Haaga-Helia
ammattikorkeakoulu Oy

Juomanäyttö - Sovelluksen suunnittelu, kehitys ja testaus

Noora-Karoliina Peltokangas

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

15.11.2019

Tekijä(t)
Noora-Karoliina peltokangas

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
“Juomanäyttö” - Sovelluksen suunnittelu, kehitys ja testaus

Sivu- ja liitesivumäärä
21 + 4

Opinnäytetyön otsikko englanniksi
DrinkDisplay - The designing, producing and testing of a software

Tämä on työnantajalta saatu toimeksianto projektinimellä “Juomanäyttö” kulkevalle, Unity:lla toteutettavalle mobiilisovellukselle.

Tässä opinnäytetyössä suunnittelen tarvekartoituksen ja vaatimusmäärittelyn avulla sovelluksen, jolla loppukäyttäjä voi selata baarin juomavalikoimaa, ja saada niistä lisätietoa.

Opinnäytetyö alkaa suunnittelusta, josta edetään loogisesti toteutukseen ja siten testaukseen samalla läpikäyden yleisiä sovelluskehityksessä käytettyjä toimintatapoja ja sitä, miten niitä hyödynnettiin tai ei.

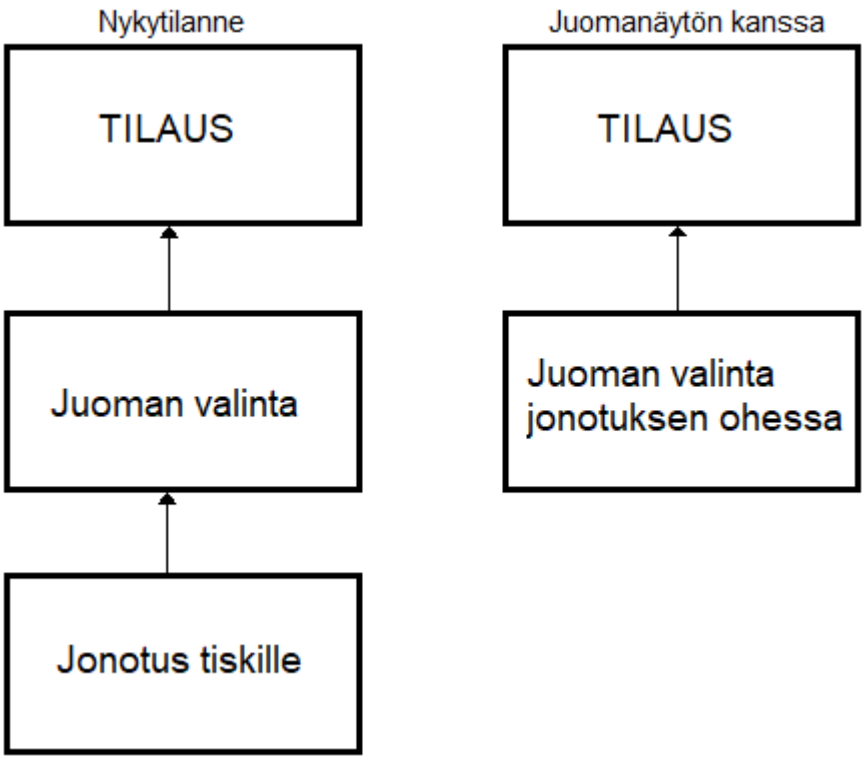
Asiasanat
Android, Mobiilisovellukset, Toimeksianto

Sisällys

1 Johdanto 1
2 Sanastoa 2
3 Opinnäytetyön rajaus 3
4 Liiketoimintatarkastelu 3
5 Vaatimusten ja tarpeiden kartoittaminen 3
5.1 Lähtötilanne 4
5.2 Tarvekartoitus 4
5.3 Vaatimukset ja toivotut ominaisuudet 5
6 Toteutus 5
6.1 Käytettävät työkalut 5
6.2 Unity 6
6.2.1 Unity Hub 7
6.2.2 Versionhallinta 8
6.3 Projektin aloitus 9
6.3.1 Scenet 9
6.3.2 UI Itemit 10
6.3.3 Skriptit 12
6.3.4 Grafiikka 14
6.3.5 Tietojen tallentaminen ja lukeminen 15
6.3.6 Applikaation viimeistely 16
7 Testaus 17
7.1 Testaustavat 18
7.2 Mitä testata 20
8 Pohdinta 20
8.1 Projektin luovutus toimeksiantajalle 20
8.2 Jatkokehitysideat 21
Lähteet 22
Liitteet 24
Liite 1. Näyttökaappauksia applikaatiosta 24
Liite 2. Raakoja suunnitelmia käyttöliittymästä 25

1 Johdanto

Toimeksiantaja Luotone Oy havaitsi puutteen ravintola-alalla. Äänenvoimakkuudeltaan usein kovin meluisissa yökerhoissa tai täydessä baarissa puheensorinan keskellä on henkilökunnan kanssa kommunikaatio vaikeaa. Useat toistuvat kysymykset ja väärinymmärrykset hidastavat palvelua ja näin aiheuttavat tiskillä ruuhkaa ajoittain. Tämä kuormittaa työntekijöitä ja näin laskee palvelun tasoa entisestään. Näin päädyttiin ideaan digitaalisesta juomalistasta, josta asiakkaat pystyvät selaamaan juomavalikoimaa, hintoja ja tarkistamaan drinkkien allergeeneja. Koska kilpailevia tuotteita oli suomessa vain yksi, nähtiin tässä suuri markkinapotentiaali ja päätettiin toteuttaa kevyt prototyyppi. Digitalisaatio on jo vallannut alaa ravintolapuolella, sillä useiden pikaruokaketjujen toimipisteessä on jo mahdollista tilata ruokaa täysin digitaalisesti ravintolasta löytyviltä laitteilta. Tämän idean implementoimisella juomien myynnin puolella saattaisi potentiaalisesti nostaa liikkeen myyntiä ja täten vahvistaa kilpailukykyä.



Kuva 1.

2 Sanastoa

Proof of Concept (PoC) = soveltuvuusselvitys

Fall back = edelliseen versioon palaaminen.

Backup = varmuuskopionti

Scene = Näkymä

Panel = Taso

User Interface (UI) = käyttöliittymä

Runtime = ajonaikaisesti tapahtuva asia

Canvas = piirtoalue

Framework = sovelluskehys

Child = Asia joka perii sijaintinsa tai toiminnallisuuksia yms. toiselta.

Integrated development environment (IDE) = integroitu kehitysympäristö

JSON = JavaScript Object Notation

Quality Assurance (QA) = Laaduntarkkailu tai laatulupaus

Front-End = sovelluksen käyttäjälle näkyvä osa

Deployment = Käyttöönotto

3 Opinnäytetyön rajaus

Projektin tähän versioon ei sisälly juomien tilausmahdollisuutta eikä etusivulla näkyviä tarjouksia, sillä nämä toteutetaan vasta kun kehitys ja asiakkaiden kiinnostus on saatu tietylle tasolle. Myöskään kovin laajaa testausta ei suoriteta, sillä käytössä olevan Unity:n omassa ympäristössä testaaminen on tarpeeksi pätevää tähän kyseiseen *PoC* versioon.

4 Liiketoimintatarkastelu

Ansaintamalliksi on suunniteltu vuokraperusteista laskutusta. Mitään tarkkaa hintaa ei ole vielä päätetty, mutta lisenssimaksuksi on kaavailtu 29-79€/kk per lisenssi. Jos ravintolalle tulee toimittaa myös tablet-laitteet, lisäävät ne kuukausilaskutusta n. 9,90-19,90€/kk. Tavallista korkeampi laitevuokrahinta on perusteltavissa suuremmalla riskillä vahingonteolle, kuin esim. kotisohvalla käytettäessä. Lisenssi- ja laitevuokraan kuuluu huolto sekä ylläpito, joka koskee applikaation päivittämistä ja kustomoimista sekä laitteiden asennusta ja vaihtoa. Kulut Luotone Oy:lle työntekijäkustannusten lisäksi ovat baaritiskille sekä pöytiin kiinnitettävien tablet-telineiden tuotantokustannukset, jotka tuotetaan alihankintana tutulta yritykseltä. Tämän toimeksiannon lopputuotteena on kevyt idean validoimiseksi soveltuva Unity:lla toteutettu applikaatio Android -käyttöjärjestelmää käyttävälle tabletille.

Asiakkaiden näkökulmasta asiointia helpotetaan ja suoraviivaistetaan. Tarjouksia tai uutuuksia ei tarvitse erikseen kysellä, vaan ne ovat nähtävissä selkeästi applikaation etusivulla, eli päävalikossa.

5 Vaatimusten ja tarpeiden kartoittaminen

Applikaatiota tai mitä tahansa ohjelmistoa kehittäessä hyvä pohjatyö on ensiarvoisen tärkeää. On tärkeää kuunnella asiakkaan ja loppukäyttäjien toiveet, sillä ilman niitä toivottuun lopputulokseen pääseminen on likipitäen mahdotonta. Vaatimusmäärittely ja tarvekartoitus ovat tapoja kerätä projekteille nk. “raamit”, joiden sisällä pysytään, jotta projekti pysyy aikataulussa ja että asiakkaalle toimitetaan vain sovitut toiminnallisuudet. Vaatimukset ovat siis asiakkaan ja toimittajan sopimia asioita, joita luovutettavassa lopputuotteessa tulisi olla. Niistä sovitaan heti projektin alussa, mutta joskus vaatimuksia saattaa tulla myös lisää kesken projektin, jolloin niistä sovitaan osapuolten välillä taas erikseen. Vaatimuksista tehdään erillinen dokumentti, jossa asiakkaan tarpeet ovat kartoitettuna, jota päivitetään aina kun projekti etenee tai jotakin muuttuu. Tämä dokumentti on hyvä pitää ajan tasalla, sillä isoissa ja monimutkaisissa

ohjelmistoprojekteissa on tärkeää pystyä pitämään kaikki langat käsissä. (Kotonya et al. 1997)

Nykymaailmassa vaatimusmäärittely on kuitenkin jäänyt vähemmälle käytölle Ketterien menetelmien suosion noustua, jolloin vanhat vesiputousmenetelmät ovat jääneet vähemmälle käytölle, mutta niiden käyttöä kyllä vielä jatketaan esimerkiksi lääketieteen ja reaaliaikaisten järjestelmien puolella. Lisäksi tilanteissa, joissa osapuolet ovat sopineet jo projektille kiinteän hinnan vaikkapa kilpailutuksen yhteydessä, on vaatimusmäärittelydokumentti hyvä kerätä heti alkuun.
Kun vaatimukset muuttuvat ja elävät projektien edetessä, on parempi nykyään edetä ketterästi ja edetä projektissa ns. “sprinteissä” jotka ovat yleensä viikosta kolmeen kestäviä työjaksoja, joiden jälkeen käydään aina läpi mitä on saavutettu ja mitä uusia toiminnallisuuksia tai vaatimuksia on mahdollisesti noussut esiin.

Tässä projektissa kuitenkin päädyttiin käyttämään yksinkertaista vaatimusmäärittelyä, sillä tässä vaiheessa ketterien sprinttien käyttöä ei koettu tarpeelliseksi, koska mahdollisesti applikaatiosta kiinnostuneilla baareilla oli vain muutamia vaatimuksia, jotka eivät tule muuttumaan. Myös haastateltujen loppukäyttäjien toiveet olivat lähes samat kuin henkilökunnalla. Kun kiinnostuneet baarit haluavat lähteä mukaan jatkokehitykseen, siirrytään agiileihin metodeihin. Tästä huolimatta toimeksiantajan kanssa kommunikointi suoritettiin ketterästi projektin aikana, jotta heidän visionsa oikeellinen toteutuminen saatiin varmistettua.

5.1 Lähtötilanne

Nykytilanteessa monissa baareissa on tiskillä vain ylhäällä katonrajassa tai seinillä listat juomista ja lisäksi vielä paperiset listat, jotka usein illan mittaan tahmaantuvat kaatuneilla juomilla tai häviävät. Usein asiakkaat myös kyselevät juomista yksityiskohtia kuten makua, suutuntumaa, aineksia, alkoholiprosentteja yms. Tiskin henkilökunnan on vaikea keskittyä juomien valmistamiseen ja asiakkaiden kysymyksiin vastaamiseen samanaikaisesti, jolloin heidän täytyy keskeyttää tekemisensä kuunnellakseen asiakkaan kysymykset.

5.2 Tarvekartoitus

Alustavaa tarvekartoitusta lähdettiin suorittamaan projektista kiinnostuneista baareista ja yökerhoista. Omistajia ja baarityöntekijöitä haastateltiin lyhyesti, jonka aikana ilmeni seuraavat asiat. Omistajat toivoivat applikaatiolta selkeyttä, helppokäyttöisyyttä ja minimaalista huoltoa. Applikaation tulisi helpottaa baaritiskien ruuhkaa ja “turhaa” baarityöntekijän häirintää, eikä olla tiellä tai vaivaksi. Toiveena oli myös VIP-tilojen

asiakkaille mahdollisuus tilata applikaatiolla juomia pöytiin, tämä toteutetaan myöhemmässä vaiheessa, eikä näin sisälly tähän toimeksiantoon.

Useissa baareissa asiakkaalle myytävään “kokemukseen” kuuluu se, kun asiantunteva baarin työntekijä kertoo eri juomista ja osaa suositella asiakkaalle juomia, mutta haastatelluissa baareissa juomapuoli on paljon suoraviivaisempi ja yksinkertaisempi, joten he kokisivat helpommaksi, että asiakkaat voisivat tabletilta lukea lisätietoja juomista, sillä näissä paikoissa asiakasmäärä saattaa olla paikoin suuri ja tiskillä paljon ruuhkaa.

5.3 Vaatimukset ja toivotut ominaisuudet

Laitteelta pitää pystyä selaamaan baarin valikoimaa kategorioittain. Kategoriajaottelu olisi karkeasti seuraavanlainen: hanajuomat, joihin kuuluu siidereitä, oluita ja lonkeroita, sekä pulloissa tai lasittain myytävät juomat, kuten mahdolliset erikoisoluet- ja siiderit ja viinit. Huolena on laitteiden likaantuminen, rikkoutuminen tai varastaminen. Toimeksiantajan ehdotuksena tähän oli metalliset jalustat laitteille, joita Luotone Oy voi tilata ja vuokrata/myydä heille laitteen ja applikaation ohella.

6 Toteutus

Tässä kappaleessa syvennytään projektissa käytettyihin työkaluihin ja teknologioihin, sekä siihen, kuinka toimeksiannon lopputuote rakennetaan pala palalta Unity:lla ja mitä kaikkea niihin vaiheisiin sisältyy.

6.1 Käytettävät työkalut

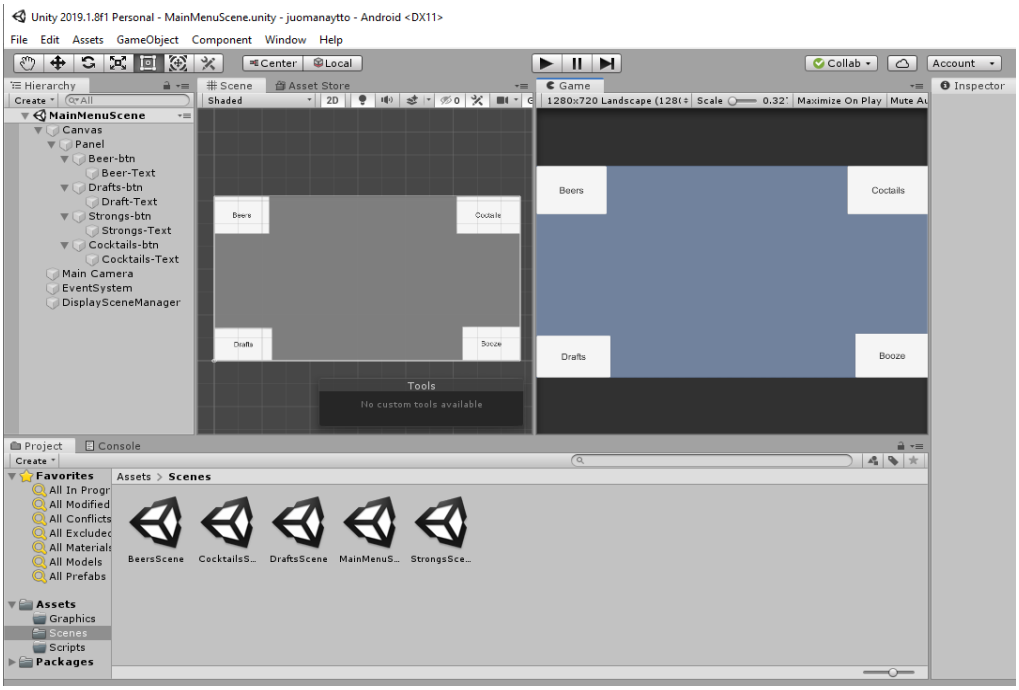
Tässä kappaleessa kerrotaan käytetyistä työkaluista ja toimintamalleista. Tässä toimeksiannossa hyödynnetään Luotone Oy:n kanssa ketteriä menetelmiä, jotta kommunikaatio olisi vaivatonta ja sulavaa. Ohjelmoinnin osalta käytetään Clean Code:n prinsiippejä luettavan ja ymmärrettävän lähdekoodin takaamiseksi. Yksi tärkeimmistä asioista on muuttujien, skriptien ja näkymien selkeä ja konsistentti nimeäminen. Toinen on koodin kommentoiminen tarvittaessa. Ihannetilanteessa koodia ei tarvitsisi kommentoida lainkaan, sillä koodia lukemalla tulisi olla ilmiselvää, mitä siinä tapahtuu missäkin kohtaa, mutta tämä on harvoin mahdollista sillä projektien ohjelmoijat saattavat vaihtua tiuhaankin tahtiin lennosta, jolloin toisen koodia ilman kommentteja ja taustatietoa on hankalaa tulkita. Tämä johtaa usein ohjelmoijan kuormittumiseen ja työaikaa kuluu turhaan, kun soitellaan edellisen ohjelmoijan perään. (Martin, 2015)
Tässä projektissa tosin oli vain yksi ohjelmoija, joten edellä mainittu ei ollut ongelma.

6.2 Unity

Unity on Unity Technologiesin kehittämä monialustainen pelimoottori ja visuaalinen pelinkehitysympäristö, jota käytetään 2- ja 3D pelien kehitykseen. Ensimmäisen kerran se on julkaistu kesällä 2005 ja nykyinen uusin versio Unity 5 on julkaistu vuonna 2015. Unityn ensimmäisessä versiossa tuettiin mm. Java, JavaScript ja Boo ohjelmointikieliä, joiden tuki sittemmin deprekoitiin kun Unity 5 julkaistiin. Nykyään C# on standardi kieli Unityä käytettäessä. Unity oli alkujaan suunniteltu vain Mac laitteille, mutta jo muutaman kuukauden kuluttua siitä tehtiin portattu versio Windows käyttöjärjestelmää käyttäville koneille. Kohdealustoja on myös versioiden ja päivitysten myötä lisätty, ensimmäinen kohdealusta Mac-koneiden lisäksi oli *cross-platform web player* vuonna 2006, iPhone vuonna 2008 ja Android vuonna 2010. Ja nykyään listasta löytyy myös tunnetut konsolit kuten PlayStation ja Xbox. Uusin lisäys kohdealustojen kasvavaan listaan on WebGL, joka on uusin selaimissa käytetty 3D *framework.* (Hocking, 2015)

Unity:n käyttämisen helppous johtuu sen tavasta muodostaa asiat komponenteista, pienistä sekalaisista paketeista, ja objektit taasen ovat näiden komponenttien summia. Kirjassa *Unity in Acton*, kirjoittaja vertaakin komponenttijärjestelmää olio-ohjelmointiin. Unity tarjoaa myös laajan kustomointimahdollisuuden, joka tarkoittaa sitä, että käyttäjä voi kirjoittaa omia skriptejä, joilla luodaan lisää toiminnallisuuksia editoriin ja sen käyttöliittymään. Tässä projektissa käytettiin toiminnallisuuksia, jotka löytyvät Unitystä oletuksena. Sisäänrakennetun emulaattorin ansiosta pelejä voi testata paljon intuitiivisemmin ja voit muokata asioita samalla kun emulaattori pyörittää peliä (Hocking, 2015). Toimeksiantajan CTO:n ohjeistus oli tästä poiketen kuitenkin, ettei asioita tulisi muokata samalla kun emulaattori pyörittää ja rakentaa applikaatiota. Tällä saattaa saada aikaan muutoksia, joita on erittäin vaikea korjata muuten kuin palaamalla versionhallinnassa edeltävään versioon.

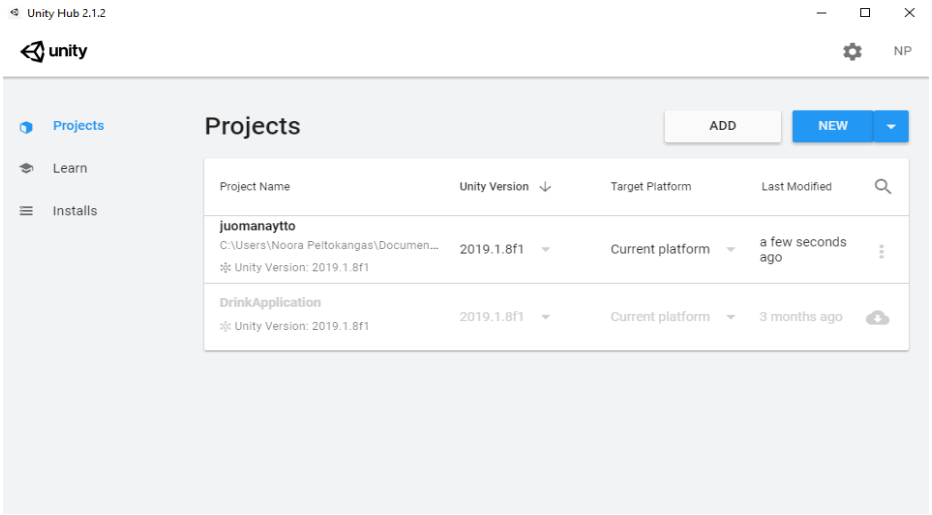
Unity on ensisijaisesti rakennettu pelien kehittämiseen, siksi Unityn käyttäjien keskuudessa näyttäisikin olevan vahvahko mielipide siitä, ettei Unityä kannata käyttää muiden kuin pelien kehitykseen. Tutkimustyötä tehtäessä ei löytynyt paljoakaan tietoa Unity:lla tehdyistä applikaatioista, jotka eivät ole pelejä. Mutta 2D komponentteja ja UI itemejä hyödyntäen voidaan Unityllä rakentaa mobiiliapplikaatioita samaan tyyliin kuin Android Studio:lla. Unityn käyttö on Luotone Oy:lla vakio prototyyppejä tehdessä. Saadaan tehtyä applikaatioita kaikille asiakkaille relevanteille alustoille ja nopeasti. Varjopuoli Unityn käytössä saattaa olla kuitenkin se, että jos ei tiedä mitä tekee, saattaa saada aikaan erittäin raskaita applikaatioita, jotka vievät paljon tehoa käytettävästä laitteesta.



Kuva 2.

6.2.1 Unity Hub

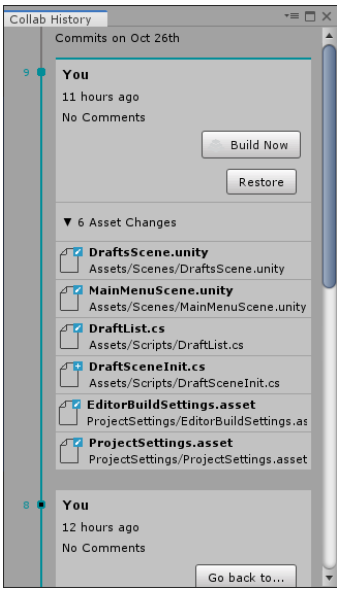
Unity Hub on itsenäinen ohjelma, jolla käyttäjä voi hallita Unityn versioita ja projektejaan ja käynnistää niitä sitä kautta, se on ikään kuin kojelauta tai laukaisualusta Unitylle. Version valinnassa on hyvä lukea sen version tiedot läpi; mitä ominaisuuksia siinä on, ja mitä kaikkea siitä puuttuu esimerkiksi uudempiin versioihin verrattuna. Projektia luotaessa päätetään, luodaanko 2- vai 3D sovellus. Tätä et voi muuttaa kesken projektin. Tämä kyseinen applikaatio toteutetaan 2D:nä. Samalla applikaatio nimetään, tässä pitää muistaa Clean Code:n nimeämiseen liittyvät asiat: selkeä ja kuvaava. Tässä tapauksessa käytetään projektitiimin keskuudessa vakiintunutta “Juomanäyttö” nimeä, joka kuvaa hyvin sitä mitä applikaatio tekee ja taipuu helpommin loppukäyttäjien puheeseen kuin esimerkiksi “Digital Drinking Display”.



Kuva 3.

6.2.2 Versionhallinta

Versionhallinta on yksi ohjelmistotuotannon kulmakivistä. Se on yksinkertaisesti työskentelytapa, jossa lähdekoodista otetaan varmuuskopiot jokaisen muutoksen jälkeen, joka mahdollistaa saumattoman vaihtelun versioiden välillä mikäli jotain ei-toivottua tapahtuu uusimmassa versiossa. Nykypäivänä tähän on olemassa muutamia eri palveluja, kuten GitHub, BitBucket ja muut vastaavat. Nämä versionhallinnan työkalut muodostavat versioista puumallisen rakenteen, jossa oksat ovat uusia muutoksia tai variaatioita kehityssuunnasta. Nämä oksat voidaan sitten yhdistää osaksi tämän puurakenteen runkoa eli “masteria”.



Kuva 4.

Unitylla on käytössään oma versionsa tästä, joka kulkee nimellä “UnityTeams”. Tämä on sisäänrakennettu ominaisuus Unityn editorissa, joka edellyttää Unity-tilin luomista. Kuvassa 4. on versionhallinnan historianäkymä. Tässä nähdään viimeisimmän muutoksen

tiedot ja sen kuka tämän muutoksen on tiimistä tehnyt. Tämä noudattaa täysin samaa kaavaa kuin muutkin versionhallintatyökalut, tiimin jäsenet pysyvät ajan tasalla toistensa tekemistä muutoksista eikä päällekkäisiä muutoksia näin pääse tapahtumaan. Tässä työssä tätä käytettiin vain *backup* tarkoituksiin, sillä projektissa ohjelmoijia oli vain yksi.

6.3 Projektin aloitus

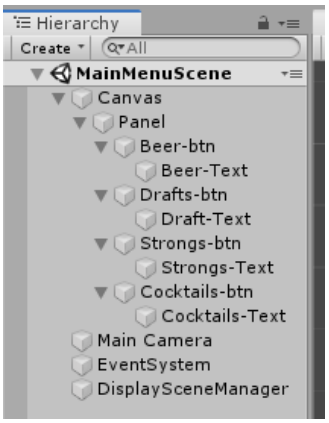
Tässä kappaleessa käydään läpi itse toimeksiannon toteuttaminen vaihe vaiheelta, syventyen samalla asioiden teoreettisempaan puoleen ja siihen, miten asiat toimivat Unityssa.

6.3.1 Scenet

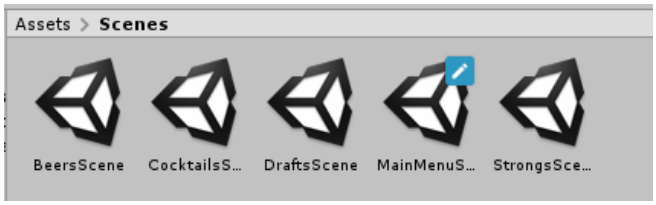
Scene, on applikaation tai pelin sillä hetkellä näytöllä näkyvä osa tai ympäristö. Pelikehityksessä usein *scene* = *level* eli pelin yksi taso. Scene sisältää koordinaatiston jolle komponentit sijoitetaan. Koska Unity on tarkoitettu alun perin vain 3D-pelien kehitykseen, on koordinaatistokin kolmiulotteinen eli X-, Y- ja Z-akselit ovat käytössä. (Thorn, 2013)
Vielä vuonna 2013 2D-pelien kehityksessä komponentit ja UI itemit vain konfiguroitiin toimimaan 2D:nä, mutta nykyään näin ei tarvitse enää tehdä vaan Unity disabloi kolmannen akselin, kun kyseessä on 2D-peli.

Juomanäyttöön tulee viisi sceneä, jotka tulevat olemaan päävalikko eli menu, hanajuomat, oluet, drinkit ja väkevät. Jokaiseen näkymään tulee omat UI komponentit, joista lisää seuraavassa osiossa. Visuaaliset materiaalit ja inspiraatiot on saatu Luotone Oy:n designerilta.

“Kaikki scenen sisällä olevat asiat ovat peliobjekteja”. Näin summaa Learn Unity For 2D Game Development -kirjan tekijä Alan Thorn. Peliobjektin muodostavat komponentit, jotka ovat fundamentaalinen osa scenen rakentamisessa. Ilman komponentteja ei ole peliobjekteja. Komponentit voivat käsitellä useita tapahtumia objektin toiminnoista riippuen. (Thorn, 2013)



Kuva 5. hierarkia näkymä, jossa nähdään sillä hetkellä muokattavan scenen komponentit ja objektit.



Kuva 6. Scene valikko

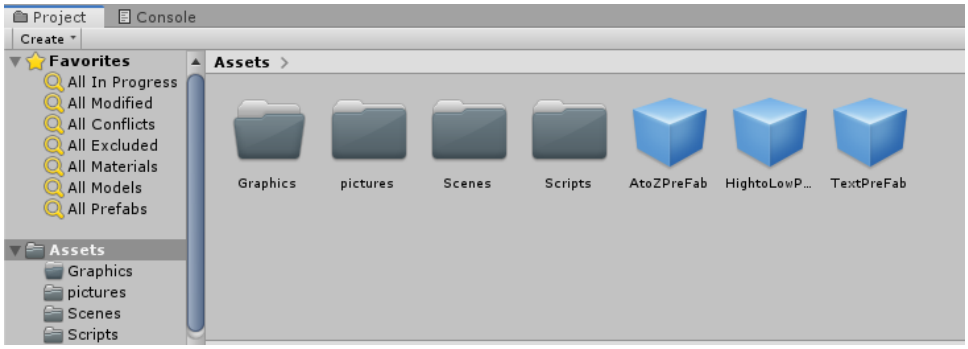
6.3.2 UI Itemit

UI itemit ovat visuaalisia komponentteja. Näitä ovat käyttöliittymissä usein nähtävät painikkeet, vierityspalkit, paneelit jne. Scenen pohjimmainen UI osa on *canvas*, joka on itsessään näkymätön komponentti, joka toimii ikään kuin pohjana kaikelle näkyvälle. Unityssa ohjelman hierarkiassa päällimmäisenä näkyvä on itseasiassa alimmaisena, joten canvasin sisään tulee ensimmäinen panel joka on näkyvä taso esimerkiksi taustakuvaa varten. Jokaisen UI itemin on oltava jollain tapaa canvasin child, muuten kyseiset UI itemit eivät näy oikein. (Thorn, 2013)
Tämän sisään tulee taas lisää tarvittavia komponentteja, kunnes kyseinen scene on saatu valmiiksi. Saman panelin sisään ei kuitenkaan kannata laittaa liikaa komponentteja, vaan on hyvä miettiä, että tarvitseeko niille luoda oma objektinsa. Liiat komponentit tekevät hierarkianäkymän tulkinnasta hankalaa ja saattaa tehdä applikaatiosta raskaan pyörittää.

Juomanäytön päävalikko sceneen, tulee neljä painiketta ja tekstiobjekti otsikoksi. Painikkeita painamalla käyttäjä pääsee selaamaan neljän eri kategorian juomia. Kaikkiin muihin sceneihin tulee kolme nappia, jotka ovat takaisinpaluu ja lajitteluun tarkoitetut nappulat “*A to Z*” ja “*Low to High*”, vierityspalkki ja neljä tekstiobjektia, joista yksi on taas otsikko. Kolme muuta ovat *prefab* objekteja, joihin tulostuu valikoima.
Prefabit ovat objekteja, joita voit “monistaa”. Peleissä on usein tarve luoda monta samanlaista objektia samaan tilaan, kuten ympäristön puut, NPC:t ja viholliset. Tässä tapauksessa haluttiin tekstiobjekti, joka hakee listasta objektin ja tulostaa sen attribuutit näkyville. Mistä tahansa objektista voidaan luoda prefab, ja se näkyy sinisenä kuutiona

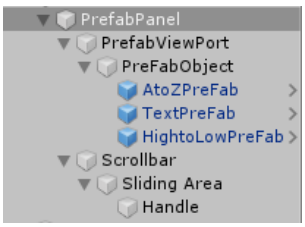
Asset näkymässä. Kun objektista luodaan prefab, luotu prefab ei ole objekti vaan se toimii enemmänkin assettina, jota voidaan sitten käyttää missä scenessä tahansa. (Hocking, 2015)

Asset on mikä tahansa tiedosto, kuten scripti, scene tai kuva, joka näkyy Project-näkymässä käyttöliittymän alareunassa, samassa kohdassa Scene-näkymän ohella. (Hocking, 2015)



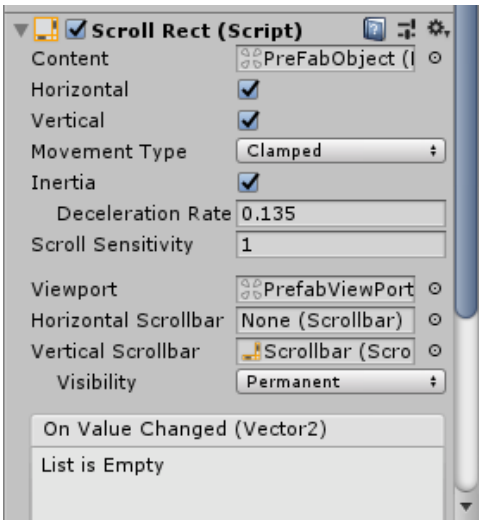
Kuva 7.

Aluksi luotiin kaikki tarvittavat scenet aloittaen päävalikosta. Kun kaikki scenet oli luotu, aloitettiin niiden täyttäminen. Tähän ei ole yhtä ja oikeaa tapaa, vaan jokainen ohjelmoija voi päättää itse, missä järjestyksessä etenee. Seuraavaksi luotiin painikkeet ja tekstiobjektit ja määriteltiin niille omat paikkansa sijoittamalla ne x-y-koordinaatistolle. Vierityspalkki on hieman mutkikkaampi ja vaatii vähän työtä. Ensimmäisenä tarvitaan ViewPort, jona toimii panel, jonka lapsiksi laitetaan objektit, joita tahdotaan scrollatessa liikuttaa. Tässä tapauksessa ne ovat luomamme prefabit.



Kuva 8.

ViewPortin lapsena oleva PreFabObject taas sisältää kaikki kolme prefabia. Vierityspalkki löytyy valmiina UI iteminä siinä missä painikkeetkin. Näille löytyy omat asetuksensa Inspectorissa, joka löytyy käyttöliittymän oikeasta laidasta (*Kuva 2. sivulla 7*).



Kuva 9.

PrefabPaneliin liitetään Inspectorin ”add component” kohdassa Scroll Rect, joka on yksi unityn valmiista scripteistä. Tässä siis määritetään mitä sisältöä scrollataan ja millä vierityspalkilla. Vasta kun kaikki tarvittavat itemit on luotu, siirrytään scripteihin.

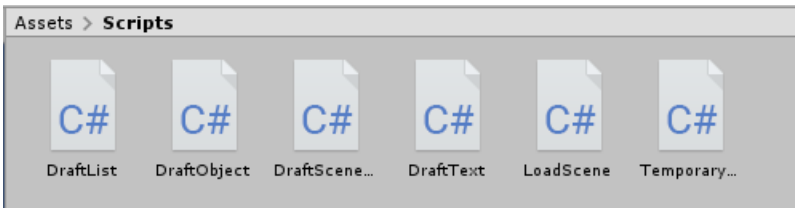
6.3.3 Skriptit

Skripti on yksinkertaisesti selitettynä itsenäinen koodi, joka tulkataan ajonaikaisessa ympäristössä. Toisin kuin käännetyillä kielillä, “*compiled languages”,* kirjoitetut ohjelmat, jotka ennen ohjelman suorittamista käännetään tietylle kohde kielelle ja suoritetaan vasta sitten tai tallennetaan myöhempää käyttöä varten. (Gorman, 2016)

Tässä projektissa ja Unity 5:ssä yleensäkin käytetty kieli on C#, joka on Microsoftin kehittämä ohjelmointikieli. Se luotiin yhdistämällä C++ ja Visual Basic. Tämä .NET-alustalle tarkoitettu vuonna 2000 kehitetty kieli yhdistää C-ohjelmointikielen syntaksin, olio-ohjelmoinnin ja C++:n tehokkuuden. C-ohjelmoinnista tutut pointterit ovat myös käytössä kyseisessä kielessä. (Hejlsberg et al. 2006)

Skriptejä kirjoittaessa käytetään tässä projektissa englanninkielisiä muuttujien nimiä, sillä suomenkieliset muuttujien nimet ovat turhan pitkiä ja vaivalloisia lukea, kun editori ja Unityn omat sisäänrakennetut toiminnallisuudet ovat englanniksi. Tämä tekee skriptien lukemisesta ja ymmärtämisestä helppoa ja vähentää ongelmia jatkokehityksessä. Editorina käytetään Visual Studio Codea, joka on Microsoftin kehittämä avointa lähdekoodia käyttävä editori.

Juomanäytössä käytetyt skriptit ovat suurimmaksi osaksi yksinkertaisia. Scenejen vaihtamiseen käytetyt LoadScene scriptit ja valikoiman tulostamiseen käytettyt scriptit joita on yksi kullekin kategorialle.



Kuva 10.

DraftList scriptissä alustetaan ja luodaan hanajuomille tarkoitettu objektilista. DraftObjektissa taas luodaan DraftListin käyttämä objekti, jolla on kolme attribuuttia: nimi, tyyppi ja hinta. Tämä osa muistutti kovin paljon Javalla kirjoitettua objektiluokkaa.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine.SceneManagement;
4  using UnityEngine;
5
6  public class LoadScene : MonoBehaviour
7  {
8      public void LoadScenes(string scene) {
9          SceneManager.LoadScene(scene);
10     }
11 }
12
```
Kuva 11. Scenen vaihtamiseen käytetty LoadScene

Kategorioiden tulostamiseen käytetty script oli tämän projektin haastavin osuus ohjelmoinnin osalta. Hanajuomien kategoriassa tämä script on nimetty DraftText:ksi. Tässä scriptissä alustetaan tulostukseen tarvittavat prefabit ja asetetaan niille paikat transformia käyttäen. Tämä script sisältää *for loop:*in joka tulostaa sen kategorian juomalistan. Tämän jälkeen tulee kaksi *else if* - lauseketta, yksi kutakin sorttausmetodia kohden.

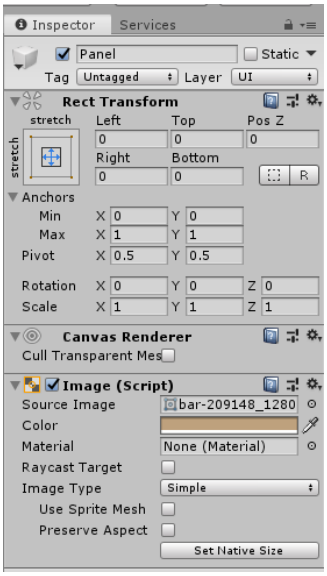
```
else if (TemporarySaves.sortOrder == "a-to-z") {
      for (int x = 0; x < draftDrinks.Count; x++) {
            Instantiate(AtoZPreFab, PreFabObject);
            AtoZPreFab.transform.SetParent(null);
            AtoZPreFab.transform.SetParent(PreFabObject);
            draftDrinks.Sort((c, d) => c.draftName.CompareTo(d.draftName));
            AtoZPreFab.GetComponent<Text>().text =  draftDrinks[x].draftName + "    " + draftDrinks[x].draftType + "   " +
draftDrinks[x].draftPrice + "€";
      }
}
```

Yllä olevassa snipetissä on aakkosittain sorttaamisen *else if.* TemporarySaves on apu scripti joka tallentaa itseensä listan sen hetkisen tilan. Tässä myös ns. ”käynnistetään” tarvittavat prefabit käyttämällä Instantiate komentoa. Compare.to funktio on C# kielen lajittelu- eli *sort* -metodi kuten esimerkiksi JavaScriptin .sort. Sulkuihin tulevia itemeitä verrataan toisiinsa, jonka perusteella lista sitten järjestetään ja tulostetaan.

C#:n syntaksissa on muutamia omalaatuisuuksia. Kaksoispiste tarkoittaa C#:n kontekstissa periytyvyyttä, eli kuvan n. *LoadScene* perii *monoBehaviour*:n ominaisuudet. Yllä olevassa C# koodin esimerkissä ensimmäisen rivin == vertaa *TemporarySaves*:ia ja *a-to-z*:tä. *GetComponentin* <Text> on parametri, joka viittaa siihen, että haettava komponentti on nimenomaan tekstiobjekti.

6.3.4 Grafiikka

Grafiikan ja taustakuvien lisäys tapahtuu luomalla asset-näkymään grafiikoille ja kuville dedikodut kansiot. Kun kaikki tarvittava on lisätty näihin, voidaan niitä nyt lisäillä käyttöliittymään. Taustakuva lisätään scenen ensimmäiseen panel objektiin, jolloin se näkyy ”pohjimmaisena” ja käyttäytyy toivotusti taustakuvan tavoin. Tämä lisäys tapahtuu Inspector-näkymässä, kun haluttu panel on valittuna.



Kuva 12.

Hiirellä raahataan valittu kuva ”Source image” kohtaan, jolloin taustakuva asettuu paikalleen. Kuvassa 12. näkyy myös työkalu, jolla kaikki käyttöliittymän komponentit siirretään omille paikoilleen koordinaatistossa. Tämä sama metodi toimii myös painikkeiden grafiikan lisäämiseen.

6.3.5 Tietojen tallentaminen ja lukeminen

Tässä projektissa pääasiana oli “prototyyppi, jolla pystyy selaamaan baarin tarjontaa”. Applikaation pitäisi siis saada dataa ja näyttää se käyttäjälle. Tämän toteuttamiseen on muutamia eri tapoja, joita käydään tässä lävitse.

Objectlist, on nimensä mukaisesti lista objekteja, joilla on *n* attribuuttia, tässä tapauksessa nimi ja hinta, jos kyseessä on tavallinen hanajuoma. Jos kyseessä on pulloittain myytävät erikoisjuomat niin attribuutteina tulisi olla nimi, tyyli, eli onko kyseessä siideri vai olut ja jos olut, oluen tyylilaji ja hinta. Drinkeissä attribuutteja on eniten, kun edellä mainittujen lisäksi mukaan tulee ainesosaluettelo ja mahdollisten allergeenien erityismaininnat. Tämä on kuitenkin erittäin huono ratkaisu ja toimii vain tässä *proof of concept* prototyypissä, sillä näin saadaan mahdollisille asiakkaille näytettyä applikaation toimintamalli vähällä vaivalla. Ongelmaksi nousee se, että tällaista listaa on vaikeaa ylläpitää pitkällä aikavälillä, varsinkin jos baarin valikoima on suuri. Valikoiman päivittäminen vaatisi myös sen, että huolto tapahtuu siellä missä laitekin on eikä päivitys etänä onnistuisi.

Seuraavana vaihtoehtona on JSON, joka on usein käytetty ratkaisu, kun halutaan kevyt data Interchange formaatti, joka on tekstiformaatti ja hyvin ihmisystävällistä luettavaa ja kirjoitettavaa. JSON on täysin kieliriippumaton, mutta siitä löytyy yhteneväisiä tapoja mm. C-kieliperheen (C, C++ ja C#) kanssa. Unity 5.ssä on *JSON serialization* toiminto, jolla objekteja voidaan muuttaa JSONiksi ja toisinpäin. (JSON)

```
[
 {
   "name": "Sommersby",
   "flavor" : "päärynä",
   "alcPercent" : "4,5",
   "price" : "8,50",
   "placeOfOrigin" : "Tanska"
 },
 {
   "name": "Crowmoor",
   "flavor" : "omena, kuiva",
   "alcPercent" : "4,5",
   "price" : "7,40",
   "placeOfOrigin" : "Suomi"
 }
]
```

Yllä on esimerkki JSON:n syntaksista ja siitä, kuinka luettavaa ja ymmärrettävää se ihmiselle on. Yksi objekti on { } -sulkeiden sisässä ja sinne listataan kyseisen objektin

atribuutit. Objekteja voi olla JSON tiedostossa niin paljon kuin on tarpeen ja järkevää käsitellä. Objektit listataan peräjälkeen pilkuilla eroteltuna. Kaikki nämä objektit tulevat hakasulkeiden sisää.

Sqlite on relaatiotietokanta, jossa SQL:n tapaan on *table*-rakenne. Relaatiotietokannalla tarkoitetaan lyhyesti tietokantaa, joka tukee relaatiomallin mukaista dataa. Sqlite tarjoaa vähemmän datatyyppejä ja toiminnallisuuksia kuin SQL ja siksi onkin erittäin suosittu mobiilikehityksessä. Mobiiliapplikaatioissa harvemmin käytetään niin moninaista dataa, että se vaatisi sqlitestä puuttuvia toimintoja. (Rizwan, 2018)

```
CREATE TABLE DraftDrinks (
  Id Int,
  Name Varchar,
  Type Varchar,
  Price Numeric
);

INSERT INTO DraftDrinks VALUES(1, 'Original', 'lonkero', 6.50);
```

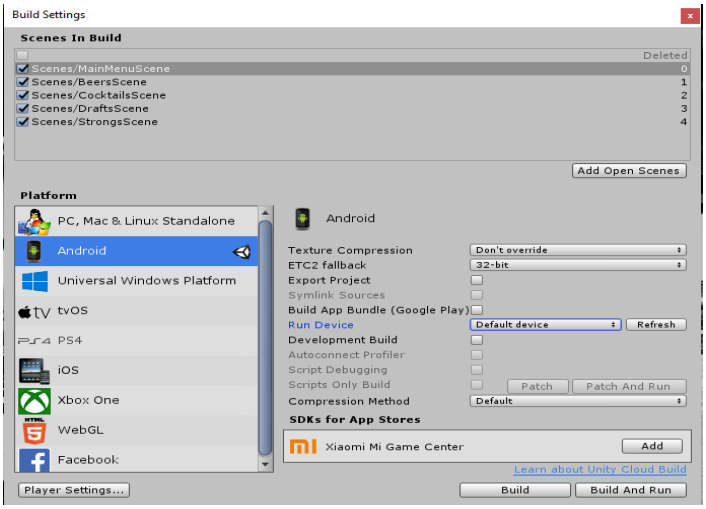
Sqliten syntaksi on identtistä SQL:n kanssa. Table luodaan komennolla *create table* ja sulkeiden sisään tulee attribuutit pilkulla eroteltuna, jolloin myös data tyyppi julistetaan.

6.3.6 Applikaation viimeistely

Koska Juomanäytön loppukäyttäjistä suurin osa tulee olemaan suomea puhuvia tai sitä muuten osaavia, tulee käyttöliittymän tekstit kääntää suomeksi. Parempaa olisi, jos heti alkuun käyttöliittymän tekstit olisivat suomeksi, tämä englannin kielen käyttö on vain ohjelmoijan henkilökohtainen preferenssi.

Unitystä löytyy ”build settings” eli asetukset, jossa asetetaan alusta tai alustat, jolla applikaation tulee toimia. Tässä projektissa alustaksi valitaan Android-käyttöjärjestelmä. Jos applikaatiosta toivotaan pelikaupasta ladattavaa, saa senkin asetettua tässä valitsemalla “Build app bundle”. Tämän alla valitaan laite, jolla ohjelmaa halutaan pyörittää. Tässä tapauksessa jätetään se vielä oletusasetukseksi, joka tarkoittaa Unityn omaa ympäristöä. Tätä voidaan vaihtaa myöhemmin, kun aloitetaan testaus kohdelaitteella. Näissä asetuksissa valitaan myös ne näkymät, jotka halutaan mukaan senhetkiseen versioon. Scenejen numerointi alkaa nollasta, ykkösen sijaan, kuten monesti on tapana ohjelmoinnissa.

Kun asetukset ovat senhetkiseen versioon sopivat, voi joko vain ”rakentaa” applikaation, joka tallentuu koneelle .apk tiedostona tai rakentaa ja suorittaa sen, jolloin emulaattori käynnistyy.



Kuva 13.

7 Testaus

Testaus on tärkeä osa ohjelmistokehitystä. Applikaatioiden ja ohjelmistojen kunnollinen testaaminen ennen tuotantoa säästää aikaa ja vaivaa. Kiirehdityt julkaisut johtavat loppukäyttäjien ongelmiin ja turhautumiseen. Pahimmassa tapauksessa applikaation kehittäjä menettää asiakkaitaan, kun ohjelmat eivät toimi halutuilla tavoilla.
On olemassa muutamia eri tapoja lähteä toteuttamaan testausta, niistä yksi on Test Driven Development (TDD). TDD:ssä luodaan ensin testit, jotka ohjelman on läpäistävä ja vasta sitten lähdetään ohjelmoimaan itse ohjelmaa. Näillä usein testataan toimivuutta ja integraatioita. Toinen tapa on Behaviour Driven Development (BDD) joka on myös *test first* tekniikka, mutta testit tehdään enemmän loppukäyttäjän näkökulmasta. (Jithin, 2018)

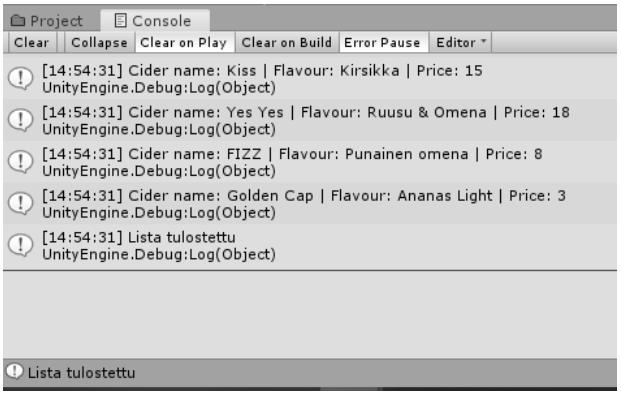
Feature driven development (FDD) ei ehkä itsessään ole tapa testata, vaan pikemminkin tapa rakentaa sovellus kuten TDD ja BDD. FDD:ssä toiminnallisuuksia käsitellään samalla tavalla kuin esimerkiksi *scrum*:ssa käyttäjätarinoita. FDD:ssä usein käytetään viiden askeleen mallia (*Five process steps*). Siinä määritellään viidessä osassa, miten sovelluksen rakentamisessa tulisi edetä. Se alkaa mallin rakentamisesta, jossa asetetaan raamit mm. arkkitehtuurille. Toisessa kohdassa luodaan lista toiminnallisuuksista, jossa koko projekti jaetaan eri ”toiminnallisuuksiin”. Kolmannessa vaiheessa katsotaan, että mistä toiminnallisuudesta sovelluksen rakentaminen aloitetaan. Yleensä tässä kohdassa nämä toiminnallisuudet jaettaisiin eri tiimeille, mutta tässä projektissa oli vain yksi ohjelmoija, joten se ei ole tarpeen. Neljännessä vaiheessa suunnitellaan toiminnallisuuksia hieman syvällisemmin, rakennetaan luokkia ja suunnitellaan visuaalista

ilmettä. Viimeisessä vaiheessa aletaan kehittää itse koodia ja Unit testata sitä. Tässä vaiheessa edetään noin kahden viikon ”design by feature, build by feature” sprintteinä ja näissä keskitytään niihin toiminnallisuuksiin, jotka ovat asiakkaan mielestä tärkeimpiä. (Garg, 2017)
Tämä kehitystapa on ehkä lähimpänä kuvaamaan projektin kehitystyötä, sillä Juomanäyttöä lähdettiin kehittämään juurikin toiminnallisuudet edellä.

Quality Assurance, tunnetaan myös nimellä QA Testing, on enemmänkin erilainen tapa kehittää ohjelmistoja, kuin pelkkä tapa testata. QA testauksessa määritellään ohjelman kehityksen kaari siten, että saadaan tarjottua laadukas ja toimiva lopputuote sidosryhmille. (Westland, 2018)

7.1 Testaustavat

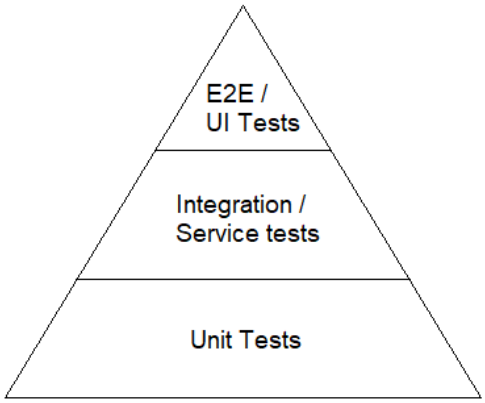
Yksinkertaisin tapa testata on koodin tai scriptin sisään kirjoitettava *console.log* komento, jolla voidaan testata, toimiiko koodi halutulla tavalla. Jos terminaaliin ilmestyy virheilmoituksia, voi niiden avulla määrittää missä kohtaa scriptiä virhe tapahtuu, jos konsolissa näkyvä virheilmoitus on epäselvä. Unityssä console.login korvaa *Debug.Log* Joka toimii täsmälleen samaan tapaan kuin console.log. Unityssä on myös oma Console-terminaalinsa, josta scriptien suorittamista voi seurata.



Kuva 14.

Compatibility testing tarkoittaa applikaation testausta kaikilla kohdealustoilla. Sillä voi selvittää, miten applikaatio soveltuu käytettäväksi vaikkapa Android-laitteille, kun se on alun perin suunniteltu IOS laitteille. Tähän sisältyy myös eri valmistajien laitteet. Valmistajia on nykypäivänä monia ja laitteiden ominaisuuksissa on suurta vaihtelua, mikä usein johtaa siihen, että compability testing jätetään usein vähälle huomiolle, ja keskitytään vain siihen että applikaatio toimii muutaman suositun valmistajan laitteilla (esim. Samsung ja Apple). (Cheng et al. 2015)

Compability testausta ei sovellettu tässä projektissa, sillä alun alkaen applikaatio on tarkoitettu vain tietyllä Android-tabletilla toimivaksi. Mikäli tähän tulee tulevaisuudessa muutoksia, on tämä testaus silloin pakollista.



Kuva 15.

Testing pyramid on yleisesti ohjelmoijien keskuudessa käytetty metafora testien lajittelusta eri kategorioihin. Alempana avataan näitä tapoja ja kategorioita hieman enemmän.

End-to-End (E2E) testauksessa nimensä mukaisesti testataan applikaation tai ohjelman toiminta käynnistämisestä loppuun. Mahdollisesta kirjautumisesta tai tunnuksen luomisesta eteenpäin loogista ja suunniteltua reittiä. Tämä on testauskategorioista ”hitain” sillä tämä vie eniten aikaa. Nämä testit ovat usein myös hieman epäluotettavampia, sillä mitä monimutkaisempi ohjelma, sitä helpommin tulee valheellisia virheitä. (Vocke, 2018)

UI testit ja E2E laitetaan yleensä saman nimen alle, mikä saattaa osittain olla loogista, sillä testataksesi applikaatiosi alusta loppuun, täytyy sinun testata samalla UI:n toimivuus, mutta UI:n toimivuutta ei voi testata E2E tyyliin. UI testaus voidaan ajaa Unit testeinä *Front-End*in koodissa tai vaikka *Selenium*in tai *Robot Framework*in kaltaisilla testikirjastoilla Integraatiotesteinä. (Vocke, 2018)
Integration Testing on tapa, jossa keskitytään testaamaan yhtä luokkaa tai funktiota, tai muuta osaa ohjelmasta. Testit suoritetaan niin, että testi syöttää dataa ohjelmaan tarvittaviin kohtiin, ja tutkii sitten ohjelman ulos antamaa dataa. (Kapelonis, 2018)

Unit testauksella tarkoitetaan lähdekoodin sisään kirjoitettuja testejä, jotka käyttävät jotakin ulkopuolista kirjastoa. Näin testeillä on pääsy lähdekoodiin ja näkymä kaikkeen tarvittavaan. Tämä testausmetodi vaatii hieman vähemmän työtä kuin *Integration*-testaus, joka vaatii yleensä jonkin deploymentin tai setupin toimiakseen oikein, siinä missä Unit

testauksessa voidaan vain linkittää käytettävä kirjasto tai kirjastot. Nämä testit ovat kolmesta kategoriasta nopeimpia. (Vocke, 2018)

7.2 Mitä testata

Juomanäytön kohdalla päädyttiin testaamaan yksinkertaisia asioita Debug.Log:lla sekä E2E testejä unityn omassa ympäristössä. Testaus unityn omassa ymäpäristössä on helppoa ja nopeita testejä voi tehdä kehityksen lomassa. Esimerkkinä sivun 13. skripti jossa listan objektien tulostus ja prefabit määritellään. Testaamalla tulostusten toimintaa ja prefabien paikkoja scenessä kehittämisen ohessa, vältyttiin mahdollisilta logiikkavirheiltä ja turhilta buildeilta. Turhaa projektin buildausta kannattaa Unity:ssä välttää, sillä Gradlen rakentaminen buildiin vie aikaa, varsinkin jos projekti on suuri.

8 Pohdinta

Tämä projekti oli jo hieman alkujaan poikkeava normaalista mobiilisovelluksen kehityksestä. Projekti oli oppimiskokemus niin tekijälle kuin toimeksiantajallekin. Retrospektiivissä ajatellen, on muutamia asioita, joita olisi kannattanut suunnitella enemmän tai tehdä ehkä jopa täysin toisin. Sovelluksen testausta olisi ehdottomasti pitänyt suunnitella paremmin, sillä Unityn tarjoamaan emulaattoriin oli liian helppo tukeutua. Myös loppukäyttäjillä testaaminen olisi ollut hyväksi, mutta projektin tiukan aikataulun vuoksi moinen ei ollut mahdollista. On kuitenkin hyvä muistaa, että esimerkiksi BDD ja TDD ovat vain muutamia monesta eri tavasta kehittää sovelluksia, joten on hyvä harjoittaa kriittistä ajattelua siitä, mitä juuri tämä projekti tarvitsee. Joskus mikään valmis muotti ei tavalla tai toisella sovi, joten silloin on parempi vain soveltaa ja valita parhaat palat eri paikoista. Tärkeintä on edetä niin, että saadaan aikaan laadukas sovellus, eikä niin, että kuka seurasi minkäkin kehitysmetodin sääntöjä parhaiten.

8.1 Projektin luovutus toimeksiantajalle

Projektin lopputuotteen luovutuspäiväksi on sovittu 25.11, eli tämän opinnäytetyön valmistumisen jälkeen, sillä toimeksiantaja halusi näin varmistaa, että itse sovelluksen hienosäätöön jää tarpeeksi aikaa. Luovutukseen sisältyy sovellus ja siihen sisältyvät scriptit ja muut materiaalit. Samalla luovutetaan toimeksiantajalta lainaan saatu kannettava tietokone.

8.2 Jatkokehitysideat

Kun idea on saatu validoitua ja sopimuksia toimittamisesta on allekirjoitettu, suosittelen koko applikaation rakentamista React Nativella Unityn sijaan, jolloin applikaatiosta saadaan kevyempi ja niin sanotusti ”solakampi”. Se on muun muassa paljon taipuvaisempaa, jolloin saa juuri sellaista kuin haluaa eikä vain tarpeeksi lähellä olevaa. Siten saataisiin tuotanto vielä enemmän Luotone Oy:n omiin käsiin. React Nativen käyttäminen mahdollistaisi myös applikaation käytön tulevaisuudessa muilla laitteilla, mikäli joskus siirryttäisiin Play- tai Apple-kaupasta ladattavaan versioon, jonka loppukäyttäjä lataisi omalle laitteelleen.

Unityllä tehdyissä applikaatioissa on vakiona Unityn oma splash screen joka näkyy aina, kun applikaation käynnistää. Tämä on pieni yksityiskohta, joka harvemmin käyttäjiä häiritsee, mutta pienten detaljien huomioon ottaminen luo kokonaisvaltaisesti paljon ammattitaitoisemman kuvan. Ja koska ihmisillä on niin vahva mielikuva Unitysta nimenomaan pelien kehittämisen alustana, saattaa applikaatiosta tulla ristiriitainen ja ”halpa” kuva. Toinen asia on käyttöliittymän kielivalinnat, jotka tulisi ehdottomasti olla vähintään kahdella kotimaisella kielellä, joiden lisäksi olisi englanti. Myös esteettömyys olisi hyvä muistaa kun sovellusta viedään eteenpäin kehityksessä. Selkeät ja helposti juettavat fontit ja vaikkapa mahdollisuus suurentaa fonttia tarvittaessa.

Lähteet

Cheng J., Gao j., Uehara T., Zhang T. 2015. Compatibility Testing Service for Mobile Applications. Luettavissa: https://www.researchgate.net/profile/Jerry_Gao/publication/274370752_Compatibility_Testing_Service_for_Mobile_Applications/links/551bf4d00cf20d5fbde22d8c/Compatibility-Testing-Service-for-Mobile-Applications.pdf Luettu: 21.10.2019

Garg Priyanka. 2017. 9 Things You Must Know About FDD – Feature Driven Development. Luettavissa: https://www.openxcell.com/9-things-must-know-fdd-feature-driven-development Luettu: 14.11.2019

Gorman Daniel. 2016. A Crash Course in Interpreted vs. Compiled Languages. Luettavissa: https://medium.com/@DHGorman/a-crash-course-in-interpreted-vs-compiled-languages-5531978930b6 Luettu: 15.10.2019

Hejlsberg A., Wiltamuth S., Golde P. 2006. The C# Programming Language. Toinen painos. Addison-Wesley Professional. USA

Hocking Joseph. 2015. Unity in Action. Manning Publications Co. USA

Jithin Nair. 2018. TDD vs BDD – What's the Difference Between TDD and BDD? Luettavissa: https://blog.testlodge.com/tdd-vs-bdd/ Luettu: 20.10.2019

Kapelonis Kostis 2018. Software Testing Anti-patterns. Luettavissa: http://blog.codepipes.com/testing/software-testing-antipatterns.html Luettu: 09.10.2019.

Kotonya G., Sommerville I, P. 1997. Requirements engineering: Processes and techniques. 3. painos. Wiley. USA.

Martin Robert C. 2015. Clean code: a handbook of agile software craftsmanship. 15. painos. Prentice Hall. USA

Mudrakola Sukesh. 2018. Android kiosk mode: Google's new mobile device enterprise strategy. Luettavissa: http://techgenix.com/android-kiosk-mode/ Luettu: 22.10.2019

Rizwan Asif. 2018. SQLite and Unity: How to do it right. Luettavissa: https://medium.com/@rizasif92/sqlite-and-unity-how-to-do-it-right-31991712190 Luettu: 15.10.2019

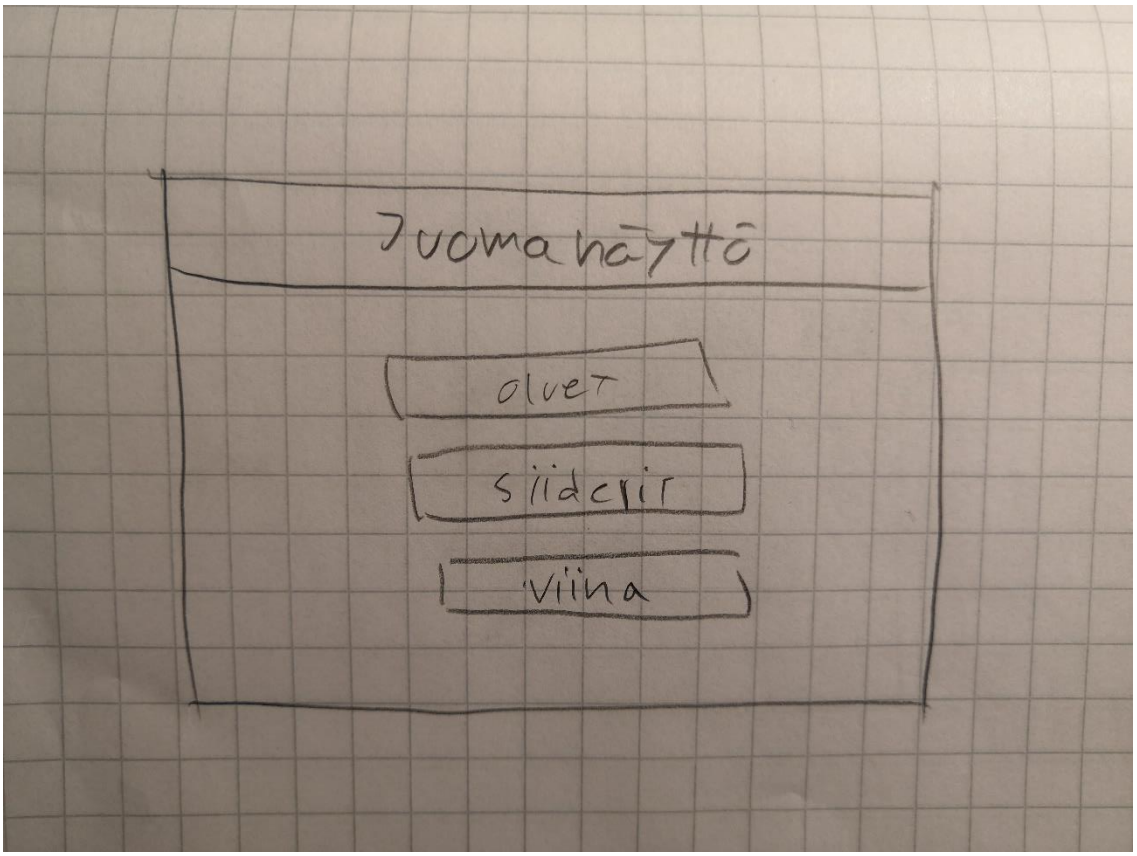
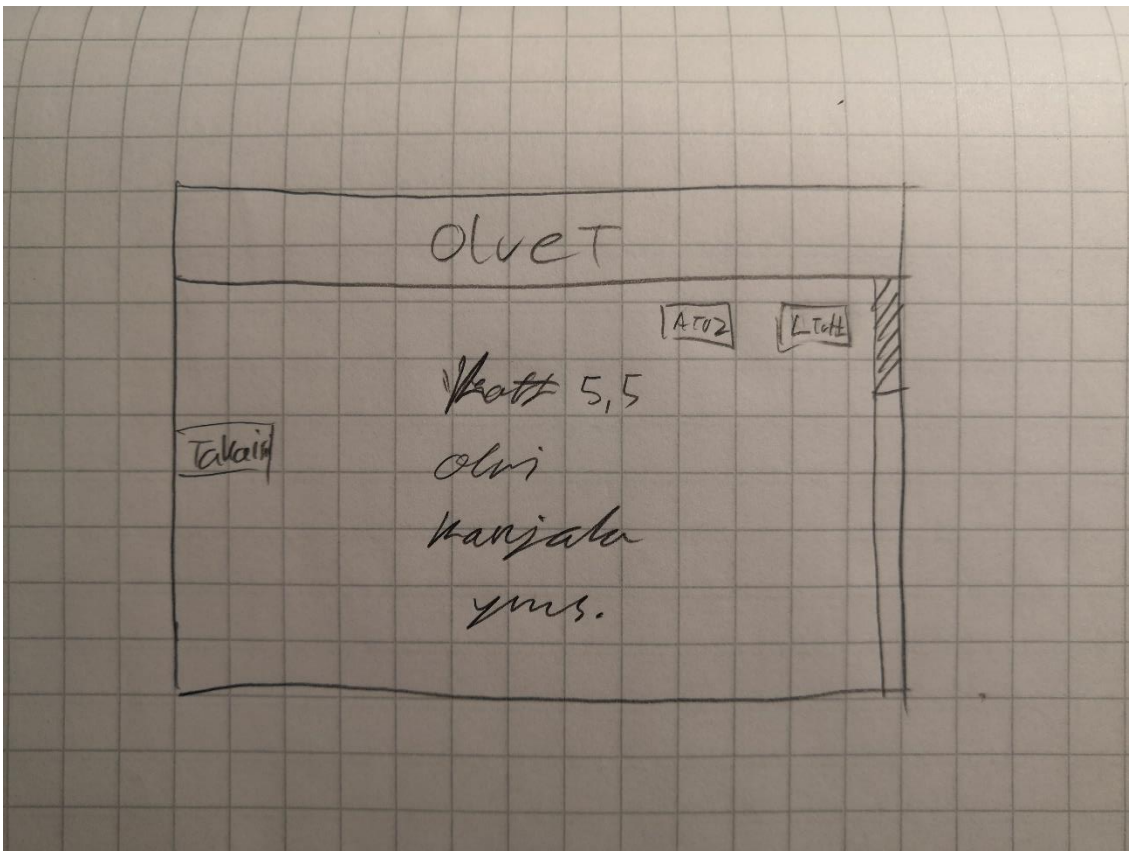
Thorn Alan. 2013. Learn Unity For 2D Game Development. Apress. USA

Vocke Ham. 2018. The Practical Test Pyramid. https://martinfowler.com/articles/practical-test-pyramid.html Luettu: 26.10.2010

Westland Jason. 2018. Quality Assurance and Testing: A Quick Guide. https://www.projectmanager.com/blog/quality-assurance-and-testing Luettu: 26.10.2019

https://www.json.org/ Luettu: 21.10.2019

Liitteet

Liite 1. Näyttökaappauksia applikaatiosta

Liite 2. Raakoja suunnitelmia käyttöliittymästä