Android-mobiilipelin kehittäminen Unityllä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittely, Hämeenlinna

Kevät 2023

Miska Vuorinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittely Tiivistelmä

Hämeenlinna

Tekijä Miska Vuorinen Vuosi 2023

Työn nimi Android-mobiilipelin kehittäminen Unityllä

Ohjaajat Esa Huiskonen

TIIVISTELMÄ

Tämän opinnäytetyön idea syntyi kiinnostuksesta mobiilipelien kehittämistä kohtaan. Opinnäytetyön tekijällä on kokemusta Android-sovellusten kehittämisestä Android Studio -kehitysympäristössä ja tietokonepelien kehittämisestä Unity-kehitysympäristössä.

Tavoitteena oli kehittää mobiilipeli, jonka keskeiseksi kehitysympäristöksi valikoitui Unity. Työssä päätettiin käyttää Googlen Firebase -alustan tarjoamaa Cloud Firestore -tietokantapalvelua ja tarkoituksena oli selvittää, että kuinka kyseinen palvelu toimii Unityn kanssa. Työssä haluttiin myös selvittää, että miten käyttöliittymäsuunnittelu vaikuttaa pelin pelattavuuteen.

Työssä käydään läpi asiaa sovelluskehityksestä sekä käyttöliittymäsuunnittelusta, jonka jälkeen syvennytään Android-pelien kehittämiseen. Käytännön osuudessa käydään läpi pelin suunnittelu ja toteutus. Lopputuloksena työ antaa ohjeen, miten saadaan aloitettua mobiilipelien kehittäminen Unityllä hyödyntäen Google Firestore -tietokantapalvelua.

Yksinkertaisen mobiilipelin kehittämisestä saatiin ymmärrys käyttöliittymäsuunnittelun tärkeydestä. Selväksi tuli myös se, että Unity on sopiva ympäristö Android-pelien kehittämiseen ja Cloud Firestore -palvelun hyödyntäminen Unityllä oli helppoa.

Avainsanat Mobiilipelit, Android, Unity, Käyttöliittymäsuunnittelu

Sivut 39 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna

Abstract

Author Miska Vuorinen
Year 2023
Subject Android mobile game development using Unity
Supervisors Esa Huiskonen

ABSTRACT

The idea of this thesis was born by interest in mobile game development. The background is experience in developing Android apps with Android Studio and development of computer games in Unity development environment.

The aim was to develop mobile game using Unity as a key development enviroment. It was decided to use Cloud Firestore database service in the thesis and the purpose was to find out how the service in question works with Unity. One of the objectives was also to figure out how user interface design affects gameplay.

Initially, the thesis contains information about software development and user interface after which thesis focuses on the development of Android games. In the practical part of the thesis game design and game development is reviewed. As a result, the thesis provides instructions of how to start developing mobile games with Unity using Google Firestore database service.

Development of simple mobile game clarified importance of user interface design. It was also clear that Unity was an appropriate environment for Android game development and utilization of Cloud Firestore service was easy with Unity.

Keywords Mobile games, Android, Unity, User interface design
Pages 39 pages and appendices 1 page

Sanasto

Figma	Käyttöliittymän suunnittelutyökalu
Unity	Unity Technologiesin kehittämä pelimoottori, jolla on mahdollista luoda kaksi- tai kolmiulotteisia videopelejä monelle eri alustalle.
Google Firebase	Googlen alusta, joka sisältää palveluja sovelluskehitystä varten.
Cloud Firestore	Google Firebasessa oleva dokumenttiorientoitunut tietokantapalvelu mobiili- ja web-sovelluksia varten.
UI	User interface tarkoittaa käyttöliittymää.
UX	User experience tarkoittaa käyttökokemusta.
SDK	Software Development Kit on työkalusarja ohjelmistokehitystä varten yhdessä asennettavassa paketissa.
JDK	Java Development Kit on kehitysympäristö, joka sisältää työkalut Java-ohjelmien kehittämiseen.
NDK	Native Development Kit on työkalusarja, jonka avulla voidaan toteuttaa sovelluksen osa C tai C++ -kielellä ja kääntää se natiivikoodiksi Androidille.
NoSQL	Perinteisestä relaatiomallin tietokannasta poikkeava tietokanta.
Kanban	Työkalu työn tehokkuuden optimointiin, jonka keskeisin periaate on työn kulun visualisointi Kanban-taululle.

Sisällys

Kuvat, taulukot ja kaavat

Liitteet

1 Johdanto

Älypuhelimien sovelluskaupat ovat pullollaan erilaisia pelejä ja niitä ilmestyy lisää päivittäin. Onkin selvää, että mobiilipelien kehittäminen on vakavasti otettava liiketoiminnan ala tänä päivänä. Tässä opinnäytetyössä tutustutaan mobiilipelien kehittämiseen erityisesti Android-laitteille ja kehitetään valmis mobiilipeli sillä ajatuksella, että tekniikoita voidaan hyödyntää myös myöhemmin.

Aluksi työssä käydään läpi asiaa sovelluskehityksestä ja käyttöliittymäsuunnittelusta, jonka jälkeen käydään läpi Android-pelien kehitysprosessi ja mitä erityispiirteitä liittyy mobiilipelien kehittämiseen Androidille. Tämän lisäksi tutustaan käyttöliittymään ja käyttökokemukseen, jotka yhdessä vaikuttavat pelin tuottamaan pelikokemukseen. Työkalut ja menetelmät, joita työssä käytetään, tullaan esittelemään työn teoriaosuudessa. Näihin lukeutuu Kanban-menetelmä, Figma-suunnittelutyökalu, Unity-kehitysympäristö ja Googlen Firebase-alustan tarjoama Cloud Firestore -tietokantapalvelu.

Pelin kehittäminen aloitetaan suunnittelulla, johon lukeutuu idean keksiminen ja käyttöliittymäsuunnitelman luominen. Tämän jälkeen luodaan tarvittavat projektit kehitysalustoille ja tehdään muut tarpeelliset alustavat toimenpiteet. Lopuksi varsinainen pelin toteutus tehdään Unity-kehitysympäristössä, joka yhdistetään Cloud Firestore -tietokantaan.

Keskeisimmät tutkimuskysymykset:

- Miten käyttöliittymäsuunnittelu vaikuttaa pelin pelattavuuteen?
- Miten voidaan tehdä peli Androidille hyödyntäen Unity-pelimoottoria?
- Miten Cloud Firestore -tietokantaa voidaan hyödyntää pelissä, joka tehdään Unityllä?

2 Sovelluskehitys

Sovellus on tietokoneella, kuten älypuhelimella, tietokoneella tai älykellolla suoritettava ohjelma tai ohjelmien kokonaisuus, jolla pyritään suorittamaan tietty tai useita tiettyjä tehtäviä. Näiden sovellusten rakennusprosessia kutsutaan Sovelluskehitykseksi. (Ite wiki, 2019)

Mobiilisovellus voidaan luokitella natiivi-, web- tai hybridisovellukseksi riippuen siitä, että kuinka koodi on kirjoitettu. Näitä sovellustyyppejä voidaan käyttää myös kuvaamaan työpöytäsovelluksia. Natiivisovellus on sovellus, joka kohdennetaan tietylle laitteelle, jolloin sovelluksen täytyy olla yhteensopiva kohdelaitteen käyttöjärjestelmän kanssa. Web-sovellus on sovellus, jota käytetään tyypillisesti verkkoselaimen kautta. (TechTarget, 2021)

Natiivisovelluksista poiketen hybridisovellus toimii yhdellä ja samalla koodipohjalla eri käyttöjärjestelmissä. Hybridisovellukset paketoidaan natiivikuoreen, jolloin ne voivat rajapintojen avulla käyttää puhelimien natiiviominaisuuksia kuten kameraa tai bluetoothia. Yhden koodipohjan ansioista sovelluksen päivittäminen helpottuu siten, että ei tarvitse kuin muokata yhtä koodia, jolloin koodi päivittyy kaikille alustoille. (Tecinspire Oy, 2022)

Käytännössä sovelluksen rakennusprosessi sisältää yleensä ainakin seuraavat vaiheet: konseptointi, määrittely, suunnittelu, toteutus, testaus, käyttöönotto ja ylläpito. Rakennusprosessia varten on luotu useita valmiita malleja, kuten vesiputousmalli, nopean kehittämisen malli ja ketterät menetelmät. (Hurja, 2021)

2.1 Vesiputousmalli

Vesiputousmalli on jaettu selkeisiin vaiheisiin: vaatimusten kerääminen, suunnittelu, toteutus, testaaminen, käyttöönotto ja ylläpito. Jokainen vaihe täytyy tehdä valmiiksi ennen kuin voi siirtyä seuraavaan vaiheeseen. (Tutorials Point, n.d.-c)

Vaatimusten keräämisvaiheessa tavoitteena on ymmärtää täsmälliset vaatimukset asiakkaalta ja dokumentoida ne asianmukaisesti. Vaatimukset analysoidaan ja ne

dokumentoidaan ohjelmiston vaatimusmäärittely -dokumenttiin, joka toimii sopimuksena kehitystiimin ja asiakkaan välillä. (Pal, 2018)

Suunnitteluvaiheessa ohjelmiston vaatimusmäärittelyssä olevat vaatimukset muunnetaan muotoon, jossa kyseiset vaatimukset ovat ohjelmoitavissa ohjelmointikielellä. Lopputuloksena on ohjelmistosuunnitteluasiakirja, joka sisältää korkeatasoisen ja yksityiskohtaisen suunnitelman sekä koko ohjelmistoarkkitehtuurin. (Pal, 2018)

Toteutusvaiheessa suunnitelman mukaiset moduulit koodataan itsenäisinä kokonaisuuksina. Moduuleille suoritetaan yksikkötestit, jotta voidaan todeta, että toimivatko moduulit tarkoitetulla tavalla vai eivät. (Pal, 2018)

Testausvaihe alkaa, kun kaikki moduulit ovat koodattu ja yksikkötestit ovat menneet läpi. Tämän jälkeen moduulien yhdistäminen aloitetaan. Moduulien yhdistämisen jälkeen koko järjestelmä testataan. Kun kaikki testaukset ovat tehty, tuote otetaan käyttöön joko asiakkaan ympäristössä tai tuote julkaistaan markkinoille. (Tutorials Point, n.d.-c)

Lopuksi siirrytään ylläpitovaiheeseen, joka on ohjelmistojen elinkaaren tärkein vaihe. Ylläpitoon käytettävä työmäärä on 60 % ohjelmiston kehittämisen kokonaistyömäärästä. (Pal, 2018)

Ylläpidon voi jakaa kolmeen eri kategoriaan: korjausylläpitö, kehittävä ylläpito sekä mukauttava ylläpito. Korjausylläpito tarkoittaa, että korjataan virheet, joita ei havaittu tuotteen kehitysvaiheessa. Kehittävässä ylläpidossa parannetaan järjestelmän toimintoja asiakkaan pyynnöstä. Mukauttavaa ylläpitoa tarvitaan silloin, kun ohjelmisto halutaan siirtää uuteen ympäristöön, kuten toimimaan uudella alustalla tai käyttöjärjestelmällä. (Pal, 2018)

2.2 Nopean kehittämisen malli

Nopean kehittämisen malli perustuu minimaaliseen suunnitteluun ja nopeiden prototyyppien luomiseen. Nämä prototyypit vastaavat toiminnallisesti tuotteen osaa. Osat luodaan rinnakkain eri tiimien toimesta, jotka lopuksi yhdistetään valmiiksi tuotteeksi. Koska nopean kehittämisen malli ei sisällä yksityiskohtaista suunnitelmaa, muutoksien teko kehitysprosessin aikana on helpompaa. (Tutorials Point, n.d.-b)

Nopean kehittämisen mallissa tuotteen osat luodaan tiettyjen vaiheiden mukaisesti. Nämä vaiheet ovat liiketoiminnan mallinnus, datan mallinnus, prosessin mallinnus, sovelluksen tuottaminen sekä testaaminen ja osan luovuttaminen. (Tutorials Point, n.d.-b)

Ensimmäinen vaihe on liiketoiminnan mallinnus, jossa määritetään tiedon kulku liiketoiminnoissa. Tämä vaihe määrittää, että mikä tieto vaikuttaa liiketoimintaprosessiin, mitä tietoa syntyy, kuka tietoa tuottaa, minne tiedot menevät ja kuka sitä prosessoi. (Tutorials Point, n.d.-b)

Datan mallinnus -vaiheessa tiedot liiketoiminnan mallinuksesta jalostetaan dataobjektien joukoksi, jotka tukevat liiketoimintaa. Dataobjektien ominaisuudet tunnistetaan ja dataobjektien väliset suhteet määritetään. (Tutorials Point, n.d.-b)

Prosessin mallinnuksessa datan mallinnus -vaiheen dataobjektien joukkoja muutetaan siten, että saadaan aikaan liiketoiminnallinen tiedonkulku, jota tarvitaan tiettyjen liiketoimintatavoitteiden saavuttamiseksi liiketoiminnan mallinuksen mukaisesti. Dataobjektien joukkoihin kohdistuvat muutokset tai parannukset määritetään tässä vaiheessa. Dataobjektien lisäämistä, poistamista, noutamista ja muokkausta varten määritetään prosessin kuvaukset. (Javatpoint, n.d.)

Sovelluksen tuottamisvaiheessa todellinen järjestelmä luodaan. Koodaus tehdään automaatiotyökaluilla, jotka muuntavat prosessi ja data mallit prototyypeiksi. Lopuksi valmiit osat testataan ja luovutetaan eteenpäin. Jokainen komponentti siis testataan itsenäisesti, joka vähentää suurten ongelmien riskiä myöhemmin. (Tutorials Point, n.d.-b)

2.3 Ketterät menetelmät

Ketterät menetelmät perustuvat uskomukseen, jonka mukaan jokaista projektia ei voida hoitaa samalla tavalla ja jossa jo valmiiksi olevia menetelmiä räätälöidään vastaamaan parhaiten projektin vaatimuksiin. Ketterissä menetelmissä lopullinen tuote kehitetään itsenäisissä osissa, joista jokainen toteutetaan tietyssä ajassa, tyypillisesti vähintään yhdessä viikossa ja enintään kolmessa viikossa. Yhden toimivan osan kehitysjaksoa kutsutaan toistoksi (engl. sprint) ja toiston lopuksi toimiva osa esitellään asiakkaalle ja muille tärkeille

sidosryhmän jäsenille. Toistoja toistetaan, kunnes kaikki asiakkaan vaatimat ominaisuudet ovat kehitetty. (Tutorials Point, n.d.-a)

Kuuluisimpia ketteriä menetelmiä ovat Rational Unified Process, Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development ja Dynamic Systems Development. (Tutorials Point, n.d.-a)

Toisin kuin perinteinen vesiputousmalli, ketterät menetelmät ovat hyvin mukautuvia. Ne eivät sisällä yksityistarkkaa suunnitteluvaihetta ja tulevista tehtävistä tiedetään vain se, että mitä ominaisuuksia pitää kehittää. Kehitys on siis ominaisuuspohjaista ja kehitystiimi reagoi vaihtuviin tuotevaatimuksiin dynaamisesti. (Tutorials Point, n.d.-a)

3 Käyttöliittymäsuunnittelu

Käyttöliittymäsuunnittelun tavoitteena on ennakoida mitä toimintoja lopulliset käyttäjät tarvitsevat järjestelmässä ja varmistaa, että nämä toiminnot ovat helposti saatavilla, niitä on helppo ymmärtää ja ne ovat helppokäyttöisiä. (Usablity.gov, 2014)

Vuonna 1994 Jakob Nielsen kehitti kymmenen käytettävyysheuristiikkaa käyttöliittymäsuunnittelua varten. Taulukko 1 kuvaa kyseiset käytettävyysheuristiikat. Ne ovat yleisiä periaatteita käytettävyyden varmistamiseksi eikä täsmällisiä ohjeita. (Nielsen, 2020)

Taulukko 1. Käyttöliittymäsuunnitelun kymmenen käytettävyysheuristiikkaa (Nielsen, 2020)

KÄYTETTÄVYTYYSHEURISTIIKKA	KUVAUS
Järjestelmän tilan näkyvyys	Tietämällä järjestelmän tilan, käyttäjä oppii tunnistamaan kuinka järjestelmä reagoi vuorovaikutukseen. Ennalta arvattavat vuorovaikutukset lisäävät tuotteen ja brändin luotettavuutta.
Järjestelmän ja reaalimaailman kohtaaminen	Käyttöliittymän tulisi noudattaa käyttäjille sopivaa kieltä. Sisäpiirislangin sijaan tulisi käyttää tuttuja sanoja, lauseita ja konsepteja reaalimaailmasta. Näin käyttäjän on helpompi oppia ja muistaa kuinka käyttöliittymä toimii, joka mahdollistaa intuitiivisen käyttökokemuksen.
Käyttäjän hallinta ja vapaus	Käyttäjät tekevät usein toimintoja vahingossa, jota varten olisi syytä luoda helposti tunnistettava toiminnallisuus, jolla voidaan perua ei-toivottu toiminto ilman suurempaa prosessia. Tämä luo käyttäjälle vapauden ja luottamuksen tunteen. Näin

	myös vältetään jumiin jäämistä ja turhautumista.
Johdonmukaisuus ja standardit	Johdonmukaisuuden noudattaminen mahdollistaa helposti opittavan järjestelmän ja näin ei kuormita käyttäjää pakottamalla oppimaan uusia asioita.
Virheiden ehkäisy	Hyvät virheilmoitukset ovat tärkeitä, mutta hyvin suunnitellut käyttöliittymät estävät virheiden syntymisen alkujaan. Tämä saavutetaan joko välttämällä virhealttiit olosuhteet tai tunnistamalla virhetilanteet ja tuottamalla varmistusilmoitus käyttäjille ennen vastaavaa toimintoa.
Tunnistaminen ennen muistamista	Tarvittavien tietojen pitäisi aina olla näkyvillä tai helposti saatavilla. Käyttäjän ei tulisi tarvita muistaa tietoja eri käyttöliittymän osasta, joita tarvitaan sen hetkiseen käyttöliittymän osaan.
Joustavuus ja käytön tehokkuus	Prosessien suorittaminen kannattaa mahdollistaa usealla eri tavalla, jotta käyttäjä voi itse valita hänelle sopivimman tavan toimia. Esimerkiksi oikoteiden mahdollistaminen nopeuttaa käyttöä kokeneemmille käyttäjille.
Esteettinen ja minimalistinen muotoilu	Käyttöliittymän ei tulisi sisältää epäolennaista tietoa tai sellaista tietoa, jota harvoin tarvitaan. Ylimääräiset tiedot vähentävät oleellisen tiedon näkyvyyttä.
Auta käyttäjiä tunnistamaan, diagnosoimaan ja toipumaan virheistä	Virheilmoitukset tulisi esittää selkokielellä, jotta käyttäjä ymmärtää ongelman. Käyttäjälle tulisi myös rakentavasti ehdottaa ratkaisua ongelman ratkaisemiseksi.

Apu ja dokumentointi	Optimaalista olisi, jos järjestelemä olisi niin helppokäyttöinen, ettei lisäselvityksiä tarvittaisi. Joissakin tapauksissa kuitenkin dokumenttien laatiminen on tarpeen, että käyttäjät varmasti ymmärtävät, että kuinka toimintoja käytetään.

4 Pelien kehittämien Androidille

On hyvin yleistä, että sovelluskehittäjä vaihtaa pelikehityksen puolelle ja toisinpäin. Monet, jotka vaihtavat pelikehityksen puolelle eivät vaihda lähestymistapaansa pelikehityksen mukaiseksi. Pelikehittäjän tulisi pitää mielessä, että peli ei ole pelkkä sovellus. Peliä voidaan luonnehtia interaktiiviseksi järjestelmäksi, jonka päätavoite on viihdyttää sen käyttäjää. Sovelluksen tavoite taas on helpottaa käyttäjää jonkin tehtävän kanssa. Kehitystyö näiden kahden välillä on siis varsin erilainen. (Avisekhar, 2016, s. 5)

Peli on aina sovellus, mutta kaikki sovellukset eivät ole pelejä, joka viittaa siihen, että pelikehitys tuo mukanaan enemmän muuttujia ja sen täytyy sisältää kaikki sovelluksen tunnuspiirteet. Sovelluskehitys on siis teknologialähtöistä, kun taas pelikehityksellä tavoitellaan viihtyvyyttä. Tämä vaikeuttaa pelikehityksen prosessia, sillä peliä kehittäessään kehittäjä ei voi varmaksi tietää, että kuinka viihdyttävä pelistä lopulta tulee. Sovelluskehityksessä taas voidaan olla varmoja, että tavoitteeseen päästään, jos tekniset tiedot vastaavat vaatimuksiin. Jotta sovellus voidaan määritellä peliksi, täytyy sen täyttää tietyt kriteerit. Kuten edellä mainittu, pelin täytyy olla viihdyttävä, jonka lisäksi pelin tulisi sisältää tavoitteita ja käyttäjiä tulisi palkita näiden tavoitteiden saavuttamisesta. Pelillä tulisi myös olla dynaaminen käyttöliittymä, olla visuaalisesti näyttävä ja sen tulisi olla ennemmin suorituskeskeinen kuin ominaisuuskeskeinen. (Avisekhar, 2016, ss. 5–6)

4.1 Pelin kehittämisen prosessi

Ensimmäinen toimenpide pelinkehityksessä tulisi olla suunnitteluvaihe, joka sisältää paljon eri valintoja ennen varsinaista ohjelmointityötä. Aluksi pitäisi päättää, että minkä tyyppistä peliä lähdetään kehittämään, mikä on kohdeyleisö, mille laitteille se on tarkoitettu, kuinka pelaaja hallitsee pelin kulkua, halutaanko sillä tehdä rahaa ja kuinka sillä tehdään rahaa. Yksi parhaista tavoista lähestyä suunnitteluun liittyviä kysymyksiä on tehdä luonnos siitä miltä peli näyttäisi, joko käyttäen kynää ja paperia tai siihen soveltuvaa ohjelmaa. (James, 2013, ss. 14–15)

Hyvän idean keksiminen voi olla yksi suunnitteluprosessin vaikeampia vaiheita. Hyvän pelattavuuden tuoman idean keksiminen usein vaatii useita aivoriihi-istuntoja. Kun idea

selkeytyy ajatuksen tasolla, on aika kirjoittaa se ylös. Kirjoitetaan ylös idean pääkohdat: mikä pelissä on hauskaa, kuinka peli voitetaan, mikä hankaloittaa voittamista, kuinka pelaaja pääsee hankaluuksien yli Ja kuka peliä haluaisi pelata. Yksi tapa testata ideaa on kuvakäsikirjoittaminen, jossa päämääränä on kuvittaa pelin idea paperille. Tämä voi säästää paljon ohjelmointiin käytettävää aikaa, jättämällä huonot ideat pois alusta alkaen. (Scolanstici & Nolte, 2013, s. 319)

Työskennellessä tiimissä on kannattavaa käyttää jonkinlaista versionhallintaan soveltuvaa työkalua. Tämä työkalu auttaa organisoimaan mahdolliset muutokset lähdekoodissa. Versionhallinta myös mahdollistaa palaamaan takaisin siihen vaiheeseen, jolloin koodi on toiminut. Versionhallinnan käyttö myös yksin työskennellessä on hyvä käytäntö, koska pelit voivat olla monimutkaisia tai kehittäjä voi löytää itsensä tilanteesta, jossa ohjelmointivirheen korjaaminen osoittautuu liian hankalaksi ja palaaminen toimivaan versioon on helpompi ratkaisu. Jos muodollista versionhallinta tapaa ei käytetä, on silti tarpeellista kehittää jokin tapa varmuuskopioida ja palauttaa koodin eri versiot. Jos peliä kehittäessä ei noudateta muodollista sovelluskehitysprosessia, olisi syytä silti seurata seuraavia työvaiheita: Ensin mietitään, että mitä halutaan peliä pelaavan käyttäjän kokevan, seuraavaksi suunnitellaan pitäen haluttu kokemus mielessä, rakennetaan peli tähän suunnitelmaan pohjatuen, testataan vaiheittain ja parannellaan testien perusteella. Nimenomaan pelejä on hyvä testailla paljon. On kannattavaa testauttaa peliä ulkopuolisilla ihmisillä ja antaa heidän pelata sitä. Tärkeintä on selvittää, että pitävätkö ulkopuoliset ihmiset peliä hauskana vai eivät. (James, 2013, ss. 14–20)

Kuinka peli jaetaan muille, riippuu siitä, halutaanko pelillä tienata rahaa vai ei. Se on yksinkertaista, jos peli halutaan julkaista saataville ilmaiseksi. Silloin peli voidaan siirtää ilmaiseksi kaikkiin markkinoihin, jotka ovat saatavilla. Julkaisu kuitenkin monimutkaistuu, jos pelillä halutaan tienata rahaa. Joissakin markkinoissa on rajoituksia, että kuinka kalliilla peliä voidaan myydä. Rajoituksena voi olla, että peliä ei saa myydä halvemmalla tietyssä kauppapaikassa toiseen kauppapaikkaan nähden. Pelin mahdollinen kaupallistaminen täytyy ottaa huomioon jo pelin suunnitteluvaiheessa, sillä jos vaikka halutaan kaupallistaa peli mainoksilla, vievät mainokset tilaa itse pelin sisällöltä ja näin täytyy suunnitella paras paikka mainoksia varten. Käyttäjät saattavat ärsyyntyä, jos mainokset ovat liian lähellä pelin ohjaimia. (James, 2013, s. 20)

4.2 Mobiilipelien kehittäminen

Älypuhelimet ja sovelluskaupat ovat muuttaneet paljon ihmisten tapaa pelata pelejä ja kuinka kehittäjät niitä tekevät. Mobiilipelien tulon myötä pelikehitys on palannut sen juurille, antaen yksittäisille ihmisille mahdollisuuden kehittää pelejä kotoa käsin ilman suurta tiimiä tai budjettia. Käytännössä kuka tahansa tietokoneen omistava henkilö voi tehdä pelejä Androidille, joka on ilmaista ottamatta huomioon tietokonetta ja Googlen ottamaa välityspalkkiota, jos sovellus on maksullinen. (James, 2013, s. 11)

Pelikehityksestä on tullut hyvin yleinen ammatti nykypäivänä. Ennen kehitystyö oli rajoittunut PC:hen, konsoleihin ja muutamiin pelilaitteisiin. Nykymaailma on täynnä erilaisia moderneja laitteita, jotka ovat paremmalla teknologialla varustettuja, kannettavuudeltaan parempia, joustavampia ja laadultaan parempia. Tämä on avannut mahdollisuuksia kehittäjille laadukkaampien pelien tekemiseen ja vähemmillä rajoituksilla. (Avisekhar, 2016, s. 1)

Android on modernin ajan käyttöjärjestelmä, jota käytetään useissa eri laitteissa alustana. Androidin maailma on tullut pelikehittäjien kohteeksi. Tehokkain ja hyödyllisin kohdeyleisö on Android älypuhelimet ja tabletit. Androidin maailmanlaajuinen markkinaosuus puhelinten käyttöjärjestelmänä oli vuonna 2015 78-80 %. Android ei ole enää vain mobiilikäyttöjärjestelmä, vaan sitä käytetään myös televisiossa ja älykelloissa. (Avisekhar, 2016, s. 1)

Pelikehityksen ero mobiilialustoille ja muille alustoille, kuten tietokoneelle on merkittävä. Eroja ovat laitteistojen valmiudet, toteutusprosessi, liiketoimintamallit ja hinnoittelupolitiikka. Mobiilitekniikan edistys on mahdollistanut huippulaadukkaita laitteita, jotka yltävät samoihin valmiuksiin entisten sukupolvien konsoleiden kanssa. Huomioonotettavaa on silti se, että tavallinen mobiililaitteen omistaja omistaa keskinkertaisen mobiililaitteen, joka on kyvykkyydeltään rajallinen etenkin muistin ja prosessointi tehon osalta. Mobiilimarkkinat ovat massamarkkina, joten on hyvä käytäntö kohdentaa peli mahdollisimman laajalle yleisölle, joten peli tulisi suunnitella niin, että se toimii moitteetta myös vanhemmilla älypuhelimilla ja ettei pienempi näyttö tekisi siitä vähemmän viihdyttävää pelattavaa. Laitteiston vaihtelevuus on myös merkille pantava asia.

Etenkin jos peli kohdennetaan Android-markkinoille, pelin täytyy toimia usealla eri laite konfiguraatioilla. Laitteiden näytön koko, resoluutio, suorittimen nopeus ja kosketusnäytön responsiivisuus vaihtelevat. Suurin osa mobiilialustojen laitteiden näytöistä on pieniä. Tämän vuoksi on syytä miettiä, kuinka paljon tietoa tarvitaan näytölle, niin että pelattavuus on merkityksellistä ja kuinka nämä tiedot saadaan tunnistettaviksi. (Scolanstici & Nolte, 2013, s. 324)

4.3 Käyttöliittymä ja käyttökokemus

Pelikehityksessä hyvän käyttöliittymän ja käyttökokemuksen saavuttaminen on tärkeää, sillä hyvä käyttöliittymä ja hyvä käyttökokemus yhdessä mahdollistavat hyvän pelikokemuksen, jolloin ihmiset viihtyvät pelin parissa. (Kristiadi ym., 2017)

Peleissä käyttöliittymää voi luonnehtia järjestelmäksi, joka tuottaa pelaajalle olennaiset tiedot pelistä ja työkalut, joiden kanssa pelaaja voi olla vuorovaikutuksessa pelin kanssa. Käyttöliittymä pelissä on mikä tahansa ominaisuus, joka tuottaa tietoa tai mahdollistaa vuorovaikutuksen pelaajan ja pelin välille. Käyttöliittymä on myös näiden ominaisuuksien kokonaisuus, johon sisältyy laitteiston ominaisuudet kuten säätimet ja näyttö sekä sovelluksen sisältämät ominaisuudet, kuten pelin audiovisuaaliset ominaisuudet. (Llanos & Jørgensen, 2011)

Mobiilipelien käyttöliittymää suunniteltaessa on huomioonotettavaa, että virtuaaliset painikkeet vievät osan pelialueesta, joten hyvän tasapainon löytäminen pelimekaniikoiden ja peliohjaimien välillä on tärkeää. Hyvä yleissääntö on, että mobiilipelien ei tulisi sisältää liian monimutkaisia ohjaimia. Toisaalta kosketusnäyttö mahdollistaa erilaisia tapoja toimia pelin kanssa, kuten napauttaminen ja laahaus. Näiden huomioiminen jo suunnitteluvaiheessa on suositeltavaa, jolloin käyttöliittymästä ja käyttökokemuksesta saadaan kaikki irti. (Scolanstici & Nolte, 2013, ss. 325–326)

Kokemuksien havainnointi on luontainen osa ihmisyyttä. Kokemuksella yleisesti ottaen viitataan henkilökohtaisesti kohdattuun, läpikäytyyn tai elettyyn kokemukseen. Tästä eroten käyttökokemuksella viitataan kokemukseen tai kokemuksiin, joissa käyttäjä kohtaa jonkin järjestelmän, joten määritelmänä käyttökokemus on kokemusta tarkempi.

Käyttökokemuksen keskiössä on itse varsinainen kokemus järjestelmää käyttäessä, mutta siihen liittyy myös epäsuoria kokemuksia ennen järjestelmän ensikohtaamista. Nämä epäsuorat kokemukset muodostuvat jo koetuista kokemuksista, jotka ovat syntynyt samankaltaisten teknologioiden, brändien, mainoksien, esityksien, demonstraatioiden tai muiden mielipiteiden myötä. Epäsuoria kokemuksia syntyy myös järjestelmän käyttämisen jälkeen esimerkiksi pohdiskelemalla edellistä käyttökertaa. (Roto ym., 2011)

Käyttökokemukseen vaikuttavia tekijöitä on useita, jotka voidaan jakaa kolmeen pääkategoriaan: käyttäjää ja järjestelmää ympäröivä konteksti, käyttäjän tila ja järjestelmän ominaisuudet. Kontekstin muuttuessa on täysin mahdollista, että myös käyttökokemus muuttuu, vaikka järjestelmä pysyisi täysin samana. Kontekstilla käyttökokemuksessa viitataan sosiaaliseen kontekstiin, fyysiseen kontekstiin, tehtävän kontekstiin sekä tekniseen ja informaatio kontekstiin. Käyttäjän tilalla viitataan henkilön motivaatioon käyttää järjestelmää, hänen mielialaansa, henkisiin ja fyysisiin resursseihin sekä oletuksiin. Käyttäjän käsitys järjestelmän ominaisuuksista vaikuttaa luonnollisesti käyttökokemukseen. Näitä ominaisuuksia on toiminnallisuus, estetiikka, suunniteltu vuorovaikutteinen käyttäytyminen ja mukautuvuus. Näihin ominaisuuksiin lukeutuu myös ominaisuudet, jotka käyttäjä lisää tai muuttaa järjestelmässä tai jotka aiheutuvat järjestelmää käyttäessä, kuten myös brändin tai valmistajan imago. Käyttökokemusta itsessään ei voida kuvailla kuvailemalla siihen vaikuttavia tekijöitä, mutta käyttökokemukseen vaikuttavia tekijöitä ja niiden pääkategorioita voidaan käyttää kuvailemaan tilannetta, jossa henkilö kokee tietynlaisen käyttökokemuksen. Käyttökokemukseen vaikuttavat tekijät auttavat myös tunnistamaan syitä tietynlaiselle kokemukselle. (Roto ym., 2011)

Käyttökokemuksen aineettoman luonteen vuoksi on vaikea arvioida kuinka eri suunnittelu valinnat vaikuttavat käyttökokemukseen. Joihinkin käyttökokemukseen liittyviin tekijöihin, kuten sosiaalisiin, emotionaalisiin tai esteettisiin tekijöihin on vaikea, ellei mahdoton vaikuttaa suoranaisesti. Suunnitteluvaiheessa on tärkeä tunnistaa soveltuvat ja toteutettavat menetelmät, työkalut ja kriteerit, joilla voidaan vaikuttaa käyttökokemukseen vaikuttaviin tekijöihin. (Roto ym., 2011)

Yleisesti hyväksyttyä käyttökokemuksen mittaustapaa ei ole olemassa, mutta käyttökokemuksen voi muuntaa arviointikelpoiseksi monella eri tavalla. On olemassa

työkaluja, joilla arvioidaan yksinkertaisesti sitä, että onko herätetty tunne positiivinen tai negatiivinen. Lisäksi on olemassa menetelmiä ja välineitä, jotka ovat erityisesti kehitetty arvioimaan jotakin tiettyä käyttökokemukseen liittyvää piirrettä, kuten luottamusta, läsnäoloa, tyytyväisyyttä tai hauskuutta. Arviointivälineiden ja -menetelmien valitseminen riippuu siitä, että mitä kokemuksellisia ominaisuuksia järjestelmällä tavoitellaan, mikä on arvioinnin tarkoitus ja mitä käytännön rajoitteita on olemassa, kuten ajalliset ja taloudelliset rajoitteet. (Roto ym., 2011)

5 Valitut työkalut ja teknologiat

Vähimmäisvaatimukset mitä työkaluihin tulee Android-pelien kehityksessä, on tietokone, Java Development Kit, integroitu ohjelmointiympäristö ja Android SDK. Tietokonetta lukuun ottamatta, muut ovat ilmaisia. Android SDK sisältää emulointiominaisuudet, joka mahdollistaa virtuaalisten Android laitteiden luomisen tietokoneella, mutta ne toimivat hitaasti. On suositeltavaa omistaa edes yksi Android-laite, jolloin ei tarvitse käyttää hidasta emulaattoria ja pelikokemuksen käsittää paremmin, kun sitä pelataan oikealla laitteella. (James, 2013, s. 21)

5.1 Kanban

1940-luvun alussa teollisuusinsinööri ja liikemies Taiichi Ohno kehitti ensimmäisen kanban-järjestelmän Toyotan autoteollisuutta varten. Tarkoituksena oli luoda yksinkertainen järjestelmä, jonka avulla voidaan valvoa ja hallita työnkulkua ja inventaariota kaikissa tuotantovaiheissa optimaalisesti. David J. Anderson oli ensimmäinen, joka sovelsi kanban konseptia IT-alalle määrittelemällä Kanban menetelmän vuonna 2004. (Nimble, 2022)

Kanban menetelmän perusajatuksena on paloitella projekti pieniksi osiksi, visualisoida tehtävä työ ja auttaa työn optimoinnissa yksinkertaisten sääntöjen ja mittausmenetelmien avulla. Koko menetelmän keskeisin periaate on kanban-taulu, johon työtehtävät ja etenemisvaiheet visualisoidaan. Tyypillisesti kanban-taulussa on vähintään kolme saraketta: tekemättä, työn alla ja tehty. Työtehtävät esitetään kortteina taulussa ja järjestellään sarakkeisiin tehtävän valmistumisasteen mukaisesti. (Koskinen, 2021)

Työn visualisointi niin, että koko projektitiimi voi hahmottaa jokaisen projektin vaiheen ja tehtävän yhteiseltä taululta on kanban-menetelmän yksi perussäännöistä. Tämän lisäksi on tärkeää mitata menetelmän tehokkuutta ja tuottavuutta. Menetelmän tehokkuutta voidaan mitata tehtävien keskimääräisellä valmistusajalla, eli ajalla, joka kuluu tehtävän siirtymisestä tekemättä-sarakkeesta tehty-sarakkeeseen. On myös tärkeä huolehtia siitä, että työnkulkuun ei muodostu pullonkauloja. Jotta työnkulku on sujuvaa, täytyy keskeneräisen työn määrää rajoittaa. Kun huomataan, että käynnissä olevien tehtävien määrä uhkaa kasvaa liian suureksi, pitäisi uusien tehtävien aloittaminen lopettaa. Kanban-taulu auttaa myös

hahmottamaan sen, että missä järjestyksessä tehtäviä kannattaa alkaa tekemään, jotta työnkulku olisi mahdollisimman sujuvaa. (Koskinen, 2021)

5.2 Figma

Figma on moderni työkalu käyttöliittymäsuunnittelua varten. Figma mahdollistaa sujuvan yhteistyön suunnittelijoiden välille sen ominaisuuksien ansioista, joihin lukeutuu mahdollisuus katsoa ja muokata samaa tiedostoa samanaikaisesti muiden kanssa, sekä mahdollisuus jättää ja vastaanottaa palautetta suoraan suunnitelmiin tai prototyyppeihin kommenttien avulla. Figma on selainpohjainen työkalu, joka tarjoaa työpöytäversion Windows-, Chrome-, MacOS- ja Linux-käyttöjärjestelmille. (Figma, n.d.)

Figman etuja ovat muun muassa se, että se on rajoituksineen ilmainen ja se toimii usealla eri käyttöliittymällä, toisin kuin toinen vastaava suunnittelutyökalu nimeltä Sketch, joka toimii vain MacOS-käyttöjärjestelmällä. Figmaan on mahdollista tuoda Sketch-tiedostoja, mutta tiedostojen siirtäminen Sketchiin ei ole mahdollista. Figmaan on myös integroitu tekniset tiedot kehittäjiä varten sekä mahdollisuus luoda prototyyppejä. (Vallaure, 2022)

5.3 Unity

Pelimoottoreita pidetään tehokkaana työkaluna pelikehittäjien keskuudessa. Pelimoottori on ohjelmistokerros, joka auttaa pelin kehitysprosessissa mahdollistaen kehittäjien keskittymisen pelkästään pelilogiikan luomiseen. (Aleem ym., 2016)

Suosituimpien pelimoottorien, kuten Unityn ja Unrealin kasvun myötä useita pelimoottorin vastuualueita pidetään itsestään selvinä. Tänä päivänä pelimoottorin odotetaan huolehtivan monista asioista, kuten käyttäjän syötteestä, animoinnista, renderöinnistä, fysiikan simuloinnista, profiloinnista ja vianetsinnästä sekä monista muista seikoista. Pelimoottori koostuu siis useista moduuleista, joilla on oma roolinsa, kuten äänimoottori tai fysiikkamoottori. Yleisesti ottaen moduulien lisäksi pelimoottori sisältää järjestelmän, joka huolehtii eri osien välisestä vuorovaikutuksesta. (Rădulescu, 2020)

Yleisimmät pelimoottorit mahdollistavat pelin koodaamisen skripteillä ja sisäänrakennetuilla komponenteilla. On myös mahdollista tuoda materiaaleja ja resursseja 3D-mallinnussovelluksista, äänenkäsittelyohjelmista, kuvankäsittelyohjelmista ja useista muista lähteistä. (Rădulescu, 2020)

Unity on yksi suosituimmista alustariippumattomista pelimoottoreista. Sillä on laaja yhteisö, joka koostuu yli 2,5 miljoonasta rekisteröidystä kehittäjästä. On monia syitä sille miksi Unity on niin suosittu. Yksi niistä on se, että Unity mahdollistaa monien erityyppisten pelien luomisen, kuten 2D- tai 3D-pelit, VR- ja AR-pelit sekä mobiilipelien saralla Unity hallitsee yli puolta markkinoista. Tämän lisäksi Unityn yksi vahvuus on sen kauppapaikka nimeltä Unity Asset Store. Sieltä kehittäjät voivat hakea erilaisia materiaaleja projekteihinsa, kuten ääniä, 3D-malleja, kuvioita, tekstuureja tai animaatioita. (Arnia Software, 2022)

5.3.1 Perusrakenteet ja komponentit

Tässä luvussa käsitellään Unityn perusrakenteita ja komponentteja. Näihin lukeutuu GameObject, Scene, Camera, Asset ja Prefab.

Jokainen objekti Unityllä tehdyssä pelissä on peliobjekti (GameObject). Peliobjektit edustavat hahmoja, rekvisiittaa ja näkymää. Peliobjektit toimivat säiliönä erilaisille komponenteille, jotka toteuttavat objektin toiminnallisuuden ja ominaisuudet. (Unity Technologies, 2017a)

Transform on komponentti, joka määrittää peliobjektin sijainnin, kuinka päin peliobjekti on ja mikä on objektin koko. Jokaisella peliobjektilla on Transform-komponentti eikä sitä ole mahdollista poistaa. (Unity Technologies, 2023c)

Kaikki sisältö Unityssä sijaitsee yhdessä tai useamassa Scenessä. Yksi Scene voi siis sisältää pelin tai sovelluksen kokonaisuudessaan tai vain osan siitä. Monimutkaisemmissa peleissä Scene voi edustaa yhtä tasoa pelissä, joissa jokaisessa on omat ympäristöt, hahmot, esteet, koristeet ja käyttöliittymä. (Unity Technologies, 2023b)

Unityssä Scene on kolmiulotteinen tila, jossa peliobjektit sijaitsevat. Koska peliä tullaan pelaamaan kaksiulotteisella näytöllä, tarvitsee Unityn kaapata näkymä ja litistää se näyttöä

varten. Sitä varten on olemassa Camera-komponentti, jota päästään käyttämään lisäämällä se peliobjektiin. (Unity Technologies, 2023a)

Asset edustaa mitä tahansa elementtiä, jota voidaan käyttää Unity-projektissa. Asset voi olla Unityn ulkopuolella luotu tiedosto, kuten 3D-malli, äänitiedosto, kuva tai minkä vaan tyyppinen tiedosto, jota Unity tukee. Joitakin Asset-tyyppejä voidaan luoda Unityssä, kuten animaattoriohjain ja äänimikseri. (Unity Technologies, 2017b)

Joskus Unityä käyttäessä tulee tarve käyttää uudelleen tietyllä tavoin konfiguroitua peliobjektia, kuten ei-pelattavaa hahmoa tai lavastetta. Tällöin peliobjekti kannattaa muuntaa Prefabiksi. Unityssä on myös mahdollista kopioida ja liittää peliobjekteja, mutta Prefab-järjestelmä mahdollistaa pitämään kaikki kopioit ajan tasalla ilman, että täytyy tehdä muutoksia jokaiseen kopioon erikseen. (Unity Technologies, 2018)

Prefab-järjestelmän käyttö on myös suositeltavaa, kun halutaan luoda peliobjekti pelin ollessa käynnissä esimerkiksi, kun halutaan, että ei-pelattava hahmo ilmestyy oikealla hetkellä pelin aikana. (Unity Technologies, 2018)

5.3.2 Unity ja Android-sovellukset

Jotta voidaan luoda Android-sovelluksia Unityllä, täytyy Unity-projektin tukea Androidia. Android-tuen saavuttamiseksi Unity-projektin täytyy sisältää Android Build Support -moduuli, Android Software Development Kit (SDK), Native Development Kit (NDK) ja Java Development Kit, jonka OpenJDK-versiota Unity oletusarvoisesti käyttää. Nämä riippuvuudet ovat saatavilla moduuleina, jotka voi asentaa Unity Hubista. Ne voi asentaa joko Unity-muokkausohjelman asennuksen yhteydessä tai lisäämällä ne jo valmiiksi asennettuun Unity-muokkausohjelmaan. On suositeltavaa asentaa kyseiset moduulit Unity Hubin kautta, mutta on olemassa poikkeustilanteita, joissa SDK-, NDK- tai JDK-version vaihtaminen on kannattavaa. (Unity Technologies, 2022a)

5.3.3 Unity Remote

Unity Remote on ladattava sovellus, joka on luotu avustamaan kehitystyössä Android-, iOS- ja tvOS-käyttöjärjestelmille. Se yhdistää kohdelaitteen Unity-muokkausohjelmaan ja näyttää muokkausohjelmasta visuaalisen tulostuksen kohdelaitteen näytöllä pienennetyllä ruudunpäivitysnopeudella. Se myös lähettää reaaliajassa käyttäjän syötteen kohdelaitteelta takaisin käynnissä olevaan projektiin Unityssä. Tämän avulla voidaan katsoa miltä sovellus näyttää ja kuinka se toimii kohdelaitteella ilman projektin rakentamista. (Unity Technologies, 2022b)

Unity Remotea käytettäessä Unityn muokkausohjelma suorittaa sovelluksen varsinaisen prosessoinnin tietokoneella, joten sovelluksen suorituskyky ei ole tarkka esitys siitä miten valmiiksi tehty sovellus toimisi kohdelaitteella. Jotta saadaan tarkempi arviointi sovelluksen suorituskyvystä, täytyy ajoittain rakentaa sovellus ja kokeilla sitä kohdelaitteella. (Unity Technologies, 2022b)

Kun yhdistetään Unity Remote Android-laitteeseen, tietokoneella täytyy olla Android SDK asennettuna. Unity Remote asetetaan käyttökuntoon asentamalla sovellus kohdelaitteelle, jonka jälkeen avataan sovellus ja yhdistetään kohdelaite tietokoneeseen. Jos kyseessä on Android-laite, käytetään USB-yhteyttä. Kun kohdelaite on yhdistetty tietokoneeseen, Unity automaattisesti havaitsee laitteen. Unity Remote yhdistetään kohdelaitteelta Unity-muokkausohjelmaan avaamalla Unity muokkausohjelman Editor-asetukset (Edit > Project Settings > Editor) ja määrittämällä käytettävä laite Unity Remote-kohdasta. (Unity Technologies, 2022b)

Sovellusta voidaan esikatsella painamalla Play-painiketta Unity Editorissa, jolloin sovellus näkyy kohdelaitteen näytöllä, sekä Editorin pelinäkymässä. Unity Remote lähettää käyttäjän syötteen takaisin Unity editorille ja tehdyt skriptit käsittelevät syötteen niin kuin ne olisivat kohdelaitteella itsessään. (Unity Technologies, 2022b)

5.4 Firebase

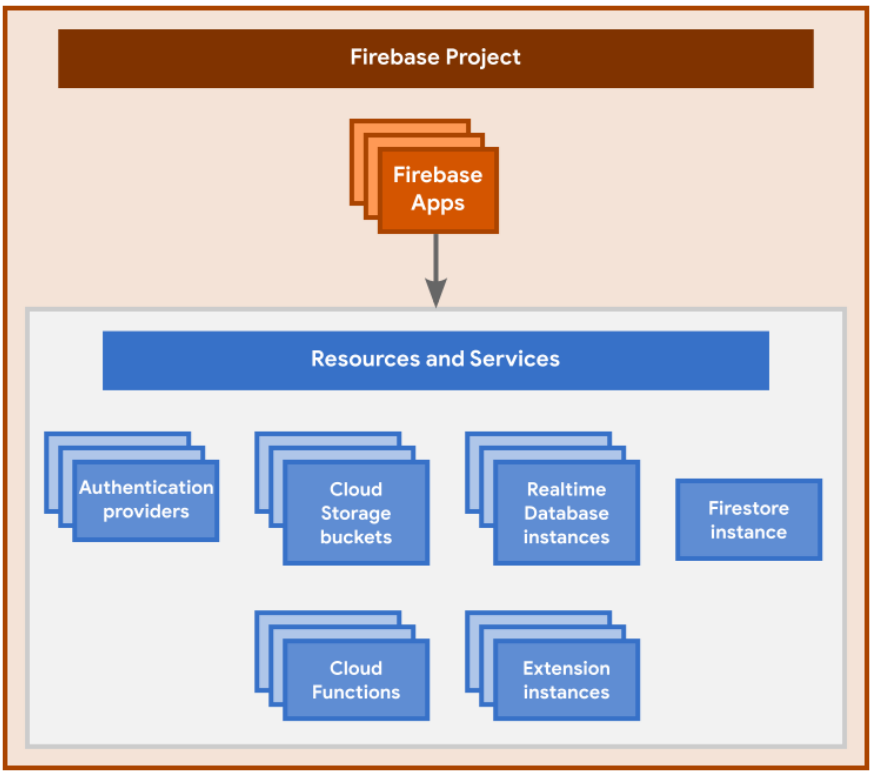
Joka päivä sovelluskauppoihin ilmestyy sovelluksia, joidenka tavoitteena on tuottaa paras mahdollinen suorituskyky sovelluksien käyttäjille. Tämän tavoitteen onnistumisen tueksi on olemassa mobiili backend -palveluita, joidenka avulla pystytään saavuttamaan parempi infrastruktuuri sovellukselle. On hyvin yleistä, että mobiilisovellukset tarvitsevat backendin, joka on yhteydessä internettiin. BaaS (Backend-as-a-Service) on palvelu, joka avulla se voidaan toteuttaa. BaaS-palvelu tarjoaa backendin mobiilisovelluksille, rajapinnan ja työkalut, joilla voidaan integroida useita eri ohjelmointikieliä sovelluksen backendin kanssa. BaaS-palveluihin sisältyy myös muita lisäpalveluita, kuten pilvipohjaista tallennustilaa, push-ilmoitukset sekä käyttäjätilien ja tiedostojen hallinta mahdollisuus. BaaS-palvelu on palvelu, joka on kohdennettu kehittäjille, toisin kuin taas SaaS (Software-as-a-Service) on palvelu loppukäyttäjiä varten. Tunnettuja BaaS-palveluratkaisuja ovat Amazon Web Services, Kinvey, Apple CloudKit ja Google Firebase. (Fatima, 2017)

Firebase julkaistiin huhtikuussa vuonna 2012 James Tamplinin ja Andrew Leen toimesta. Alkuaikoina Firebase tuotti rajapinnan, joka auttoi sopeuttamaan verkkochattimoduuleita verkkosivuille. Nykyään Firebase on yksi hallitsevista BaaS-alustoista, joka jatkuvasti kehittää uusia ominaisuuksia ja toiminnallisuuksia palveluunsa. Firebase soveltuu parhaiten tilanteeseen, jossa kehitystyön aikataulu on tiukka ja sovellus vaatii datan käsittelyä reaaliajassa. (Ashok Kumar S, 2018, s. 7)

5.4.1 Firebase-projekti

Firebase-projekti on ylimmän tason kokonaisuus Firebasessa. Kuvassa 1 esitetään Firebase-projektin hierarkia. Projekti mahdollistaa iOS-, Android- ja Web-sovelluksien rekisteröinnin Firebasen kanssa. Kun sovellus on rekisteröity projektiin, voidaan sovellukseen lisätä eri Firebase-palveluiden ohjelmistopaketteja, kuten Analytics tai Cloud Firestore. Firebase-projekti on ikään kuin säiliö sovelluksille sekä resursseille ja palveluille, jotka sisältyvät projektiin. Firebase-projektiin voi olla liitettynä useampi Firebase-sovellus esimerkiksi iOS- ja Android-versio tai maksullinen ja maksuton versio sovelluksesta. Kaikki saman projektin alaiset Firebase-sovellukset jakavat samat resurssit sekä palvelut. Kuva 1 näyttää Firebase-projektin hierarkian. (Firebase, 2022c)

Kuva 1. Firebase-projektin hierarkia (Firebase, 2022c)



Firebase-projektia luodessa kulisseissa luodaan Google Cloud -projekti. Google Cloud -projektia voidaan luonnehtia virtuaalisena säiliönä dataa, konfiguraatioita ja palveluita varten. Firebase-projekti on Google Cloud -projekti, joka sisältää täydentävät Firebase konfiguraatiot ja palvelut. Google Cloud -projektin luominen ensin ja siihen Firebasen liittäminen myöhemmin on myös mahdollista. (Firebase, 2022c)

5.4.2 Firebase-projektin lisääminen Unity-projektiin

Jotta voidaan lisätä Firebase onnistuneesti Unity-projektiin, täytyy seuraavien edellytyksien täyttyä. Unityn version täytyy olla 2018.4 tai uudempi, vaikka jotkut aikaisemmat versiot saattavat olla yhteensopivia, niitä ei aktiivisesti tueta Firebasen puolelta. Unityn version 2018.4 tukea pidetään yleisesti jo vanhentuneena, jota ei seuraavan merkittävän julkaisun jälkeen enää tueta. Android-laitteille suunnatun Unity-projektin kohde API-tason tulisi olla 19 (KitKat) tai suurempi. Sovelluksen suorittamista varten tarvitaan joko fyysinen Android-puhelin tai emulaattori, johon sisältyy Google Play -palvelut. (Firebase, 2022d)

Firebasen lisääminen Unity-projektiin aloitetaan luomalla Firebase-projekti Firebase konsolissa. Tämän jälkeen liitetään sovellus Firebase-projektiin, joka tehdään konsolissa projektin yleiskatsauskohdassa painamalla Unity-kuvaketta. Tämä käynnistää työnkulun, joka ohjeistaa käyttäjän liittämisprosessin läpi. Työnkulku ohjaa käyttäjän ensimmäisenä rekisteröimään sovelluksen Firebase-projektiin, jonka jälkeen ladataan konfiguraatiotiedosto tai tiedostot, jotka lisätään Unity-projektiin. Seuraavaksi työnkulku ohjeistaa lataamaan Firebase Unity SDK:n ja lisäämään siitä halutut Firebase-palvelut Unity-projektiin. Lopuksi varmistetaan Google Play -palveluiden versiovaatimukset. (Firebase, 2022d)

5.4.3 Cloud Firestore

Cloud Firestore on Firebasen uusin tietokanta, joka on luotu mobiilisovellusten kehitystä varten. Firebasen alkuperäinen tietokanta on nimeltään Realtime Database, jonka onnistumisien pohjalta Cloud Firestore luotiin. (*Choose a Database*, 2022)

Cloud Firestore on pilvessä oleva mukautuva ja skaalautuva NoSQL-tietokanta mobiili-, web- ja palvelinkehitystä varten. Firebase alustan lisäksi Cloud Firestorea voidaan käyttää myös Google Cloudin kautta. Cloud Firestore pitää datan synkronoituna eri sovellusten välillä. Se sisältää myös offline-tuen mobiili- ja webkehitys puolella, jolloin voidaan tehdä responsiivisia sovelluksia, jotka toimivat viiveestä ja internet-yhteydestä huolimatta. (Firebase, 2022a)

Cloud Firestoren keskeisiin ominaisuuksiin lukeutuu sen joustavuus, tapa toteuttaa kyselyitä, reaaliaikaiset päivitykset, offline-tuki ja kuinka se on suunniteltu laajenemaan. Joustavuus ilmenee siten, että Cloud Firestoren tietomalli tukee joustavia ja hierarkkisia tietorakenteita, jossa tieto tallentuu dokumentteihin, jotka taas ovat lajiteltu kokoelmiin. Cloud Firestoressa pystyy kyselyillä noutamaan yksittäisen tietyn dokumentin tai noutamaan kaikki dokumentit kokoelmasta, joka vastaa kyselyn parametreihin. Kyselyt voivat sisältää useita ketjutettuja suodattimia sekä suodattamista ja lajittelua yhdistettynä. Kyselyä vastaavat dokumentit indeksoidaan oletusarvoistesti, joten kyselytehokkuus on verrannollinen tuloksiin, eikä koko tietomäärään. Cloud Firestore käyttää tiedon synkronointia päivittääkseen tiedot siihen yhteydessä oleviin laitteisiin ja se tallentaa aktiivisesti sovelluksen käytössä olevat tiedot välimuistiin, joten sovellus voi kirjoittaa, lukea, kuunnella tai tiedustella tietoa, vaikka laite ei olisi yhdistetty internettiin. Laitteen yhdistyessä internettiin, Cloud Firestore synkronoi

paikalliset muutokset tietokantaan. Cloud Firestore on kykenevä laajenemaan ja käsittelemään raskaita tietokanta työmääriä Google Cloudin tehokkaan infrastruktuurin ansioista. (Firebase, 2022a)

Cloud Firestoressa dokumentit tukevat useita eri tietotyyppejä, kuten yksinkertaisia kokonaislukuja, merkkijonoja ja totuusarvoja sekä myös monimutkaisempia tietotyyppejä, kuten taulukoita ja maantieteellisiä sijainteja. (Firebase, 2022b)

Cloud Firestoren käytön aloittaminen on ilmaista. Tähän kuuluu rajoituksia: tallennetun tiedon määrä täytyy olla alle yhden gigatavun, dokumenttien lukukertoja täytyy olla alle 50000 päivässä, dokumenttien kirjoituskertoja täytyy olla alle 20000 päivässä, dokumenttien poistoja täytyy olla alle 20000 päivässä ja verkosta poistuvan liikenteen täytyy olla alle kymmenen gigatavua kuukaudessa. (Firebase, 2023)

6 Pelin toteutusprosessi

Pelin toteutus aloitetaan idean keksimisellä, jonka pohjalta luodaan käyttöliittymäsuunnitelma Figmalla. Unity on keskeinen kehitysympäristö koko projektille ja Cloud Firestore -tietokantaa hyödynnetään pelin toiminnallisuuksissa. Pelin kehityksen tukena käytetään Kanban projektinhallintamenetelmää. Projekti pilkotaan pieniin osiin ja tehtävät asetetaan kanban-taulun tekemättä-sarakkeeseen. Tämän jälkeen voidaan havaita helposti, että mitä tehtävää kannattaa lähteä tekemään, jolloin tehtävä asetetaan työn alla -sarakkeeseen. Tehtävän valmistuttua tehtävä siirretään tehty-sarakkeeseen, jonka jälkeen voidaan aloittaa uusi tehtävä.

Tässä työssä keskitytään esittämään, miten valittuja työkaluja käyttäen saadaan luotua toimiva peli. Työssä esitellään pelin perusidea, jotta pelin toiminnallisuudet on helpompi ymmärtää. Projektiin sisältyvien C#-skriptien osalta käydään läpi, kuinka tiedon hakeminen ja lähettäminen Cloud Firestore -tietokannasta onnistuu.

6.1 Suunnitelma

Käytännön osuus aloitetaan pelin suunnittelulla, johon sisältyy pelin idean keksiminen ja ideaan perustuvan käyttöliittymäsuunnitelman luominen Figmalla. Peliä ideoidessa otetaan huomioon, kuinka Cloud Firestore -tietokantaa tullaan pelissä hyödyntämään.

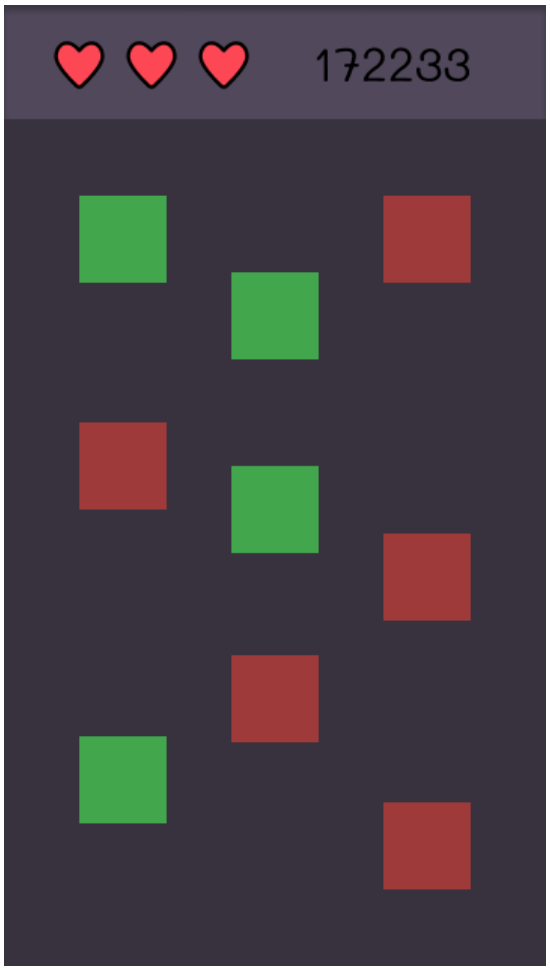
6.1.1 Pelin idea

Peliä pelataan puhelin pystysuunnassa. Pelin yläreunasta tippuu laatikoita alas ja pelaajan päämääränä on kerätä mahdollisimman paljon pisteitä painamalla oikean värisiä eli vihreitä laatikoita. Vihreiden laatikoiden seassa on myös punaisia laatikoita, joiden painamista pelaajan on tarkoitus välttää. Jos pelaaja painaa kolme kertaa punaista laatikkoa, pelaajalta loppuu elämäpisteet ja peli päättyy. Vihreät laatikot, jotka kerkeävät tippua alas peliruudun näkyvistä ilman että pelaaja kerkeää niihin koskea, aiheuttaa miinus pisteitä. Pelin päätyttyä pelaajalla on mahdollisuus tallentaa tuloksensa tulosluetteloon, jos hänen pisteensä yltävät kymmenen parhaan joukkoon.

6.1.2 Käyttöliittymäsuunnitelma Figmalla

Käyttöliittymäsuunnitelmaa tehtäessä on otettava huomioon, että tila käytetään mahdollisimman järkevästi siten, että painikkeiden painaminen on mahdollisimman helppoa ja näin virhepainalluksilta vältyttäisiin. Tämän työn pelitilan käyttöliittymä suunnitellaan hyvin yksinkertaiseksi, jossa suurin osa näytön tilasta varataan pelin interaktiiviselle osalle eli osalle, jossa käyttäjä pystyy painamaan putoavia laatikoita. Kuva 2 näyttää, kuinka pelitilan käyttöliittymä on suunniteltu. Näytön yläosa on varattu pelin kannalta oleellisimmille tiedoille eli tiedot pelaajan sen hetkisistä elämäpisteistä ja pisteistä. Elämäpisteet osoitetaan punaisilla sydänkuvakkeilla, jotka katoavat sitä mukaan, kun pelaaja menettää elämäpisteitä painamalla punaista laatikkoa.

Kuva 2. Pelitilan käyttöliittymäsuunnitelma



6.2 Ympäristöjen asennus

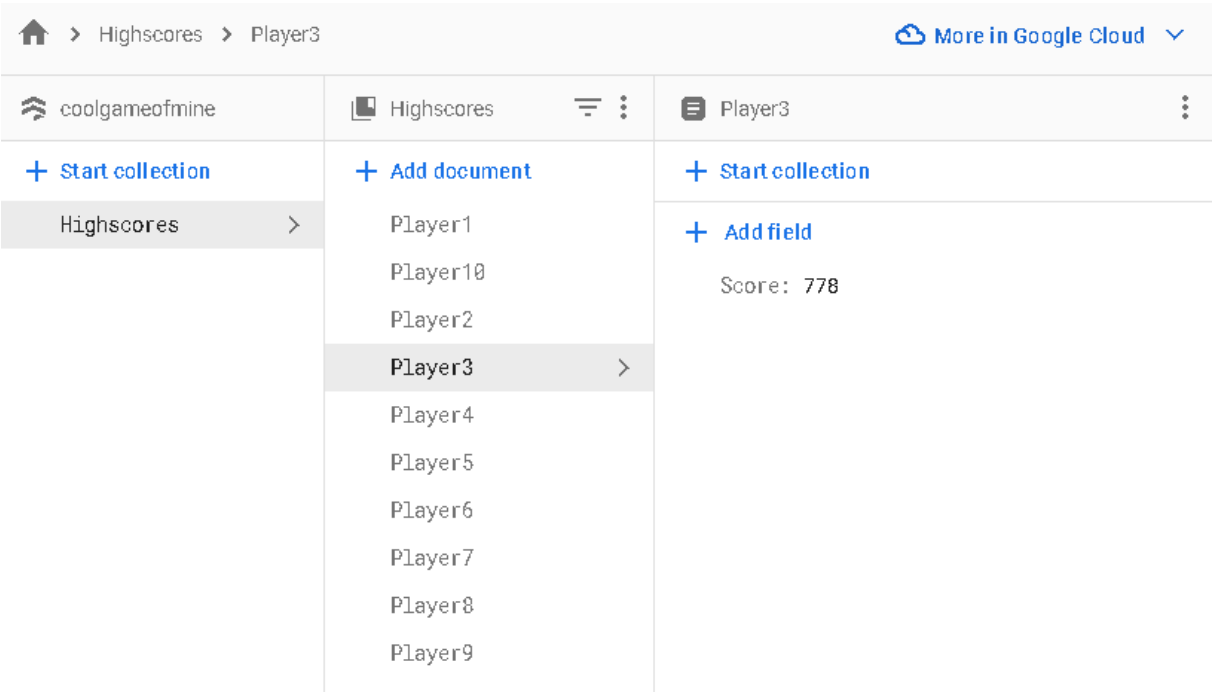
Aluksi asennetaan uusin stabiili ja tuettu versio Unity-muokkausohjelmasta. Unity muokkausohjelma asennetaan Unity Hubin kautta, mutta ennen sitä on syytä tarkistaa, että itse Unity Hub on päivitetty uusimpaan versioon, jotta vältytään mahdollisilta virheiltä. Unity muokkausohjelmaa asentaessa täytyy muistaa valita tarvittavat moduulit, jotta Android-sovelluksien kehittäminen on mahdollista.

Seuraavaksi luodaan Unity- ja Firebase-projektit. Unity-projektia luodessa Unity Hub tarjoaa valmiita projektimalleja, joista 2D Mobile -malli valitaan tätä työtä varten. Jotta voidaan käyttää Firebase-palveluita, täytyy Unity-projekti lisätä Firebase-projektiin, jonka jälkeen ladataan uusin versio Firebase Unity SDK -tiedostosta, josta lisätään Cloud Firestore Unity-projektiin. Tämän jälkeen Unity-projektilla on valmius saada yhteys Cloud Firestore -tietokantaan.

6.3 Cloud Firestore -tietokannan luominen

Kun valmius Cloud Firestoren käyttöä varten on saavutettu, luodaan itse tietokanta. Tietokanta sisältää yhden kokoelman nimeltä ”Highscores”. Tämän kokoelman alle luodaan dokumentit, joiden nimet vastaavat pelaajien valitsemaa nimimerkkiä heidän tallentaessa pisteensä tulosluetteloon. Nämä dokumentit sisältävät yhden tietokentän nimeltä ”Score”, joka nimensä mukaisesti on pisteistä varten oleva tietokenttä. Jotta tietokannan ja pelin välistä toimintaa pystytään testaamaan, tietokantaan laitetaan manuaalisesti yli kymmenen dokumenttia pisteineen. Kuva 3 näyttää, että minkä näköinen tietokanta on Firebasen konsolista käsin.

Kuva 3. Cloud Firestore -tietokannan rakenne



6.4 Pelin toteutus Unityllä

Unityssä tätä työtä varten tarvitaan kolme Sceneä. Yksi Scene päävalikolle, yksi pelitilalle ja yksi tulosluettelolle. Scenet toteutetaan käyttöliittymäsuunnitelmaa mukaillen. Pelissä olevien objektien väri saadaan täsmäämään käyttöliittymäsuunnitelman värejä kopioimalla heksadesimaaliset värikoodit Figma-työkalun ”Design”-välilehdeltä.

6.4.1 HighscoreData-tietue

Jotta päästään käyttämään Firestore-kirjastoa koodissa, täytyy koodissa olla viittaus kyseiseen kirjastoon ennen luokkamääritelmää. Kyseinen kirjasto saadaan käyttöön ”using Firebase.Firestore”-komennolla.

C#-skripteissä halutaan päästä luomaan objekteja vastaamaan Cloud Firestore -tietokannan tietoja, jotta niitä voidaan käsitellä koodissa. Tätä varten luodaan tietue, johon määritetään tarvittavat muuttujat. Kuva 4 näyttää, kuinka tietue on määritelty. Tässä tapauksessa muuttujia tarvitaan vain yksi pisteitä varten, jolle annetaan nimeksi ”Score”. Tietue määritetään vielä erikseen FireStoreData-tyyppiseksi, jolloin se on määritetty käytettäväksi

Firestore-tietokannan kanssa. Score-muuttujalle annetaan määritelmäksi FirestoreProperty, jolla viitataan tietoon, jonka tulee sisältyä Firestore-tietokanta siirtoihin.

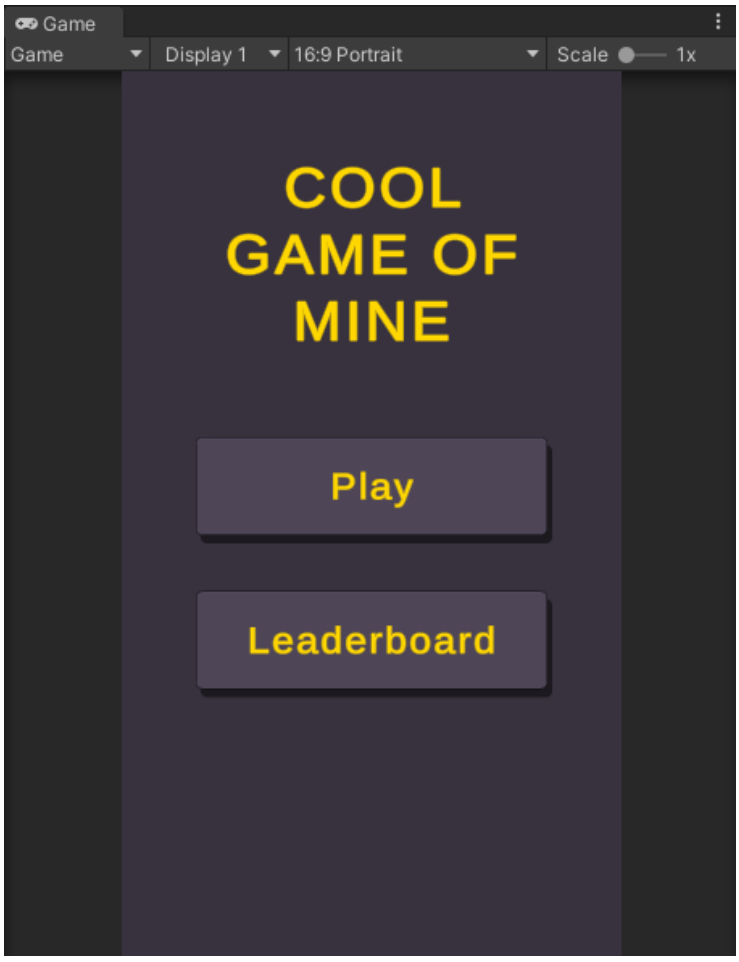
Kuva 4. HighScoresData-tietue määritelmä

```
1   using Firebase.Firestore;
2
3   [FirestoreData]
    4 references
4   public struct HighscoreData
5   {
6       [FirestoreProperty]
        3 references
7       public int Score { get; set; }
8   }
```

6.4.2 Päävalikko

Päävalikkoon luodaan tekstikenttä, johon kirjoitetaan pelin nimi, joka on ”Cool Game Of Mine” ja kaksi painiketta. Kuva 5 näyttää, että minkä näköinen päävalikko on Unityn pelinäkymässä. ”Play”-painikkeella päästään pelitilaan ja ”Leaderboard”-painikkeella päästään katsomaan tulosluetteloa. Painikkeita varten tehdään skripti, joka hyödyntää SceneManager-luokan LoadScene-metodia ladatakseen halutun Scenen.

Kuva 5. Päävalikko pelinäkymässä

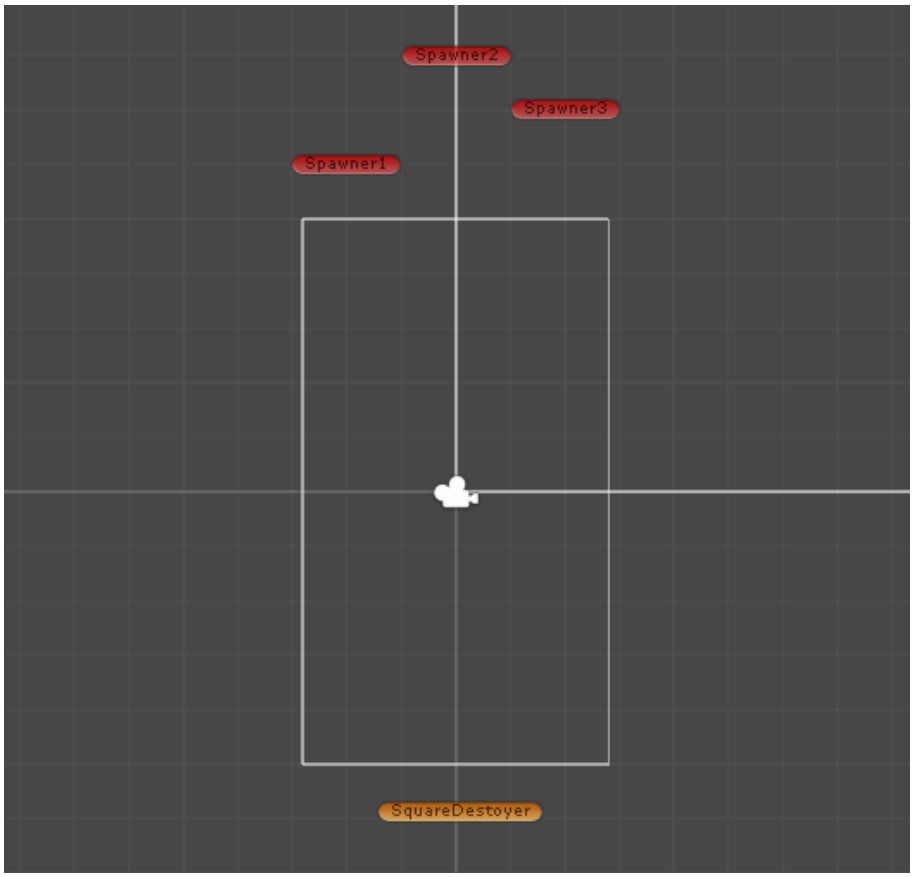


6.4.3 Pelitila

Pelitila on jaettu kahteen pääobjektiin: GameScreen ja GameEndScreen. Pelin käynnistyessä GameScreen-objekti on aktiivinen, kunnes pelaajan elämäpisteet loppuvat, joka aktivoi GameEndScreen-objektin.

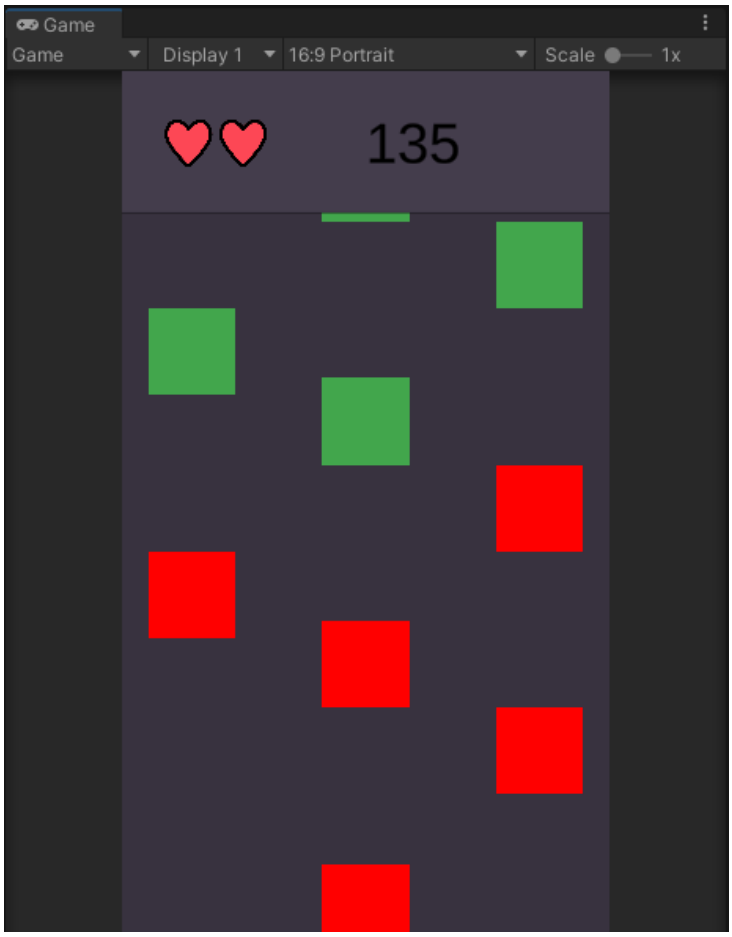
GameScreen-objektin alle luodaan kolme ”Spawner”-objektia, joihin liitetty skripti luo tippuvia laatikoita tietyn väliajoin. Kyseinen skripti luo sattumanvaraisesti joko vihreän laatikon tai punaisen laatikon. Kummastakin laatikkotyypistä luodaan oma Prefab, joita varten myös luodaan omat skriptinsä, koska vihreä ja punainen laatikko käyttäytyvät eri tavalla toisistaan. Peliruudun alapuolelle luodaan ”SquareDestroyer”-niminen objekti, joka tuhoaa laatikot niiden osuessa kyseiseen objektiin. Kuva 6 esittää pelitilan Scene-näkymässä, jossa voidaan havaita objektit pelinäkymän ulkopuolelta.

Kuva 6. Pelitilan Scene-näkymä



GameScreen-objektin alle luodaan myös tarvittavat objektit esittämään oleelliset tiedot pelistä. Kuva 7 esittää pelin pelinäkymän sen ollessa käynnissä.

Kuva 7. Pelitilan pelinäkymä



Pelin päättyessä GameEndScreen aktivoituu ja tällöin peli ottaa ensimmäistä kertaa yhteyden Cloud Firestore -tietokantaan, koska halutaan tietää, että riittävätkö pelaajan pisteet kymmenen parhaan joukkoon, jotta voidaan tarjota mahdollisuutta pelaajalle tallentaa hänen pisteensä tulosluetteloon. Tätä varten tietokantaan lähetetään kysely, joka noutaa kymmenen parasta pistemäärää.

Kyselyä varten skripti tarvitsee viittauksen tietokantaan. Viittaus tietokantaan saadaan käyttäen FirebaseFirestore-luokan DefaultInstance-metodia. Tämän lisäksi tarvitaan viittaus kokoelmaan. Itse kysely luodaan luomalla objekti Query-luokasta. Kuva 8 esittää tarvittavat viittaukset ja kyselyn määrittelyn. Objektiin liitetään viittaus kokoelmaan ja tarvittavat rajoittimet. Tässä tapauksessa halutaan noutaa dokumentteja laskevassa järjestyksessä pisteiden mukaan ja rajoittaa kysely kymmeneen dokumenttiin. Nämä onnistuvat Query-luokan OrderByDyDescending- ja Limit-metodeilla.

Kuva 8. Viittaus Firestore-kokoelmaan ja kyselyn määrittäminen

```
42          // Database
43          var db = FirebaseFirestore.DefaultInstance;
44
45          // Collection
46          CollectionReference highscoresRef = db.Collection("Highscores");
47
48          // Query
49          Query query = highscoresRef.OrderByDescending("Score").Limit(10);
50
```

Kysely toteutetaan GetSnapshotAsync-metodilla, joka palauttaa kyselyn mukaiset dokumentit QuerySnapshot-muodossa. Kuva 9 esittää metodin kokonaisuudessaan. QuerySnapshot-objektin sisältämät dokumentit käydään läpi foreach-silmukalla ja muunnetaan HighscoreData-muotoon, jonka jälkeen sen Score-arvo lisätään taulukkoon. Silmukan päätyttyä voidaan tarkistaa, että onko pelaajan pisteet riittäneet kymmenen parhaan joukkoon vertaamalla niitä taulukon viimeiseen arvoon, joka vastaa tietokannassa olevaa kymmeneksi suurinta pistemäärää. Tämän tiedon avulla voidaan pelaajalle näyttää tilanteeseen sopiva käyttöliittymä: joko mahdollisuus lisätä pisteet kymmenen parhaan joukkoon tai ”pelaa uudelleen”-nappi.

Kuva 9. Tiedon hakeminen Firestore-tietokannasta GetSnapshotAsync-metodilla

```
32          query.GetSnapshotAsync().ContinueWithOnMainThread(task => {
33              QuerySnapshot scoreQuerySnapshot = task.Result;
34              int count = 0;
35              foreach (DocumentSnapshot documentSnapshot in scoreQuerySnapshot.Documents)
36              {
37                  var highScoreData = documentSnapshot.ConvertTo<HighscoreData>();
38
39                  highScores[count] = highScoreData.Score;
40
41                  count++;
42              };
43              CheckIfScoreTop10();
44              ShowCorrectUI();
45          });
```

Pisteet lähetetään tietokantaan luomalla HighScoreData-objekti, jonka Score-arvo asetetaan vastaamaan pelaajan saamia pisteitä. Dokumentin nimi asetetaan vastaamaan pelaajan syötettä tekstikentästä. Objekti lähetetään tietokantaan käyttäen SetAsync-metodia. Kuva 10 esittää SetAsync-metodin käytön koodissa.

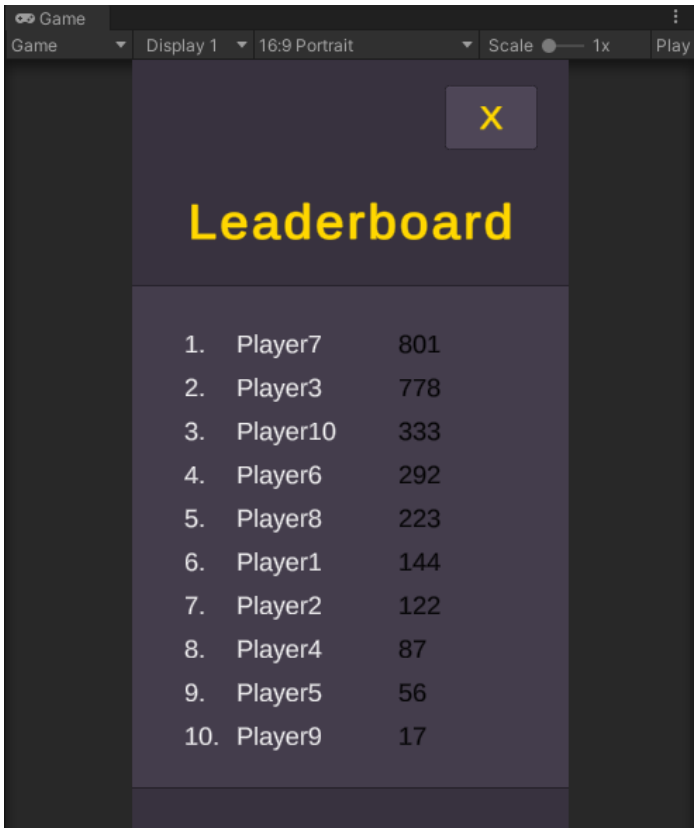
Kuva 10. Tiedon lähettäminen Firestore-tietokantaan SetAsync-metodilla

```
25          _submitButton.onClick.AddListener(() =>
26          {
27              highscore.Score = int.Parse(_scoreText.text);
28
29              var db = FirebaseFirestore.DefaultInstance;
30
31              db.Document("Highscores/" + _nameField.text).SetAsync(highscore);
32
33              // Load Leaderboard scene
34              SceneManager.LoadScene(2);
35          });
```

6.4.4 Tulosluettelo

Tulosluetteloon tehdään tekstiobjektit kymmenen parhaan pelaajan nimimerkeille ja heidän pisteillensä. Tulokset haetaan Cloud Firestore -tietokannasta, kun tulosluettelo avataan. Kuva 11 näyttää, että minkä näköinen tulosluettelo on Unityn pelitilassa pelin ollessa käynnissä. Tulokset haetaan samalla menetelmällä kuin pelitilassa haettiin kymmenen parhaan pisteet.

Kuva 11. Tulosluettelo Unityn pelitilassa



7 Johtopäätökset

Opinnäytetyön aikana on selkeytynyt, että oikeiden työkalujen ja työskentelymenetelmien valitseminen pelikehityksessä on tärkeää. Tässä opinnäytetyössä käytetyt työkalut ovat helppokäyttöisiä, lähtökohtaisesti ilmaisia ja niiden avulla pystytään kehittämään nopeasti toimiva peli. Se että itse pelin kehitysprosessi on mahdollisimman nopeaa ja vaivatonta, voi vapauttaa resursseja muihin asioihin, kuten pelin ideointiin.

Työssä nostettiin esiin pelin viihdyttävyyden tärkeyttä ja asiaa käyttöliittymästä ja käyttökokemuksesta. Tätä työtä voisikin kehittää pidemmälle ja viedä aihetta siihen suuntaan, että millä menetelmillä ja työkaluilla pelikokemuksesta saisi mahdollisimman viihdyttävän.

Pelissä käytettiin tiedon hakemista ja noutamista Cloud Firestore -tietokannasta, mutta tietokannan reaaliaikaisuutta ei hyödynnetty pelissä mitenkään, joka olisi ollut haastavaa pelin luonteen vuoksi.

Tässä opinnäytetyössä päästiin tavoitteeseen, joten työn voi sanoa onnistuneen. Työ sisältää hyödyllistä tietoa kelle tahansa mobiilipelien kehitykseen kiinnostuneelle. Pelin julkaiseminen ja kaupallistaminen jäi työn rajauksen ulkopuolelle, joten työ sopii erinomaisesti niille, jotka haluavat opetella mobiilipelikehityksen perusteita ja harjoitella lähtökohtaisesti ilmaisten työkalujen avulla sitä itse.

8 Yhteenveto

Käyttöliittymäsuunnittelu on tärkeässä roolissa pelin pelattavuuden kannalta. Käyttöliittymäsuunnittelussa on tärkeää osata ajatella asioita loppukäyttäjän näkökulmasta ja lisäksi, kun kyse on pelistä, on myös mietittävä, että onko pelin käyttöliittymä tarpeeksi viihdyttävä. Jos peli olisi tarkoitus julkaista, olisi tärkeää tiedostaa pelin kohdeyleisö. Tällöin olisi järkevää hankkia palautetta pelin käyttöliittymästä kohdeyleisöön kuuluvilta henkilöiltä ennen pelin varsinaista toteutusta.

Unity-pelimoottori sopii mainiosti Android-pelin toteutukseen. Unityn laajan yhteisön vuoksi onkin yleensä helppo löytää vastaus askarruttaviin kysymyksiin. Keskeneräisen pelin kokeileminen kohdelaitteella on mahdollista ja helppoa Unity Remote -sovelluksella, jolloin voidaan kokeilla, että toimiiko jokin pelin osa halutulla tavalla ja tehdä sen mukaan mahdollisia muutoksia.

Firebase tarjoaa selkeän dokumentoinnin omista palveluistaan Unitylle, joten Cloud Firestore -tietokannan hyödyntäminen Unityllä on helppoa. Cloud Firestoren käyttöä kannattaa kuitenkin miettiä tarkasti, jos tarkoituksena on julkaista peli. Tällöin on tärkeää tiedostaa mistä joutuu maksamaan ja otettava tämä huomioon jo pelin suunnitteluvaiheessa ja näin voidaan välttyä tarpeettomilta kustannuksilta.

Lähteet

Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Critical Success Factors to Improve the Game Development Process from a Developer's Perspective. *Journal of Computer Science and Technology*, *31*(5), 925–950. https://doi.org/10.1007/s11390-016-1673-z

Arnia Software. (2022). *What Makes Unity So Popular in Game Development?* Arnia Software. https://www.arnia.com/what-makes-unity-so-popular-in-game-development/

Ashok Kumar S. (2018). *Mastering Firebase for Android Development: Build real-time, scalable, and cloud-enabled Android apps with Firebase*. Packt Publishing Ltd.

Avisekhar, R. (2016). *The Android Game Developer's Handbook*.

Choose a database: Cloud Firestore or Realtime Database | Firebase. (2022). https://firebase.google.com/docs/firestore/rtdb-vs-firestore

Fatima, H. (2017, joulukuuta 12). Why is Firebase the best Mobile Backend-as-a-Service. *ResellerClub Blog*. https://blog.resellerclub.com/why-is-firebase-the-best-mobile-backend-as-a-service/

Figma. (ei pvm.). *Free, Online UI Design Tool & Software For Teams*. Figma. Noudettu 7. lokakuuta 2022, osoitteesta https://www.figma.com/ui-design-tool/

Firebase. (2022a). *Firestore*. Firebase. https://firebase.google.com/docs/firestore

Firebase. (2022b). *Supported data types | Firestore*. Firebase. https://firebase.google.com/docs/firestore/manage-data/data-types

Firebase. (2022c). *Understand Firebase projects | Firebase Documentation*. Firebase. https://firebase.google.com/docs/projects/learn-more

Firebase. (2022d, marraskuuta 15). *Add Firebase to your Unity project | Firebase for Unity*. Firebase. https://firebase.google.com/docs/unity/setup

Firebase. (2023). *Usage and limits | Firestore*. Firebase. https://firebase.google.com/docs/firestore/quotas

Hurja. (2021, elokuuta 23). Sovelluskehitys ja liiketoiminnan digitalisointi | Hurja. *Hurja Solutions Oy*. https://www.hurja.fi/blogi/sovelluskehitys-liiketoiminnan-digitalisoinnin-mahdollistajana/

Ite wiki. (2019, syyskuuta 6). *Sovelluskehitys | Ite wikin digitalisoinnin opas*. https://www.itewiki.fi/opas/sovelluskehitys/

James, D. (2013). *Android Game Programming For Dummies*.

Javatpoint. (n.d.). *Rapid Application Development Model | RAD Model—Javatpoint*. Www.Javatpoint.Com. Noudettu 27. joulukuuta 2022, osoitteesta https://www.javatpoint.com/software-engineering-rapid-application-development-model

Koskinen, I. (2021, maaliskuuta 26). *Mikä on Kanban? Katsaus menetelmään ja sen käyttöön ketterässä projektinhallinnassa*. Severa. https://severa.fi/blogi/mika-on-kanban-katsaus-menetelmaan-ja-sen-kayttoon-ketterassa-projektinhallinnassa/

Kristiadi, D., Udjaja, Y., Supangat, B., Yoga Prameswara, R., Spits Warnars, H. L. H., Heryadi, Y., & Kusakunniran, W. (2017). *The effect of UI, UX and GX on video games*. 158–163. https://doi.org/10.1109/CYBERNETICSCOM.2017.8311702

Llanos, S. C., & Jørgensen, K. (2011). *Do Players Prefer Integrated User Interfaces? A Qualitative Study of Game UI Design Issues*. 12.

Nielsen, J. (2020). *10 Usability Heuristics for User Interface Design*. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Nimble. (2022, marraskuuta 23). *What Is Kanban? An Overview Of The Kanban Method*. https://www.nimblework.com/kanban/what-is-kanban/

Pal, S. K. (2018, maaliskuuta 18). Software Engineering | Classical Waterfall Model. *GeeksforGeeks*. https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

Rădulescu, R. (2020, elokuuta 15). *Game Engine Basics · GDQuest*. GDQuest. https://www.gdquest.com/tutorial/getting-started/learn-to/game-engine-basics/

Roto, V., Law, E., Vermeeren, A., & Hoonhout, J. (2011). *Arnold Vermeeren Delft University of Technology The Netherlands a.p.o.s.vermeeren@tudelft.nl*. 12.

Scolanstici, C., & Nolte, D. (2013). *Mobile Game Design Essentials*.

TechTarget. (2021). *What Is an Application? Definition from SearchSoftwareQuality*. Software Quality. https://www.techtarget.com/searchsoftwarequality/definition/application

Tecinspire Oy. (2022). Miten valitsen uuden sovelluksen teknologian? *Tecinspire*. https://tecinspire.com/miten-valitsen-uuden-sovelluksen-teknologian/

Tutorials Point. (n.d.-a). *SDLC - Agile Model*. TutorialsPoint. Noudettu 13. helmikuuta 2023, osoitteesta https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

Tutorials Point. (n.d.-b). *SDLC - RAD Model*. TutorialsPoint. Noudettu 13. helmikuuta 2023, osoitteesta https://www.tutorialspoint.com/sdlc/sdlc_rad_model.htm

Tutorials Point. (n.d.-c). *SDLC - Waterfall Model*. Noudettu 9. joulukuuta 2022, osoitteesta https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Unity Technologies. (2017a). *Unity - Manual: GameObjects*. https://docs.unity3d.com/Manual/GameObjects.html

Unity Technologies. (2017b). *Unity—Manual: Asset Workflow*. https://docs.unity3d.com/560/Documentation/Manual/AssetWorkflow.html

Unity Technologies. (2018). *Unity - Manual: Prefabs*. https://docs.unity3d.com/Manual/Prefabs.html

Unity Technologies. (2022a). *Unity - Manual: Android environment setup*. https://docs.unity3d.com/Manual/android-sdksetup.html

Unity Technologies. (2022b). *Unity Manual*. https://docs.unity3d.com/Manual/UnityRemote5.html

Unity Technologies. (2023a). *Unity - Manual: Cameras*. https://docs.unity3d.com/Manual/CamerasOverview.html

Unity Technologies. (2023b). *Unity - Manual: Scenes*. https://docs.unity3d.com/Manual/CreatingScenes.html

Unity Technologies. (2023c). *Unity - Manual: Transforms*. https://docs.unity3d.com/Manual/class-Transform.html

Usablity.gov. (2014, toukokuuta 21). *User Interface Design Basics*. Department of Health and Human Services. https://www.usability.gov/what-and-why/user-interface-design.html

Vallaure, C. (2022, helmikuuta 3). *Figma: All you need to know*. Medium. https://uxplanet.org/figma-all-you-need-to-know-156b52b88e54

Liite 1: Aineistonhallintasuunnitelma

Tekniset tiedot projektista, kuten ohjelmistojen ja työkalujen versiot pidetään tallessa, jolloin mahdollisten vikojen selvittäminen ja korjaaminen helpottuu. Opinnäytetyön sisältö ja kaikki siihen liittyvä dokumentointi pidetään tallessa opiskelijan tietokoneella C-asemalla ja kyseiset tiedot myös varmuuskopioidaan muistitikulle säännöllisin väliajoin.