OULUN AMMATTIKORKEAKOULU

Joonas Kopsala

2D-PELIN TOTEUTUS JA PUHELIMEN LIIKESENSORIT UNITY-YMPÄRISTÖSSÄ

2D-PELIN TOTEUTUS JA PUHELIMEN LIIKESENSORIT UNITY-YMPÄRISTÖSSÄ

Joonas Kopsala
Opinnäytetyö
Kevät 2017
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Joonas Kopsala
Opinnäytetyön nimi: 2D-pelin toteutus ja puhelimen liikesensorit Unity-ympäristössä
Työn ohjaaja(t): Jaakko Kaski, Niko Alakastari
Työn valmistumislukukausi ja -vuosi: Syksy 2017
Sivumäärä: 9 + 2 liitettä

Tämä opinnäytetyö toteutettiin kahdessa eri osassa. Ensimmäinen osa oli 5 opintopistettä ja toinen 10. Ensimmäisessä osassa tutustuttiin Unity-ympäristöön aloittelijan näkökulmasta sekä kehitettiin yksinkertainen peli. Työn alussa minulla ei ollut edeltävää kokemusta Unity-ympäristöstä, joten työ toimi hyvänä tapana oppia. Pelin ohjelmoinnissa käytettiin C#-kieltä.

Opinnäytetyön toisessa osassa tutustuttiin puhelimen liikesensoreihin ja toteutettiin pieni mobiilipeli Unity-ympäristössä käyttäen kyseisiä liikesensoreita pelikontrolleina. Työllä oli tilaajana FrozenVision Oy. Työhön kuului tutkimusosa, jossa kerättiin tietoa puhelimen liikesensoreista ja selvitettiin, miten ne toimivat. Työn lopussa perehdyttiin tarkemmin siihen, miten kyseiset sensorit toimivat Unity-ympäristössä ja miten niitä käytetään.

Molempien osien aikana saavutettiin kattava osaaminen Unityn parissa. Molemmissa töissä saatiin valmiiksi pieni peli, jonka avulla demonstroitiin työn tulosta. Töiden tavoitteet saatiin hyvin täytettyä ja tilaaja oli tyytyväinen. Kokonaisuudessaan työ tarjosi paljon tietoa Unity-ympäristöstä. Tutuksi tulivat niin 2D- kuin 3D-puolet.

Asiasanat: Unity, koosteopinnäytetyö, liikesensorit, peliohjelmointi, gyroskoopit, kiihtyvyyssensorit

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Joonas Kopsala
Title of thesis: 2D game development and smartphone motion sensors in Unity environment
Supervisor(s): Jaakko Kaski, Niko Alakastari
Term and year when the thesis was submitted: Fall 2017
Pages: 9 + 2 appendices

This thesis was done in two parts. First part is about developing a 2D game using Unity engine. It is written from beginner perspective. This part is about what Unity is and what can be done with it. It starts with studying Unity and ends with developing a small 2D game. Scripts in game are written in C# programming language.

Second part is about motion sensors in modern smart phones and how they are used in Unity game engine. This part explains how motion sensors work in Unity and how they can be used. Small mobile game was made during this part and it uses motion sensors as game controls. Scripts are written in C# programming language and game demo was made for company named FrozenVision Oy.

In whole, this thesis gives a good overview about Unity environment. It goes through both 2D and 3D engines. First part is more beginner oriented but second part is more theoretic and has advanced Unity features in it.

Keywords: Unity, C#, Motion Sensors, Gyroscope, Accelerometer, Programming

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö poikkeaa normaalista opinnäytetyöstä siten, että se suoritettiin kahdessa osassa. Ensimmäinen osa kirjoitettiin toisen lukuvuoden keväällä ja se on laajuudeltaan 5 opintopistettä. Toinen osa on laajuudeltaan 10 opintopistettä ja se kirjoitettiin viimeisen lukuvuoden lopulla. Opinnäytetyön osittaminen on aloitettu Oulun ammattikorkeakoulussa tietotekniikan koulutusohjelmassa vuonna 2014.

Työn ensimmäisessä osassa tutustutaan Unity-ympäristöön aloittelijan näkökulmasta. Työn edetessä kehitetään yksinkertainen 2D-peli ja tutustutaan Unityn eri ominaisuuksiin. Työn tavoitteena oli paneutua perusasioihin ja tuottaa esittelyversio pelistä.

Toisessa osassa paneuduttiin älypuhelimien liikesensoreihin ja niiden käyttöön Unity-ympäristössä. Työn tarkoituksena oli tutkia, miten liikesensoreita voitaisiin käyttää hyväksi, kun kehitetään mobiilipeliä Unity-ympäristössä. Työn tilaajana oli oululainen FrozenVision Oy. Työn tilaaja toivoi, että työstä saataisiin tuloksena pieni peli, jossa näitä tekniikoita käytettäisiin. Tarkoituksena oli lähinnä tutkia, mitä liikesensoritekniikalla on mahdollista saada aikaiseksi Unity-ympäristössä.

2 ENSIMMÄISEN OSAN ESITTELY

Tämän opinnäytetyön ensimmäinen osa tehtiin toisen lukuvuoden keväällä ja se on laajuudeltaan 5 opintopistettä. Työn aiheena oli yksinkertaisen 2D-pelin kehitys Unity-ympäristössä ja sen tavoitteena oli paneutua Unityn perusteisiin sekä tuottaa pieni peli. Tämän työn alussa minulla ei ollut edeltävää kokemusta Unity-ympäristöstä, joten työ on kirjoitettu aloittelijan näkökulmasta. Aiheen opiskelu toteutettiin itsenäisesti.

Tuotteena tästä työstä oli yksinkertainen 2D-peli. Peli on tyyliltään tasohyppelypeli, jossa kerätään kolikkoja. Pelin tarkoituksena on kerätä vaadittava määrä kolikoita, ennen kuin aika loppuu. Pelin kehittämisen aikana saavutettiin perustason osaaminen Unityn parissa.

Opinnäytetyön ensimmäisen osan tuotteena oli pieni peli. Tavoitteena oli oppia käyttämään Unity-ympäristöä ja saavuttaa perustason ymmärrys siitä, mitä Unityllä voi tehdä. Nämä tavoitteet saavutettiin hyvin ja Unity tuli tutuksi työn ohella.

3 TOISEN OSAN ESITTELY

Tämän opinnäytetyön toinen osa on laajuudeltaan 10 opintopistettä ja sen tilaajana toimi oululainen FrozenVision Oy. Toinen osa kirjoitettiin viimeisen lukuvuoden lopussa. Työn aiheena oli älypuhelimien liikesensorit ja niiden käyttö Unity-ympäristössä. Työn tuotteena oli pieni mobiilipeli, jossa käytettiin kontrolleina liikesensoreita.

Toisen osan tarkoituksena oli tutkia puhelimien liikesensoreita ja sitä, miten niitä käytetään Unity-ympäristössä. Työn alkupuolella käydään läpi kattavaa teoriaa gyroskoopista ja kiihtyvyyssensorista. Työn loppupuolella paneudutaan liikesensorien käyttöön Unity-ympäristössä. Lopuksi paneudutaan kehittämääni peliin. Tavoitteena oli saavuttaa kattava ymmärrys liikesensoreista ja niiden käytöstä Unity-ympäristössä.

Toisen osan tuloksena oli toimiva peli, jossa käytetään kontrolleina puhelimen liikesensoreita. Peliä pelataan kääntelemällä puhelinta. Kaikki ohjelmointi toteutettiin käyttäen C#-ohjelmointikieltä. Työn aikana saavutettiin kattava osaaminen liikesensoreihin ja Unityyn liittyen.

4 YHTEENVETO

Tätä opinnäytetyötä ei suoritettu yhtenä isona kokonaisuutena, vaan kahdessa eri osassa. Ensimmäinen osa tehtiin toisena vuonna ja toinen osa viimeisenä vuonna. Tästä syystä osissa on nähtävissä selvää kehitystä niin ohjelmoinnissa kuin yleisessä osaamistasossa. Osat yhteen sitova yhtäläisyys on Unity-ympäristö. Molemmissa töissä tuotoksena oli Unity-ympäristössä kehitetty peli.

Ensimmäinen osa oli arvokas oppimiskokemus, jonka avulla saavutin perustason osaamisen Unity-ympäristössä. Tästä oli myöhemmin hyötyä, kun sain harjoittelupaikan Unity-kokemukseni ansiosta. Toista osaa kehittäessä minulla oli jo kattava osaaminen Unityn parista, joten paneuduin enemmän teoreettiseen puoleen. Tämänkin osan parissa opin paljon ja pääsin tutkimaan mielenkiintoisia asioita. Kaiken kaikkiaan molemmat työt onnistuivat mielestäni hyvin ja tulokset olivat kiitettäviä.

Opinnäytetyön toteuttaminen osissa on mielestäni toimiva tapa vähentää ison opinnäytetyön aiheuttamaa taakkaa. Joillekin voi tietysti sopia iso työ paremmin, mutta luulen enemmistön nauttivan pienistä osista enemmän. Minun oli itsekin tarkoitus alun perin suorittaa opinnäytetyö kolmessa 5 opintopisteen osassa, mutta toisen osan tekemättä jättäminen aiheutti sen, että kokonaisuutena työstä tuli kaksiosainen.

LIITE 1

OULUN AMMATTIKORKEAKOULU

Joonas Kopsala

YKSINKERTAISEN 2D-PELIN TOTEUTUS UNITY-YMPÄRISTÖSSÄ

YKSINKERTAISEN 2D-PELIN TOTEUTUS UNITY-YMPÄRISTÖSSÄ

Joonas Kopsala
Opinnäytetyö 5op
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Tässä työssä tarkoituksena oli opetella ja tutustua Unityyn ja yrittää saada perustavanlaatuista kuvaa siitä, mikä Unity on ja mitä sillä on mahdollista tehdä. Minulla ei ollut edeltävää kokemusta Unityn käytöstä ja toivoin tämän kautta saavani jonkinlaista kokemusta pelimoottorista ja pelien kehittämisestä sitä käyttäen. Opettelemien toteutettiin itsenäisesti opiskelemalla internetistä löytyviä tutoriaaleja.

Tämän työn aikana toteutettiin yksinkertainen 2D-peli, jossa oli pelaajan hahmo, kaksiulotteinen pelikenttä, jossa on eri tasoja, sekä kerättäviä kolikkoja. Pelissä voidaan liikkua hahmolla sivuille ja hyppiä. Jos tippuu alas, kuolee. Pelin tarkoituksena on kerätä tarvittava määrä kolikkoja ennen kuin aika loppuu.

Työssä oli tarkoituksena paneutua perusasioihin ja saavuttaa perustavanlaatuinen osaaminen, josta lopuksi olisi näytteenä mahdollinen esittelyversio pelistä.

Tavoitteena oli myös yksinkertaisen 2D-pelin kehittämisen ymmärtäminen ja selvittää esimerkiksi mikä Unity on, mitä sillä on mahdollista tehdä ja miten kaikki toteutetaan.

Kiinnostus tähän aiheeseen lähti omasta mielenkiinnosta peleihin ja oman pelin tuottamiseen. Yleinen kiinnostus pelaamiseen ja peleihin on ollut olemassa jo ihan pienestä lapsesta asti.

2 PELIOHJELMOINTI

Pelien ohjelmoiminen ja tekeminen alusta saakka vaatii paljon työtä ja vie aikaa. Se vaatii myös hyviä ohjelmointitaitoja. Tästä syystä monet käyttävät valmiita pelimoottoreita, jolloin säästetään aikaa ja nopeutetaan pelin syntymistä verrattuna siihen, että kaikki tehtäisiin itse alusta asti. Pelimoottoreissa on yleensä valmiina fysiikanmallinnus, komentojen syöttäminen ja grafiikan piirtäminen. (4). Käytännössä pelimoottori on siis ikään kuin luuranko, jonka ympärille pelin tekijä rakentaa toimivan kokonaisuuden.

Markkinoilla on saatavilla monia eri pelimoottoreita ja kehitysympäristöjä. Näistä mainittakoon esimerkiksi CryEngine, joka on CryTek-yhtiön valmistama pelimoottori, Unreal Engine, joka sai vastikään paljon huomiota sen visuaalisen näyttävyyden ansiosta, ja tässä työssä käytetty Unity3D, jota voitaisiin kutsua kolmikosta alkeellisimmaksi, mutta minusta se on tässä tilanteessa pelkästään positiivinen asia.

Unity on aloittelijalle helpoiten lähestyttävä pelinkehitysympäristö, suurimmaksi osaksi sen helppolukuisen käyttöliittymän ansiosta. Siitä on saatavilla ilmainen versio, jossa on hieman vähemmän ominaisuuksia, ja maksullinen versio, jossa taas enemmän ominaisuuksia.

Unity tukee kolmea eri ohjelmointikieltä, jotka ovat JavaScript, C# ja Boo. Tässä työssä käytetään logiikan ohjelmoimiseen C#-kieltä, koska se on minulle ennestään tuttu ja omasta mielestäni myöskin helpoin.

Tässä työssä Unity on valittu kehitysympäristöksi juurikin siksi, että harrastelija- ja ammattiyhteisön keskuudessa yleinen mielipide on, että Unity on aloittelijalle hyvä valinta. Siinä on helppo ja selvä käyttöliittymä ja Internetistä löytyy paljon opiskelumateriaalia kyseiseen ympäristöön liittyen.

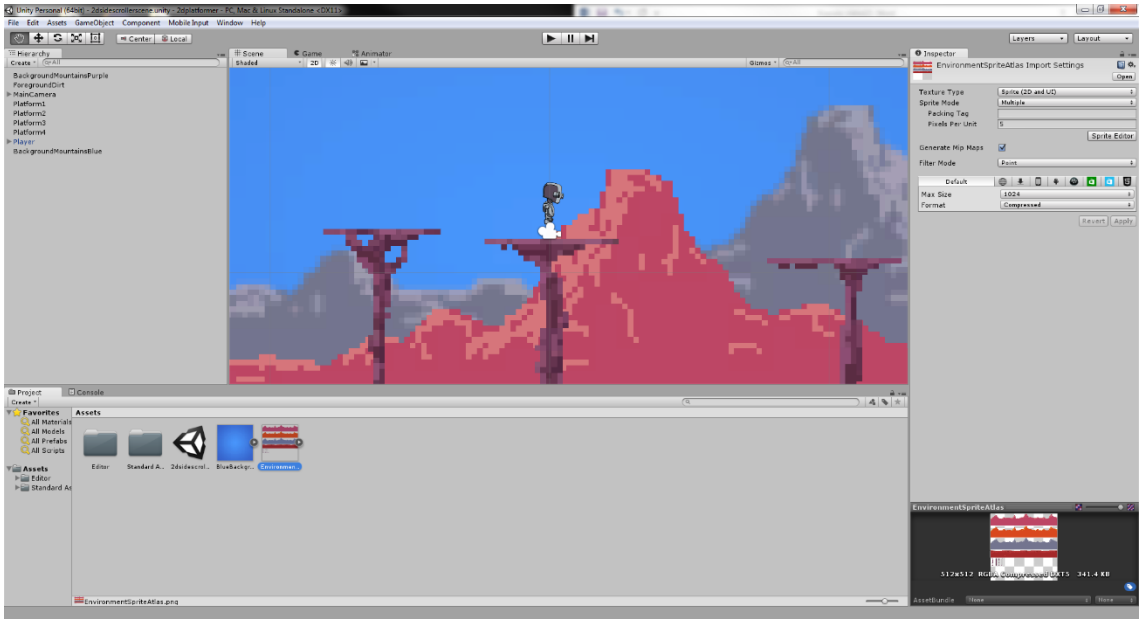
3 TEORIAA UNITYSTÄ

Pelimoottori (engl. game engine) on pelin ohjelmistokehys, jota hyödyntäen on mahdollista tehdä pelejä kaikille laitteille (5).

Unity on Unity Technologiesin kehittämä pelimoottori ja kehitysympäristö, jossa on renderöintimoottori sekä 2D- että 3D-grafiikalle. Se sisältää myös esimerkiksi fysiikkamoottorin, törmäyksen tunnistuksen ja animoinnin työkalut ja paljon muuta.

Uusin versio, eli Unity 5, julkaistiin maaliskuun 3. päivä 2015. Sen mukana tuli monia uusia ominaisuuksia, sekä parannuksia vanhoihin. Ensimmäinen versio Unitystä, Unity 1.0, julkaistiin 2005.

Kuvassa 1 näkyy Unityn käyttöliittymä, kun valittuna on 2D-esiasetukset. Vasemmalla näkyy luettelossa pelin eri elementit, keskellä itse pelinäkymä eli skene, oikealla asetusvalikko ja alhaalla asset-osio. Käyttöliittymä on tehty helppokäyttöiseksi ja selkeäksi. Tämän takia Unity onkin aloittelijoiden suosima.



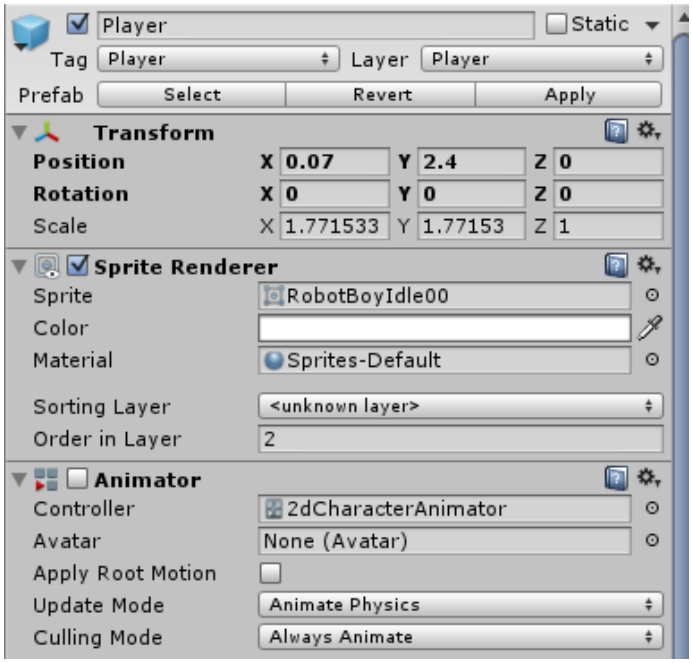
KUVA 1. Unityn käyttöliittymä.

4 PROJEKTIN TOTEUTUS

Kun aletaan tekemään 2D-peliä, luodaan aluksi uusi projekti ja jo luontivaiheessa valitaan projektille 2D-esiasetukset. Ensimmäinen asia mitä projektin luomisen jälkeen tehdään, on omia tarpeita vastaavan asset-paketin lataaminen. Assetit ovat käytännössä osia, joista projekti koostuu. Niihin sisältyy skriptit, hahmot ja kuvat, eli käytännössä kaikki. Unity 5:ssä oli perus-assetit valmiina, jolloin ei tarvitse ainakaan alkuvaiheessa ladata mitään. Näistä perus-asseteista voitiin lisätä projektiin suurin osa perusasioista, joita yksinkertaisessa pelissä tarvitaan.

4.1 Hahmon ja liikkumispinnan asetuksien tekeminen

Unityssä hahmo tunnistetaan antamalla sille Player-tagi, jolloin sille on saatavilla kaikki asetukset ja ominaisuudet, joita 2D-pelissä pelaajan hahmon hallintaan tarvitaan. Sille asetetaan collider, eli törmäysominaisuus, muokataan sen fysiikkaan liittyviä ominaisuuksia, tai ohjelmoidaan sille oma skripti, joka sisältää kaiken toiminnallisuuden, jota tarvitaan tietyssä asiassa.



KUVA 2. Joitakin pelihahmon asetuksia.

Hahmo voidaan valita asseteista ja se asetetaan ruudulle muuttamalla sen koordinaatteja asetuksista, tai sijoittamalla se haluttuun paikkaan hiirellä vetämällä. Yleensä tässä vaiheessa ei ole tarvetta alkaa tekemään suuria muutoksia hahmon asetuksiin, koska valmiissa hahmossa on tärkeimmät asiat valmiina. Hahmolle kannattaa kuitenkin asettaa oma taso, jotta se ei jää muiden elementtien alle, joita jatkossa projektiin lisätään.

Valmiissa asset-paketissa on monenlaisia vaihtoehtoja liikkumispinnalle, joten voi valita sen joka itseä miellyttää. Itse tässä vaiheessa valitsin tavallisen valkoisen palkkimaisen alustan. Se asetetaan hahmon alle ja se asetettiin törmäämään hahmon kanssa asettamalla sille collider. Nyt hahmo voi olla vuorovaikutuksessa kyseisen elementin kanssa, eli kävellä sen päällä. Tässä vaiheessa peliä pystyy jo ”pelaamaan”, sillä play-nappia painamalla hahmoa pystyy liikuttelemaan tason päällä nuolinäppäimillä.

4.2 Omien assettien tuominen projektiin ja niiden asentaminen

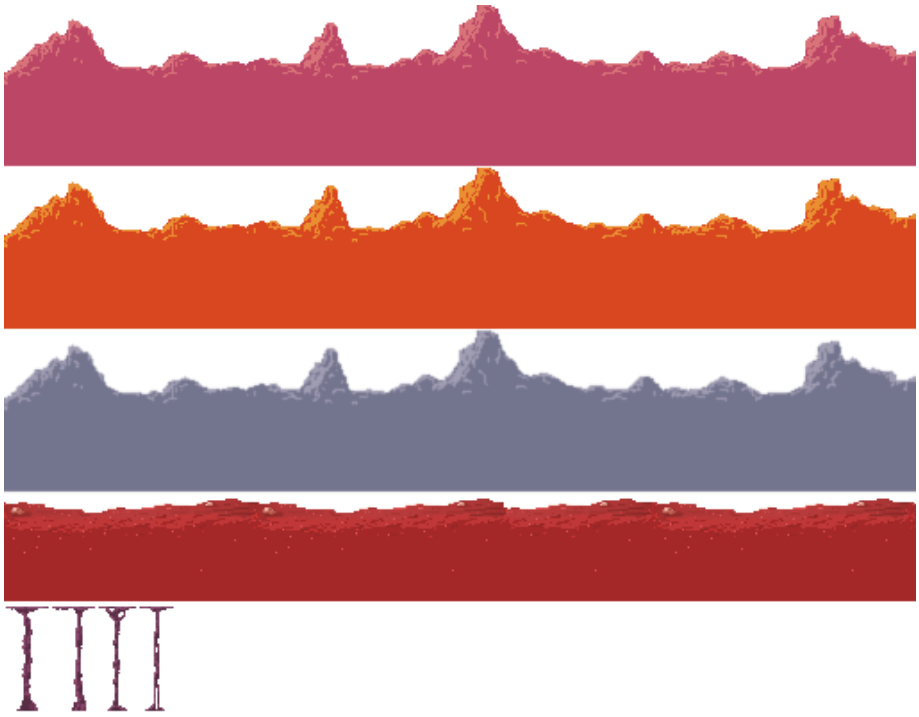
Unityssä on tietenkin mahdollista olla käyttämättä valmiita assetteja. Tällaisessa tapauksessa käyttäjä voi itse piirtää tai luoda omat hahmonsa, kenttänsä, vihollisensa ja esineensä. Tätä varten Unity tarjoaa työkalun, jolla esimerkiksi omien tasojen lisääminen peliin on suhteellisen helppoa ja vaivatonta.

Omien liikkumistasojen ja taustojen lisääminen projektiin onnistuu yksinkertaisesti vetämällä ne Unityyn Kuvan 1 alareunassa näkyvään assets-ikkunaan. Näitä kutsutaan spriteiksi. Unity tunnistaa esimerkiksi Photoshopin oman kuvaformaatin, mutta myös yleisimmät formaatit kuten jpeg ja png käyvät. Tilan ja selkeyden kannalta on yleensä viisasta laittaa esimerkiksi taustat samaan kuvaan (kuva 3).

Kuvassa 2 näkyvät eri taustaelementit voidaan eritellä omiksi, erillisiksi spriteiksi, Unityn Sprite Editorilla. Sprite Editorissa on automaattinen työkalu, joka erittelee eri osat erilleen automaattisesti, mutta usein se ei toimi ihan täydellisesti, jolloin erittely pitää viimeistellä käsin. Eri osille kannattaa myös antaa omat nimensä selkeyden parantamiseksi.

Kun omien assettejen tai spritejen asetukset ovat muuten kunnossa, voidaan ne lisätä projektiin vetämällä ne asset-osiosta. Tässä vaiheessa niitä on mahdollista skaalata ja

siirrellä ja muokata niiden asetuksia, jos tälle on tarvetta. Taustat saadaan lomittumaan keskenään asettamalla niille paikat, esimerkiksi punaiselle 0, siniselle -1 ja niin edes päin. Tällöin siis pienemmän arvon omaava on suuremman arvon omaavan takana ja luo illuusiota taustan syvyydestä.



KUVA 3. Omat alustat ja taustat.

4.3 Parallaksivieritys ja sen toteuttaminen

Parallaksivieritys (engl. Parallax scrolling) on grafiikkaan liittyvä menetelmä, jolla taustalla olevia kuvia liikutetaan eri nopeudella kuin etualalla olevia kuvia. Tällä tavalla luodaan illuusio syvyydestä. Tekniikkaa käytetään yleensä 2d-peleissä, mutta viime vuosina siitä tuli muoti-ilmiö myös nettisivuilla (3).

Parallaksiskrollaukseen ei ole Unityssä mitään valmista työkalua, joten se pitää toteuttaa skriptin avulla. Unityn omilla sivuilla on tarjolla tutoriaaleja ja valmiita skriptejä parallaksiskrollauksen toteuttamiseen, joten käytännössä tätä ei tarvitse itse osata ohjelmoida.

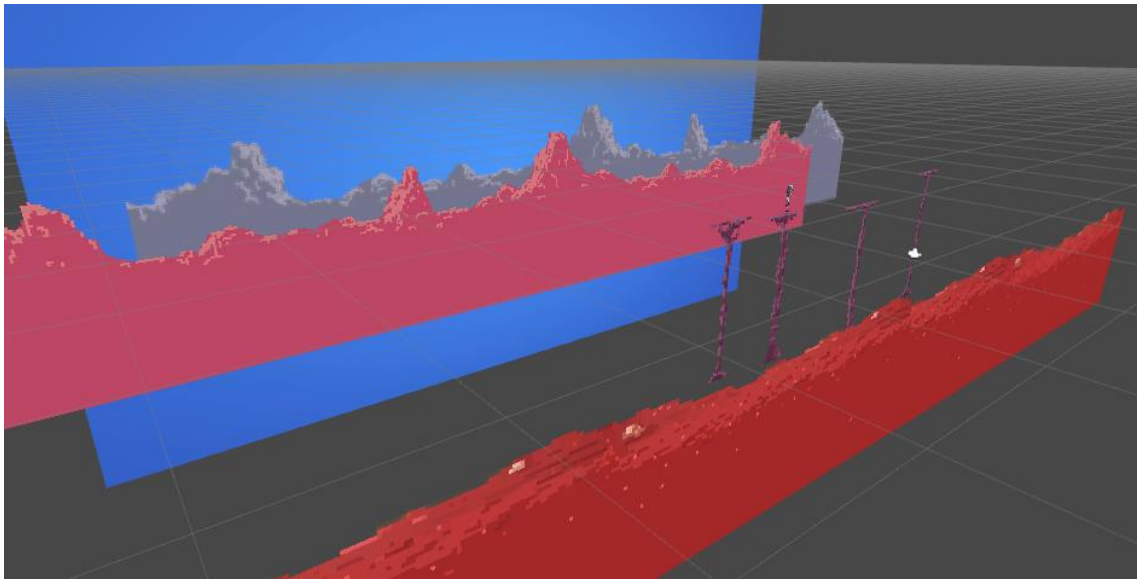
```
    // Update is called once per frame
    void Update () {

        // for each background
        for (int i = 0; i < backgrounds.Length; i++) {
            // the parallax is the opposite of the camera movement because the previous frame multiplied by the scale
            float parallax = (previousCamPos.x - cam.position.x) * parallaxScales[i];

            // set a target x position which is the current position plus the prallax
            float backgroundTargetPosX = backgrounds[i].position.x + parallax;

            // create a target position which is the backgrounds current position with its tarhet x's position
            Vector3 backgroundTargetPos = new Vector3 (backgroundTargetPosX, backgrounds[i].position.y, backgrounds[i].position.z);

            // fade between current pos and the target pos using lerp
            backgrounds[i].position = Vector3.Lerp (backgrounds[i].position, backgroundTargetPos, smoothing * Time.deltaTime);
        }

        // set the previousCamPos to the camera's position at the end of the frame
        previousCamPos = cam.position;
    }
}
```

KUVA 4. Ote skriptistä Parallaxing.cs.

Kun skripti on valmis, eli ohjelmointi on suoritettu loppuun, se liitetään aiemmin luotuun objektiin. Kun se on liitetty siihen, Unity tunnistaa skriptissä olevan Transform-tyyppisen muuttujan ja siihen pääsee tämän jälkeen käsiksi Unityn käyttöliittymästä. Transform-muuttujaan laitetaan numero, joka kertoo, montako objektia halutaan parallaksivierityksellä liikuttaa. Kun numero on syötetty, aukeaa siihen kentät joihin voi valita haluamansa taustan elementit.

Tämän jälkeen pitää vielä taustan eri elementit siirtää, joko kauemmaksi taustalle tai lähemmäksi eteen, riippuen siitä, onko se pelissä edessä vai takana. Tämä tehdään Unityssä poistumalla 2D-näkymästä ja sen jälkeen hiirellä siirtämällä elementit haluamilleen paikoille. (Kuva 5.)

Tässä skriptissä, jolla minä toteutin parallaksivierityksen, oli myös smoothing-muuttuja, johon pääsi myös käsiksi Unityn käyttöliittymästä. Sitä muuttamalla voitiin muuttaa parallaksivierityksen ”voimakkuutta”.

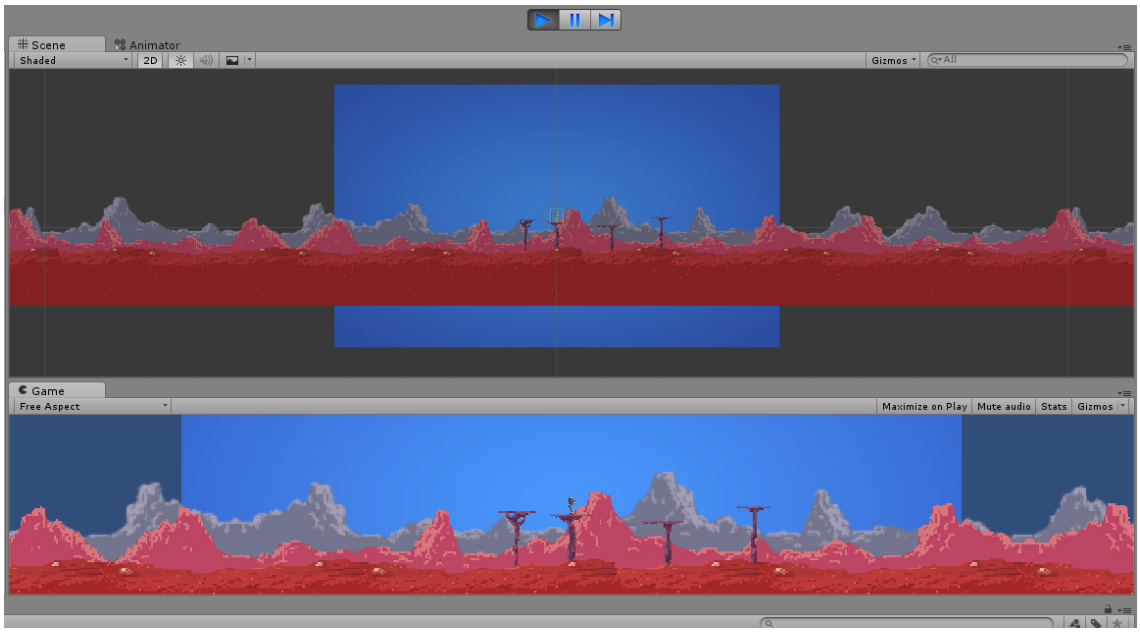


KUVA 5. Astuttu ulos 2D-näkymästä.

Ilman parallaksivieritystä peli saattaa tuntua tökeröltä ja tausta ei välttämättä tunnu olevan oikeasti kaukana hahmosta, ja toisaalta etualalla olevat asiat eivät tunnu olevan lähellä. Siksi se on melkoisen tärkeä ominaisuus kaikissa 2D-peleissä.

4.4 Taustan toistuvuus ja sen toteuttaminen

Taustan toistuvuus (engl. Tiling) tarkoittaa eräänlaista taustan looppaamista. Se ei ole sinänsä monimutkainen asia, mutta tämän kaltaisessa sidescroller – pelissä se on varsin tärkeä, sillä se on käytännössä melkeinpä pakollinen. Jos tausta olisi vain alussa näkyvä palikka, eikä looppautuisi pelaajan liikkuessa eteenpäin, olisi peli melkoisen ankea. Tässä tapauksessa se toteutetaan yksinkertaisesti lisäämällä kopio olemassa olevasta taustasta näkyvän taustan perään, jolloin luodaan illuusio vuoriston jatkumosta. Käytämme siis edellä esitettyjä vuoristokuvia, kuten kuvasta 6 näkyy.



KUVA 6. Taustan toistuvuus

Taustan toistuvuuden toteuttamiselle ei ole Unityssä olemassa omaa työkalua, joten se on toteutettava skriptin avulla, joko kirjoittamalla oma tai käyttäen olemassa olevia.

Tässä projektissa käytetyssä skriptissä toistuvuus toteutettiin siten, että kun kameran reuna on ylittämässä taustan reunan, tausta loopataan. Huomaa, että looppaus toteutetaan juuri ennen kuin kamera ylittää reunan, ei ylittämishetkellä. Tässä käytetään x-akselin offsettiä, eli käytännössä huijataan kameraa luulemaan, että se on ylittänyt reunan. Tämä pehmentää kokemusta ja vältytään välkkymiseltä ja kaikenlaisilta ongelmilta. Kuvassa 6 kamera on zoomattu ulos, jotta efekti näkyisi paremmin.

```
// Update is called once per frame
void Update () {
    if (hasALeftBuddy == false || hasARightBuddy == false) {

        float camHorizontalExtend = cam.orthographicSize * Screen.width/Screen.height;

        float edgeVisiblePositionRight = (myTransform.position.x + spriteWidth/2) - camHorizontalExtend;
        float edgeVisiblePositionLeft = (myTransform.position.x - spriteWidth/2) + camHorizontalExtend;

        if (cam.transform.position.x >= edgeVisiblePositionRight - offsetX && hasARightBuddy == false)
        {
            MakeNewBuddy (1);
            hasARightBuddy =  true;
        }
        else if (cam.transform.position.x <= edgeVisiblePositionLeft + offsetX && hasALeftBuddy == false)
        {
            MakeNewBuddy (-1);
            hasALeftBuddy = true;
        }
    }
}
```

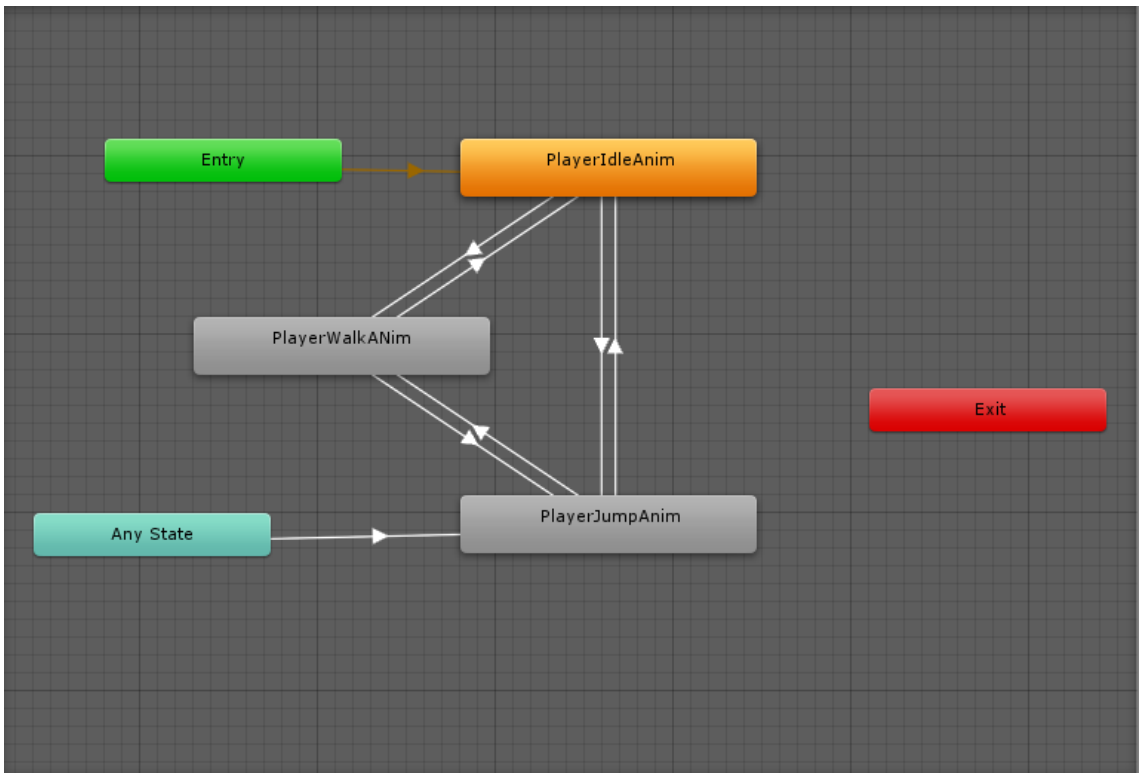
KUVA 7. Ote skriptistä Tiling.cs

Kuvassa 7 on ote skriptistä juurikin siitä osasta, jossa tarkistetaan, että onko kamera ylittänyt tietyn pisteen offsettiä hyväksi käyttäen. Koodissa esiintyvät ”Buddyt” ovat juurikin niitä luotuja ”looppeja”.

4.5 Hahmon animointi

Tässä työssä hahmon animointi toteutettiin yksinkertaisesti vaihtamalla kuvaa, kun jotain tapahtuu. Esimerkiksi, kun hahmo kävelee maassa eteenpäin, loopataan kuvasarjaa jossa jalat liikkuvat. Tämä toteutettiin Unityn oman Animaattorin avulla.

Unityn animaattorissa luotiin hahmolle eri tiloja, esimerkiksi juosta ja hypätä. Kun tilat oli luotu, liitettiin niihin niille kuuluvat animaatiot. Tämän jälkeen siirryttiin eräänlaiseen tilakaavionäkymään, jossa määrättiin siirtymät animaatiosta toiseen (Kuva 8).



KUVA 8. Hahmon animaation tilakaavio

Tilasta toiseen siirrytään aina, jos jokin tietty ehto täyttyy. Jos hahmo ei ole maassa, siirrytään hyppyanimaatioon, ja taas jos hahmo on liikkumatta, siirrytään paikallaanoloanimaatioon. Näitä tiloja hallitaan erilaisten boolean muuttujien avulla, jotka ovat arvoltaan joko true tai false ja niitä muutetaan skriptien avulla.

4.6 Kolikoiden kerääminen ja aikarajoitus

Joka pelissä pitää olla jotain muutakin tekemistä kuin ympäriinsä hyppely, tässä pelissä kerätään kolikoita ennen kuin aika loppuu. Kolikoiden kerääminen toteutetaan lisäämällä peliin uusi objekti, kolikko, jolle asetetaan collider. Sitä ei kuitenkaan lisätä suoraan peliin tässä tapauksessa, vaan luodaan kolikoille spawn pointteja, eli paikkoja, joihin kolikot pelin edetessä ilmestyvät (kuva 9).

KUVA 9. Kolikoiden spawn pointit.

Kolikot saadaan ilmestymään näihin ennalta määritettyihin paikkoihin yksinkertaisen skriptin avulla. Kyseinen skripti tarkastaa, että onko näytöllä yhtään kolikkoa, ja jos ei ole, niin se sijoittaa satunnaiseen paikkaan kolikon. Ote skriptistä kuvassa 10, jossa näkyy juuri se kohta, jossa kolikko asetetaan satunnaiseen paikkaan.

```
void Spawn(){

    spawning = true;

    Transform coinSpawns = coinSpawn[Random.Range (0, coinSpawn.Length)];
    GameObject coinPrefab = (GameObject)Instantiate (coins, coinSpawns.position, coinSpawns.rotation);
}
```

KUVA 10. Ote skriptistä CoinSpawner.cs

Kolikoille itsessään piti kirjoittaa oma skripti, jossa määriteltiin, että mitä tapahtuu, kun pelaaja osuu kolikkoon. Tässä tapauksessa kolikko tuhotaan ja lisätään yksi piste kolikkolaskuriin ja kelloon 3 sekuntia lisää. Samassa skriptissä myös tarkistetaan jatkuvasti, että ylittääkö kerättyjen kolikkojen määrä maksimiarvon. Jos niin käy, pelaaja voittaa pelin ja näytetään voittoteksti. Tätä ohjaava koodinpätkä näkyy kuvassa 11.

```
void OnTriggerEnter2D(Collider2D col){

    if (col.gameObject.tag == "Coin") {
        Destroy (col.gameObject);
        curCoins ++;
        coinSpawn.spawning = false;
        timeText.timer += 3;
    }
```

KUVA 11. Ote skriptistä Coins.cs

Pelaajan aloittaessa pelaamisen, käynnistyy kello, joka alkaa laskemaan alaspäin 25 sekunnista. Jos kello pääsee nollaan, peli pysähtyy ja näytölle ilmestyy häviämisestä ilmoittava viesti. Taasen, jos pelaaja saa kerättyä tarvittavan määrän, tällä hetkellä kymmenen, kolikkoa, ennen kuin kello menee nollaan, hän voittaa ja peli näyttää voittoviestin (kuva 12).

KUVA 12. Voittoteksti

Kaikki kolme eri tekstiä, eli Kolikoita-, Aika ja Voitto-tekstit ovat UI-text elementtejä. Ne ovat osa UI:ta eli käyttöliittymää, joten pelaaja ei ole niiden kanssa vuorovaikutuksessa ja ne pysyvät koko ajan samassa paikassa ruudun reunassa. Niihin pääsee käsiksi kaikista skripteistä kunhan muistaa ottaa käyttöön UI-elementtien muokkauksen lisäämällä skriptin alkuun ”using UnityEngine.UI”.

5 YHTEENVETO

Työ kokonaisuudessaan oli erittäin opettavainen. Jouduin opetella kaiken uutena asiana ja se oli paikoittain haastavaa, mutta kaikista vaikeuksista selvittiin. Unity itsessään osoittautui mukavaksi ympäristöksi ja herätti kiinnostuksen vastaavanlaisiin projekteihin jatkossakin. Työn tavoitteena oli oppia käyttämään Unityä ja opetella tekemään yksinkertainen 2D-peli. Tavoitteet täyttyivät hyvin.

Eniten haasteita tuotti pelin pelattavuuden saaminen mukavaksi. Tällä tarkoitan hahmon liikkeitä ja sitä, miten hahmo on vuorovaikutuksessa ympäristön kanssa. Mukavuuden saavuttamiseksi tarvittiin paljon pientä säätöä ja vielä enemmän kokeilua ja testailua.

Kaiken kaikkiaan työ ja projekti onnistuivat mielestäni hyvin ja jäi jälkeenpäin hyvä olo, kun oppi paljon uusia asioita ja sai hyvän pohjan tuleville peliprojekteille. Tätäkin peliä olisi suhteellisen helppo lähteä jatkokehittämään, esimerkiksi tehdä paremmat grafiikat, lisää ja parempia kenttiä ja niin edelleen.

6 LÄHTEET

1. Unity Technologies. 2015. http://unity3d.com/

2. Brackeys 2015. http://brackeys.com

3. Creativebloq, luettu 8.4.15, http://www.creativebloq.com/web-design/parallax-scrolling-1131762

4. Ward, J. 2008. What is a Game Engine? GameCareerGuide.com, luettu 15.4.2015. http://www.gamecareerguide.com/features/529/what_is_a_game_.php

5. Wikipedia, Pelimoottori. http://fi.wikipedia.org/wiki/Pelimoottori

LIITE 2

OULUN AMMATTIKORKEAKOULU

Joonas Kopsala

PUHELIMEN LIIKESENSORIT JA NIIDEN KÄYTTÖ UNITY-YMPÄRISTÖSSÄ

PUHELIMEN LIIKESENSORIT JA NIIDEN KÄYTTÖ UNITY-YMPÄRISTÖSSÄ

Unity-ohjelmointia liikesensoreita hyväksi käyttäen

Joonas Kopsala
Opinnäytetyö
Kevät 2017
Tietotekniikka
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on käydä läpi nykyaikaisten älypuhelinten liikesensorit ja esitellä, miten niitä voidaan hyödyntää Unity-ympäristössä osana 3D-peliä. Työn tilaajana on Oulussa sijaitseva FrozenVision Oy ja työn tarkoituksena on tutkiskella pääasiallisesti kahta sensoria, jotka ovat kulmanopeusanturi ja kiihtyvyyssensori. Työn pääpainona on sensorien teorian lisäksi niiden käyttö Unity-ympäristössä mobiilipelin kontrolleina. Työni on ajankohtainen siinä mielessä, että pelit kehittyvät jatkuvasti eteenpäin ja varsinkin mobiilipuolella on jo jonkin aikaa käytetty hyväksi puhelimesta saatavaa sensoridataa, jota voidaan hyödyntää joko pelin kontrolleina tai muuten pelin mekaniikkoihin liittyen.

Työni pohjana on ajatus siitä, minkälaista olisi toteuttaa Android-puhelimelle peli, joka käyttäisi kontrolleina liikesensoreita. Ajatuksena oli, että pelissä pelaaja on kuin tykin ampuja ja tähtäisi kääntelemällä puhelinta. Pääpainona pelin kontrolleissa on kulmanopeusanturi eli gyroskooppi. Työssä selvitetään, miten liikesensoridata saadaan Unity-ympäristöön puhelimesta sekä miten sitä käytetään ohjelmoinnissa.

Aluksi käyn läpi yleisesti kaksi tärkeintä sensoria liikkeen mittaamiseen Unity-ympäristössä. Käyn läpi teoriaa ensin kulmanopeusanturista ja tämän jälkeen kiihtyvyyssensorista. Esittelen esimerkkejä, miten kutakin sensoria voisi hyödyntää. Paneudun myös molempien sensorien toimintaan rautatasolla sekä selitän, miten ne toimivat. Tavoitteena on saada lukijalle selväksi, mihin tarkoitukseen kumpaakin sensoria käytetään ja miten se toimii.

Seuraavassa osassa vertaillaan näiden kahden toimintaperiaatteita ja rakenteita keskenään ja käsitellään kummankin vahvuudet ja heikkoudet. Tässä osassa käydään pintapuolisesti selvimmät eroavaisuudet läpi molemmista sensoreista, jotta saadaan selvästi eroteltua ne toisistaan. Tässä osassa otetaan kuvaesimerkit esille ja ulkonäön lisäksi erotellaan myös arvoja, joita ne palauttavat.

Neljännessä osassa katsotaan, miten näitä tekniikoita käytetään Unity-ympäristössä tutustuen Unityn omiin kirjastoihin. Tässä osassa katsotaan hieman, miten sensorit otetaan koodissa käyttöön ja miten arvoja saadaan irti. Tässä osassa otetaan esimerkkejä koodista, jota kirjoitin pelidemoa varten, ja käydään läpi, mitä olen koodilla hakenut ja miten toteuttanut eri ominaisuuksia. Pyrin tässä osassa avaamaan tarkemmin, mitä eri muuttujat Unityn liikesensorikirjastoissa tekevät ja mihin niitä yleensä käytetään.

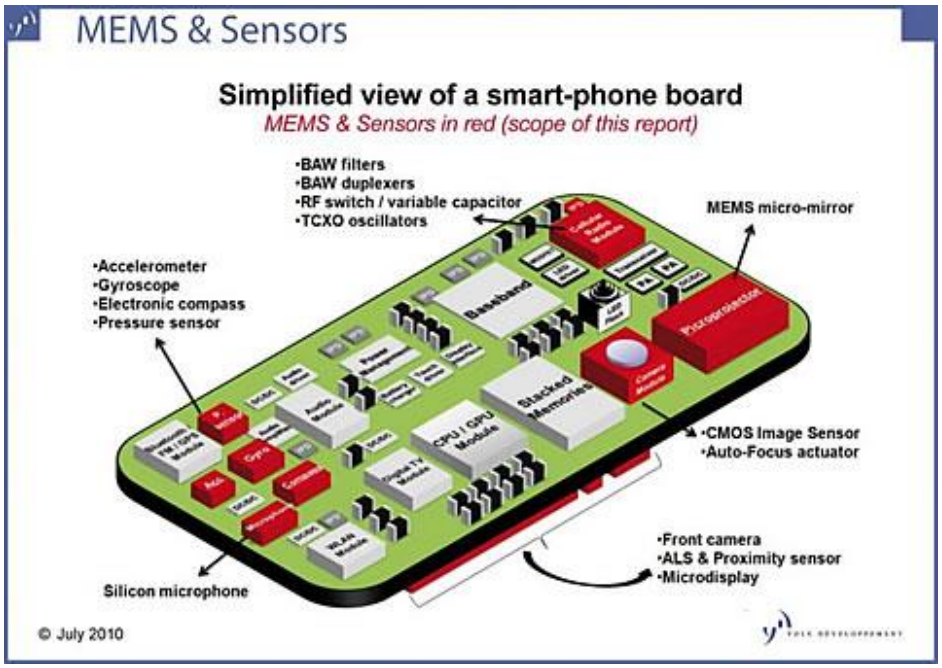
Viimeisessä osassa tutustutaan tarkemmin pelidemooni ja käydään läpi, mitä ratkaisuja tein demoa rakentaessa. Tässä osassa kerron muun muassa, miten olen käyttänyt dataa hyväkseni ja minkälaisiin ratkaisuihin olen päätynyt. Yritän avata lukijalle demon rakentamisen prosessia ja tuoda esille tekemiäni havaintoja liikesensoridatan käyttämisestä Unity-ympäristössä. Pääpainona tässä osassa on liikesensoridatan käyttö ja hyödyntäminen, mutta pyrin esittelemään myös, miten demon perusrakenne on toteutettu. Tämän osan lopussa esittelen ajatuksiani ja ideoitani siitä, minkälaisia muutoksia demoon tulisi tehdä jatkokehitystä varten.

2 LIIKESENSORIT

Nykyajan älypuhelimista löytyy liikkeentunnistukseen pääasiassa kaksi sensoria. Ne ovat kulmanopeusanturi eli gyroskooppi (engl. gyroscope) ja kiihtyvyyssensori (engl. accelerometer). Näiden lisäksi on myös tätä tarkoitusta sivuava kompassi. Puhelimissa liikesensorit ovat pakattuna pieniin komponentteihin ja samassa paketissa voidaan täten mitata moniakin eri asioita. (Kuva 1.)

Näitä sensoreita voidaan käyttää erilaisiin tarkoituksiin laitteen asennon tarkastelussa, kun halutaan esimerkiksi tietää, onko laite pysty- vai vaakatasossa. Tätä tietoa voidaan käyttää vaikkapa kamera-sovelluksessa, kun halutaan kääntää ruutu puhelimen asentoa vastaavaan tilaan. Muuta liikkeeseen liittyvää sensoridataa voidaan nykyajan älypuhelimissa käyttää esimerkiksi siihen, että tiedetään, onko puhelun aikana puhelin käyttäjän korvalla vai ei, jolloin joko näyttö on sammuksissa tai päällä.

Vanhemmissa älypuhelimissa oli pitkään pelkkä kiihtyvyyssensori, mutta nykyään lähes kaikissa uusissa älypuhelimissa on sekä gyroskooppi että kiihtyvyyssensori. Tietenkin nykyajan älypuhelimissa on paljon muitakin sensoreita, kuten esimerkiksi valon kirkkautta tunnistava sensori tai etäisyyttä mittaava sensori (1), mutta tässä työssä tarkastellaan tarkemmin vain liikesensoreita.

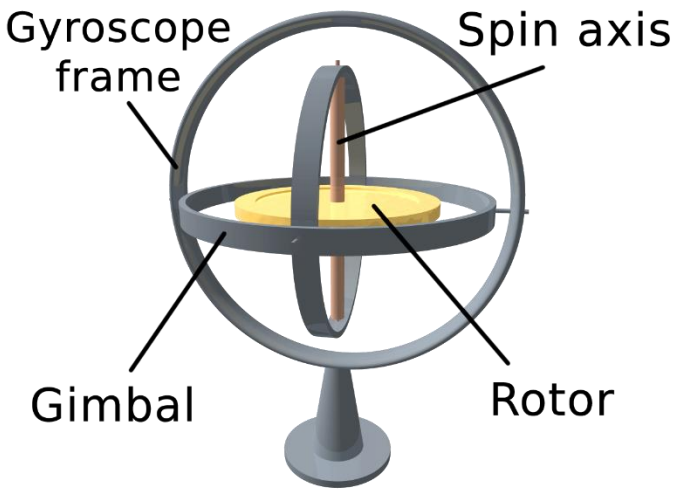


KUVA 1. Yksinkertaistettu näkymä älypuhelimen osista. Sensorit merkitty punaisella. (2.)

1.1 Kulmanopeusanturi eli gyroskooppi

Kulmanopeusanturi eli gyroskooppi (engl. gyroscope) on laite, joka mittaa liikettä tarkasti seuraten gyroskoopin toimintaperiaatetta. Gyroskooppi on suunniteltu mittaamaan liikettä ja se tekeekin sen todella tarkasti. Tämän takia se on käytössä esimerkiksi Virtual reality -tekniikkaa käyttävissä peleissä, koska siinä ympäristössä epätarkka liikkeen seuranta voi johtaa epämukavaan käyttökokemukseen tai vaikkapa pahoinvointiin. Gyroskooppia hyödynnetään useimmiten asennon mittaamisessa ja vakauttamisessa (3).

Perinteisen gyroskoopin rakenne on monista pyöristä koostuva (kuva 2), mutta nykyajan gyroskoopit ovat usein optisia. Esimerkkejä optisista gyroskoopeista ovat lasertekniikkaan perustuvat gyroskoopit tai kuituoptiset gyroskoopit. Luonnollisesti nykyajan älypuhelimissa olevat gyroskoopit eivät ole ulkonäöltään sellaisia kuin perinteinen gyroskooppi, vaikkakin toimintaperiaate on pohjimmiltaan sama. Nykyään älypuhelimissa on käytössä muun muassa MEMS-gyroskooppeja, jotka ovat pienikokoisia ja edullisia. (4.)



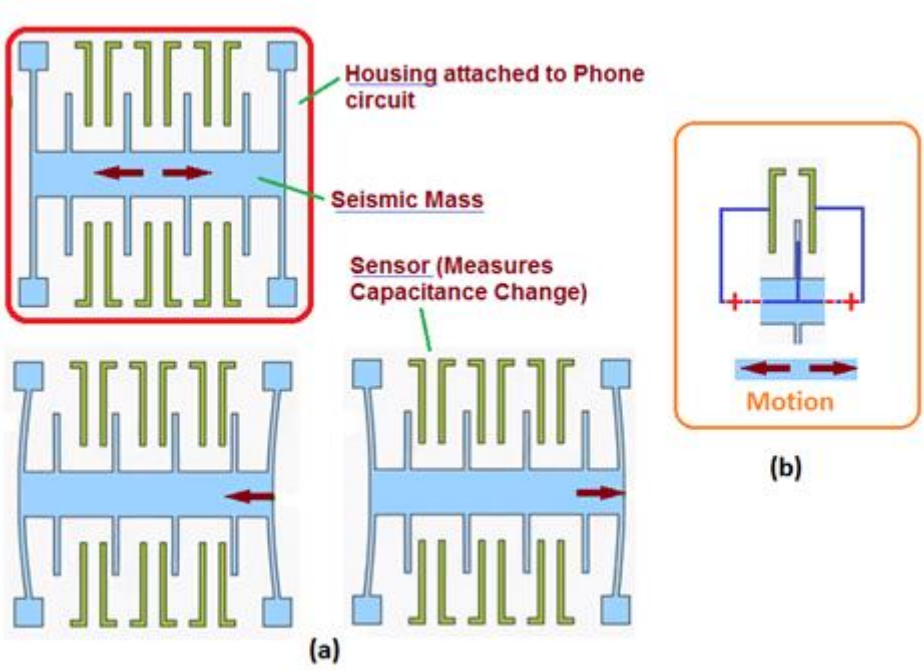
KUVA 2. Perinteinen gyroskooppi (3)

Gyroskooppeja on olemassa muutamia erilaisia, jotka ajavat samaa asiaa, mutta pohjautuvat eri tekniikoille ja täten sopivat eri tarkoituksiin. Näistä mainittakoon esimerkiksi lasermittauksilla toimiva gyroskooppi ja värähtelyn avulla toimiva gyroskooppi. Gyroskoopin vahvuuksiin lukeutuu sen kyky mitata pyörimisliikettä tietyn akselin ympärillä.

Gyroskooppeja käytetään arkielämässä muun muassa kannettavissa tietokoneissa putoamisen tunnistamiseen, jolloin sammutetaan kovalevyt hajoamisen estämiseksi, laivoissa kallistumisen tunnistamiseen ja vakauttamiseen sekä lentokoneissa myös asennon tarkasteluun.

1.2 Kiihtyvyyssensori

Kiihtyvyyssensori (engl. accelerometer) mittaa puhelimen asentoa itseiskiihtyvyyden avulla (5). Kiihtyvyyssensoria käytetään yksinkertaisemmillaan siihen, että tiedetään, pitääkö puhelimen näytön kääntyä pysty- tai vaaka-asentoon. Sensori toimii siten, että se laskee laitteen liikkeet painovoiman avulla ja ilmoittaa arvot x-, y- ja z-akseleiden mukaan (kuva 3). Käytännössä kiihtyvyyssensori mittaa puhelimen kallistumista eri akseleiden mukaan seuraamalla kulmaa, jossa puhelin on. Kiihtyvyyssensorin toiminnan kannalta onkin keskeistä sen akselipohjainen mittaustapa. (6.)



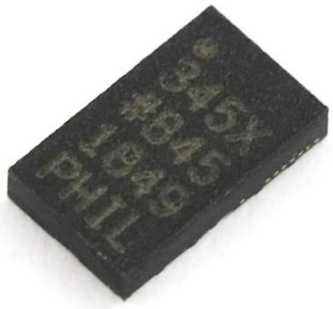
KUVA 3. Kiihtyvyyssensorin toimintamalli (3)

Kiihtyvyyssensori löytyy nykyään käytännössä jokaisesta älypuhelimesta, koska se on ollut markkinoilla jo verrattain kauan ja johtuen tästä se on myös halvempi valmistaa. (4.) Kiihtyvyyssensori löytyi aikanaan jo ensimmäisestä Applen iPhonesta, joka julkaistiin vuonna 2007, (7) kun taas kulmanopeusanturi löytyi vasta iPhone 4 -puhelimesta, joka julkaistiin 2010 (8).

3 VERTAILUA

Kiihtyvyyssensori ja gyroskooppi ovat molemmat tärkeitä osia nykyajan älypuhelimissa ja voi helposti tuntua, että ne ovat olemassa samaa tarkoitusta varten, mutta näin ei asia kuitenkaan ole. Kiihtyvyyssensori mittaa nimensä mukaan kiihtyvyyttä, mutta ei pyörimisliikettä. Unity-ympäristössä voi helposti erehtyä luulemaan, että kiihtyvyyssensori mittaa puhelimen pyörimisliikkeitä, sillä tietyt muutokset puhelimen asennossa vaikuttavat sen antamaan arvoon, jonka Unityn kirjasto palauttaa yhtenä vektoriarvona. Todellisuudessa kiihtyvyyssensori mittaa vain kiihtyvyyttä. Gyroskooppi taasen mittaa pyörimisliikettä, joten siitä saatu data kertoo, miten ja kuinka kovalla vauhdilla kyseinen laite pyörii milläkin hetkellä.

KUVA 4. Gyroskooppi (7)



KUVA 5. Kiihtyvyyssensori (7)

Ulkonäöllisesti laitetasolla gyroskooppi (kuva 4) ja kiihtyvyyssensori (kuva 5) näyttävät melko pitkälle samalta. Kummatkin ovat puhelimen piirilevyssä olevia komponentteja. Joskus ne voivat olla jopa samassa kokonaisuudessa, kuitenkin toimien itsenäisesti. Ohjelmistotasolla ne kuitenkin täydentävät toisiaan usein. Näin on myös Unityn kirjastoissa, joihin palaan myöhemmin.

4 KÄYTTÖ UNITY-YMPÄRISTÖSSÄ

Kun kehitetään Android-alustan peliä Unityllä, ovat gyroskooppi- ja kiihtyvyyssensori-data helposti saatavilla Unityn omien kirjastojen ansiosta. Molemmille sensoreille on omat kirjastonsa, joiden avulla saadaan kattava määrä dataa käyttöön. Kuvasta 6 näkyy, miten kiihtyvyyssensorin arvot voidaan hakea suoraan Input.acceleration-komennolla halutulta akselilta (x, y tai z).

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class AccelerometerInput : MonoBehaviour
5   {
6       void Update ()
7       {
8           transform.Translate(Input.acceleration.x, 0, -Input.acceleration.z);
9       }
10  }
```

KUVA 6. Kiihtyvyyssensorin arvojen hakeminen Unityssä C#-kielellä

Gyroskooppia käytettäessä pitää ensin ottaa gyroskooppi käyttöön Input.gyro.enabled = true; -komennolla (kuva 7). Tämä käyttöönottokomento on erittäin tärkeä, sillä minkäänlaista dataa ei saada irti, jos tätä ei ole tehty. Tämän jälkeen gyroskoopin antamat arvot ovat saatavilla Input.gyro.attitude-komennon kautta. Näitä arvoja voidaan sitten käyttää haluttuun tarkoitukseen, esimerkiksi kameran liikutteluun.

```
void Start()
{
    Input.compensateSensors = true;
    Input.gyro.enabled = true;
}

void FixedUpdate()
{
    TorniObjecti.transform.Rotate(0, 0, -Input.gyro.rotationRateUnbiased.y*3);
    PiippuObjecti.transform.Rotate(-Input.gyro.rotationRateUnbiased.x/2, 0, 0);
    Kamera.transform.Rotate(-Input.gyro.rotationRateUnbiased.x/2.5f, 0, 0);
}
```

KUVA 7. Gyroskoopin käyttöönotto ja objektien liikuttelua gyroskoopin sensoridatalla

Jos laitetta käytetään vaaka-asennossa, voi olla tarpeellista käyttää Input.compensateSensors-komentoa (kuva 7). Tällä komennolla sensorit ottavat huomioon laitteen asennon ja täten data käsitellänä tämä huomioon ottaen. Gyroskoopin Unityn sisäisen kirjaston vaihtoehdoista Gyroscope.rotationRate on ainut joka käyttää pelkkää gyroskooppidataa. Kaikki muut käyttävät muita sensoreja tukena, kunhan ne ovat saatavilla. Täten saavutetaan parempi mittaustulos oikean maailman liikkeistä.

Tarkastellessa tarkemmin Unityn kirjastoja kiihtyvyyssensoriin ja gyroskooppiin liittyen, voi huomata, että kiihtyvyyssensorin kirjasto on erittäin suppea. Se sisältää pelkästään Input.acceleration-komennon (10), kun taas gyroskoopin kirjastosta löytyy montakin eri muuttujaa, joista saadaan dataa irti (11). Tämä johtunee sensorin luonteesta, eli kiihtyvyyssensorilla on olemassa vain yksi muuttuja, joka on kiihtyvyys. Gyroskoopin tapauksessa muuttujien määrä selittyy sillä, että gyroskoopin tarjoamasta datasta on mahdollista saada paljon tietoa irti (kuva 8).

Variables

attitude	Returns the attitude (ie, orientation in space) of the device.
enabled	Sets or retrieves the enabled status of this gyroscope.
gravity	Returns the gravity acceleration vector expressed in the device's reference frame.
rotationRate	Returns rotation rate as measured by the device's gyroscope.
rotationRateUnbiased	Returns unbiased rotation rate as measured by the device's gyroscope.
updateInterval	Sets or retrieves gyroscope interval in seconds.
userAcceleration	Returns the acceleration that the user is giving to the device.

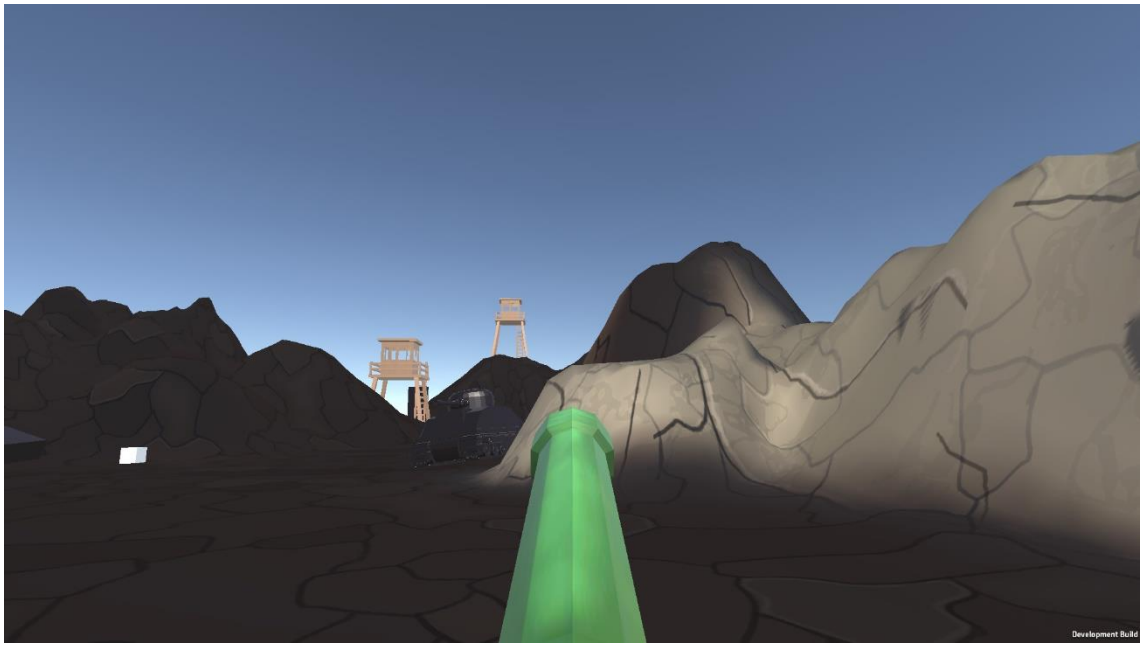
KUVA 8. Gyroskoopin kirjaston kaikki muuttujat (11)

Kaiken kaikkiaan sensorien käyttöönotto ja käyttäminen on Unityn puolelta tehty melkoisen helpoksi, kiitos valmiiden kirjastojen. Tavallisen käyttäjän ei tarvitse ymmärtää välttämättä mitään sensoreista, vaikka käyttäisikin niitä omassa projektissaan. Edistyneemmän käyttäjän tulee kuitenkin jo hieman ymmärtää, mikä on sensorien toimintaperiaate ja mitä mihinkin tulee käyttää.

5 PELIDEMO

Rakensin tätä työtä varten pienen näytepelin Unity-ympäristössä. Pelin kehityksen aikana pyrin testaamaan ja tutkimaan tekniikoita liittyen liikesensoridataan ja sen käyttöön peliympäristössä. Projektia aloittaessa minulla ei ollut entuudestaan kokemusta liikesensorien käyttämisestä Unity-ympäristössä, joten tämän työn tutkimusosa tuli siinä mielessä tarpeeseen. Aloitin projektin kehittämisen tyhjästä Unity 3d-projektista.

Tavoitteena oli rakentaa peli, jonka perustoiminnallisuutena olisi kameran liikuttaminen puhelinta kääntämällä ikään kuin olisi itse tykin ohjissa. Tarkoituksena oli saada peli tuntumaan siltä, että pelaaja olisi itse tykin juuressa tähtäämässä. Kuvassa 9 on näkyvillä pelaajan näkymä, kun peli alkaa. Peliä rakentaessa pääpaino oli kontrolleissa ja siinä, miten kameraa ja tykkiä liikutetaan liikesensoridataa hyväksikäyttäen. Toki lisäsin peliin pelillisiä elementtejä, kuten ampumismahdollisuuden ruutua napauttamalla, rikkoutuvia torneja ja vihollistankkeja sekä mäkiä pelikarttaan.



KUVA 9. Näkymä pelidemossa

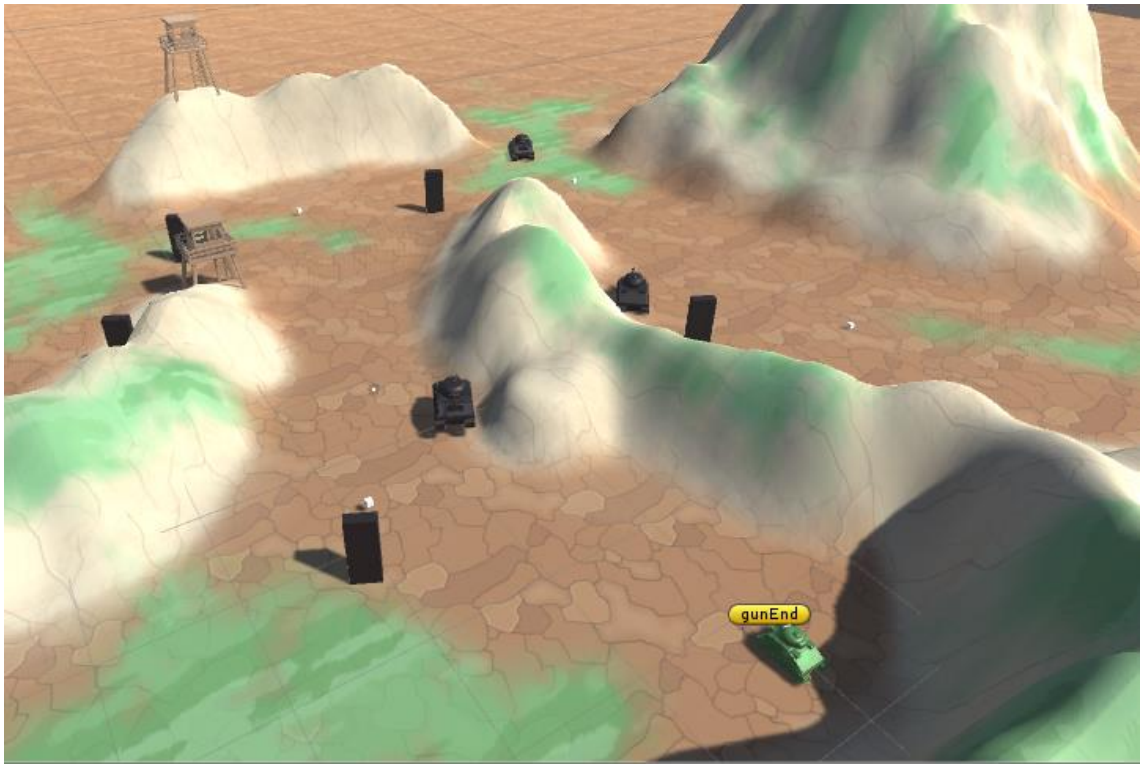
5.1 Liikesensoridatan käsittely projektissa

Tutkimisen ja asioihin perehtymisen jälkeen päädyin gyroskoopin käyttämiseen projektissa. Tietysti, kuten aiemmin mainittiin, Unityn gyroskooppikirjastot käyttävät tukenaan muidenkin sensorien dataa, joten muidenkin sensorien dataa on mukana. Näin laajuudeltaan pienessä demossa kameran ja panssarivaunun kontrolloimiseen käytetty koodi jäi pituudeltaan melko lyhyeksi. Jos peliä olisi kehitetty vielä eteenpäin, olisi kaikki ohjelmointikin monimutkaistunut uusien tarpeiden ilmestyessä. Kuvasta 7 voi nähdä, että liikutan panssarivaunun eri osia eri riveillä koodia. Tämä on tehty sen takia, jotta säilytettäisiin illuusio siitä, että ollaan todella panssarivaunun ohjaimissa. Oikeassa panssarivaunussa torni liikkuu eri nivelellä kuin tykki (kuva 10), joten jos näitä molempia olisi pelissä liikutettu samalla rivillä, illuusio oikean panssarivaunun liikkeistä ei olisi säilynyt. Kuvasta 7 näkee myös, että olen jakanut ja kertonut gyroskoopin arvoja eri luvuilla. Näillä saavutettiin se, että tietty liikemäärä oikeassa maailmassa kääntyy sopivaksi liikkeeksi pelin sisään.

KUVA 10. Pelissä olevan panssarivaunun malli ja tykin nivelet

5.2 Demon rakentaminen

Lähdin rakentamaan demoa tyhjästä Unity 3D -projektista, johon lisäsin terrain-kentän eli muokattavan maastoalustan. Kenttään nostelin terrain-työkalulla mäkiä ja vuoria, jotta pelinäkymä olisi miellyttävämpi. Alkuajatuksena oli tehdä peli, jossa tankki liikkuisi ikään kuin raidetta pitkin itsestään tasaisella vauhdilla, jolloin pelaaja voisi keskittyä ampumiseen ja tähtäämiseen liikkumisen sijaan. Testailin aluksi eri metodeja tykin ja kameran liikutteluun ja päädyin melko nopeasti gyroskoopin käyttöön. Tämä tuntuu jälkeenpäin itsestäänselvyydeltä, mutta aluksi asiasta tietämättömänä se ei näin ollut. Lisäsin demoon pelattavuuden takia yksinkertaisen ampumistekniikan, joka perustuu raycast-tekniikalle. Tämä tarkoittaa sitä, että tykin päästä luodaan tietyn mittainen, tässä tapauksessa keltainen, säde, joka osuessaan johonkin luo ampumisefektin. Lisäsin myös ammuttavia palikoita, tankkeja ja hajoavia torneja, joilla testasin pelattavuutta kehitysvaiheessa.



KUVA 11. Pelikentän näkymä ylhäältäpäin

Pyrin pitämään pelin kontrollit mahdollisimman yksinkertaisina, sillä mielestäni se on tärkeä ominaisuus mobiilipeleissä jo ihan käyttömukavuuden takia. Tällä hetkellä pelissä ainoat kontrollit ovat puhelimen kääntely, jolla tähdätään, sekä näytön napautus sormella, jolla ammutaan.

5.3 Demosta paremmaksi peliksi

Tässä luvussa esittelen ajatuksiani siitä, minkälaisia muutoksia, parannuksia ja uusia ominaisuuksia pelidemoon tulisi tehdä, jotta päästäisiin lähemmäksi valmista peliä. Pelin kehitys on pitkä prosessi ja yleensä asiat muuttuvat paljonkin, kun projekti etenee. Rakentamani demo on tällä hetkellä rakenteeltaan melko pelkistetty, siinä ei ole paljonkaan pelillisiä ominaisuuksia eikä olemassa olevia ominaisuuksia ole hiottu loppuun asti. Toisaalta demon onkin tarkoitus olla vain näyte.

Mielestäni raidepohjainen liikkuminen on ajatuksena toimiva. Kun panssarivaunu liikkuu itsestään, pelaajalle jää enemmän aikaa keskittyä tähtäämiseen ja ampumiseen ja täten pelikokemus on mukavampi. Tämän voisi siis pitää samana hieman paranneltuna. Ajatuksena kuitenkin jonkinlainen nopeuden säätö voisi olla mielenkiintoinen.

Ampumisen voisi muuttaa raycast-tekniikasta jonkinlaiseen kunnolliseen tekniikkaan, jossa pelissä tykin suusta lentäisi oikea panos. Tämä olisi tuntumalta paljon parempi, vaikkakin paljon vaativampi toteuttaa. Ammuksen osumasta tulisi myös seurata jonkinlainen räjähdys esimerkiksi partikkelityökalua käyttäen.

Viholliset eivät tällä hetkellä ole uhka pelaajalle. Pelaajalla tulisi olla syy pelätä vihollisvaunuja, joten mielestäni vihollisten tulisi ampua takaisin esimerkiksi tietyn ajan kuluttua pelaajan saapumisesta niiden ampumaetäisyydelle. Viholliset voisivat esimerkiksi ilmestyä mäkien takaa tai vaikkapa kurkistaa tornista. Tämä edellyttäisi erilaisten vihollistyyppien luomista peliin.

Kameran ja tykin kontrolleja tulee hioa ja jatkokehittää eteenpäin, sillä ne ovat suurimmassa osassa tässä pelissä. Jos kontrollit tuntuvat epämukavilta, koko pelikokemus muuttuu huonoksi. Mielestäni tässä tulee löytää sopiva kompromissi oikean elämän liikkeiden ja pelin sisäisten liikkeiden välillä. Tulee muistaa, että puhelinta pidetään käsissä pelin aikana eikä päässä, kuten esimerkiksi Virtual Reality -laseja pidettäisiin.

Tässä tuli esille mielestäni monta hyvää ideaa ja sain esiteltyä hyvin tärkeimmät asiat pelin jatkokehityksen kannalta. Mielestäni tällaisella pelillä on hyvä potentiaali tulla mukavaksi ja kiinnostavaksi peliksi.

6 YHTEENVETO

Tämän opinnäytetyön tarkoituksena oli tutustua liikesensoritekniikkaan Unity-ympäristössä ja tutkia, miten kyseistä tekniikkaa voitaisiin käyttää osana mobiilipeliä. Osana opinnäytetyötä toteutettiin pieni pelidemo, jonka tarkoituksena oli saavuttaa käytännön ymmärrys liikesensoritekniikasta osana Unityllä kehitettyä mobiilipeliä. Demo voisi myös toimia pohjana mahdolliselle jatkokehitykselle. Työn tilaajana toimi FrozenVision Oy, joka on oululainen startup-yritys, joka on viime aikoina kehittänyt pääasiassa mobiilipelejä.

Työn aikana saavutin melko kattavan ymmärryksen gyroskoopista ja kiihtyvyyssensorista yleisellä tasolla, sekä Unity-ympäristössä. Opin myös paljon uusia asioita demoa kehittäessä. Nämä asiat huomioon ottaen voidaan siis sanoa, että työn aikana päästiin tavoitteisiin. Työ oli myös mielenkiintoinen ja demon parissa oli kiinnostava työskennellä. Koen, että tästä työstä on minulle hyötyä tulevaisuutta ajatellen.

Liikesensoritekniikat ovat pelimaailmassa jatkuvasti yleistyvä ja entistä ajankohtaisempi asia. Jo vuosien ajan eri konsolivalmistajat ovat panostaneet liikesensoridataa hyödyntäviin ohjaimiin ja laitteistoihin. Esimerkkejä näistä ovat Oculus Rift Virtual Reality -laitteisto ja Playstationin käsissä pidettävät liikeohjaimet. Älypuhelinmaailmassa liikesensorit ovat olleet arkipäivää jo ensimmäisistä älypuhelimista lähtien jo peruskäyttäjänkin laitteissa. Luulen, että tulevaisuudessa nämä tekniikat kehittyvät vielä lisää eteenpäin ja pääsemme näkemään uusia innovaatioita liikesensoritekniikassa.

LÄHTEET

1. Android sensors overview. 2017. Android.com. Saatavissa: https://developer.android.com/guide/topics/sensors/sensors_overview.html. Hakupäivä 22.8.2017.

2. Don Scansen. 2013. MEMS & Sensors picture. Engineering. Saatavissa: http://www.engineering.com/ElectronicsDesign/ElectronicsDesignArticles/ArticleID/6124/How-MEMS-Enable-Smartphone-Features.aspx. Hakupäivä 6.9.2017.

3. Gyroskooppi. 2017. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Gyroskooppi. Hakupäivä 6.9.2017.

4. Jjules. Unity, and the Accelerometer vs. the Gyroscope: A Complete Guide. 2017. Saatavissa: https://forum.unity3d.com/threads/unity-and-the-accelerometer-vs-the-gyroscope-a-complete-guide.451496/. Hakupäivä 22.8.2017.

5. Gujarati, Paresh 2013. What is Accelerometer and how does it work on smartphones. Techulator. Saatavissa: http://www.techulator.com/resources/8930-How-does-smart-phone-accelerometer-work.aspx. Hakupäivä 22.8.2017.

6. Goodrich, Ryan 2013. Accelerometers: What They Are & How They Work. Livescience. Saatavissa: https://www.livescience.com/40102-accelerometers.html. Hakupäivä 22.8.2017.

7. Apple Iphone. 2017. Gsmarena. Saatavissa: http://www.gsmarena.com/apple_iphone-1827.php. Hakupäivä 22.8.2017.

8. Apple Iphone 4. 2017. Gsmarena. Saatavissa: http://www.gsmarena.com/apple_iphone_4-3275.php. Hakupäivä 22.8.2017.

9. Accelerometer and Gyroscope pictures. 2017. Sparkfun. Saatavissa: http://www.sparkfun.com/. Hakupäivä 22.8.2017.

10. Accelerometer library in Unity. 2017. Unity Technologies. Saatavissa: https://docs.unity3d.com/ScriptReference/Input-acceleration.html. Hakupäivä 22.8.2017.

11. Gyroscope library in Unity. 2017. Unity Technologies. Saatavissa: https://docs.unity3d.com/ScriptReference/Gyroscope.html. Hakupäivä 22.8.2017.