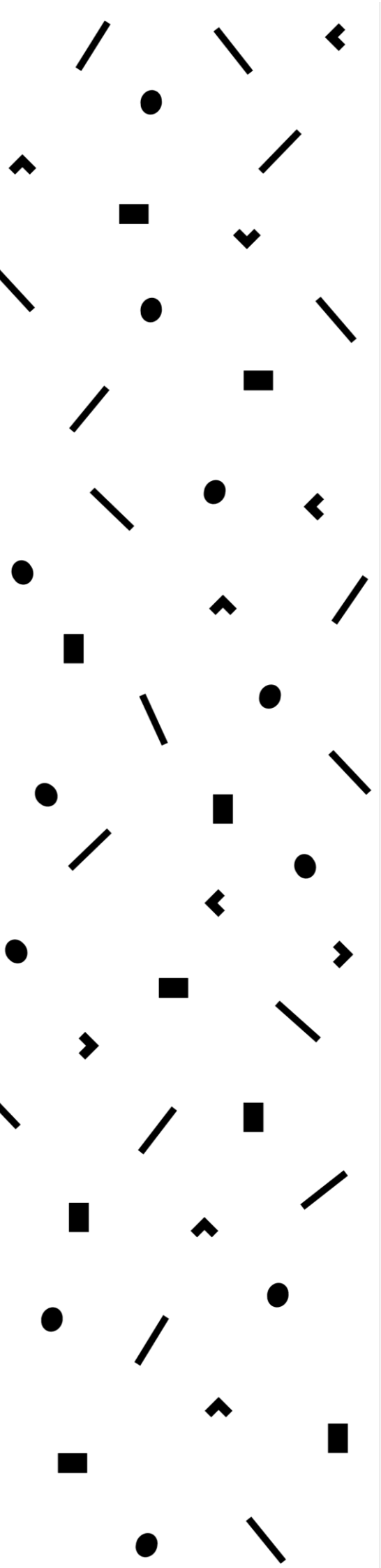


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

# Hiustyyligeneraattori Unity-pelimoottorilla

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikka
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2017
Teemu Hämäläinen

Lahden ammattikorkeakoulu
Tietotekniikka

HÄMÄLÄINEN, TEEMU:

Hiustyyligeneraattori
Unity pelimoottorilla

Ohjelmistotekniikan opinnäytetyö, 44 sivua, 0 liitesivua

Kevät 2017

TIIVISTELMÄ

---

Opinnäytetyön tarkoituksena oli toteuttaa hiustyyligeneraattori mobiilisovellus Unity-pelimoottorilla Android-laitteille. Työn toimeksiantajana toimi L'Oréal Finland. Työn tekemisessä oleellisia asioita olivat käyttöliittymän suunnittelu, kaksiulotteinen grafiikka, tabletin kosketusnäytön ja laitteen kamera. Työn vaatimus oli saada aikaan hiustyyligeneraattori, jolla käyttäjä voi kokeilla eri hiustyylejä itselleen.

Teoriaosuudessa käytiin läpi käyttöliittymän suunnittelua, Unityn toiminnallisuutta sekä sen käytönperusteita, kuten käyttöliittymää ja projektien koostumusta. Materiaalina toimivat alan kirjallisuus ja verkkosivut, pääosin Unityn omat sivut.

Työn edetessä Unity osoittautui varsin toimivaksi kehitysalustaksi ja suurilta ongelmilta vältyttiin. Sillä pystyttiin toteuttamaan kaikki haluttu toiminnallisuus. Joitakin ongelmia sovellusta tehdessä tuli vastaan, mutta niistä selvittiin dokumentaatiota tutkimalla. Jatkokehitettäväksi jäi otetun kuvan tallennus ja sen edelleen hyödyntäminen.

Työn tuloksena kehitettiin hiustyyligeneraattori-sovellus. Unity havaittiin toimivaksi kehitystyökaluksi, joka tarjoaa laajoja ominaisuuksia.

Asiasanat: Unity, Android, hiustyyligeneraattori, mobiilisovellus

Lahti University of Applied Sciences
Degree Programme in Information Technology

HÄMÄLÄINEN, TEEMU: Hair-style generator using the Unity game engine

Bachelor's Thesis in software engineering, 44 pages, 0 pages of appendices

Spring 2017

ABSTRACT

---

The goal of this Bachelor's thesis was to develop hair-style generator mobile application for android devices, using the Unity game engine. The thesis was commissioned by L'Oréal Finland. The essential subjects were the user interface, two-dimensionel graphics, the touch screen for the tablet and the devices camera. The aim was to create a hair-style generator application which the user can use to try different hair-styles for themselves.

The theoretical part of the thesis deals with designing a user interface, Unity's features and the basics of its use. The sources used were literature and websites.

Unity proved to be a very effective platform for development and major problems were avoided. Whit Unity it was possible to implement all the desired functionality. Some problems were encountered, but they were solved by studying Unity's documentation. For further development, there are the following things to be done: image saving and further exploitation of the saved image.

As a result of the thesis, a hair-style generator application was developed. Unity was found to be a functional development tool offering extensive features.

Key words: Unity, Android, hair style application, mobile application

SISÄLLYS

# 1 JOHDANTO

Mobiilisovellus on sovellus, joka on suunniteltu toimimaan mobiililaitteissa, kuten älypuhelimissa ja tablet-laitteissa. Mobiilisovellusten käyttö on yleistynyt viime vuosina, sillä yhä useampi ihminen omistaa älypuhelimen tai tabletin. Koska mobiililaitteiden käyttö on kasvanut, on kasvanut kysyntä myös erilaisille mobiilisovelluksille. Sovelluksen suunnittelijan on tärkeä ymmärtää mobiililaitteiden ominaisuudet, jotta sovelluksesta voidaan tehdä mahdollisimman käytännöllinen.

Nykypäivänä lähes kuka tahansa voi alkaa itse kotikoneellaan kehittämään mobiilisovelluksia. Monenlaiset kehitystyökalut mahdollistavat tekniikasta kiinnostuneiden ihmisten toteuttaa erilaisia sovelluksia. Mobiililaitteiden yleistyessä yhä useampi yritys haluaa parantaa asiakkaidensa käyttäjäkokemusta juuri heidän tarpeisiin suunnitellun sovelluksen avulla. Tämä tarjoaa mahdollisuuden kehittää yrityksen liiketoimintaa uudelle tasolle. Toimiva tuote luo asiakkaalle paremman käyttäjäkokemuksen. Toisin kuin verkkosivut, voidaan sovellus rakentaa toimivaksi myös ilman verkkoyhteyttä. Työn toimeksiantajana toimi L'Oréal Finland.

Opinnäytetyön tavoitteena oli kehittää kaksiulotteinen hiustyyligeneraattori-mobiilisovelluksen ensimmäinen versio tablet-laitteelle. Sovelluksella käyttäjä voi ottaa henkilöstä kuvan ja kokeilla eri hiustyylejä sekä värejä kuvatulle henkilölle. Opinnäytetyön tutkimusongelma oli, kuinka hiustyyligeneraattorisovellus toteutetaan Unity-pelimoottorilla.

Lisäksi opinnäytetyössä käydään läpi käyttöliittymän suunnittelua, testausta ja Unity-pelimoottoria, joka toimi sovelluksen teossa kehitystyökaluna. Käyttöliittymän suunnittelua käydään läpi yleisellä tasolla. Pelimoottoreita, jotka tukevat mobiilisovelluksia, on monia, mutta tässä opinnäytetyössä ei tulla käsittelemän muita pelimoottoreita kuin Unityä.

Työn aihe valittiin, koska toimeksiantaja halusi hiustyyligeneraattori-mobiilisovelluksen testikäyttöön parturikampaamoketjuun. Näin saataisiin selville, tuoko sovellus lisäarvoa asiakkaalle.

# 2 KÄYTTÖLIITYMÄN SUUNNITTELU

Käyttöliittymän suunnittelun tarkoituksena on koneiden ja ohjelmistojen käyttöliittymien käytettävyyden- ja käyttäjäkokemuksen parantaminen. Käyttäjäkokemuksella tarkoitetaan sovelluksen käyttäjän tunteita ja asenteita käytettävää sovellusta kohtaan. Koneita, joihin käyttöliittymäsuunnittelua tehdään, ovat esimerkiksi tietokoneet, kodinkoneet, mobiililaitteet ja muut elektroniset laitteet, mutta tässä opinnäytetyössä keskitytään mobiilisovelluksen käyttöliittymään.

Tavoitteena on tehdä käyttäjän ja käyttöliittymän vuorovaikutus mahdollisimman yksinkertaiseksi ja tehokkaaksi. Huonosti suunniteltu käyttöliittymä vaikuttaa paljon loppukäyttäjän käyttäjäkokemukseen ja sitä kautta sovelluksen lopputulokseen. Graafisella suunnittelulla voi tehdä käyttöliittymästä selkeämmän ja johdattaa käyttäjän sinne, missä olennainen toiminnallisuus sijaitsee.

Sivustoa suunniteltaessa ja tarkasteltaessa voidaan käyttöliittymän suunnitteluun soveltaa heuristisen arvioinnin menetelmiä. Heuristinen arviointi perustuu heuristiikkoihin, jotka muodostuvat säännöistä ja ohjeista, joita käytettävyydeltään hyvän käyttöliittymän tulisi noudattaa. Heuristinen arviointi tuottaa lopputulokseksi listan käytettävyyspuutteista ja ongelmista, jotka havaitaan arvioinnin aikana. (Kuutti 2003, 49–50.)

Sovelluksen käyttöliittymän suunnittelussa on tärkeää huomioida käyttöliittymän interaktio sovelluksen kanssa. Erilaisia tapoja ovat esimerkiksi näppäimistö, hiiri, kosketusnäyttö ja erilaiset ohjaimet. Esimerkiksi mobiilisovelluksen näyttö on verrattain pieni, joten sovelluksen painikkeista täytyy tehdä riittävän suuret. Tämän lisäksi mobiilisovellusten suunnitteluun vaikuttavat asiat, kuten käyttö ulkona, jolloin näyttöön heijastuu enemmän valoa, niiden käyttö tilanteissa, joissa sitä täytyy pystyä käyttämään nopeasti yhdellä kädellä tai ilman sen kummempaa huomion kiinnittämistä itse näyttöön.

## 2.1 Käyttöliittymä

Käyttöliittymä on käyttäjän ja koneen välisen vuorovaikutuksen väline eli käyttäjän ja sovelluksen välinen "ohjausnäkymä". Käyttöliittymä on mobiilisovelluksen osa, jolla käyttäjä käyttää kyseistä tuotetta. Käyttöliittymä voi olla mobiilisovelluksen graafinen käyttöliittymä kuin myös astianpesukoneen käyttöpaneeli.

Lähtökohtana käyttöliittymälle on, että käyttäjä voi käyttää sovellusta ja liikkua siinä kuten haluaa. Käyttöliittymää käyttäessä sovelluksen käyttäjä ei halua lukea ohjeistuksia tai perehtyä käyttöliittymän tapaan toimia.

Käyttöliittymiä on erilaisia, riippuen niiden käyttäjäryhmästä ja tarkoituksesta. Käytetyin on perinteinen käyttöliittymä painettavilla napeilla sekä nykyisin kosketusnäytöllinen käyttöliittymä.

## 2.2 Käytettävyys

Käytettävyys on tärkeä osa kaikkien laitteiden käyttöliittymää. Hyvin usein käytettävyyteen kiinnitetään liian vähän huomiota, jolloin syntyy sovelluksia, joissa on paljon toiminnallisuuksia, mutta niitä on hankala käyttää ja tapahtumat tehdään monen vaiheen kautta.

Käyttöliittymän hyvä käytettävyys on etulyöntiasema kilpailevia tuotteita ajatellen. Käyttäjät helposti suosittelevat käyttämiänsä tuotteita eteenpäin, jos ovat saaneet niistä hyviä kokemuksia. Hyvän käyttäjäkokemuksen saanut käyttäjä myös palaa käyttämään käyttöliittymää uudelleen.

Tieteenalana käytettävyys tutkii ja käsittelee tuotteen ominaisuuksia, jotka tekevät sen käytettävyydestä hyvän tai huonon. Se käsittelee myös menetelmiä, joita voi käyttää hyväksi tuotteen hyvän käytettävyyden suunnittelussa ja sen käytettävyyden arvioimisessa. (Kuutti 2003, 14.)

Käytettävyys määritellään viidellä laatumääreellä:

- opittavuus
- tehokkuus
- muistettavuus
- virheettömyys
- tyytyväisyys (Nielsen 1993, 26–27).

Opittavuudella tarkoitetaan, että järjestelmän tulisi olla helposti opittavissa niin, että käyttäjä pääsee nopeasti suorittamaan haluttua tehtävää järjestelmässä. Tehokkuus tarkoittaa tasoa, jolle järjestelmän käytön nopeus sijoittuu, kun se on opittu hyvin. Muistettavuus on tärkeää erityisesti sellaisten toimintojen ja tuotteiden kohdalla, joita käytetään harvoin. Muistettavuus liittyy siihen, miten helposti toimintojen, termien ja graafisten merkkien sisältö on muistettavissa sen jälkeen, kun järjestelmän käyttö on kerran opittu. Sen takia muistettavuus liittyy vahvasti järjestelmien satunnaiseen käyttöön. Virheettömyydellä tarkoitetaan sitä, että järjestelmässä tulisi syntyä virhetilanteita harvoin, ja virheiden sattuessa tulisi käyttäjän saada niistä palaute nopeasti ja pystyä palautumaan niistä tehokkaasti. Tyytyväisyys kuvaa käyttäjän tyytyväisyyttä järjestelmään (Koivunen & Nieminen 1995 22– 24).

Nielsenin listassa käytettävyysopit on listattu helposti sovellettaviin sääntöihin.

- Vuorovaikutuksen käyttäjän kanssa tulee olla yksinkertaista ja luonnollista.
- Vuorovaikutuksessa tulee käyttää käyttäjän omaa kieltä.
- Käyttäjän muistin kuormitus tulee minimoida.
- Käyttöliittymän tulee olla yhdenmukainen.
- Järjestelmän tulee antaa käyttäjälle kunnollista palautetta reaaliajassa.
- Ohjelmassa ja sen osissa tulee olla selkeät poistumistiet.
- Oikopolkuja ja tehokasta työskentelyä pitäisi tukea.

- Virheilmoitusten tulee olla selkeitä ja ymmärrettäviä.
- Virhetilanteisiin joutumista tulisi välttää.

Käyttöliittymässä tulee olla kunnolliset avustustoiminnot ja dokumentaatio (Kuutti 2003, 49).

## 2.3 Käyttäjät

Käyttäjät toimivat käyttöliittymän suunnittelun perustana. Hyvä käyttäjätuntemus on usein hyvän käytettävyyden edellytys. Tämän takia on tärkeä ymmärtää, millainen ihminen käyttöliittymää käyttää. Käyttäjien toimintaa ohjaa monta eri tekijää: koulutus, ikä, toiminnan rajoitteet ja tila, jossa järjestelmää käytetään (Sinkkonen, Kuoppala, Parkkinen & Vastamäki 2006, 29). Tiedot tuotteen lopullisesta käyttötarkoituksesta, käyttötilanteesta ja käyttöympäristöstä ovat hyödyllisiä parempaa käytettävyyttä suunniteltaessa. Tämä auttaa arvioimaan käyttäjäryhmän mahdolliset toiveet ja kertyneen kokemuksen mobiilisovelluksista.

Käyttäjä tekee aisteillaan havaintoja ympäröivästä maailmasta ja toimii sen kanssa interaktiivisesti. Aistit ja niihin liittyvät ajatustoiminta ja päättelymekanismit on hyvä huomioida käytettävyyttä suunniteltaessa. Biologian lisäksi kulttuuri vaikuttaa käyttäytymiseemme. Kulttuuriset lähtökohdat voivat luoda käyttäjälle tietyt normit ja tavat, joiden mukaan hän toimii. Näiden biologisten ja kulttuuristen ominaisuuksien summa ohjaa ihmistä käyttäytymään tietyllä tavalla tietyssä tilanteessa. Nämä normit huomioiden käyttöliittymää suunniteltaessa voidaan käytöstä tehdä luontevaa ja helppoa (Kuutti 2003, 22–23).

## 2.4 Käyttöliittymän testaus

Käyttöliittymän testauksen tarkoituksena on löytää käyttöliittymässä olevat ongelmakohdat, joita voidaan ratkaista tuotteen jatkokehityksessä. Ongelmakohtia voivat olla esimerkiksi käyttöliittymän epäselvyys ja ristiriitaisuus. Testauksen lopputulokset käydään läpi kehitystiimin kanssa ja keskustellaan siitä, miten ongelmakohtia voidaan mahdollisesti

parantaa. Ongelmakohdat kannattaa aina perustella, miksi se on käytettävyyden kannalta huono ratkaisu.

Käyttöliittymän testaus on tärkeää, koska sovelluksen kehittäjä ei välttämättä ole perillä kaikista asioista, joita tuotteen loppukäyttäjät toivovat. Tuotteen varsinaisessa käyttöympäristössä tehty testaus voi paljastaa ongelmakohtiakohtia, joita ei muuten ole mahdollista saada selville.

## 2.5 Hyvä käyttöliittymä

Hyvä käyttöliittymä helpottaa käyttäjän sen hetkistä tehtävää vetämättä tarpeetonta huomiota itseensä. Hyvä käyttöliittymä kertoo käyttäjälle sen, mitä käyttäjän juuri sillä hetkellä tarvitsee tietää. Käyttöliittymässä tulee olla selkeä ja johdonmukainen rakenne, jota käyttäjän on luonnollista käyttää.

Tärkeintä on muistaa, ettei käyttäjä yleisesti ottaen välitä teknologiasta käyttöliittymän takana, vaan käyttäjä tahtoo suorittaa tehtävänsä käyttöliittymän avulla. Liian usein kuitenkin suunnittelija, yritysjohto tai markkinointi haluaa lisätä käyttöliittymään jotain uutta hienoa teknologiaa, jolloin inhimilliset tekijät kärsivät (Kortum 2008, 12 - 13).

# 3 UNITY-PELIMOOTTORI

Pelimoottori on pelinkehitysalusta, joka tarjoaa valmiita apuvälineitä pelin tekijälle. Näin pelin- tai sovelluksentekijä voi keskittyä oleelliseen eli itse tuotteen ideointiin ja toteuttamiseen.

Unity on tehokas ja joustava pelimoottori 3D- ja 2D-peleille ja sovelluksille. Unityn kehitysalustaa voidaan käyttää mobiilipelien- ja sovellusten tekemiseen. Sillä voi kehittää sovelluksia lähes kaikille mahdollisille alustoille, minkä takia se on noussut suureen suosioon pelinkehittäjien keskuudessa.

Unity-projekti koostuu näkymistä, peliolioista ja niiden komponenteista, sekä tiedostoista. Uutta projektia luotaessa valitaan ensin, onko sovellus kaksiulotteinen vai kolmiulotteinen, mutta tämä vaikuttaa vain kameran ja tekstuurien oletusasetuksiin ja valintoja voi muokata myöhemmin. Projektia luodessa voi myös ottaa projektiin mukaan Unityn tarjoamia paketteja, joissa on muun muassa. valmiita skriptejä ja graafisia efektejä.

Jokaisella projektilla on oltava vähintään yksi tallennettu näkymä, jotta sitä voidaan testata tai se voidaan julkaista. Yksinkertaistettuna näkymiä voidaan pitää esimerkiksi pelin tasoina, mutta myös valikot tai välianimaatiot voivat olla omia näkymiä. Näkymillä on omat peliolionsa, mutta jokaisessa näkymässä on oltava kameraolio. Unity luo uuden projektin mukana näkymän kameraolion kanssa, mutta tämä tulee itse tallentaa.

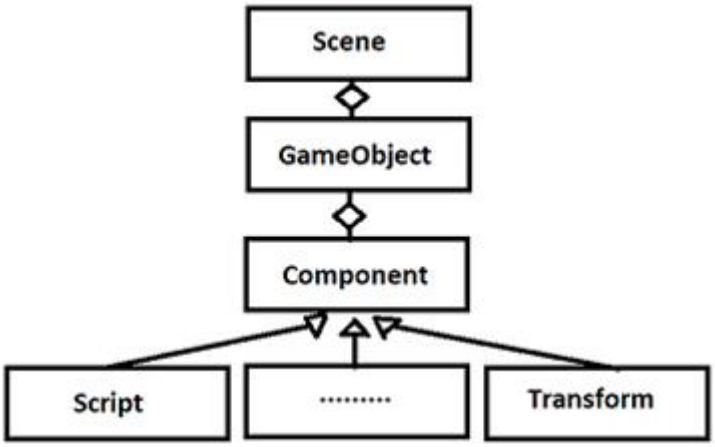
## 3.1 Unity-pelimoottorin toimintaperiaate

Unity-pelimoottorilla toteutettu sovellus (kuvio 1) koostuu näkymistä (Scene). Jokaisella sovelluksella on oltava vähintään yksi näkymä. Näkymä on peliobjektien (GameObject) säiliö. Loppukäyttäjä näkee aina yhden näkymän kerrallaan (Unity 2017f).

Jokainen asia, joka peliin luodaan, on peliobjekti. Jokainen peliobjekti on mahdollista määrittää erilaiseksi, kun sitä verrataan toiseen peliobjektiin.

Eroa voi olla esimerkiksi muodossa ja värissä. Peliobjektin ei välttämättä tarvitse näkyä käyttäjälle näkymässä. Unityssä voidaan luoda peliin peliobjekti, joka laskee, kuinka monta painallusta käyttäjä on tehnyt. Peliobjekteja voivat olla muun muassa. painikkeet, kuvat, valot ja äänet.

Peliobjekti ei tee mitään itsekseen, vaan toimii säiliönä komponenteille (Component), jotka tuottavat varsinaisen toiminnallisuuden (Unity 2017b). Komponentti tarkoittaa jotakin osaa, joka on liitetty johonkin peliobjektiin. Sitä voi verrata siihen, kun ihminen laittaa repun selkään, tällöin hän käyttää kyseistä reppua. Sama pätee objekteihin ja komponentteihin. Jos objektiin liittää jonkin komponentin, objekti pystyy käyttämään komponentin ominaisuuksia.




KUVIO 1. Näkymän rakenne

Komponentit määrittävät sen, millainen peliobjektista tulee. Peliobjektille voidaan määrittää yksilöllinen toiminta lisäämällä siihen skripti-komponentti (Script) (kuvio 2). Erilainen ulkonäkö saadaan aikaan tekstuuri (Texture) komponentilla (kuvio 3). Komponentteja hallitaan editorin Tarkistus-osiossa (Kuvio 4).

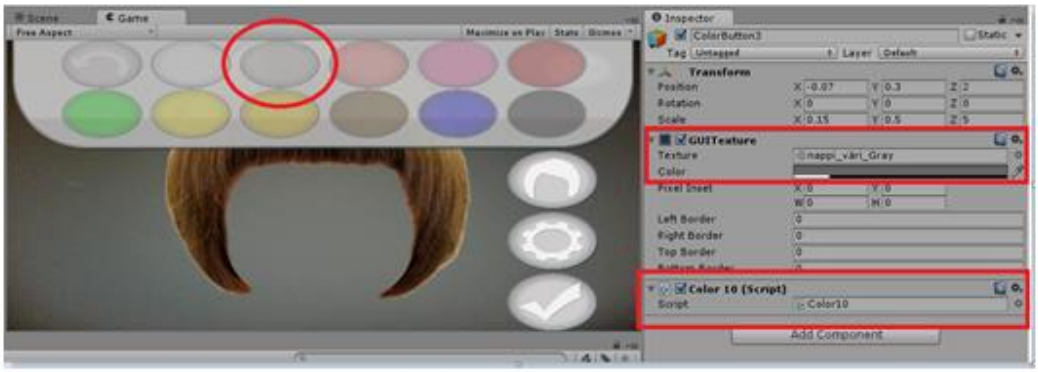
```
using UnityEngine;
using System.Collections;

public class Color10 : MonoBehaviour {
    private GameObject hairPicture;
    // Use this for initialization
    void Start () {
        hairPicture = GameObject.FindGameObjectWithTag ("Hair");
    }
    // Update is called once per frame
    void Update () {

    }
    void OnMouseDown(){
        hairPicture.guiTexture.color = new Color (0.3F, 0.3F, 0.3F, 0.8F);
    }
}
```

KUVIO 2. Harmaaseen nappiin kiinnitetty skripti-komponentti

KUVIO 3. Harmaan napin tekstuuri-komponentti




KUVIO 4. Harmaan napin texture- ja skripti-komponentit Tarkistus-osiossa

## 3.2 Yleistä skriptauksesta

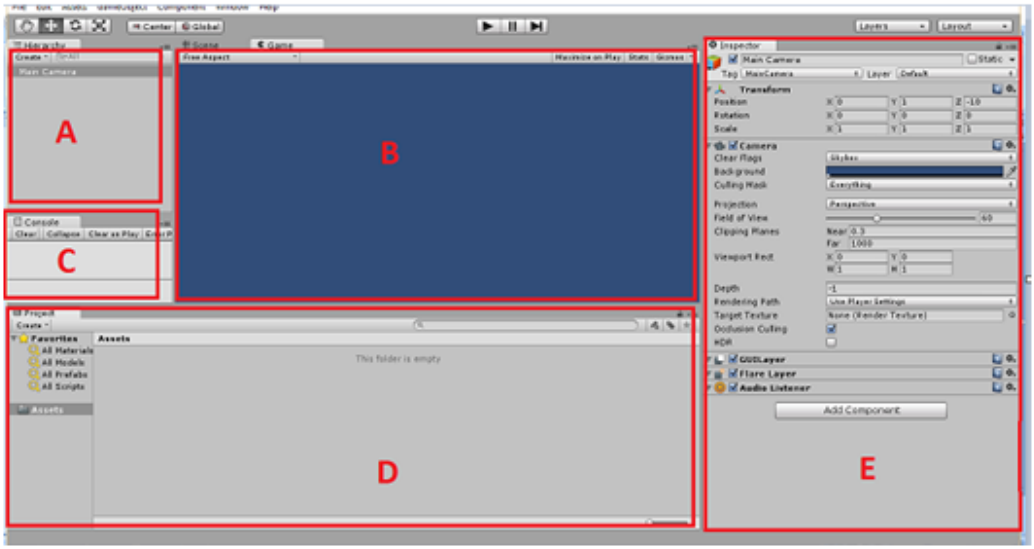
Unity-pelimoottorissa on mahdollista käyttää ohjelman kirjoittamiseen joko unityscript-kieltä tai C# -ohjelmointikieltä. C# on Microsoftin kehittämä, oliopohjainen ohjelmointikieli ja UnityScript on Unity-pelimoottoria varten suunniteltu, JavaScriptin kaltainen, ohjelmointikieli.

Unityllä sovelluksen ohjelmoimiseen voi käyttää mitä tahansa tekstieditoria, koska Unity suorittaa itse ohjelmakoodin käännöksen. Unityssä C#- tai Unityscript ohjelmointikielellä luotuja ohjelmia kutsutaan skripteiksi.

## 3.3 Käyttöliittymä

Unityn käyttöliittymä (kuvio 5) on jaettu osiin. Nämä osat ovat Hierarkia- (kohta A), Peli-, Näkymä- (kohta B), Konsoli- (kohta C) Projekti- (kohta D) ja Tarkistus-osioista (kohta E). Näillä osilla hallitaan projektin peliobjekteja.

Unityn käyttöliittymä on yksi syy siihen, että Unityä on niin nopea ja helppo käyttää. Assetteja voi siirtää sovellukseen raahaamalla ja pudottamalla. Assetit ovat tässä työssä tekstuureita ja skriptejä, mutta ne voivat olla myös, vaikka äänitiedostoja.




KUVIO 5. Havainnollistettu käyttöliittymä

## 3.4 Navigointi

Tässä luvussa käydään tarkemmin läpi Unityn käyttöliittymää havainnollistettujen kuvien avulla. Koska Unityn rakennetta on helpompi havainnollistaa jo olemassa olevan sisällön avulla, on kuvina käytetty hiusgeneraattori projektia.

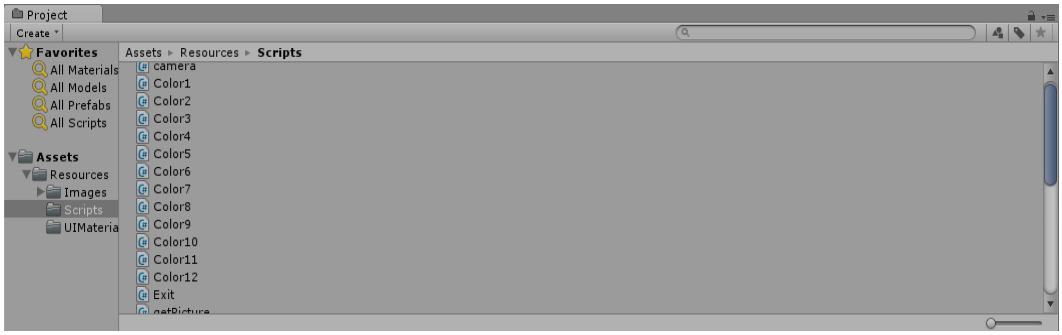
Hierarkia-osiossa (kuvio 6) on listattuna kaikki kentän staattiset peliobjektit (Unity 2017c). Peliobjektit ovat hierarkisessa järjestyksessä. Yhdellä peliobjektilla voi myös olla monta aliobjektia, joita kutsutaan lapsiksi. Hierarkia mahdollistaa peliobjektien ryhmittelyn: kun vanhempaa siirretään, sen lapset siirtyvät myös.


KUVIO 6. Hierarkia osio

Sovelluksen kaikki peliobjektit näkyvät Peli- ja Näkymä-osiossa (kuvio 7). Tätä osiota voidaan käyttää valikon luomiseen, uuden tason luomiseen tai vaikka animaation näyttämiseen (Unity 2017h).


KUVIO 7. Peli- ja Näkymä-osio

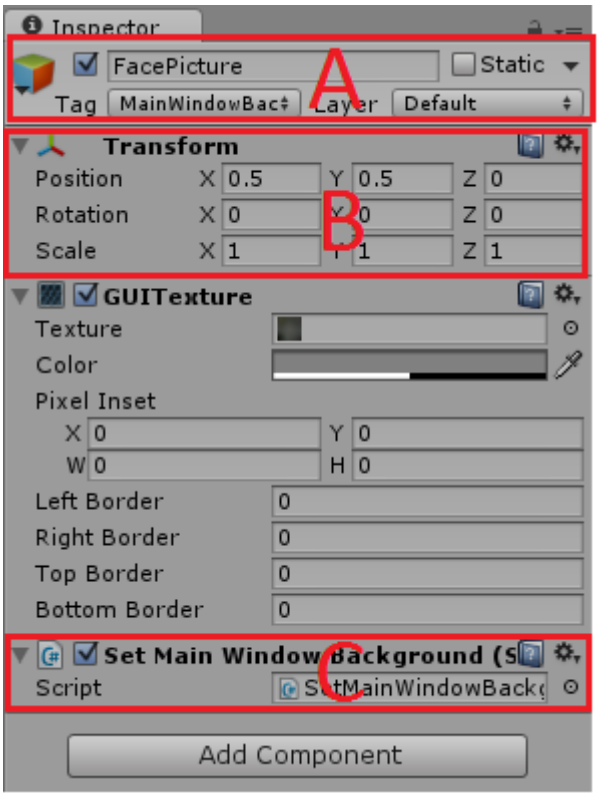
Projekti-osio (kuvio 8) listaa kaikki projektiin kuuluvat tiedostot ja kansiot. Tiedostoja ovat esimerkiksi pelinappuloiden ja kuvien mallit, äänet, komentosarjat ja objektit (Unity 2017e). Tästä näkymästä voi myös lisätä näkymään peliobjekteja ja peliobjekteihin komponentteja raahaamalla.




KUVIO 8. Projekti-osio

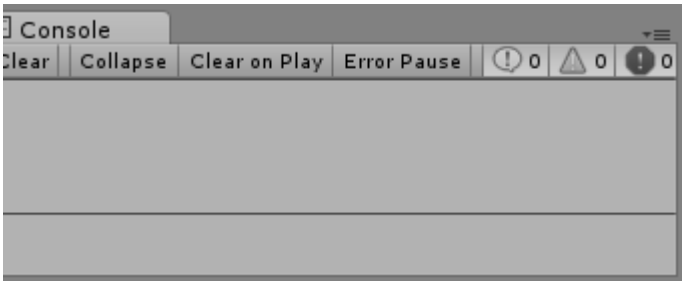
Tarkistus-osiossa (kuvio 9) näytetään valitun objektin tiedot ja komponentit sekä komponenttien tiedot (Unity 2017g). Tagien (kohta A) avulla voidaan hakea näkymästä tietyn nimisiä peliobjektia. Kaikilla peliobjekteilla on pakollinen Transform-komponentti (kohta B). Muita komponentteja ovat esimerkiksi objektille lisättävä skripti (kohta C), jossa voidaan esimerkiksi määritellä, mitä tapahtuu, kun ohjelman nappia painaa. Transform-komponentti kertoo peliobjektin sijainnin suhteessa sen vanhempaan.

Mikäli objektilla ei ole vanhempaa, peliobjektin sijainti kerrotaan suhteessa kentän nollapisteeseen. Peliobjekteihin voi lisätä komponentteja työkaluvalikosta tai vetämällä ne hiirellä Projekti-osiosta Tarkistus-osioon. Komponentit voi poistaa pikavalikon Delete-käskyllä.




KUVIO 9. Tarkistus-osio

Kuviossa 10 on esitetty konsoli-osio (kuvio 10). Tämä osio toimii apuna, kun skriptien koodista etsitään virheitä. Siinä näkyy pelin ajon aikana näkyvät käyttäjän lähettämät viestit, virheet ja varoitukset (Unity 2017a).

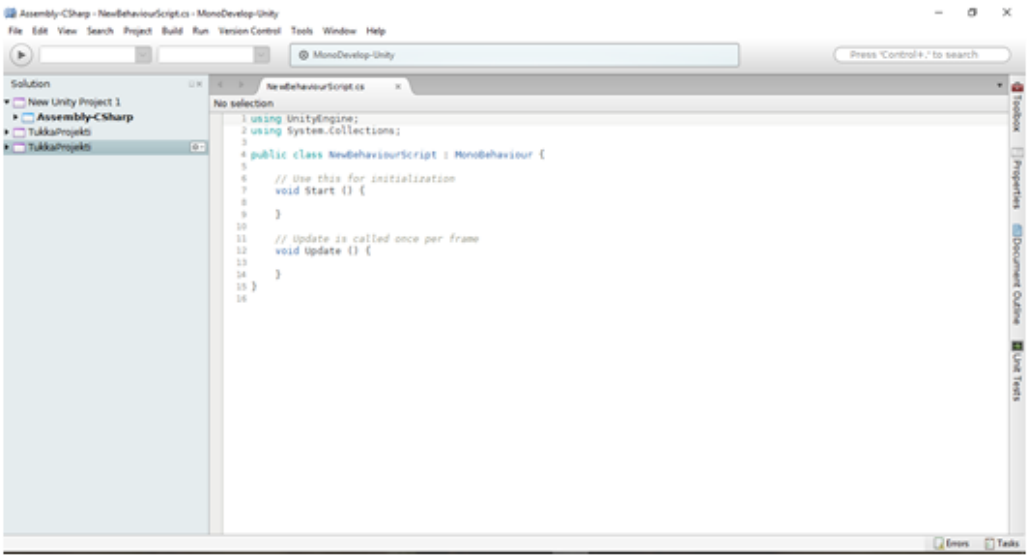



KUVIO 10. Konsoli-osio

## 3.5 MonoDevelop

MonoDevelop (kuvio 11) on avoimen lähdekoodin kehitysympäristö Linux-Windows- ja Mac OSX -käyttöjärjestelmille. Useimmin sitä käytetään tehtäessä projekteja, jotka käyttävät .NET ja MONO frameworkkeja.

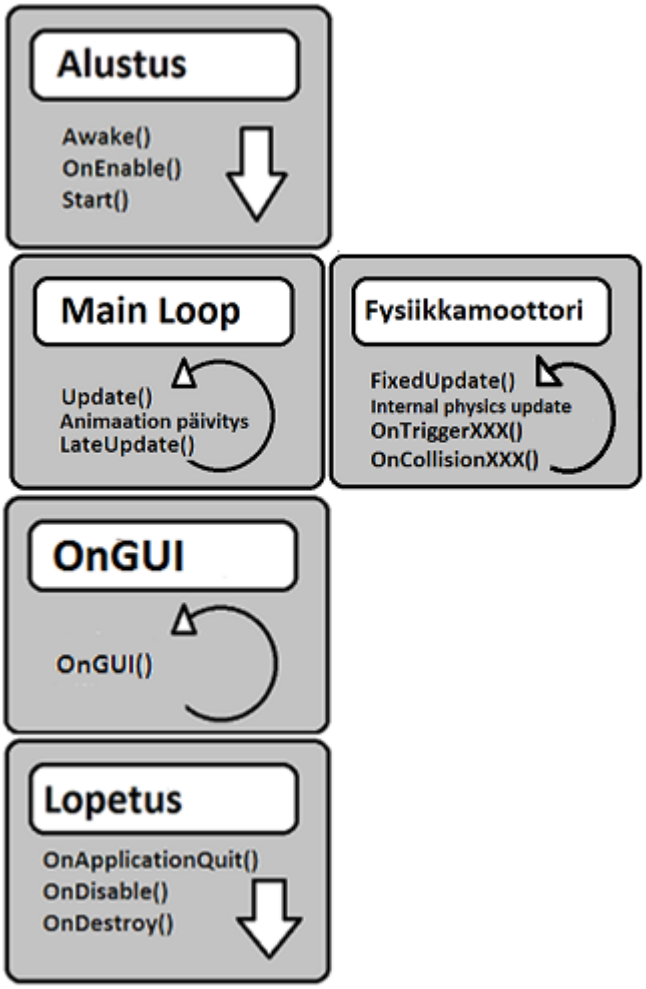
Unityn mukana tulee kustomoitu versio MonoDevelopista. Unityn tapauksessa MonoDevelopia on muokattu, jotta sillä pystyy testaamaan itse Unity-editorissa tehtyä sovellusta. MonoDevelop kääntää ohjelmoijan ohjelmakoodin ja ajaa sen toiminnassa Peli-näkymässä.




KUVIO 11 MonoDevelop

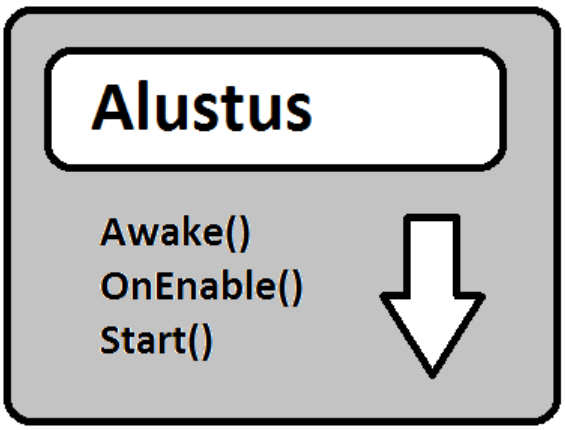
## 3.6 Game loop

Jokainen peliobjektin luokka periytyy MonoBehaviour-luokasta (Unity 2017d), jolloin niihin pätee Unityn skriptien elinkaari (kuvio 12). Peliobjektille voidaan myös määrittää verkkokäyttäytyminen (NetworBehaviour), joka voidaan periyttää peliobjektille antaen sille verkkotoimivuuden. Koodin elinkaari suoritetaan tietyssä järjestyksessä käyttämällä siihen tarkoitettuja funktioita. Elinkaaren vaiheita on lukuisia, mutta näistä olennaisempia tässä työssä ovat skriptin alustus, ruudun päivitykset ja skriptin tuhoutuminen. Unityllä on omat silmukat ruudunpäivitykseen ja fysiikkamoottorille.



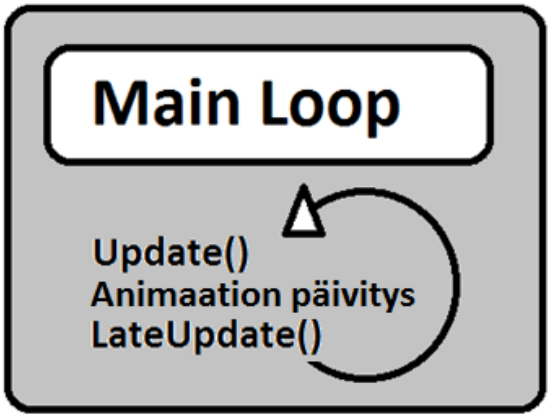

KUVIO 12. Yksinkertaistettu skriptin elinkaari

Koodin alustukseen (kuvio 13) käytetään näkymän ensimmäisellä latauskerralla kutsuttavia Awake- ja OnEnable-funktioita. Lisäksi alustukseen kuuluu ennen ensimmäistä ruudunpäivitystä kutsuttava Start-funktio. Awake-funktio kutsutaan aina, kun peliobjekti on aktiivinen. Awake-funktiota kutsutaan aina ennen Start-funktiota. OnEnable-funktiota kutsutaan silloin kun objekti on otettu käyttöön.



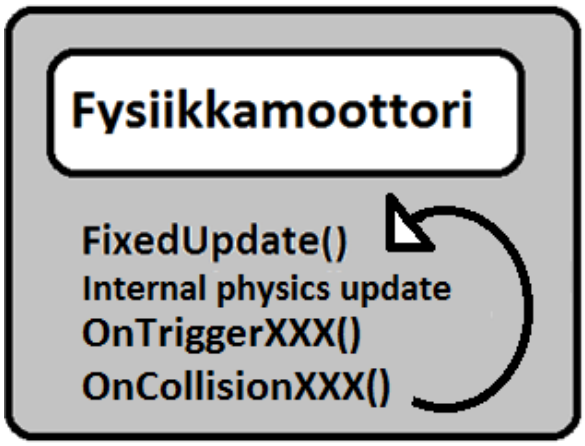

KUVIO 13. Alustuksen toiminta

Update-funktioita (kuvio 14) käytetään sovelluksen käyttäjän syötteiden lukemiseen, kun ei käytetä fysiikkamoottoria. Update-funktiota kutsutaan kerran ruudunpäivityksen yhteydessä. Unityssä oletuksena kuvataajuus on 60 fps. LateUpdate kutsutaan myös kerran ruudunpäivityksen yhteydessä ja sitä käytetään kolmannen persoonan kameroissa.



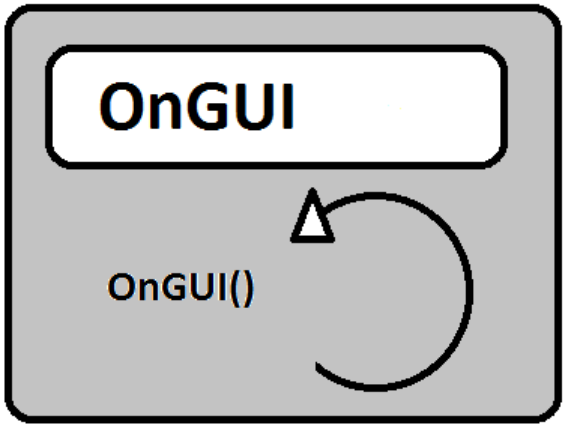

KUVIO 14. Update-funktioiden toimintajärjestys

Kaikki fysiikkamoottorin (kuvio 15) toiminta tapahtuu heti FixedUpdate funktion jälkeen. FixedUpdate-funktiota kutsutaan ajastimella. Fysiikkamoottoria voidaan kutsua useasti ruudunpäivityksen yhteydessä. Törmäystapahtumissa fysiikkamoottori vastaa OnCollision- ja OnTrigger-funktioiden kutsumisesta.



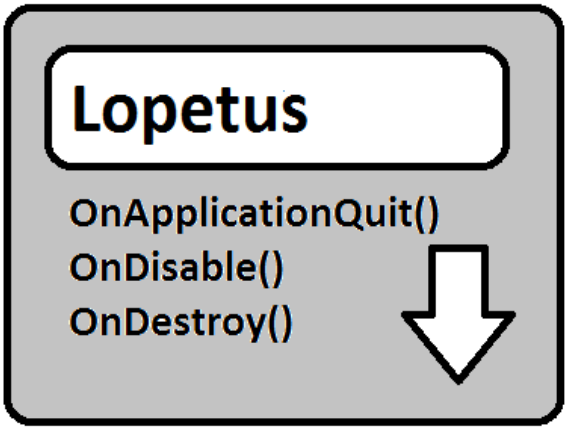

KUVIO 15. Fysiikkamoottorin toiminta

GUI on luokka, joka toimii rajapintana Unityn omaan graafiseen käyttöliittymään. OnGUI-funktiota (kuvio 16) käytetään vastaamaan GUI:n tapahtumiin. OnGUI-funktiota kutsutaan kerran ruudunpäivityksen yhteydessä kuten Update-funktiotakin. GUI-luokasta kerrotaan lisää luvussa 4.5.




KUVIO 16. OnGUI-funktio

Sovelluksen lopuessa (kuvio 17) käytetään funktioita OnDisable, OnDestroy ja OnApplicationQuit. Näillä funktioilla voidaan esimerkiksi tallentaa sovelluksen käyttäjän tiedot sitten kun sovellus suljetaan.




KUVIO 17. Sovelluksen lopetuksen funktioiden toimintajärjestys

## 3.7 Ohjelmointirajapinta (API)

Kuviossa 18 on tekstieditorissa näkyvä C#-skripti, joka on luotu Unity-editorissa. Skriptin nimi on GO.cs ja se sisältää oletuksena kuvassa näkyvät rivit.

GO.cs

ection

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class GO : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
17
```

KUVIO 18. Skriptin perusrakenne

MonoBehaviour mahdollistaa muiden Unityn sisäisten luokkien sekä funktioden toimimisen skriptissä (kuvio 18). Skriptin on periydyttävä MonoBehaviour-luokasta, jotta skriptin voi lisätä peliobjektiin

komponenttina. Kun uusi skripti luodaan, on siinä oletuksena tapahtumafunktiot Start ja Update. Start-funktiota voidaan kutsua alustusfunktioksi ja Updatefunktiota päivitysfunktioksi.

### 3.8 GUI-luokka

GUI-luokka mahdollistaa käyttöliittymän tekemisen skriptaamalla. Siinä ei ole visuaalista työkalua joka helpottaisi käyttöliittymän tekemistä.

Vahvuudet

- helppo ottaa käyttöön, ei vaadi lisäosia
- yksinkertainen perusidea
- hyvä dokumentaatio (Unity Scripting API)
- paljon apufunktioita (Unity Scripting API).

Heikkoudet

- kaikki toiminnallisuus tehtävä itse (valikot)
- suorituskyky. (Jokaisella elementillä oma piirtokutsu).

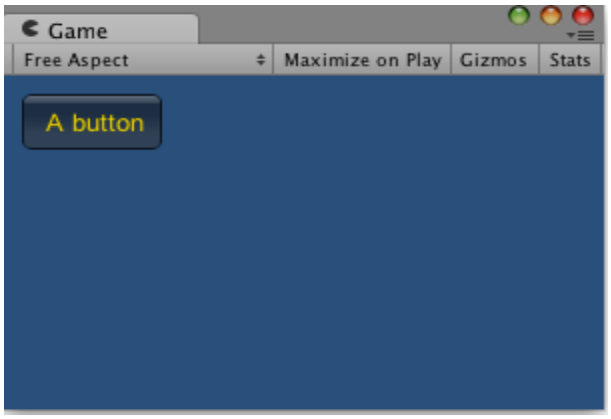
Unityn GUI-luokka piirtää käyttöliittymän elementit OnGUI-funktiossa. Sitä kutsutaan jokaisella framella aivan kuten Update-funktiotakin. Kaikki GUI renderöinti täytyy tapahtua joko OnGUI-funktion sisällä tai funktioissa, joita OnGUI-funktio kutsuu. Unityn sisäänrakennettu GUI-luokka onkin helppo ottaa käyttöön. Sen yksinkertainen perusidea mahdollistaa käyttöliittymän tekemisen aloittamisen nopeasti.

Unityn GUI-luokka ei vaadi lisäosia tai ennakkovalmisteluja projektin asetuksiin. GUI-luokka sisältää paljon ominaisuuksia hyvän käyttöliittymän tekemiseen. Näitä ominaisuuksia ovat esimerkiksi funktiot yleisimpien käyttöliittymäelementtien luomiseen. Elementtien lisäksi GUI-luokassa on paljon apufunktioita. Näillä funktioilla voidaan esimerkiksi järjestellä elementtejä ryhmiin ja piirtää tekstuureita.

Unityn GUI-luokka on hyvä pohja luoda yksinkertaisia GUI-elementtejä. Se on kuitenkin vain pohja. Jos käyttöliittymään haluaa toteuttaa monimutkaisempia ominaisuuksia, on kaikki toiminnallisuus tehtävä itse. Moderneissa käyttöliittymissä on monia ominaisuuksia, joiden toteuttaminen GUI-luokalla on vaikeaa ja paljon aikaa vievää. Esimerkiksi rullaavat valikot tai elementistä toiseen siirrettävät kuvakkeet voivat kuulostaa yksinkertaiselta asialta, mutta ne ovat monimutkaisia toteuttaa.

Alapuolella käydään läpi GUI-luokkaa läpi muutaman esimerkin avulla. Ensimmäisessä esimerkissä tehdään nappi (kuvio 19), jonka sisällön väri asetetaan keltaiseksi ja paikka asetetaan parametreilla x, y, leveys ja korkeus (kuvio 20).

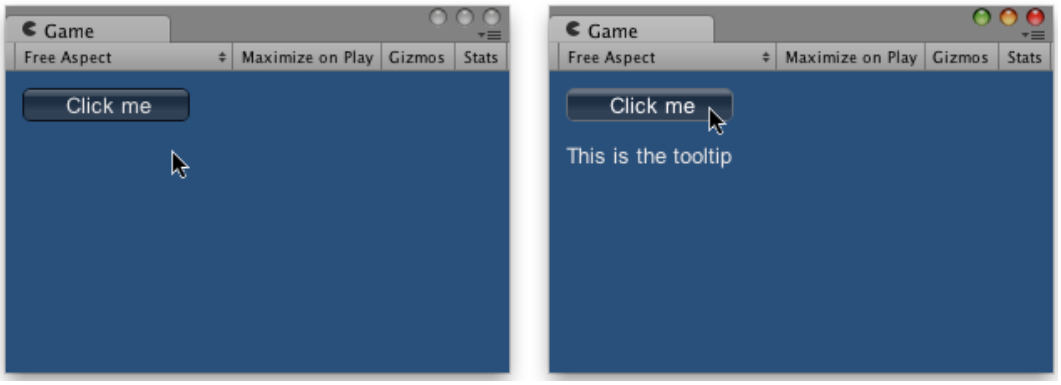
Toisessa esimerkissä on nappi (kuvio 21), jonka sisältöparametriksi vaihdetaan GUIContent-objekti (kuvio 22). GUIContent-objektilla näytetään vihjelaatikko käyttäjälle.




KUVIO 19. Pelinäkymä, jossa on nappi

```
16    void OnGUI(){
17            GUI.contentColor = Color.yellow;
18            GUI.Button(new Rect(10, 10, 70, 30),"A button");
19    }
```

KUVIO 20. Nappiin liitetty skripti




KUVIO 21. Nappi ja vihjelaatikko

```
16 void OnGUI(){
17
18     GUI.Button(new Rect(10, 10, 100, 20),new GUIContent("Click me", "This is the tooltip"));
19     GUI.Label (new Rect (10, 40, 100, 40), GUI.tooltip);
20 }
```

KUVIO 22. Vihjelaatikollisen napin skripti

# 4 HIUSTYYLIGENERAATTORI

Hiustyyligeneraattori toteutettiin Android 2D -mobiilisovelluksena. Sovellus toteutettiin C# ohjelmointikielellä. Pelin käyttöliittymä on totetutettu GUI-texture peliobjekteilla, joihin lisättiin tarvittavat komponentit (texture, script). Tässä projektissa ei käytetty Unityn valmiita tekstuuri- tai skriptipaketteja.

Aikaisempaa kokemusta ohjelmoinnista ja pelien kehittämisestä oli saatu koulun kursseilla. Mobiilisovellusten kehittämistä oli tehty koulun projekteissa. Perusteet olivat siis hallussa, mutta Unity oli täysin uusi asia opinnäytetyön tekemistä aloitettaessa. Toimeksiantajan kanssa sovittiin niin, että sovellus olisi ensimmäinen versio, jota voisi myöhemmin jatkokehittää.

## 4.1 Sovelluksen kulku

Sovellus koostuu kahdesta näkymästä. Ensimmäisessä näkymässä käyttäjä ottaa kuvan kuvattavasta henkilöstä ja toisessa näkymässä tapahtuu kaikki muu toiminnallisuus.

Käyttöliittymän päätarkoitus on kuvan ottaminen ja hiuskuvan muokkaaminen. Selvennetään käyttöliittymän toimintaa yksinkertaisen esimerkkikäyttötapauksen avulla.

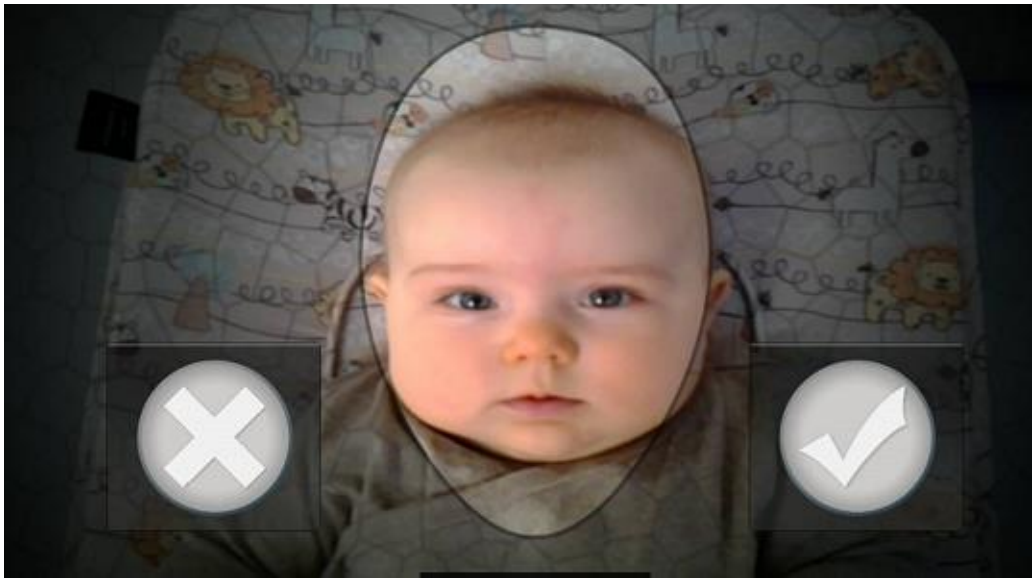
### 4.1.1 Alkutilanne

Sovelluksen käynnistyessä (kuvio 23) ruudulle ilmestyy kohdistin, laukaisin nappi ja taustalla näkyy laitteen kameran syöte. Käyttäjän tehtävänä on kohdistaa kohdehenkilön kasvot kohdistimella ja painaa laukaisinta.

KUVIO 23. Sovelluksen käynnistymisnäkymä

### 4.1.2 Kuvan hyväksyminen

Kun käyttäjä on painanut laukaisinta, niin taustalla näkyvä kameran syöte pysähtyy ja laukaisin nappi vaihtuu "hyväksy"- ja "hylkää" nappeihin (kuvio 24). Tässä vaiheessa käyttäjä voi valita hyväksyykö hän otetun kuvan painamalla "hyväksy" nappia. Jos käyttäjä painaa "hylkää" nappia niin sovellus palaa alkutilanteeseen.



KUVIO 24. Sovelluksen näkymä kuvan ottamisen jälkeen

### 4.1.3 Hiuskuvan valinta ja muokkaus

Kun käyttäjä on painanut "hyväksy" nappia, aukeaa sovelluksen varsinainen käyttöliittymä (kuvio 25), jossa kohdehenkilö voi valita itselleen hiustyylin (kuvio 26), muokata hiuskuvan kokoa, liikuttaa hiuskuvaa ja vaihtaa hiuskuvan väriä (kuvio 27).

KUVIO 25. Sovelluksen käyttöliittymä

KUVIO 26. Sovelluksen käyttöliittymä

KUVIO 27. Sovelluksen käyttöliittymä

## 4.2 Käyttöliittymä

Käyttöliittymän suunnittelussa oli tavoitteena tehdä siitä selkeä ja helposti omaksuttava, sillä hiusgeneraattorin käyttö haluttiin tehdä mahdollisimman helpoksi ja nopeaksi käyttäjälle. Käyttäjäryhmän mahdollinen vähäinen kokemus mobiilisovelluksista otettiin myös huomioon suunnitteluvaiheessa.

## 4.3 MenuHandler

MenuHandler-luokalla (kuvio 28) hallitaan sitä, mitkä käyttöliittymän osat käyttäjä näkee kerrallaan. Käyttöliittymän eri osilla tarkoitetaan sovelluksen valikoita.

Sovelluksen käyttöliittymä on jaettu osiin tagien avulla käyttäen GameObject.FindGameObjectWithTag()-funktioita. Tämän funktion avulla voidaan hakea näkymästä tietyn nimisiä GameObject-olioita.

```
4  public class MenuHandler : MonoBehaviour {
5      private GameObject topBlock2, leftBlock, topBlock;
6      // Use this for initialization
7      void Start () {
8          topBlock2 = GameObject.FindGameObjectWithTag ("TopBlock2");
9          leftBlock = GameObject.FindGameObjectWithTag ("LeftBlock");
10         topBlock = GameObject.FindGameObjectWithTag ("TopBlock");
11         topBlock.SetActive (false);
12         topBlock2.SetActive (false);
13         leftBlock.SetActive (true);
14         DontDestroyOnLoad (this);
15     }
16     // Update is called once per frame
17     void Update () {
18     }
19     public void OpenMenu(int i){
20         //
21         if (i == 0) {
22             if (topBlock.activeSelf == true) {
23                 topBlock.SetActive (false);
24             } else {//open topmenu 1, close topmenu 2
25                 topBlock.SetActive (true);
26                 leftBlock.SetActive(true);
27                 topBlock2.SetActive (false);
28             }
29         }
30         else if (i == 1) {
31             if (topBlock2.activeSelf == true)
32                 topBlock2.SetActive (false);
33                 leftBlock.SetActive (true);
34             } else {//open topmenu 2, close topmenu 1
35                 topBlock2.SetActive (true);
36                 topBlock.SetActive(false);
37                 leftBlock.SetActive(false);
38             }
39         }
40     }
```

KUVIO 28. MenuHandler-luokka

MenuHandler-luokan openMenu-funktion muuttujan arvo voi olla 0 tai 1. Tämä määrittää sen kumpi topblock käyttäjälle näkyy eli onko näkyvissä hiusmallit vai värit. OpenMenu-funktion arvo määritetään erillisessä skriptissä, (kuvio 29) jossa MenuHandler-luokan openMenu-funktion muuttujan arvolle annetaan arvoksi 0

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class HairMenuOpen : MonoBehaviour {
5     private MenuHandler menuHandler;
6     // Use this for initialization
7     void Start () {
8         menuHandler = Camera.main.GetComponent<MenuHandler> ();
9     }
10    // Update is called once per frame
11    void Update () {
12    }
13    void Awake(){
14    }
15    void OnMouseDown(){
16
17        menuHandler.OpenMenu (0);
18    }
19 }
```

KUVIO 29. HairMenuOpen luokka

Käyttöliittymä on jaettu kolmeen osaan (kuvio 30). Näytön keskelle (Kohta 1 topblock/topblock2) avautuu hius- tai värivalikko (kuviot 31, 32). Vasemmalle reunalle (kohta 2 leftblock) ilmestyy navigointinäppäimet, (kuvio 33), joilla hiusmallia voi säätää tarkemmin käyttäjän kasvoihin sopivaksi. Oikealle puolelle (kohta 3 rightblock) ilmestyy kolme nappia, joilla käyttäjä voi avata ja sulkea sovelluksen valikoita (kuvio 34). Lisäksi oikeassa yläkulmassa sijaitsee sovelluksen sulkemisnappi.


KUVIO 30. Sovelluksen käyttöliittymä jaettuna osiin

KUVIO 31. Topblock

KUVIO 32. Topblock2

KUVIO 33. Leftblock


KUVIO 34. Rightblock

Oikean valikon näppäimillä (kuvio 34) navigoidaan sovelluksen käyttöliittymässä. A kohtaa painettaessa ylälaitaan ilmestyy hiusvalikko, ja alalaitaan avautuu kaksi näppäintä joilla voi valita selattavan hiustyylin (lyhyet ja pitkät). B kohtaa painamalla käyttäjälle avautuu värivalikko, jossa pystyy vaihtamaan käytössä olevan hiuskuvan väriä. C kohtaa käytetään tulevaisuudessa kuvan tallentamiseen.

## 4.4 Kuvan ottaminen

Kun käyttäjä painaa laukaisinta, sovellus ottaa laitteen kameran syötteestä näytön kokoisen kuvakaappauksen, (kuvio 35) joka sijoitetaan Texture2D-muuttujaan ja lopulta GetPicture-luokan setSnapshot-metodille.

```
13      public Texture2D picture;
14      private getPicture gpClass;


67      void TakeSnapshot()
68      {
69          Texture2D snap = new Texture2D(mCamera.width, mCamera.height);
70          snap.SetPixels(mCamera.GetPixels());
71          snap.Apply();
72
73          picture = snap;
74          gpClass.setSnapshot (picture);
75      }
```

KUVIO 35. Kameran syötteen kuvankaappaus.

GetPicture-luokkaa (kuvio 36) käytetään kuvan tuomiseen näkymästä toiseen. DontDestroyOnLoad-funktiota käytetään siksi, että muuttuja ei tuhoutuisi seuraavan näkymän latauksen yhteydessä.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class GetPicture : MonoBehaviour {
5     private Texture2D snapShot;
6     // Use this for initialization
7     void Start () {
8
9     }
10    void Awake(){
11        DontDestroyOnLoad (this);
12    }
13
14    public Texture2D getSnapshot(){
15        return this.snapShot;
16    }
17    public void setSnapshot(Texture2D snapshot){
18        this.snapShot = snapshot;
19    }
20 }
21
```

KUVIO 36. GetPicture- luokka

SetMainWindowBackground-luokassa (kuvio 37) Texture2D-muuttuja asetetaan sovelluksen käyttöliittymän taustakuvaksi käyttäen FindGameObjectWithTag-funktiota ja GetPicture-luokan GetSnapshot-metodia.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class SetMainWindowBackground : MonoBehaviour {
5     private getPicture gpClass;
6     GameObject background;
7     Texture2D savedSnapshot;
8     // Use this for initialization
9     void Start () {
10        background = GameObject.FindGameObjectWithTag("MainWindowBackground");
11        savedSnapshot = gpClass.getSnapshot ();
12        background.GetComponent<GUITexture>().texture = savedSnapshot;
13
14    }
15
16    // Update is called once per frame
17    void Update () {
18
19    }
20    void OnGUI () {
21
22    }
23 }
```

KUVIO 37. SetMainWindowBackground-luokka

## 4.5 Hiusvalikko

Hiusvalikko jaettiin kahteen osaan, lyhyet ja pitkät hiukset (kuvio 38). Hiuskuvien jako tehtiin siksi, että käyttäjä löytäisi halutun hiusmallin nopeammin.

KUVIO 38. Hiusvalikko

Hiuskuvien haku (kuvio 39) tapahtuu käyttäen Resources.LoadAll-funktiota. Tämä toimii kuitenkin vain, jos hiuskuvat on tallennettu Resources-kansioon. Tämä tapa osoittautui hyväksi tavaksi hakea kuvia dynaamisesti sovelluksen ollessa käynnissä. Resources-kansio ei tue itsestään alakansioita, vaan LoadAll-funktiolle täytyy antaa kansion polku.

```
57      public void hairMenus(int i){
58          if(i == 0)
59              hairStyles = Resources.LoadAll<Texture2D>("Images/HairstylesShort");
60          else if(i == 1)
61              hairStyles = Resources.LoadAll<Texture2D>("Images/HairstylesLong");
62          }
```

KUVIO 39. Hiuskuvien haku

Hiusvalikossa näkyy kolme hiuskuvaa kerrallaan, joita selataan nuolinäppäimillä. Nuolinäppäimiin kiinnitettiin incrementIndex- ja reduceIndex-funktiot (kuvio 40), jotka toteuttavat varsinaisen toiminnallisuuden.

```
34      public void incrementIndex(){
35          if (indexZero < hairStyles.Length - 3)
36                          indexZero += 1;
37                  else
38                          indexZero = 0;
39      }
40      public void reduceIndex(){
41          if (indexZero > 1)
42              indexZero -= 1;
43          else
44              indexZero = 0;
45      }
46


63      void OnGUI(){
64
65          hairButton1.GetComponent<GUITexture>().texture = hairStyles [indexZero];
66          hairButton2.GetComponent<GUITexture>().texture = hairStyles [indexZero + 1];
67          hairButton3.GetComponent<GUITexture>().texture = hairStyles [indexZero + 2];
68      }
```

KUVIO 40. IncrementIndex- ja reduceIndex-funktiot

## 4.6 Värivalikko

Värivalikossa (kuvio 41) käyttäjä voi vaihtaa hiuskuvan väriä tai palauttaa alkuperäisen hiusvärin. Toisin kuin hiusvalikossa, värivalikossa ei ole ruudun alaosassa valintanappeja ja kaikki värivaihtoehdot näkyvät samaan aikaan käyttäjälle.

KUVIO 41. Värivalikko

Sovelluksen värivalikon (kuvio 42) alustukseen käytetään FindGameObjectWithTag-funktiota, jolla etsitään haluttu peliobjekti, jolle on luotu oma tagi. OnMouseDown-funktiolla määritetään peliobjektille uusi väri r, g, b, a komponenteilla. Vasemmassa yläkulmassa olevalla napilla voi palauttaa alkuperäisen värin.

```
7     void Start () {
8         hairPicture = GameObject.FindGameObjectWithTag ("Hair");
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15    void OnMouseDown(){
16        hairPicture.GetComponent<GUITexture>().color = new Color (0.6F, 0.6F, 0.6F, 0.8F);
17    }
18 }
```

KUVIO 42. Värinapin skripti

## 4.7 Hiuskuvan säätö

Sovelluksen alussa oleva tähtäin asettaa kuvatun henkilön kasvot oikeaan kohtaan. Henkilöiden erilaisten kasvonpiirteiden takia valittavan hiusmallin tarkempi kohdennus saattaa olla kuitenkin tarpeen (kuviot 43, 44). Tätä toiminnallisuutta varten luotiin käyttöliittymän vasempaan reunaan hiuskuvalle omat säätönäppäimet.

KUVIO 43. Säätämätön hiuskuva

KUVIO 44. Säädetty hiuskuva

### 4.7.1 Hiuskuvan zoomaus

Hiuskuvan zoomaus toteutettiin OnMouseDown-funktiolla, jota kutsutaan käyttäjän näppäimen painalluksesta. Näppäimiin (kuvio 45) on kiinnitetty ZoomIn- ja ZoomOut-skriptit (kuviot 46, 47). Kun käyttäjä painaa nappia peliobjektin transform-komponentin arvo muuttuu käyttäen x, y, z komponentteja.

KUVIO 45. Zoomaus näppäimet

```
4 public class ZoomIn : MonoBehaviour {
5     private GameObject zoomThis;
6     // Use this for initialization
7     void Start () {
8         zoomThis = GameObject.FindGameObjectWithTag ("Hair");
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15    void OnMouseDown(){
16        zoomThis.transform.localScale += new Vector3(0.05F, 0.05F, 0);
17    }
18 }
```

KUVIO 46. ZoomIn-luokka

```
4 public class ZoomOut : MonoBehaviour {
5     private GameObject zoomThis;
6     float Max = 1; float Min = 0;
7     // Use this for initialization
8     void Start () {
9         zoomThis = GameObject.FindGameObjectWithTag ("Hair");
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16    void OnMouseDown(){
17        if (zoomThis.transform.localScale.x - 0.05f > 0) {
18                zoomThis.transform.localScale -= new Vector3 (0.05F, 0.05F, 0);
19        }
20    }
21 }
```

KUVIO 47. ZoomOut-luokka

### 4.7.2 Hiuskuvan liikuttaminen

KUVIO 48. Hiuskuvan liikuttaminen

Hiuskuvan liikuttaminen (kuvio 48) toteutettiin myös OnMouseDown-funktiolla, jota kutsutaan käyttäjän näppäimen painalluksesta. Näppäimiin on kiinnitetty omat skriptit (kuviot 49, 50, 51, 52). Kun käyttäjä painaa nappia peliobjektin transform-komponentin arvo muuttuu.

```
4 public class MoveDown : MonoBehaviour {
5     private GameObject moveThis;
6     // Use this for initialization
7     void Start () {
8         moveThis = GameObject.FindGameObjectWithTag ("Hair");
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15    void OnMouseDown(){
16        moveThis.transform.localPosition += new Vector3 (0,-0.01F,0);
17    }
18 }
```

KUVIO 49. MoveDown-luokka

```
4 public class MoveLeft : MonoBehaviour {
5     private GameObject moveThis;
6     void Start () {
7         moveThis = GameObject.FindGameObjectWithTag ("Hair");
8     }
9
10    // Update is called once per frame
11    void Update () {
12
13    }
14    void OnMouseDown(){
15        moveThis.transform.localPosition += new Vector3 (-0.01F,0,0);
16    }
17 }
```

KUVIO 50. MoveLeft-luokka

```
4 public class MoveRight : MonoBehaviour {
5     private GameObject moveThis;
6     void Start () {
7         moveThis = GameObject.FindGameObjectWithTag ("Hair");
8     }
9
10    void Update () {
11
12    }
13    void OnMouseDown(){
14        moveThis.transform.localPosition += new Vector3 (0.01F,0,0);
15    }
16 }
```

KUVIO 51. MoveRight-luokka

```
4 public class MoveUp : MonoBehaviour {
5     private GameObject moveThis;
6     // Use this for initialization
7     void Start () {
8         moveThis = GameObject.FindGameObjectWithTag ("Hair");
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15    void OnMouseDown(){
16        moveThis.transform.localPosition += new Vector3 (0,0.01F,0);
17    }
18 }
```

KUVIO 52. MoveUp-luokka

## 4.8 Hiuskuvan tekstuurit

Hiuskuvat (kuviot 53, 54) kuvasivat kaksi hiusalan opiskelijaa. Hiuskuvien valokuvaamisessa haastavaa oli samanlaisen valotuksen luominen ja oikeanlaisten hiusmallin omaavien hiusmallien löytäminen.

Hiuskuvia otettiin noin 100 kappaletta, joista vain osa todettiin toimiviksi sovellukseen. Hylkäykset johtuivat valokuvien erilaisista valotuksista ja ihmisten erilaisista hiusrajoista.

KUVIO 53. Sovelluksessa käytettyjä pitkiä hiusmalleja

KUVIO 54. Sovelluksessa käytettyjä lyhyitä hiusmalleja

## 4.9 Testaus

Unityn ympäristössä pystyy ajamaan ohjelman, joten aluksi testausta suoritettiin suoraan tietokoneelta. Sovellusta testattiin aina, kun koodiin tehtiin jonkinlaisia muutoksia. Mitään erillistä testaustyökalua työssä ei käytetty vaan testausta suoritettiin kokeilemalla mikä toimii ja mikä ei. Mikäli jokin asia ei toiminut, sitä työstettiin, että sovellus saatiin toimimaan halutulla tavalla. Myöhemmin koululta saatiin tabletti, jolla päästiin testaamaan toiminnallisuutta tarkemmin laitteen luonnollisessa käyttötarkoituksessa.

Ensimmäisissä tabletilla tehdyissä testauksissa havaittiin käyttöliittymän selkeyttämisen tarve, kun siirryttiin pienemmälle näytölle. Testausta suoritettiin tabletilla aina uuden toiminnallisuuden teon jälkeen. Tällä varmistettiin, että uusi toiminnallisuus toimii myös pienemmällä kosketusnäytöllä.

Viimeinen testausvaihe suoritettiin ensimmäisen version valmistuttua. Testausta suoritettiin pienellä kohderyhmällä. Testauksen tuloksena saatiin tietoja, joita käytettiin sovelluksen käyttäjäkokemuksen parantamiseen.

Parannuksia tehtiin mm. käyttöliittymän painikkeisiin, joista tehtiin suuremmat. Hiuskuvien valikkoon tehtiin erilliset osiot lyhyille ja pitkille hiuksille. Tällä tavoin sovelluksen käyttäjä löytää halutun hiusmallin nopeammin. Värivalikon väripaletin painikkeita muokattiin isommiksi, jotta haluttu väri olisi helpompi valita.

# 5 YHTEENVETO

Mobiilisovelluksen kehittämisessä tabletille on useita eli aspekteja, joita asiaan tutustumaton ei välttämättä tiedosta. Mobiililaitteille sovelluksen kehittäminen tuo mukanaan niin rajoituksia kuin vapauksiakin. Pääosin kosketusnäytöllä toteutettavat kontrollit on rajoitettava muutamaan eri painikkeeseen tai eleeseen, muuten näytön alasta peittyy liian suuri osuus, eikä käyttäjä enää näe tarpeeksi.

Opinnäytetyön aiheena oli hiustyyligeneraattori mobiilisovelluksen kehittäminen Unity-pelimoottorilla. Hiustyyligeneraattori vaikutti aluksi haastavalta projektilta, koska aiempaa kokemusta pelimoottoreista ei ollut. Unityn manuaalin ohjeet tarjosivat kuitenkin hyvät lähtökohdat projektin aloitukselle. Manuaalista löytyi kattavasti tietoa kaikista Unityssä käytetyistä luokista ja funktioista esimerkkeineen, joten itseopiskelu oli verrattain helppoa.

Tavoite oli, että sovelluksella käyttäjä voi ottaa kohdehenkilöstä kuvan ja mallintaa hänelle erilaisia hiustyylejä ja värejä. Asetettu tavoite saavutettiin.

Projektin aikana saatiin paljon kokemusta mobiilisovelluksen elinkaaresta aina suunnittelusta toimivaan sovellukseen. Projekti oli mielenkiintoinen ja sen aikana opittiin paljon myös käyttöliittymän suunnittelusta ja pelinkehityksestä Unity-pelimoottorilla.

Hiustyyligeneraattori-projektissa työskentely oli mukavaa, mutta myös haastavaa. Isoin haaste oli aikatauluissa pysyminen, toiminnallisuudesta tinkimättä, sekä käyttöliittymän toteutus pienelle näytölle. Jos projekti aloitettaisiin uudestaan nyt, aikaa käytettäisiin enemmän käyttöliittymän suunnitteluun ennen koodin kirjoittamisen aloitusta. Paremmalla suunnittelulla oltaisiin käyttöliittymän ongelmat huomattu aiemmin, eikä koodia olisi tarvinnut kirjoittaa uudestaan.

Tulevaisuudessa voitaisiin hyödyntää opittuja asioita uusien pelien kehittämiseen Unityllä ja mahdollisesti myös tämän projektin jatkokehitykseen.

Sovelluksen jatkokehityksen mahdollisuuksia olisi useita. Yhtenä esimerkkinä voitaisiin pitää asiakkaan yhteystietojen tallentamista tietokantaan.

LÄHTEET

Koivunen, M-R. & Nieminen, M. 1996. Ohjelmiston käytettävyys. Teoksessa Kalimo Anna (toim.) Graafisen käyttöliittymän suunnittelu. Espoo: Suomen ATK-kustannus Oy, 22-24.

Kortum, P. 2008. HCI beyond the GUI: design for haptic, speech, olfactory and other nontraditional interfaces. Burlington: Morgan Kaufman publishers.

Kuutti, W. 2003. Käytettävyys, suunnittelu ja arviointi. Saarijärvi: Gummerus.

Nielsen, J. 1993. Usability engineering. San Diego. Academic Press.

Sinkkonen, I. Kuoppala, H. Parkkinen, J & Vastamäki, R. 2006. Käytettävyyden Psykologia. Helsinki: Edita Publishing Oy.

Unity Technologies 2017a, Console [viitattu 15.2.2017]. Saatavissa: https://docs.unity3d.com/Manual/Console.html

Unity Technologies 2017b, GameObject [viitattu 13.1.2017]. Saatavissa: https://docs.unity3d.com/Manual/class-GameObject.html

Unity Technologies 2017c, Hierarchy [viitattu 10.1.2017]. Saatavissa: https://docs.unity3d.com/Manual/Hierarchy.html

Unity Technologies 2017d, MonoBehaviour [viitattu 15.4.2017]. Saatavissa: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Unity Technologies 2017e, Project view [viitattu 10.1.2017]. Saatavissa: https://docs.unity3d.com/Manual/ProjectView.html

Unity Technologies 2017f, Scenes [viitattu 10.1.2017]. Saatavissa: https://docs.unity3d.com/Manual/CreatingScenes.html

Unity Technologies 2017g, Using the inspector [viitattu 12.1.2017]. Saatavissa: https://docs.unity3d.com/Manual/UsingTheInspector.html

Unity Technologies 2017h, Using the scene view [viitattu 10.2.2017]. Saatavissa: https://docs.unity3d.com/Manual/UsingTheSceneView.html