Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TONI JANTUNEN

Ohjelmointi Taloustieteen Tutkimuksessa

SÄHKÖ- JA AUTOMAATIOTEKNIIKAN
TUTKINTO-OHJELMA
2024

TIIVISTELMÄ

Jantunen, Toni: Ohjelmointi Taloustieteen Tutkimuksessa
Opinnäytetyö, AMK
Sähkö- ja automaatiotekniikka
Toukokuu 2024
Sivumäärä: 30

Päädyin professori Katri Siebergin projektiin Tampereen Yliopistolle, koska he tekivät taloustieteellistä tutkimusta, mutta heillä ei ollut ohjelmointitaitoista henkilöä heidän ryhmässään. Taloustieteellisessä tutkimuksessa testattiin teoriaa tosielämässä käyttämällä esimerkiksi pelikokeita. Pelikokeella (experimental economics) tarkoitetaan sellaista tutkimusta, jossa vapaaehtoiset ihmiset saivat tulla osallistumaan pelikokeeseen, joka perustuu teoreettiseen malliin, jonka toimivuutta halutaan testata tosielämässä. Pelikokeeseen osallistuneet pystyivät ansaitsemaan rahaa. Projektin toteutus haluttiin tehdä oTree:lla (Avoimen lähdekoodin alusta web pohjaisille vuorovaikutusta vaativille pelikokeille). oTree oli vielä niin uusi, että tutkimusalalla ei ollut osaamista sen ohjelmoimisessa. Minun työni projektissa oli vastata tietotekniikkaan ja projektin tekniseen puoleen liittyvistä asioista. Työnkuvaani kuului pelikokeiden koodaaminen oTreella (Python, html, css, javascript), palvelimen hallinnointi ja muut it-asioihin liittyvät tehtävät, tutkimusohjeiden kääntäminen, tutkimuksen suunnitteluun osallistuminen, suomenkielisten pelikokeiden vetäminen (ja osallistuminen englanninkielisiin kokeisiin), sekä osallistujien rekrytointi. Työn lopputuloksena oli toimiva pelikoe, jota käyttämällä suoritettiin pelikokeita vapaaehtoisille koehenkilöille. Näistä suoritetuista pelikokeista kerättiin dataa, jolla tehtiin tutkimustyötä. Tehdystä tutkimuksesta tehtiin julkaisu (Karunadasa, M., Sieberg, K. K. & Jantunen, T. T. K., 2023). Ohessa linkki julkaistuun tutkimukseen: https://doi.org/10.3390/g14030046

Avainsanat: oTree, Pelikokeet, Taloustieteen tutkimus

Abstract

Jantunen, Toni: Programming In Economics Research
Bachelor’s thesis
Electrical and Automation Engineering
May 2024
Number of pages: 30

I was hired to work in Professor Katri Sieberg's project at Tampere University, because they were doing economic research, but they didn't have anyone with programming skills in their group. The project uses economic research and theory was tested in real life using, for example, experiments. An experiment (experimental economics) refers to a type of research where volunteers participate in a game which is based on a theoretical model, the results of which are tested in real life. Those who take part in the game are able to earn money. The implementation of the project was to be done with oTree (Open source code platform for web-based game experiments requiring interaction). oTree was still so new that few people in economics research knew how to program it. My job in the project was to respond to issues related to information technology and the technical side of the project. My job description included coding experiments with oTree (Python, html, css, javascript), server administration and other IT-related tasks, translating research instructions, participating in research planning, running Finnish-language experiments (and participating in English-language experiments), and recruiting participants. The final project of my work was a functional game, which was used to conduct experiments with voluntary test subjects. From these completed games, data was collected, which was used for research. The research was published as a journal article (Karunadasa, M., Sieberg, K. K. & Jantunen, T. T. K., 2023). Attached is a link to the published article: https://doi.org/10.3390/g14030046

Keywords: oTree, experimental economics, economic research.

ALKUSANAT

Projekti saatiin onnistuneesti päätökseen, sekä saimme tehtyä tutkimustyöstämme julkaisun. Haluan kiittää Katri Siebergiä, sekä Manela Karunadasaa mielenkiintoisesta projektista, sekä mukavasta ja ammattitaitoisesta työryhmästä. Haluan myös kiittää Ilmari Hentusta, sekä Petri Nuutista opinnäytetyön aikana saadusta kannustuksesta.

SISÄLLYS

1 ENSIMMÄINEN LUKU/ JOHDANTO

Valitsin kyseisen aiheen opinnäytetyökseni koska se todistaa kuinka tärkeää ohjelmointitaito ja it-osaaminen on sellaisilla aloilla, joissa se ei yleensä ole päältä päin nähtävissä. Kyseinen projekti oli myös hyödyllinen yhteiskunnallisesti, se tutkii tärkeää aihetta, joka vaikuttaa kaikkien kuluttajien saamaan palveluun arkipäiväisissä asioissa. Koin myös aiheen ja työtehtävät erittäin mielenkiintoisiksi ja halusin tarttua tähän mahdollisuuteen haastaakseni itseäni, vaikka tiesin sen olevan varmasti haastavaa.

Tutkimuksen tarkoitus oli tutkia asiakkaan näkökulmasta, suorittavatko palveluntarjoajat työtehtävänsä eri tavalla, jos heidän palkkaustyyppinsä on kiinteä kuukausipalkka, verrattuna provisiopalkkaan. Jotta kyseistä tutkimusta pystyttiin tekemään, tarvittiin toteutus, jolla pystyttiin keräämään dataa pelikokeisiin osallistuneiden henkilöiden valinnoista. Toteutuksen pohjana käytettiin oTree alustaa (Avoimen lähdekoodin alusta web pohjaisille vuorovaikutusta vaativille pelikokeille). Jotta kyseinen toteutus pystyttiin tekemään, tarvittiin ohjelmointitaitoinen henkilö tekemään tekninen toteutus.

Tämä opinnäytetyö käsittelee projektissa käytettyjä teknologioita, tutkimuksen teknisen toteutuksen läpi käymistä, valmistuneen teknisen toteutuksen kuvausta tosi elämän tilanteissa, sekä valmistuneen projektin lopputuloksen pohdintaa.

2 OHJELMOINTITAIDON TÄRKEYS TUTKIMUSTYÖSSÄ

2.1 Mitä on kokeellinen taloustiede

Kokeellisessa taloustieteessä (experimental economics) testataan teoriaa käytännössä. Luodaan pelejä, joissa osallistujat tekevät päätöksiä ja tästä saadulla datalla tehdään tutkimustyötä. Joskus taloustiedettä on vaikeaa testata käytännössä, koska kaikkia osia teoriasta ei voida mitata, tai arvottaa. Toinen ongelma taloustieteen teorioiden testaamisessa tosielämässä on se, että usein oikeassa elämässä on paljon muuttujia, joita ei voida teoriassa mitata. Kokeellisella taloustieteellä näitä teorioita pystytään testaamaan paremmin. Muuttujia voidaan kontrolloida, sekä määrittää kiinteiksi.

Esimerkiksi tätä opinnäytetyötä käsittelevän projektin pelikokeissa testattiin teoriaa siitä, miten eri palkkajärjestelmät vaikuttivat asiakkaalle tarjottuihin palveluihin. Esimerkiksi jos palveluntarjoajan työ on sellaista, että asiakas ei välttämättä ymmärrä saamansa palvelun laatua, ja tai tarvetta, niin tekeekö palveluntarjoaja juuri sen mitä asiakas tarvitsee, vai tekeekö hän enemmän, tai vähemmän oman hyötynsä puolesta?

Esimerkiksi jos asiakas ei tiedä autoista mitään ja vie oman autonsa autokorjaamolle, mekaanikko voi käyttää tilannetta omaksi hyödykseen tekemällä tarpeettomia korjauksia, jotta hän ansaitsisi enemmän palkkaa. Jos mekaanikko saisi kiinteän korvauksen riippumatta korjausten määrästä, vaikuttaisiko tämä hänen työntekoonsa? Tekisikö mekaanikko tällöin vähemmän töitä, kuin oikeasti tarvitsee?

Oikeassa elämässä voi olla vaikeaa tietää, mikä on ”oikea” määrä tehtyä työtä, koska asiakkaita on erilaisia. Esimerkiksi terveydenhuollossa on vaikea tietää mikä on oikea työn/tutkimuksen määrä per potilas ja on vaikea todistaa tietääkö lääkäri mikä olisi optimaalinen työmäärä per potilas. Kokeellisella taloustieteellä voidaan asettaa palveluntarjoajalle tehdyn työn optimaalinen määrä ja

optimaalisen työn määrä selitetään kaikille osallistujille faktana. Täten teoriaa on helpompi testata.

2.2 Taloustieteen pelikokeiden toteutukset ennen oTree:ta

Alkuperäisesti taloustieteen pelikokeita tehtiin kynällä ja paperilla, tai esimerkiksi pelikorteilla. Tämän kaltaisten pelikokeiden tekeminen oli hyvin hidasta, sekä työlästä. Näistä syistä johtuen, haluttiin luoda järjestelmä, joka pystyi laskemaan nopeasti osallistujien tulokset, sekä jakamaan heidät satunnaisesti ryhmiin. Yksi näistä järjestelmistä oli z-Tree (Fischbacher, 1999). z-Tree oli merkittävä edistys aikaisempiin kynällä ja paperilla tehtyihin pelikokeisiin verrattuna, mutta siinä oli silti puutteita. Yksi puutteista oli esimerkiksi se, että z-Tree:n käyttäminen vaati kaikkien osallistujien olevan fyysisesti samassa tilassa tekemässä koetta ja sovellus täytyi olla asennettuna jokaisella käytettävällä tietokoneella. Jos ei halunnut käyttää z-Tree:ta, oli mahdollista käyttää internetistä löytyviä valmiiksi tehtyjä pelejä kuten Holt:in veconlab (Holt, 2005). Edellä mainitussa tavassa huono puoli oli se, että pelejä ei pystynyt muokkaamaan.

2.3 Miksi ohjelmointitaitoa tarvitaan kokeellisessa taloustieteessä

Yleensä tutkimuksia tekevien henkilöiden päätyö ja pääpaino liittyy itse tutkimukseen ja taloustieteeseen, eikä heillä välttämättä ole osaamista ohjelmoimisessa. Tästä syntyy tarve ohjelmoijille, jotka voivat ohjelmoida teknisen toteutuksen heidän puolestaan. Yleensä toivotut lopulliset ratkaisut ovat monimutkaisia, sekä ne tutkivat jotain tiettyä uniikkia teoriaa, joten valmista ratkaisua harvoin löytyy jo valmiina. Tämän takia ohjelmointitaitoinen henkilö on hyvinkin tarpeellinen osa tutkimusta tekevässä ryhmässä.

3 TEORIAOSUUS

3.1 Taustatietoa käytetyistä teknologioista

oTree on edelleenkin vuonna 2024 uusi järjestelmä (Chen et al. 2016). Etuja oTree:n käyttämisessä on se, että pelaajat voivat olla missä vain. Vaatimukset osallistujalle ovat, että heillä täytyy olla toimiva verkkoyhteys, sekä tietokone. oTree:n heikkous on tämän opinnäytetyön kirjoituksen aikaan se, että oTree on vielä niin uusi, että esimerkkejä ja pohjia on haastava löytää (oTree, n.d.).

Visual Studio Code, lyhennetynä (VS Code) on Microsoftin kehittämä suosittu ilmainen avoimen lähdekoodin koodieditori. VS Code:n ominaisuuksia pystyy lisäämään lataamalla eri tahojen tekemiä lisäosia, joilla voidaan esimerkiksi parantaa VS Coden ehdottamia syntaksiehdotuksia eri kielissä (Microsoft, 2024). Syntaksilla tarkoitetaan sitä kielioppia, miten eri ohjelmointikielten välillä ohjelma saadaan suorittamaan erilaisia käskyjä. Käytännössä syntaksi tarkoittaa sitä, miten kyseisessä kielessä koodia kirjoitetaan. Syntaksiehdotus on koodieditorin antama ehdotus koodin oikein kirjoittamista varten. Syntaksiehdotusta voisi verrata esimerkiksi Word:in antamiin kieliopillisiin ehdotuksiin.

Python on ohjelmointikieli, joka on erittäin suosittu sen helpon luettavuuden sekä nopean kirjoitettavuuden takia. Huono puoli kielen käytössä on sen hitaus verrattuna sellaiseen kieleen, joka kääntää koodin konekieliseksi, esimerkiksi C kielet (Python Software Foundation, 2024).

Html eli Hyper Text Markup Language, on verkkosivujen luomiseen tarkoitettu kieli. Html:llä määritellään sivun asettelu ja sisältö html elementeillä (Refsnes Data, 2024). Css eli Cascading Style Sheets, on verkkosivujen tyylittelyyn tarkoitettu kieli (Refsnes Data, 2024). Html:ää ja Css:ää käytetään yhdessä verkkosivujen luomiseen ja tyylittelyyn. Näitä molempia käytettiin projektin aikana sivujen asettelun ja sisällön luomisessa.

Linux on ilmainen avoimen lähdekoodin käyttöjärjestelmä. Tämä tarkoittaa sitä, että Linux on yhteisön yhdessä luoma projekti ja käytännössä kenellä vain on pääsy lähdekoodiin ja vapaus muokata sitä haluamallaan tavalla. Projektin aikana valmistunut pelikoe hostattiin Linux pohjaisella palvelimella.

WinSCP, on tietokone sovellus, jolla pystytään helposti siirtämään tiedostoja eri tietokoneiden välillä etäyhteydellä. SFTP (Secure File Transfer Protocol), on protokolla tiedostojen siirtämiseen eri tietokoneiden välillä. Projektin aikana WinSCP:tä ja SFTP:tä käytettiin siirtämään valmis toteutus tietokoneeltani palvelimelle.

PuTTY, on tietokone sovellus, jolla pystytään muun muassa ottamaan etäyhteys toiseen tietokoneeseen käyttäen SSH yhteyttä. SSH eli (Secure Shell), on protokolla, jolla pystytään ottamaan etäyhteys toiseen tietokoneeseen käyttöjärjestelmästä riippumatta. Näitä käytettiin yhdessä projektin aikana, esimerkiksi palvelimen käynnistämiseen ja päivittämiseen.

4 KÄYTÄNNÖN OSUUS

4.1 Tutkimusongelma

Tutkimusongelmana koko projektille oli ymmärtää, miten palkkajärjestelmät vaikuttavat ihmisten työntekoon, kun asiakas ei ymmärrä saamansa palvelun laatua tai tarvetta. Taloustieteen mukaan ihmiset yrittävät tavoitella maksimaalista omaa hyötyä. Tässä tapauksessa taloustieteen mukaan palveluntarjoaja tekisi liikaa, tai liian vähän työtä riippuen kumpi olisi hänelle kaikista hyödyllisintä rahallisesti.

Testasimme teoriaa käyttämällä pelikokeita, joissa oli kaksi erilaista palkkajärjestelmää. Jokaisen pelikoesession aikana käytimme toista näistä palkkajärjestelmistä ja vertailimme näistä kerätyn datan tuloksia.

Yksi palkkajärjestelmistä oli fee for service (FFS), jossa palveluntarjoaja sai palkkansa tehtyjen tehtävien mukaan. Teorian mukaan tällaista palkkajärjestelmää käyttävä palveluntarjoaja haluaa suorittaa mahdollisimman paljon tehtäviä saadakseen maksimaalisen korvauksen, riippumatta asiakkaan tarvitsemasta työn määrästä.

Toinen palkkajärjestelmistä oli salary, eli kiinteä palkka. Palveluntarjoaja sai saman korvauksen riippumatta suoritettujen tehtävien määrästä, mutta maksoi kustannuksen jokaisesta suoritetusta tehtävästä. Teorian mukaan tällaista palkkajärjestelmää käyttävä palveluntarjoaja haluaisi suorittaa mahdollisimman vähän tehtäviä.

Halusimme myös testata, vaikuttaisiko asiakkaalla oleva vakuutus, joka korvaisi osan asiakkaan kustannuksista palveluntarjoajan päätöksiin. Halusimme myös testata, vaikuttaisiko se, että asiakkaalla ei pelin alussa ole vakuutusta ja pelin toisessa osassa asiakkaalla on vakuutus (FFS low to high), siihen että, pelin alussa on vakuutus ja toisessa osassa ei ole vakuutusta (FFS high to low).

Tämän teorian testaamiseksi tuli tarve tehdä pelikokeet, johon oikeat ihmiset, pääsivät osallistumaan ja saataisiin mahdollisimman realistista dataa tutkia tätä teoriaa. Osallistujat olivat pelin aikana palveluntarjoajan roolissa ja saivat itse päättää, kuinka paljon työtä he halusivat tehdä. Työn määrän mukaan heille maksettiin oikeaa rahaa ja mikäli he olivat ansainneet rahaa asiakkaalle, se raha meni tutkimusrahastoon.

4.2 Teknisten tavoitteiden kartoitus

Pelikokeeseen tuli olla mahdollista osallistua etänä ilman että osallistujan täytyi erikseen asentaa sovellusta omalle tietokoneelleen. Pelikokeen tuli seurata osallistujien tekemiä valintoja, sekä tallentaa niitä ylös, jotta kokeen lopussa tulokset saataisiin CSV-tiedostona ulos ohjelmasta tutkittavaksi data-

analyyttisin menetelmin. Toteutuksen tuli myös laskea osallistujien valintojen mukaan heidän ansaitsemansa rahamäärä, joka heille pelin lopussa maksettiin. Pelin lopussa tuli olla kyselylomake, johon osallistujien tuli vastata kokeen suorittamisen jälkeen. Kokeen alussa tuli olla ohjeistus pelin kulusta, joka luettaisiin myös ääneen osallistujille. Ohjeiden jälkeen, siirryttiin itse peliin, jossa osallistujien tuli suorittaa tehtäviä (real effort task) eri kierroksissa, mistä he ansaitsivat pisteitä. Real effort task on tehtävä, jossa osallistuja joutuu suorittamaan annetun tehtävän. Esimerkiksi tässä projektissa osallistujan tehtävä oli muuntaa numeroita kirjaimiksi annetusta listasta, jossa kerrottiin mitä kirjainta mikäkin numero vastaa. Real effort task:ien tarkoitus on simuloida tehtyä työtä. Asiakkaat, joita osallistujat palvelivat tehtävien aikana, olivat hypoteettisia, mutta jokaista oikein suoritettua tehtävää kohden asiakkaat saivat hyödyn. Pelin lopussa kaikkien asiakkaiden hyödyt laskettiin yhteen ja 60% siitä summasta siirrettiin rahastoon.

Pelistä tehtiin kolme eri versiota, joista pidettiin useita eri pelikokeita eri osallistujille ja tutkittiin tulosten eroja; FFS (fee-for-service) high to low, FFS low to high, sekä salary.

FFS tyyppisissä peleissä pelaajien ansiot määräytyivät suoritettujen tehtävien mukaan. Mitä enemmän tehtäviä he suorittivat, sitä enemmän rahaa he tienasivat, mutta jokainen suoritettu tehtävä maksoi heille yhden ECU:n (experimental currency unit). ECU:lla tarkoitetaan niitä pisteitä, joita osallistujat ansaitsivat pelin aikana. Pelin lopussa nämä pisteet muunnettiin euroiksi, tässä pelissä muuntosuhde oli 11 ECU = 1 €. He siis ansaitsivat suorittamiensa tehtävien määrään perustuvan ansion, josta vähennettiin 1 ECU per tehty tehtävä. Osallistujien ei myöskään tarvinnut suorittaa tehtäviä oikein ansaitakseen rahaa. Tämän tyyppisissä peleissä oli kaksi osaa.

FFS high to low pelin ensimmäisessä osassa asiakkaalla oli vakuutus, joka korvasi koko kustannuksen, joten asiakas sai täyden hyödyn suoritetuista tehtävistä. Toisessa osassa koetta asiakkaalla ei ollut vakuutusta, joten asiakas ei saanut täyttä etua suoritetuista tehtävistä.

FFS low to high peli toimi samalla periaatteella kuin FFS high to low, mutta erosi siten, että asiakkaalla ei ollut pelin ensimmäisessä osassa vakuutusta ja toisessa osassa asiakkaalla oli vakuutus. Eli käytännössä käännetty järjestys verrattuna FFS high to low peliin.

Salary tyyppisessä pelissä ei ollut kahta eri osaa, vaan asiakkaalla oli kaikkien pelin kierrosten aikana vakuutus. Osallistujan ansiot määräytyivät vakio palkan ja osallistujan kustannusten erotuksesta. Kustannus oli yksi ECU per tehty tehtävä. Mitä enemmän tehtäviä osallistuja suoritti, sitä vähemmän hän tienasi.

Kokeen aikana osallistujat olivat palveluntarjoajien rooleissa, jotka saivat satunnaisen hypoteettisen asiakkaan, jolla oli jonkinlainen ongelma, jonka he pystyivät korjaamaan. Pelissä oli kolme erilaista asiakastyyppiä. Jokaiselle asiakastyypille oli optimaalinen määrä suoritettuja tehtäviä. Osallistujat saivat valita kuinka monta tehtävää he suorittivat: he pystyivät olla tekemättä yhtään tehtävää, tehdä alle optimaalisen määrän tehtäviä, tehdä optimaalisen määrän tehtäviä, tai maksimi määrän, joka oli 10 tehtävää per kierros. Osallistujat saivat itse valita, mikä oli heidän mielestään optimaalinen määrä tehtäviä. Osallistujat ansaitsivat valinnoistaan pisteitä (ECU), jotka muunnettaisiin pelin lopussa euroiksi, 11 ECU = 1 €. Osallistujien ansiot vaihtelevat 0 ja 23 ECU:n (experimental currency unit) välillä per kierros. Ennen jokaista kierrosta osallistujien tuli nähdä tulevan kierroksen asiakastyyppi, suoritettujen tehtävien mukaan maksettava ansio, omat kustannukset per suoritettu tehtävä, ansaittu summa vähennettynä kustannuksista, sekä asiakkaan hyöty per suoritettu tehtävä. Tämä on esitetty alla (Kuva 1).

Customer Type 1

Number of tasks	Your payment	Your cost	Your profit	Customer Benefit
0	0	0	0	0
1	1	1	0	1
2	3	2	1	2
3	6	3	3	4
4	7	4	3	7
5	9	5	4	10
6	11	6	5	9
7	13	7	6	8
8	18	8	10	7
9	21	9	12	6
10	23	10	13	5

Kuva 1. Tässä näytönkaappauksessa esitettynä pelin ensimmäisen osan asiakastyypin yksi pisteet pelissä FFS high to low.

4.3 Tekninen toteutus

Projektin alussa, perehdyin useaan kertaan tehtävän vaatimuksiin ja keskustelin tiimini kanssa toteutuksesta, jotta saisin mahdollisimman vahvan ymmärryksen toivotusta lopputuloksesta. Kysyin, että onko olemassa pelikokeita, jotka voisivat olla samantyyppisiä, tai jotka toimisivat vähän samalla tavalla kuin meidän toivottu lopputuloksemme. Täten voisin etsiä valmiiksi tehtyjä projekteja ja oppia oTreen syntaksia, sekä nähdä esimerkkejä rakenteesta. Tämä olikin haastavaa, koska projekteja oli vaikea löytää. Katselin youtubesta eri videoita oTree:sta, tämä oli todella hyödyllistä. Luin oTreen virallista dokumentaatiota, jota käytin myös useasti toteutusta tehdessäni (oTree, n.d.).

4.3.1 Ohjelmointi

Käytin tekstieditorinani Visual Studio Code ohjelmaa (lyhennettynä VS Code), koska tämä oli minulle jo ennestään tuttu. Aloitin toteutuksen tekemisen lataamalla oTree:n python kirjaston käyttämällä pip install komentoa (Pip, 2024). Kun olin asentanut oTree kirjaston, loin testiprojektin ja käytin aikaa testaillen ja tutkien miten kyseinen kirjasto toimii. (oTree, n.d.)

Kysyin tutkimustiimiltäni, että voisivatko he näyttää minulle esimerkkejä jo tehdyistä toteutuksista, koska minulla ei ennen tätä projektia ollut kokemusta eikä ymmärrystä taloustieteen pelikokeista. Tiimini lähetti minulle esimerkin dictator game tyyppisestä pelikokeesta, joka oli tehty oTree alustalla. Dictator game on pelikoe, jossa osallistujat arvotaan pareiksi ja toinen osallistujista saa päättää miten heille tarkoitettu rahamäärä jaetaan (Eckel & Grossman, 1996). Se oli selkeästi erilainen peli, kuin se mitä meidän tutkimuksemme tarvitsi, mutta sitä kautta pystyin oppimaan syntaksia, sekä oTree:n rakennetta.

Tarvitsimme myös toteutukseemme real effort taskeja, joita osallistujien tuli suorittaa kokeen aikana. Löysimme otreehub:ista projektin, jota pystyimme hyödyntämään meidän toteutuksessamme (Chris @ Otree, 2021).

Nyt minulla oli alustavasti tarvittava ymmärrys ja referenssi materiaalia, joita pystyin hyödyntämään toteutuksen aloittamista varten. Aloitin toteutuksen päättämällä, että ohjelmoin ensin high to low pelin perustoiminnot, jonka jälkeen voin käyttää sitä pohjana low to high ja salary pelien tekemiseen. Aloitin suunnittelemalla page sequencen. Tällä tarkoitetaan sitä järjestystä, miten peli etenee ja missä järjestyksessä osallistujat näkevät sivut. Page sequence toimii siten, että ohjelma näyttää osallistujalle järjestyksessä määritetyt sivut. Jotkin sivut voidaan määrittää näkymään vain tietyillä kierroksilla, tai näkymään esimerkiksi vain tietyn roolin omaaville osallistujille. Määritetty page sequence ja siihen määritettyjen sivujen ehdot käydään läpi joka kierroksella. Esimerkiksi: jos pelin kierrosten lukumääräksi on asetettu 10, käydään page_sequence 10 kertaa järjestyksessä läpi. Alla olevassa koodissa (Ohjelma 1), on muuttuja page_sequence, johon voidaan määritellä näytettävien html sivujen järjestys.

```
page_sequence = [instructions1, instructions1_low, in-
structions2, instructions2_low, customer_type1, cus-
tomer_type1_low,
customer_type2, customer_type2_low, customer_type3, cus-
tomer_type3_low,
instructions_summary, instructions_summary_low, round_a,
round_a_low, round_b, round_b_low,
```

```
round_c, round_c_low, Game, ResultsForRound, Results,
questionnaire]
```

Ohjelma 1. page_seguence muuttuja, johon voidaan määritellä näytettävien html sivujen järjestys.

Jokaiselle sivulle, tuli tehdä oma html tiedosto, jossa määritettiin, mitä elementtejä sivu sisälsi. Pythonilla määritettiin mitä arvoja elementit sisälsivät. Pythonilla määriteltiin luokka, joka periytyy luokasta Page, luokan nimi tuli olla sama, mikä halutun html tiedoston nimi oli. Luokka Page kuuluu oTree kirjastoon. Kuten aiemmin mainitsin; page sequence käydään läpi joka kierroksella, alla olevassa esimerkissä (Ohjelma 2), on määritelty, että kyseinen instructions1 sivu näytetään vain ensimmäisellä kierroksella.

```
class instructions1(Page):
    @staticmethod
    def is_displayed(player: Player):
        if player.round_number == 1:
            return True
        else:
            return False
```

Ohjelma 2. Instructions1 sivun toiminnallinen määrittely.

Pelaajien oli tärkeää myös päästä liikkumaan seuraavalle sivulle page_sequencessa, painamalla sivulla olevaa näppäintä. Lisäsin jokaiselle html sivulle vastaavan näppäimen (Ohjelma 3).

```
<div>
    <button id="next" class="btn btn-dark">Next</button>
</div>
```

Ohjelma 3. Next page napin määrittely html tiedostossa.

Pelikokeen alussa pelaajille tuli selittää pelin kulku. Katri Sieberg oli kirjoittanut valmiiksi nämä kyseiset ohjeet, jotka hän antoi minulle word tiedostona. Lisäsin peliin jokaiselle sivulle omat html tiedostot ohjeiden kera, lisäsin niille ehdot __init__.py tiedostoon, jossa määrittelin niiden ohjelmallisen toiminnan. Tässä tapauksessa ohjelmallinen toiminta oli se, että ohjeiden tuli näkyä vain ensimmäisellä kierroksella ja niiden tuli näkyä ensimmäisenä sivuna, kun osallistuja

avasi pelin. Koska tässä tapauksessa oli kyse high to low pelistä, puolessa välissä peliä asiakkaan vakuutus muuttui, joten tuli silloin osallistujalle kertoa uudelleen pelin kulusta ja siihen liittyvistä muutoksista toisessa osassa peliä. Lisäsinkin toiset ohjeet peliin samalla tavalla kuin aikaisemmin, mutta nämä uudet lisätyt ohjeet tulisivat näkyviin vain, jos pelaaja oli kuudennella kierroksella (high_to_low pelissä oli yhteensä 12 kierrosta). Ohjeiden lisääminen oli yksinkertaista; sivun asettelu ja sisältö lisättiin suoraan html tiedostoon.

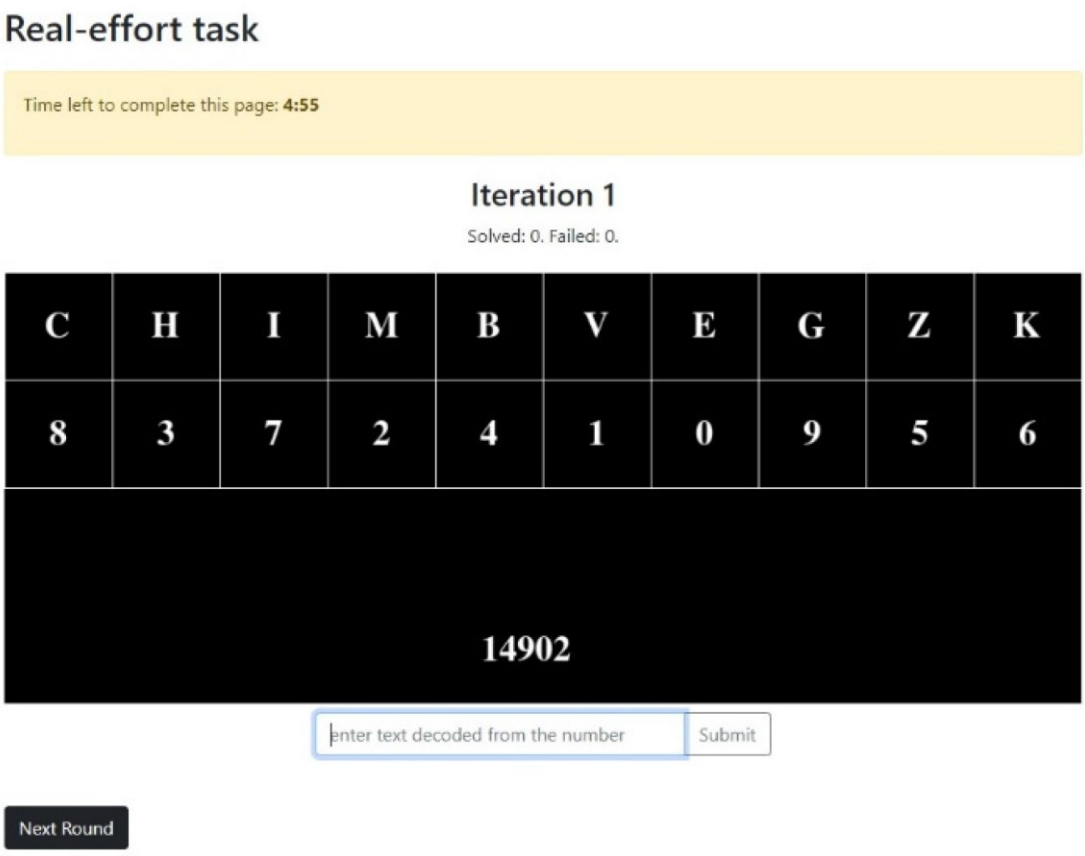
Kun pelaaja oli edennyt ohjeista eteenpäin, päätimme että pelaajaa tulisi vielä ennen varsinaisia real effort taskeja jokaisella kierroksella muistuttaa, mikä asiakastyyppi ja siihen liittyvät ansiot ja vakuutukset seuraavaksi palveltavalla asiakkaalla olisivat. Asiakastyyppejä oli yhteensä kolme, mutta koska pelin toisessa osassa asiakkaan vakuutus muuttuu, tuli näistä kolmesta asiakastyypistä olla yhteensä kuutta eri variaatiota. Lisäsin jälleen kerran jokaiselle asiakastyypille omat html tiedostonsa ja näiden ehdot __init__.py tiedostoon. Html tiedostoihin lisättiin kyseisen asiakastyypin taulu (Kuva 1), sekä esimerkkejä ansaituista pisteistä eri skenaarioissa. Alla esimerkki html tiedostoon määritellyistä ohjeista (Ohjelma 4).

```
<p>If you successfully solve one task and fail one task,
you will earn 3 ECU - 2ECU for a profit of 1 ECU,
        and 60% of 1 ECU will be donated to a fund.</p>
```

Ohjelma 4. Html tiedostoon määriteltiin osallistujan näkemät ohjeet.

Tämän jälkeen pelaaja siirtyi itse real effort taskeihin. Kuten aikaisemmin mainitsin, hyödynsimme otreehubista (Chris @ Otree, 2021), löytämäämme toteutusta real effort taskien tekemiseen. Real effort taskina toimi yksinkertainen peli, jossa pelaajalle annettiin numerosarja, joka hänen tuli muuntaa kirjaimiksi. Jokainen numero vastasi jotain kirjainta ja nämä kirjaimet ja numerot olivat satunnaiset jokaisella iteraatiolla. Maksimi määrä suoritettuja tehtäviä jokaisella kierroksella oli kymmenen. Pelaajalla tuli olla myös mahdollisuus painaa next round painiketta, jos hän oli mielestään suorittanut haluamansa määrän tehtäviä kyseisellä kierroksella. Pelaajalle annettiin myös maksimissaan 5 minuuttia aikaa suorittaa haluamansa määrä tehtäviä, jonka jälkeen peli tallensi hänen edistymisensä ja siirtyi automaattisesti seuraavalle sivulle. Pelin

tuli myös seurata oikein- ja väärin suoritettujen tehtävien lukumäärää, koska näitä tultaisiin käyttämään datana tutkimustyössä, pelaajan ansioiden laskemisessa, sekä hyväntekeväisyyteen lahjoitettavan rahamäärän laskennassa. Alla kuvankaappaus real effort taskista (Kuva 2).



Kuva 2. Kuvankaappaus pelaajan näkymästä pelin aikana tapahtuvasta real effort taskista.

Kun osallistuja oli suorittanut haluamansa määrän tehtäviä, tai aika tuli umpeen, pelin tuli laskea kyseisellä kierroksella ansaitut pisteet. Real effort task sivun jälkeen pelaajalle näytettiin kierroksen tulokset. Tulokset pitivät sisällään päättyneellä kierroksella pelaajan ansiot, kaikkien aikaisempien kierrosten yhteenlasketut ansiot, hyväntekeväisyyteen ansaitut pisteet, sekä yhteenlasketut hyväntekeväisyyspisteet kaikilta aikaisemmilta kierroksilta.

Osallistujan kierroksen aikana ansaitut pisteet laskettiin aina Game sivulla, ennen ResultsForRound sivulle siirtymistä. Otree kirjasto sisälsi metodin before_next_page(), jolla edellä mainitut pisteet pystyttiin laskemaan nimensä

mukaisesti ennen sivun vaihtumista (oTree, n.d.). Metodilla tarkoitetaan sellaista kirjoitettua koodia, jota voidaan kutsua- ja käyttää -uudelleen eri tilanteissa. Tällä tavoin ohjelmoijan ei tarvitse kirjoittaa toistuvaa koodia. Joku toinen henkilö on jo voinut kirjoittaa metodin toiminnallisuuden aikaisemmin ja ohjelmoijan ei tarvitse aina edes tietää metodin taustatoimintaa, vaan että mikä metodin suorittamisen lopputulos on. Tällä tavoin ohjelmoija säästää aikaa ja saa keskittyä olennaiseen. Tässä tapauksessa tiedettiin, että jos kyseistä metodia käytettiin jollain sivulla, kaikki mitä kyseisen metodin määriteltiin laskemaan, tapahtuisi siirtymävaiheessa ennen seuraavaa sivua. Alla esimerkki pelkistetystä pisteenlaskennasta (Ohjelma 5).

```
    @staticmethod
    def before_next_page(player: Player, timeout_hap-
pened):

        # players payoffs
        if player.num_trials == 0:
            player.payoff = 0

        # customer type 1 high
        if player.round_number == 1 or player.round_num-
ber == 4:

            if player.num_correct == 0:
                player.charity = 0
```

Ohjelma 5. Ennen seuraavalle sivulle siirtymistä laskettiin pelaajan pisteet.

Nämä lasketut arvot tallennettiin Player luokassa sijaitseviin muuttujiin. Luokalla tarkoitetaan esimerkiksi tässä tapauksessa, jokaiselle osallistujalle pelin alussa luotavaa Player oliota. Jokainen Player olio sisältää määritelmät niille muuttujille, mitä kyseisen olion tulisi sisältää. Koska jokaisen uniikin pelaajan pisteitä ja edistymistä haluttiin seurata erikseen, jokaiselle pelaajalle tuli luoda oma olio pelin alussa. Alla esimerkki Player luokasta (Ohjelma 6).

```
class Player(BasePlayer):
    iteration = models.IntegerField(initial=0)
    num_trials = models.IntegerField(initial=0)
    num_correct = models.IntegerField(initial=0)
    num_failed = models.IntegerField(initial=0)
    charity = models.CurrencyField(initial=0)
```

```
total_charity = models.CurrencyField(initial=0)
```

Ohjelma 6. Player luokan muuttujat.

Muuttujalla tarkoitetaan jotain arvoa, joka voidaan tallentaa käytettäväksi myöhemmin koodissa. Muuttujan arvoa voidaan nimensä mukaisesti myös muuttaa myöhemmin koodissa. Esimerkiksi num_trials, yläpuolella olevassa koodiesimerkissä on muuttuja, johon tallennettiin kyseisellä kierroksella pelaajan yritysten määrä. Kun taas num_failed muuttujaan tallennettiin, epäonnistuneiden yritysten määrä.

Otree muodosti automaattisesti jokaisen osallistujan muuttujista datan tarkasteltavaan muotoon admin käyttöliittymään (Kuva 3). Data muodostui jokaisesta kierroksesta erikseen ja jokainen kierros oli erikseen tarkasteltavissa. Kaikkien kierrosten ja pelaajien data pystyttiin pelin lopuksi lataamaan sivustolta CSV-tiedostona.

	group	id_in_group	role	payoff	iteration	num_trials	num_correct	num_failed	charity	total_charity	group. id_in_subsession
P1	1	1		3.0	5	4	3	1	4.0	4.0	1
P2	1	2		1.0	3	2	1	1	1.0	1.0	1
P3	1	3		3.0	4	3	0	3	0.0	0.0	1
P4	1	4		0.0	0	0	0	0	0.0	0.0	1

Kuva 3. Kuvankaappaus admin käyttöliittymän Data välilehdestä, kuvassa oleva data on ensimmäiseltä kierrokselta.

4.3.2 Palvelin

Kun pelit toimivat halutulla tavalla, oli aika luoda palvelin,
joka hostaisi pelejä. Otimmekin siis yhteyttä Tampereen Yliopiston help deskiin ja esitimme palvelin tarpeemme. Palvelin oli Linux pohjainen. Help desk loi minulle tunnukset palvelimelle, joilla pystyisin hallitsemaan palvelinta. Loin myös SSH avainparin, joka lisättiin palvelimelle, jotta saimme pidettyä palvelimen turvallisempana. Tämä mahdollistaa sen, että palvelimelle ei pystytä

yhdistämään SSH yhteydellä, jos yhdistävä osapuoli ei omista toista avainparia. Käytännössä avainparien luonti mahdollisti sen, että palvelimelle ei voinut yhdistää minun tunnuksillani muualta kuin minun koneeltani. Käytin palvelimelle yhdistämiseen SSH yhteyttä. Ohjelmana toimi PuTTY. Käytin tiedostojen siirtämiseen palvelimelle SFTP-yhteyttä. Ohjelmana toimi WinSCP.

Nyt olikin tarpeellista asettaa oTree production tilaan. Tähän asti peliä oli hostattu development tilassa, joka on tarkoitettu kehitysvaiheessa käytettäväksi, mutta sitä ei tulisi käyttää, kun pidetään oikeita pelikokeita. Tällä tavoin sivusto pystyttiin esimerkiksi myös suojaamaan käyttäjätunnuksilla ja salasanalla, jotta kuka vain ei pääsisi käynnistämään ja muokkaamaan pelejä. Kansion juureen lisättiin environment variable eli ympäristömuuttuja, joka sisälsi määrittelyt salasanalle, käyttäjätunnukselle, sekä pelin autentikaatio tasolle. Environment variable käytännössä pitää sisällään avainarvo pareja, joita oTree prosessi voi sieltä käydä ”kysymässä” ilman että tarvitsisi kirjoittaa salassa pidettävää tietoa, kuten salasanoja itse kooditiedostoon.

Autentikaatio tasoksi asetettiin tässä tapauksessa ”STUDY”, koska haluttiin käyttää peliä production tilassa. STUDY tilassa, peliä voi pelata vain, jos osallistujalle annetaan linkki kyseiseen peliin, tai hän tietää sivuston admin käyttäjätunnukset. Admin käyttäjätunnukset ovat järjestelmänvalvojan tunnukset ja niillä voidaan esimerkiksi käynnistää uusia pelikokeita, sekä tarkastella aikaisemmin pelattujen pelien dataa. Autentikaatio taso on oletuksena ”DEMO”, joka on sopiva pelin kehitysvaiheessa, mutta ei oikeassa pelikoe tilanteessa. DEMO tilassa kuka vain voi vierailla sivustolla ja pelata demo versiota peleistä, ilman että on yksilöityä linkkiä, tai käyttäjätunnuksia (oTree, n.d). Oli myös tärkeää hostata peliä salatulla https yhteydellä salaamattoman http yhteyden sijaan. Tässä käännyimme jälleen kerran help deskin puoleen, josta ratkaisu löydettiin.

4.3.3 Haasteet liittyen tekniseen toteutukseen

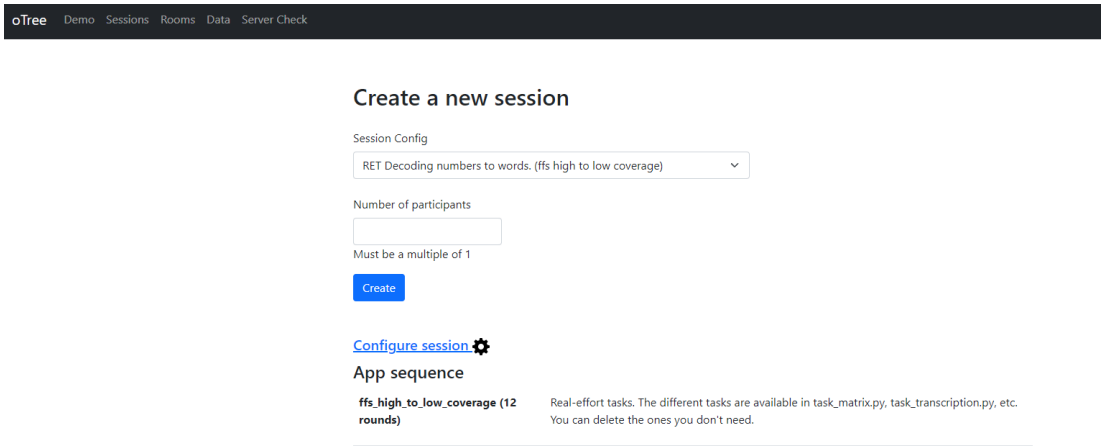
Kuten aiemmin mainitsin, tutkimustamme vastaavaa toteutusta ei ollut aikaisemmin tehty, joten esimerkkien löytäminen oli haastavaa, sekä työlästä. Jos tekisin toteutuksen uudelleen; ohjelmoinnin näkökulmasta kiinnittäisin enemmän huomiota koodin luettavuuteen, koodin kommentointiin, sekä yrittäisin vähentää toistuvasti kirjoitettua koodia. Näin koodin muokkaaminen tulevaisuudessa olisi helpompaa, sekä joku toinen osapuolikin ymmärtäisi helpommin koodin toimivuuden, joka johtaisi nopeampaan kehitystyöhön. Kiinnittäisin myös enemmän huomiota muuttujien ja tiedostojen nimeämiseen. Näin projektin jälkeen, kun luen omaa koodiani, huomaan että jotkut nimeämäni muuttujat saattavat jättää epäselväksi, mitä ne tarkoittavat. Esimerkiksi nimesin ohjesivut ”instructions1” ja ”instructions1_low”, jos tekisin tämän uudestaan, nimeäisin ne ”instructions1_high”, sekä ”instructions1_low”, jotta ei olisi epäselvää missä osassa peliä näitä kyseisiä sivuja käytetään.

Projektin aikana huomasin myös moneen otteeseen dokumentoinnin tärkeyden, välillä tuli tilanteita, joissa en enää tarkalleen muistanut mitä, tai miten olin jonkun tietyn ominaisuuden ohjelmoinut, näissä tilanteissa olisi ollut erittäin hyödyllistä olla parempaa dokumentaatiota projektin kulusta. Vaikkakin kirjoitin ”päiväkirja” tyyppistä dokumentaatiota silloin tällöin, tätä olisi pitänyt tehdä vielä enemmän. Esimerkiksi nyt lopputyötäni kirjoittaessa olisi ollut erittäin avuliasta, jos olisin päässyt tarkastelemaan aikajärjestyksessä, mitä olin tehnyt milloinkin.

4.4 Pelikokeiden vetäminen

Pidimme pelikokeet Tampereen Yliopiston DMLaboratory luokassa (decision making lab). Manela oli aikaisemmin tehnyt esitteet pelikokeesta ja niitä jaettiin eri alojen opiskelijoille. Pelikokeesta kiinnostuneet pystyivät ilmoittautumaan pelikokeeseen ja valitsemaan heille sopivan päivän pelikokeeseen osallistumiseen. Peliin osallistujat eivät tienneet minkä tyyppinen pelikoe oli kyseessä. He saivat valita, osallistuivatko he pelikokeeseen, jossa pelinkulku ja säännöt käytiin läpi suomeksi tai englanniksi.

Pelikoepäivinä saavuimme Katrin, sekä Manelan kanssa Yliopistolle noin puolta tuntia ennen pelikokeen alkamista. Varmistimme että luokka oli siisti ja valmis käytettäväksi. Sovimme tässä kohtaa minkä tyyppisen pelin tulisimme suorittamaan (salary, low to high, high to low). Tarkastimme osallistujamäärät ja laitoimme jokaiselle tietokone pisteelle allekirjoitettavat dokumentit, jotka olivat: suostumuslomake, henkilötietolomake sekä maksutosite. Käynnistin pelikokeen selaimesta kirjautumalla web käyttöliittymän kautta sisään pelikokeen admin käyttöliittymään. Valitsin Sessions välilehdeltä suoritettavan pelin tyypin ja osallistujamäärän, alla näytönkaappaus käyttöliittymästä (Kuva 4).



Kuva 4. Kuvankaappaus admin käyttöliittymästä pelin asetusten valitsemisesta.

Otree loi automaattisesti niin monta uniikkia linkkiä, kuin osallistujamäärä oli. Kun peli oli luotu, enää puuttuivat osallistujat, joille voitaisiin lähettää linkit peliin osallistumiseksi.

Kun pidimme ensimmäisiä pelikokeita, päätimme itse asettaa kirjautumislinkit jokaiselle tietokoneelle valmiiksi, ennen kuin osallistujat saapuivat paikoilleen. Myöhemmin muutimme lähestymistapaamme, antamalla jokaiselle paperinpalan, josta osallistujat joutuivat itse kirjoittamaan linkin manuaalisesti. Tämäkään ei ollut optimaalinen ratkaisu mutta vähensi mahdollisuutta kirjautua vahingossa kahdelle koneelle samalla linkillä. Syy miksi emme lähettäneet linkkejä digitaalisesti osallistujille oli se, että osallistujat eivät pelanneet peliä omilla laitteillaan, vaan he käyttivät Tampereen Yliopiston DMLabissa olevia

tietokoneita. Näillä kyseisillä tietokoneilla ei myöskään ollut sähköposti osoitteita, joihin linkit olisi voinut jakaa.

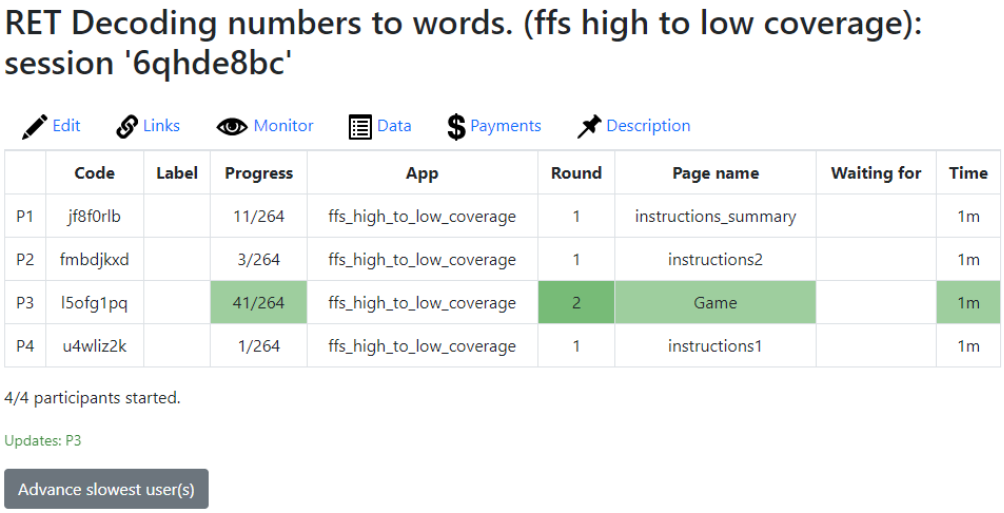
Kun osallistujat saapuivat luokkaan, heidät ohjattiin omille paikoilleen ja täyttämään edellä mainitut dokumentit. Osallistujille myös mainittiin, että he eivät saa vielä alkaa pelaamaan peliä vaan kaikki aloittavat samaan aikaan ja käydään ensin ohjatusti pelin kulku läpi.

Joskus kaikki peliin ilmoittautuneet eivät syystä tai toisesta saapuneet paikalle. Koska olimme jo aikaisemmin käynnistäneet pelin ilmoittautuneiden osallistuja määrällä (pelin käynnistämiseen kuluvan ajan säästämiseksi), pelissämme oli näissä tapauksissa enemmän pelaajapaikkoja, kuin osallistujia oli oikeasti. Näissä tapauksissa merkitsimme ylös ne ”tyhjät” osallistuja tunnukset, joissa ei ollut oikeaa pelaajaa. Pelin lopuksi poistimme, tai jätimme datan analysointi vaiheessa nämä kyseiset osallistuja tunnukset pois analyysistä.

Sitten oli aika aloittaa itse pelikoe. Koe alkoi pelin kulun ja ohjeiden ohjatulla läpikäynnillä. Jos sessio oli englanninkielinen Katri tai Manela johtivat tätä vaihetta. Silloin kun sessio oli suomenkielinen, minä vastasin pelikokeen vetämisestä.

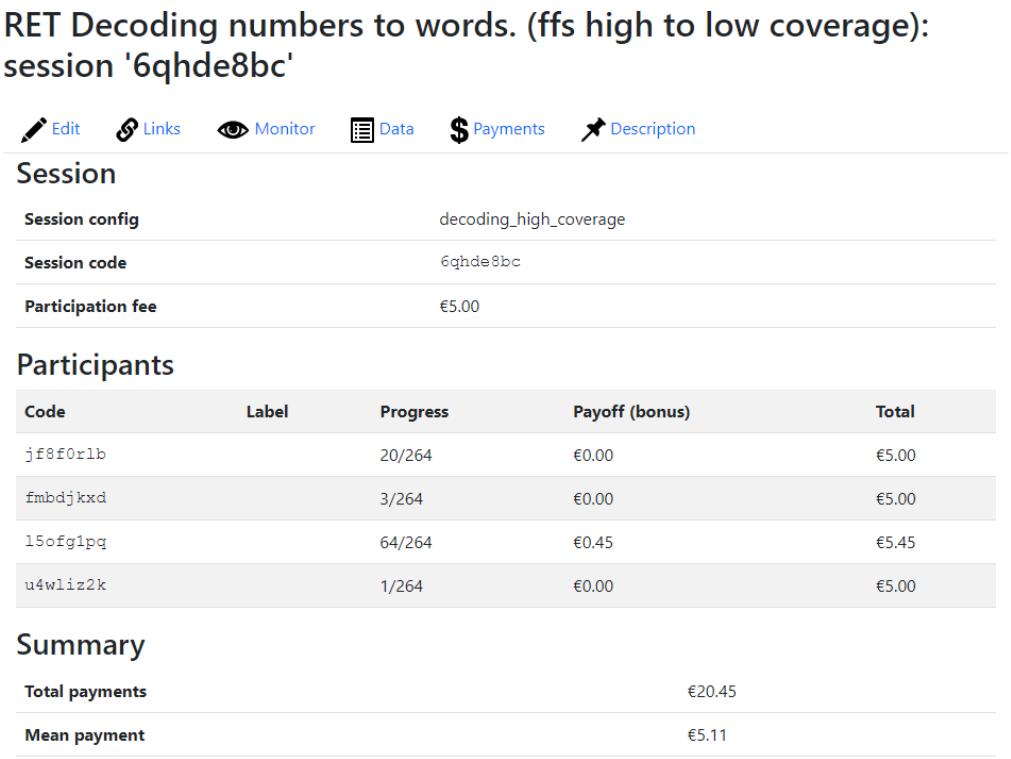
Pelaajille luettiin pelin jokaisen sivun ohjeet suomeksi tai englanniksi (riippuen kyseisestä sessiosta) ja ne käytiin yksitellen läpi yhdessä. Kun olimme yhdessä käyneet kaikki ohjeet läpi, kysyimme osallistujilta, onko heillä jotain kysyttävää ja ohjeistimme nostamaan kätensä ylös, jos pelin aikana tulee jotain ongelmia tai kysyttävää. Kerroimme myös, että pelin lopussa on linkki kyselyyn, joka tulee täyttää ennen, kuin he voivat lunastaa pelin aikana kerätyt ansiot. Kun kysely oli täytetty, osallistujaa ohjeistettiin ottamaan heidän pöydällään oleva paperilappu (joka sisälsi osallistujalinkin), mukaansa ja tulemaan luokan eteen, jossa heille maksettaisiin pelin aikana kerätyt ansiot. Kyseinen linkki sisälsi pelaajan tunnuksen, jonka kautta pystyimme tarkastamaan, mikä pelaaja on kyseessä ja tarkastamaan hänen suoriutumisensa pelissä, sekä hänen ansionsa.

Kun pelin kulku ja ohjeet olivat kaikille selkeää, annoimme luvan aloittaa pelin. Pelin aikana pystyimme seuraamaan, millä kierroksella ja mitä valintoja osallistujat olivat tehneet reaaliajassa. Tästä näytönkaappaus alapuolella (Kuva 5).



Kuva 5. Näytönkaappaus admin käyttöliittymästä pelin aikana osallistujien reaaliaikaisesta seurannasta.

Kun osallistuja oli saanut suoritettua kokeen loppuun ja täyttänyt loppukyselyn, hän saapui luokan eteen oman osallistuja tunnuksensa ja pelin alussa täytettyjen dokumenttiensa kanssa. Tarkastimme, että kyseinen osallistuja oli suorittanut pelin loppuun ja mitkä hänen ansionsa olivat admin käyttöliittymän ”Payments” välilehdeltä. Tästä näytönkaappaus: (Kuva 6).



Kuva 6. Kuvankaappaus admin käyttöliittymästä, josta näimme pelaajien reaaliaikaiset ansiot

Jokainen pelaaja sai kokeeseen osallistuessaan vähintään 5 € osallistumisrahaa, johon lisättiin pelin aikana tehdyistä valinnoista koostuva summa. Osallistujille maksettiin joko käteisellä, tai MobilePay:lla.

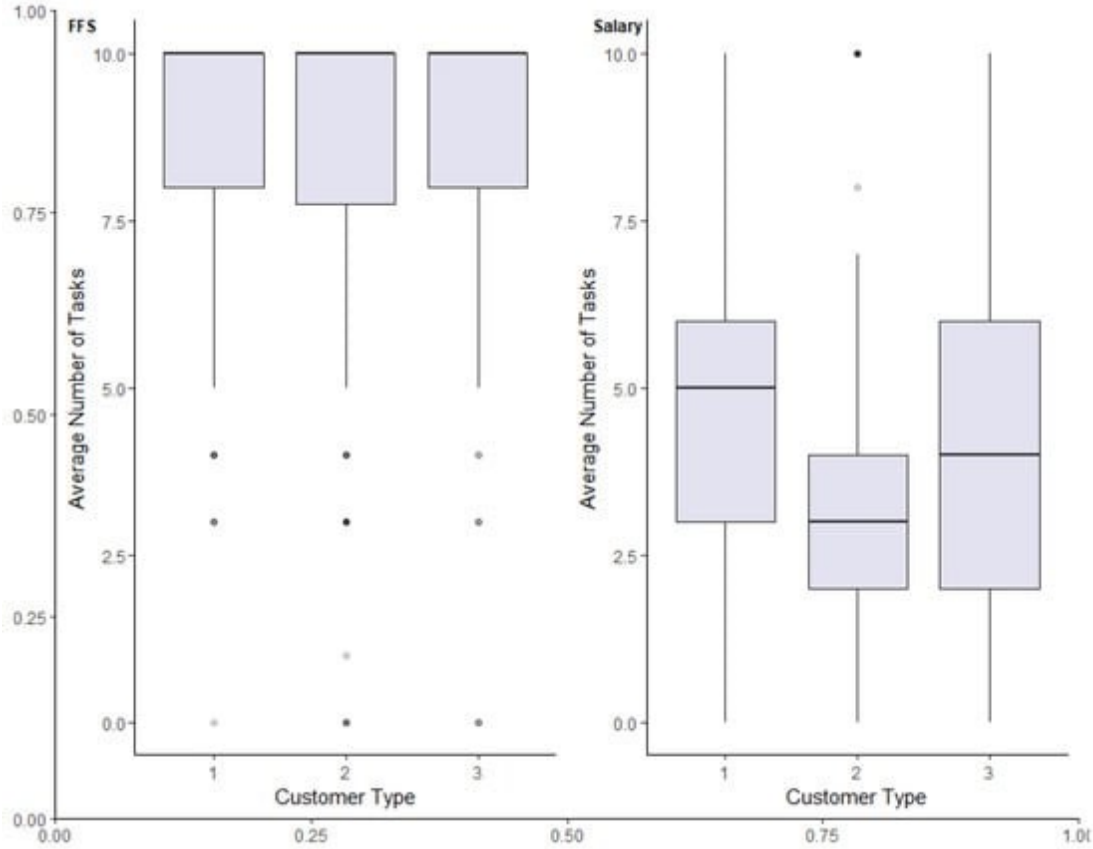
Kun kaikille osallistujille oli maksettu ja he olivat poistuneet, latasimme kyseisen session datan välittömästi talteen ja nimesimme sen pelin tyypin, sekä päivämäärän mukaan. Tallensimme pelin datan yhteiseen DropBox kansioomme, johon vain meidän tutkimus ryhmällämme oli pääsy.

Data saatiin ladattua csv tiedostona suoraan admin käyttöliittymästä ja se sisälsi jokaisen osallistujan tekemät valinnat kaikilla kierroksilla. Tästä kyseisestä datasta siivottiin pois ”tyhjät” osallistujat (aikaisemmin mainitut ilmoittautuneet pelaajat, jotka eivät osallistuneetkaan peliin).

5 JOHTOPÄÄTÖKSET

Projektin lopputulos oli onnistunut ja opettavainen. Ennen projektin aloittamista oma tietämykseni tutkimustyöstä, oTreesta, sekä kokeellisesta taloustieteestä oli rajallista ja koen että projektin myötä ymmärrykseni on parantunut kaikkia näitä kohtaan huomattavasti. Myös ohjelmointitaitoni, sekä teknologioiden ymmärrykseni parantui huomattavasti projektin kuluessa.

Tutkimuksen tulokset olivat erittäin mielenkiintoisia. FFS palkkajärjestelmän omanneet osallistujat yrittivät maksimoida omaa hyötyään. Eli he tekivät liikaa työtä asiakkaan tarpeeseen nähden. Salary palkkajärjestelmän omanneet osallistujat tekivät yleisesti liian vähän työtä asiakkaan tarpeeseen nähden. Voitiin huomata, että FFS osallistujat eivät välittäneet asiakkaan tarpeesta, vaan yrittivät vain maksimoida omaa hyötyään. Salary osallistujat taas huomioivat osittain asiakkaan tarpeen, mutta tekivät silti liian vähän työtä asiakastyypille kolme. Alla olevasta kaaviosta (Kuva 7), voidaan vertailla tuloksia ja asiakkaiden tarpeita. Asiakastyyppi yhden optimaalinen määrä suoritettuja tehtäviä oli viisi. Asiakastyyppi kahden optimaalinen määrä suoritettuja tehtäviä oli kolme. Asiakastyyppi kolmen optimaalinen määrä suoritettuja tehtäviä oli seitsemän.



Kuva 7. Boxplot kaavio (Karunadasa, M., Sieberg, K. K. & Jantunen, T. T. K., 2023, s. 14) tutkimusten lopullisista tuloksista.

FFS osallistujat tekivät enemmän virheitä, kuin salary peliin osallistuneet. Vakuutus ei vaikuttanut tuloksiin. Datan perusteella salary palkkajärjestelmä on turvallisempi ja hyödyllisempi asiakkaan näkökulmasta katsottuna.

Tutkimuksesta julkaistiin artikkeli, jossa olin itse myös osallisena Katri Siebergin ja Manela Karunadasan kanssa (Karunadasa, M., Sieberg, K. K. & Jantunen, T. T. K., 2023).

Tutkimuksen julkaisun jälkeen olemme aloittaneet uuden projektin, joka on yhä lopputyön kirjoittamisen aikaan käynnissä. Tässä uudessa projektissa tehtävänkuvani oli sama kuin kirjoittamassani opinnäytetyön projektissa.

Uudessa projektissa muutettiin palveluntarjoaja lääkäriksi ja asiakas potilaaksi. Halusimme nähdä vaikuttaisiko terveydenhuollon teema osallistujien valintoihin. Puolet osallistujista olivat potilaan roolissa ja toinen puoli lääkärin

roolissa. Lääkärit saivat palveltavakseen satunnaisesti uuden potilaan, joka kierroksella. Potilaille maksettiin rahasummana, se määrä, jonka lääkärit olivat heille kierrosten aikana ansainneet. Tutkimuksen tässä vaiheessa näyttäisi siltä, että lääkäri osallistujat kiinnittävät enemmän huomiota potilaiden tarpeisiin, kuin aikaisemmassa tutkimuksessa, jota tämä opinnäytetyö käsittelee.

Vaikka tekninen toteutus on ollut lääkäri- potilas -projektissa haastavampaa, se on silti ollut myös erittäin palkitsevaa ja mielenkiintoista. Olen erittäin otettu, että pääsin näin hienoihin projekteihin mukaan työskentelemään erittäin ammattitaitoisten ja osaavien henkilöiden kanssa.

LÄHTEET

Chen, D. L., Schonger, M. & Wickens, C. (2016). oTree—An open-source platform for laboratory, online, and field experiments. Journal of Behavioral and Experimental Finance, 9, (s. 88–97). https://doi.org/10.1016/j.jbef.2015.12.001

Chris @ oTree. (2021). otree-realeffort. Otreehub. https://www.otreehub.com/projects/otree-realeffort/

Eckel, C. C., Grossman, P. J. (1996) Altruism in Anonymous Dictator Games, Games and Economic Behavior, Volume 16, Issue 2: 181-191, ISSN 0899-8256. https://doi.org/10.1006/game.1996.0081

Fischbacher, U. (1999). z-Tree. Toolbox for readymade economic experiments. IEW Working paper 21, University of Zurich.

Holt, C. A. (2005). Veconlab. Experimental Economics Laboratory. https://veconlab.econ.virginia.edu/

Karunadasa, M., Sieberg, K. K. & Jantunen, T. T. K. (2023). Payment Systems, Supplier-Induced Demand, and Service Quality in Credence Goods: Results from a Laboratory Experiment. Games 14, 46. https://doi.org/10.3390/g14030046

Microsoft. (2024). Visual Studio Code. https://code.visualstudio.com/docs

Otree. (n.d.) oTree:n dokumentaatiosivu. Haettu 17.04.2024 osoitteesta https://otree.readthedocs.io/en/latest/

Otree. (n.d.) oTree:n dokumentaatiosivu admin käyttöliittymästä. Haettu 17.04.2024 osoitteesta https://otree.readthedocs.io/en/latest/admin.html

Otree. (n.d.) oTree:n dokumentaatiosivu sivujen toiminnasta. Haettu 17.04.2024 osoitteesta https://otree.readthedocs.io/en/latest/pages.html

Otree. (n.d.) oTree:n dokumentaatiosivu oTree:n asentamisesta ja käyttöönotosta. Haettu 17.04.2024 osoitteesta https://otree.readthedocs.io/en/latest/install-nostudio.html#install-nostudio

Pip. (2024). Pypi. https://pypi.org/project/pip/

Python Software Foundation. (2024). Python. https://www.python.org/

Refsnes Data. (2024). CSS Introduction https://www.w3schools.com/css/css_intro.asp

Refsnes Data. (2024). HTML Introduction. https://www.w3schools.com/html/html_intro.asp