Kari Putkonen

ANDROID-PELIOHJELMOINTI UNITYLLA

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Joulukuu 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** 27.11.2013
Tekijä(t) Kari Putkonen	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittely
Nimeke Android-peliohjelmointi Unitylla	

Tiivistelmä

Pelikehitys on siirtynyt yhä enemmän ja enemmän mobiililaitteiden piiriin. Nykyään lähes jokaisella on taskussaan älypuhelin joko Android- tai iOS -käyttöjärjestelmällä tai laukussa tabletti vastaavilla käyttöjärjestelmillä. Mobiililaitteet ovat tuoneet pelaamisen yhä useamman ulottuville.

Opinnäytetyö ei pyri olemaan opas Unityn käyttöön vaan opinnäytetyön pääpaino on mobiililaitteiden näytön käytössä ja näytön erikoispiirteissä.

Aluksi esittelen työkalut, joita tarvitsee, että Unitylla voi tehdä pelejä Androidille sekä hieman kosketusnäytön erityispiirteitä. Opinnäytetyössäni selvitin, kuinka näiden mobiililaitteiden näyttöä pystyy ohjaamaan ja kuinka pelin tekeminen eroaa perinteisesti tietokoneelle tapahtuvasta peliohjelmoinnista Unitylla. Pyrin myös selvittämään, kuinka paljon Unityn valmiit mobiilipaketit helpottavat pelien tekemistä.

Käytännön toteutuksena tein yksinkertaisen sivusta kuvatun pelin Unityllä, jossa ei ole käytetty muita Unityn valmispaketteja kuin primitiivisiä muotoja sekä materiaaleja. Toteutuksen pohjalta pitäisi olla mahdollista luoda oma peli, mutta ohjeisiin joutuu mahdollisesti tekemään pieniä muutoksia riippuen käyttöjärjestelmästä jota käyttää.

Asiasanat (avainsanat)

Android, peliohjelmointi, Unity

Sivumäärä	**Kieli**	**URN**
34	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Jukka Selin	Jukka Selin

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis**
Author(s) Kari Putkonen	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

Android game development with Unity

Abstract

Game development has gradually been moving towards the mobile platforms. Almost everyone these days carries a mobile device with an Android or iOS operating system in their pocket or handbag. These new systems have brought gaming to an even wider audience. This bachelor's thesis did not try to be a complete tutorial for Unity. Instead, it focused on the touch screen and utilizing its special features.

First, I introduced the tools needed in order to make games for Android with Unity, as well as the special features of the touch screen. I studied how to control the screen of these mobile devices and how the development of the mobile games using Unity differed from the traditional desktop computer games. I also tried to figure out how much the mobile assets of Unity helped in creating a mobile game.

The practical part of this thesis was a simple mobile game that utilized the special features of the touch screen and only uses the primitive shapes and materials provided in Unity's assets. The practical part should help anyone in creating a game, but minor changes may be needed depending on the operating system used.

Subject headings, (keywords)

Android, game development

Pages	**Language**	**URN**
34	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	Jukka Selin

SISÄLTÖ

1 JOHDANTO

Peliohjelmoinnin historia ylettyy jopa 1950-luvulle saakka, jolloin tehtiin ensimmäinen hyvin yksinkertainen kahdestaan pelattava peli, jossa tarkoituksena oli lyödä omalla mailalla pallo vastustajan mailan ohi. Tämä peli oli myöhemmin julkaistavan Pong! pelin esi-isä (Everything2 2001). Ajat ovat muuttuneet paljon ja 50-luvulta nykypäivään mennessä on nähty useita pelikonsoleja sekä tietokoneen nousu pelikoneena. Myöskin pelintekomenetelmät ovat muuttuneet radikaalisti entisajoista, ennen periaatteessa kaikki koodattiin käsin ja nykyään on erinäisiä pelimoottoreita joissa pystyy graafisesti tekemään pelejä, muunmuassa Unreal Engine sekä FPS creator, joista FPS creator on hieman rajoittuneempi kuin opinnäytetyössäni käsittelemä Unity. Nykyisin melkein jokaisella on jonkinlainen älypuhelin taskussa joten puhelimelle pelien tekeminen on vain luonnollinen suunta jatkaa pelisuunnittelua.

Opinnäytetyössäni tutustun minkälaisia ongelmia ja haasteita peliohjelmointi mobiililaitteille asettaa ohjelmoijille. Minkälaisia rajoituksia pienempi näyttö, joka toimii kosketuksella, sekä rajallinen tallenustila asettaa pelintekemiselle. Kuinka Unity ja Unityn valmiit koodit ja materiaalit nopeuttavat ja helpottavat pelintekemistä.

Opinnäytetyön toisessa luvussa kerron mitä kaikkea tarvitsee, että Unitylla on ylipäätänsä mahdollista tehdä pelejä Android-alustalle. Pyrin antamaan mahdollisimman kattavat asennusohjeet näille ohjelmille ja kuinka ne lopulta saadaan työskentelemään yhdessä. Samassa luvussa tutkin hiukan Unityn tarjoamia mahdollisuuksia mobiiliohjelmointiin sekä muuta Unitysta ja peliohjelmoinnista.

Neljännessä luvussa kerron oman projektin luomisesta Android-alustalle alusta loppuun sekä hieman testauksesta ja optimoinnista. Neljännen luvun perusteella pitäisi pystyä luomaan oma yksinkertainen Android-peli Unitylla, pieniä muutoksia tehden riippuen mobiililaitteesta mille suunnittelit pelin tehdä. Viimeisessä luvussa pohdin hieman tulevaisuuden näkymiä sekä oman projektin onnistumista ja mikä sen kohtalo on tulevaisuudessa.

2 LAITTEET JA TYÖVÄLINEET

Jotta Android-laitteelle pystyy tekemään pelejä Unityllä tarvitsee asentaa muutama ohjelmisto koneelle. Asennettavat ohjelmat unityn lisäksi ovat ilmaisia ja tulevat samassa paketissa. Tietysti pelin kehittämistä ja testaamista helpottaa, jos omistat jonkinlaisen Android-mobiililaitteen.

2.1 Android

Android on Googlen, vuonna 2007 julkaisema käyttöjärjestelmä älypuhelimille ja muille mobiililaitteille. Android-käyttöjärjestelmästä on julkaistu myös kannettaville tietokoneille oma versio. Android on avoimen lähdekoodin alusta ja sille kehittäminen ja sen käyttäminen on ilmaista. Android-puhelimia valmistaa suuret yritykset, muutamina esimerkkeinä HTC, LG, Samsung, Motorola ja Sony, (Android Suomi 2013.), mutta myöskin Android-puhelimista on olemassa niin kutsuttaja halpamalleja, joita tehdään huonompilaatuisista komponenteista ja joiden hinta on tällä tavalla saatu alaspäin. Myös Google on julkaissut oman Nexus-sarjansa, jotka pohjautuvat Android-käyttöjärjestelmään ja tähän sarjaan kuuluu puhelin sekä tabletti.

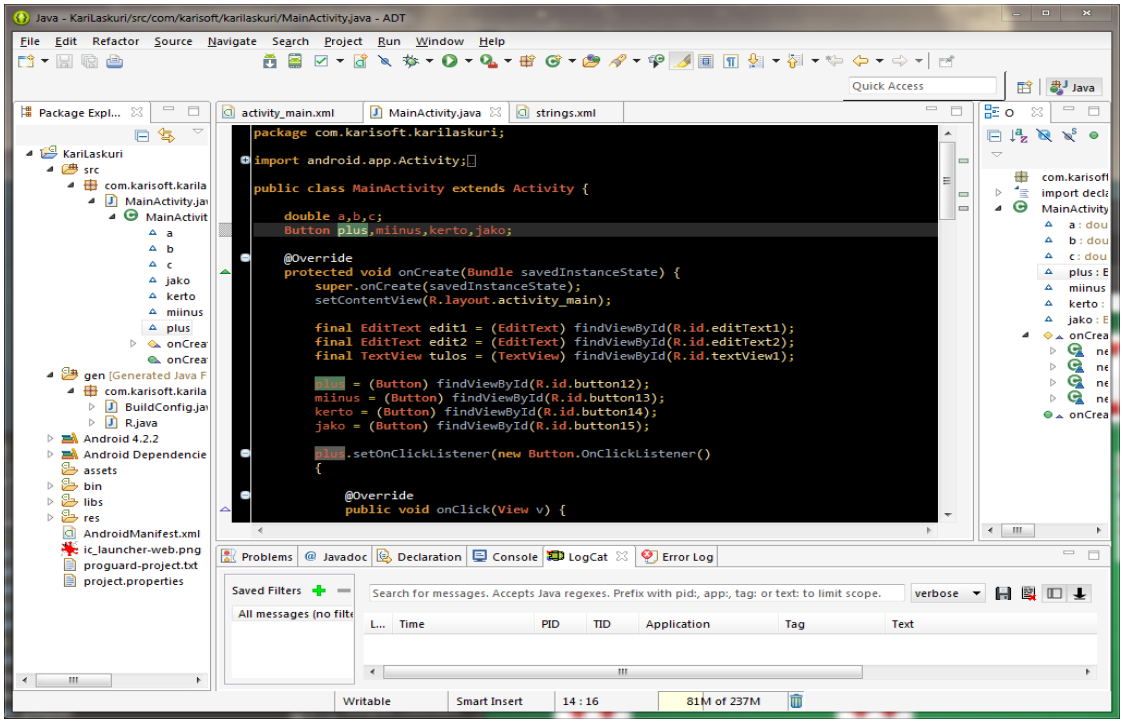
Android-laitteet vaihtelevat suorituskyvyiltään ja ominaisuuksiltaan hyvin paljon. Tehokkaimmat laitteet mahdollistavat FullHD-videoiden tallennuksen sekä vaativienkin 3D-ominaisuuksia käyttävien pelien ajamisen. (Android Suomi 2013.)

Android-käyttöjärjestelmä koostuu kahdesta osasta, käyttöjärjestelmän pohjana on Googlen mobiilikäyttöön muokkaama Linux ja Android-sovellukset toimivat Javaan perustuvan Dalvik-virtuaalikoneen päällä. Sovelluskehitys tehdään Java-kielellä ja Googlen tarjoamalla Android SDK -työympäristöllä, joka on ladattavissa ilmaiseksi. (Android Suomi 2013.)

2.2 Android SDK

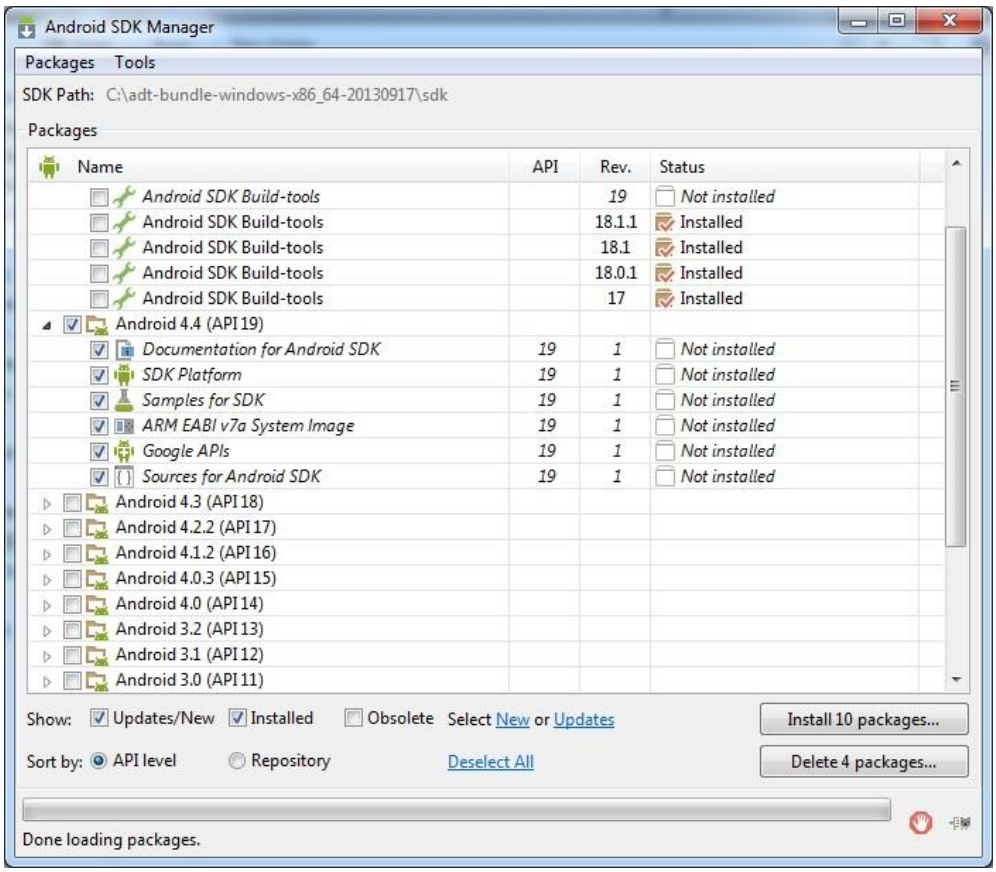
Android SDK sisältää API (Application Programming Interface) kirjastot ja tarpeelliset työkalut Android sovellusten kehitykseen, testaukseen ja virheiden etsintään (Android Developers 2013). Android SDK:n voi ladata Androidin kehittäjien kotisivulta http://developer.android.com/sdk/, SDK on saatavilla Windows-, Mac-

sekä Linux -käyttöjärjestelmille, joko 32- tai 64-bittisenä. Android SDK sisältää kaiken, mitä tarvitset omien Android-sovellusten tekemiseen, Eclipse+ ADT lisäosan, Android SDK työkalut, Android alusta-työkalut, viimeisin Android alustan sekä viimeisimmän Android järjestelmän levykuvan emulaattoria varten (Android Developers 2013). Mikäli Eclipse ohjelmointiympäristö on valmiiksi asennettuna, ei ole tarvetta ladata täydellistä Android SDK pakettia, Android kehittäjien sivuilla on paketti myöskin valmiiksi asennetun ohjelmointiympäristön käyttöä varten. Suurin ero pakettien välillä on Eclipse ohjelmointiympäristön puuttuminen.



KUVA 1. Android SDK:n mukana tullut Eclipse-ADT

Android SDK:n lisäksi tarvitsee myös Java Development Kitin (JDK), sekä Eclipse ohjelmointiympäristön (IDE), joka toimitetaan Android SDK:n mukana ellei sitä ole valmiiksi asennettuna. Viimeisimmän JDK:n pystyy lataamaan javan kehittäjän, Oraclen kotisivuilta http://oracle.com/technetwork/java/javase/downloads/. (Dornin ym. 2012, 3-4.)



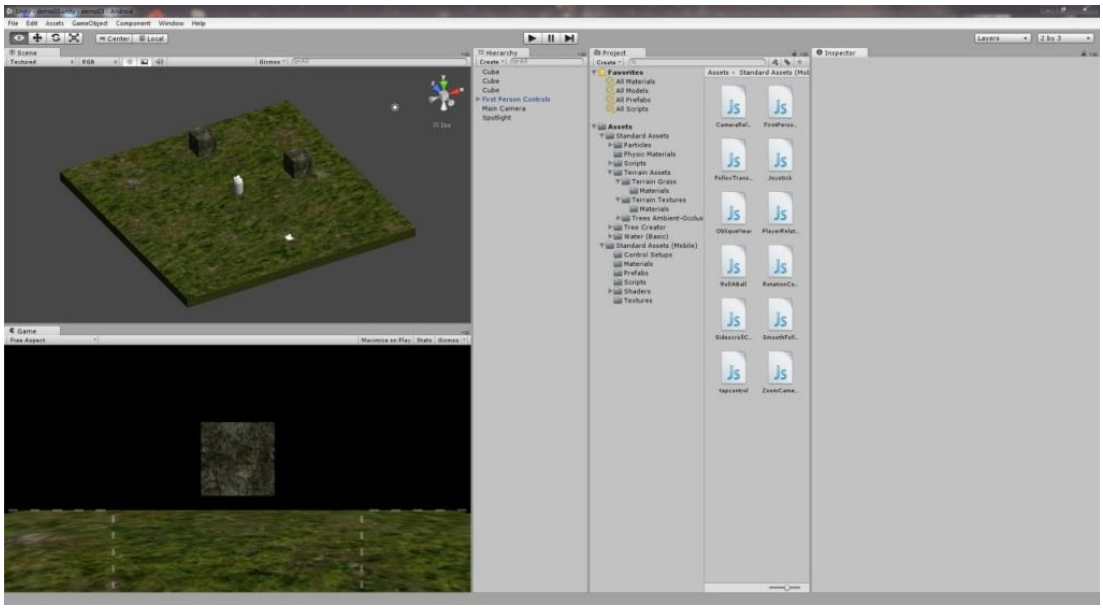
Kuva 2. Android SDK Manager

Jotta Unityllä pystyy kääntämään Android-paketteja, käyttäjän täytyy asentaa Android SDK:n mukana tulevalla Android SDK managerilla eri sovellusalustat ja levykuvat, joita haluaa käyttää ohjelmisto- tai pelikehityksessä (kuva 2).

2.3 Unity

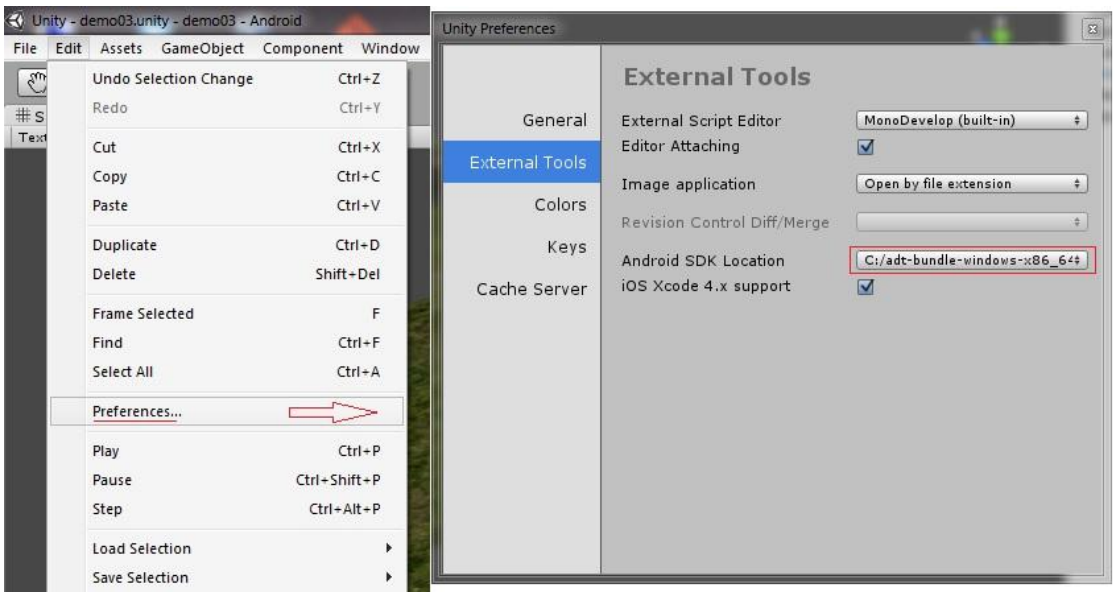
Unity on pelimoottori, jonka kehitys alkoi varhain 2000-luvulla kolmen nuoren ohjelmoijan toimesta. Ensisijainen kohdeympäristö Unityllä oli Mac-tietokoneet, joilla tuohon aikaan oli vielä pienempi markkinaosuus kuin nykyään, varsinkin pelialustoina. Vuonna 2005 Unitystä julkaistiin ensimmäinen versio, vuoteen 2008 mennessä pelimoottorista oli tullut kehittyneempi ja pelimoottori alkoi myödä paremmin. Nykyään Unityllä on 285 työntekijää ympäri maailman. (Brodkin 2013.) Unityllä on mahdollista tehdä niin 3D-pelejä kuin myös 2D-pelejä ja sillä on noin 2 miljoonaa rekisteröitynyttä käyttäjää ja noin 400 tuhatta kuukausittain aktiivista käyttäjää ja Unityn selainlisäosaa on asennettu 225 miljoonaa kertaa. Unityllä pystyy tekemään pelejä Windowsille, Macille, Linuxille, Unity Web Playerille, iOS:lle, Androidille, XBOX 360:lle sekä PS3:lle. Tulevaisuudessa pelejä pystyy tekemään

myös Wii U:lle, PS4:lle, Playstation Vitalle, Playstation Mobilelle, Windows Phone 8:lle, BlackBerry 10:lle ja Tizenille. Unity pelimoottoria käyttävät niin suuret yritykset, suuret ja pienet pelitalot, yksityiset kehittäjät, oppilaat ja harrastelijat. (Fun Facts 2013.)



KUVA 3. Unity

Jotta Unityllä saadaan käännettyä pelejä Android-ympäristöön Unityssa pitää olla Android-lisäosa, lisäosa on ilmainen Unityn perusasennuksessa, joka mahdollistaa .apk-pakettien luomisen Android SDK:n kautta. Tämän takia koneella, jolla Android-peliohjelmointia tehdään on oltava Android SDK asennettuna ja vähintään tuki Android 2.3 versiolle asennettuna Android SDK:sta.



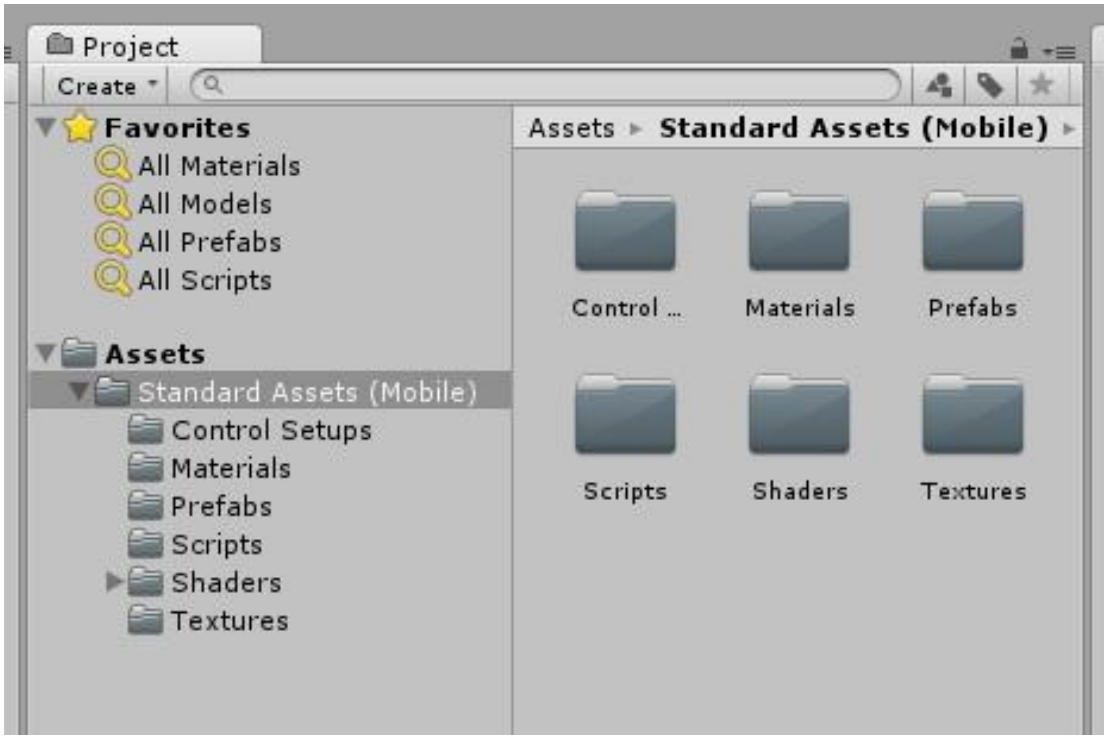
KUVA 4. Unityn ulkoisten työkalujen asetukset

Unity kysyy polkua Android SDK:n asennuskansioon, kun käännät ensimmäistä kertaa peliä Androidille. (Android SDK Setup 2013.) Vaihtoehtoisesti Android SDK:n polku voidaan määritellä Unityn asetuksista (kuva 4).

2.3.1 Unity Mobile Assets

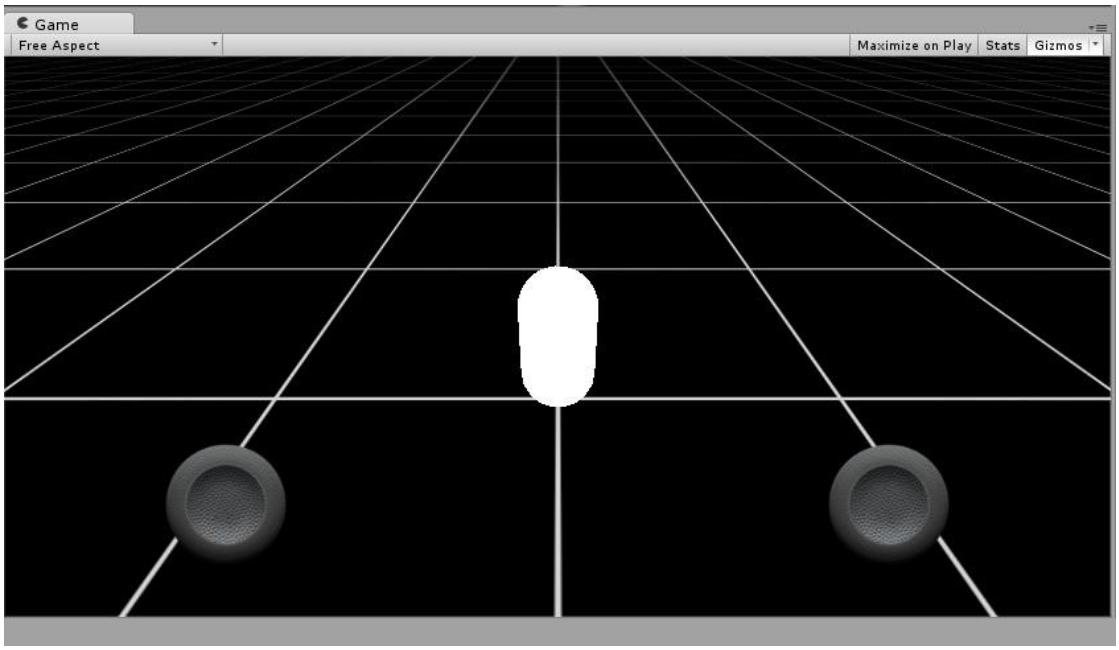
Unityllä työskennellä kaikkea ei tarvitse tehdä alusta asti itse. Unityn perusasennukseen sisältyy erinäisiä komponenttipaketteja, jotka sisältävät useasti tarvittuja ominaisuuksia, kuten ensimmäisen ja kolmannen persoonan hahmo-ohjaimet, maastotekstuureja, puiden luonnin, veden luonnin sekä monia muita paketteja. (Smith ym., 2013.)

Mobile Assets ei eroa edellisestä mitenkään, se sisältää usein käytettyjä mobiilipaketteja. Mobile asseteissa on pieniä eroja perinteisiin assetteihin verrattuna, esimerkiksi hahmon ohjaukseen käytettävistä skripteistä on esimerkkiversiot tarjolla, joilla pystyy testaamaan assettien toimivuutta nopeasti laitteella, jolle pelin aikoo tehdä, esimerkeissä on valmiiksi luotu skene, jossa on yksinkertainen esimerkki hahmon kontrolleista (kuvat 6-10.).



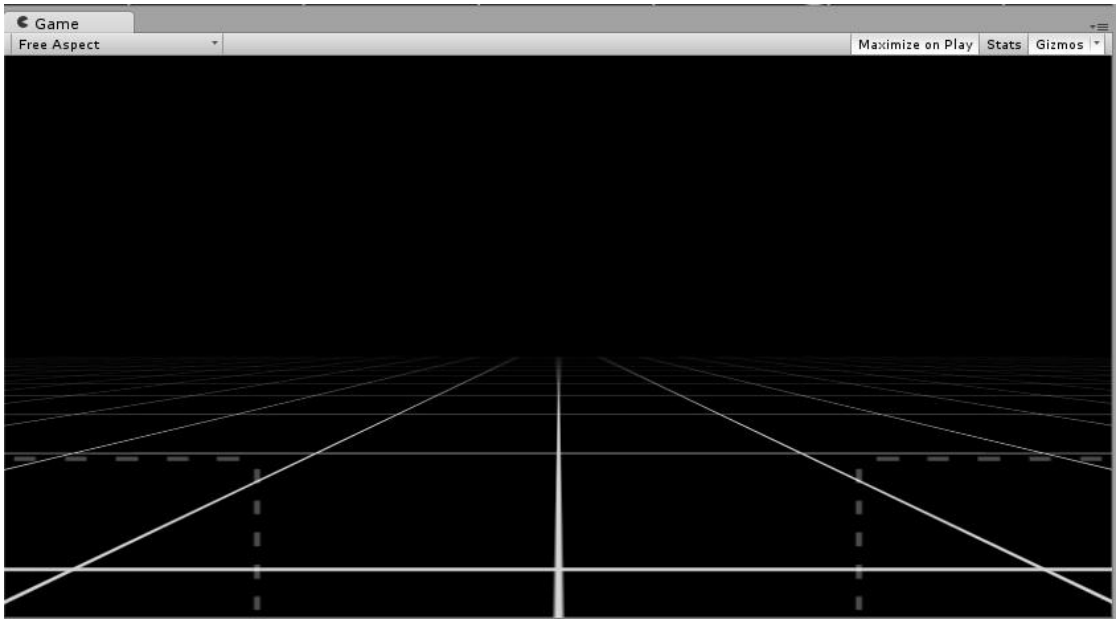
Kuva 5. Mobile Assets

Mobile assetit sisältää kuusi kansiota, Control Setups kansiosta löytyy aikaisemmin mainitut esimerkit eri hahmon kontrollointitavoista. Yleensä esimerkkiskeneissä on luotu yksinkertainen taso, johon on laitettu vastaava prefab-komponentti. Esimerkkejä näytön hallintaan, kuten esimerkiksi näytön liikuttaminen tai pinch zoom, mobile asseteista ei löydy vaan nämä täytyy kirjoittaa itse.



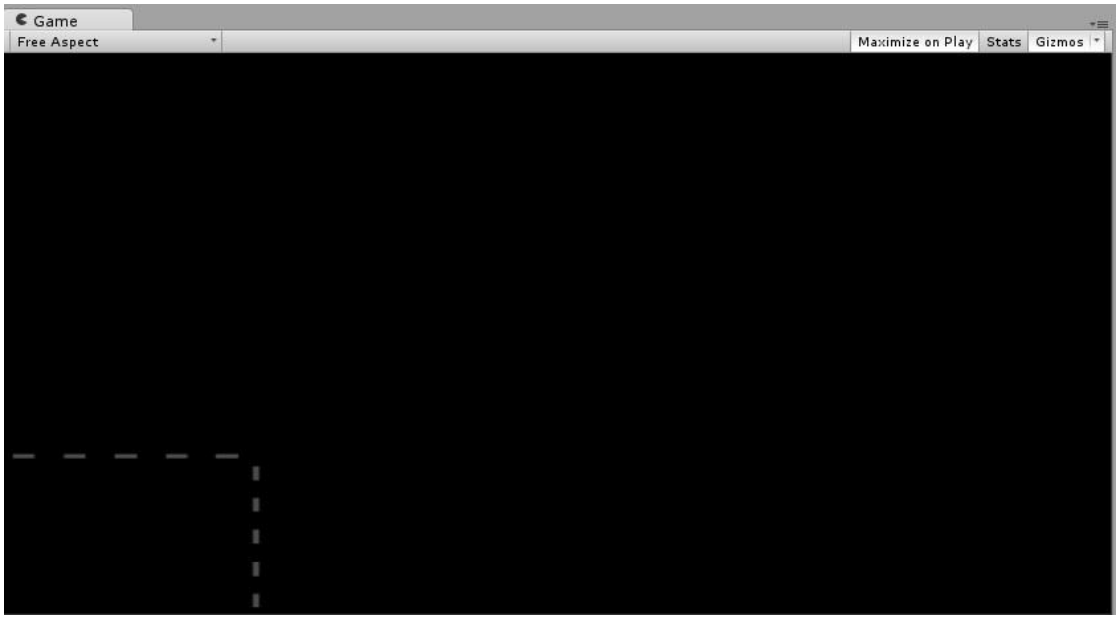
Kuva 6. Esimerkki kontrollipaketista, CameraRelativeSetup

Kuvassa 6 näkyvä kontrollipaketti seuraa pelaajaa kolmannesta persoonasta.



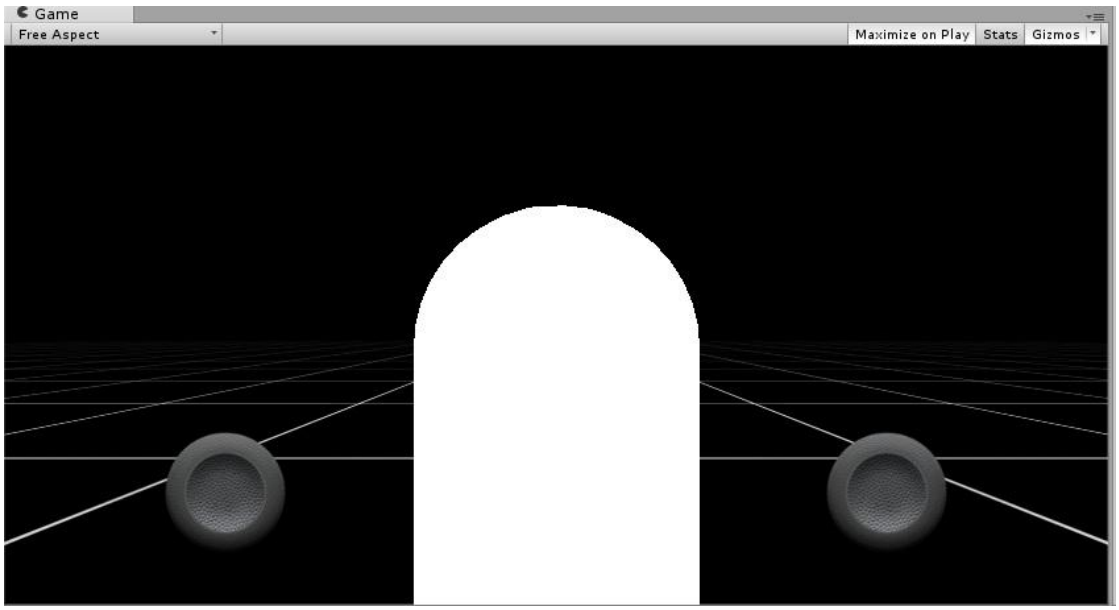
Kuva 7. FirstPersonSetup

Kuvassa 7 näkyvä kontrollipaketti on ensimmäisestä persoonasta kuvattu, hahmolla on mahdollista hypätä vasenta ohjausaluetta tuplanäpäyttäen.

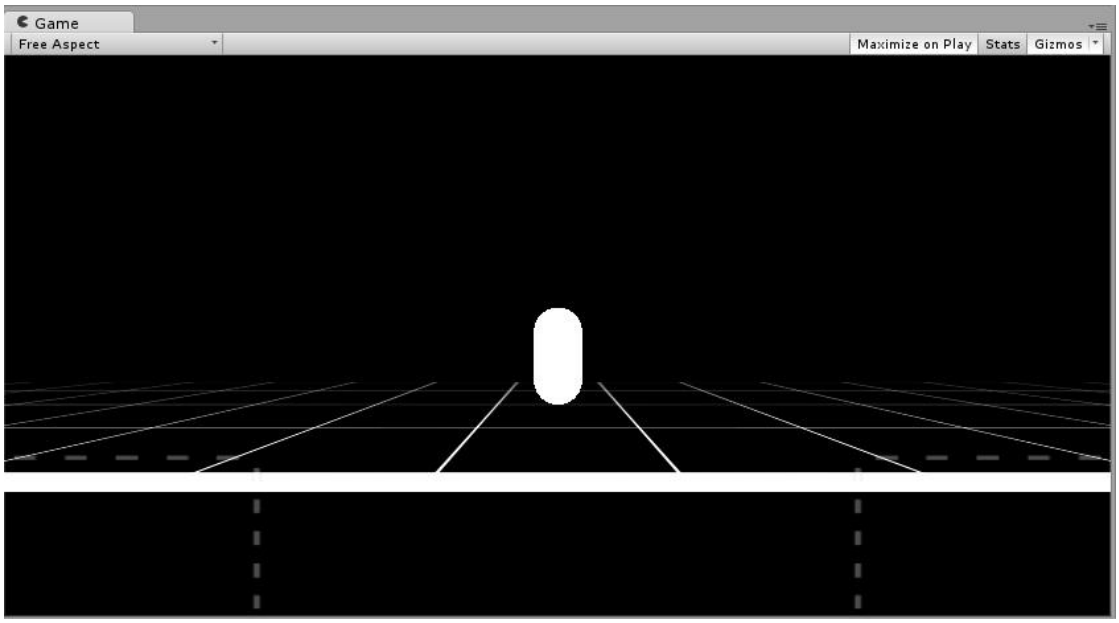


Kuva 8. FirstPersonTilt

Kuvassa 8 ei näi juuri mitään, mutta kontrollipaketin pitäisi toimia siten, että puhelinta kääntelemällä hahmo liikkuisi.



Kuva 9. PlayerRelativeSetup



Kuva 10. SidescrollSetup

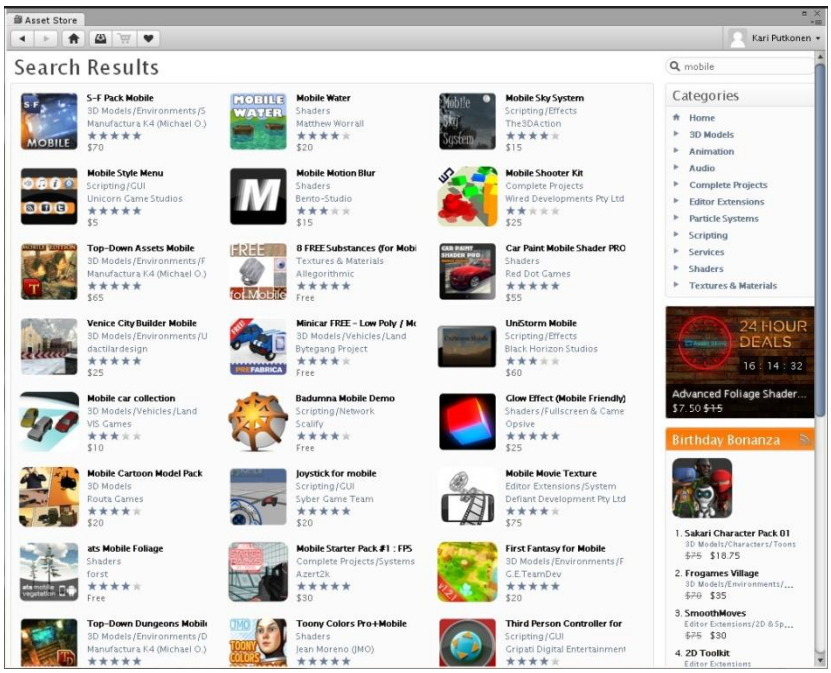
Kuvassa 10. näkyvä kontrollipaketti on aika yksiselitteinen, se on sivusta päin kuvattu ja oikeasta ohjausalueesta ylöspäin liu'uttamalla voi hypätä. Kuvista puuttuu vielä TapControlSetup, jonka kuvassa näkyy vain mustaa ja oikea joystick, nimen mukaisesti tässä ohjaustavassa liikutaan näpäyttämällä näyttöä.

Prefabs kansiossa puolestaan on valmiiksi luodut paketit eri ohjaustavoista, jotka on helppo lisätä omaan projektiin, tämä tarkoittaa sitä, että yleensä capsule

komponenttiin on lisätty tarvittavat materiaalit sekä koodit, jotta kehitys olisi mahdollisimman sulavaa. Lisäksi prefabs kansiosta löytyy kuvassa 6. nähtävillä olevat ohjaimet, ohjaimia on kahdenlaisia, kuvassa näkyvät joystick tyyliset sekä pelkästään rajatun alueen kokoiset laatikot joiden sisällä kosketus toimii. Scripts kansiossa on eri ohjaustyylien koodit, koska prefab-komponentit tarvitsevat näitä skriptejä, sekä näiden koodien avulla on mahdollista toteuttaa myös omat ohjaimet.

2.3.2 Unity Asset Store

Eräs nopea ja helppo tapa saada lisää paketteja projektiin on Unityn Asset Store, Asset Storesta löytyy ilmaisia ja maksullisia paketteja laidasta laitaan omiin projekteihin ladattaviksi taikka ostettaviksi. Mikäli pakettien tekijät ovat päättäneet pyytää rahaa omista tuotoksistaan he saavat 70% jokaisen myydyn paketin tuotoista, loput 30% menee Unitylle (Sell Asssets, 2013). Kauppapaikan käyttö on helppoa niin myyjän kuin ostajankin kannalta, myyjällä täytyy olla Unityn käyttäjätunnus. Myyjän täytyy myös lukea sopimukset liittyen kauppapaikkaan, ladata Asset Store Tool jonka jälkeen hän voi ladata pakettinsa kyseisellä työkalulla Unityn työntekijöiden tarkastettavaksi. Latauksen jälkeen työntekijät tarkastavat paketin haitallisen koodin tai sopimuksen vastaisen materiaalin varalta ja laittavat paketin kauppapaikkaan myyntiin (Sell Assets, 2013).



Kuva 7. Unity Asset Store

Unityn kauppapaikasta ostettaessa sinulla tulee olla käyttäjätunnus, jolle kaikki ostokset liitetään ja voit ladata ostamasi paketit ja materiaalit milloin tahansa haluat.

3 PINTAA SYVEMMÄLTÄ

Kosketusnäytön omaavissa puhelimissa ja tableteissa on erinäköisiä tapoja hallita näytön toimintaa, seuraavassa muutamia esimerkkejä. Pelimoottorina ja pelinkehitysvälineenä Unity on hyvin kehittynyt ja helppokäyttöinen, mutta sen pinnan alla on kuitenkin hyvin tehokkaita työkaluja pelien luomiseen, mutta Unityssäkään ei ole kaikkea valmiiksi sisäänrakennettuna vaan jotkut haluamasi ominaisuudet joudut itse luomaan.

3.1 Kosketusnäytön hallinta

Android-puhelimissa on ollut aina tuki yhden sormen sormieleille, joita ovat näpäytys sekä liu'utus. Android käyttäjille tuli kuitenkin eräänlainen "elekateus", koska esimerkiksi Applen iPhone tuki monisormieleitä joista tunnetuin lienee nipistysele. Android on tukenut versiosta 2.0 lähtien monisormieleitä. (Burnette, 2010.) Unityssä kosketusnäytön ohjauksesta vastaa Input.touches luokkakirjasto, jonka kautta pystytään ohjaamaan eri sormieleitä.

3.1.1 Tap touch (Näpäytys)

Perinteisin kosketusnäyttöpuhelimen ohjaustavoista, jossa käyttäjä koskettaa ruutua yhdellä sormella ja ottaa sen jälkeen sormen pois ruudulta. Esimerkiksi ohjelmat käynnistetään pikakuvakkeesta tällä sormieleellä. Android-peleissä tätä toiminnallisuutta käytetään esimerkiksi valikoissa tai pelin toimintanapeissa.

Esimerkiksi kuinka monta sormea näytöllä milloinkin on saadaan seuraavalla pienellä koodilla, esimerkkikoodi on kirjoitettu Javascriptillä (Input Touches, 2013).

```
function Update() {
var fingerCount = 0;
for (var touch : Touch in Input.touches) {
if (touch.phase != TouchPhase.Ended && touch.phase != TouchPhase.Canceled)
```

```
{
fingerCount++;
}
if (fingerCount > 0) {
print ("User has " + fingerCount + " finger(s) touching the screen");
}
```

Koodi 1. Tap touch

Esimerkissä ensin alustetaan muuttuja fingerCount joka pitää muistissa, montako sormea meillä on näytöllä. Tämän jälkeen tehdään toistolause, jossa katsotaan tuleeko lisää sormia ruudulle. Tämän jälkeen if-lauseessa tulostetaan fingerCount-muuttujan sisältö. If-lauseen sisältöä voidaan muuttaa koodin toiminnallisuutta.

3.1.2 Double touch (Tuplanäpäytys)

Tuplanäpäytystä käytetään Android-laitteilla zoomaamiseen tiettyyn kohteeseen, esim. internet selaimella voi zoomata tiettyyn osaan tekstistä taikka kuvaan. Tuplanäpäytystä käytetään myös tekstinvalintaan normaaleissa Android-sovelluksissa.

Mobiilipeliohjelmoinnissa tuplaklikkausta voidaan esimerkiksi hyödyntää hyppäämisessä. Unityn valmiissa first person control sekä side scroll control ohjauspaketeissa on valmiiksi implementoituna tuplahyppy.

3.1.3 Slide (Liu'utus)

Liu'utus on toinen Androidilla alusta asti olleista sormieleistä. Liu'uttamista voidaan esimerksiksi käyttää valikoissa joissa on yllättäen liukusäätimiä. Liu'uttamista voidaan myös käyttää Android-peleissä joystick-ohjauksessa. Unityyn on sisäänrakennettuna muutamia pelientekoa helpottavia skriptejä, jotka helpottavat ja nopeuttavat ohjelmointia. Yksi näistä on joystick.js -tiedosto, jolla saadaan Android-peliin määriteltyä alueet joilla sormea liu'uttamalla saadaan esimerkiksi pelihahmoa liikutettua tai tehtyä jotain muuta, mitä pelintekijä haluaakaan saada aikaiseksi. Liu'utuksen hallintaan löytyy myös esimerkkikoodi Unityn dokumentaatiosta, seuraava esimerkki on javascript kielellä.(Input.getTouch, 2013).

```
var speed : float = 0.1;
function update () {
if (Input.GetTouch > 0 && Input.GetTouch(0).phase == TouchPhase.Moved) {
var touchDeltaPosition:Vector2 = Input.GetTouch(0).deltaPosition;
transform.Translate (-touchDeltaPosition.x * speed, -touchDeltaPosition.y * speed, 0)
}
```

Koodi 2. Slide

Edellisessä esimerkissä alustetaan ensin muuttuja speed, jolla voidaan määritellä nopeus liikuteltavalle asialle. Tämän jälkeen tarkastetaan, onko näytöllä sormia ja Input.GetTouch(0).phase koodilla katsotaan onko sormi liikkunut näytöllä. Jos sormi liikkuu näytöllä niin transform.Translate koodilla siirretään objektia. Kyseisessä esimerkissä objekti liikkuu vastakkaiseen suuntaan kuin mihin sormea on liikutettu. Objektin saa liikkumaan samaan suuntaan kuin sormi, kun -touchDeltaPosition kohdista otetaan etuliitteenä oleva miinus pois.

3.1.4 Pinch Zoom (Nipistys)

Android-peliohjelmoinnissa tiettyjä asioita voidaan tehdä nipistyssormieleellä (kuva 8), esimerkiksi zoomaus, kuten normaaleissakin Android-sovelluksissa.

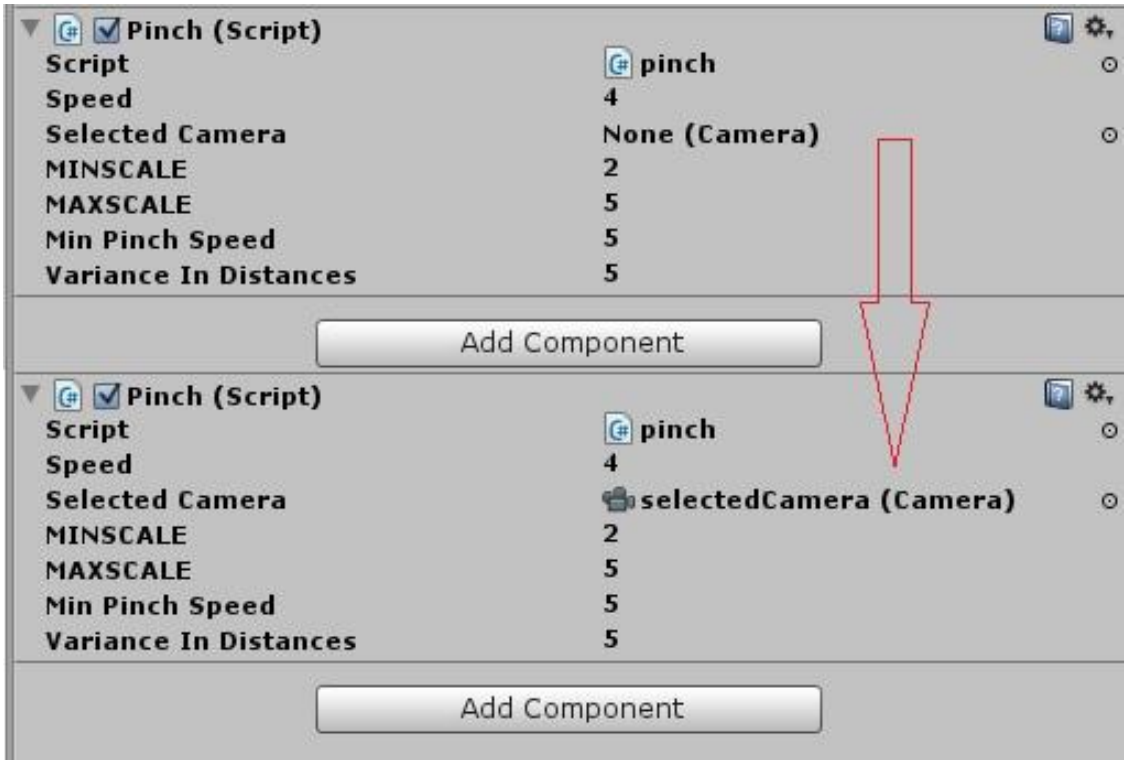
Kuva 8. Pinch zoom

Unityn virallisissa dokumentaatiossa ei ole esimerkkikoodjea tälle toiminnallisuudelle, mutta Unityn yhteisön keskustelualueelta löytyy useitakin esimerkkejä kuinka tälläinen sormiele on mahdollista toteuttaa. Seuraavaksi 1337GameDev nimimerkillä kirjoittavan jakama esimerkki Unityn keskustelualueelta osoitteesta http://answers.unity3d.com/questions/163614/easiest-way-to-do-pinch-zoom.html/, esimerkki on kirjoitettu C# kielellä ja testattu toimivaksi.

```
using UnityEngine;
using System.Collections;

public class pinch : Monobehaviour {
public int speed = 4;
public Camera selectedCamera;
public float MINSCALE = 2.0F;
public float MAXSCALE = 5.0F;
public float minPinchSpeed = 5.0F;
public float varianceInDistance = 5.0F;
private float touchDelta = 0.0F;
private Vector2 predDist = new Vector2(0,0);
private Vector2 curDist = new Vector2(0,0);
private float speedTouch0 = 0.0F;
private float speedTouch1 = 0.0F;

void Start () { }

void Update ()
{
if (Input.touchCount == 2 && Input.GetTouch(0).phase == Touchphase.Moved &&
Input.GetTouch(1).phase == TouchPhase.Moved)
{
// current distance between finger touches
curDist = Input.GetTouch(0).position - Input.GetTouch(1).position;
// difference in previous locations using delta positions
prevDist = ((Input.GetTouch(0).position - Input.GetTouch(0).deltaPosition) - (In-
put.GetTouch(1).position - Input.GetTouch(1).deltaPosition));
```

```
touchDelta = curDist.magnitude - prevDist.magnitude;
speedTouch0 = Input.GetTouch(0).deltaPosition.magnitude / Input.GetTouch(0).deltaTime;
SpeedTouch1 = Input.GetTouch(1).deltaPosition.magnitude / Input.GetTouch(1).deltaTime;

if ((touchDelta + varianceInDistances <= 1) && (speedTouch0 > minPinchSpeed) && (speedTouch1 > minPinchSpeed))
{
selectedCamera.fieldOfView = Mathf.Clamp(selectedCamera.fieldOfView + (1 * speed), 15, 90);
}

if ((touchDelta + varianceInDistances > 1) && (speedTouch0 > minPinchSpeed) && (speedTouch1 > minPinchSpeed))
{
selectedCamera.fieldOfView = Mathf.Clamp(selectedCamera.fieldOfView - (1 * speed), 15, 90);
}
}
}
```

Koodi 3. Pinch zoom



KUVA 9. Kameran valitseminen koodiin.

Eräs asia tätä koodiä käytettäessä tulee ottaa huomioon, kun tämän koodin liittää kameraan tai objektiin niin on erittäin tärkeää liittää koodiin selectedCamera parametri, jonka tulisi olla kamera johon koodi on liitetty (kuva 9).

3.2 Moninpeli

Moninpelin voi toteuttaa monella eri tavalla. Yleinen moninpelimuoto varsinkin vanhemmissa peleissä on samalla koneella toimiva moninpeli, jossa kukin pelaaja ohjaa omaa toimintaansa joko omalla tai yhteisellä ohjaimella paikallisella palvelimella tai lähiverkon yli. Internetin yleistymisen myötä myöskin internetin yli toimivat moninpelit yleistyivät.

Yksinkertaisimmillaan internetin yli toimiva moninpeli koostuu kahdesta eri rakennuspalasta ja niiden suhteesta, asiakas joka pyytää tietoa ja palvelin joka lähettää tietoa asiakkaille. Palvelin ja asiakas voi olla myös sama tietokone johon otetaan yhteyttä ulkoverkosta, kun palvelin on pystytetty ja asiakas on ottanut siihen yhteyden nämä kaksi tietokonetta voivat lähettää ja vastaanottaa tietoa pelin tarpeiden mukaan. (High Lever Networking Concepts, 2011.) Palvelimen ja asiakkaan luominen onnistuu molemmat Unityllä ja internetistä löytyy hyviä oppaita näiden tekemiseen.

3.2.1 Autoritatiivinen palvelin

Autoritaarinen palvelinta käytettäessä palvelin hoitaa kaiken simulaation, pelin "sääntöjen" toteuttamisen sekä pelaajien syötteet. Jokainen asiakastietokone lähettää palvelimelle tietoa, näppäinpainallusten tai toivotun toiminnon muodossa, ja jatkuvasti vastaanottaa tietoa pelin tilasta. Peli itse ei tee ikinä muutoksia pelin tilaan, vaan se lähettää halutun toiminnon palvelimelle joka puolestaan tekee tarvittavat toiminnot asiakkaan toiveiden mukaan ja lähettää tiedon asiakkaalle mitä pelin tilalle tapahtui.

Periaatteessa mitä asiakas haluaa tehdä ja mitä asiakas näkee ruudulla tapahtuvan välillä on pieni viive joka johtuu siitä, että palvelin vastaanottaa kaikkien asiakkaiden lähettämät tiedot ennen kuin palvelin päättää kuinka se päivittää pelin tilaa. Tälläinen palvelinmalli vaikeuttaa esim. huijaamista, koska asiakas ei pysty sanomaan palvelimelle, että joku toinen asiakas on kuollut, koska tästä toiminnallisuudesta vastaa palvelin. Asiakas voi vain kertoa palvelimelle, että ase ammuttiin paikasta x, suuntaan y ja palvelimen tehtäväksi jää päättää osuuko panos keneenkään.

Toinen esimerkki autoritatiivisestä palvelimesta on esimerkiksi fysiikkaan perustuva moninpeli. Jos tälläisessä pelissä asiakas voisi itse ajaa omia fysiikkasimulaatioita pienet muutokset pelissä mahdollisesti ajavat asiakkaiden pelejä epäsynkronisaatioon ajan kuluessa. Palvelimen hoitaessa kaiken pelin tilaan liittyvän voidaan varmistaa, että pelit ovat samassa pelitilassa jatkuvasti. (High Level Networking Concepts, 2011.)

Autoritatiivisen palvelimen huono puoli on viive joka muodostuu internetin yli lähetetyistä tiedoista. Tähän voi olla useita syitä ja tähänkin on keksitty ratkaisuja, yksi näistä on "asiakaspuolen ennustus" jolla tarkoitetaan sitä, että asiakas voi päivittää omaa paikallista versiota pelitilasta. Tällöin asiakkaan täytyy pystyä vastaanottamaan palvelimelta korjauksia pelitilaan. Ennustusta tulisi käyttää vain yksinkertaisten pelitoimintojen kanssa. (High Level Networking Concepts, 2011.)

3.2.2 Ei-autoritatiivinen palvelin

Ei-autoritatiivinen palvelin on autoritatiivisen palvelimen vastakohta kuten nimestäkin voi jo päätellä. Ei-autoritatiivinen palvelin ei hoida mitään pelilogiikan osa-alueita vaan se toimii eräänlaisena viestinviejänä asiakkaiden välillä. Asiakkaat itse laskevat fysiikan- ja pelitilan muutokset ja ilmoittavat mitä tapahtui palvelimelle. (High Level Networking Concepts, 2011.)

4 CASE TYKKIPELI

Omassa casessa yritin miettiä peliä, joka on yksinkertainen tehdä ja mihin saisi mahdollisimman paljon puhelimen kosketusnäytön toimintoja lisättyä. Pitkän mietinnän ja monien kokeilujen jälkeen päädyin tekemään yksinkertaisen sivusta kuvatun tykkipelin, jossa tarkoituksena on tuhota palloja ampumalla niihin. Tiukan aikataulun takia peliin ei ollut mahdollista lisätä hienoja grafiikoita ja muita hienouksia. Pyrin pitämään vaatimusmäärittelyn mahdollisimman yksinkertaisena toteuttaa ja keskityin enemmän mobiililaitteiden ominaisuuksien tuomisen peliin kuin siihen, että keskittyisin yleispätevään pelintekemiseen Unityllä. En ole myöskään käyttänyt Unityn mukana tulleita mobile assettien ohjaustoimintoja, jotka helpottavat mobiililaitteille kehittämistä uskomattoman paljon, syy tähän on se, että tälläiselle pelille valmiita assetteja ei juurikaan ole, mitä projektissa hainkin takaa.

4.1 Vaatimusmäärittely

Valikko, toiminnallisuuksille
Piipun nostaminen
Piipun laskeminen
Ampuminen
Voiman nostaminen
Voiman laskeminen
Kohteet, joita ammutaan
Kameran zoomaus sisään
Kameran zoomaus ulos
Kameran liikuttaminen pystysuunnassa
Kameran liikuttaminen vaakasuunnassa

Taustalla taivas

Tason vaihto

4.2 Suunnittelu

1. Sprint
- Ensimmäinen taso
- Tykki
- Maalit

2. Sprint
- Kameran skiptit
- Tykin koodit
- Maalien koodi

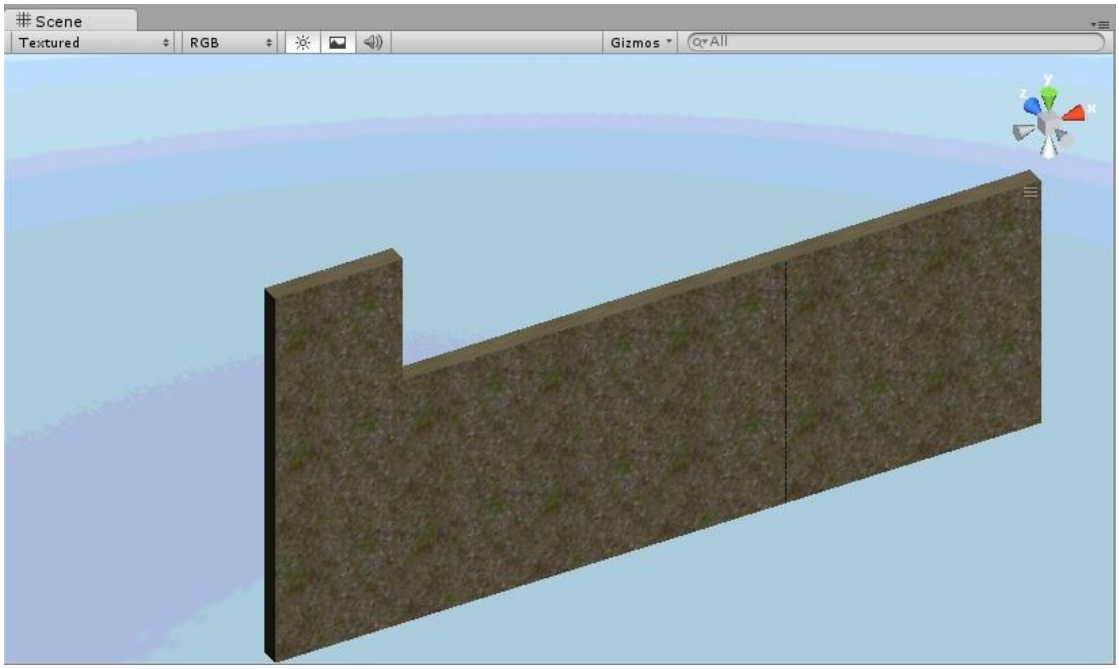
3. sprint
- Käyttöliittymä toiminnallisuuksille
- Päävalikon luominen

4.3 Toteutus

Pelin toteutuksen tein niin kutsutuissa sprinteissä, eli jaetaan projekti osiin ja tietyn aikavälin sisällä olisi tarkoitus tehdä tietyt asiat. Pelistäni löytyi kolme helposti eroteltavaa kokonaisuutta joten sprinttien sisällöt oli helppo keksiä ja sain pelin tehtyä loogisessa järjestyksessä.

4.3.1 Sprint 1

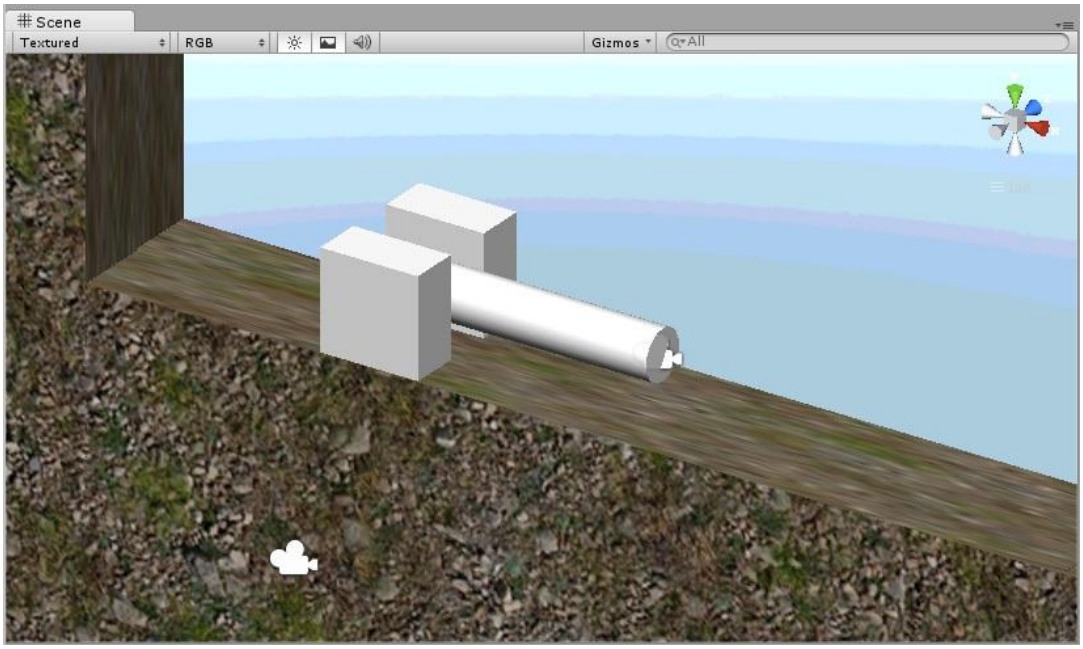
Aloitin pelinteon luomalla tyhjän unityprojektin johon otin mukaan Standard Assets ja Standard Assets (Mobile) paketit, paketeilla ei tässä vaiheessa ole suurta merkitystä, koska ne pystyy tuomaan projektiin myöhemminkin missä vaiheessa tahansa. Kaksiulotteisessa pelissä kentän luominen onnistuu mukavasti primitiivisillä muodoilla (kuva 10), kuutioiden mittakaavaa on muutettu isommaksi jotta niitä ei tarvitsisi satoja kappaleita sekä niille on määritelty materiaali ilmaisista asseteista joka onnistuu helposti raahaamalla materiaali kappaleen päälle Scene-ikkunassa.



Kuva 10. Ensimmäinen taso

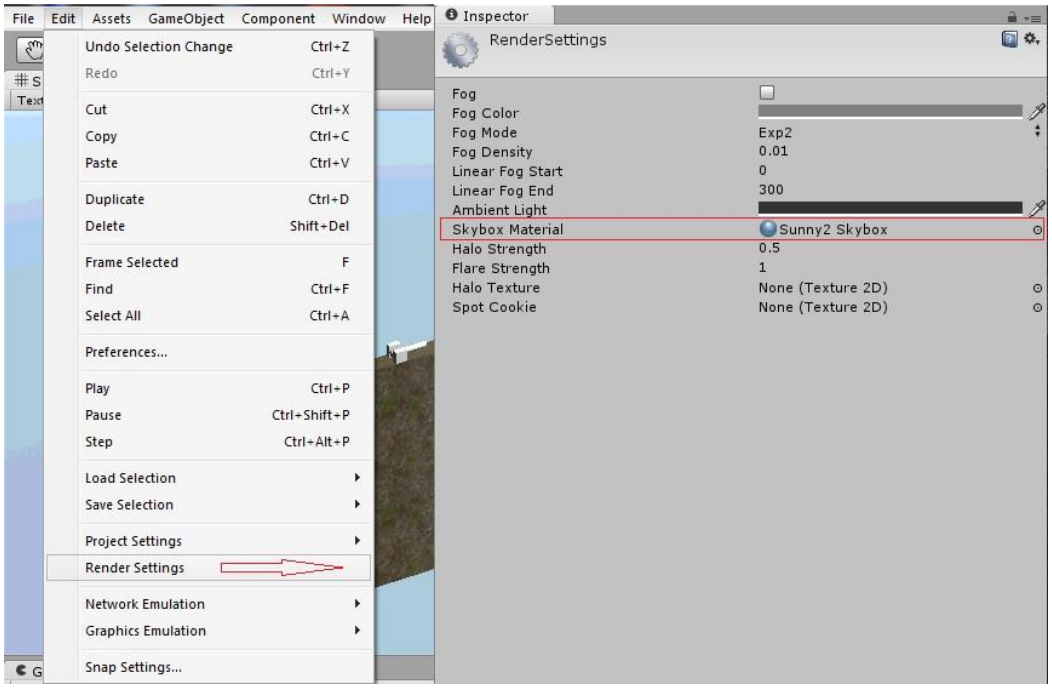
Käytin tykin luomiseen myös primitiivisiä muotoja, koska aika oli rajallinen ja yksittäisille primitiivisille muodoille on helpompi saada haluttu toiminnallisuus koodilla. Tykin luomiseen käytin kahta kuutiota sekä sylinteriä, sylinterin päähän sijoitin vielä toisen kameran jotta kuulat saisi lentämään ulos tykin suusta. Ammuksen luomiseen lennosta tarvitaan ammuksen prefab-tiedosto, prefab-tiedosto on kokoelma tietyn objektin ominaisuuksia ja muuttujia, tälläinen tiedosto kannattaa luoda, kun objektia kloonataan paljon ja olisi työlästä lisätä kaikki koodit ja toiminnallisuudet objektille joka kerta erikseen. Tässä tapauksessa prefab-tiedoston käyttö on lähes pakollista, koska ammukset luodaan lennosta, pelin pyöriessä. Uusien kenttien luomisen helpottamiseksi tykin piipustakin kannattaa tehdä prefab-tiedosto, tällöin kameraa ei tarvitse sijoittaa tykin päähän aina uudestaan, sekä siinä säästyy vaiva koodin lisäämisestä piipun pään kameraan.

Ammuksen prefab-tiedoston luominen onnistuu tekemällä Scene-ikkunaan halutun kokoinen pallo, laittamalla sille halutut koodit sekä materiaalit, jonka jälkeen voidaan luoda uusi, tyhjä prefab-tiedosto haluttuun kansioon. Tyhjän prefab-tiedoston luomisen jälkeen aikaisemmin luotu pallo vedetään prefab-tiedoston päälle Scene-ikkunasta. Nyt prefab-tiedosto voidaan nimetä ammus1:ksi ja poistaa alkuperäinen ammus Scene-ikkunasta. Muut prefab-tiedostot pystyy luomaan tällä samalla periaatteella.



Kuva 11. Tykki

Kohteet, joita ammutaan tykillä ovat normaaleja primitiivisiä pallo-objekteja. Palloihin on liitetty koodi joka tuhoaa ne (koodi 10), kun niihin osuu jokin toinen objekti, tässä tapauksessa tykin ammus, koska kyseessä on sivusta kuvattu kaksiulotteinen peli täytyy ammuksen liikkumista rajoittaa z-akselilla ettei pallot lentele ihan mihin sattuu..



Kuva 12. Render Settings

Skyboxin lisäys peliin on yksinkertaista render settings valikon kautta (kuva 12), kuvassa 10 on laitettu skybox jo valmiiksi paikoilleen

4.3.2 Sprint 2.

Suunnittelin käyttäväni käyttöliittymässä GUI-kirjaston nappi-elementtejä, joten minun täytyi koodien kirjoitusvaiheessa ottaa tämä huomioon ja kirjoittaa periaatteessa jokainen toiminnallisuus omaan funktioon, jotta nappia painettaessa pystyn kutsumaan tiettyä funktiota ja suorittamaan sen koodin. Funktiot alkavat sanalla void ja sulkeiden sisällä oleva koodi ajetaan sitä kutsuttaessa, muuttujat yleensä alustetaan funktioiden ulkopuolella.

Tykkiin liittyvät koodit kannattaa kirjoittaa kaikki samaan tiedostoon, koska tämä helpottaa nappien luomista myöhemmin. Panoksen ampuminen tykistä on helppoa, seuraavalla koodillä luodaan uusi ilmentymä aikaisemmin luodusta ammus1 prefab-tiedostosta. Ensiksi alustetaan uusi julkinen Rigidbody ammus1 muuttuja tyhjäksi, jotta tähän voidaan myöhemmin lisätä inspector ikkunasta ammus1 prefab. Voima ja suunta muuttujia käytetään koodin loppupäässä kertomaan mihin suuntaan ja millä voimalla panoksen tulisi lähteä.

```
public Rigidbody ammus1 = null;
public float voima = 10000;
public Vector3 suunta = new Vector3(1f, 0.5f, 0f);

void ammuPanos()
{
Rigidbody ammus = (Rigidbody)Instantiate(this.ammus1, this.transform.position, this.transform.rotation);
ammus.name = "ammus"
ammus.rigidBody.Addforce(this.suunta * this.voima);
}
```

Koodi 4. Panoksen ampuminen

Tykin piipun nostaminen ja laskeminen on hyvin yksinkertaista, mutta piipun kulmaa muutettaessa täytyy ottaa huomioon se, että piippuu pyörii origonsa ympäri kulmaa

muutettaessa. Tämän takia piipun x-koordinaattia täytyy muuttaa samassa suhteessa kulman kanssa, jotta saadaan suhteellisen sulava animaatio aikaiseksi piipun nostamisesta. Laskeminen hoituu periaatteessa samalla koodilla, ainoa ero tykin nostamiseen on se, että tykin laskemisessa positiiviset arvot on vaihdettu negatiivisiksi. Ensiksi etsitään piippu-peliobjekti ja tämän jälkeen vaihdetaan sen arvoja nappia painamalla.

```
void nostaTykki()
{
// muuttaa piipun y-akselia
GameObject.Find("piippu").transform.Rotate(0f,0f,2f);
// muuttaa piipun x-koordinaattia
GameObject.Find("piippu").transform.position += new Vector3(0f, 0.1f, 0f);
}
```

Koodi 5. Tykin piipun nostaminen

```
void laskeTykki()
{
GameObject.Find("piippu").transform.Rotate(0f, 0f, -2f);
GameObject.Find("piippu").transform.position += new Vector3(0f, -0.1f, 0f);
}
```

Koodi 6. Tykin piipun laskeminen

Voiman muuttaminen koodin avulla on hyvin yksinkertaista, meidän täytyy vain lisätä haluamamme summa voima muuttujaan tai vähentää vastaavasti tietty summa samasta muuttujasta, joka on alustettu jo tykillä ampumista varten.

```
void nostaVoima()
{
// += on lyhenne voima = voima + 1000; kirjoitustyylistä
voima += 1000;
}
```

Koodi 7. Voiman nostaminen

```
void laskeVoima()
{
// vastaavasti - = (yhteenkirjoitettuna) tarkoittaa voima = voima - 1000;
voima -= 1000;
}
```

Koodi 8. Voiman laskeminen

Lisäsin myös voimamuuttujalla GUIText komponentin peliruutuun johon voima-muuttuja tulostetaan. Tämä tapahtuu yksinkertaisella koodillä joka lisätään tiedostoon funktioiden ulkopuolelle. Koodi on yksinkertaisuudessaan seuraavanlainen, etsitään tekstin peliobjekti ja vaihdetaan sen alkuperäinen teksti voiman muuttujan arvolla.

```
GameObject.Find("voima_teksti").guiText.text = "Voima " + voima;
```

Koodi 9. Voiman tekstin näyttö ruudulla

Maalien tuhoutuminen on myös hyvin lyhyt C# koodi joka ei tee muuta kuin tarkastaa, että osuuko objektiin toinen objekti ja tämän jälkeen hävittää objektin pelistä, koodi tulee luoda omaan tiedostoon jotta se on helppo lisätä maaleihin.

```
void onCollisionEnter()
{
Destroy(this.gameObject);
}
```

Koodi 10. Maalin tuhoaminen

Seuraavaksi kameran ohjaukseen käytettävät koodit, toinen näistä on aikaisemminkin esittelty pinch zoomin koodi ja toinen liu'utuksen koodi jolla kontrolloidaan kameran paikkaa pelissä. Päätin käyttää pelissäni liu'utus-koodista C#-versiota, se eroaa javascriptillä toteutetusta vastaavasta koodista vain ulkoasultaan. Molemmat kamerakoodit kirjoitetaan omiin tiedostoihin ja liitetään skenen pääkameraan.

```
using UnityEngine;
using System.Collections;

public class pinch : Monobehaviour {
```

```
public int speed = 4;
public Camera selectedCamera;
public float MINSCALE = 2.0F;
public float MAXSCALE = 5.0F;
public float minPinchSpeed = 5.0F;
public float varianceInDistance = 5.0F;
private float touchDelta = 0.0F;
private Vector2 predDist = new Vector2(0,0);
private Vector2 curDist = new Vector2(0,0);
private float speedTouch0 = 0.0F;
private float speedTouch1 = 0.0F;

void Start () { }

void Update ()
{
if (Input.touchCount == 2 && Input.GetTouch(0).phase == Touchphase.Moved &&
Input.GetTouch(1).phase == TouchPhase.Moved)
{
// current distance between finger touches
curDist = Input.GetTouch(0).position - Input.GetTouch(1).position;
// difference in previous locations using delta positions
prevDist = ((Input.GetTouch(0).position - Input.GetTouch(0).deltaPosition) - (In-
put.GetTouch(1).position - Input.GetTouch(1).deltaPosition));
touchDelta = curDist.magnitude - prevDist.magnitude;
speedTouch0 = Input.GetTouch(0).deltaPosition.magnitude / In-
put.GetTouch(0).deltaTime;
SpeedTouch1 = Input.GetTouch(1).deltaPosition.magnitude / In-
put.GetTouch(1).deltaTime;

if ((touchDelta + varianceInDistances <= 1) && (speedTouch0 > minPinchSpeed)
&& (speedTouch1 > minPinchSpeed))
{
selectedCamera.fieldOfView = Mathf.Clamp(selectedCamera.fieldOfView + (1 *
speed), 15, 90);
```

```
}

if ((touchDelta + varianceInDistances > 1) && (speedTouch0 > minPinchSpeed) &&
(speedTouch1 > minPinchSpeed))
{
selectedCamera.fieldOfView = Mathf.Clamp(selectedCamera.fieldOfView - (1 *
speed), 15, 90);
}}}
```

Koodi 11. Pinch zoom

```
using UnityEngine;
using System.Collections;

public class siirto : MonoBehaviour {
public float speed = 0.1f;
void Update()
{
if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
{
Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
transform.Translate(-touchDeltaposition.x * speed, -touchDeltaposition.y * speed, 0);
}
}
}
```

Koodi 12. Liu'utus

4.3.3 Sprint 3.

Koodien luomisen jälkeen vuorossa on nappien tekeminen, jotta funktioita voidaan ajaa. Tykin koodien kanssa samaan tiedostoon luodaan napit piipun nostolle, piipun laskulle, voiman nostolle, voiman laskulle ja itse ammuksen ampumiselle. Tämä onnistuu helposti Unityn GUI-kirjaston avulla, ensiksi toiminnallisuuksille luodaan laatikko johon napit sijoitetaan toiminnallisuuksien mukaan, GUI-kirjaston napit ovat hyvin helppo tapa lisätä toiminnallisuutta mobiilipeleihin, koska nappeihin sisältyy automaattisesti kosketuksen tarkastus joka muuten pitäisi hoitaa erillisellä koodilla.

Jotta voimme ajaa aikaisemmin luotuja skriptejä napeista sijoitetaan niiden sisään funktiokutsut.

Nappeja luodessa täytyy huomioida se, että Unityn peli-ikkunan ja puhelimen näytön välillä on hyvin suuri resoluutioero josta aiheutuu se, että vaikka laatikot olisivat hyvänkokoiset ja luettavissa olevat, saattavat ne olla täysin erikokoiset puhelimen näytöllä. Tämän takia projektiin täytyy tuoda oma fontti, jonka ominaisuuksia pystymme itse muokkaamaan, ja jota käytämme nappien kanssa. Fontin kokoa voi vaihtaa joko inspector-ikkunasta taikka koodilla, käytin samaa fonttia eri paikoissa joten suurensin sitä valmiiksi inspector-ikkunasta ja seuraavassa koodissä pienennän sitä hieman koodilla. Jotta fonttia voidaan käyttää asseteista siitä täytyy luoda julkinen muuttuja jonka voi vaihtaa inspector-ikkunasta oikeaan fonttiin. GUIStyle-muuttujalla voidaan fontti määrittää koodin sisällä GUI-elementeille.

```
public Font myFont;

voin OnGUI()
{
GUIStyle myStyle = new GUIStyle();
myStyle.font = omaFont;
myStyle.fontSize = 20;

// luodaan taustalaatikko, new Rect(x-koord., y-koord., x-leveys, y-leveys), "Otsikko")
GUI.Box(new Rect(10, 10, 150, 100), "Functions");

// luodaan napit laatikkoon ja samalla tarkastetaan painetaanko nappia
if (Gui.Button(new Rect(30,60, 100,40), "Ammu"))
{
ammuPanos();
}

// pistetään saman toiminnallisuuden vastakohdat samaan laatikkoon
GUI.Box(new Rect(10, 120, 150, 150), "Kulma");

if (GUI.Button(new Rect(30, 170, 100, 40), "Ylös"))
```

```
{
nostaTykki();
}
if (Gui.Button(new Rect(30, 220, 100, 40), "Alas"))
{
laskeTykki();
}
GUI.Box(new Rect(10, 290, 150, 150), "Voima");

if (GUI.Button(new Rect(30, 340, 100, 40), "Ylös"))
{
nostaVoima();
}
if (GUI.Button(new Rect(30, 390, 100, 40), "Alas"))
{
laskeVoima();
}
}
```

Koodi 13. Käyttöliittymän koodi

Mikäli tykin piipusta on tehnyt prefab-tiedoston sen lisäämällä uuteen skeneen myös samalla luo tykin käyttöön tarkoitetut napit, jotta tämä olisi relevanttia tarvitsemme useamman kentän ja jotta kenttää voisi vaihtaa, tarvitsemme koodin joka hoitaa tämän. Yksi helppo tapa lisätä kentän valintaan pääsy on lisätä yksi nappi lisää tykin kontrollointiin tarkoitettuun koodiin, tämä kentän valinta toimii samalla päävalikkona pelissä.

Tämä on yksinkertaista ja onnistuu aikasemmin esitellyillä komennoilla paitsi, että emme suorita mitään omaa kirjoittamaa funktiota vaan käytämme Application.LoadLevel() -funktiota eri skenen lataamiseen. Tätä laatikkoa luodessa täytyy ottaa huomioon mobiililaitteen näytön koko sitä sijoittaessa. Samsung Galaxy S3:ssa näytön resoluutio sivuasennossa on 1280*700 ja halusin seuraavan laatikon näytön oikeaan yläreunaan. Tämän napin voi periaatteessa lisätä mihin haluaa, mutta tällöin joutuu koordinaatteja muokkaamaan.

```
GUI.Box(new Rext(1160, 10, 120, 100), "");

if (GUI.Button(new Rect(1170, 30, 100, 40), "Menu"))
{
Application.LoadLevel(0);
}
```

Koodi 14. Valikon painike

Päävalikon tekemiisen käytämme samaa tekniikkaa kuin Menu-painikkeen tekemiseen. Application.LoadLevel() -funktion sulkuihin voidaan kirjoittaa joko kääntämisvaiheessa selville tuleva skenen indeksinumero (kuva11) taikka skenen koko nimi lainausmerkkeihin, molemmat tavat ovat täysin toimivia. Päävalikossa käytämme taas aikaseimmin projektiin tuotua fonttia jonka kokoa olemme muuttaneet inspector-ikkunasta. Päävalikon C#-koodi on yksinkertaisuudessaan seuraavanlainen.

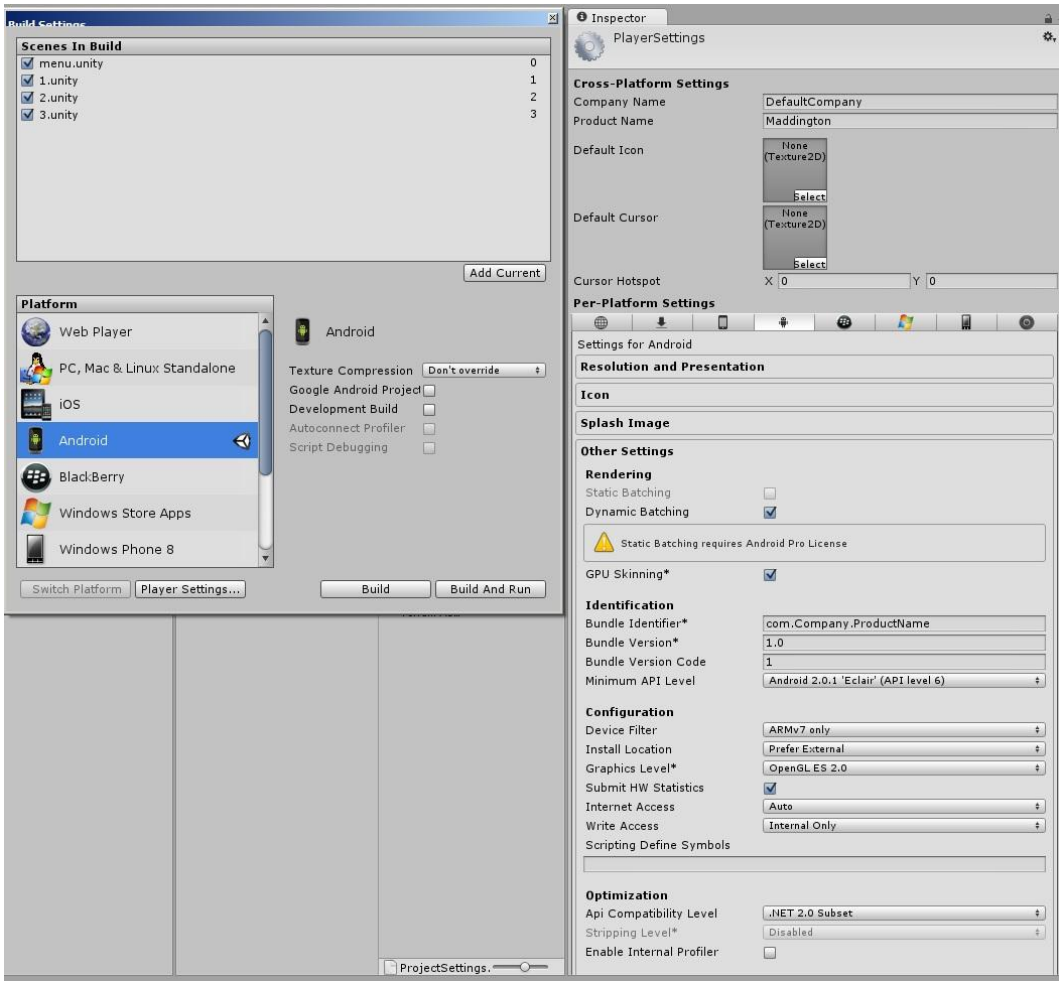
```
public Font myFont;

void OnGUI()
{
GUIStyle myStyle = new GUIStyle();
myStyle.font = myFont;

GUI.Box(new Rect(10, 10, 1260, 700), "Tason valinta");
if (GUI.Button(new Rect(335, 100, 600, 100), "Taso 1")){
Application.LoadLevel(1);
}
if (GUI.Button(new Rect(335, 250, 600, 100), "Taso 2")){
Application.LoadLevel(2);
}
if (GUI.Button(new Rect(335, 400, 600, 100), "Taso 3")){
Application.LoadLevel(3);
}
```

Koodi 15. Valikon koodi

Kuva 13. Build Settings sekä skenejen indeksinumerot

Unityllä pelejä tehdessä ne täytyy kääntää halutulle alustalle. Androidille käännettäessä (kuva 13) voidaan tehdä muutamia valintoja asetusten suhteen. Mikäli pelin tekemisen aloitti muulle kuin Android-alustalle, Switch Platform -napilla alustan voi vaihtaa Androidiksi jolloin Unity muokkaa paketit ja materiaalit Androidille sopiviksi. Player Settings avaa inspector-ikkunaan eri asetuksia koskien Unityn ajoa laitteella jolle peli tehdään. Asetuksissa on muutamia kohtia, jotka kannattaa katsoa läpi, ensimmäisenä inspector-ikkunassa näkyy oman ikonin ja kursorin valinta peliin, koska peli tehdään mobiililaitteelle kursoria ei tarvita mihinkään. Ikonin voi tehdä ja kannattaakin tehdä kaupallisille sovelluksille, koska se lisää tunnistettavuutta. Näytön suunnan voi lukita Resolution and Presentation välilehdeltä, Default Orientation

kohdasta, tämä on lähinnä mielipidekysymys kuinka päin puhelinta tykkää pitää kädessä, sivusta kuvattu peli tosin kannattaa laittaa pelattavaksi landscape asentoon, jolloin puhelinta pidetään sivuttain kädessä.



Kuva 14. Player Settings

Other Settings välilehdeltä voi muokata pelin versionumeroa sekä julkaisijan nimeä. Samalta välilähdeltä voi valita alimman Android API-tason, jolla peli toimii, nämä ovat suoraan verrannolliset siihen, mitä API-tasoja Android SDK:lla on asentanut.

4.4 Testaus

Pelin testaaminen onnistuu parhaiten puhelimella taikka mobiililaitteella mille peli on suunniteltu julkaistavaksi, peliä voi myös testata koodivirheiden varalta ajamalla sen Unityssä, jos pelin koodeissa on virheitä ei Unity suostu sitä edes puhelimeen siirtämään. Kuten mainittu Unitylla pystyy siirtämään pelin vaivattomasti mobiililaitteelle, kunhan kyseisen laitteen ajurit ovat asennettuna. GUI elementtien,

esim nappien testaus onnistuu suoraan Unitysta joka nopeuttaa toiminnallisuuksien testaamista huomattavasti (kuva 13), kun joka kerta pienen muutoksen jälkeen peliä ei tarvitse uudelleen siirtää puhelimeen.

Kuva 15. Testausta puhelimessa

Jos laitteelle ei löydy ajureita ja haluaisit testata sitä mobiililaitteessasi, käännösvaiheessa Unity kysyy mihin haluaisit luoda pelin .apk -paketin, joka sisältää kaikki pelin tiedostot. Liität vain laitteesi ulkoiseksi kiintolevyksi ja siirrät .apk paketin Android-mobiililaitteellesi ja asennat pelin laitteen paketinhallintajärjestelmällä ja voit testata peliäsi.

4.5 Optimointi

Aivan kuten tietokoneilla mobiililaitteidenkin suorituskyvyt vaihtelevat suuresti eikä ole yllättävää löytää esimerkiksi puhelinta joka on kymmenen kertaa tehokkaampi renderöimään kuvaa näytölle kuin jokin toinen malli. Yksinkertainen ohjesääntö skaalaukseen ja optimointiin on se, että ensin varmistaa huonolla puhelimella pelin toimivuuden ja tämän jälkeen lisäilee silmäkarkkia parempien puhelimien asetuksiin. (Mobile Optimisation, 2013.)

Optimointi on tärkeä osa pelikehitystä ja huomasin tämän omissa kokeiluissani selvästi, esimerkiksi Unityn omilla työkaluilla luotu maasto oli oli aivan liian raskas puhelimen prosessorille ja näytönohjaimelle, tämä asettaa rajoituksia pelinteolle, jolloin täytyy miettiä kuinka, tässä esimerkissä, pelin kentät toteutetaan. Mikäli pelinteossa käyttää 3D-objekteja, ne täytyy suunnitella hyvin, että ne olisivat mahdollisimman kevyitä, Asset Storessa on paljon 3D-objekteja myynnissä, jotka on suunniteltu juuri pelikäyttöä varten.

5 POHDINTAA

Unityn ja Androidin yhteensopivuus on todella hyvä, ohjelma ja mobiililaite työskentelevät saumattomasti yhdessä, sekä Unityn kehittäjät ovat lisänneet pelinkehittämistä tukevia assetteja paljon ohjelmaan. Pelinkehitys on nopeaa ja vaivatonta, Unitylla ennenkin pelejä tehneenä mobiililaitteet olivat uusi mukava tuttavuus, sellainen kuitenkin kulkee taskussa mukana joka päivä.

Omaksi projektiksi päätin valita sivusta kuvatun pelin, koska silloin siihen sai implementoitua muitakin kosketusnäytön eleitä kuin Unityn valmiilla ratkaisuilla, tämä rajoitti hieman valinnan varaa ja lopulta päädyin tykkipeliin, koska se on helposti laajennettavissa ja tykkipeliin on helppo keksiä lisää ominaisuuksia, sekä useat koodeista, joita pelissä käytetään ovat yleispäteviä ja uutta koodia ei hirveästi tarvitse kirjoittaa mikäli peliä haluaa laajentaa.

Projektin kulku oli selvän idean ja tutun ohjelman ansiosta todella sulavaa ja missään vaiheessa ei oikeastaan tarvinnut jäädä miettimään, että mitäköhän seuraavaksi pitäisi tehdä. Yhtenä huonona puolena pelissä on se, että se on todella pieni kooltaan sekä yllättävän kevyt, peliä testaillessa ei pysty koittamaan puhelimen rajoja. Tähän tarkoitukseen pitäisi tehdä paljon suurempi ja monimutkaisempi peli. Puhelimen näytön kanssa oli pieniä ongelmia, mutta niihin löytyi helposti vastauksia internetistä. Mobiilikehittäminen on yllättävän suosittua ja yleisimpiin ongelmiin varmasti löytyykin etsimällä vastaus Unityn keskustelualueilta. Peliä tehdessä oppi paljon Unityn mobiilikehitystyökaluista, sekä mobiililaitteille kehittämisestä yleensäkin.

Älypuhelimien markkinaosuudet tuntuvat vain kasvavan ja tehokkaampien puhelimien hinnat tulevat vain alaspäin joten laadukkaille mobiilipeleille tulee varmasti kysyntää. Suomestakin on ponnistanut kaksi yritystä maailmankartalle mobiilipeleillä, kenellekään tuskin on Rovio ja Supercell aivan tuntemattomia yrityksiä. Supercell on tehnyt mobiilipelillään tänä vuonna jo huimasti rahaa ja Rovion Angry Birds on itsestään jo huima myyntimenestys, mutta Angry Birds ei jäänyt pelkästään peliksi vaan poiki brändin.

Peliteollisuus on nykypäivänä nousussa ja Unityn kaltaiset ilmaiset työkalut luovat harrastelijoille ja aloittaville peliohjelmoijille hyvän alustan toteuttaa omia unelmiaan ja harjoitella taitojaan, hyville tekijöille on enemmän ja enemmän kysyntää sekä pelialaan erikoistuneita koulutusohjelmia on avattu useita eri puolille Suomea.

LÄHTEET

Android Developers 2013. Android Developers. WWW-dokumentti. http://developer.android.com/. Luettu 27.9.2013.

Android SDK Setup. 2013. Unity3d.com. WWW-dokumentti. http://docs.unity3d.com/Documentation/Manual/android-sdksetup.html/. Päivitetty 18.7.2013. Luettu 27.9.2013.

Android Suomi, 2013. WWW-dokumentti. http://blog.androidsuomi.fi/mika-on-android/. Luettu 27.9.2013.

Brodkin, Jon 2013. How Unity3D Became a Game-Development Beast. WWW-dokumentti. http://slashdot.org/topic/cloud/how-unity3d-become-a-game-development-beast/. Päivitetty 3.6.2013. Luettu 27.9.2013.

Burnette, Ed, 2010. Hello, Android Introducing Google's Mobile Development Platform. USA.

Dornin, Laird, Mednieks, Zigurd, Meike, G. Blake & Nakamura, Masumi 2012. Programming Android 2nd Edition. California: O'Reilly Media, Inc.

Everything2, 2001. The history of Video Game Programming. WWW-dokumentti. http://everything2.com/title/The+history+of+Video+Game+Programming/. Päivitetty 23.10.2001. Luettu 14.10.2013.

Fun Facts. 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/company/public-relations/. Luettu 27.9.2013

High Level Networking Concepts. 2011. Unity3d.com. WWW-dokumentti. http://unity3d.com/Documentation/Components/net-HighLevelOverview.html/. Päivitetty 18.11.2011. Luettu 27.10.2013.

Input Touches. 2013. Unity3d.com. WWW-dokumentti. http://docs.unity3d.com/Documentation/ScriptReference/Input-touches.html/. Luettu 2.10.2013.

Input getTouch. 2013. Unity3d.com. WWW-dokumentti. http://docs.unity3d.com/Documentation/ScriptReference/Input.gettouch.html/. Luettu 2.10.2013.

Mobile Optimisation. 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/Documentation/Manual/MobileOptimisation.html/. Päivitetty 12.8.2013. Luettu 3.11.2013.

Sell Assets. 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/asset-store/sell-assets/. Luettu 3.11.2013.

Smith, Matt, Queiroz Chico, 2013. Unity 4.x Cookbook. Birmingham: Packt Publishing Ltd.