Mobiilipelin toteuttaminen Unity 2D:llä

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tietojenkäsittelyn koulutusohjelma

Hämeenlinna, kevät 2017

Miklas Hakkarainen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki, Hämeenlinna

Tekijä Miklas Hakkarainen **Vuosi** 2017

Työn nimi Mobiilipelin toteuttaminen Unity 2D:llä

Työn ohjaaja/t Tommi Saksa

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli luoda yksinkertainen mobiilipeli Android-käyttöjärjestelmälle. Mobiilipeli toteutettiin käyttäen Unityä. Pelin script-tiedostojen luomiseen ja muokkaamiseen käytettiin Microsoft Visual Studio 2010 -ohjelmointialustaa. Ohjelmointikielenä pelissä toimi C#(C-Sharp). Lopullisen työn tarkoitus oli luoda selkeä kuva, kuinka luoda mobiilipeli käyttäen Unityä.

Opinnäytetyön teoriaosassa käydään läpi tietoa mobiilipelaamisesta, Unitystä sekä tietoa pelisuunnittelusta. Mobiilipelaamisen osiossa käydään läpi mobiilipelaamisen historiaa sekä erityispiirteitä ja etuja, joita mobiilipelaaminen sisältää. Unitystä avataan yleisimpiä käsitteitä ja termejä, jotka ovat hyödyllisiä peliä kehittäessä. Pelisuunnittelusta käsitellään yleisiä asioita kuten ideointi, yleiset kompastuskivet ja mobiilialustan haasteet.

Käytännön osuudessa luodaan yksinkertainen 2D-mobiilipeli sekä ohjeet, kuinka sitä testataan Android-käyttöjärjestelmällä. Ensimmäisessä osiossa selitetään mitä tarvitaan pelin testaamiseen Android-alustalla ja toteutusohjeet. Toisessa osiossa käsitellään pelin päävalikon luomista ja sen ominaisuuksia sekä script-tiedostoja ja sitä, mitä ominaisuuksia ne pelissä toteuttavat.

Unity-pelimoottori osoittautui helpoksi ja käteväksi vaihtoehdoksi mobiilipelin toteuttamiseen, ja se soveltuu hyvin aloittelevalle pelintekijälle. Työn tuloksena on yksinkertainen, toimiva mobiilipeli.

Avainsanat mobiilipelaaminen, peliohjelmointi, Unity, c#, Android

Sivut 28 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki, Hämeenlinna

Author Miklas Hakkarainen **Year** 2017

Subject Creating mobilegame on Unity 2D

Supervisors Tommi Saksa

ABSTRACT

The goal of the thesis was to create a simple mobilegame on Android device. The game was created using Unity. The script files in the game were created and modified using Microsoft Visual Studio 2010-programming tool. C# (C-Sharp) was used as programming language. The idea of the thesis was to create a simple manual for beginning programmers.

In the theory part of the thesis there is information about mobile gaming, Unity and game designing. The mobile gaming part includes information about its history, special features and the benefits. From the Unity there is explanations of all the most common concepts and terms which are useful when producing a game. From the game designing there is information about brainstorming, common troubles and challenges of mobile platform.

The practical part of the thesis there is created a simple 2D-mobile game and also includes instructions how to test it on Android device. In the first part there is explained what is needed to test the game on Android device and the instructions. The second part is about creating the main menu for the game and also of the games script files and which features are they implementing.

Unity game engine proved to be easy and handy option for developing a mobile game and it is suitable for beginning programmers. Result of the project is a simple and working mobile game.

Keywords mobilegaming, game programming, Unity, c#, Android

Pages 28 pages

SISÄLLYS

1 JOHDANTO

Mobiilipelaaminen on kehittynyt huomattavan paljon viimeisten vuosien aikana. Puhelimet ovat kehittyneet niin paljon, että niille voidaan luoda jo sisällöltään suuriakin pelejä. Tässä opinnäytetyössä esittelen mobiilipelaamisen historiaa, sen merkitystä sekä tuon esille monia erityispiirteitä, joita mobiilipelin ohjelmoijan tulee ottaa huomioon. Lisäksi työn aikana luon oman mobiilipelin Unity-pelimoottoria käyttäen. Teoriaosassa tuon esille paljon tietoa Unity-pelimoottorista, jotka ovat todettu hyödyllisiksi. Lisäksi teoriaosassa käyn läpi pelien suunnittelemista, minkälaiset asiat siihen vaikuttavat ja mitä suunnittelussa tulee ottaa huomioon.

Opinnäytetyön tavoitteena on luoda yksinkertainen peli Android-käyttöjärjestelmälle käyttäen Unity-ohjelmointialustaa. Unityn valinta ohjelmointialustaksi oli luonteva, koska sitä käytettiin jonkin verran koulun kursseilla, se on helppokäyttöinen ja käyttäjäystävällinen aloitteleville pelinkehittäjille. Kiinnostukseni Unityä kohtaan on kasvanut päivä päivältä. Ohjelmointikielenä käytän C# (C Sharp)-kieltä, koska minulla on siitä jonkin verran kokemusta ja se on kokemuksieni perusteella yksinkertainen ja helposti opittava ohjelmointikieli.

Opinnäytteestä tulee opas aloittelevalle pelinkehittäjälle. Sen tarkoitus on tarjota lukijalle hyvä pohja kehittää oma mobiilipeli. Opas osoittaa, millaisia erityispiirteitä pitää ottaa huomioon mobiilipeliä kehitettäessä. Erityisesti tulen opinnäytetyössäni keskittymään mobiilipelin erityisvaatimuksiin, mobiilipelin luomiseen Unityllä sekä siihen, miten tietyt mobiilipeleissä keskeiset ominaisuudet toteutetaan. Hyvä esimerkki mobiilipelien keskeisimmistä ominaisuuksista on peliobjektin järkevä kontrollointi kosketusnäytöllä.

2 MOBIILIPELIT

Tässä luvussa käyn läpi mobiilipelaamisen historiaa, mitä mobiilipelaaminen nykyaikana on ja kuinka paljon se on vuosien varrella kehittynyt. Mobiilipelaamisella tarkoitetaan pelaamista kannettavilla elektronisilla pelilaitteilla. Mobiilipelaaminen on nykyisin yksi nopeimmin kasvaneista peliteollisuuden osa-alueista. (Paavilainen, 2009.)

Mobiilipelit voidaan luokitella pelaamisen yhdeksi alalajiksi. Mobiililaitteiden kehittyminen ja monipuolistuminen ovat vaikuttaneet mobiilipelien tarjontaan suuresti. Mobiilipelien kehittämiseen ei juurikaan tarvita suuria resursseja, joten ne yleensä tuotetaan pienissä muutaman hengen kokoonpanoissa. (Kurtti 2015.)

2.1 Historia

Mobiilipelaamisen historia alkaa jo niinkin aikaisin kuin 1970-luvulta. Silloin ilmestyivät ensimmäiset kannettavat elektroniset pelilaitteet markkinoille. (Paavilainen, 2009.)

Ensimmäinen markkinoille saapunut pelilaite olit Mattel Auto Race (Mattel, 1976), joka sisälsi yhden pelin. Alkuaikoina näille pelilaitteille oli tyypillistä, että yksi laite sisälsi vain yhden pelin. Mattel Auto Race -pelin idea oli että pelaaja ohjasi pientä valopistettä kolmikaistaisella radalla, väistellen vastaantulijoita. Peli sisälsi myös 99 sekunnin aikarajan ja neljä vaihdetta. (Paavilainen, 2009.)

Vuonna 1977 Mattel julkaisi pelin nimeltä Mattel Electronic Football. Peli simuloi amerikkalaista jalkapalloa. Sears-tavarataloketju ei uskonut pelistä tulevan suurta menestystä, mutta silti peliä myytiin myöhemmin parhaillaan 500 000 kappaletta viikossa helmikuussa 1978. (Paavilainen, 2009.)

Huippusuosituiksi nousseet Game & Watch -kannettavat pelilaitteet Nintendo julkaisi vuonna 1980. Näissä pelilaitteissa oli tyypillisesti yksi tai kaksi näyttöä ja siihen aikaan erityinen plus-merkin näköinen D-Pad-ohjain, joka pelimaailmassa otettiin myöhemmin standardiksi. Game & Watch -pelilaitteisiin sisältyi aina yksi peli, joka käsitti yleensä kaksi valittavaa vaikeustasoa. Pelin lisäksi laitteet sisälsivät myös kellon ja herätyksen. (Paavilainen, 2009.)

Nintendo kehitti myös vuonna 1989 pelilaitteen Game Boy, joka nousi maailman suosituimmaksi kannettavaksi pelilaitteeksi. Myös muutkin valmistajat kuten Sega ja Atari kehittivät omat kannettavat pelilaitteensa, mutta näistä ei koskaan tullut kilpailijaa Game Boy -pelilaitteelle tai sen myöhemmille versioille. (Paavilainen, 2009.)

Kuva 1. Matopeli. (Pelkonen 2015.)

Mobiilipelaaminen kehittyi vuonna 1997 huimasti eteenpäin, kun Nokia julkaisi 6110-matkapuhelimen, joka sisälsi Snake-pelin. Tämä klassikkopeli liitettiin lähes jokaiseen Nokian uuteen puhelimeen. (Paavilainen, 2009.) Kuvassa 1 esiintyy yksi ensimmäisistä versioista matopeliin Nokian puhelimella.

2.2 **Mobiilipelien erityispiirteet**

Mobiililaitteiden suurimmat vahvuudet ovat mm. laitteiden aktiivinen käyttöprofiili, suurten alustatoimittajien digitaaliset jakelukanavat, suuret käyttäjämäärät, sosiaalinen verkottuneisuus ja laitteisiin integroitu sensoriteknologia. Yksi suurimmista haasteista pienen mobiililaitteen käyttämisessä pelialustana on mobiililaitteen käyttö samanaikaisesti sekä pienen kosketusnäytön käyttäminen peliohjaimena, mikä voi helposti johtaa suuriin ongelmiin käyttöliittymässä. (Tukiainen 2013.)

Kuva 2. Kuvankaappaus Hill Climb Racing 2 pelistä.

Älypuhelimien pientä kosketusnäyttöä joudutaan lähes poikkeuksetta käyttämään pelien pelitapahtumien esittämiseen sekä peliohjaimena. (Tukiainen 2013.) Kuva 2 toimii erinomaisena esimerkkinä tästä, jossa peliruudun molemmissa laidoissa ovat auton pedaalien kuvakkeet, jolla pelissä painetaan kaasua ja jarrutetaan.

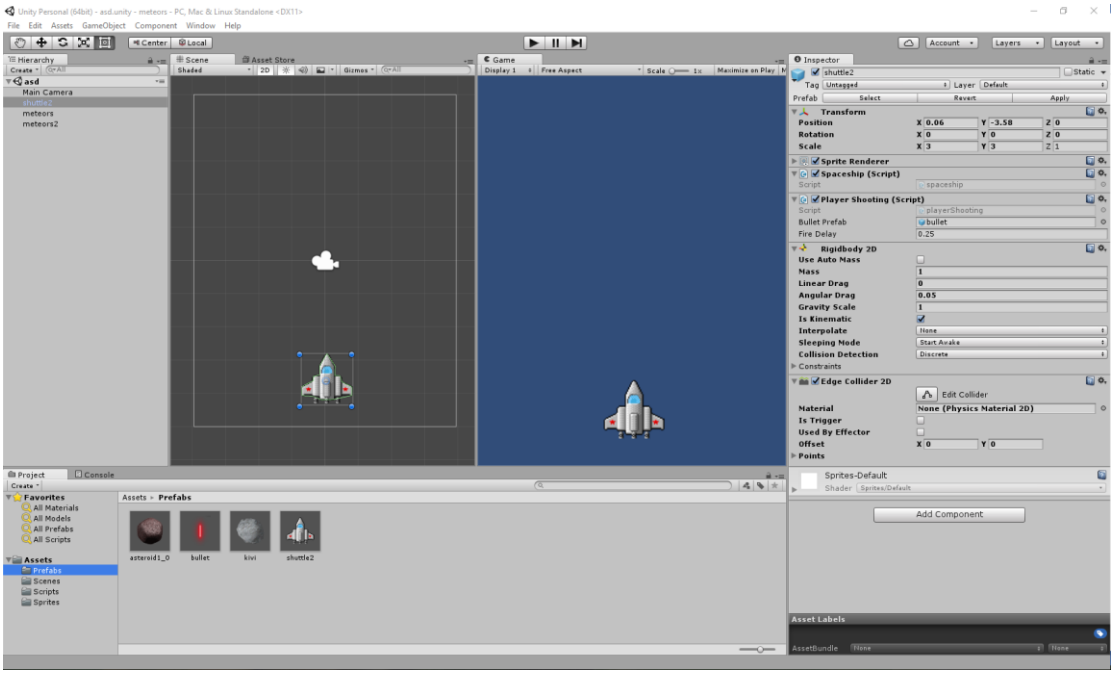
3 UNITY-PELIMOOTTORI

Unity-pelimoottori on pyritty toteuttamaan mahdollisimman helppokäyttöiseksi ja tehokkaaksi työvälineeksi alustariippumatonta pelituotantoa varten. Unityn tuki ylettyy iOS- ja Android-alustojen lisäksi muun muassa seuraaville alustoille: Windows, Windows Store Apps, Windows 8 Phone, OS-X, Linux, verkkoselaimet, PS3, Xbox 360 ja Wii U. (Tukiainen 2013.)

Unity-pelimoottoria käytetään 2D- ja 3D-pelien toteuttamiseen. Tällä pelimoottorilla voidaan kehittää selain-, konsoli- ja PC-pelejä. Unityn lisäksi on saatavilla maksullinen versio Unity Pro. Unityllä ohjelmoidaan kolmella eri ohjelmointikielellä Boo, JavaScript (UnityScript) ja C#. Uusin versio pelimoottorista on Unity 5 ja se julkaistiin 3. maaliskuuta 2015. (Unity Technologies, 2016.)

3.1 Käyttöliittymä

Unityn käyttöliittymää kutsutaan modulaariseksi ja se koostuu viidestä eri elementistä, joiden avulla rakenteen pystyy muuttamaan juuri haluamakseen. Peliä ohjelmoitaessa näkyvyys on jaettu kahteen osaan, scene- ja Game-elementteihin. (Husso 2013.)



Kuva 3. Kuvankaappaus Unityn käyttöliittymästä.

Kuvassa 3 keskellä vasemmalla puolella näkyy Scene-elementti, jolle ohjelmoija voi asettaa peliobjekteja visuaalisesti. Oikealla puolella kuvaa esiintyy Game-elementti, joka esittää tilaa jonka pelaaja näkee. Kun peli ajetaan Unityssä, nähdään molemmista näkymistä reaaliaikaisesti mitä pelissä silloin tapahtuu.

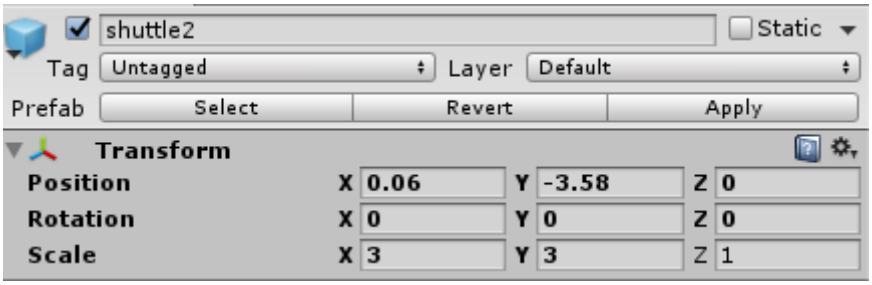
Jokainen peliin lisätty objekti näkyy Project-elementissä ja tämä elementti vastaa reaaliaikaisesti oman projektikansion sisältöä käyttöjärjestelmän resurssienhallinnassa. Hierarchy-elementti toimii Project-elementin apuna ja näyttää vain Scene-elementtiin tallennetut peliobjektit. Hierarchy-elementistä valittu peliobjekti näkyy korostettuna ja muokattavana, jolloin oikeassa reunassa kuvaa näkyvä Inspector-elementti aktivoituu. Inspector-elementistä näkyvät kaikki peliobjektiin liitetyt script-tiedostot, fysiikkamoottorin törmäyksentunnistustyypit ja tekstuurit. (Husso 2013.)

3.2 Scenet

Scene tarkoittaa erillisiä tiloja pelissä, johon on tallennettu erilaisia peliobjekteja. Esimerkkinä on tasohyppelypeli jossa on erilaisia kenttiä, jokainen kenttä pelissä on erillinen scene. Pelissä ei pystytä liikkumaan scenestä toiseen ilman jonkinlaista latausaikaa. Tässä tapauksessa varsinkin mobiilipeliä kehittäessä täytyy ottaa huomioon scenejen koot. Pelin päävalikosta kannattaa myös tehdä oma scene, josta siirrytään seuraavaan sceneen pelaajan syötteen mukaisesti. (Husso 2013.)

3.3 Komponentit

Komponentti on yleistermi erilaisille osille Unityssä, joista peliobjekti koostuu. Komponentit ovat perusrakenneosia, joista koostuvat pelin oliot ja näiden käyttäytyminen. Otetaan esimerkiksi Transform-komponentti, joka määrittää peliolion sijainnin, suunnan sekä mittakaavan. Kaikki pelioliot sisältävät Transform-komponentin oletuksena. (Päiveröinen 2014.) Kuvassa 4 esiintyy peliobjektin Transform-komponentti.



Kuva 4. Kuvankaappaus Unityn Transform-komponentista.

Peliolioille voidaan asettaa myös muitakin komponentteja. Alle on listattu tärkeimpiä komponentteja, joita peliolioihin voidaan liittää (Päiveröinen 2014.):

- Transform määrittää objektin sijainnin, suunnan ja koon
- MeshFilter lukee Mesh-verkon Assets-kansiosta sekä siirtää sen MeshRenderer-komponentille
- MeshRenderer piirtää peliobjektin
- Rigidbody mahdollistaa peliobjektien käyttäytymisen fysiikan lakien mukaisesti
- Collider komponentin avulla Rigidbody-komponentti reagoi törmäyksiin
- CharacterController on realistisemmaksi luotu törmäyskomponentti kuin Rigidbody

- Camera mahdollistaa pelimaailman katselun pelaajan hahmon näkökulmasta
- Light-komponenttia käytetään pelimaailman valaistuksessa
- GUIText on tekstikenttä, joka voidaan liittää peliobjektiin
- MonoBehaviour on pohjaluokka script-tiedostoille, joilla luodaan lisää toiminnallisuuksia peliobjekteille.

Liitettäessä komponentti peliolioon se sisältää paljon erilaisia arvoja tai ominaisuuksia, joita ohjelmoija voi muokata peliä kehittäessä tai script-tiedostoilla pelin suorittamisen aikana. (Päiveröinen 2014.)

3.4 **Script-tiedostot**

Script-tiedostot ovat JavaScriptillä tai C#:lla koodattuja koodinpätkiä, joilla voidaan antaa haluamilleen peliobjekteille toimintoja. Esimerkkejä tällaisista objekteista ovat tekoälyn ohjaama vihollinen, seinä ja pelaajan ohjaama pelaaja. Unityyn on myös rakennettu suuri kirjasto, joka sisältää paljon hyödyllisiä funktioita ohjelmoijalle. MonoBehaviour-luokka on yksi Unityn tärkeimmistä luokista, joka sisältää esimerkiksi Start()-, Update()- ja Awake()-funktiot. Näitä funktioita voidaan asettaa peliobjektille, jotka se ajaa läpi ensimmäisen ruudunpäivityksen, jokaisen ruudunpäivityksen kohdalla tai kun peliobjekti ladataan ensimäistä kertaa ruudulle. (Husso 2013.)

```
using UnityEngine;
using System.Collections;

public class playerShooting : MonoBehaviour {

  public GameObject bulletPrefab;
  public float fireDelay = 0.25f;
  float cooldownTimer = 0;

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

    cooldownTimer -= Time.deltaTime;
    if (Input.GetButton("Fire1") && cooldownTimer <= 0)
    {
      //Debug.Log("Pew!");
      cooldownTimer = fireDelay;

      Instantiate(bulletPrefab, transform.position, transform.rotation);
    }

                }
}
```

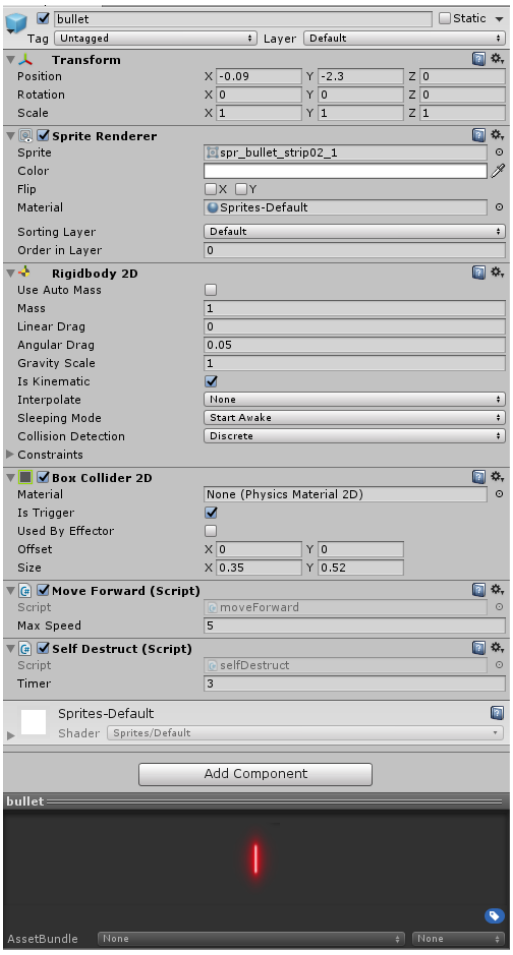
Esimerkki 1. playerShooting script-tiedosto

Esimerkissä 1 esiintyy yksinkertainen script-tiedosto, jossa pelaajalle asetetaan ampumistoiminto. Script-tiedostossa on käytetty hyvin tärkeää Update()-funktiota, jonka sisälle sijoitetut komennot ajetaan jokaisen ruudunpäivityksen aikana. CooldownTimer-muuttujalla asetetaan kuinka usein pelaaja voi ampua. If-lauseen sisällä katsotaan kun pelaaja painaa hiiren ensimmäistä painiketta, tällöin aluksen kohdalle ilmestyy ammuksesta tehty prefabi.

3.5 **Prefabit**

Prefabi on Unityn kirjastossa sijaitseva peliobjekti, joka on tallennettu yhdeksi paketiksi. Prefabit ovat myös käytettävissä muissa peleissä, kuin siinä missä ne on luotu. Prefabeja voidaan monistaa script-tiedostoissa. Tämä on tärkeää esim. ammusten ja muiden objektien kannalta mitkä jotka monistaa. (Husso 2013.)

Tehdyt muutokset prefabissa heijastuvat välittömästi kaikissa sceneissä, mutta voidaan ohittaa osia ja asetuksia kussakin tapauksessa erikseen. (Unity Technologies, 2016.)



Kuva 5. Kuvankaappaus prefabista Inspector-elementissä.

Kuvassa 5 on kuvankaappaus Asteroid-pelissä toimivasta luoti-prefabista, jossa luodille on annettu Rigidbodyllä ja Box Colliderilla jotain fysiikkaa, joilla se voi esimerkiksi törmätä objekteihin. MoveForward script -tiedostolla luoti on asetettu liikkumaan eteenpäin, ja se tuhoutuu osuessaan johonkin muuhun peliobjektiin tai tuhoaa itsensä SelfDestruct script -tiedoston avulla. Esimerkissä 1 näkyvällä script-tiedostolla luodin sijainti määräytyy aluksen sijainnin mukaan.

3.6 **Unity-pelimoottorina mobiilipelissä**

Nykypäivänä Unity on erittäin suosittu pelimoottori mobiilikehityksessä. Pelimoottorista löytyvät omat hyvät ja huonot puolensa. Unityn suurimpia hyötyjä ovat sen helppo käyttöliittymä ja komponenttimaiset objektit. Unityn vahvuuksiin kuuluu myös se että graafikot ja ohjelmoijat voivat työskennellä samalla työkalulla. Käyttäjillä on myös mahdollisuus käyttää Unityn omaa versionhallintaa, mikä helpottaa tiedostojen siirtämistä tekijältä toiselle. Unityn käyttö on helposti opittavissa, varsinkin henkilöiden joilla on kokemusta jonkin pelimoottorin käyttämisestä pelien tekemisessä. (Uurainen 2013.)

Ominaisuus joka kääntää pelin usealle alustalle tulee esimerkiksi yrityksille paljon halvemmaksi kuin ostaa jokaiselle mahdolliselle ympäristölle oma kehitysympäristönsä. Arvojen muuttaminen reaaliajassa, mallien vaihtaminen sekä uuden koodin lisääminen, kääntämättä peliä uudelleen on myös Unity-editorin suuri vahvuus. Useimmiten muissa pelimoottoreissa koko projekti on käännettävä uudelleen, jos sitä halutaan testata. Unityssä projektin käännösaika ei yleensä ylitä 30 sekuntia kun taas muissa pelimoottoreissa se yleensä vie minuutista kymmeniin, pelimoottorista riippuen. Peliin on myös sitä pelattaessa mahdollista tehdä muutoksia, joka auttaa esimerkiksi kun testataan erilaisia nopeuksia. Nämä muutokset eivät kuitenkaan ole pysyviä kun pelin ajamisen lopettaa. (Uurainen 2013.)

Komponenttimainen ohjelmointi on yksi Unityn haitoista, sillä Unity ei noudata olio-ohjelmoinnin periaatteita. Muodostimia ei käytetä, ja komponenttimaisuus aiheuttaa sen, että luokkien periminen tuntuu vaikealta, koska useimmiten otetaan jokin komponentti käyttöön ja käytetään tämän julkisia metodeja. Tämä voi myös toisaalta olla hyödyllinen toiminto, sillä jokaisen peliobjektin ei luultavasti tarvitse toteuttaa fysiikkaa tai liikkua. Unity ei siis pakota jotakin luokkaa perimään fysiikkaominaisuutta vaikka jokin sen ylempi luokka tarvitsee sen. (Uurainen 2013.)

Oppimiseen liittyvä vaara Unityssä kuitenkin piilee siinä että pelimoottori sisältää paljon valmiita komponentteja. Se herättää kysymyksen tarvitseeko ohjelmoijan ohjelmoida mitään itse vai käyttääkö ainoastaan Unityn valmiita komponentteja. Aloittelevalle ohjelmoijalle tämä voi olla suurikin apu, mutta voi johtaa siihen, että asioiden syvempi tutkiminen jää vähemmälle. (Uurainen 2013.)

4 PELISUUNNITTELU

Pelisuunnittelu tarkoittaa ideoiden keksimistä ja myös näiden ideoiden tarkempaa määrittelyä ja tämän informaation välittämistä koodaajille. Pelin koodaamisen aloittaminen tai edes pelin julkaiseminen ei tarkoita suunnittelun loppua. Hyvänä esimerkkinä toimii pelitasapaino, jota voidaan jälkeenpäin korjailla. Pelisuunnittelijan täytyy erityisesti ottaa huomioon peliä suunnitellessaan, että jokaisen idean täytyy myös olla mahdollinen toteuttaa. Pelin laadun kannalta keskeinen elementti on pelisuunnittelu, koska graafisesti näyttävä peli ei ole hyvä jos sen pelaaminen ei ole miellyttävää. (Kellomäki 2012.)

4.1 Yleistä

Olennainen osa peliä on sen suunnitteleminen. Loogisen pelin rakentaminen on miltei mahdotonta ilman hyvää suunnitelmaa. Peliä suunniteltaessa on tärkeää miettiä pelin genre, ikäluokat, jolle se suunnataan, pelin ominaisuuksia, hahmoja ja jokin tarina. Yksi tärkeimmistä suunnittelun osa-alueista on suunnitelman dokumentointi, joka usein unohdetaan. Verkossa liikkuu monia tarinoita siitä, kuinka monet projektit ovat epäonnistuneet huonon dokumentaation tai sen päivityksen johdosta. Kehitysryhmällä täytyy olla hyvä dokumentaatio, jotta se voi tarkistaa jonkin asian sen ollessa epäselvää. (Uurainen 2013.)

4.2 Ideointi

Pelien ideoinnissa luovuus on epämääräinen käsite. Omaa peliä on vaikea ryhtyä suunnittelemaan tyhjästä, koska yleensä pelin ideointi rakentuu paljon helpommin jo olemassa olevan idean päälle. Esimerkiksi kahden vanhemman idean yhdistäminen on hyvä tapa luoda uusia peli-ideoita. Brainstorming on hyvä tapa luoda uusia ideoita peleihin, esimerkiksi listojen ja ajatuskarttojen avulla. Yksi yleisimmistä ideoista uusiin peleihin löytyy vanhoista peleistä. (Kellomäki 2012)

Kuva 6. Kuvankaappaus Angry Birds pelistä. (Rovio entertainment Ltd. n.d.)

Yksi parhaista esimerkeistä vanhan peli-idean käytöstä näkyy kuvasta 6. Angry Birds -pelin formaatti on ollut olemassa jo vuosia ennen kuin itse Angry Birds luotiin. Peli on toteutettu vain paljon paremmin kuin aikaisemmin olemassa olleet pelit, koska se on koukuttava siinä on mahdollisuus hankkia uusia ominaisuuksia, se on hauska ja helppo pelata ja se soveltuu hyvin niin aikuisen kuin myös lapsenkin pelattavaksi.

4.3 Yleiset kompastuskivet

Yksi yleisimmistä kompastuskivistä pelin suunnitteluvaiheessa on useimmiten se, että suunnittelijan suunnitelmia ei voida ohjelmointivaiheessa toteuttaa peliin. Pelisuunnittelusta on vaikea luetella minkäänlaisia objektiivisia kriteereitä. Suunnitteluvaiheessa on suotavaa pitää peli yksinkertaisena, sillä se on toimivaa, usein törmätään pelien ylisuunnitteluun sekä liikojen ominaisuuksien vaatimiseen. Simulaatiopeleissä on yleinen ongelma, että niitä suunnitellaan liikaa ja niihin yritetään saada mahtumaan liikaa ominaisuuksia. (Kellomäki 2012.)

Peliä kehitettäessä on vaikeata hahmottaa kuinka paljon jokaisen erillisen pelin osan kehittäminen vaatii aikaa. Pelimoottorin hankinnassa on otettava monia asioita huomioon, kuten ostetaanko koko paketti vai tietyt osat kuten fysiikka ja renderöinti ja muut työkalut toteutetaan itse pelimoottoriin. Tässä tapauksessa jäljelle jäisi muun muassa animaatioiden, käyttöliittymän, kenttäeditorin ja tapahtumien toteutus. (Uurainen 2013.)

4.4 **Mobiilialustan haasteet**

Mobiilialustalle luodut pelit ovat monesti liian monimutkaisia. Silloin pelin ohjauksen opettelemisessa usein menee kauan aikaa. Kun huomaa oppineensa kontrolloimaan hahmoa, tajutaan että pelaajan kontrolloiminen on huonosti toteutettua ja epäloogista. Yksi ja ainoa tapa mobiilipeleissä hahmon kontrolloimiseen on kosketusnäyttö, koska ei ole olemassa nappeja tai mitään muuta. Jotkut pelintekijät ovat alkaneet täyttää näyttöä erilaisilla näppäimillä puhelimessa. Mobiililaitteiden kosketusnäytön koko on hyvin rajallinen, joten erilaisten näppäimien asettelu voi usein olla hankalaa, sillä pelin käyttöliittymästä pitäisi saada tehtyä looginen. Kosketusnäyttöön asetettavien näppäimien ongelma on, että niihin on liian vaikea osua sormella tai ne peittävät liian ison osan näytöstä. (Uurainen 2013.)

4.5 **Oma peli-idea**

Opinnäytetyön peli-idea on yksinkertainen ylhäältäpäin kuvattu 2D-autopeli. Peliin luodaan autoon arcade-tyylinen ohjaaminen ja yksi rata, jossa otetaan aikaa, kun pelaaja ajaa rataa ympäri. Kentässä ajetaan rataa ympäri ja siihen yritetään saada mahdollisimman nopeata aikaa. Peliin yritetään myös luoda jonkinlainen huijauksenesto, esimerkiksi checkpointtien avulla. Tarkoituksena pelissäni on käyttää puhelimen ominaisuuksia auton ohjaamiseen kuten kiihdytysanturin käyttö kääntymiseen ja näkymättömät napit puhelimen reunoissa kaasun ja jarrun käyttämiseen.

5 PROJEKTIN TAVOITTEET

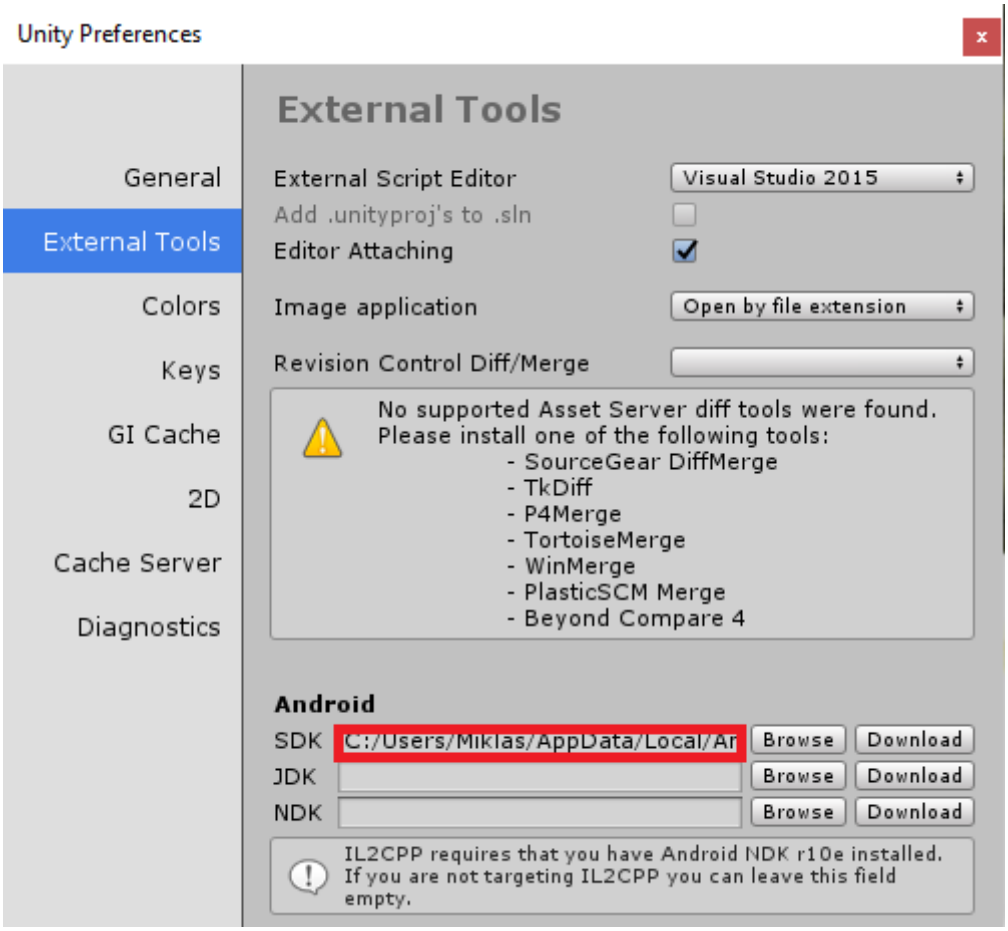
Opinnäytetyön yksi keskeisimmistä tavoitteesta on saada luotua yksinkertainen toimiva mobiilipeli käyttäen Unity-pelimoottoria. Yritän myös käyttää monia mobiilipeleihin keskeisiä erityispiirteitä. Hyviä esimerkkejä mobiilipelien erityispiirteistä ovat esimerkiksi näytön koko, käyttöliittymä, hahmon ohjaaminen kosketusnäytön avulla, puhelimen prosessorin teho. Itse tulen käyttämään kiihdytysanturia pelissäni auton kääntämiseen.

Tavoitteenani on myös oppia ohjelmoimaan toimiva mobiilipeli ja saada opinnäytetyön lukijalle myös selvä kuva mitä ohjelmointiin tarvitaan ja miten peli luodaan. Projektin aion luoda käyttäen Unity-pelimoottoria ja koodikielenäni toimii C#(C-Sharp)-ohjelmointikieli. Unity-pelimoottorin valinta opinnäytetyöhön oli yksinkertainen, sillä se on ainoa pelimoottori josta minulla on kokemusta, ja se on helposti opittavissa laajan manuaalin sekä hyvien opetusvideoiden takia. Työni aion luoda oppaan tavalla ja havainnoin asioita lukijalle koodinpätkillä sekä kuvilla, jotka tullaan yksityiskohtaisesti selittämään.

Aiheeni opinnäytetyöhöni olen valinnut sen perusteella, että mobiiliohjelmointi on nykyaikana tärkeää ja se kiinnostaa itseäni paljon. Projektin idea on yksinkertainen ylhäältä päin kuvattu 2D-autopeli. Syy pelin yksinkertaisuuteen on myös oma kokemattomuuteni mobiiliohjelmoinnissa.

6 ANDROIDIN KÄYTTÖÖNOTTO UNITY-PELIMOOTTORISSA

Androidin käyttöönotto Unity-pelimoottorissa on hyvinkin yksinkertaista. Ensimmäiseksi käyttäjän tulee ladata Android Studion asennuspaketti Android Studion omilta kotisivuilta. Asennuksen aikana käyttäjä voi valita pelkän Android SDK -paketin, muuta tässä projektissa ei tarvita. Android SDK -paketin polku on syytä ottaa muistiin.

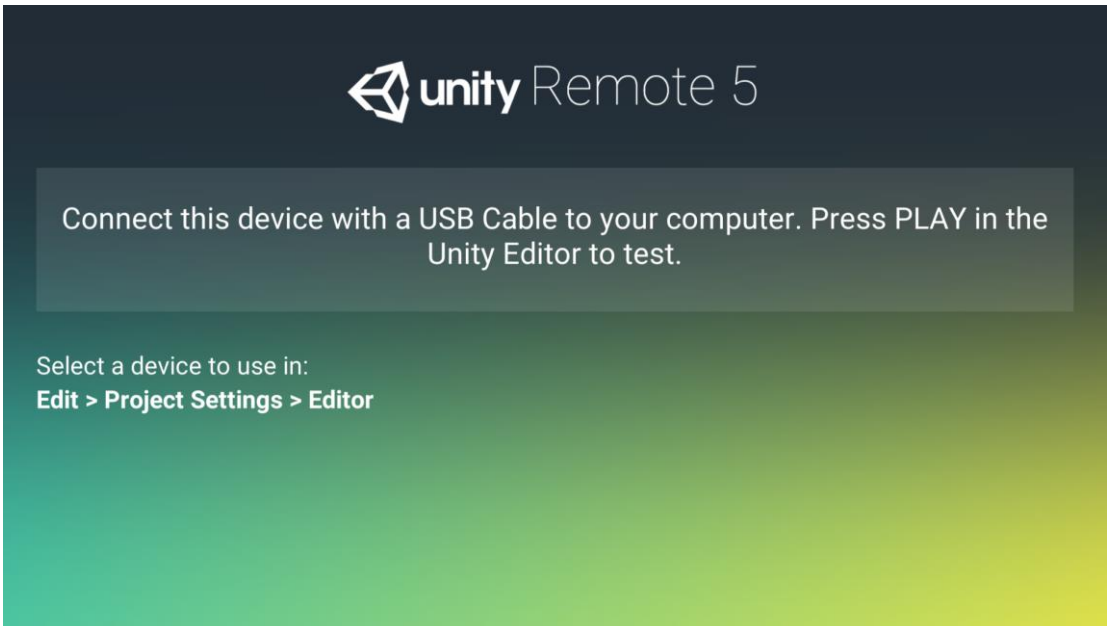


Kuva 7. Kuvankaappaus Unityn preferences ikkunasta.

Kuvasta 7 löydetään punaisella rajattu kohta, johon syötetään Android SDK -paketin polku omalla tietokoneella. Tähän ikkunaan käyttäjä pääsee Unityssä polkua Edit – Preferences – External tools. Oman Android -laitteen tulee olla liitettynä tietokoneen USB-porttiin, ja puhelin asetettuna käyttämään kehittäjätyökaluja, joista täytyy vielä asettaa päälle USB-viankorjaustila.

Tämän jälkeen omaan Android -laitteeseen tulee ladata Google Play-kaupasta Unity Editor -työkalu, jota tarvitaan jotta peliä voidaan testata Android -laitteella peliä kehittäessä. Tämän jälkeen vielä Unitystä täytyy

asettaa Unityn Remote -laitteeksi mikä tahansa Android -laite. Tähän paikkaan polku on Edit – Project Settings – Editor, josta kohtaan Device vaihdetaan Any Android Device.



Kuva 8. Kuvankaappaus Unity Editorista.

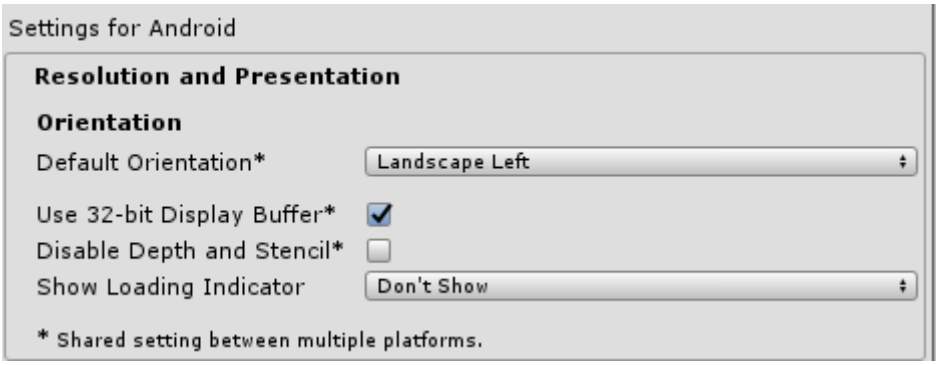
Kuten kuvassa 8 on mainittu, kaikkien ylläolevien vaiheiden jälkeen omaa peliä voidaan testata Android -laitteella vain painamalla Play-painiketta Unityssä. Tämän jälkeen pelin tulisi normaalisti käynnistyä Android-laitteella, jossa voidaan testata ominaisuuksia mitä on rakennettu toimimaan Android-laitteella.

7 MOBIILIPELIN TOTEUTTAMINEN

Mobiilipelien toteuttaminen tehdään suurelta osin samoilla periaatteilla kuin pelien toteuttaminen muille alustoille. Toteuttamisessa tulee ottaa huomioon vain mobiilipelien erityispiirteet, kuten kosketusnäyttö, kiihdytysanturi, prosessorin teho ja näytön rajoitettu koko. Työssäni mobiilipeli toteutetaan käyttäen Unity-pelimoottoria ja ohjelmointikielenä käytetään C# (C-sharp). Mobiilipeleissä on myös tärkeää muistaa, että pelien pitää olla helppokäyttöisiä, mutta myös omalla tavallaan koukuttavia. Tähän syy löytyy siitä, että mobiilipelejä pelataan tilanteissa joissa ei ole paljon aikaa ja pelaajan pitää pystyä ymmärtämään peli-idea hyvin nopeasti.

7.1 Ulkoasu ja käyttöliittymä

Pelin ulkoasua ja käyttöliittymää suunniteltaessa tulee ottaa huomioon nykyisten puhelimien näytön koko ja prosessorin teho. Esimerkkinä prosessorin tehon huomioimiseen voidaan käyttää jokaisen erillisen pelitilanteen sijoittaminen eri sceneihin. Näytön koko tulee ottaa huomioon jokaisessa pelissä eri tavoin, työssä luotavassa pelissä on näyttö pakotettu vaaka-asentoon, jotta peli näyttäisi järkevältä ja sen pelattavuus olisi hyvä. Työn peliin on pakotettu 1280 x 720 -resoluutiot.



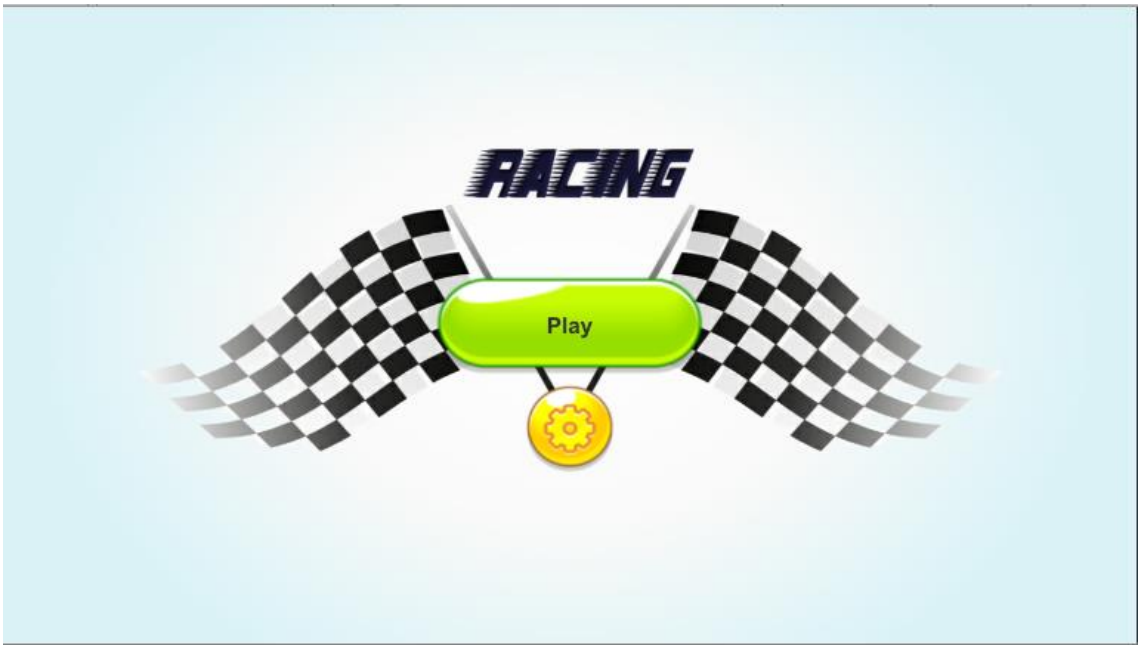
Kuva 9. Kuvankaappaus pelin Android asetuksista.

Kuvassa 9 näkyy Unitystä löytyvät näytön resoluution ja kuvan orientaation asetukset. Kohdassa Default Orientation on asetettu valinnaksi Landscape Left, joka asettaa pelin puhelimen näytölle vaaka-asentoon. Syynä orientaation pakottamiseen on pelaamisen helpottaminen puhelimen ollessa vaaka-asennossa ja se, että peliin pitäisi tehdä erikseen oma muotoilunsa myös pystyasentoon.

7.1.1 Main Menu

Pelin päävalikolle ja pelitilanteelle on mobiilipeleissä suotavaa luoda omat scenensä. Päävalikosta löytyy tarvittavat napit, kuten pelin käynnistäminen ja pelin omien asetusten muuttaminen. Päävalikon luominen Unityssä

on yksinkertaista. Ensimmäisenä täytyy scenelle luoda canvas-peliobjekti, johon esimerkiksi kaikki napit sekä taustakuva voidaan sijoittaa.

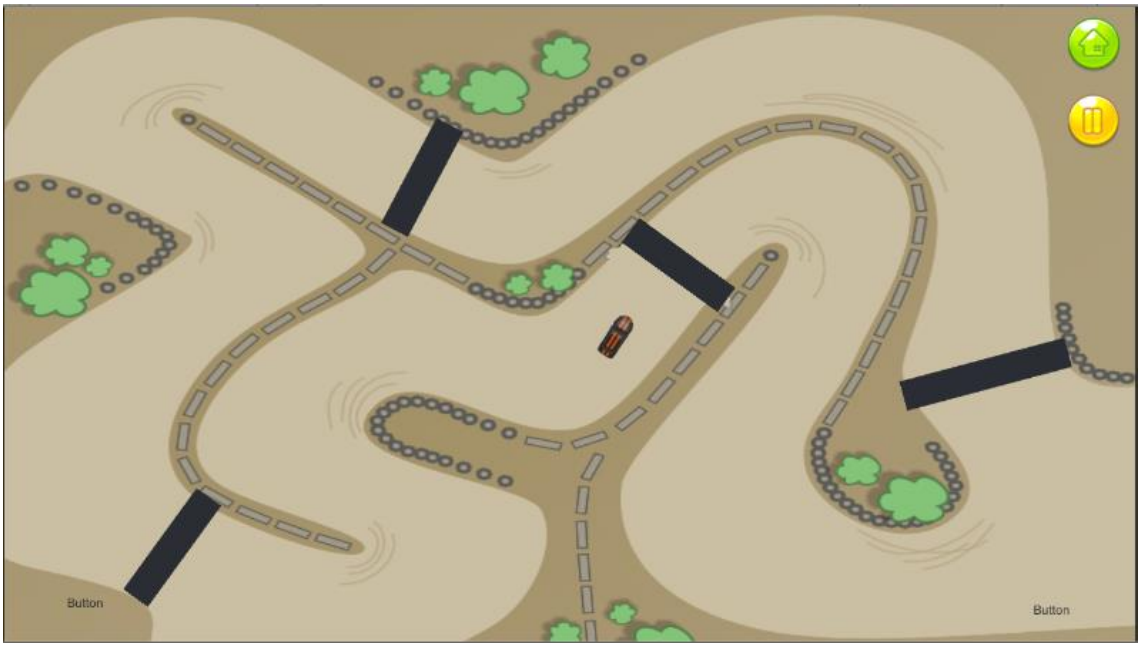


Kuva 10. Kuvankaappaus Main Menu -valikosta.

Kuvassa 10 päävalikolle on luotu oma scene nimeltä Main Menu ja tässä scenessä sijaitsevat esimerkiksi napit, joista pelaaja pääsee siirtymään pelin pelaamiseen tai asetuksiin.

7.1.2 Pelitilanne

Pelin pelaamisen vaiheeseen päävalikosta pääsee painamalla play-nappia. Alla kuvassa 11 olevan pelitilanteen ideana on ajaa rataa, josta otetaan aikaa. Mustat palkit kuvassa ovat niin sanottuja Checkpointteja, jotka peliin on luotu huijaamisen estoksi, jottei pelaaja voi ajaa maaliviivan yli esimerkiksi heti kahta kertaa. Kuvan oikeasta yläkulmasta löytyy myös kaksi nappia, ylemmästä napista avautuu pop-up -ikkuna, josta pääsee siirtymään takaisin päävalikkoon. Alemmasta napista pelin voi asettaa pause -tilaan.



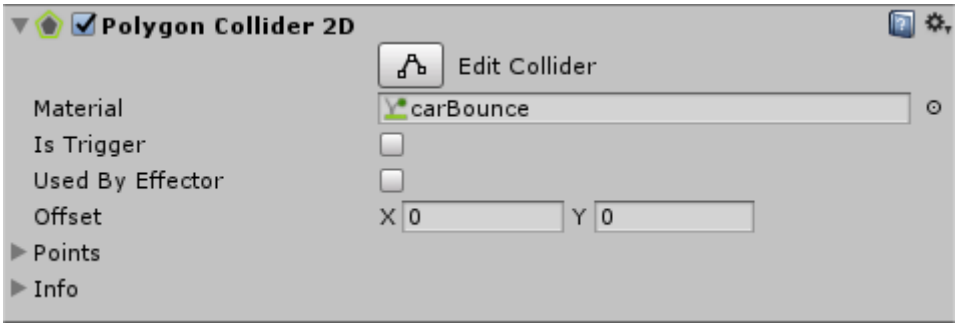
Kuva 11. Kuvankaappaus pelitilanteesta.

Kuvan 11 molempiin alakulmiin on sijoitettu näkymättömät napit, joista vasemman puoleisesta jarrutetaan autoa ja oikeanpuoleisesta autolla liikutaan eteenpäin. Auton kääntämistä hallitaan puhelimen kiihdytysanturin avulla. Näistä ominaisuuksista selitetään myöhemmissä kappaleissa tarkemmin.

Alempana kuvassa 12 on kuva pelikentän ajorataan asetetusta Polygon Colliderista, joka on asetettu sen takia, ettei pelaaja voi ajaa missään muualla kartalla kuin ajoradalla. Myös auton peliobjektiin täytyy olla liitettynä oma colliderinsa, jotta auto pystyy törmäämään seiniin.

Kuva 12. Kuvankaappaus Polygon Colliderista.

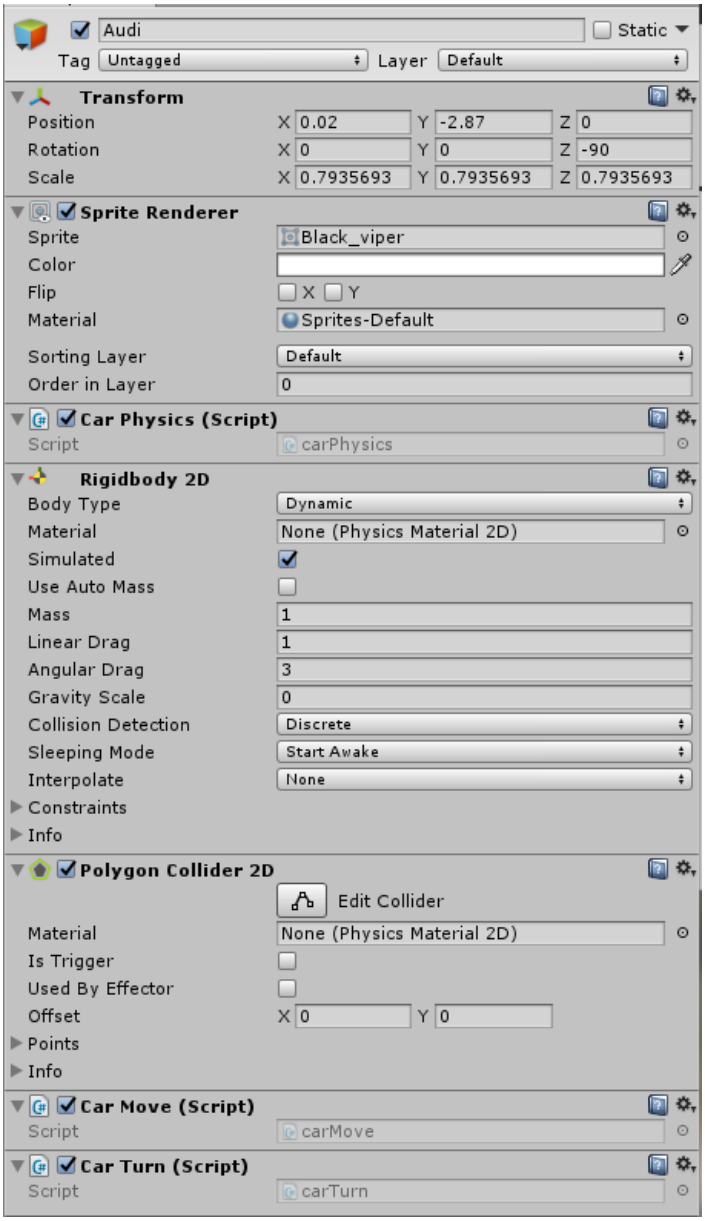
Polygon collideriin voidaan liittää jonkinlaisia fysiikoita, kuten työn pelissä on sille asetettu arcade-tyyliset fysiikat. Tässä tapauksessa siihen on liitetty Physics Material 2D -komponentti, jonka avulla saadaan auto ottamaan pienimuotoinen pomppu osuessaan seinään, jonka tarkoituksena on se, että pelaaja jää harvemmin seiniin jumiin. Physics Material -komponentti täytyy lisätä Assets-kansioon, jolloin sen voi liittää kuvassa 13 näkyvällä tavalla Polygon Colliderin Material -kohtaan. Työssä tämä fysiikkamateriaali on nimetty nimellä carBounce.



Kuva 13. Kuvankaappaus Polygon Colliderin ominaisuuksista.

7.2 Scriptien käyttö pelissä

Kaikki pelin sisäiset toiminnot on määritelty script-tiedostoissa, jotka ovat liitettynä omiin peliobjekteihinsa. Tässä projektissa tärkein peliobjekti on pelaajan ohjaama auto, jolla liikutaan radalla. Myös huomaamattomilla peliobjekteilla kuten seinillä saattaa olla omat script-tiedostonsa. Projektissa olen paloitellut script-tiedostot pienempiin osiin, jotta ne ovat helpompia esitellä opinnäytetyössä. Oman projektini kaikki script-tiedostot on ohjelmoitu käyttäen C# (C-sharp)-ohjelmointikieltä ja työkaluna script-tiedostojen muokkaamiseen Microsoft Visual Studio 2010 -ohjelmointiympäristöä. Script-tiedostojen käyttäminen peliobjekteissa on helppoa, koska ne voidaan paloitella ne niin pieniin palasiin ja tehdä niitä niin monta kuin halutaan. Tällä tavalla myös oikeiden script-tiedostojen löytäminen on helpompaa.



Kuva 14. Kuvankaappaus auton peliobjektin inspector-elementistä.

Kuvassa 14 näkyy auton peliobjektin inspector-elementti. Inspector-elementistä nähdään kaikki peliobjektiin liitetyt komponentit, muun muassa script-tiedostot. Inspector-elementin alareunasta löytyy myös nappi Add Component, josta pystytään peliobjektiin lisäämään joko Unityn valmiita komponentteja tai sitten itse kirjoitettuja script-tiedostoja. Peliobjekteja voidaan siirrellä eri tasojen välillä. Esimerkiksi normaalisti pelissä halutaan, että kartta tai kenttä on alimmalla tasolla ja pelihahmo kulkee tämän päällä. Script-tiedostojen kohdalla näkyvät kaikki siitä löytyvät julkiseksi asetetut muuttujat, joita voidaan reaaliaikaisesti muokata silloinkin kun peliä testataan, mutta testaamisen aikana muutetut arvot eivät pysy kun testaaminen lopetetaan.

7.2.1 CarPhysics.cs

Ensimmäisessä script-tiedostossa annetaan autolle eräitä realistisia fysiikoita. Esimerkiksi jos pelaajaa yrittää kääntää autoa liian nopeasti, auto joutuu sivuluisuun, josta peliin saadaan todenmukaisuutta. Auton kääntymiseen on myös luotu ominaisuus, jonka avulla auto ei pysty kääntymään paikoillaan mikä luo peliin realistisen tunteen.

```
float driftFactor = driftFactorSticky;
    if(RightVelocity().magnitude > maxStickyVelocity)
    {
        driftFactor = driftFactorSlippy;
    }
```

Esimerkki 2. If-lause

Esimerkissä 2 tarkastellaan liikkuuko auto liian nopeasti kun sillä käännytään, jonka jälkeen auto joutuu sivuluisuun.

7.2.2 CarMove.cs

CarMove.cs-script-tiedostossa annetaan näytön oikeaan ja vasempaan reunaan sijoitetuille napeille ominaisuudet, joilla kutsutaan Accelerate()- ja Brakes()-funktioita. Napin painallukset toimivat puhelimen kosketusnäytöllä samalla tavalla kuin niitä painettaisiin tietokoneella hiiren painikkeella.

```
public void Accelerate()
{
    Rigidbody2D rb = GetComponent<Rigidbody2D>();
    rb.AddForce(transform.up * speedForce);
}

public void Brakes()
{
    Rigidbody2D rb = GetComponent<Rigidbody2D>();
    rb.AddForce(transform.up * -speedForce / 2);
}
```

Esimerkki 3. Accelerate()- ja Brakes()-funktiot.

Esimerkissä 3 näkyy funktiot, joilla auto saadaan liikkumaan eteen- ja taaksepäin. Ainoa ero funktioissa on, että Brakes()-funktiossa speedForce-muuttuja on negatiivinen luku, joka saa auton liikkumaan taaksepäin.

7.2.3 CarTurn.cs

Kiihdytysanturin käyttö on mobiilipeleissä yksi tärkeimmistä erityispiirteistä, koska sitä joudutaan usein käyttämään johonkin mobiilipelin ominaisuuteen. Tässä script-tiedostossa luodaan funktio AccelerometerMove(), jossa haetaan puhelimen kiihdytysanturin lukema. Tämän jälkeen kiihdytysanturin lukemaa käytetään yhtälössä, jolla auto saadaan kääntymään. Kun funktio on luotu toimivaksi, sitä kutsutaan Update()-funktiossa.

```
void AccelerometerMove()
{
    float accMeter = Input.acceleration.x;
    //Tulostaa kiihdytysanturin lukeman konsoliin
    Debug.Log("X = " + accMeter);

    Rigidbody2D rb = GetComponent<Rigidbody2D>();
    float tf = Mathf.Lerp(0, torqueForce, rb.velocity.magnitude / 2);
    rb.angularVelocity = Input.acceleration.x * tf;

}
```

Esimerkki 4. AccelerometerMove()-funktio.

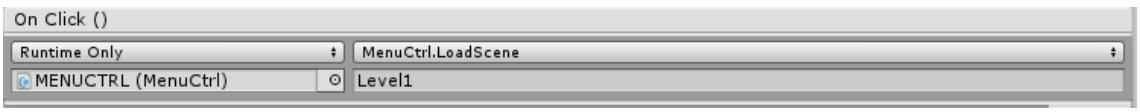
Esimerkissä 4 haetaan puhelimen kiihdytysanturin arvo accMeter-muuttujaan, jonka jälkeen se tulostetaan konsolissa. Alemmassa kaavassa luodaan autolle kääntyminen näppäimien sijaan puhelimen kiihdytysanturin avulla. Funktiota täytyy myös kutsua Update()-funktion sisällä, jotta se toimii.

7.2.4 MenuCtrl.cs

MenuCtrl-script-tiedoston tarkoituksena on hallita pääosin menuvalikon nappien painalluksia sekä sitä, että nappia painettaessa esimerkiksi vaihdetaan sceneä. Varsinkin mobiilipeleissä on tärkeää sijoittaa esimerkiksi menu ja pelikentät eri sceneihin, jotta puhelimen ei tarvitse käsitellä niin paljon dataa kerrallaan.

```
public void LoadScene(string sceneName)
{
    SceneManager.LoadScene(sceneName);
}
```

Esimerkki 5. LoadScene()-funktio.



Kuva 15. Kuvankaappaus OnClick()-funktiosta

Esimerkissä 5 esiintyy LoadScene()-funktio, joka liitetään napin painallukseen. Se että funktion sulkujen sisällä on muuttujan nimi tarkoittaa, että funktio palauttaa jonkun arvon. Tässä tapauksessa arvolla tarkoitetaan scenen nimeä ja se täytyy antaa jokaiselle napille erikseen. Kuvassa 15 näkyy napin oma OnClick()-funktio, johon on liitetty peliobjekti. Tämä peliobjekti sisältää MenuCtrl-script-tiedoston ja sieltä on valittuna LoadScene()-funktio, johon on syötetty arvo Level1. Peliä pelatessa pelin yläkulmasta löytyy myös Home-nappi, joka käyttää samaa LoadScene()-funktiota, jolla pääsee pelin Main Menu -ikkunaan.

Samasta script-tiedostosta löytyy myös Pause()-funktio, jonka avulla pelaaja voi pelaamisen aikana pysäyttää pelin painamalla pause-näppäintä. Painamalla tätä samaa nappia uudelleen peli jatkuu kohdasta johon pelaaja sen pysäytti.

```
public void Pause()
{
    if(Time.timeScale == 1)
    {
        Time.timeScale = 0;
    }
    else if (Time.timeScale == 0)
    {
        Time.timeScale = 1;
    }
}
```

Esimerkki 6. Pause()-funktio.

Tässä if-lauseessa tarkastellaan onko peli käynnissä vai pysäytetyssä tilassa. Kun Pause()-funktio liitetään pause-napin OnClick()-funktioon niin se ajaa if-lauseen läpi nappia painettaessa ja tarkastaa onko peli käynnissä, jos peli on käynnissä tämä toiminto pysäyttää sen niin pitkäksi aikaa kun pelaaja painaa samaa nappia uudestaan.

7.2.5 UiCtrl.cs

UiCtrl.cs-script-tiedosto hallitsee pelin aikana tapahtuvia asioita, kuten esimerkiksi home-napin painamista. Nappien sijoittelu on erityisen tärkeää mobiilipeleissä puhelimen pienen näytön takia, jotta napit ovat helposti huomattavissa ja painettavissa eivätkä myöskään peitä liian isoa osaa näytöstä. Mobiilipeleissä ei juurikaan tarvitse muuta miettiä nappien suhteen, koska kosketusnäytön painallukset toimivat periaatteessa samalla logiikalla kuin esimerkiksi tietokoneella hiiren painikkeet. Home-nappia painamalla peli asettuu pause -tilaan ja keskelle näyttöä ilmestyy ponnahdusikkuna, jossa varmistetaan haluaako pelaaja varmasti siirtyä pelin päävalikkoon. Ponnahdusikkuna on luotu mahdollisten vahinkoklikkauksien takia.

```
Canvas pauseCanvas;
void Start () {
    pauseCanvas = GetComponent<Canvas>();
    pauseCanvas.enabled = false;
}

public void HomeMenu()
{
    Time.timeScale = 0;
    pauseCanvas.enabled = true;
}

public void Cancel()
{
    Time.timeScale = 1;
    pauseCanvas.enabled = false;
}

public void Accept(string sceneName)
{
    SceneManager.LoadScene(sceneName);
}
```

Esimerkki 7. UiCtrl.cs script-tiedosto.

Kuva 16. Kuvankaappaus pelin pause-tilanteesta.

Ensimmäisenä luodaan pauseCanvas-muuttuja johon haetaan Start()-funktiossa arvo ja asetetaan se pois päältä, kuten yllä olevassa esimerkissä 7 on tehty. Tämän jälkeen luodaan HomeMenu()-funktio, joka asetetaan Home-näppäimelle pelin yläkulmaan, jossa peli asetetaan pause -tilaan ja tuodaan pop-up-ikkuna nimeltä pauseCanvas näkyviin. Yllä olevasta kuvasta

16 keskeltä ruutua löydetään pop-up-ikkunan. Pop-up-ikkunassa on kaksi nappulaa ja alemmat Cancel()- ja Accept()-funktiot on asetettu näille, kuten esimerkissä 7 on esitetty.

7.2.6 Checkpoint.cs ja CheckpointManager.cs

Checkpoint.cs- ja CheckpointManager.cs-script-tiedostot hallitsevat pelissä checkpointteja. Kun pelaaja ajaa vihreän checkpointin läpi muuttuu seuraava checkpoint mustasta vihreäksi, joten pelaajan täytyy ajaa niiden läpi oikeassa järjestyksessä. Checkpoint-systeemi on luotu peliin huijauksen estoksi, jotta pelaaja ei esimerkiksi pysty ajamaan maaliviivan yli heti kahta kertaa.

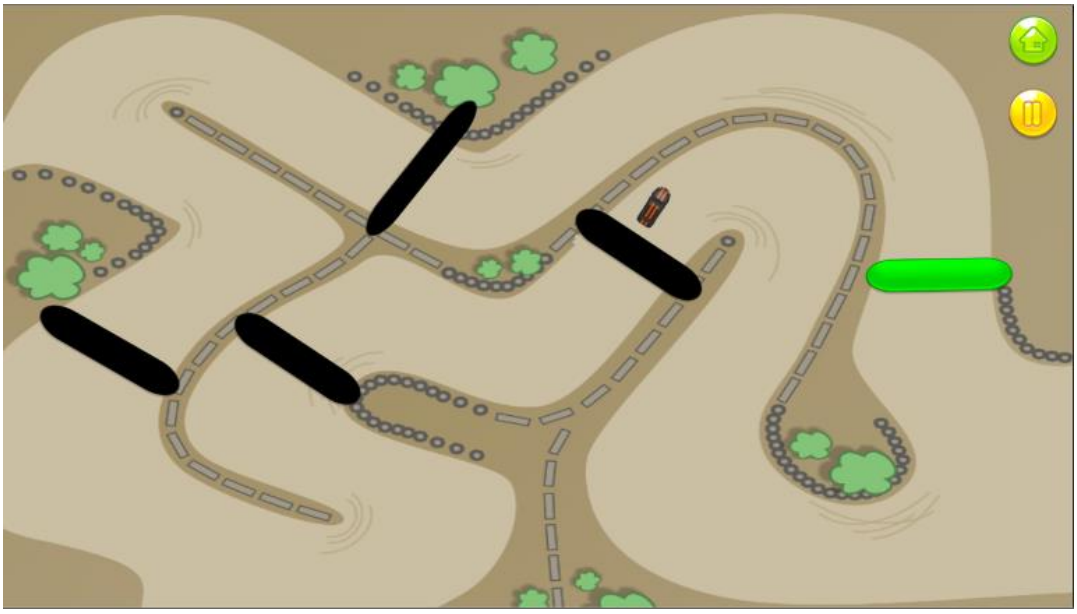
```
void Start()
{
    laps = 0;
    checkPoints.Clear();

    foreach (Transform cp in transform)
    {
        checkPoints.Add(cp.gameObject);
    }
    currentCheckPoint = checkPoints[1];
}

void Update()
{
    foreach (GameObject cp in checkPoints)
    {
        if (cp.GetInstanceID().Equals(currentCheckPoint.GetInstanceID()))
        {
            cp.GetComponent<SpriteRenderer>().color = Color.green;
        }
        else
        {
            cp.GetComponent<SpriteRenderer>().color = Color.black;
        }
    }
}
```

Esimerkki 8. checkpointManager.cs script-tiedosto.

CheckpointManager script -tiedostossa luodaan kaikille checkpointeille lista, jonka avulla niitä on helpompi hallita. Update()-funktion sisällä tarkastetaan minkä checkpointin läpi pelaajan pitää ajaa seuraavaksi ja muuttaa tämän värin vihreäksi alla olevan kuvan 17 mukaisesti.



Kuva 17. Kuvankaappaus checkpointtien toiminnasta.

Jokaiselle checkpointille tarvitaan myös script-tiedosto, jonka avulla pystytään havaitsemaan kun pelaajan ohjaama auto kulkee checkpointien lävitse. Tämän script-tiedoston nimi on checkPoint.cs, ja se täytyy olla liitettynä jokaiseen yksittäiseen checkpointiin.

```
  void OnTriggerEnter2D(Collider2D other)
  {
    if (other.gameObject.tag == "Player" && this.gameObject.Get-
InstanceID().Equals(checkPointManager.currentCheckPoint.Get-
InstanceID()))
    {
      if (transform.GetSiblingIndex() == 0)
      {
        checkPointManager.laps++;
        checkPointManager.currentCheckPoint =
            checkPointManager.checkPoints[1];
      }
   else if (transform.parent.childCount == transform.GetSiblingIndex() +1)
      {
        checkPointManager.currentCheckPoint = checkPointMan-
ager.checkPoints[0];
      }
      else
      {
        checkPointManager.currentCheckPoint =
checkPointManager.checkPoints[transform.GetSiblingIndex() + 1];
      }
    }
  }
```

Esimerkki 9. Checkpointien törmäyksentunnistus.

Jokaiseen yksittäiseen checkpointiin on liitettynä oma colliderinsa. Tämä ei tarkoita sitä, että checkpointeihin voi törmätä. Colliderista löytyy toiminto, jolla saadaan peliobjekti toimimaan niin sanottuna laukaisijana (On Trigger). Tämä tarkoittaa siis sitä, että voidaan havaita milloin jokin tietty peliobjekti törmää checkpointiin. Kuten esimerkissä 9 on esitelty, ensimmäiseksi tarkistetaan onko Player-peliobjekti törmännyt checkpointien peliobjekteihin. Kun pelaaja on kiertänyt jokaisen checkpointin, lisätään yksi kierros.

8 YHTEENVETO

Kokonaisuudessaan mobiilipelin luominen onnistui hyvin, vaikka muutamia ominaisuuksia pelistä jäi pois. Pelissä olisi voinut olla vielä muutamia ominaisuuksia lisää kuten kunnon asetus-valikko, autojen vaihto sekä niiden eri ominaisuudet ja toinen mahdollinen kenttä. Projektin pääasia, eli mobiilipelin luominen onnistui tarkoituksenmukaisesti ja projektissa päästiin testaamaan mobiilipelien erityispiirteitä, kuten puhelimen kiihdytysanturin käyttöä. Ajan puutteen vuoksi projektia ei vielä saatu julkaisukuntoon, koska joitain oleellisia ominaisuuksia jäi pois.

Projektin toisena tarkoituksena oli myös antaa opinnäytetyön lukijalle selvä kuva siitä, miten luodaan toimiva mobiilipeli käyttäen Unityä. Siinä projektissa onnistuttiin hyvin. Esimerkiksi koodinpätkien selitykset on kirjoitettu hyvin yksityiskohtaisesti, jotta aloittelevakin pelinkehittäjä saa niistä selkoa. Script-tiedostot ovat pilkottu pieniin osiin, joka auttoi asioiden esille tuomista opinnäytetyössä.

Unityllä mobiilipelin ohjelmoiminen osoittautui helpoksi, vaikka kokemusta tästä alustasta on vain muutaman oppitunnin ja yhden pienen projektin verran. Apunani projektin toteuttamisessa käytin Unityn laajaa manuaalia verkossa, jossa jokaisen luokan toiminnot ovat määritelty hyvin tarkasti. Suuri apu käytännön osuutta tehdessä oli laaja valikoima erilaisia opetusvideoita, joiden avulla suurimman osan yksittäisistä ominaisuuksista peliin toteutin.

LÄHTEET

Paavilainen, J. (2009). Mobiilipelaaminen. Haettu 17.1.2017 osoitteesta https://pelitieto.net/case-mobiilipelaaminen/

Unity Technologies. Unity User Manual (5.5). Haettu 18.1.2017 osoitteesta https://docs.unity3d.com/Manual/UnityManual.html

Husso, S. (2013). 2D-Peliohjelmointi Unityä käyttäen. Opinnäytetyö. Tietotekniikan koulutusohjelma. Kymenlaakson ammattikorkeakoulu. Haettu 19.1.2017 osoitteesta https://www.theseus.fi/bitstream/handle/10024/57338/Husso_Sami.pdf?sequence=1

Pelkonen, J. (2015). Matopeli Nokian puhelimella. Haettu 20.1.2017 osoitteesta http://yle.fi/uutiset/3-7970392

Rovio Entertainment Ltd. (n.d.). Kuvankaappaus Angry Birds pelistä. Haettu 27.1.2017 osoitteesta https://lh3.googleusercontent.com/1hlSoe_nPf84AbMhrlbPkq6WAYLfJ65kEeZQXVB_g9mfUCuE18DZz2UxKX6SD0af0NQw=h900

Päiveröinen, H. (2014). Unity- ja XNA/MonoGame-pelimoottorien vertailu 2D-pelien toteutuksessa. Pro gradu-tutkielma. Tietojenkäsittelytiede. Tietojenkäsittelytieteen oppilaitos. Haettu 26.1.2017 osoitteesta https://helda.helsinki.fi/bitstream/handle/10138/45442/Hannu%20Paiveroinen%20%20Unity%20ja%20XNAMonoGamejrjestelmien%20ominaisuuksien%20vertailua%202Dpelien%20toteutuksessa.pdf?sequence=2

Kurtti, S. (2015). Näppäimiltä kosketukselle – tabletilla pelaaminen. Pro gradu –tutkielma. Informaatiotutkimus ja interaktiivinen media. Tampereen yliopisto. Haettu 24.1.2017 osoitteesta https://tampub.uta.fi/bitstream/handle/10024/97494/GRADU-1435152307.pdf?sequence=1

Tukiainen, T. (2013). Mobiilipelit ja pelimoottorit. Pro gradu – tutkielma. Tietojenkäsittelytiede. Helsingin yliopisto. Haettu 25.1.2017 osoitteesta https://helda.helsinki.fi/bitstream/handle/10138/42050/20131106graduv3.pdf?sequence=2

Kellomäki, T (2012). Pelisuunnittelu. Haettu 27.1.2017 osoitteesta http://www.cs.tut.fi/~peliohj/peliohj2012-03-Pelisuunnittelu-webversio.pdf

Uurainen, M. (2013). Unity mobiilipelien kehityksessä. Opinnäytetyö. Tietotekniikka. Kymenlaakson ammattikorkeakoulu. Haettu 28.1.2017 osoitteesta http://www.theseus.fi/bitstream/handle/10024/56873/Uurainen_Mika.pdf?sequence=1