OULUN AMMATTIKORKEAKOULU

Mikael Havia

Mobiilipelin kehitys

Valmiiden koodikatkelmien hyödyntäminen mobiilipelin kehitystyössä

Mobiilipelin kehitys

Valmiiden koodikatkelmien hyödyntäminen mobiilipelin kehitystyössä

Mikael Havia
Opinnäytetyö
Kevät 2018
Tietojenkäsittelyn koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Internet-palvelut ja digitaalinen media

Tekijä(t): Mikael Havia
Opinnäytetyön nimi: Mobiilipelin kehitys: Ohjelmointi osuuden minimointi
Työn ohjaaja: Matti Viitala
Työn valmistumislukukausi ja –vuosi: Kevät 2018 Sivumäärä: 29 + 0

Opinnäytetyö on tehty oman peli-idean suunnittelemisesta ja prototyypin toteuttamisesta minimaalisella ohjelmoinnilla. Toimeksiantajaa ei opinnäytetyössä tämän takia ole. Tavoitteena oli saada peli-ideasta toimiva prototyyppi, jota olisi helppo työstää eteenpäin tulevaisuudessa. Peliprojektia lähdettiin toteuttamaan mahdollisimman pienellä ohjelmointityöllä ja kopioimalla jo valmiiksi olemassa olevia skriptejä. Valmiita skriptejä ja koodikatkelmia löytyy internetistä lukemattomia määriä ja olen aina miettinyt, kuinka pitkälle ja monimutkaisia projekteja voisi toteuttaa niitä käyttämällä.

Pelimoottorina käytettiin Unity-pelimoottoria ja ohjelmointi toteutettiin C# -ohjelmointikielellä. Unity 3D-pelimoottori valittiin oman kokemuksen takia ja ohjelmointikieli sen yleisyyden takia.

Peli-idean prototyyppi saatiin toteutettua tavoitteiden mukaisesti toimivaksi. Prototyypin rakenne saatiin rakennettua helposti kopioitavaksi, jonka takia lisäsisällön tekeminen onnistuu tulevaisuudessa helposti.

Asiasanat: Mobiilipeli, Peli, Ohjelmointi, Android

ABSTRACT

Oulu University of Applied Sciences
Information technology, Internet services and digital media

Author(s): Mikael Havia
Title of thesis: Mobile game development: Minimize the programming part
Supervisor(s): Matti Viitala
Term and year when the thesis was submitted: Spring 2018 Number of pages: 29 + 0

This thesis is about developing a mobile game idea with minimizing the programming part. The goal was to make a prototype of that game idea that would be easy to work on for the future. The game project was started to implement as little programming as possible and by copying already existing scripts. I have always wondered how far you can develop games by using existing codes.

Game engine used in this projects was Unity 3D. Programming part was done in C# programming language. Unity 3D-game engine was selected because the familiarity and personal experience. C# programming language is very much used and only language that we have knowledge.

As planned, good and playable prototype was implemented. The prototype structure was designed for easily to copy and make additional content for the future.

Keywords: Mobile Game, Game, programming, Android

SISÄLLYS

1 JOHDANTO 7
2 UNITY 3D 8
3 UNITY ANDROID-YHTEENSOPIVAKSI 9
3.1 Unity remote 10
3.2 Unity remote toiminta 11
4 KONSEPTI 13
4.1 Muutama idea 13
4.2 Pelin ideointi 13
4.3 2D vai 3D 14
4.4 Android vai iOS 15
5 PELI 16
5.1 Toteutus 17
5.2 Kontrollit 17
5.3 Joystick, Painike 19
5.4 Kenttäsuunnittelu 21
6 UI 23
6.1 Päävalikko 23
6.2 Kuolema ja maali Valikko 24
6.3 Ajanotto 25
7 POHDINTA 27
8 LÄHTEET 28

TERMISTÖ

Scene

Scene sisältää peliin lisätyt esineet ja asiat. Jokainen valikko ja kenttä voi olla omalla erillisellä scenellä.

Play-tila

Play-tila on Unity-editorissa käytettävä tila, joka mahdollistaa pelin toimivuuden kokeilemisen.

Skripti

Skripti on koodia sisältävä tiedosto.

Assetti

Assetti (eng. asset) on yleinen termi, jolla tarkoitetaan tiedostoja. Esimerkiksi kuva, 3d-malli tai ääniraita voi olla assetti.

Prefab

Prefab on kopio jostakin peliobjektista ja joka voi sisältää useamman peliobjektin. Prefabia muokkaamalla kaikki samanlaiset prefabit päivittyvät.

Peliobjekti

Peliobjekti on pelissä oleva asia, joka käsittää muokattavaa tietoa. Peliobjekti voi olla, esimerkiksi hahmo, kamera, valo tai efekti.

Sprite

Sprite on yksi tietty graafinen kokonaisuus. Esimerkiksi pelaaja hahmo voi olla yksi sprite tai jokin muu pelissä näkyvä objekti.

Collider

Collider on näkymätön ja tiettyä muotoa rajaava alue. Colliderille voi määrittää erilaisia toimintoja esimerkiksi törmättävyysominaisuuksia.

1 JOHDANTO

Opinnäytetyössä aikomuksena oli kehittää ensimmäinen oma mobiilialustoille tarkoitettu peli. Oman mobiilipelin kehitys oli ollut mielessäni jo pidemmän aikaa, ja nyt aukesi mahdollisuus toteuttaa idea. Peli ideaa ei ollut ja se osoittautui haasteelliseksi, mutta lopulta idea pelille löytyi.

Oma taitotaso pelinkehittäjänä on vielä alhainen, joten projektista ei tullut teknisesti kovin syvällinen. Suurin tavoite oli saada kehitettyä peliprojekti ilman itse kirjoitettua koodia. Pyrkimys oli oppia ymmärtämään koodia ja sen käyttöä. Yritin löytää kaikki tarvitsemani koodit valmiina internetistä.

Projektini toinen tavoite oli selittää ja osoittaa projektin eri vaiheet mahdollisimman yksinkertaisesti, jotta tätä opinnäytetyötä voisi pitää jonkinlaisen ohjeena tehdessään ensimmäistä mobiilipeliä. Kerron projektin suunnittelusta aina viimeistelyyn saakka. Tuon esille kohtaamiani ongelmia ja ratkaisuja.

Tutustuin Unity-pelimoottoriin, jota käyttämällä mobiilipelini myös kehitin. Tutustuin tässä opinnäytetyössä myös Android- ja iOS-käyttöjärjestelmiin tarkemmin aloittelevan mobiilipelien kehittäjän silmin. Tutustuin myös Unityn kehittämään Unity Remote 5 -sovellukseen, joka helpotti projektin etenemistä huomattavasti.

2 UNITY 3D

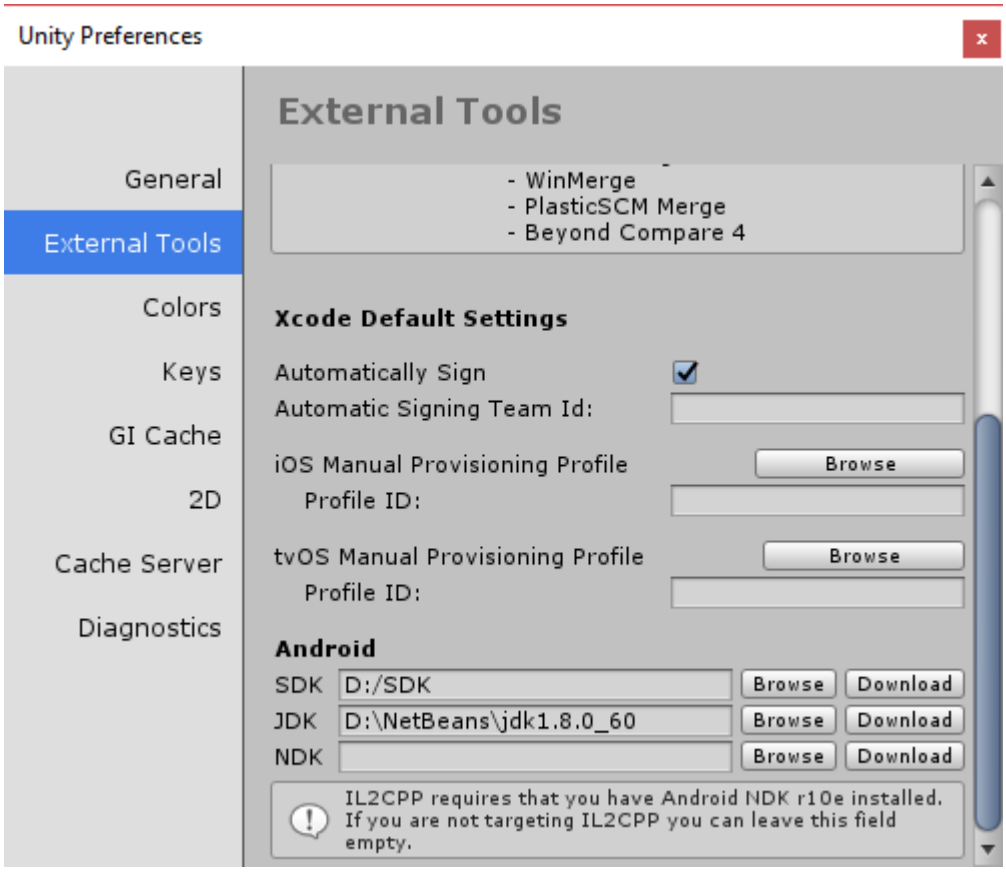
Unity on erittäin suosittu pelimoottori, jonka on luonut Unity Technologies -niminen yritys. Unitya käyttävät niin ammattilaiset kuin aloittelijatkin. Aloittelijan on helppo lähestyä Unitya, sillä se on käyttäjäystävällinen ja aloittelijakin voi saada aikaiseksi melko kohtalaisen pelin, jopa ilman minkäänlaista koodin kirjoitus osaamista.

Mikä Unity sitten on? Moni pelaaja, varsinkin indie-pelien pelaajat, ovat varmasti törmänneet Unity-pelimoottoriin, sillä suurin osa pienemmistä pelitaloista käyttää Unitya pelien kehitystyökaluna. Unity on monille alustoille toimiva 3D-pelimoottori, joka on suunniteltu olemaan käyttäjäystävällinen aloittelijoille, mutta silti tarpeeksi tehokas myös ammattilaisten käyttöön. Unity on pelimoottori, joka käsittää lait ja käyttäytymiset eri peliobjekteissa. Se myös käsittelee grafiikkaa, valaistusta ja fysiikkaa, kuten painovoimaa ja vauhtia.

3D-pelimoottoreita on useita erilaisia, kuten Unreal Engine 4, mutta Unity on yksi yksinkertaisimmista käyttää unohtamatta hyvin suunniteltua käyttöliittymää. Unityssa etuna on myös se, että pelejä voi tehdä useammalle eri alustalle, kuten iOS, Xbox One, PS4, Android, MacOS ja Windowsille. Myös virtuaalitodellisuuspelit ovat mahdollisia. (Sinicki 2016, viitattu 02.02.2018.)

3 UNITY ANDROID-YHTEENSOPIVAKSI

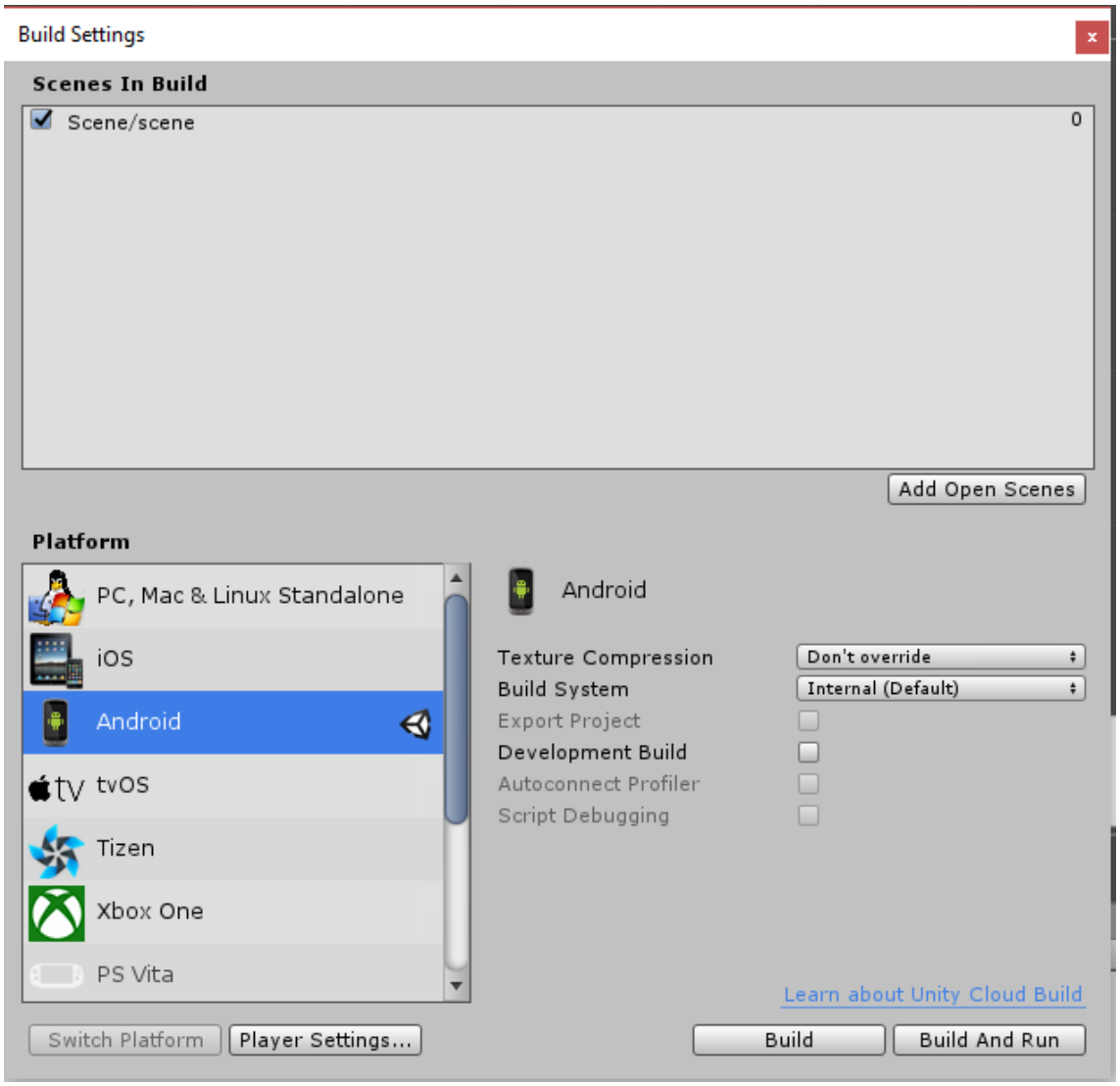
Unitylla voi tehdä pelejä suoraan Androidille ilman suurempi muutoksia. Joitakin muutoksia ja lisäyksiä täytyy kuitenkin tehdä, ennen kuin pelit toimisivat suoraan Android-pohjaisilla laitteilla. Unity tarvitsee Android Software Development Kit (SDK) -kehitystyökalun. Android SDK:n voi ladata Adnroidin omilta internet-sivuilta. Todettiin helpommaksi ladata koko Android-studio-ohjelmisto, jonka mukana tulee myös SDK-kehitystyökalut. Tarvittiin myös Java Developement Kit (JDK) -ohjelmistokehityspaketti, jonka lisääminen onnistuu samalla tavalla kuin SDK:n lisääminen. Unitylle pitää osoittaa tietokoneesta oikea kansio, mihin SDK on tallennettu. Oikea paikka löytyy Edit > Preferences > External tools -ikkunasta. Kuviossa 1 on esitettynä kehitystyökalujen lisääminen.



KUVIO 1. SDK ja JDK lisääminen Unity-pelimoottoriin

Unitylle pitää kertoa ennen projektin aloittamista, että mille alustalle projektia on tekemässä. Tämän asetuksen muuttaminen onnistuu File > Build Settings -kohdasta. Ensin lisätään käytettävät scenet Scenes In Build -kohtaan ja sen jälkeen valitaan Platform -kohdasta tarvittava alusta, mikä tässä

tapauksessa on Android. Kuviossa 2 on esitettynä alustan valinta. (Charger Games 2017, viitattu 07.03.2018.)



KUVIO 2. Build Settings

Edellä mainittujen muutosten jälkeen sovellusten ja pelien kehittäminen Android-pohjaisille laitteille onnistuu.

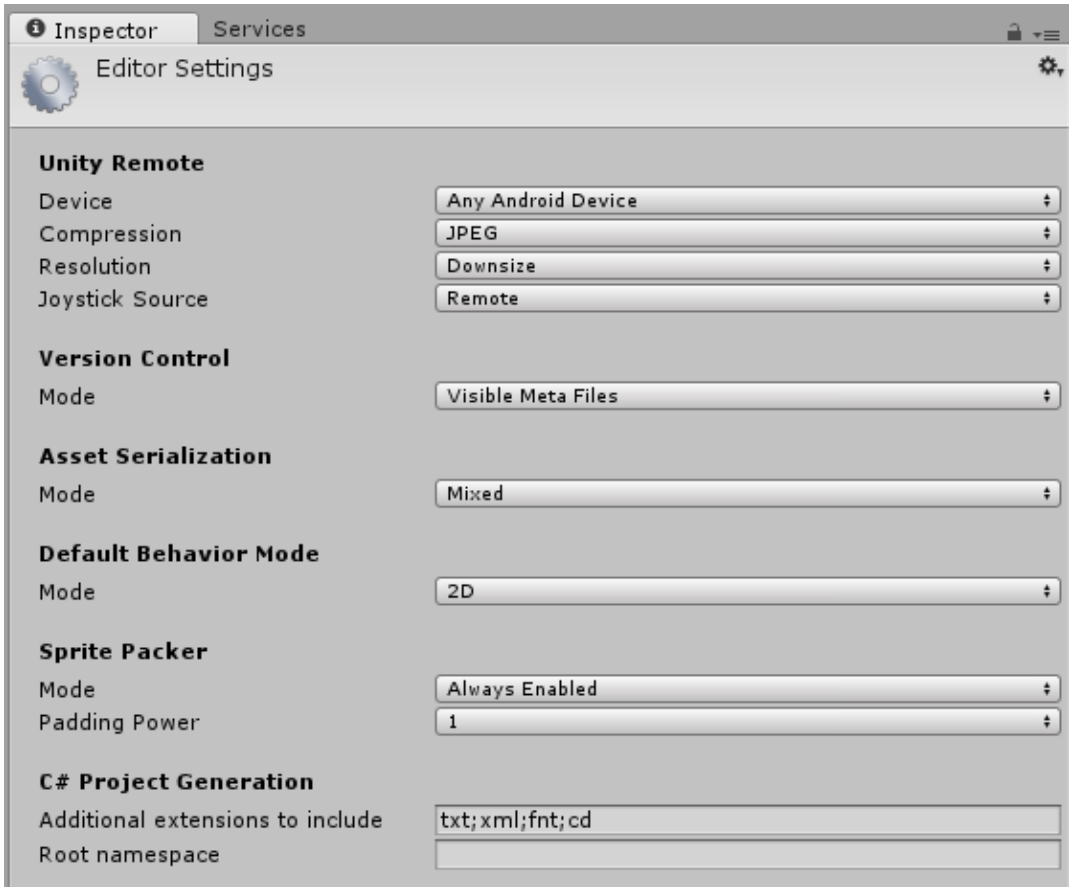
3.1 Unity remote

Projektin alkuvaiheissa mietittiin mahdollisuutta saada Android-laite näyttämään Unityn play-tila suoraan mobiililaitteen ruutuun. Unity play-tilassa voi kokeilla oman projektin toimivuutta ilman projektin asentamista tietokoneelle erikseen. Projektia helpottaisi suuresti, jos play-tilan voisi näyttää

suoraan mobiililaitteelle. Projektin eteneminen olisi ollut hidasta ja vaivalloista, jos olisi joutunut aina pienenkin muutoksen jälkeen asentamaan keskeneräisen projektin koneelle ja siirtämään sen mobiililaitteelle. Unity on kehittänyt sovelluksen nimeltä Unity Remote 5. Unity Remote 5 mahdollistaa play-tilan käytön mobiililaitteella reaaliajassa.

3.2 Unity remote toiminta

Ensimmäisenä täytyy ladata Unity Remote 5 -sovellus sovelluskaupasta, joka on ilmainen. Unityssa pitää muuttaa myös asetus, että remote-sovellus ja Unity osaisivat kommunikoida keskenään. Asetus löytyy kohdasta Edit > Project Settings > Editor (katso kuvio 3). Uuden ikkunan avauduttua löytyy kohta Unity Remote, jonka alapuolelta löytyy kohta Device. Device-kohtaan täytyy vaihtaa valintaikkunasta Any Android Device. Unity ja Unity Remote 5 toimivat nyt keskenään, mutta Android-laitteeseen täytyy vielä tehdä muutos. (Kelly 2016, viitattu 14.03.2018.)



KUVIO 3. Unity Editor Settings

Ennen Play-tilan käyttämistä Android-laitteilla, täytyy Android-laite saada kehittäjä-tilaan. Mobiililaitteen saa kehittäjä-tilaan menemällä Asetukset > Järjestelmä > Tietoja puhelimesta > Ohjelmistoversionnumero. Painamalla useamman kerran Ohjelmistoversionumero kohdasta tulee ilmoitus ”olet nyt kehittäjä”. Tämän jälkeen asetuksiin tulee näkyviin kehittäjäasetukset. Kehittäjäasetuksista löytyy kohta Usb-Vianetsintä tai Usb-Debug. Usb-vianetsintä täytyy kytkeä päälle, jonka jälkeen puhelin kytketään usb-kaapelilla tietokoneeseen ja avataan Unity Remote 5-sovellus. Kun sovellus on käynnistetty voi tietokoneelta avata oman Unity-projektin. Play-tila näkyy nyt myös reaaliaikaisena mobiililaitteessa. (Kelly 2016, viitattu 14.03.2018.)

Unity Remote 5 on loistava työkalu, kun kehittää sovellusta tai peliä mobiilialustalle. Aikaa säästyy todella paljon ja pääsee heti kokeilemaan ja näkemään, miten oma projekti soveltuu mobiili-laitteille. (Kelly 2016, viitattu 14.03.2018.)

4 KONSEPTI

Pelin keksiminen osoittautui vaikeaksi. Oli todella vaikea keksiä jotain omaa ja uniikkia, mitä ei olisi jo keksitty. Käytettiin paljon aikaa uusien peli-ideoiden miettimiseen. Erilaisia pelejä ladattiin sekä Android-laitteille, että iOS-laitteille. Pelejä pelattiin ja koitettiin samalla keksiä erilaisia ratkaisuja samanlaisten pelien toteutukseen. Piirreltiin paperille erilaisia peli ideoita, ja muutama idea osoittautui kehittämisen arvoiseksi.

4.1 Muutama idea

Muutamia peli-ideoita olivat esimerkiksi 2D-tasohyppely, jossa pelaaja liikkuu automaattisesti eteenpäin. Se mitä pelaaja voi kontrolloida on liikkuminen ylös ja alas tai / ja hyppiminen, mikä riippuu onko pelaaja jokin hahmo, joka voi liikkua vain kiinteällä alustalla vai onko pelaaja jokin lentävä esine tai asia. Tämän kaltaisia 2D- ja 3D-pelejä on tarjolla huomattava määrä niin Android-laitteille kuin iOS-laitteille. Yritettiin miettiä jotain uutta lisää vastaaviin peleihin, mikä voisi erottaa peliprojektin muiden joukosta. Uusien ideoiden keksimättömyys johti peli-ideasta luopumiseen.

Toinen idea oli peli, jossa pelaaja ohjaa pallonmuotoista peliobjektia. Pelaajan pyrkimys on läpäistä erilaisia kenttiä eli päästä pisteestä A pisteeseen B. Pelin kentät sijoittuisivat korkealle pelimaailmassa. Kentät vaatisivat tarkkaa ohjausta pelaajalta, ettei pelaaja tippuisi kentältä pois. Vastaavia pelejä löytyy jo, ja yksi hyvä esimerkki on Ball Resurrection, jonka voi ladata Androidille, että myös iOS-laitteille. Peliä testattua tuli todettua pelin olevan juurikin samanlainen, kuin suunnitelmissa. Peli oli tehty myös Unitylla. Prototyyppi kyseisestä pelistä toteutettiin, mutta sitä ei otettu projektiksi.

4.2 Pelin ideointi

Pelin ideoinnin ilmettyä haastavaksi muutettiin ajatusmaailmaa. Peli-ideaa mietittiin aluksi jokseenkin ammattimaisesti miettien eri kohderyhmiä ja genrejä. Lähestyttiin kuitenkin ideaa ajatellen omia pelimieltymyksiä. Haluttiinko matkapuhelimelle, tabletille vai molemmille? Minkälaiset kontrollit ovat mieleen, ja voiko peliä pelata hetken silloin tällöin vai vaatisiko se enemmän aikaa? Mieleeni ovat

pelit, joissa kilpaillaan aikaa vastaan, missä on myös mahdollisuus pelata toista pelaajaa vastaan, mutta peliä ei tarvitse pelata yhdellä kertaa liian kauan.

Mobiilipeli, josta inspiraatiota haettiin, on nimeltään Bike Race (katso kuvio 4). Kyseinen peli löytyy niin Androidille sekä iOS:lle. Peli on 2D-kilpa-ajopeli, jossa ajetaan erilaisilla moottoripyörillä eri radoilla. Pelissä voi haastaa kaverin tai kenet vain peliä pelaavan. Haaste-pelimuodossa kumpikin ajaa saman radan läpi milloin itsellä on aikaa ja radan nopeimmin suorittanut voittaa. Peli on yksinkertainen, mutta viihdyttävä ja yhden kentän kerkeää suorittamaan vaikka hissiä odotellessa.

KUVIO 4. Kuva Bike Race pelistä

4.3 2D vai 3D

3D-pelin kehittämisessä tarvitsee enemmän tietoa ja taitoa niin koodaamisessa kun artistipuolella. 3D-sisällön tekeminen vaatii monia teknisiä ja taiteellisia taitoja sekä syvällistä tietoa 3D-työkaluissa. Jos mietitään pitkällä aikavälillä, on 2D-pelin tekeminen nopeampaa ja halvempaa, kuin 3D-pelin tekeminen. Oli niin tai näin, peli-ideaa kehittäessä täytyy tarkasti miettiä haluaako 2D-pelin vai 3D-pelin. Kaikki pelit eivät sovi 3D maailmaan ja toisinpäin. (Naser 2015, viitattu 03.02.2018.)

Oma taitotaso täytyy tietää varsinkin, jos pelin tekee yksin. Mobiilille 3D-pelin tekeminen edellyttää entistä enemmän tietotaitoa puhelimien ja tablettien suorituskyvyn takia. Tehoa puhelimista saa

vain jonkin verran, joten täytyy tietää mitä tekee. 2D-pelit eivät ole niin raskaita, joten ei tarvitse olla niin tarkkana. Tässä projektissa päädyttiin 2D-peliin juurikin oman taitotason arvioinnin jälkeen.

4.4 Android vai iOS

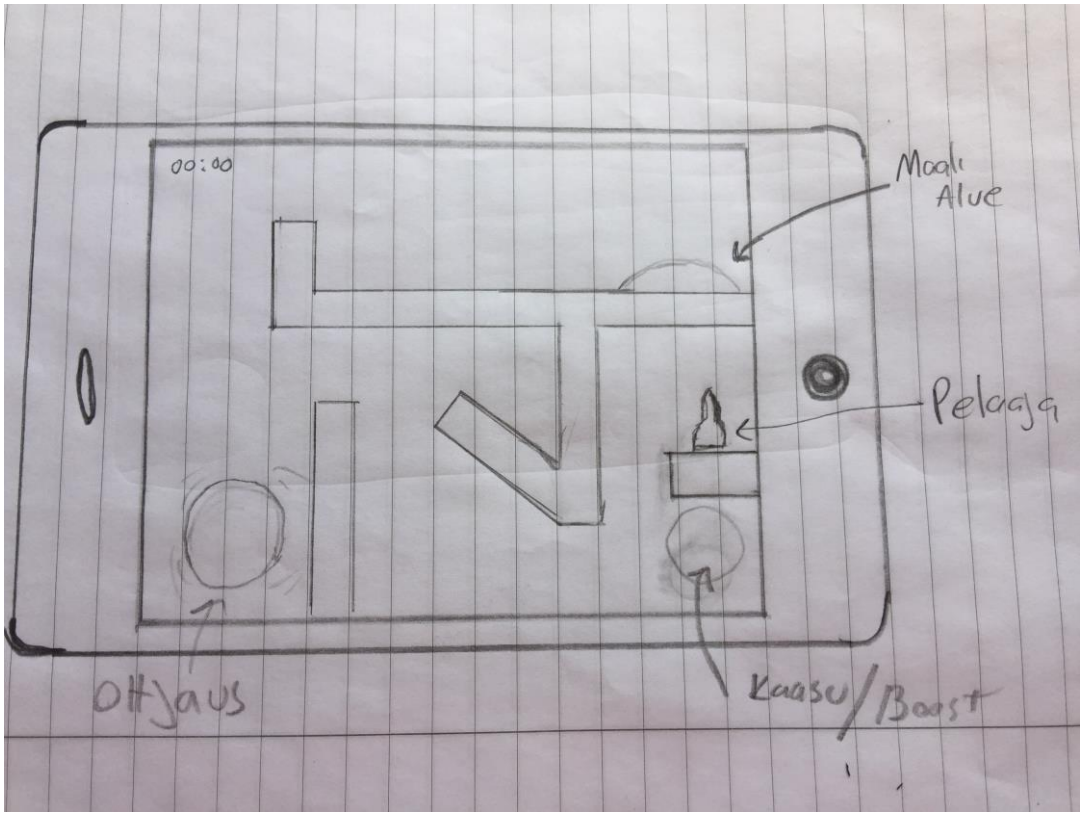
Mobiilipelejä voi tehdä usealle alustalle, mutta yleisimmät ja suosituimmat ovat Android ja iOS. Henkilökohtaisesti pidän Android-laitteita parempina, kuin iOS-käyttöjärjestelmää käyttäviä laitteita. Yksi tavoite on saada opinnäytetyö projekti joskus tulevaisuudessa julkaistua Android- ja / tai iOS-laitteille. Applen ja Googlen sovelluskauppojen sivuilta löytyy hyvät ohjeet, kuinka lisätä oma tuote myyntivalikoimaan. Applen ja Googlen välillä on kuitenkin yksi suuri ero. Googlen sovelluskauppaan omien tuotteiden julkaiseminen hoituu kertamaksulla. Google veloittaa n.25 €, jonka jälkeen sovelluksia voi julkaista rajattoman määrän koko eliniän ajan. Applen appstore toimii hieman erilailla. Apple veloittaa n.99 € joka vuosi samoista palveluista. Omalla kohdalla valinta oli aika selvä. Olin aina ajatellut tekeväni pelini Androidille, ja hintojen selvittyä oli helppo lyödä päätös lukkoon. (Sims 2014, viitattu 30.02.2018.)

5 PELI

Peli kehitettiin 2D-muodossa syistä, joita mainittiin kappaleessa 4.3, ja peli tehtiin Android-käyttöjärjestelmää käyttäville laitteille syistä, joita mainittiin kappaleessa 4.4.

Pelaaja ohjaa rakettia, jonka tavoitteena on päästä kentän alusta kentän loppuun sille merkitylle paikalle eli maaliin. Rakettia ohjataan kosketusnäytön vasemmasta alareunasta. Pelaaja voi muuttaa vain raketin kärjen suuntaa kyseisellä ohjausmekaniikalla. Kosketusnäytön oikeassa alareunassa on painike, jolla raketille annetaan työntövoimaa.

Pelin kentistä tehtiin yksinkertaisia helpottaakseen pelaajaa ymmärtämään pelin mekaniikat ja miten maalialueelle pääsee. Kentät täytettiin esteillä, joilla pyritään muodostamaan ymmärrettävä rata kohti maalia. Raketin osuessa kentissä oleviin esteisiin raketti tuhoutuu ja kyseinen kenttä alkaa alusta. Pelaaja näkee koko kentän ruudussaan, eli kentät ovat kohtalaisen pieniä, mikä toi haasteen saada pelaajaobjektin liikuttaminen sulavaksi. Pelaaja pelaa ensisijaisesti kelloa vastaan ja pyrkii läpäisemään kunkin kentän nopeimmalla mahdollisella ajalla.



KUVIO 5. Ensimmäinen hahmotelma

5.1 Toteutus

Kun Unitylle ja mobiililaitteelle oli saatu asetettua oikeat asetukset, kokeiltiin miten kosketusnäyttö reagoi kosketukseen. Internetistä löytyi skripti, jonka avulla pystyi selvittämään, että reagoiko Unity kosketusnäyttöä koskettaessa. Unity remotea käyttäen testattiin toimivuus. Kosketusnäyttöä koskettaessa saatiin ilmoitusteksti, joka ilmoitti montako sormea kosketti kosketusnäyttöä, ja mikä oli niiden sijainti ruudulla (katos kuvio 6).

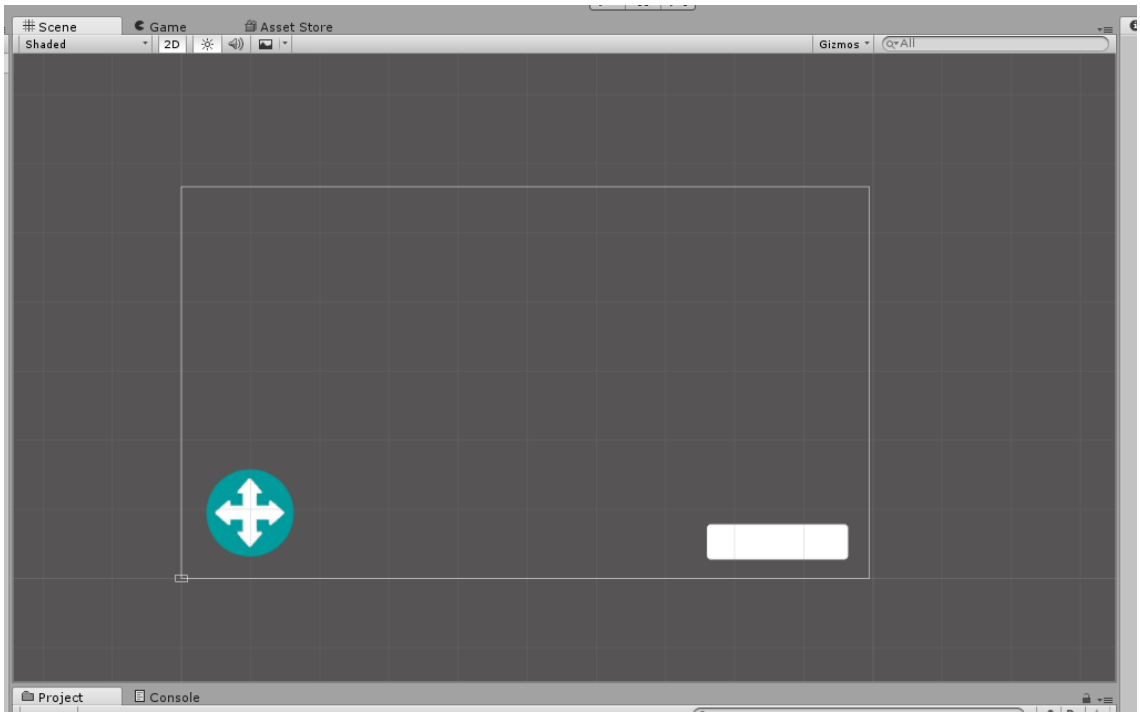
```
5  public class TouchScript : MonoBehaviour {
6
7
8      // Update is called once per frame
9      void Update ()
10     {
11     }
12
13     void OnGUI()
14     {
15         foreach (Touch touch in Input.touches)
16         {
17             string message = "";
18             message += "ID: " + touch.fingerId + "\n";
19             message += "Phase: " + touch.phase.ToString() + "\n";
20             message += "TapCount: " + touch.tapCount + "\n";
21             message += "Pos x: " + touch.position.x + "\n";
22             message += "Pos y: " + touch.position.y + "\n";
23
24             int num = touch.fingerId;
25             GUI.Label (new Rect (0 + 130 * num, 0, 120, 100), message);
26         }
27     }
28 }
```

KUVIO 6. Kosketusnäytön testaamisessa käytetty koodi.

5.2 Kontrollit

Kaikkien asetuksien selvittämisen ja testaamisen jälkeen projekti pääsi kunnolla vauhtiin. Ensimmäisenä tehtiin kontrollit. Unityn omista Standard Asseteista löytyi valmiit kontrollit mobiilipeliä varten. Oikean assetti-paketin voi ladata Unity editorin kautta Assets > Import Package > Cross Plat-

form Input. Paketista löytyy useampia kontrolleja eri tarkoitukseen, mutta tässä projektissa käytettiin Mobile Single Stick Control Prefabia. Prefabiin kuului joystick, painike sekä tarvittavat skriptit (katso kuvio 7). (Curry 2015, viitattu 22.03.2018.)



KUVIO 7. Mobile single stick prefab

Pelissä joystick ohjaa pelaaja-objektia ja painiketta painettaessa annetaan pelaaja-objektille työntövoimaa. Kontrollit olivat melkein sellaiset kuin niistä haluttiin jo valmiiksi, mutta joystickin skriptiin piti tehdä muutoksia. Joystickia liikuttaessa oli sen liikealue rajattu neliön muotoiseksi ja se piti saada ympyrän muotoiseksi, koska neliönmuotoinen liikealue ei tuntunut optimaaliselta. Internetistä löytyi ohje valmiiksi, kuinka muuttaa joystickin liikealue halutuksi(katso kuvio 8, 9). (Curry 2015, viitattu 22.03.2018.)

```
    if (m_UseX)
    {
        int delta = (int)(data.position.x - m_StartPos.x);
        //delta = Mathf.Clamp(delta, - MovementRange, MovementRange);
        newPos.x = delta;
    }

    if (m_UseY)
    {
        int delta = (int)(data.position.y - m_StartPos.y);
        //delta = Mathf.Clamp(delta, -MovementRange, MovementRange);
        newPos.y = delta;
    }
    transform.position = Vector3.ClampMagnitude(new Vector3(newPos.x, newPos.y, newPos.z), MovementRange) + m_StartPos;
    UpdateVirtualAxes(transform.position);
}
```

KUVIO 8. Joystick koodi

```
    if (m_UseX)
    {
        int delta = (int)(data.position.x - m_StartPos.x);
        delta = Mathf.Clamp(delta, - MovementRange, MovementRange);
        newPos.x = delta;
    }

    if (m_UseY)
    {
        int delta = (int)(data.position.y - m_StartPos.y);
        delta = Mathf.Clamp(delta, -MovementRange, MovementRange);
        newPos.y = delta;
    }
    transform.position = Vector3.ClampMagnitude(new Vector3(newPos.x, newPos.y, newPos.z), MovementRange) + m_StartPos;
    UpdateVirtualAxes(transform.position);
}
```

KUVIO 9. Muokattu Joystick koodi

5.3 Joystick, Painike

Kontrollien ulkonäöllinen käyttäytyminen saatiin mieleiseksi. Seuraavaksi luotiin yksinkertainen pelaajaobjekti, joka täytyi saada toimimaan halutulla tavalla. Haluttu pelaajaobjektin käyttäytyminen osoittautui kuitenkin hieman ongelmalliseksi. Täysin oikeanlaista koodia ei löytynyt, vaan koodia joutui hieman muokkaamaan.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

public class Player2DController : MonoBehaviour {


    public float moveForce;//boostMultiplier = 2;
    Rigidbody2D myBody;

    void Start ()
    {
        myBody = this.GetComponent<Rigidbody2D>();
    }


    void Update ()
    {

        //Vector2 moveVec = new Vector2 (CrossPlatformInputManager.GetAxis("Horizontal_2"),
        //CrossPlatformInputManager.GetAxis("Vertical_2")) * moveForce;
        // float moveVec = (CrossPlatformInputManager.GetButtonDown("boost")) * moveForce;

        Vector3 lookVec = new Vector3 (CrossPlatformInputManager.GetAxis("Horizontal"),
            CrossPlatformInputManager.GetAxis("Vertical"), 4096);

        if (lookVec.x != 0 && lookVec.y !=0)
        transform.rotation = Quaternion.LookRotation (lookVec, Vector3.back);


        //bool isBoosting = CrossPlatformInputManager.GetButton ("Boost");
        //myBody.AddForce (moveVec * (isBoosting ? boostMultiplier : 1));

        if (CrossPlatformInputManager.GetButton ("Boost"))
            myBody.AddForce(transform.up * moveForce);
```

KUVIO 10. Pelaajaobjektin ohjauskoodi

Kuviossa 10 näkyvä koodi toimi niin, että ruudulla oli kaksi joystickia, joista toisella ohjattiin pelaajaobjektin suuntaa ja toisella joystickilla liikutettiin pelaajaobjektia haluamaan suuntaan. Ruudulla oli myös painike, jolla pystyi nopeuttamaan pelaajaobjektin liikkumista painiketta painamalla.

Pelaajan ohjaaminen haluttiin toimivan siten, että yhdellä joystickillä ohjataan pelaajaobjektin suuntaa ja painiketta painamalla annettiin työntövoimaa. Haluttu toiminta saatiin aikaan kuviossa 10 näkyvää koodia muokkaamalla siten, että harmaat koodilauseet poistettiin ja loppuun lisättiin if-lause. (Curry 2016, viitattu 04.04.2018.)

5.4 Kenttäsuunnittelu

Pelin hahmottelu alkoi ensimmäisen kentän luomisella (katso kuvio 11). Kentästä tehtiin hyvin yksinkertainen, missä oli toimivat kontrollit, pelaajaobjekti ja esteitä. Internetistä löytyi nopeasti etsittynä tarvittavat spritet grafiikkaa varten, mitkä saatiin nopeasti lisättyä peliin, jolloin peliä pystyi paremmin hahmottelemaan.

KUVIO 11. Level1-hahmotelma

Kenttään asetettujen esteiden jälkeen lisättiin colliderit esteille, jotta niiden läpi ei päässyt liikkumaan pelaajaobjektilla. Esteiden todettua toimiviksi tehtiin tarvittavat valikot. Tarvittiin ns. kuolema-valikko, joka avautuisi pelaajaobjektin tuhoutuessa. Tarvittiin valikko myös pelaajaobjektin päästessä kentän maalialueelle. Valikoiden jälkeen täytyi tehdä toiminnallisuudet, jotta valikot käyttäytyisivät oikealla tavalla.

Löytyi useita eri vaihtoehtoja, kuinka suorittaa pelaajaobjektin tuhoutuminen ja valikoiden avautuminen. Päätettiin käyttää toimintoa, joka havaitsee pelaajaobjektin ja esteiden colliderien osuminen toisiinsa. Esteisiin ja kentän ulkopuolelle laitettiin colliderit, jotka asetettiin Unity editorissa Hierarchy valikossa tyhjän peliobjektin alle. Tyhjä peliobjekti nimettiin ”KillZones”. Pelaajaobjekti tägättiin tag-toiminnolla ”Player”. Tarvittiin toiminto, joka lopettaa pelin ja avaa kuolema-valikon pelaajan osuessa player-objektilla KillZones-objektin alla oleviin collidereihin. Toiminto toimii siten, että pelaajan player-objektin osuessa KillZone-collideriin, peli keskeytyy ja kuolema-valikko aukeaa. Valmis skripti löytyi valmiiksi, joka vastasi haluttua toimintoa. Skriptiin tehtiin vain pieni muutos, että saatiin haluttu toiminto toimimaan (katso kuvio 12).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class DeathMenu : MonoBehaviour {

    public GameObject DeathMenuUI;

    void OnTriggerEnter2D(Collider2D KillZones)
    {
        if(KillZones.gameObject.CompareTag("Player"))
        {
            DeathMenuUI.SetActive (true);


        }
```

KUVIO 12. Pelaajan osuessa killzone-collideriin

Seuraavaksi kenttään piti saada maalialue, johon pelaajan päästessä peli pysähtyy ja aukeaa valikko, johon tulee näkyviin kentän läpäisyyn tarvittu aika sekä mahdollisuus jatkaa seuraavaan kenttään. Toiminto tehtiin samalla systeemillä, kuin miten pelaajan tuhoutuminen tehtiin. Skriptiin tuli muutos, joka asettaa KillZone-colliderit pois käytöstä pelaajan saapuessa maaliin. Tällä estettiin useamman valikon päällekkäinen avautuminen tietyssä tilanteessa (katso kuvio 13). (Unity 2014, viitattu 10.04.2018.)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FinishMenu : MonoBehaviour {

    public GameObject KillZones;

    public GameObject FinishMenuUI;

    void OnTriggerEnter2D(Collider2D Testi)
    {
        if(Testi.gameObject.CompareTag("Player"))
        {
            FinishMenuUI.SetActive (true);

            KillZones.gameObject.SetActive (false);
        }
    }
}
```

KUVIO 13. Maalialueen koodi

6 UI

KUVIO 14. Päävalikko

6.1 Päävalikko

Alkuvalikko tehtiin omalle scenelle. Valikosta on mahdollisuus tehdä kolme eri suoritusta. Play-nimistä painiketta painamalla pelaaja aloittaa pelin ensimmäisestä kentästä. Levels-painiketta painamalla alkuvalikon tilalle tulee kenttävalikko, josta voi valita haluamansa kentän. Options-painiketta painamalla aukeaa options-valikko (katso kuvio 14). (Brackeys 2017, viitattu 10.04.2018.)

Valikkojen väliset vaihtumiset on tehty painikkeiden omalla On Click -toimintoa käyttäen. Toimintoa käyttämällä voi suorittaa lukuisia toimintoja ilman koodaamista. Toimintoa käytettiin valikoiden aktivoimiseksi ja aktivoimiseksi pois päältä. Painikkeita painamalla auki oleva valikko aktivoituu pois näkyvistä ja jokin muu valikko aktivoituu näkyväksi. (Brackeys 2017, viitattu 10.04.2018.)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour {

    public void PlayMenu ()
    {
        SceneManager.LoadScene("MainMenu");
    }
}

```

KUVIO 15. Kentän valinta koodi

Projektissa on kolme kenttää, jotka voi valita levels-valikosta ja jotka ovat omilla sceneillä. Myös aloitusvalikko on omalla scenellä. Kuvion 15 koodi kopioitiin kaikkiin sceneihin sopivaksi.

6.2 Kuolema ja maali Valikko

Pelaajan kuollessa avautuu valikko, jota kautta pääsee aloittamaan saman kentän uudestaan tai palaamalla takaisin aloitusvalikkoon. Samaa koodia käytettiin mitä aloitusvalikossakin määrittelemään oikean scenen latautuminen.

Pelaajan päästessä kentän maalialueelle aukeaa valikko, jossa onnitellaan kentän läpäisystä. Valikosta on mahdollista aloittaa seuraava kenttä tai palata takaisin aloitusvalikkoon. Seuraavan kentän latausta varten kopioitiin erilainen koodi. (Brackeys 2017, viitattu 10.04.2018.)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class NextLevel : MonoBehaviour {

    public void PlayGame()
    {
        SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex + 1);
    }


}
```

KUVIO 16. Seuraavan kentän latauskoodi

6.3 Ajanotto

Projektin toimivuutta testattiin paljon kaikkien toimintojen lisäämisen jälkeen. Kaikkien toimintojen todettua toimiviksi lisättiin ajanotto peliin. Ruudun vasempaan yläkulmaan asetettiin ajastin. Ajastin lähtee käyntiin, kun valittu kenttä latautuu pelattavaksi. Ajanotto loppuu, jos pelaaja kuolee ennen maaliin tuloa. Ajanotto loppuu myös pelaajan saapuessa maalialueelle. Maalivalikon auettua tulee valikkoon kulunut aika näkyville.

Valmiita ajanottokoodeja löytyi useita, mutta ei juuri sopivaa tähän käyttötarkoitukseen. Koodi, jota käytettiin, saatiin toimivaksi yhdistämällä kuviossa 17 näkyvä koodi sekä kuviossa 12 näkyvä koodi. (N3K EN 2015, viitattu 22.04.2018.)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour {

    public Text timerText;
    public Text timerText2;
    private float startTime;
    private bool stopTimer = false;

    // Use this for initialization
    void Start () {

        startTime = Time.time;
    }

    // Update is called once per frame
    void Update () {
        if (stopTimer)
            return;

        float t = Time.time - startTime;

        string minutes = ((int)t / 60).ToString ();
        string seconds = (t % 60).ToString ("f2");

        timerText.text = minutes + ":" + seconds;
        timerText2.text = minutes + ":" + seconds;
    }

    void OnTriggerEnter2D(Collider2D player)
    {
        if(player.gameObject.CompareTag("DeathCollision"))
        {

            stopTimer = true;
            //print ("works");
        }

}
}
```

KUVIO 17. Ajanottokoodi

7 POHDINTA

Minulla on kokemusta Unity pelimoottorista, mutta en ennen opinnäytetyötä ollut tehnyt omaa peliä omia taitoja käyttäen. Opinnäytetyön aiheeksi olin aina miettinyt mobiilipelin tekemistä. Pelikehityskurssien aikana ja Oulu Game Labilla opiskellessani aina mietin, että miksi ei etsitä internetistä valmiita skriptejä vaan aina ne kirjoitettiin tyhjästä. Oma tieto ja taito koodaamiseen liittyen oli todella minimaalinen ennen opinnäytetyön tekemistä. Halusin selvittää onko mahdollista tehdä edes yksinkertaista peliä ilman, että itse joutuu koodaamaan.

Jokaiseen toimintoon mitä tähän projektiin tarvittiin, löytyi skriptejä valmiiksi. Todella moni skripti kävi sellaisenaan, eikä niitä tarvinnut muokata ollenkaan. Oli kuitenkin skriptejä, joita joutui muokkaamaan, mutta muokkaukset olivat sen verran yksinkertaisia, että ne onnistuin itse tekemään.

Opin koodaamista ja ennen kaikkea ymmärtämään koodia tämän projektin aikana todella paljon. Yksinkertaisen pelin voi tehdä ilman, että itse koodaa, mutta sitä täytyy kuitenkin vähän ymmärtää ja käyttää oikein.

Peliprojektiin olen tyytyväinen. Pelin sain pelattavaan kuntoon ja kaikki toiminnallisuudet toimimaan. Prototyyppiin sain tehtyä toimivan alkuvalikon sekä kolme erilaista ja toimivaa kenttää. Tulevaisuudessa olen ajatellut jatkavan pelin kehittämistä eteenpäin.

8 LÄHTEET

Brackeys, 2017. Start MENU in Unity. Viitattu 10.04.2018,
https://www.youtube.com/watch?v=zc8ac_qUXQY.

Charger Games, 2017. How To Make an Android Game With Unity – Complete Tutorial 2017. Viitattu 07.03.2018,
https://www.youtube.com/watch?v=nM0h5pQYQxM.

Curry, D. 2015. Unity 5 Mobile Joysticks Tutorial – Tpuch Input 2D Spaceship Controller. Viitattu 22.03.2018,
https://www.youtube.com/watch?v=nGYObojmkO4.

Curry, D. 2016. Unity Twin Stick Touch Input Addendum.
Viitattu 04.04.2018, https://www.youtube.com/watch?v=qtS-paRqJYU.

Kelly, S. 2016. Unity Remote 5 Setup and First Test (Multi-Touch). Viitattu 14.03.2018, https://www.youtube.com/watch?v=usT9Sv00v8w&index=19&list=PLZo2FfoM-kJeEyG8QokK2z9JvI7qCvdGZg.

Naser, A. 2015. Why to Develop 2D Games for Mobile Instead of 3D Games. Viitattu 03.02.2018, https://www.linkedin.com/pulse/why-develop-2d-games-mobile-instead-3d-ahmad-hammad.

N3K EN, 2015. How to create a Timer [Tutorial][c#] – Unity 3d. Viitattu 22.04.2018,
https://www.youtube.com/watch?v=x-C95TuQtf0.

Sinicki, A. 2016. An introduction to Unity#D for easy Android game development. Viitattu 02.02.2018,
https://www.androidauthority.com/an-introduction-to-unity3d-666066/.

Sims, G. 2014. Publishing your first app in the Play Store: what you need to know. Viitattu 30.02.2018,
https://www.androidauthority.com/publishing-first-app-play-store-need-know-383572/.

Unity, 2014. How to make it so collision only triggers with a certain object. Viitattu 10.04.2018, https://answers.unity.com/questions/743826/how-to-make-it-so-collision-only-triggers-with-a-c.html.