Mobiilipelin luominen Unity 3D:llä

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tietotekniikka

Riihimäki, kevät 2018

Riku Pennanen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietotekniikka
Riihimäki

Tekijä Riku Pennanen **Vuosi** 2018

Työn nimi Mobiilipelin luominen Unity 3D:llä

Työn ohjaaja/t Petri Kuittinen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli luoda yksinkertainen ja helposti opittava, Endless Runner -tyyppinen peli Android-käyttöjärjestelmälle. Pelin visuaaliset osat tehtiin käyttäen Unityä. Ohjelmointia vaativiin osiin käytössä oli Microsoft Visual Studio 2013 –ohjelma ja ohjelmointikielenä käytössä oli C#.

Opinnäytetyössä kerrotaan myös tietoa Unitystä ja mobiilipeleistä. Mobiilipelien osassa käydään läpi niiden historiaa sekä mitä mobiilipelit ovat tänä päivänä. Opinnäytetyössä kerrotaan myös Unityn pelimoottorissa esiintyvistä yleisimmistä käsitteistä ja termeistä, jotka olisi hyvä tietää aloittelevana pelinkehittäjänä. Tämän lisäksi käydään teoriaa läpi mobiilipelien eduista ja mitä kaikkea olisi siitä ottaa huomioon mobiilipelejä kehittäessä.

Lopuksi käytännön osassa selvitetään mobiilipelin luomiseen Unityllä tarvittavia ohjelmia ja kuinka ohjelmalla luotuja pelejä voi muuntaa Android-alustalle sopiviksi.

Mobiilipelin luominen Unity-pelimoottoria käyttäen oli mielestäni varsin onnistunut valinta. Unity osoittautui sekä mielenkiintoiseksi että yksinkertaiseksi ohjelmaksi käyttää ja se soveltuu hyvin niin aloittelijalle kuin vähän kokeneemmallekin pelinkehittäjälle. Projektin lopputuloksena valmistui yksinkertainen ja helppokäyttöinen Endless Runner -tyylinen mobiilipeli.

Avainsanat Android, C#, mobiilipeli, Unity

Sivut 29 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information technology
Riihimäki

Author Riku Pennanen **Year** 2018

Subject Creating a mobile game on Unity 3D

Supervisors Petri Kuittinen

ABSTRACT

The goal of the project was to create a simple and easy-to-learn Endless Runner -style game for an Android device. The game was created using Unity and the script files were made and modified using Microsoft Visual Studio 2013. The programming language was C# (C-Sharp).

In the theory part of the thesis there is information about Unity and mobile games. The mobile games part includes information on the history of the games and some information about what mobile games are today. I also describe about the most common concepts and terms that you should know in the Unity game engine if you are a novice game developer. In addition, I go through the theories of mobile gaming and what is good to know about developing mobile games.

Finally, in the practical part, the reader can see how to create a mobile game with Unity and how the games created can be converted to an Android platform.

Creating a mobile game with the Unity game engine was in my opinion, a very successful choice. Unity proved to be both interesting and simple to use, and it is well suited for beginners as well as for a slightly more experienced game developer. As a result of the project, a simple and easy-to-use Endless runner -style mobile game was created.

Keywords Android, C#, mobile game, Unity

Pages 29 pages

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on kertoa lukijalle tietoa mobiilipeleistä ja siitä, miten niiden kehitys on muuttunut vuosien varrella. Teoriaosiossa kerron myös hieman mobiilipelien historiaa ja miltä sen nykytilanne vaikuttaa. Lopuksi esittelen vaiheita oman 3D -mobiilipelini luomisesta, jonka valmistelin käyttäen Unityn tarjoamaa pelimoottoria.

Opinnäytetyön tarkoituksena on luoda yksinkertainen ja käyttäjäystävällinen, mobiilipeli Unity-pelimoottoria käyttäen. Pelin käyttöjärjestelmäksi valikoitui Android, sillä pelitestaukset onnistuvat sillä vaivatta omaa puhelinta testilaitteena käyttäen. Pelimoottorin valinnassa vaihtoehtoina itselle oli joko Unity tai Unreal Engine, mutta tällä kertaa halusin valita Unityn. Molemmista pelimoottoreista on kokemusta, sillä niitä käytettiin jonkin verran koulun tarjoamilla kursseilla. Ohjelmointikieleksi valikoitui lopulta C#. C-Sharp ohjelmointikieleen päätyminen oli mielestäni luonteva valinta, sillä se on mielestäni melko helposti ymmärrettävä ja selkeä ohjelmointikieli.

Tavoitteenani on saada pelistä toimiva ja helppokäyttöinen. Tarkoituksena on luoda peliin yksi taso, joka sisältää muutamia erilaisia esteitä sekä aloitussivun, josta pääsee etenemään peliin tai poistumaan pelistä. Lisäksi peliin olisi tarkoitus lisätä kerättäviä kolikoita sekä mahdollisesti myös muita kerättäviä esineitä, joista on apua pelin edetessä. Tällaisia esineitä voisi olla esimerkiksi jonkinlainen magneetti, joka vetää kolikoita puoleensa sekä pelihahmon kuolemattomuus. Kuolemattomuuden aikana pelihahmolla olisi mahdollista juosta eteen tulevien esteiden läpi. Pelin taustalla on tarkoitus soida taustamusiikki ja pelin olisi mahdollista pistää tauolle pysäytysvalikon avulla.

Tästä opinnäytetyöstä tulee hyvä tietoisku pelisuunnittelusta kiinnostuneelle aloittelijalle. Kerron myös paljon informaatiota mobiilipelien kehityksestä ja siitä, mitä kaikkea niiden luomisessa pitäisi ottaa huomioon. Erityisesti Unityn käyttöön ja sen lukuisiin ominaisuuksiin perehdytään tarkemmin. Unityn käytöstä kerron myös tarjoamalla esimerkkejä suoraan omasta mobiilipelistäni.

2 MOBIILIPELAAMINEN

Tässä luvussa tarkoituksena on kertoa lukijalle tietoa mobiilipelien historiasta ja niiden synnystä aina tämän päivän peleihin. Käyn läpi pelien kehitystä ja miten pelit ovat muuttuneet vuosien varrella. Mobiilipelaaminen tarkoittaa kannettavalla elektronisella laitteella, esimerkiksi tabletilla tai älypuhelimella, tapahtuvaa pelaamista. Nykyään mobiilipelaaminen on erittäin suosittua ja se on yksi nopeimmin kasvaneista peliteollisuuden osa-alueista. (Paavilainen, 2009.) Suuren suosionsa mobiilipelit ovat saavuttaneet helpon saatavuutensa takia, sekä paikkariippumattomuutensa ansiosta. Mobiilipelejä pystyy siis pelaamaan oikeastaan missä tahansa, kunhan vain käytössä on jokin kannettava mobiililaite.

Peliteollisuudessa mobiilipelit ovat nostaneet päätään todella kovaa vauhtia ja sillä onkin yksi alan suurimmista markkina-arvoista tällä hetkellä. Mobiilipelien valmistus ei myöskään vaadi kovinkaan suuria kehittäjäjoukkoja, jonka takia niitä kehitetään pienemmissä, muutamien henkilöiden ryhmissä.

2.1 Mobiilipelien historia

Mobiilipelien historia alkaa loogisesti samaan aikaan, kuin kannettavien mobiililaitteidenkin historia. Ensimmäiset kannettavat elektroniset mobiililaitteet tulivat markkinoille jo 1970-luvulla, joten voidaan sanoa, ettei mobiilipelaaminen ole mikään aivan uusi juttu.

Maailman ensimmäinen markkinoille asti saapunut kannettava pelilaite oli Mattel Auto Race (Mattel, 1976). Laite sisälsi vain yhden pelin ja siinä tarkoituksena oli ohjata pientä valopilkkua kolmikaistaisella tiellä ja pyrkiä väistelemään vastaantulijoita. Vuotta myöhemmin Mattel julkaisi seuraavan pelinsä, Mattel Football, josta tuli hieman yllättäen suuri myyntimenestys. Peli simuloi amerikkalaista jalkapalloa.

Seuraava kannettavien pelilaitteiden suurempi kehitysvaihe saatiin markkinoille vuonna 1979, kun Milton Bradley Company julkaisi Microvision-nimisen pelilaitteen. Laite oli ensimmäinen kannettava pelilaite, joka sisälsi useamman pelin. Microvision oli kuitenkin melko epävakaa laite, ja se oli erityisen herkkä staattiselle sähkölle. Tämän vuoksi sen tuotanto jouduttiin lopettamaan jo vuonna 1981.

Vuonna 1980 Nintendo tuli mukaan kannettavien pelilaitteiden markkinoille julkaisemalla Game & Watch -laitteen. Niitä valmistettiin ajan saatossa 59 erilaista peliä myytäväksi, joiden lisäksi yksi malli oli saatavilla vain kilpailupalkintona, jolloin yhteensä laitteita valmistettiin 60 erilaista. Game & Watch -laitteissa oli yleensä joko yksi tai kaksi näyttöä sekä D-Pad ohjain. Ne sisälsivät yhden pelin, jossa oli muutama eri vaikeusaste. Lisäksi laitteessa oli yleensä kello sekä herätysominaisuus. Game & Watch -laitteista tuli huippusuosittuja ja niitä on myyty vuosien saatossa yli 43 miljoonaa kappaleita maailmanlaajuisesti. Myös laitteeseen suunnitellusta D-Pad ohjaimesta tuli myöhemmin standardi peliteollisuuden käyttämässä ohjaintyylissä.

Vajaa kymmenen vuotta Game & Watch -pelilaitteiden julkaisusta – tarkalleen ottaen vuonna 1989 – Nintendo julkaisi uuden käsikonsolin, joka kantoi nimeä ”Game Boy”. Siitä tuli maailman suosituin kannettava pelilaite ja sitä sekä sen myöhempiä versioita (Game Boy Color, 1998 ja Game Boy Advance, 2001) on myyty yli 200 miljoonaa kappaletta ympäri maailmaa. Alkuperäinen Game Boy -pelilaite sisälsi ”Tetris” nimisen pulmapelin, jonka avulla laitteesta tulikin niin suosittu. Kuvassa 1 on kuvattuna Nintendon Game Boy sekä kaksi siihen liitettävää peliä.

Kuva 1. Nintendo Game Boy (1989).

Matkapuhelimet mullistivat mobiilipelaamista huimasti. Vuonna 1997 Nokia julkaisi 6110-matkapuhelimen, joka sisälsi huippusuositun matopelin. Matopelistä tuli Nokian klassikkopeli ja se kuuluikin lähes jokaiseen Nokian puhelimeen. (Paavilainen, 2009.)

Muun muassa käsikonsoleistaan ja peleistään tunnettu Nintendo julkaisi myös vuonna 2004 Nintendo DS nimellä kulkeneen käsikonsolin. Se on simpukkamallinen konsoli ja eroaa muista kannettavista pelilaitteista kahden näyttönsä ansiosta, joista alempi toimii kosketusnäyttönä. Nintendo DS tukee myös langatonta Wi-Fi -yhteyttä ja siinä on sisäänrakennettu mikrofoni. Laitteesta julkaistiin päivitetty versio, Nintendo DS Lite, maaliskuussa 2006. Lite on alkuperäistä versiota kevyempi ja sen näytöt ovat kirkkaammat kuin edeltäjässään. Nintendo DS on ensimmäinen Nintendon valmistama konsoli, jonka saatavuudessa on ollut katkoja liian suuren kysynnän vuoksi. Lokakuuhun 2008 mennessä Nintendo DS- ja DS Lite -konsoleita oli myyty maailmanlaajuisesty yli 84 miljoonaa kappaletta. (Wikipedia, 2018)

2.2 **Mobiilipelit nykyaikana**

Mobiilipelit ovat kehittyneet huomattavasti 2000-luvulle tultaessa. Nykypäivänä mobiilipelien markkinat ovat niin suuret, että peliteollisuus on kasvanut yhdeksi suurimmista markkina-aloista. Mobiilipelejä kehitetään peliteollisuuden suurissa yrityksistä sekä pienemmissä ryhmissä. Oikeastaan kuka tahansa voi nykypäivänä luoda oman mobiilipelinsä kehittyneen teknologian ja tietotaidon ansiosta. Tämän takia mobiilipelejä valmistuu useita päivässä ja niitä pystyy lataamaan esimerkiksi älypuhelimiin ja tabletteihin helposti erilaisten sivustojen ja ohjelmien kautta.

Laitteiden kehityksen myötä myös pelien grafiikat sekä muut ominaisuudet ovat parantuneet. Nykyään mobiilipelejä on mahdollista pelata internet yhteyden omaavissa laitteissa moninpelinä muita pelaajia vastaan tai muiden pelaajien kanssa. Tämän lisäksi uusimmissa mobiilipeleissä käytetään apuna myös lisättyä todellisuutta, eli augmented realityä (lyhennettynä AR). Lisätty todellisuus on tietokonegrafiikalla tuotettuja elementtejä joita käyttäjä voi nähdä esimerkiksi puhelimen kameran kautta ilmestyvän peliin mukaan. Lisättyä todellisuutta pelissä voi olla esimerkiksi, jokin teksti tai hahmo joka näkyy kameran kuvassa, vaikka todellisuudessa sitä ei kameran linssin edessä oikeasti olisikaan. Yksi suosituimmista lisättyyn todellisuuteen perustuvista mobiilipeleistä maailmalla tällä hetkellä on Niantic Labsin heinäkuussa vuonna 2016 julkaisema Pokemon GO. Peli on valmistettu käyttäen Unityn pelimoottoria ja siinä peli käyttää puhelimen GPS-sijaintia piirtääkseen pelaajan sijaintia vastaavan kartan, jota pitkin pelaaja kulkee. Liikkuessaan oikeassa maailmassa, pelissä oleva avatar-hahmo liikkuu kartalla saman matkan. Kun pokemon on pelaajan lähistöllä, ilmestyy se pelikartalle. Tällöin pelaaja voi painaa pokemonin kuvaa ja silloin peli siirtyy taistelutilaan, jossa pokemonin voi yrittää napata itselleen. Lisätyn todellisuuden ollessa aktivoituna, voi kyseisen pokemonin nähdä kännykän ruudulla aivan kuin se olisi siinä oikeassa maailmassakin. Pokemon Go:ta ladattiin vuoden 2016 loppuun mennessä yli 500 miljoonaa kertaa

maailmanlaajuisesti ja tällähetkellä peliä on ladattu yli 750 miljoonaa kertaa. Aktiivisia pelaajia Pokemon Go:lla on kuukausittain noin 65 miljoonaa.

Kuva 2. Pokemon GO ja lisätty todellisuus (2016)

Kosketusnäyttöisten mobiililaitteiden myötä näyttöä käytetään peleissä lähes poikkeuksetta peliohjaimena pelin esittämisen lisäksi.

Maaliskuussa 2017 Nintendo julkaisi Nintendo Switch pelikonsolin. Kyseistä laitetta voi käyttää sekä kannettavana käsikonsolina että telakoituna konsolina kytkettynä televisioon. Laitteessa on kaksi Joy-Con-ohjainta, joita voi käyttää joko erikseen tai yhteen liitettyinä. Switchiä on myyty tähän päivään mennessä lähes 18 miljoonaa kappaletta ja se on sekä Nintendon historian nopeiten myyvä konsoli että Japanin ja Amerikan nopeiten myyvä konsoli. (Wikipedia, 2018)

3 PELIMOOTTORI

Pelimoottorin valinta on pelintekijälle hyvin tärkeää. Tarjolla on useita pelimoottoreita, joista selvästi suosituimmat ovat Unity, Unreal Engine sekä

nyt uutuutena tullut Lumberyard. Kaikki pelimoottorit eroavat toisistaan ominaisuuksiensa ja ulkoasujensa myötä, joten on hyvä valita sellainen pelimoottori, jossa on tarpeelliset ominaisuudet omaa peli-ideaa varten. Oma valintani näistä pelimoottoreista osui tällä kertaa Unityyn. Syitä Unityn valintaan oli muutamia, esimerkiksi aikaisempi kokemus sekä selkeä ja helppokäyttöinen käyttöliittymä.
Amazonin kehittämän Lumberyardin olemassaolosta en ollut tietoinen projektin alkuvaiheilla, joka on pääsyynä sille etten sitä valinnut. Sitä olisi ollut kyllä hauska päästä kokeilemaan, mutta se tapahtunee vasta tulevissa projekteissa. Unreal Enginestä puolestaan itselläni on myös kokemusta yhden pelinkehityskurssin verran. Unreal Enginen hyviä puolia olivat muun muassa blueprintit, joiden avulla pystyi tekemään peliin tarvittavia muutoksia ilman ohjelmointia. Se on siis mielestäni ehkä kaikkein paras vaihtoehto sellaiselle pelintekijälle, jonka ohjelmointitaidot eivät ole riittävät valmiin pelin luontiin asti. Kaikilla näillä pelimoottoreilla on tehty hyviä ja menestyneitä pelejä. Esimerkiksi yksi tämän hetken suosituimmista peleistä, Fortnite, on kehitetty käyttäen Unreal Engineä.

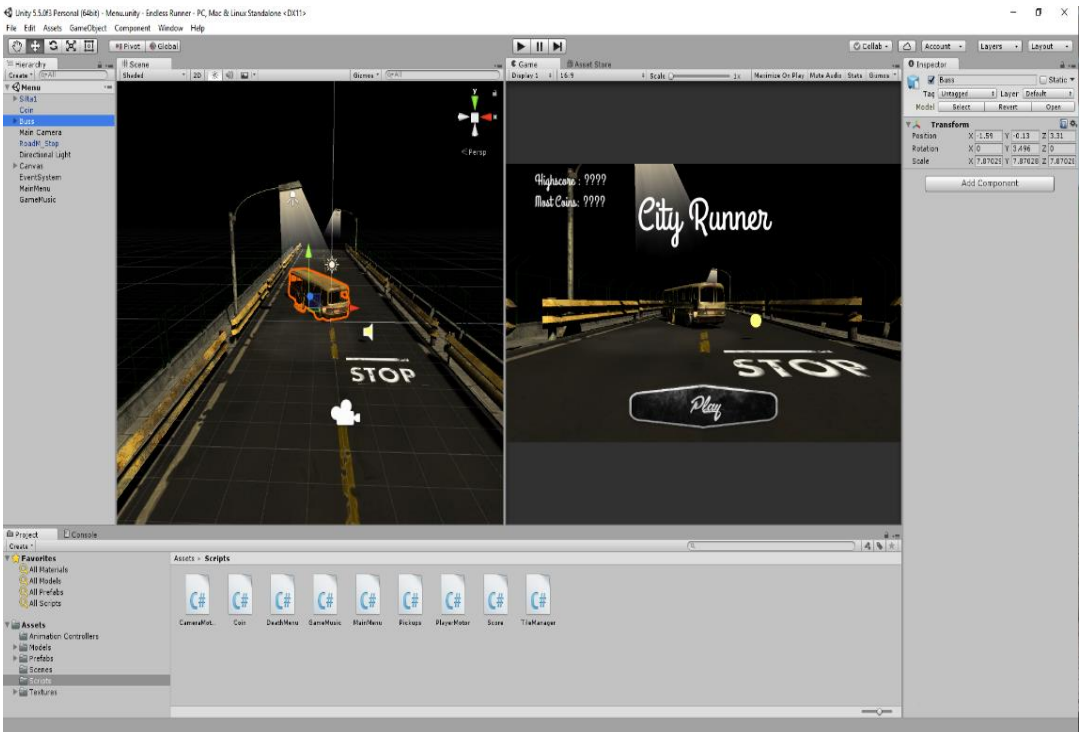
3.1 **Unity pelimoottorina**

Unity-pelimoottori on monipuolinen ja tehokas työväline pelituotantoa varten. Se tukee monia eri alustoja, kuten esimerkiksi mobiilipuolelta iOS-sekiä Android alustoja. Muita tuettuja suosittuja alustoja ovat muun muassa Windows, Linux, verkkoselaimet, PS3, Xbox 360 sekä Wii U. (Tukiainen 2013.)

Unityn laajat ominaisuudet mahdollistavat 2D- ja 3D-pelien toteuttamisen. Sillä voidaan helposti luoda erilaisia objekteja, joita on myös mahdollista muokata monien erilaisten koodien, eli scriptien avulla. Unityllä on mahdollista käyttää ohjelmointiin kolmea eri ohjelmointikieltä; Boo, JavaScript sekä C#. Näistä ohjelmointikielistä Boo on jo lähes hävinnyt kokonaan käytöstä ja JavaScriptin käyttö on myös vähenemässä. Pelimoottoria pyritään päivittämään jatkuvasti ja uusin versio ohjelmasta on Unity version 2017.4.1, joka on julkaistu 6. huhtikuuta 2018. Unitystä on saatavilla ilmaisen version lisäksi myös Unity Plus ($35/kk) sekä Unity Pro ($125/kk), jotka ovat maksullisia ja sisältävät enemmän ominaisuuksia kuin ilmainen Unity Personal.

3.2 **Käyttöliittymä**

Unityn käyttöliittymä on melko hyvin käyttäjän muokattavissa oman mieltymyksensä mukaan. Se koostuu viidestä eri moduulista, joiden kokoa ja paikkaa voi muuttaa juuri haluamallaan tavalla. Tämän takia käyttöliittymää kutsutaan modulaariseksi. Käyttöliittymän viisi eri moduulia ovat nimeltään ”Project”, ”Scene”, ”Game”, ”Hierarchy” sekä ”Inspector”.



Kuva 3. Kuvankaappaus Unityn käyttöliittymästä, jossa näkyvät kaikki eri moduulit.

Project-moduulissa näkyvät kaikki peliin lisätyt objektit, joten se toimii ikään kuin pelin sisällysluettelona. Tässä moduulissa oleva sisältö vastaa oman projektikansion sisältöä työaseman resurssienhallinnassa. Pelin tai projektin ulkonäön kannalta tärkeimmät moduulit ovat Scene ja Game. Scene-moduulissa pystytään tekemään pelin sisäisiä muokkauksia ja se on tarpeellinen juurikin peliä ohjelmoitaessa. Game-moduuli taas näyttää, miltä kyseinen peli tai projekti näyttää ”pelitilassa”.

Hierarchy -moduuli näyttää vain Scene-moduuliin tallennetut objektit. Valittaessa jonkin objektin aktiiviseksi Hierarchy-moduulissa, se näkyy muokattavana scenessä. Objektin aktivoinnin myötä myös Inspector-moduuli aktivoituu. Inspector näyttää kaikki kyseisen objektin tiedot, siihen liitetyt script-tiedostot sekä tekstuurit. Tässä moduulissa objektin fysiikkamoottoriin voi tehdä muutoksia.

3.3 **Scenet**

Pelinkehityksessä scenellä tarkoitetaan pelissä olevia tiloja, joihin on lisätty pelissä olevia komponentteja ja objekteja. Scenestä toiseen liikkuminen vaatii aina jonkinlaisen latausajan objektien lataamiseen. Tämän takia niitä kutsutaan myös yleensä pelin kentiksi tai radoiksi. Monissa peleissä oleva päävalikko on hyvä esimerkki scenestä. Siinä seuraavaan sceneen

siirrytään usein pelaajan valinnan mukaisesti. Mitä suurempi scene on, sitä pidemmän latausajan se vaatii.

3.4 Scriptit

Scriptit ovat esimerkiksi JavaScriptillä tai C# luotuja koodinpätkiä, joilla voidaan ohjelmoida pelin sisäisiä objekteja tekemään monenlaisia tehtäviä. Tällaisia tehtäviä voivat olla esimerkiksi peliavatarin tai tekoälyn liikkuminen, ammusten lentäminen sekä pelissä olevien tavaroiden tai muiden objektien kerääminen. Scriptien avulla peliin pystyy luomaan oikeastaan minkälaista sisältöä vain.

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.)

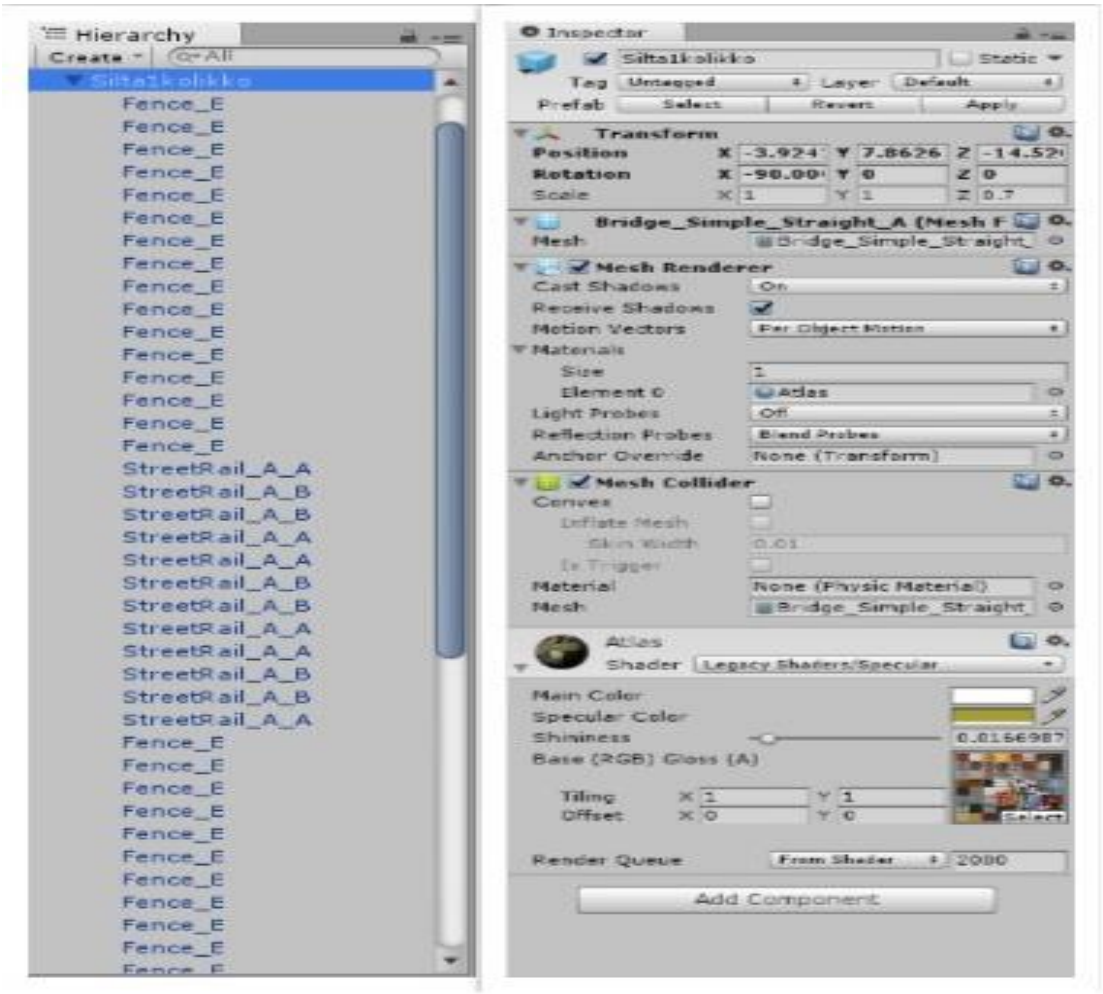
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Coin : MonoBehaviour {

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

        if (gameObject.tag == "Pickups")
                {
                transform.Rotate(0, 0, 2);
                }
        }
}
```

Esimerkki 1. Coin.cs tiedosto

Esimerkissä 1 on esitetty erittäin yksinkertain script-tiedosto, missä pelissä kerättävälle objektille – tässä tapauksessa kolikolle – on lisätty pyörimisominaisuus. Tiedostossa käytetään MonoBehaviour-luokan Update() -funktiota, joka ajaa sisällään olevat komennot jokaisen ruudunpäivityksen aikana. Kyseinen Update -funktio sisältävässä if-lausekkeessa määritellään pyörimisominaisuus kaikille peliobjekteille, joihin on lisätty tagi ”Pickups”.

3.5 **Prefabit**

Prefabit ovat useasta eri objektista luotuja peliobjekteja, jotka on tallennettu yhdeksi kokonaisuudeksi. Prefabeihin tehdyt muutokset vaikuttavat heti kaikissa sceneissä, joissa muokattua prefabia käytetään



Kuva 4. Kuvakaappaus prefabin Hierarchy sekä Inspector -moduuleista.

Kuvassa 4 on esiteltynä kuvakaappaus Endless Runner -pelissä olevasta siltaobjektista. Kyseinen prefab sisältää monta aitaa, sekä tienkaidetta. Lisäksi siltaobjektiin on aseteltu hajonneita autoja sekä kaatuneita pylväitä hankaloittamaan pelihahmon liikkumista. Näiden objektien lisäksi prefabiin on tallennettu myös muutamia kerättäviä kolikoita, joita pelaajan olisi tarkoitus yrittää kerätä pelin aikana.

4 PELISUUNNITTELU

4.1 Yleistä

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.)

Pelin suunnitteluvaiheessa on tärkeää saada käsitys minkä ikäisille pelaajille peli suuntautuu. Myös pelin genre määräytyy yleensä tässä vaiheessa, ennen kuin mitään suurempia suunnitelmia pelistä on tehty. Peliä suunniteltaessa on hyvä tehdä itselleen muistiinpanoja ja kirjata ylös mahdolliset ominaisuudet ja muut tärkeät osat, joita peliin tullaan myöhemmin lisäämään. Oman projektini suunnitteluvaiheessa kirjasin muistiinpanoaj ideoista ranskalaisilla viivoilla. Se on mielestäni hyvä ja yksinkertainen tapa tehdä muistiinpanoja ja siihen on helppo lisätä uusia suunnitelmia ja ajatuksia.

4.2 Mobiilipelien haasteet

Mobiilipelien suurimpia haasteita on mielestäni saada pelistä tarpeeksi mielenkiintoinen. Jos peli ei innosta pelaajaa tai saa häntä koukuttumaan pelistä, sitä ei montaa kertaa jaksa pelata. Myös mobiilialustoille luotavissa peleissä pelihahmon kontrollointi ja liikuttaminen on usein luotu liian haastavaksi. Ohjainpainikkeet ovat joko liian suuria tai niitä on liikaa, jolloin ne peittävät kuvaruutua liian paljon ja näin ollen hankaloittavat pelistä nauttimista. Tämä on yleinen ongelma ja johtuu mobiililaitteiden kosketusnäytöstä. Sen koko on rajallinen eikä sitä voi kasvattaa, jolloin pelihahmon kontrollointiin tarvittavia painikkeita on lisättävä näytölle.

Pelien suunnitteluvaiheessa on myös muistettava, että halutut ominaisuudet pitää olla tarpeeksi helposti ohjelmoitavissa, jotta ne saadaan lisättyä peliin. Jos ominaisuudet ovat liian hankalia toteuttaa, ei niitä välttämättä saa toimimaan pelissä kunnolla, jolloin pelistä tulee kömpelö ja pelikokemus kärsii. Pelejä on kuitenkin mahdollista päivittää jälkikäteenkin, joten on hyvä pitää ohjelmoinnit tarpeeksi yksinkertaisina alkuun. Päivityksillä voi sitten kehittää ja korjata pelissä olevia ominaisuuksia.

Mobiilipelien haasteita lisää myös laitteiden teknologia, joka ilmenee erityisesti reaaliajassa toimivissa moninpeleissä, eli synkronisissa peleissä.

Synkroniset moninpelit vaativat usein nopeaa ja tarkkaa ohjausta, joka voi olla hankalaa mobiililaitteen kosketusnäytöllä toteutettavaksi.

4.3 **Pelin idea**

Tässä opinnäytetyössä tehtävässä mobiilipelissä peli-ideana on luoda yksinkertainen kolmiulotteinen ”Endless Runner” -tyyppinen juoksupeli. Pelissä esiintyy käyttäjän ohjaama pelihahmo, joka pyrkii väistelemään tiellä olevia esteitä. Esteiden väistelyn lomassa pelaajan on mahdollista kerätä reitillä olevia kolikoita. Pelissä oleva pelialue muodostuu autotiestä, joka on reunustettu aidoilla, jotta pelialue pysyy hallinnassa ja pelaaja ei voi poistua reitiltään. Pelin tavoitteena on pystyä väistelemään esteitä ja kerätä kolikkoja mahdollisimman kauan. Haastetta peliin tuo ajan myötä nopeutuva vauhti sekä pelaajan edessä etenevä sumupilvi, joka heikentää näkyvyyttä jolloin reitillä olevia esteitä ei pysty näkemään kuin vasta hieman ennen mahdollista törmäystä. Pelihahmon eteenpäin liikkuminen on tarkoitus luoda automaattiseksi, jonka lisäksi sivuttaisliikkeille ohjelmoidaan näyttöön näkymättömät painikkeet. Vaikutteita peliin on saatu mobiililaitteilla erittäin suosituiksi muodostuneista peleistä, kuten Temple Run tai Minion Rush.

Temple Run on Androidille ja iOS:lle kehitetty päättymätön juoksupeli, jossa pelihahmona toimivaa tutkija jahtaa joukko demonisia apinoita. Peli sijoittuu nimensä mukaisesti temppeliin, jota pitkin pelaaja pyrkii juoksemaan apinoita karkuun mahdollisimman kauan. Pelissä oleva scene on päättymätön ja koostuu monista erilaisista esteistä, jotka tulevat vastaan satunnaisessa järjestyksessä. Pelaaja yrittää parhaansa mukaan väistää esteitä liukumalla, hyppäämällä tai kääntymällä juuri oikeaan aikaan ja samalla kerätä pisteitä kerryttäviä kolikoita.

Minion Rushissa pelihahmona toimii ”Itse ilkimys” -elokuvista tuttu minion -hahmo. Pelin idea on käytännössä sama kuin Temple Runissa, mutta peli sijoittuu minioneille tutummassa ympäristössä.

4.4 **Projektin tavoitteet**

Opinnäytetyön tavoitteena oli luoda yksinkertainen ja toimiva mobiilipeli Unityn pelimoottoria käyttäen. Tarkoituksena oli saada valmis projekti muutettua Android käyttöliittyvälle sopivaan muotoon niin, että pelin käyttöliittymästä olisi selkeä ja helppokäyttöinen. Projekti oli tarkoitus luoda Unityn 5.5 versiolla ja ohjelmoinnit tapahtuivat Microsoft Visual Studio 2013 -ohjelmalla käyttäen C# (C-Sharp) ohjelmointikieltä. Unity-pelimoottorin valitsin tämän opinnäytetyön tekoon siksi, koska sen käytöstä minulla oli jo hieman kokemusta aikaisempien kurssien ansiosta. Lisäksi Unityn omilta sivuilta löytyy paljon hyviä ohjeita monien erilaisten ohjelmoitavien ominaisuuksien tekoon. Työssä käytiin läpi projektin kaikki osa-

alueet vaihe vaiheelta sekä havainnollistettiin kuvien ja ohjelmoitujen koodien kanssa asioita ja ominaisuuksia.

Tavoitteenani oli myös saada itselle hieman enemmän kokemusta pelinkehityksestä sekä ohjelmoinnista. Pyrin myös välittämään opinnäytetyön lukijalle selvän idean, mitä mobiilipelien teko todellisuudessa on ja mitä kaikkea siihen vaaditaan. Opinnäytetyön aiheeksi valitsin mobiilipelin valmistuksen siksi, koska pelinkehitys ja pelaaminen ylipäänsä on aina kiinnostanut minua ja olin halukas oppimaan lisää aiheesta sekä ohjelmoinnista. Valmista projektista tuli melko pelkistetty ja yksinkertainen, sillä olen vielä melkoinen aloittelija ohjelmoinnin osalta.

5 PELIN TOTEUTUS

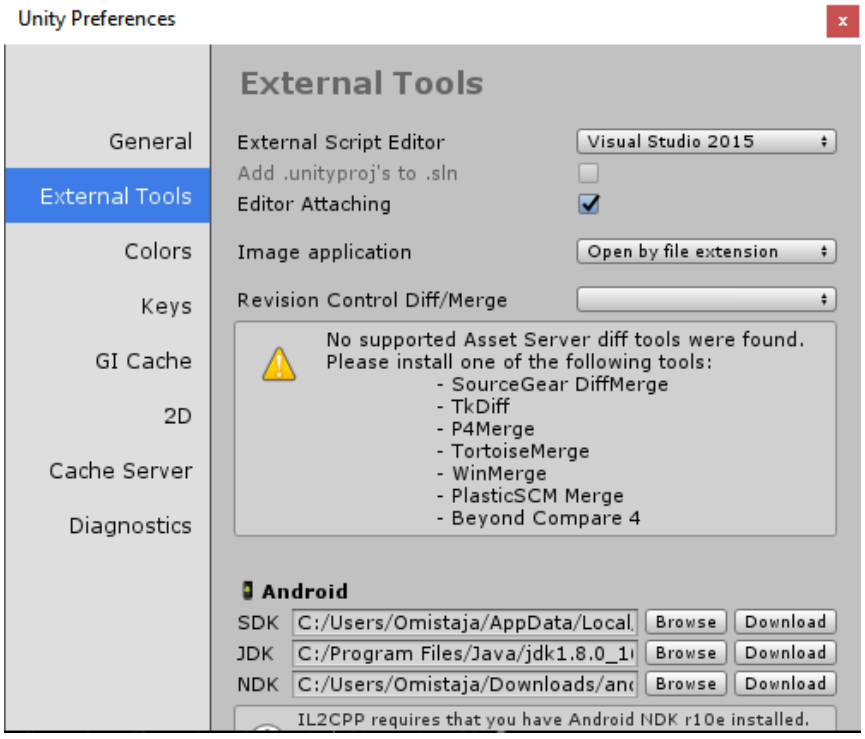
Tämä mobiilipeli on toteutettu käyttämällä Unityn valmistamaa pelimoottoria ja tarvittavat scriptit on tehty käyttäen C# -ohjelmointikieltä. Mobiilipelien tekemisessä on hyvä ottaa huomioon se, että valmis tuote olisi riittävän helppokäyttöinen. Tämän lisäksi pelin olisi hyvä olla myös tarpeeksi kiinnostava, jotta sitä jaksaa pelata useasti, eikä pelikerrat jäisi vähäisiksi. Toteutuksessa on hyvä muistaa myös mobiilipelien erityispiirteet verrattuna tavallisiin konsoli- tai tietokonepeleihin. Mobiilipelejä pelatessa ohjaimena käytetään useimmissa tapauksissa laitteen kosketusnäyttöä ja mobiililaitteet eivät ole yhtä tehokkaita kuin esimerkiksi tietokoneet.

5.1 Androidin käyttöönotto Unityssä

Jotta Unity-pelimoottori voisi muuntaa valmiin pelin Androidille sopivaksi paketiksi, täytyy Unityyn asentaa Androidin SDK lisäosa. Tämän asentaminen on onnistuttu tekemään melko yksinkertaiseksi. Ensimmäinen vaihe lisäosan asennuksessa on ladata Android Studion asennuspaketti heidän omilta kotisivuiltaan tai suoraan Unityn Build Settings sivulla olevasta painikkeesta. Paketin asennusvaiheessa kysytään, mitkä kaikki asennuspaketit käyttäjä haluaa ladata, mutta tässä projektissa tarvitaan vain Android SDK -pakettia.

Kun Unityyn on asennettu Android SDK, on projektitiedostot mahdollista saada muokattua .apk tiedostoiksi. Tämä tiedostotyyppi on Androidin tukema ja useimmat Android sovellukset onkin toteutettu tässä muodossa. Omaa projektia on mahdollista päästä kokeilemaan myös omalla mobiililaitteellaan ennen kuin sen muuntaa .apk -tiedostoksi. Tämän mahdollistaa Unityn luoma Unity Editor -työkalu, jonka saa ladattua Android laitteen Google Play -kaupasta. Unity Editorin saa käyttövalmiiksi yhdistämällä

laitteen ensin USB kaapelilla tietokoneeseen ja tämän jälkeen asettamalla Unityn puolelta Unity Remote laitteeksi ”Any Android Device”. Kyseisen muutoksen asetuksiin pääsee tekemään Unityssä menemällä Edit –> Project Settings –> Editor, ja sieltä Device kohtaan tehdään muutos.

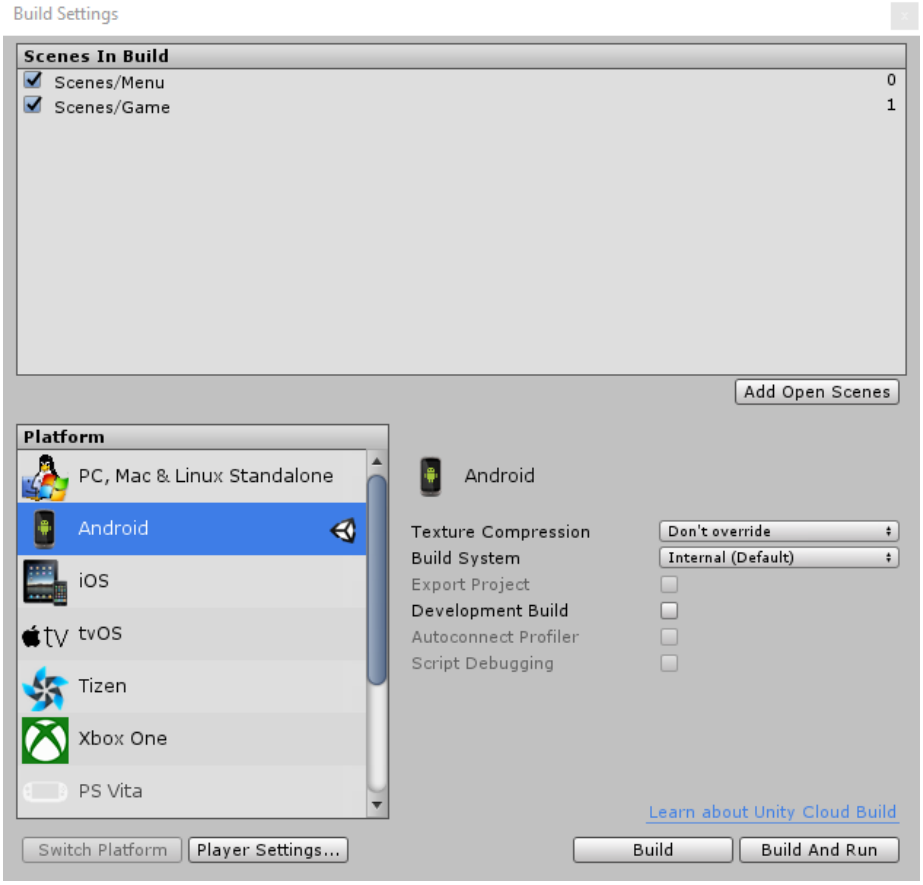


Kuva 5. Kuvakaappaus Unityn asetuksista, jossa näkyy Android SDK:n polku.

Kuvassa 6 on esiteltynä Unityn preferences ikkuna, tarkemmin sanottuna sen External Tools välilehti. Kun Android SDK on asennettuna koneelle, tulee tällä välilehdellä olevaan SDK kenttään täyttää SDK paketin asennuspolku. Joissain tapauksissa kentät hakevat asennuskansioiden tiedot automaattisesti, jolloin niitä ei tarvitse itse täyttää.

Projektin valmistuttua seuraava vaihe on muuntaa tietokoneella tehty peli Android laitteelle sopivaksi. Projektitiedostojen muuntaminen .apk -paketiksi on tehty Unityllä erittäin helpoksi. Kyseisen vaiheen pystyy suorittamaan menemällä polkuun File –> Build Settings. Seuraavaksi avautuvassa ikkunassa päästään määrittelemään mille käyttöliittymälle sopivaksi projekti pakataan. Lisäksi Build Settings ikkunalla pyydetään lisäämään kaikki projektissa käytössä olevat scenet. Scenet pitää vielä järjestää niin, että projektissa mahdollisesti oleva päävalikko on ensimmäisenä ja muut scenet vasta sen jälkeen. Tämä on tehtävä siksi, jotta pelin käynnistyessä mobiililaitteella ensimmäinen scene joka avautuu on päävalikko, eikä esimerkiksi asetusvalikko tai loppuscene. Kun scenet on lisätty ja valittu oikea käyttöliittymä – tässä tapauksessa Android – voidaan painaa Build -painiketta. Nyt ohjelma alkaa pakkaamaan projektitiedostoja oikeaan

muotoon. Pakkaaminen voi kestää useita minuutteja, riippuen projektin koosta ja ilmoittaa se ilmoittaa mahdollisista häiriöistä tai ongelmatilanteista virheilmoituksella.



Kuva 6. Build Settings ikkuna, johon on määritelty tarvittavat scenet ja käyttöliittymä.

5.2 **Ulkoasu**

Mobiilipelien kehitysvaiheessa on tärkeää ottaa pelin ulkoasu huomioon. Mobiililaitteissa näytön koko ja pelin tyyppi vaikuttavat paljon siihen, missä asennossa näyttö kannattaa pitää pelin käynnistäessä. Pelin voi myös pakottaa käynnistymään pelkästään pysty- tai vaaka asentoon, mutta sen voi myös asettaa vapaaksi, jotta näyttöä kääntämällä myös peli kääntyy. Työn mobiilipelissä peli on pakotettu avautumaan vaaka-asennossa, jotta peliavatarin ohjaus ja pelin näkyvyys olisi tarpeeksi hyvä.

5.2.1 Main Menu

Työssä pelille on tehty kaksi erillistä sceneä. Ensimmäinen scene, joka avautuu pelin käynnistyksen jälkeen, on nimeltään Main Menu eli

päävalikko. Päävalikko on toteutettu scenelle luomalla yksinkertainen canvas-objekti, jolle tarvittavat painikkeet ja taustaan tarvittavat objektit tai kuvat voi lisätä.

Kuva 7. Kuvakaappaus pelin Main Menu -päävalikosta.

Päävalikossa on esiteltynä pelille annettu nimi sekä taustakuva, joka antaa osviittaa pelin ulkonäöstä. Valikosta löytyy myös painike, josta pelin saa käynnistymään sekä tiedot kyseisellä laitteella peliin tehdyistä ennätystuloksista.

5.2.2 Pelitilanne

Pelitilanteeseen pääsee päävalikon ”Play” -nappia painamalla. Napin painamisen jälkeen sovellus avaa uuden scenen, jossa pelitilanne tapahtuu. Pelin tarkoituksena on liikuttaa pelihahmoa autotiellä ja kerätä mahdollisimman paljon alueelle asetettuja kolikoita. Peliä hankaloittamassa on erilaisia esteitä, kuten ajoneuvoja ja pylväitä, joita on asetettu peliradalle esteeksi. Pelihahmon törmätessä esteeseen, peli loppuu ja avautuu uusi scene. Tässä scenessä on esiteltynä pelissä saavutetut pisteet ja kerätyt kolikot.

Pelihahmon eteenpäin juokseminen tapahtuu automaattisesti, mutta hahmoa pystyy ohjaamaan painamalla näytön oikeaa tai vasenta laitaa. Ohjaus on jaettu kahteen pystysuuntaiseen osaan keskeltä näyttöä. Tämä tarkoittaa sitä, että painaessa esimerkiksi näytön vasenta laitaa, liikkuu peliavatar vasempaan suuntaan ja painaessa näytön oikeaa laitaa, ohjautuu hahmo oikealle.

Kuva 8. Kuvakaappaus pelitilanteesta aivan pelin alkuvaiheilla.

Yllä olevassa kuvassa 7 on esiteltynä pelitilanteessa näkyvä käyttöliittymä. Näytön ylälaidasta löytyy laskurit, jotka laskevat pelissä saavutettuja pisteitä ja kerättyjä kolikoita. Tielle on asetettu esteeksi rikkinäisiä ajoneuvoja ja muita esteitä. Tämän lisäksi etualalle on lisätty eteenpäin näkemistä hankaloittamaan useista partikkeleista muodostettu sumupilvi.

Pelissä pisteitä kerryttää hengissä pysytty aika. Ajan edetessä, myös vaikeusaste kasvaa ja pelihahmo kiihdyttää vauhtiaan. Vaikeusasteet on asetettu kasvamaan aina, kun pistemäärä on tuplaantunut. Ensimmäinen vaikeusaste kestää siis 20 pisteeseen asti, jonka jälkeen pelihahmo kiihdyttää vauhtiaan. Toinen vaikeusaste sisältää ajan, jolloin pisteitä on 20-40, ja kolmas vaihe on siis 40-80 pisteen aikavälillä. Maksimi vaikeusasteeksi on asetettu 10, joka tarkoittaa sitä, että pelihahmo voi kiihdyttää vauhtiaan maksimissaan kymmenen kertaa. Tämän jälkeen vauhti pysyy tasaisena niin kauan kunnes pelihahmo törmää esteeseen ja peli loppuu.

Pelissä pelaajan liikkuminen on rajoitettu pysymään tiellä laidassa olevien aitojen avulla. Se mahdollistaa pelin pysymisen mielenkiintoisena ja pelialue pysyy hallinnassa. Pelihahmon liikkumista hankaloittaviin esteisiin on lisätty Box Colliderit, jotta niihin on voitu liittää fysiikoita. Nämä fysiikat mahdollistavat sen, ettei pelihahmo pysty juoksemaan esteiden läpi vaan osuessaan esteeseen hahmon liike loppuu.

5.3 **Scriptit**

Scriptit määrittelevät kaikki pelin sisällä tapahtuvat toiminnot. Script-tiedostot liitetään peli-objekteihin, jolloin kyseinen objekti saa käyttöönsä scriptissä määritellyt ominaisuudet. Script-tiedostojen määrää ei ole

mitenkään rajoitettu ja niitä voi olla yhdessä objektissa useampiakin. Esittelen nyt tässä projektissa käytössä olleet scriptit sekä kerron minkälaisia ominaisuuksia niillä on saatu peliin luotua. Kaikki projektin scriptit on ohjelmoitu käyttäen C# (C-Sharp) ohjelmointikieltä.

5.3.1 PlayerMotor

Tässä scriptissä määritellään ohjattavan pelihahmon ominaisuuksia. Scriptin alkuun on määritelty muutamia scriptin sisäisiä komponentteja ja niiden arvoja. Näitä arvoja ovat esimerkiksi pelihahmon liikkumisnopeus pelin alussa, sekä painovoima.

```
using UnityEngine;
using System.Collections;

public class PlayerMotor : MonoBehaviour {

    private CharacterController controller;
    private Vector3 moveVector;

    private float speed = 5.0f;
    private float verticalVelocity = 0.0f;
    private float gravity = 12.0f;

    private float animationDuration = 3.0f;
    private float startTime;

    private bool isDead = false;
```

Alussa määriteltyjen arvojen ja luokkien jälkeen ohjelmoidaan pelin alussa tapahtuvat asiat void Start() -lausekkeessa. Tässä määritellään pelihahmon ohjausjärjestelmäksi pelaajan syöte sekä asetetaan pisteiden lasku alkamaan heti pelin käynnistyessä. Seuraavaksi määritellään void Update() -lauseke, joka tarkoittaa jokaisella ruudunpäivityksellä tapahtuvaa päivitystä scriptissä. Scripti siis tarkistaa koko ajan lausekkeessa tapahtuvia asioita ja tarpeen mukaan päivittää pelitapahtumia muutosten mukaan.

```
    void Start () {
        controller = GetComponent<CharacterControl-
        ler>();
        startTime = Time.time;
    }

    void Update () {
        if (isDead)
        return;

      if (Time.time - startTime < animationDuration)
      {
```

```
    controller.Move(Vector3.forward * speed *
    Time.deltaTime);
        return;
    }

    moveVector = Vector3.zero;

    if(controller.isGrounded)
    {
        verticalVelocity = -0.5f;
    }
    else
    {
        verticalVelocity -= gravity * Time.deltaTime;
    }

    // X-axis - Left and Right
    moveVector.x = Input.GetAxisRaw("Horizontal") *
     speed;
    if(Input.GetMouseButton(0))
    {
        if (Input.mousePosition.x > Screen.width / 2)
            moveVector.x = speed;
        else
            moveVector.x = -speed;
    }

    // Y-axis - Up and Down
    moveVector.y = verticalVelocity;

    // Z-axis - Forward and Backward
    moveVector.z = speed;
    controller.Move (moveVector * Time.deltaTime);
        }

public void SetSpeed(float modifier)
        {
        speed = 7.0f + modifier;
        }

    //It is being called every time our capsule hits
    something
    private void OnControllerColliderHit (Controller-
    ColliderHit hit)
        {
        if (hit.point.z > transform.position.z + 0.1f
        &&  hit.gameObject.tag == "Enemy")
              Death();
        }

private void Death()
{
```

```
        Debug.Log("Dead");
        isDead = true;
        GetComponent<Score>().OnDeath();
        }
}
```

Yläpuolella näkyvän PlayerMotor -scriptin void Update() -lausekkeessa on määritelty muun muassa tunnistus pelaajan näytön painalluksista, jotta peli tunnistaa painallukset ja pelihahmon liikuttaminen onnistuu. Lisäksi scripti sisältää määrityksen esteisiin törmäyksen tarkastuksesta eli se tarkastaa jokaisen ruudunpäivityksen aikana, osuuko pelihahmo johonkin ”Enemy” -tagilla merkittyyn esteeseen. Jos pelihahmo osuu tällaiseen esteeseen, peli loppuu ja scripti tallentaa kyseisellä hetkellä olevan pistemäärän muistiin.

5.3.2 CameraMotor

CameraMotor -scriptissä on määritelty pelissä käytössä olevan kameran toiminnallisuuksia. Tälläisiä ovat esimerkiksi kuvakulma ja se että kamera seuraa pelihahmoa tietyn matkan päästä koko ajan.

```
using UnityEngine;
using System.Collections;

private Transform lookAt;
      private Vector3 startOffset;
      private Vector3 moveVector;

      private float transition = 0.0f;
      private float animationDuration = 3.0f;
      private Vector3 animationOffset = new Vec-
      tor3(0,10,5);

void Start () {
      lookAt = GameObject.FindGam
      eObjectWithTag("Player").transform;
      startOffset = transform.position - lookAt.po-
      sition;
      }


void Update () {
  moveVector = lookAt.position + startOffset;
  // X
  moveVector.x = 0;
  // Y
  moveVector.y = Mathf.Clamp(moveVector.y, 11, 5);

      if (transition > 1.0f)
      {
```

```
            transform.position = moveVector;
        }

        else
        {
        transform.position = Vector3.Lerp(moveVector +
         animationOffset, moveVector, transition);
        transition += Time.deltaTime*1 / animationDuration;
        transform.LookAt(lookAt.position + Vector3.up);
                    }
            }
        }
```

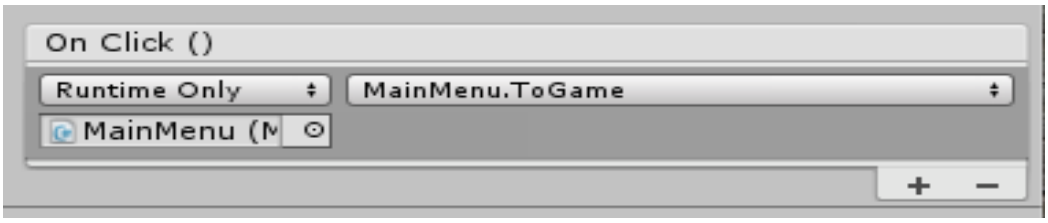
Kamerassa olevaan scriptiin määrittelin myös pelin aloitukseen liittyvän kolmen sekunnin pituisen animaation. Tämän animaation alussa kameran kuvakulma on suoraan pelihahmon yläpuolella ja kuvaa alaspäin hahmoa. Kolmen sekunnin aikana kamera liikkuu pelihahmon yläpuolelta hieman taka-alalle ja nostaa kuvakulmaa sen verran että saadaan pelissä käytettävä kuvakulma. Tämän animaation aikana myös pelihahmon liikuttaminen on lukittu, eli pelihahmo ei siis vastaa käyttäjän syötteeseen tuona ajankohtana.

5.3.3 MainMenu

MainMenu scriptissä määritellään saman nimisessä scenessä olevat ominaisuudet. Se näyttää päävalikossa pelissä ennätystuloksia ja lisäksi se mahdollistaa valikossa olevan ”Play” -painikkeen toimivuuden.

```
public class MainMenu : MonoBehaviour {

public Text mostcoinsText;
public Text highscoreText;

// Use this for initialization
void Start () {
   highscoreText.text = "Highscore : " +
   ((int)PlayerPrefs.GetFloat("High-
   score")).ToString();
   mostcoinsText.text = "Most Coins : " +
   ((int)PlayerPrefs.GetFloat("Most-
   Coins")).ToString();
}

public void ToGame()
       {
       SceneManager.LoadScene("Game");
       }
}
```

Public void ToGame() -lausekeen avulla päävalikkoon luodaan OnClick() -funktio, jonka avulla pystytään liikkumaan scenestä toiseen. Klikkaamalla päävalikossa olevaa ”Play” -painiketta, alkaa scripti ladata ”Game” -nimistä sceneä.



Kuva 9. Kuvakaappaus aloitusnäytön painikkeen OnClick() -lausekkeesta, johon on lisätty scripti.

5.3.4 TileManager

TileManager -scripti oli mielestäni kaikista vaativin ja vaati todella paljon hienosäätöä, jotta sen sai toimimaan haluamallani tavalla. Tässä scriptissä luotiin aluksi lista, johon määriteltiin useita valmiiksi tehtyjä prefab-objekteja. Lista voi olla kuinka suuri vain, mutta tätä projektia varten valmistin neljä erilaista prefabia joita käytin. Scripti käytti listassa olevia prefab-objekteja luomaan pelihahmon edelle tietyn välimatkan välein uusia tienpalasia. Tämä mahdollisti sen, ettei peliin tarvinnut luoda valmiiksi yhtä erittäin pitkää tietä, jota pelaaja tulisi kulkemaan. TileManager -scriptin avulla peli luo valmiiksi viiden prefabin verran tienpalasia. Kun pelaaja sitten ajan myötä ohittaa yhden prefabin, poistaa scripti taakse jääneen prefabin ja luo eteen uuden prefabin.

```
public class TileManager : MonoBehaviour {

    public GameObject[] tilePrefabs;

    private Transform playerTransform;
    private float spawnZ = 20f;
    private float tileLength = 60.0f;
    private float safeZone = 35.0f;
    private int amnTilesOnScreen = 5;
    private int lastPrefabIndex = 0;

    private List<GameObject> activeTiles;

    private void Start () {
        activeTiles = new List<GameObject>();
```

```
        playerTransform = GameObject.FindGa-
        meObjectWithTag("Player").transform;

  for(int i = 0; i < amnTilesOnScreen; i++)
  {
      if (i < 2)
          SpawnTile(0);
      else
          SpawnTile();
      }
}

private void Update () {
      if(playerTransform.position.z - safeZone >
      (spawnZ - amnTilesOnScreen  * tileLength))
  {
      SpawnTile();
      DeleteTile();
      }
}

private void SpawnTile(int prefabIndex = -1)
      {
      GameObject go;
      if (prefabIndex == -1)
        go = Instantiate(tilePrefabs[RandomPrefabIn-
   dex()]) as GameObject;
      else
        go = Instantiate(tilePrefabs[prefabIndex])as
   GameObject;
  go.transform.SetParent(transform);
  go.transform.position = Vector3.forward * spawnZ;
  spawnZ += tileLength;

  activeTiles.Add(go);
      }

private void DeleteTile()
      {
      Destroy(activeTiles[0]);
      activeTiles.RemoveAt(0);
      }
```

Tämä script-tiedosto mahdollistaa myös lyhyemmän latautumisajan sceneä ladatessa, sillä sen ansiosta pelin ei tarvitse luoda kerralla niin suurta määrää objekteja sceneen. Nyt riittää, kun se saa ladattua viisi ensimmäistä prefabia.

```
private int RandomPrefabIndex()
      {
```

```
        if (tilePrefabs.Length <= 1)
              return 0;

        int randomIndex = lastPrefabIndex;
        while(randomIndex == lastPrefabIndex)
        {
         randomIndex = Random.Range(0, tilePre-
    fabs.Length);
        }

        lastPrefabIndex = randomIndex;
        return randomIndex;
          }
        }
```

Lisäksi TileManager -scriptin ominaisuuksiin kuuluu satunnaisesti valittava prefabin lisäys. Tämän ansiosta peli arpoo satunnaisesti listassa olevista objekteista seuraavaksi lisättävän prefabin. Näin ollen mitä useampia valmiita prefabeja listaan lisää, sitä epätodennäköisemmin sama objekti tulee kahdesti peräkkäin.

5.3.5 Pickups

Pickups koodissa on määritelty kerättävien kolikoiden ominaisuuksia sekä kolikkolaskurin toiminta. Alkuun määriteltiin kolikkolaskurin alkuarvoksi nolla, jotta pelin alkaessa käyttäjälle ei ole ilmestynyt ”ilmaisia” kolikoita. Void Update() -lausekke sisältää kolikoiden poimintaan tarvittavat määritykset. Kolikoille on lisätty Collider -objekti, jotta niille saatiin lisättyä fysiikkaa. Pelihahmon osuessa kolikon Collider -objektiin, tarkistaa scripti että kyseessä on ”Pickups” -tagilla varustettu objekti. Kolikkoon osumisen jälkeen kolikko tuhoutuu pelinäkymästä ja kolikkolaskuriin lisätään yksi piste. Pisteet päivittyvät myös näytön yläosassa olevaan tauluun/laskuriin näkyviin.

```
public class Pickups : MonoBehaviour {

      public Text coinText;
      int coinCounter;

      // Use this for initialization
void Start () {
      coinCounter = 0;
      }

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

```
void OnTriggerEnter (Collider other)
{
      if (other.gameObject.tag == "Pickups")
       Destroy(other.gameObject);
       coinCounter++;
        // Debug.Log("Coins:" + coinCounter);
 coinText.text = "Coins: " + coinCounter.ToString();
             }
      }
```

Kolikot on myös ohjelmoitu pyörimään oman Z-akselinsa ympäri, jotta pelissä ne näyttäisivät hieman elävämmiltä ja paremmilta.

5.3.6 Score

Tässä koodissa on määritelty pisteiden lasku ja vaikeusasteiden kasvaminen. Pelin aloitus tapahtuu vaikeusasteella 1, joka on helpoin taso. Pisteitä saa kerrytettyä pysymällä pelissä mukana ja pisteiden kertyminen nopeutuu samassa suhteessa vaikeusasteen kasvun kanssa. Tämä tarkoittaa siis sitä, että mitä suurempi vaikeusaste, sitä nopeammin pisteitä kertyy.

```
public class Score : MonoBehaviour {

      private float score = 0.0f;

      private int difficultyLevel = 1;
      private int maxDifficultyLevel = 20;
      private int scoreToNextLevel = 10;

      private bool isDead = false;

      public Text scoreText;
      public DeathMenu deathMenu;

void Update () {
      if (isDead)
       return;

      if (score >= scoreToNextLevel)
       LevelUp();

       score += Time.deltaTime * difficultyLevel;
        scoreText.text = ("Time: " +
  (int)score).ToString();
      }
```

```
void LevelUp()
        {
         if (difficultyLevel == maxDifficultyLevel)
          return;

        scoreToNextLevel *= 2;
        difficultyLevel++;

        GetComponent <PlayerMotor>().SetSpeed(diffi-
        cultyLevel);

        Debug.Log(difficultyLevel);
        }
```

Pelaajan törmättyä esteeseen, scripti kertoo public void OnDeath() -lausekkeessa että isDead funktio on nyt totta. Tällöin peliruutu pimenee ja näyttöön avautuu deathMenu.

```
public void OnDeath()
{
        isDead = true;
        if(PlayerPrefs.GetFloat("Highscore") < score)
        PlayerPrefs.SetFloat("Highscore", score);
         deathMenu.ToggleEndMenu(score);
        }
}
```

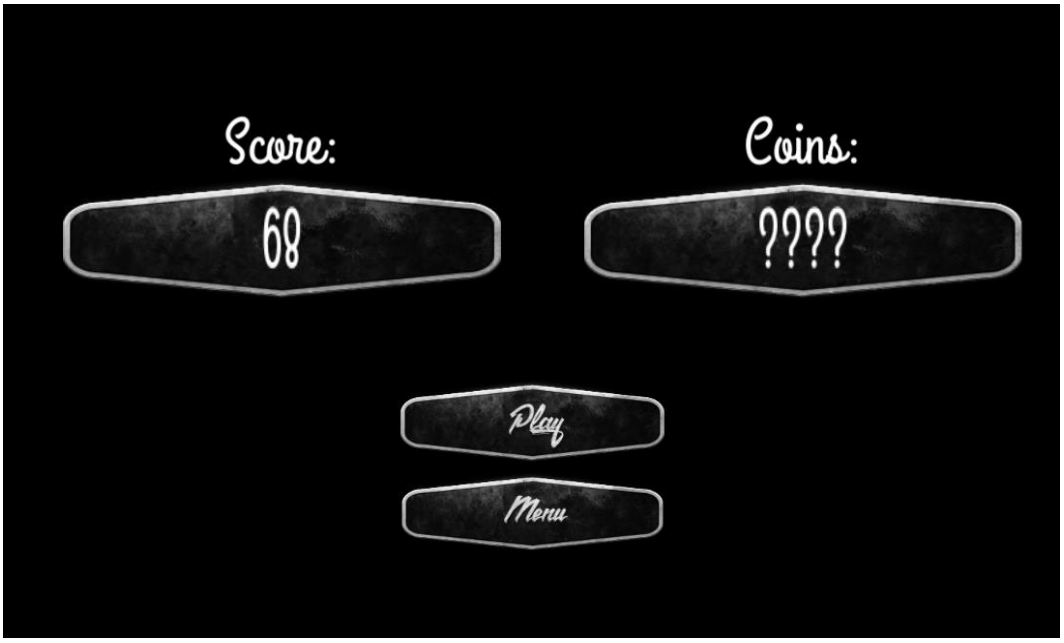
5.3.7 DeathMenu

DeathMenu on pelin viimeinen script-tiedosto. Siinä määritellään esteeseen törmäämisen jälkeen avautuvan DeathMenun ominaisuudet. Törmäyksen jälkeen näyttö muuttuu pimeäksi, jonka jälkeen näyttöön ilmestyy tiedot kerätyistä pisteistä ja kolikoista. Lisäksi tästä valikosta on mahdollisuus palata suoraan päävalikkoon tai vaihtoehtoisesti yrittää pelaamista uudelleen.

```
public class DeathMenu : MonoBehaviour {

        public Text coinText;
        public Text scoreText;
        public Image backgroundImg;

         public bool isShowned = false;
        public float transition = 0.0f;

       // Use this for initialization
       void Start () {
```

```
        gameObject.SetActive (false);
}

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

  transition += Time.deltaTime;
  backgroundImg.color = Color.Lerp (new Color(0, 0,
  0, 0), Color.black, transition);
}

public void ToggleEndMenu(float score)
{
  gameObject.SetActive (true);
  scoreText.text = ((int)score).ToString();
  isShowned = true;
}

public void Restart()
{
SceneManager.LoadScene(SceneManager.GetActiveS-
cene().name);
}

public void ToMenu()
       {
       SceneManager.LoadScene("Menu");
       }
}
```

Pelaajan valitessa ”Play”, lataa ohjelma Game -scenen uudelleen ja peli alkaa alusta. Menu -painikkeesta painettaessa latautuu päävalikon scene, ja peli ohjautuu takaisin päävalikkoon.



Kuva 10. Kuvakaappaus pelin DeathMenu valikosta.

Ylläolevasta kuvasta 9 voi nähdä, että DeathMenu -valikosta on tehty erittäin yksinkertaiseksi. Tämän lisäksi, jos pelissä ei saa yhtään kolikkoa, ei kolikkomäärää myöskään voida näyttää pelin päätteeksi.

6 YHTEENVETO

Kokonaisuudessaan opinnäytetyönä tehdyn mobiilipelin tekeminen sujui melko hyvin. Aloitusvalikon toteutus onnistui hyvin ja sain luotua tarvittavat painikkeet peliin pääsyyn ja pelin sammuttamista varten. Lisäksi valikon tausta näyttää melko hyvältä. Pelissä olisi voinut olla enemmän ominaisuuksia ja osa tavoitteissa olleista ominaisuuksista, kuten esimerkiksi pelin pysäytysvalikko ja kerättävät objektit jäivät kuitenkin tässä vaiheessa toteuttamatta. Lisäksi Temple Run -tyyliset tienristeykset olisivat antaneet hyvän lisän pelin vaikeusasteeseen. Pelissä oleva kolikkolaskuri jäi myös hieman keskeneräiseksi. Pelin aikana oleva laskuri kyllä toimii, mutta lopetusnäytössä oleva laskuri ei saa haettua kerättyjen kolikoiden määrää. Tämä sama ongelma ilmeni myös aloitusnäytössä olevassa ”Most Coins” -laskurissa. Ajatuksena on pyrkiä vielä kehittämään ja lisäämään uusia ominaisuuksia peliin tämän opinnäytetyöprojektin jälkeenkin.

Projektin päätarkoituksena ollut 3D mobiilipelin kehittäminen oli odotusteni mukaisesti melko haastavaa. Omien ohjelmointitaitojeni lisäksi jouduin turvautumaan Unityn tarjoamiin ohjeisiin. Heidän tarjoama ohjekanta

on hyvin laaja ja sieltä löytyy myös video-ohjeistuksia, mitkä antavat hyvän pohjan ohjelmoitaviin objekteihin. Ajan loppumisen takia jouduin hieman karsimaan projektissa olevia ominaisuuksia, mutta kokonaisuutena olen tyytyväinen projektissa valmistuneeseen peliin.

Toinen vaihe projektissa oli kirjoittaa opinnäytetyöstä selkeä ohjeistus, josta lukija saa käsityksen projektin vaiheista ja siitä mitä tämänkaltainen projekti tekijältään vaatii. Tässä onnistuin mielestäni hyvin ja sain kerrottua kattavasti mobiililaitteista ja niiden historiasta sekä Unity pelimoottorista itsestään. Sain kuvien ja omien kokemusteni pohjalta kerrottua Unityn ominaisuuksista ja käyttöliittymästä. Tämän lisäksi kerroin melko yksityiskohtaisesti projektissa käytetyistä script-tiedostoista ja mitä kaikkea niillä sain luotua peliin. Ohjeistuksen avulla mielestäni kokematonkin ja pelinkehityksestä kiinnostunut henkilö pystyy saamaan käsityksen, mitä scripteissä tapahtuu.

Unity käyttöliittymänä oli mielestäni helppokäyttöinen ja yksinkertainen. Ohjelmassa projektin jokaiselle osa-alueelle oli määritelty oma alueensa, jossa jokaisessa pystyi tekemään tarvittavia muutoksia objekteihin tai tiedostoihin.

LÄHTEET

Paavilainen, J (2009). Case: Mobiilipelaaminen. Haettu 10.3.2018 osoitteesta https://pelitieto.net/case-mobiilipelaaminen/

Husso, S. (2013). 2D-Peliohjelmointi Unityä käyttäen. Opinnäytetyö. Tietotekniikan koulutusohjelma. Kymenlaakson ammattikorkeakoulu. Haettu 17.3.2018 osoitteesta https://www.theseus.fi/bitstream/handle/10024/57338/Husso_Sami.pdf?sequence=1

Tukiainen, T. (2013). Mobiilipelit ja pelimoottorit. Pro gradu – tutkielma. Tietojenkäsittelytiede. Helsingin yliopisto. Haettu 20.4.2018 osoitteesta https://helda.helsinki.fi/bitstream/handle/10138/42050/20131106graduv3.pdf?sequence=2

Kellomäki, T (2012). Pelisuunnittelu. Haettu 22.4.2018 osoitteesta http://www.cs.tut.fi/~peliohj/peliohj2012-03-Pelisuunnittelu-webversio.pdf

Wikipedia. Kuvakaappaus Pokémon Go pelistä. Haettu 10.3.2018 osoitteesta
https://fi.wikipedia.org/wiki/Pok%C3%A9mon_Go

Wikipedia. Nintendo DS. Haettu 5.5.2018 osoitteesta
https://fi.wikipedia.org/wiki/Nintendo_DS

Wikipedia. Nintendo Switch. Haettu 5.5.2018 osoitteesta
https://fi.wikipedia.org/wiki/Nintendo_Switch