1. **Henkilötiedot**

otsikko:	Love It Or List It
projektin aihe:	330 Sisustussuunnittelu
nimi:	Teea Ahola
opiskelijanumero:	101744236
koulutusohjelma:	Tietotekniikka
vuosikurssi:	2023
päiväys:	18.4.2024

2. **Yleiskuvaus**

Toteutettu ohjelma on sisustussuunnittelutyökalu, jossa käyttäjä voi suunnitella valmiin pohjapiirustuksen päälle tai itse luotuun pohjaan haluamansa näköisen asunnon. Asunnon elementtejä voi liikuttaa ja muokata lisäämisen jälkeen, ja ohjelma tarkistaa, etteivät ne mene toistensa päälle. Lopputuloksen voi tallentaa kuvana.

Työstä on toteutettu vaikea versio.

3. **Käyttöohje**

Ohjelman käynnistyessä käyttäjä näkee pohjapiirustuksen suunnittelunäkymän, jossa pohjapiirustukseen voi lisätä seiniä, ikkunoita, kiintokalusteita ja ovia. Näkymän ylälaidassa olevista valikoista käyttäjä voi valita kuvan, jota käyttää pohjapiirustuksena, tallentaa nykyisen suunnitelman kuvatiedostona, asettaa suunnitelmalle mittakaavan, tyhjentäää ohjelman muistin ja aloittaa alusta, tai lukea ohjelman pikaohjeet. Halutessaan rakentaa suunnitelman itse, käyttäjä voi klikata haluamiansa elementtejä, antaa niille mitat, kääntää ne haluamaansa orientaatioon 45° välein ja lisätä ne pohjapiirustukseen. Ennen minkään elementin lisäämistä tulee käyttäjän asettaa valikosta haluamansa skaalaus pohjapiirustukselle.

Kun pohjapiirustus on valittu tai valmis, voi käyttäjä napin painalluksella vaihtaa näkymän sisustusnäkymään. Sisustusnäkymässä on useita eri värisiä ja muotoisia huonekaluja valittavana lisättäväksi pohjapiirustukseen. Huonekalut lisätään samalla tavalla kuin muut elementit, mutta niillä on useampia vaihtoehtoja niiden muokkaamiseen. Huonekaluille voi valita mittojen lisäksi nimen, materiaalin ja värin.

Muotoa lukuun ottamatta kaikkia elementin tietoja voi muokata jälkikäteen. Myös kaikkia huonekaluja voi kääntää haluamallaan tavalla.

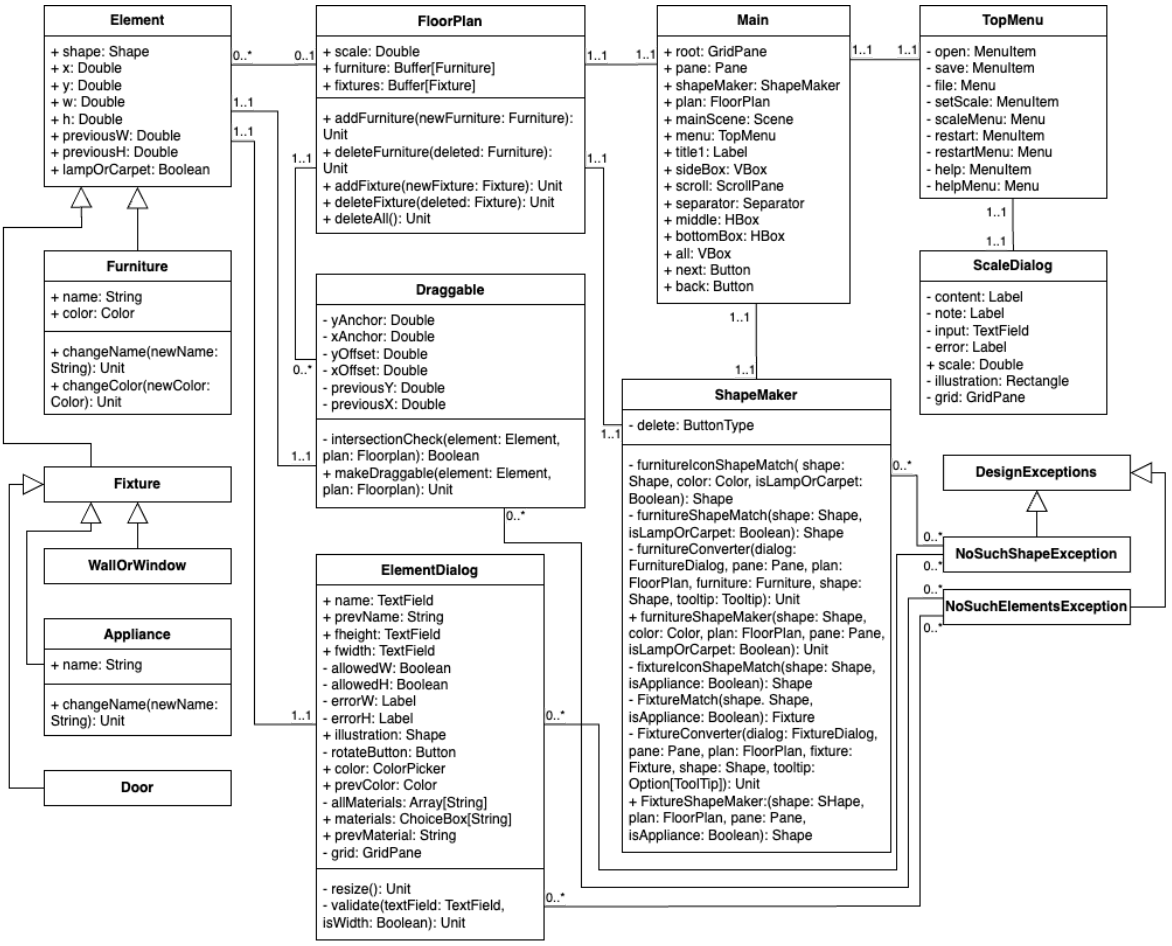
Mitään elementtiä ei voi lisätä pohjapiirustukseen antamatta sille mittoja. Elementin nimeäminen ei kuitenkaan ole pakollista jotta sen saisi lisättyä. Pohjapiirustukselle voi antaa mittakaavan, jonka mukaan kaikki lisätyt elementit skaalataan. Kun elementti on jo lisätty pohjapiirustukseen, voi sitä vielä siirtää hiirellä raahaamalla tai muokata tuplaklikkaamalla. Elementin voi poistaa sen muokkausnäkymästä.

Pääosin elementit eivät saa mennä toistensa päälle suunnitelmassa. Pohjapiirustuksen rakentamisen helpottamiseksi seinät ja ikkunat voivat kuitenkin mennä toistensa kanssa päällekkäin. Jos pohjapiirustuksen tuo kuvana, on käyttäjän itse silmämääräisesti tarkistettava että huonekalut eivät mene seinien ja kodinkoneiden päälle. Pohjapiirustukseen voi tavallisten huonekalujen lisäksi lisätä mattoja tai lamppuja, jotka voivat mennä muiden huonekalujen kanssa päällekkäin. Jotta matot näkyisivät huonekalujen alla käyttöliittymässä, on ne lisättävä pohjapiirustukseen ennen muita huonekaluja. Samasta syystä lamput on lisättävä viimeisenä. Käyttäjän yrittäessä siirtää elementtiä jonkin toisen päälle, elementti palautuu lähtökohtaansa, ja jos se on huonekalu, muuttuu punaiseksi.

Lopputuloksen voi tallentaa kuvana (jpg- tai png-tiedostona). Halutessaan, käyttäjä voi tyhjentää koko pohjapiirustuksen ja skaalauksen ja aloittaa alusta.

4. Ohjelman rakenne

Toteutettu luokkajako on hyvin erilainen kuin teknisessä suunnitelmassa esitetty. En ollut ohjelmoinut paljoa, juuri käyttänyt ScalaFX-metodeita tai luonut omaa graafista käyttöliittymää. Näin ollen alkuperäinen suunnitelma oli hyvin puutteellinen, sisälsi paljon turhia luokkia, muuttujia ja metodeita, ja siitä puuttui paljon tärkeitä asioita. Esimerkiksi oman “Muoto”-luokan tekeminen olisi ollut täysin turhaa, sillä kaikki oli hyvin paljon kätevämpää toteuttaa ScalaFX:n valmiilla Shape-luokalla. Samoin “Materiaali”-luokka oli turha, sillä materiaaleilla ei ollut kuin nimi, eli se oli kätevämpää vain toteuttaa String-arvoilla. Kiintokalusteet ja huonekalut sen sijaan yhdistin Element-luokan alaluokiksi, sillä niillä oli paljon samankaltaisuuksia (muoto, sijainti, mitat, entiset mitat) ja näin vähensin toisteisuutta koodissa.



Lopullinen UML-kaavio. Kaikkia muuttujia ja metodeita ei ole merkitty.

Alkuperäisen suunnitelmani mukaan huonekaluja olisi erilaisia tyyppejä, kuten “sänky” tai “pöytä”, ja näillä eri tyypeillä olisi eri ominaisuuksia. Tätä toteuttaessani päädyin kuitenkin lopputulokseen, että on paljon järkevämpää toteuttaa huonekalut vain eri muotoisina mutta samanarvoisina olioina. Tämä toteutus yksinkertaisti koodia ja vähensi toistoa. Tähän poikkeuksena ovat lamput ja sohvat, jotka voivat mennä muiden huonekalujen alle tai päälle (riippuen lisäämisjärjestyksestä).

Moni asia, mitä olin alunperin suunnitellut toteuttavani huonekalu- tai kiintokaluste-luokkien avulla, kuten elementin kääntäminen, tapahtuu nyt ElementDialog-luokassa. Dialogista käsin voi käyttäjä muokata elementtiä, ja muutosten tultua hyväksytyiksi, siirtyy niiden käsittely ShapeMakerin furnitureConverterille tai fixtureConverterille.

ShapeMaker-luokan keskeisimmät metodit ovat furnitureShapeMaker ja fixtureShapeMaker, joilla luodaan käyttöliittymän sivupaneeliin klikattavat muodot.

Molemmat funktiot myös luovat ja liittävät kuhunkin kuvioon oikeanlaisen elementin ja dialogin elementin muokkausta varten. Suurin osa kaikesta mitä käyttöliittymässä tapahtuu käsitellään ShapeMakerin ja ElementDialogin yhteistyöllä, kuten huonekalujen luominen ja lisääminen pohjapiirustukseen, muokkausten käsittely ja poistaminen.

ElementDialogin keskeisimmät funktiot ovat esittelykuvion skaalaamiseen käytetty resize ja tekstikenttien tarkistamiseen käytetty validate. Validate estää käyttäjää lisäämästä pohjapiirustukseen huonekalua, jolla ei ole oikeanlaisia mittoja, ja kertoo käyttäjä vääränlaisesta datasta. Validate myös kutsuu resizeä, sillä kun tekstikenttään syöttää hyväksyttävän arvon (positiivisen numeron), resize laskee muodon leveyden ja korkeuden suhteen ja skaalaa esimerkkikuvion sen mukaisesti. Näin esimerkkikuviosta ei tule liian suurta tai pientä, mutta se näyttää aina muodon leveyden ja korkeuden suhteen oikein.

Myös elementtien liikuttamiseen tarvittava Draggable-luokka puuttui alkuperäisestä suunnitelmastani kokonaan. Draggable on luokka, joka liikuttaa yhtä elementtiä, ja pitää kirjaa sen sijainnista koordinaatistossa. Tämä tapahtuu sen makeDraggable-metodilla. Draggable myös tarkistaa elementin halutun sijainnin intersectionCheck-metodilla, ja palauttaa elementin edelliselle paikalleen jos uusi sijainti ei ole sallittu. Koska erilaisilla huonekaluilla ja kiintokalusteilla on ohjelmassa omat sääntönsä (lamput ja matot saavat mennä muiden huonekalujen päälle, huonekalut eivät saa mennä minkään kiintokalusteen päälle, kiintokalusteet eivät saa mennä toistensa päälle, paitsi seinät ja ikkunat jotka saavat), on intersectionCheck-funktiolla tärkeä osa ohjelman toiminnallisuutta.

ElementDialogissa, Draggablessa ja ShapeMakerissa on jokaisessa useampi match-case rakenne, jolla käsitellään muotoja ja/tai elementtejä. Näitä varten tein DesignExceptions-luokan, joka perii Javan Exception-luokan. Jaoin sen vielä kahteen alaluokkaan, NoSuchShapeExceptioniin ja NoSuchElementsExceptioniin. Nämä poikkeukset ottavat parametreikseen kuvauksen ongelmasta ja tulevat kutsutuiksi vain jos match ei löydä käsiteltävälle muodolle tai elementille sopivaa tapausta.

5. Algoritmit

Mitään suunnitelluista algoritmeistä ei tullut toteutettua, sillä suunnitellut funktiot pystyykin toteuttamaan ScalaFX:llä helposti, kuten muotojen kääntämisen ilman kääntömatriisia. Sen sijaan esimerkiksi taustakuvan avaamisessa käytetty algoritmi skaalaa halutun kuvan oikean kokoiseksi ja sijoittaa sen näkymään. Lisätyn kuvan kuvasuhdetta verrataan panen kuvasuhteeseen, jonka perusteella kuva sovitetaan oikean kokoiseksi niin, että sen kuvasuhden säilyy ja se mahtuu näkymään kokonaan. Kuvasuhde lasketaan jakamalla kuvan leveys kuvan korkeudella:

$$w / h$$

Kuvan sovittamisen pseudokoodiesitys:

```
Jos kuvan kuvasuhde > panen kuvasuhde
        sovita kuvan leveys panen leveyteen
        sovita kuvan korkeus panen leveyteen kerrottuna kuvan
      kuvasuhteella
Jos taas panen kuvasuhde > kuvan kuvasuhde
        sovita kuvan leveys panen korkeuteen kerrottuna kuvan
      kuvasuhteella
        sovita kuvan korkeus panen korkeuteen
Jos kuvan kuvasuhde on sama kuin panen kuvasuhde
        sovita kuvan leveys panen leveyteen
        sovita kuvan korkeus panen korkeuteen
```

Samanlaista algoritmia käytetään ElementDialogin resize-funktiossa säilyttämään esimerkkikuvion sivujen suhteet koon säilyessä samana.

6. Tietorakenteet

Pohjapiirustuksen tiedot kiintokalusteista ja huonekaluista ovat muuttuvatilaisia Buffereita, sillä niiden tärkeä toiminnallisuus on elementtien nopea ja helppo lisääminen ja poistaminen tarvittaessa. Kaikki muut käytetyt kokoelmat ohjelmassa ovat Array-tyyppisiä, mutta Arrayn tilalla voisi ihan yhtä hyvin käyttää jotain muuta kokoelmatyyppiä, sillä näille kokoelmille kutsutut metodit eivät erityisesti riipu taulukkorakenteesta.

7. Tiedostot ja verkossa oleva tieto

Ohjelmaan voi halutessaan lisätä käyttäjän omalta tietokoneelta kuvatiedoston pohjapiirustukseksi. Ohjelmaan voi lisätä vain jpg- ja png-tyyppisiä tiedostoja. En päätynyt itse kehittämään omaa tiedostotyyppiä ohjelman käsiteltäväksi, sillä se vaikutti hyvin vaikealta ja minulla ei siihen ollut aikaa. Huomasin myös, että lopputuloksen voi tallentaa kuvana, joten omaa tiedostotyyppiä ei sinänsä tarvitse jos lopputulosta ei halua muokata myöhemmin. Ohjelman testausta varten ei sinänsä tarvitse asunnon pohjapiirustusta, vaan mikä tahansa kuva omalta koneelta kelpaa.

8. Testaus

Tein suurimman osan ohjelmani testauksesta graafisen käyttöliittymän kautta. En suunnitellut tai toteuttanut aikomiani yksikkötestejä, sillä ne funktiot ja tietorakenteet, joita aioin yksikkötestata tuli poistettua ennen kuin ehdin rakentamaan yksikkötestejä, kuten vektori jossa on kaikki huonekalut jonka päällä se saa olla. Toisaalta huonekalun nimen vaihtamisen pystyi testaamaan GUI:sta kun sain Tooltipit asennettua huonekaluille (sitä enne printtasin nimen konsoliin sen vaihtuessa).Kun olin saanut graafisen käyttöliittymän alustettua, aina muokattuani koodia tarkistin, että tärkeimmät toiminnallisuudet säilyivät. Hyödynsin myös tuloksen printtausta ennen graafista implementaatiota, esim. dialogien tulostenkäsittelyssä, ennen kuin sain dialogin lisäämään graafiseen käyttöliittymään mitään, tarkistin että lisätyt ButtonTypet toimivat oikein printtaamalla “success” tai “failure” konsoliin nappia painamalla.

Erityisen tärkeää testaus oli muotojen oikean käyttäytymisen tarkistamisessa. Käyttöliittymän ajamalla huomasin aina nopeasti kun olin rikkonut muotojen liikuttamisen metodit tai jos muodon skaalaus oli pahasti pielessä. Kuvasuhteiden laskut tarkistin REPLissä.

Nykyisellään ohjelma käsittelee virheelliset syötteet ja osaa kertoa minkälainen poikkeus on kyseessä jos match-case -kutsu epäonnistuu. Nämä oli helppo testata vain laittamalla positiivisen numeron haluavaan tekstikenttään tekstiä, negatiivisia lukuja tai vain tyhjäksi jättämällä. Match-case poikkeukset tarkistin poistamalla jonkun tunnetun casen tarkasteltavasta rakenteesta ja koittamalla käyttää sitä. Näin

varmistin ennen joka committia, että koodi varmasti toimii toivotusti, ja mikään uusi muutos ei ole vahingossa rikkonut mitään tärkeää.

9. Ohjelman tunnetut puutteet ja viat

Suunnittelin lisääväni ohjelmaan mahdollisuuden lisätä huonekaluille ja kodinkoneille kuvan kyseisestä esineestä, mutta en saanut toteutusta onnistumaan. Sain kuvan lisättyä, mutta kuvan vaihtaminen ei napin painalluksella onnistunut, sillä nappi oli dialogin GridPanessa, ja olisi päivittänyt kyseistä gridiä, joka muodosti syklin gridiin, joka aiheutti ajovirheen koodiin. Kuvan lisäämiseksi minun olisi täytynyt keksiä joku aivan toinen tapa rakentaa dialogin sisältö tai lisätä kuva, ja minulla ei siihen ollut yhtään ideoita.

Kun kuvion lisää suunnitelmaan, ovat sen oletuskoordinaatit (0, 0), eli suunnitelman vasen yläkulma. Sain kuviot lisättyä keskelle näkymää, mutta silloin kuvioiden siirtäminen hajosi. Tämän korjaamiseen en keksinyt aikataulun puitteissa ratkaisua, mutta sen korjaamiseksi lähtisin muuttamaan elementtien oletuskoordinaatteja Element-luokassa, sekä Draggable-luokkaa ja erityisesti sen yksityisiä muuttujia ja metodia makeDraggable. Kuvioita voi myös nyt siirtää pohjapiirustus-Panen ulkopuolelle. Sallitun paikan tarkastuksessa yritin laittaa staattisia rajoja elementeille Draggablessa, mutta tarkistusalgoritmi ei toiminut halutulla tavalla ja rikkoi siirrettävyyden. Tähän vielä tarkasti perehtymällä saisin tämän todennäköisesti toteutettua järkevästi koko luokkaa rikkomatta, mutta en ole varma miten kyseistä algoritmia voisi parantaa, joten jätän huonekalujen paikan käyttäjälle tarkistettavaksi.

Pohjapiirustuksen skaalaus tapahtuu tällä hetkellä ShapeMakerin furnitureConverterissa ja fixtureConverterissa kertomalla lisättävän tai muokattavan elementin mitat skaalauskertoimella. Näin ollen jos skaalaa muuttaa kesken suunnittelun, eivät jo suunnitelmassa olevat elementit skaalaudu uuteen mittakaavaan. Jos ne haluaa kuitenkin skaalata, voi niiden muokkausnäkymän avata ja painaa “Apply” koskematta mittoihin ja ne skaalautuvat. Vaikka tämä toimii, ei kyseessä kuitenkaan ole kovin käyttäjäystävällinen ratkaisu. Alunperin yritin toteuttaa skaalausta skaalaamalla panen, jolloin skaalaus implementoituisi

automaattisesti kaikille siinä oleville muodoille. Tämä kuitenkin skaalasi myös pohjakuvan ja rikkoi muotojen raahaamisperiaatteen.

Ideaalitilanteessa lisätty matto lisättäisi aina muiden huonekalujen alle ja lamppu muiden huonekalujen päälle. Nykyisellä implementaatiolla tämä ei kuitenkaan onnistu, sillä ShapeMakerissa kaikki elementit lisätään Panen lapsiin luomisjärjestyksessä, eli uusin on aina päällimmäinen. Yritin ratkaista tätä yhdessä vaiheessa luomalla Panen sijaan StackPanen, jolla olisi omat kerrokset matoille, kaikelle muulle ja lampuille, mutta tämä oli huono toteutustapa, sillä StackPanessa vain ylimmän kerroksen Nodeihin pääsee käsiksi. Jätin siis nykyisen version, joka toimii, vaikka ei kaikkein käyttäjäystävällisin olekaan. Jos matot haluaisi lisätä aina pohjalle, voisi sen varmaan tehdä korvaamalla Panen children-listan toisella, jossa matto on lisätty listan alkuun. Tässä pitäisi myös tarkista onko käyttäjä valinnut valmista pohjapiirustusta, sillä mattoja ei haluta sijoittaa sen alle (pohjapiirustus on aina Panen ensimmäinen Node). Lamppujen ongelmaa tämä ei ratkaisisi.

Suunnitelmaani kuului oman tiedostotyypin kehittäminen niin, että pohjapiirustuksen voisi tallentaa ja myöhemmin ava uudestaan ja jatkaa muokkausta. Nyt Pohjapiirustuksen voi tallentaa ja muokata myöhemmin, mutta kaikki aiemman session lisäykset ovat nyt osa kuvaa ja niitä ei voi enää siirtää tai muuten muokata. Ohjelma ei myöskään tarkista päällekkäisyyksiä niiden kanssa.

10. 3 parasta ja 3 heikointa kohtaa

Kolme parasta:

1. Omat dialogit ja niiden tekstikenttien syötteenkäsittely. Virheellistä syötettä ei voi antaa eteenpäin ohjelman muille osille käsiteltäväksi.
2. Dialogien havainnekuvien reaaliaikainen skaalaus kuvasuhteet säilyttäen.
3. Elementtien nimien Tooltipit. Pieni mutta omasta mielestäni hyvä yksityiskohta on, että sivupalkissa näkee elementtien tyypit viemällä hiiren niiden päälle ja pohjapiirustuksessa näkee huonekalujen nimet.

Kolme heikointa:

1. Elementtien lisäys pohjapiirustukseen. Elementit lisätään suunnittelunäkymän (Panen) vasempaan ylänurkkaan eikä keskelle.
2. Koodin toisteisuus. Refaktoroimalla olen onnistunut suurimman osan koodin toisteisuudesta poistamaan, mutta silti esim. ScaleDialog ja ElementDialog suorittavan hyvin samankaltaista koodia välillä. Monesti, kuten dialogeissa, toisteisuuden poistaminen olisi kuitenkin tehnyt koodista epäselvää luettavaa.
3. Elementtien sijoitus käyttöliittymässä. Nykyisellään elementtejä voi siirtää sivupalkin tai menun päälle ohjelman estämättä tai huomauttamatta tästä.

11. Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Kuten jo aiemmin mainitsin, en päätynyt noudattamaan suunnitelmaani kovinkaan tarkasti, sillä lähtiessäni sitä toteuttamaan, huomasin, kuinka puutteellinen se oli. Näin ollen projektin rakenne on elänyt runsaasti ajan edetessä.

Aikataulunkaan kannalta en päätynyt noudattamaan alkuperäistä suunnitelmaa. Suurimman osan ajasta käytin käyttöliittymän toteutukseen, ja suhteessa paljon vähemmän kaikkeen muuhun. Toteutusjärjestys pysyi suunnilleen samana toteuttamatonta osaa lukuun ottamatta, mutta aika-arvioni menivät täysin pieleen.

Omien laskujeni mukaan projektiin meni yhteensä n. 90 tuntia (lukuunottamatta dokumentaatioiden kirjoittamista). Tämä on vain 10h enemmän kuin teknisessä suunnitelmassa laitoin, mutta ajankäyttö toteutui täysin eri tavalla. Alkuviikkoina minulla oli paljon muita koulukiireitä, joten käytin huomattavasti vähemmän aikaa projektiin kuin olin suunnitellut. Loppuviikkoina yritin tätä kiriä ja käytin viimeisen kuukauden aikana projektiin n. 20h viikossa. Olin myös täysin väärin arvioinut paljonko aikaa mihinkin projektin osaan kuluu. Luokkarakenteen rakentamiseen kului ehkä puoli tuntia suunnitellun 10 sijaan, mutta graafisen käyttöliittymän tekemiseen ja testaamiseen meni moninkertaisesti enemmän aikaa kuin olin suunnitellut. Tiedostojen luomisen selvittelyynkin kului paljon aikaa, mutta implementaatiota ei tullut ollenkaan. Suurin osa ajasta meni graafisen käyttöliittymän kanssa painimiseen 2. sprintistä lähtien.

Koen oppineeni projektin aikana hyvin paljon kaikkea uutta. Kuten jo mainitsin, tämä oli ensimmäinen kertani tekemässä itse isoa projektia yksin, ja ensimmäinen kertani rakentamassa graafista käyttöliittymää. Esim. alkuperäisessä suunnitelmassani halusin pystyä vaihtamaan suunnittelu- ja sisustusnäkymiä helposti napin painalluksella, mutta tätä toteuttaessani törmäsin lukuisiin ongelmiin ohjelman ajonaikaisten virheiden kanssa. Päätin siis poiketa suunnitelmastani ja vain yhdistää näkymät. Kuitenkin projektin edetessä opin enemmän ScalaFX:n toiminnasta ja sainkin loppujen lopuksi halutun vaihdon toteutettua, jopa useammalla eri tavalla (nykyinen ohjelmassa oleva tapa vie vaihtoehdoista vähiten koodia, mutta ei oikeasti vaihda koko näkymää vaan vain sivupaneelin. Koin tämän kuitenkin käyttäjäkokemuksen näkökulmasta samaksi asiaksi, joten en jättänyt koodiin monimutkaisempaa tapaa).

12. **Kokonaisarvio lopputuloksesta**

Olen saavuttamaani lopputulokseen varsin tyytyväinen. Ohjelma toimii kuten aiheen vaatimuksissa kuvattiin, ja onnistuin toteuttamaan vaikean version tehtävästä. Ohjelmalla pystyy tekemään helposti monipuolisen sisustussuunnitelman ja tallentamaan sen. Ohjelma vastaa vaatimuksia ja siinä on pientä ekstraa (kuten Tooltipit), ja se on pääosin käyttäjäystävällinen. Virheelliset syötteet käsitellään ja ohjelma ei prosessoi vääränlaista dataa. Ohjelma päivittää dialogin esimerkkikuviota reaaliaikaisesti muuttamalla sen väriä, mittasuhteita ja orientaation kulmaa, ja nämä muutokset voi hyväksyä, eli lähettää eteenpäin varsinaiselle huonekalulle tai perua. Huonekaluja voi poistaa yksi kerrallaan tai kaikki samalla kerralla. Skaalauksen voi vaihtaa, ja ohjelma näyttää reaaliaikaisesti skaalan.

Elementtien luominen pohjapiirustuksen keskelle ja kuvan lisääminen huonekaluille ja kodinkoneille ovat minusta keskeneräisimmät osat ohjelmaa, ja etenkin kuvan lisäämiseen saatan palata kesällä kun on enemmän vapaa-aikaa jatkokehittää projektia. Jos jatkaisin ohjelman kanssa painimista, luokkarakenne muuttuisi varmaan monesti monella eri tavalla mitä en nyt osaa ajatella, sillä koko projektin ajan olen ahkerasti lisännyt ja poistanut uusia luokkia projektista. Tällä hetkellä toisteisuuden vähentämiseksi olen useamman luokan ja metodin yhdistänyt, mutta

tämä on johtanut monimutkaisiin match-case -ketjuihin koodissa. Tämä monimutkaisuuden ja toisteisuuden tasapaino mietityttää, etenkin ShapeMakerissa, joten sitä saattaisin ruveta toteuttamaan toisin jos vain jonkin toisen tavan keksisin. Ohjelmaan voi kohtuullisen helposti nykyisellään lisätä erilaisia muotoja, eli uudenlaiseen huonekalumuotoon voisi mielestäni järkevästi laajentaa.

Jos aloittaisin projektin nyt alusta, tietäisin paljon paremmin mihin ScalaFX:llä pystyy ja mitä on tehtävä itse, joten en tuhlaisi aikaa jo olemassa olevien asioiden uudelleenkeksimiseen, vaan keskittyisin toimivan käyttöliittymän rakentamiseen. Lukisin myös dokumentaation paljon huolellisemmin, sillä huomasin alussa lukevani verrattain vähän JavaFX:n dokumentaatiota, kun ajattelin ettei se ole niin ymmärrettävää. Kuitenkin kun projektin keskivaiheilla rupesin lukemaan dokumentaatiota huolellisemmin, löysin paljon kaikkea mitä pystyin hyödyntämään ja projekti eteni huomattavasti nopeammin kuin vain arvaamalla.

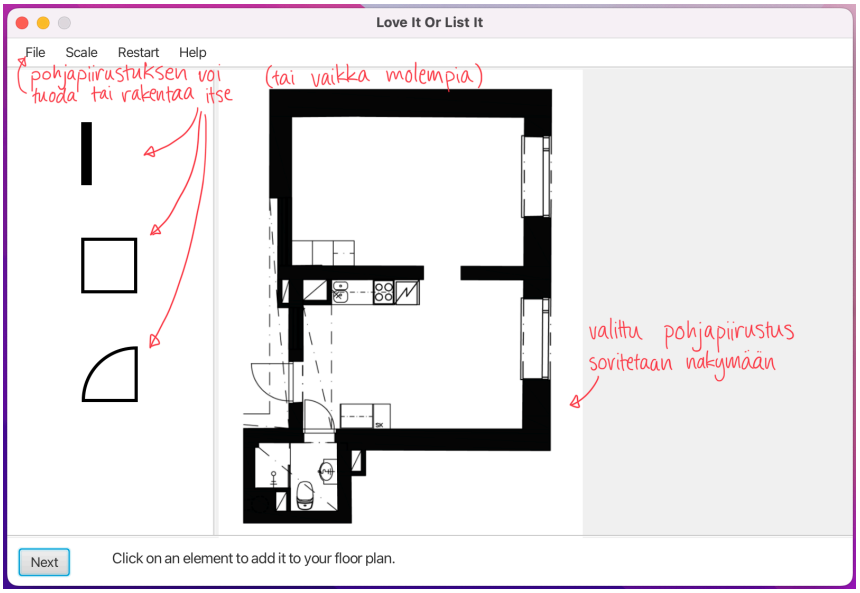
13. Viitteet ja muualta otettu

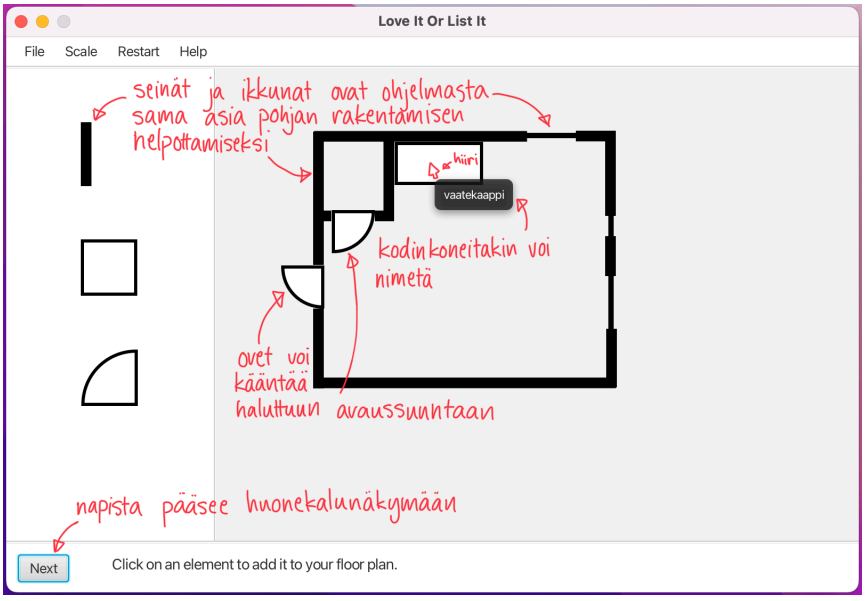
Draggable-luokan makeDraggable-metodia laatiessani katsoin mallia Eden Codingin esimerkistä[1] ja sain koodin toimivampaan muotoiluun apua Vilma Judinilta. Olen muokannut koodia omiin tarkoituksiini sopivaksi, mutta koodin pohja ei ole itse kirjoittamaani (3.-4.4. commitit Gitissä).

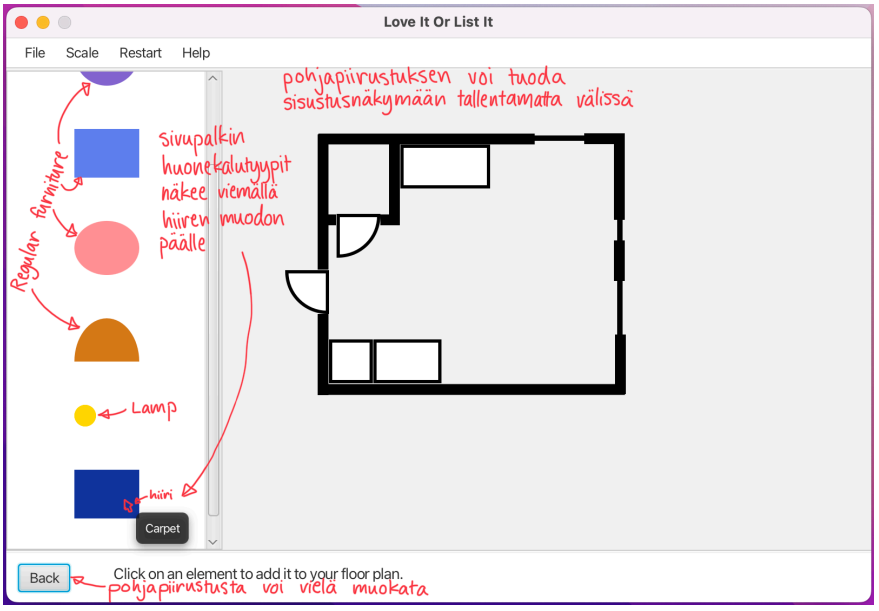
[1] Eden-Rump, Ed. “The Best Way to Drag Shapes in JavaFX”. Eden Coding. https://edencoding.com/drag-shapes-javafx/. Julkaistu 18.8.2020. Noudettu 3.4.2024

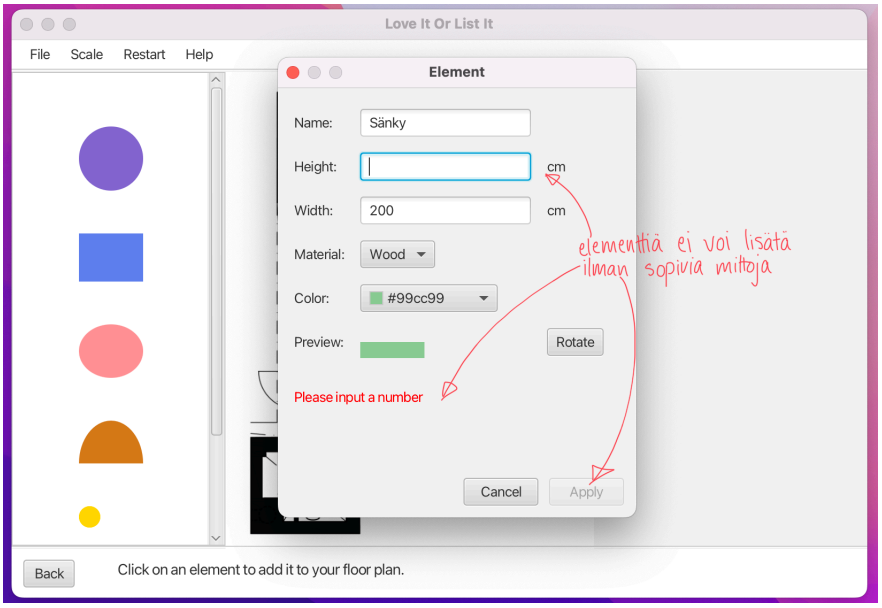
14. Liitteet

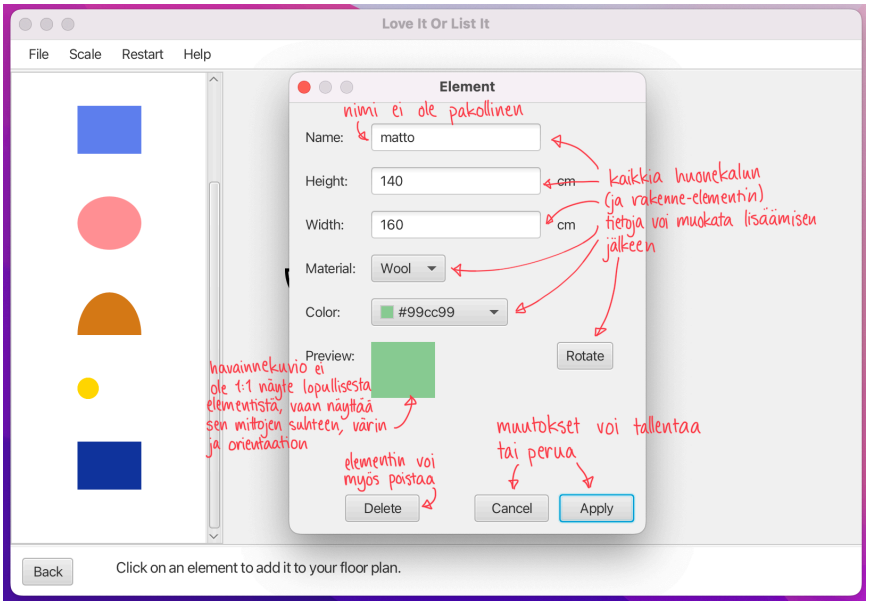
Liitteenä kuvia ja selityksiä ohjelman toiminnasta.

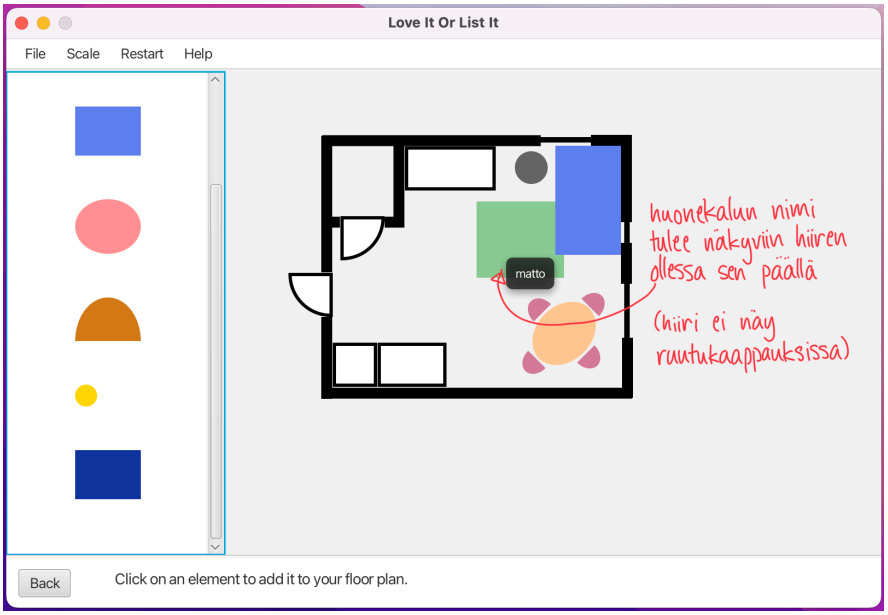


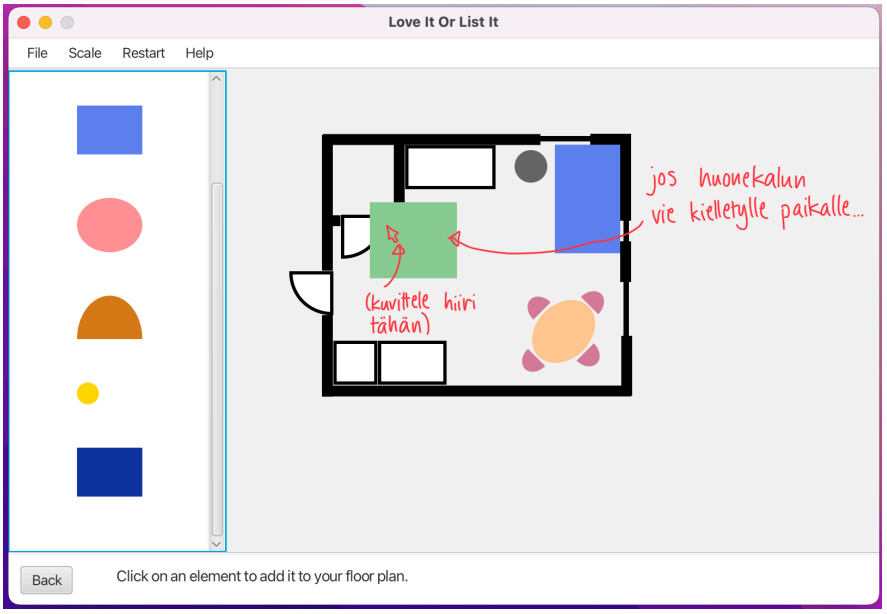


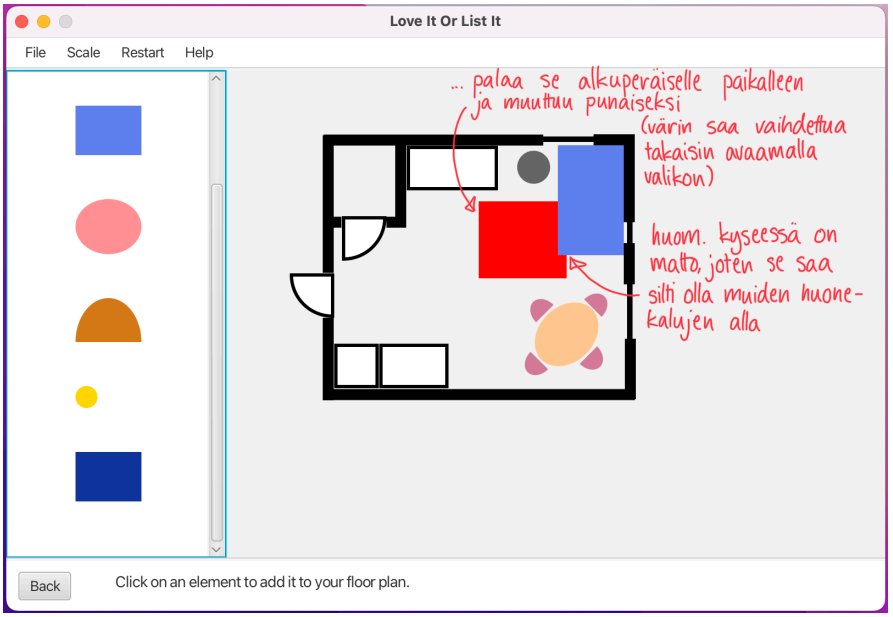

Love It Or List It
File
Scale
Restart
Help
pohjapiirustuksen voi tuoda sisustusnäkymään tallentamatta välissä
sivupalkin huonekalutyypit näkee viemällä hiiren muodon päälle
Regular furniture
Lamp
hiiri
Carpet
Back
Click on an element to add it to your floor plan.
pohjapiirustusta voi vielä muokata


Love It Or List It
File
Scale
Restart
Help
Element
Name:
Sänky
Height:
cm
Width:
200
cm
Material:
Wood
Color:
#99cc99
Preview:
Rotate
Please input a number
elementtiä ei voi lisätä ilman sopivia mittoja
Cancel
Apply
Back
Click on an element to add it to your floor plan.


Love It Or List It
File
Scale
Restart
Help
Element
nimi ei ole pakollinen
Name:
matto
Height:
140
cm
Width:
160
cm
Material:
Wool
Color:
#99cc99
Preview:
Rotate
kaikkia huonekalun (ja rakenne-elementin) tietoja voi muokata lisäämisen jälkeen
havainnekuvio ei ole 1:1 näyte lopullisesta elementistä, vaan näyttää sen mittojen suhteen, värin ja orientaation
elementin voi myös poistaa
muutokset voi tallentaa tai perua
Delete
Cancel
Apply
Back
Click on an element to add it to your floor plan.


Love It Or List It
File
Scale
Restart
Help
matto
huonekalun nimi tulee näkyviin hiiren ollessa sen päällä
(hiiri ei näy ruutukaappauksissa)
Back
Click on an element to add it to your floor plan.


Love It Or List It
File
Scale
Restart
Help
jos huonekalun vie kielletylle paikalle...
(kuvittele hiiri tähän)
Back
Click on an element to add it to your floor plan.


Love It Or List It
File
Scale
Restart
Help
... palaa se alkuperäiselle paikalleen ja muuttuu punaiseksi
(värin saa vaihdettua takaisin avaamalla valikon)
huom. kyseessä on matto, joten se saa silti olla muiden huone-kalujen alla
Back
Click on an element to add it to your floor plan.


kun tallentaa itse tehdyn pohjapiirustuk-sen, tallentuu koko suunnittelualue
reunat korostettu punaisella havainnollistamaan tallennettujen kuvien kokoeroa


kun tallentaa valmiin pohja-piirustuksen, tallentuu vain alkuperäinen kuva