlib.ee/citation.cfm?id=2754849&CFID=772017839&CFTOKEN=17231624

[7] A.Vihavainen, M.Luukkainen, P.Ihantola, „Analysis of Source Code Snapshot Granularity Levels“ http://dl.acm.org/citation.cfm?id=2656473

[8] A.Vihavainen, „Predicting Students’ Performance in an Introductory Programming Course using Data from Students’ own Programming Process“ http://ieeexplore.ieee.org.ezproxy.utlib.ee/stamp/stamp.jsp?tp=&arnumber=6602003

[9] A.Aziz, S.Mitchell, „An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET“ https://msdn.microsoft.com/en-us/library/bb299886.aspx

[10] Programmeerimise alused kursuse veebileht. https://courses.cs.ut.ee/2015/eprogalused/Main/Arvestusylesanne (6.05.2016)

Lisad

I Programmi „Tere maailm!“ loomise ja selle ühekordse käivitamise kajastamine Thonny logifailis


```
[
  {
    "editor_id": 60685424,
    "text_widget_class": "Text",
    "editor_class": "Editor",
    "text_widget_id": 79304720,
    "sequence": "EditorTextCreated",
    "time": "2016-04-12T21:25:21.136943"
  },
  {
    "editor_id": 60685424,
    "text_widget_class": "Text",
    "editor_class": "Editor",
    "text_widget_id": 79304720,
    "sequence": "NewFile",
    "time": "2016-04-12T21:25:21.136943"
  },
  {
    "text_widget_class": "Text",
    "text_widget_context": "shell",
    "text_widget_id": 60635600,
    "index": "1.0",
    "text": ">>> ",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:21.188978",
    "tags": "('toplevel', 'prompt')"
  },
  {
    "widget_id": 18744272,
    "sequence": "<FocusIn>",
    "widget_class": "Workbench",
    "time": "2016-04-12T21:25:21.189978"
  },
  {
    "text_widget_class": "Text",
    "index": "1.0",
    "text_widget_id": 79304720,
    "text": "p",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:22.583635",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.1",
    "text_widget_id": 79304720,
    "text": "r",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:22.887216",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.2",
    "text_widget_id": 79304720,
    "text": "i",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:23.045624",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.3",
    "text_widget_id": 79304720,
    "text": "n",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:23.232546",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.4",
    "text_widget_id": 79304720,
    "text": "t",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:23.438103",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.5",
    "text_widget_id": 79304720,
    "text": "(",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:23.902765",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.6",
    "text_widget_id": 79304720,
    "text": "\"",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:24.934288",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.7",
    "text_widget_id": 79304720,
    "text": "T",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:25.665249",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.8",
    "text_widget_id": 79304720,
    "text": "e",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:26.023369",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.9",
    "text_widget_id": 79304720,
    "text": "r",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:26.231208",
    "tags": "None"
  },
  {
    "text_widget_class": "Text",
    "index": "1.10",
    "text_widget_id": 79304720,
    "text": "e",
    "sequence": "TextInsert",
    "time": "2016-04-12T21:25:26.366162",
    "tags": "None"
  },
```

```
{
   "text_widget_class": "Text",
   "index": "1.11",
   "text_widget_id": 79304720,
   "text": " ",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:27.028874",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.12",
   "text_widget_id": 79304720,
   "text": "m",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:27.304125",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.13",
   "text_widget_id": 79304720,
   "text": "a",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:27.427345",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.14",
   "text_widget_id": 79304720,
   "text": "a",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:27.586660",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.15",
   "text_widget_id": 79304720,
   "text": "i",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:27.753489",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.16",
   "text_widget_id": 79304720,
   "text": "l",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:28.036243",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.17",
   "text_widget_id": 79304720,
   "text": "m",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:28.251632",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.18",
   "text_widget_id": 79304720,
   "text": "!",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:29.341685",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.19",
   "text_widget_id": 79304720,
   "text": "\"",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:29.712546",
   "tags": "None"
},
{
   "text_widget_class": "Text",
   "index": "1.20",
   "text_widget_id": 79304720,
   "text": ")",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:30.591344",
   "tags": "None"
},
{
   "widget_id": 60636528,
   "sequence": "<Button-1>",
   "widget_class": "Button",
   "time": "2016-04-12T21:25:34.414023"
},
{
   "widget_id": 18744272,
   "sequence": "<FocusOut>",
   "widget_class": "Workbench",
   "time": "2016-04-12T21:25:34.552666"
},
{
   "widget_id": 18744272,
   "sequence": "<FocusIn>",
   "widget_class": "Workbench",
   "time": "2016-04-12T21:25:42.553225"
},
{
   "editor_id": 60685424,
   "text_widget_class": "Text",
   "editor_class": "Editor",
   "text_widget_id": 79304720,
   "sequence": "SaveAs",
   "time": "2016-04-12T21:25:42.603296",
   "filename": "C:/Users/Keili/teremaailm.py"
},
{
   "text_widget_class": "Text",
   "text_widget_context": "shell",
   "index1": "1.4",
   "text_widget_id": 60635600,
   "index2": "2.0",
   "sequence": "TextDelete",
   "time": "2016-04-12T21:25:42.604298"
},
{
   "text": "%Run teremaailm.py\n",
   "text_widget_context": "shell",
   "text_widget_class": "Text",
   "text_widget_id": 60635600,
   "index": "1.4",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:42.605299",
   "tags": "('automagic', 'toplevel', 'command')"
},
{
   "command_text": "%Run teremaailm.py\n",
   "sequence": "ShellCommand",
   "time": "2016-04-12T21:25:42.773916"
},
{
   "text_widget_class": "Text",
   "index": "2.0",
   "text_widget_id": 60635600,
   "text_widget_context": "shell",
   "text": "T",
   "sequence": "TextInsert",
   "time": "2016-04-12T21:25:42.825649",
   "tags": "('io', 'stdout', 'vertically_spaced')"
```

```
    },
    {
        "text_widget_class": "Text",
        "index": "2.1",
        "text_widget_id": 60635600,
        "text_widget_context": "shell",
        "text": "ere maailm!\n",
        "sequence": "TextInsert",
        "time": "2016-04-12T21:25:42.825649",
        "tags": "('io', 'stdout')"
    },
    {
        "text_widget_class": "Text",
        "index": "3.0",
        "text": ">>> ",
        "text_widget_id": 60635600,
        "text_widget_context": "shell",
        "sequence": "TextInsert",
        "time": "2016-04-12T21:25:42.836657",
        "tags": "('toplevel', 'prompt', 'vertically_spaced')"
    },
    {
        "widget_id": 18744272,
        "sequence": "<FocusOut>",
        "widget_class": "Workbench",
        "time": "2016-04-12T21:25:53.061069"
    },
    {
        "widget_id": 18744272,
        "sequence": "<FocusIn>",
        "widget_class": "Workbench",
        "time": "2016-04-12T21:27:19.410179"
    }
]
```

II Thonny logifailide esitamise juhendid

JUHEND 1

(Thonnyt kasutamata user_logs'ini)

1. Avage Minu Arvuti/My Computer.
2. Valige sealt kaust, kuhu on installeeritud Windows.
3. Kettalt valige Users kaust, kust leidke kasutaja kaust, kes on kasutanud Thonnyt.
4. Kasutaja kaustas peaks asuma .thonny kaust.
5. .thonny kaust sisaldab kausta User_logs, kuhu on salvestatud Thonny logid..
6. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
7. Tõstke need logid ühte kausta ning pakkige kokku kas .rar või .zip failiks.
8. Esitage kokkupakitud fail Moodle' kaudu.

JUHEND 2

(Thonnyst user_logs'ini)

1. Avage Thonny.
2. Valige menüüst Tools.
3. Tools'i alt leiate Open Thonny User folder, mis avab kasutaja Thonny kausta.
4. Avage sellest kaustast user_logs.
5. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
6. Tõstke need ühte kausta ning pakkige kokku kas .rar või .zip failiks
7. Esitage kokkupakitud fail Moodle'i kaudu.

Videojuhend

https://www.youtube.com/watch?v=UkBo8F9prIc&feature=youtu.be

III Logifailide kokkuvõtte tabel

Lõputöö käigus valminud kokkuvõtte tabel koos näite jaoks vajalike muudatustega on lisana kättesaadav failina : *NäidisLogifailiKokkuvõte.xlsx* .

IV Logifailide kokkuvõtet koostav programm

Lõputöö käigus valminud kokkuvõtet koostav programm on lisana kättesaadav järgmise failina: *LogifailideKokkuvõtteKoostaja.py*. Samuti on sellele lisatud ka *LogifailideKokkuvõtte-KoostajaJuhend.txt* fail, kus asub kasutusjuhend.

V Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Keili Pedel**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **E-kursuse "Programmeerimise alused" logifailide analüüs**,
(*lõputöö pealkiri*)

mille juhendaja on Eno Tõnisson,
(*juhendaja nimi*)

1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **12.05.2016**