KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka / Ohjelmistotekniikka

Niko Nousiainen

2D PELIN LUONTI IPADILLE UNITY3D-OHJELMISTOLLA

Opinnäytetyö 2014

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka

NOUSIAINEN, NIKO	2d pelin luonti iPadille unity3d-ohjelmistolla
Opinnäytetyö	50 sivua + 15 liitesivua
Työn ohjaaja	Opettaja Paula Posio
Toimeksiantaja	Kymenlaakson ammattikorkeakoulu
Maaliskuu 2014	
Avainsanat	peli, ohjelmointi, C#, unity3d, 2d, kaksiulotteinen

Opinnäytetyön aihe oli kaksiulotteisen pelin luonti Unity3d-ohjelmistoa ja Orthello2d-lisäosaa käyttäen. Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa iPad-laitteella toimiva peli.

Opinnäytetyön tekeminen aloitettiin perehtymällä ensin kohdealustaan ja pelisuunnitteluun. Tämän perehtymisen pohjalta valittiin työhön sopivat työkalut ja tehtiin lyhyt suunnitelma pelin toteutuksesta ja toiminnallisuudesta. Peli toteutettiin pääasiassa Unity3d-ohjelmistolla ja siihen asennetulla Orthello2d-lisäosalla. Toteutukseen vaadittiin ohjelmointia, grafiikoiden luontia ja käytettyjen ohjelmistojen opettelua. Ohjelmointikielenä oli C#. Myös pelin kääntäminen iPadille oli iso osa opinnäytetyötä.

Opinnäytetyön tuloksena ei syntynyt kokonaista peliä, kuten alun perin oli tarkoitus, vaan lopputulos oli iPadilla pyörivä prototyyppi pelistä. Prototyyppi sisälsi yhden tason, missä pystyy kokeilemaan pelin toiminnallisuutta. Tason pystyy pelaamaan alusta loppuun, ja se koostuu useista eri komponenteista, joista kerrotaan opinnäytetyössä.

Pelin prototyypin luominen oli opettavainen kokemus ja kehitti projektinhallinta- ja suunnittelutaitoja sekä ohjelmointiosaamista. Valmiin prototyypin avulla pystyttiin tekemään päätöksiä pelin jatkokehityksen kannalta.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information technology

NOUSIAINEN, NIKO	Making a 2d-game for iPad using Unity3D
Bachelor's Thesis	50 pages + 15 pages of appendices
Supervisor	Paula Posio, Principal lecturer
Commissioned by	Kymenlaakso University of Applied Sciences
March 2014	
Keywords	game, programming, C#, unity3d, 2d-plugin

The subject of this thesis was making a 2d-game for iPad using Unity3d development engine and Orthello2D-plugin. Developing games for mobile devices has lately become increasingly popular due to their cheaper development costs and often higher net profit. When creating games, picking the right tools for the job is very important and therefore the study focused on the use of the chosen tools, and why they were picked. Information and examples on game design, programming and project management in game development were also included in the thesis.

The goal was to create a simple and enjoyable game for the iPad-device and to introduce some tools required to create a working 2d game. The created game prototype was a simple space flying game that could be played using just two fingers. Thesis was written to gain better understanding of many parts of game development, such as design, project management, creation of 2d-graphics and programming.

The game was designed based on research on many books and online sources. Tools were picked to suit the designed game. Information on the tools and their usage was mostly found from their developers' own websites.

Making the game was an educational experience and the end result (the created game prototype) was satisfactory. Further plans for the game could be refined on based on the created prototype.

SISÄLLYS

1 JOHDANTO

Opinnäytetyöni aihe syntyi pitkän mietinnän lopputuloksena. Olin jo pitkään halunnut luoda ensimmäisen kunnollisen peliprojektin ja opinnäytetyö tuntui hyvältä mahdollisuudelta saada se alulle. Päätin tehdä kaksiulotteisen pelin, sillä yksinkertaisen vektorigrafiikkaan pohjautuvan ulkoasun luonti ei tarvitse niin suurta graafista osaamista.

Pääasialliseksi työkaluksi valitsin Unity3d-ohjelmiston, sillä minulla oli sen käytöstä jo aiempaa kokemusta työharjoitteluni kautta. Koska Unity3d on suunniteltu alun perin kolmiulotteisten pelien luontiin, on kaksiulotteisen pelin luonti pelkästään Unity3d:n sisäänrakennetuilla ominaisuuksilla melko työlästä. Erillisen liitännäisen avulla kaksiulotteisten pelien luonti on huomattavasti luontevampaa. Opinnäytetyö keskittyykin paljon valitsemani 2d-pluginin, Orthello2d:n, käyttöön. Tätä opinnäytetyötä kirjoittaessani Unity julkaisi myös omat, Unity3d:hen sisäänrakennetut työkalut 2d-pelin luontia varten. Koska nämä työkalut olivat kuitenkin vasta kehitysvaiheessa, pitäydyin alkuperäisessä suunnitelmassani ja käytin Orthello2d:tä.

Pelin alustaksi valitsin iPadin, koska kyseessä on suosittu ja hyvin standardoitu laite, jolta löytyy suuri määrä käyttäjiä. Käyttäjien paljous tarkoittaa paljon potentiaalisia ostajia pelille tai sovellukselle. Hyvä standardointi takaa toimivuuden lähes kaikilla iPadin eri versioilla. Koska iPadille on jo aikaisemmin kehitetty lukuisia pelejä Unity3d:llä, löytyy kehitykseen paljon apua ja neuvoja lukuisista eri lähteistä.

Suunniteltaessa peliä tablettitietokoneelle piti ottaa huomioon erilaisia asioita, kuten ohjausmallin soveltuvuus kosketusnäytölle, laitteen tavallista pöytäkonetta pienempi teho ja pelin suunnittelu niin, että se soveltuu hyvin lyhytkestoisiin pelihetkiin.

Kuten pelituotannossa aina, oli tässäkin työssä useita eri vaiheita. Aloitin suunnittelusta, jonka jälkeen asensin tarvittavat ohjelmistot. Pelin varsinainen tuotanto koostui ohjelmoinnista ja graafisen sisällön tuotannosta. Kehityksen ohessa pyrin testaamaan peliä useasti. Testitulosten avulla tein tarvittaessa parannuksia peliin.

Opinnäytetyön tarkoituksena ei ollut luoda valmista peliä, vaan pikemminkin pohja jatkokehitystä varten.

2 MERKIT, LYHENTEET JA TERMIT

2d: 2d on lyhenne englanninkielen termistä two-dimensional (suom. kaksiulotteinen). Kaksiulotteisuudella voidaan viitata sekä objektin ulkonäköön (objekti on litteä kuva) tai ominaisuuteen (objekti pystyy liikkumaan vain kahdessa eri tasossa).

3d: 3d on lyhenne englanninkielen sanasta three-dimensional (suom. kolmiulotteinen). Kolmiulotteisuudella voidaan viitata objektin ulkonäköön (objekti on mallinnettu kolmen tilaulottuvuuden suhteen) tai ominaisuuteen (objekti pystyy liikkumaan kaikkiin suuntiin).

Frame: Frame-termillä tarkoitetaan pelinkehityksessä yhtä piirtokertaa päätelaitteelle. Jokaiselle framella piirretään päätelaitteelle kaikki piirrettäväksi määritellyt peliobjektit. Iso osa Unityn koodista käydään läpi kerran jokaisen framen piirron yhteydessä.

Inkscape: Inkscape on vektorigrafiikan piirtämiseen tarkoitettu ilmainen ohjelma.

iPad: Applen valmistama taulutietokone. Yksi suosituimmista laitteista mobiilipeleille.

Mesh: Mesh on kolmiulotteisen objektin määritelty muoto, joka koostuu tasoista, reunoista ja kulmapisteistä. Nämä tasot ja reunat ja kulmapisteet muodostavat kolmioita, joista 3d-malli muodostuu.

Mobiilipeli: Peli jota pelataan jollakin kannettavalla laitteella esim. tabletilla tai matkapuhelimella.

Parallax scrolling: Kaksiulotteisissa videopeleissä käytetty tehokeino, jossa taustalla olevia objekteja liikutetaan hitaasti kameran mukana. Näin ne näyttävät pelaajalle liikkuvan hitaammin, kuin etualalla olevat objektit ja pelaajalle luodaan illuusio syvyydestä.

Plugin (suom. liitännäinen): Plugin on ohjelmiston jatkokappale, joka kytkettynä pääohjelmaan lisää, parantaa, tai tehostaa tämän ominaisuuksia. Usein jos ohjelmistosta puuttuu jokin tarvittava ominaisuus, voidaan se lisätä erillisen liitännäisen avulla.

Prefab: Unity3D:ssä prefabilla tarkoitetaan valmiiksi tallennettua peliobjektin mallia, johon on määritelty valmiiksi kaikki peliobjektin komponentit ja ominaisuudet. Siirtämällä prefabin Unityn sceneen, Unity luo objektin, jolla on kaikki samat komponentit ja arvot kuin prefabilla. Samaa prefabia voi käyttää useassa eri scenessä ja muuttamalla prefabin ominaisuuksia, kaikkien siitä tehtyjen kopioiden ominaisuudet muuttuvat samalla. (1)

Scene: Unity3d ohjelmistossa scene on yksi kerralla ladattava kokonaisuus, joka voi sisältää lukuisia peliobjekteja. Peli koostuu usein monesta eri scenesta, esim. päävalikko olisi yksi scene ja jokainen pelin tasoista olisi oma scenensä. (2)

Skripti, Skriptaus: Skripti (englanniksi script) on peliobjektiin liitettävä kooditiedosto, jossa määritellään kyseisen peliobjektin toiminnallisuus. Skriptauksella (englanniksi scripting) tarkoitetaan skriptitiedoston luomista jollakin ohjelmointikielellä.

Sprite, Spritesheet: Sprite on kaksiulotteinen kuva tai animaatio, joka on osa isompaa kokonaisuutta. Spritesheet on kokoelma useita spritejä samassa kuvatiedostossa.

Unity3d: Ohjelmisto johon on sisäänrakennettuna erittäin paljon pelien luontiin tarvittavia ominaisuuksia, kuten fysiikkamoottori ja piirtojärjestelmä. Unity3d:llä voi luoda pelejä usealle eri alustalle.

Vektorigrafiikka: Vektorigrafiikka eroaa normaalista kuvasta siten, että pisteiden sijasta kuva muodostuu laskennallisesti erilaisista muodoista koordinaatistossa, kuten pisteistä, viivoista, palloista ja neliöistä. Vektorigrafiikalla tehdyn kuvan kokoa voi muuttaa ilman kuvan laadun heikentymistä.(3: 117)

3 TEORIAA PELIN SUUNNITTELUSTA JA PROTOTYYPIN IDEOINTI

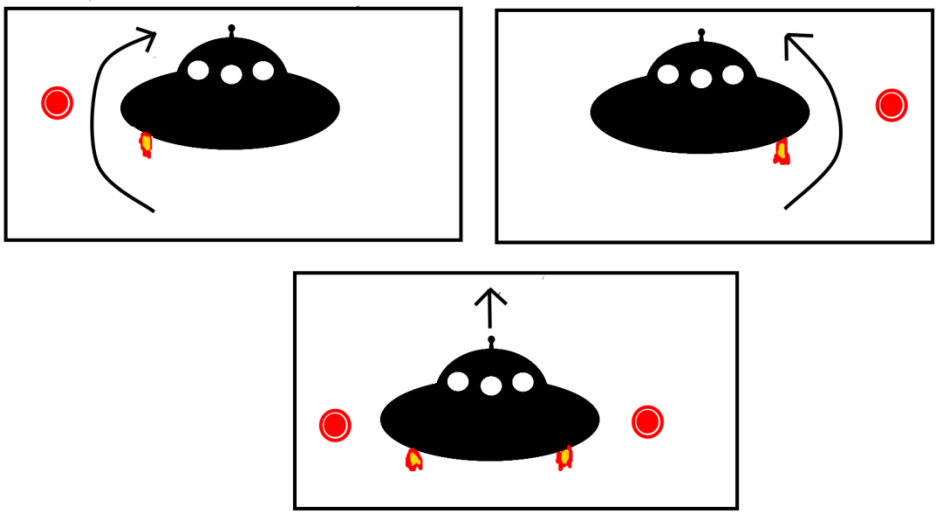
3.1 Tabletti pelialustana

Laitteena tabletti asettaa tiettyjä rajoituksia pelin suunnittelulle ja toteutukselle. Koska laitetta kannetaan paljon mukana, ja sitä käytetään lyhyitä ajanjaksoja kerrallaan (4: 44), pelin on koostuttava nopeasti pelattavista, hauskoista ja koukuttavista koko-

naisuuksista. Suurin tabletille suunnitteluun vaikuttava tekijä on ehkä kosketusnäytön asettama rajoitus pelin ohjausmekaniikalle. Koska käytössä ei ole erillistä peliohjainta, pelin kontrollien täytyisi olla mahdollisimman yksinkertaiset. Yksi opinnäytetyössä luodun pelin tärkeimmistä ominaisuuksista on, että pelaajalta saa vaatia maksimissaan kahta yhtäaikaista kosketusta. Kosketukset eivät saa myöskään olla liian monimutkaisia, jotta pelaaminen olisi helppoa.

3.2 Pelin idea

Pelin mekaniikka rakentuu yksinkertaisen kahden kosketuksen ohjausmallin ympärille. Pelaaja ohjaa hahmoaan koskettamalla tabletin ruutua joko vasemmalta tai oikealta puolelta. Pelaaja hahmo on avaruusalus, jolla on kaksi suihkumoottoria. Koskettamalla ruudun vasenta reunaa, aluksen vasen suihkumoottori aktivoituu, jolloin pelaajan alus kääntyy myötäpäivän. Vastaavasti oikeaa reunaa koskettaessa aluksen oikeanpuolinen suihkumoottori aktivoituu, jolloin alus kääntyy vastapäivään. Jos molemmilta puolilta ruutua koskettaa yhtäaikaisesti, saa alus liikuttavan voiman suoraan ylöspäin. Kuvassa 1. demonstroidaan pelaajan aluksen ohjausmallia.



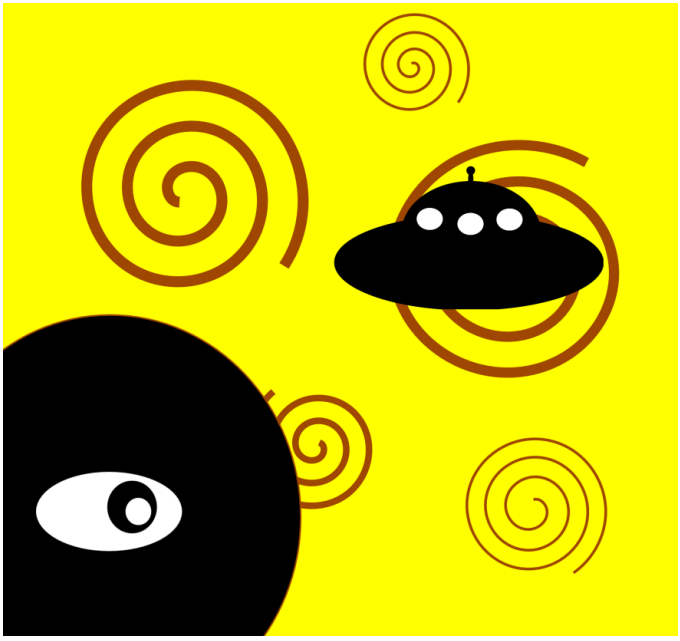
Kuva 1. Kuvassa näkyy, miten pelaaja voi ohjata alusta. Punaiset pisteet kuvaavat pelaajan sormen painallusta ja nuoli kuvaa aluksen liikkeen suuntaa.

Peli koostuu useista lyhyistä, maksimissaan viiden minuutin pituisista kentistä, joissa pelaajan tarkoituksena on ohjata alus kentän lopussa olevaan maaliin. Matkan varrella

vastaan tulee erilaisia esteitä ja vihollisia, joita pelaajan tulee väistellä. Jos pelaaja törmää seinään tai viholliseen, alus tuhoutuu ja kentän joutuu aloittamaan alusta.

3.3 Visuaalinen toteutus

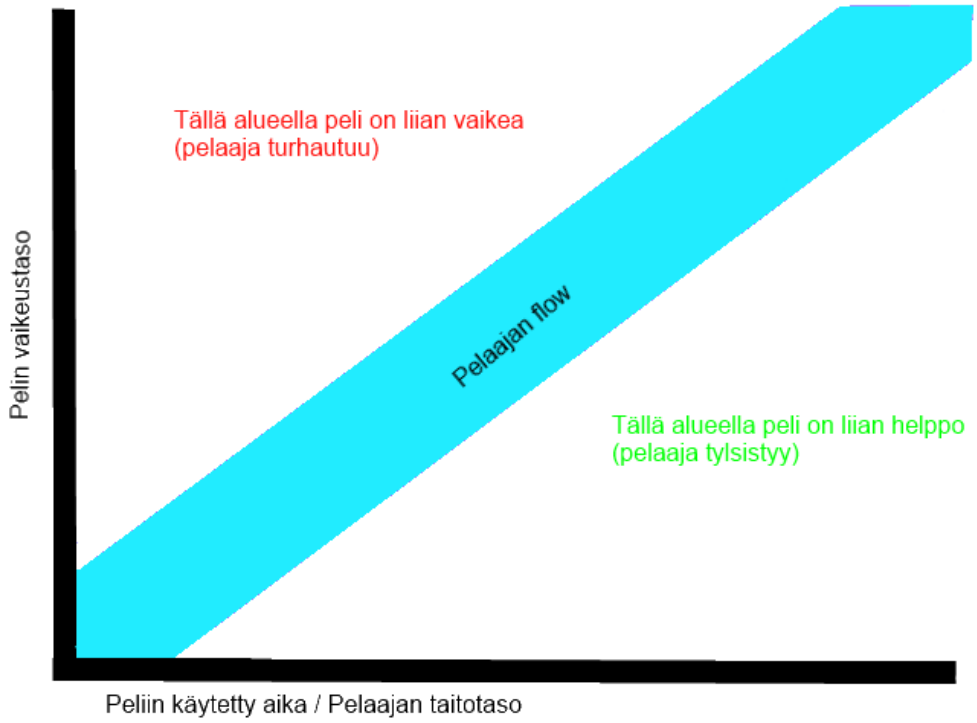
Pelin visuaalinen puoli muodostui heti sopivan peli-idean löydyttyä. Ulkonäön haluttiin olevan erittäin yksinkertaista, jotta sen toteutukseen ei kuluisi liikaa aikaa. Lopulta päädyttiin ratkaisuun, jossa pelin visuaalinen ilme luodaan värimaailman ja sen kontrastien avulla. Näin pelin taiteen ei tarvitse olla yksityiskohtaista, ja se pystytään toteuttamaan ilman ulkopuolista apua. Kuvassa 2. näkyy konseptitaidetta projektin alkuvaiheilta. Kuvasta käy ilmi, kuinka kaikki huomionarvoiset asiat erottuvat pelaajalle helposti värikkäästä taustasta tumman värinsä takia.



Kuva 2. Kuvassa on konseptitaidetta peliprojektin alkuvaiheilta. Tavoitteena oli luoda uniikki visuaalinen ilme värien kontrastien avulla.

3.4 Pelikokemus

Pelin kenttiä ja mekaniikkoja suunniteltaessa tulee ottaa huomioon pelaajan niin sanottu flow, eli optimaalinen kokemus. Flow'lla tarkoitetaan pelisuunnittelussa pelin vaikeustason nousua pelaajan taitojen mukana sopivassa suhteessa niin, ettei pelaaja missään vaiheessa turhautuisi, tai tylsistyisi pelaamiseen (5). Jokaisen kentän pitäisi olla toinen toistaan hieman haastavampia. Kuvassa 3. kuvataan yksinkertaisesti pelaajan flow.



Kuva 3. Kuvassa näkyy pelaajan flow (5).

3.5 Työvälineiden ja ohjelmistojen valinta

Ennen kuin päästään varsinaisen kehitysprosessin alkuun, täytyy valita tehtävään sopivat työkalut ja ohjelmistot. Työvälineiden valintaperusteet olivat tässä projektissa yksinkertaiset: niiden piti olla mahdollisimman helppokäyttöiset, monipuoliset ja ilmaiset. Alla on lyhyt esittely valituista työkaluista ja miksi juuri ne valittiin.

3.5.1 Unity3d

Unity3d on helppokäyttöinen pelimoottori, jossa on sisäänrakennettuna kaikki pelinrakentamiseen tarvittava. Unity3d:n peliobjektien scriptaamiseen löytyy erittäin kattavasti ohjeita Unityn omalta helppolukuiselta ”script reference”-sivustolta (6). Myös Unityn omilta keskustelupalstoilta löytyy paljon hyödyllisiä neuvoja tilanteeseen kuin tilanteeseen. Unityn skriptit voi luoda c#-ohjelmointikielellä, joka oli jo entuudestaan tuttu. Näin säästyttiin uuden ohjelmointikielen opettelun tuomalta lisätyöltä.

Myös ohjelman perusversion ilmaisuus oli tärkeä asia. Unity3d tarjoaa ilmaisen version, jossa on kaikki tarvittavat osat toimivan mobiilipelin luomiseen. Tarvittaessa Unityn voi myös päivittää pro-versioon kuukausimaksua vastaan, tai pysyvästi ostamalla

kalliimman pro-lisenssin. Unityn toiminnasta kerrotaan tarkemmin opinnäytetyön myöhemmissä vaiheissa.

3.5.2 Inkscape

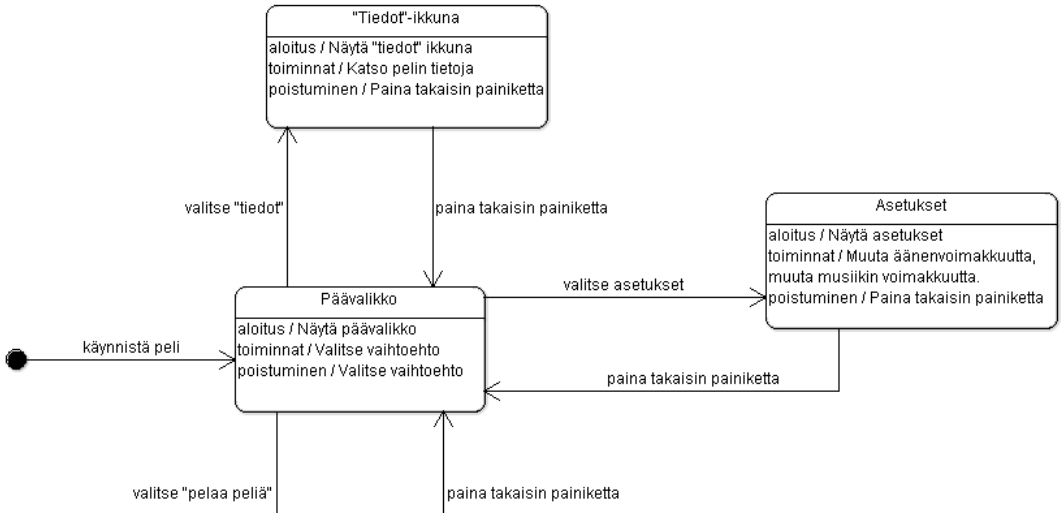
Inkscape on ilmainen vektorigrafiikan piirtämiseen tarkoitettu ohjelmisto. Suurin valintaperuste oli ohjelmiston ilmaisuus. Toinen vaihtoehto olisi ollut ammattilaisten keskuudessa paljon käytetty Adobe Illustrator. Kuitenkin Illustratorin kuukausimaksullisuuden vuoksi valittiin Inkscapen käyttö. Kaikki pelissä näkyvä grafiikka on tehty Inkscapella.

3.5.3 Orthello2d

Orthello2d on ilmainen liitännäinen Unity3d ohjelmaan, joka helpottaa kaksiulotteisten tekstuurien ja niiden animaatioiden käsittelyä ja luontia. Orthello helpottaa myös 2d-objektien prefabien hallintaa. Orthello 2d valittiin helpottamaan työntekoa sen ilmaisuuden ja opettajan antaman suosittelun takia.

3.6 Tilakaavio

Tilakaavioita käytetään ohjelmistotuotannossa kuvaamaan sitä, miten luokka tai ohjelma siirtyy eri tilojen välillä. Tilakaavio on hyvä suunnittelun työkalu ja auttaa hahmottamaan kokonaisuutta ennen ohjelmoinnin aloittamista. Koska myös pelit koostuvat erilaisista tiloista, voidaan tilakaaviota hyödyntää niiden suunnittelussa. Opinnäytetyön peliprojektiin tehtiin tilakaavio, jotta saatiin parempi käsitys kaikesta, mitä projektiin täytyi tehdä. Tilakaaviosta käy ilmi, miten pelissä on mahdollista siirtyä tahallisesti tai tahattomasti tilasta toiseen, sekä mitä pelaaja voi missäkin tilassa tehdä. Tilakaavio oli kokonaisuudessaan liian suuri laitettavaksi kuvana opinnäytetyöhön, joten se on erillisenä liitteenä (liite 1.). Kuvassa 4. on kuitenkin pieni osa tilakaaviosta esimerkin vuoksi.



Kuva 4. Kuvassa näkyy osa pelin tilakaaviosta. Koska tilakaavio on kokonaisuudessaan niin iso, on se erillisenä liitteenä (liite 1.).

Kaavion musta pallo kuvaa sitä pistettä, mistä käyttäjän kokemus alkaa, eli peli käynnistetään. Koska iPad sovelluksista ei saa löytyä omaa poistumis-painiketta – sovelluksista voi poistua ainoastaan iPadin oman fyysisen kotivalikko-painikkeen kautta, ei poistumispistettä merkitty lainkaan kaavioon. Laatikot kuvaavat jotain tiettyä tilaa sovelluksessa. Laatikon yläosassa näkyy tilan nimi, ja alla on kolme eri kohtaa, joissa kerrotaan mitä tilassa tapahtuu, kun sinne tullaan (aloitus), mitä tilassa on mahdollista tehdä (toiminnat) ja miten tilasta poistutaan (poistuminen).

4 TYÖYMPÄRISTÖN PYSTYTTÄMINEN

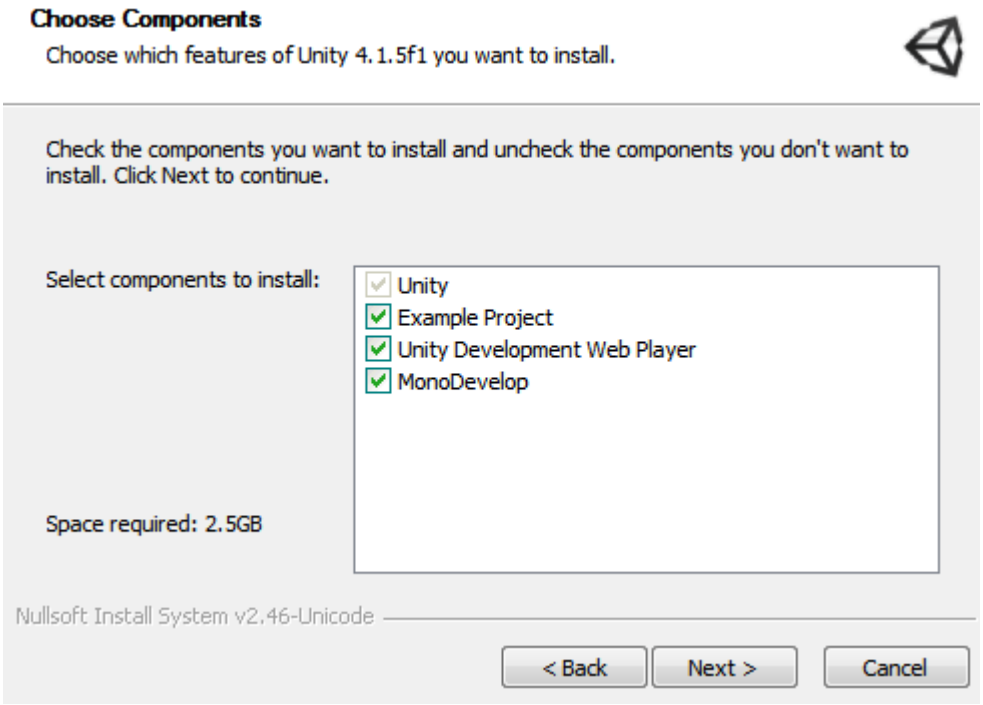
Tässä osassa opinnäytetyötä selvitetään miten tarvitut työkalut asennetaan, ja mitä niiden saaminen toimintakuntoon tavallisessa pc-laiteympäristössä vaatii.

4.1 Unity3d:n asennus ja käyttöönotto

Unity3d:n uusimman version (opinnäytetyön tekohetkellä 4.1.5) saa ladattua sen omilta kotisivuilta. Asennus tapahtuu yksinkertaisen asennustiedoston avulla. Asennus käynnistetään suorittamalla asennustiedosto. Tämän jälkeen asennusohjelma ohjaa askel askeleelta eteenpäin.

Asennuksen aikana valitaan, mitä komponentteja halutaan mukaan asennukseen. Opinnäytetyön projektia varten asennettiin kaikki mahdollinen, kuten kuvassa 5. nä-

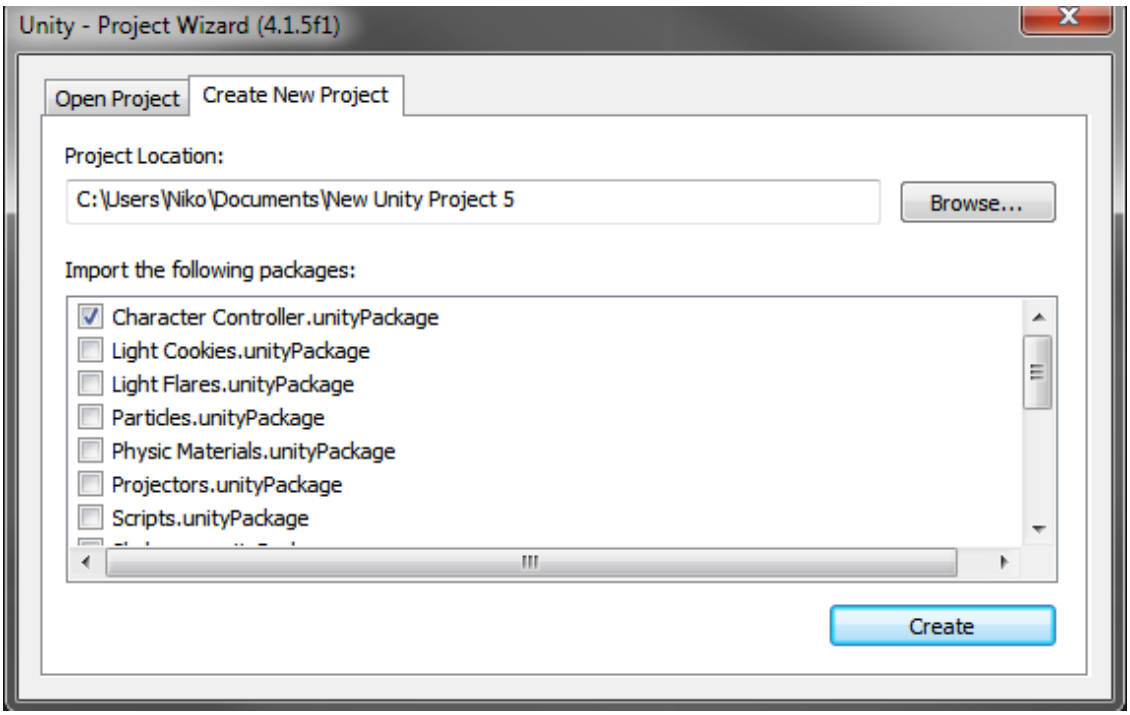
kyy. Tämä sisälsi Unity3d:n itsensä lisäksi esimerkkiprojektin, josta voi katsoa mallia tarpeen tullen, Unity Development Web Playerin, jolla voi pelata internetselaimeen upotettuja Unity-pelejä, sekä MonoDevelopin, joka on Microsoft Visual Studion kaltainen ohjelmankehitysympäristö. Ohjelmankehitysympäristö helpottaa eri ohjelmointikielillä ohjelmointia, koodin muokkaamista ja käsittelyä. Unityn mukana tuleva MonoDevelop on erikoisversio, joka on muokattu valmiiksi Unity3d-yhteensopivaksi.



Kuva 5. Kuvassa on Unityn asennuksen yhteydessä valittavat komponentit.

Kun halutut komponentit on valittu, pitää päättää, mihin Unity asennetaan. Tässä projektissa Unity asennettiin oletustiedostopolkuun (C:\Program Files (x86)\Unity\Editor). Tämän jälkeen täytyy vain odottaa, että asennus valmistuu, minkä jälkeen Unity3d on käyttövalmis. Asennus on erittäin yksinkertainen ja ongelmaton.

Uuden projektin luonti tapahtuu Unityssä ”file”-valikon alta löytyvästä ”New Project”-kohdasta. Kun uutta projektia luodaan, kysyy Unity, mitä paketteja halutaan mukaan projektiin. Nämä unityPackage-päätteiset tiedostot sisältävät valmiita materiaaleja tai objekteja eri tarkoituksiin. Opinnäytetyön peliä varten luodun projektin nimeksi tuli SuperAwesomeSpaceSaucer ja aivan kuten kuvassa 6, vain Character Controller paketti oli valittuna. Character Controller sisältää valmiin luokan pelaajan hahmon liikuttamiseen. Tätä SuperAwesomeSpaceSaucer-nimistä projektia käytetään myös myöhemmissä opinnäytetyön vaiheissa.



Kuva 6. Uuden projektin luonti Unityssä

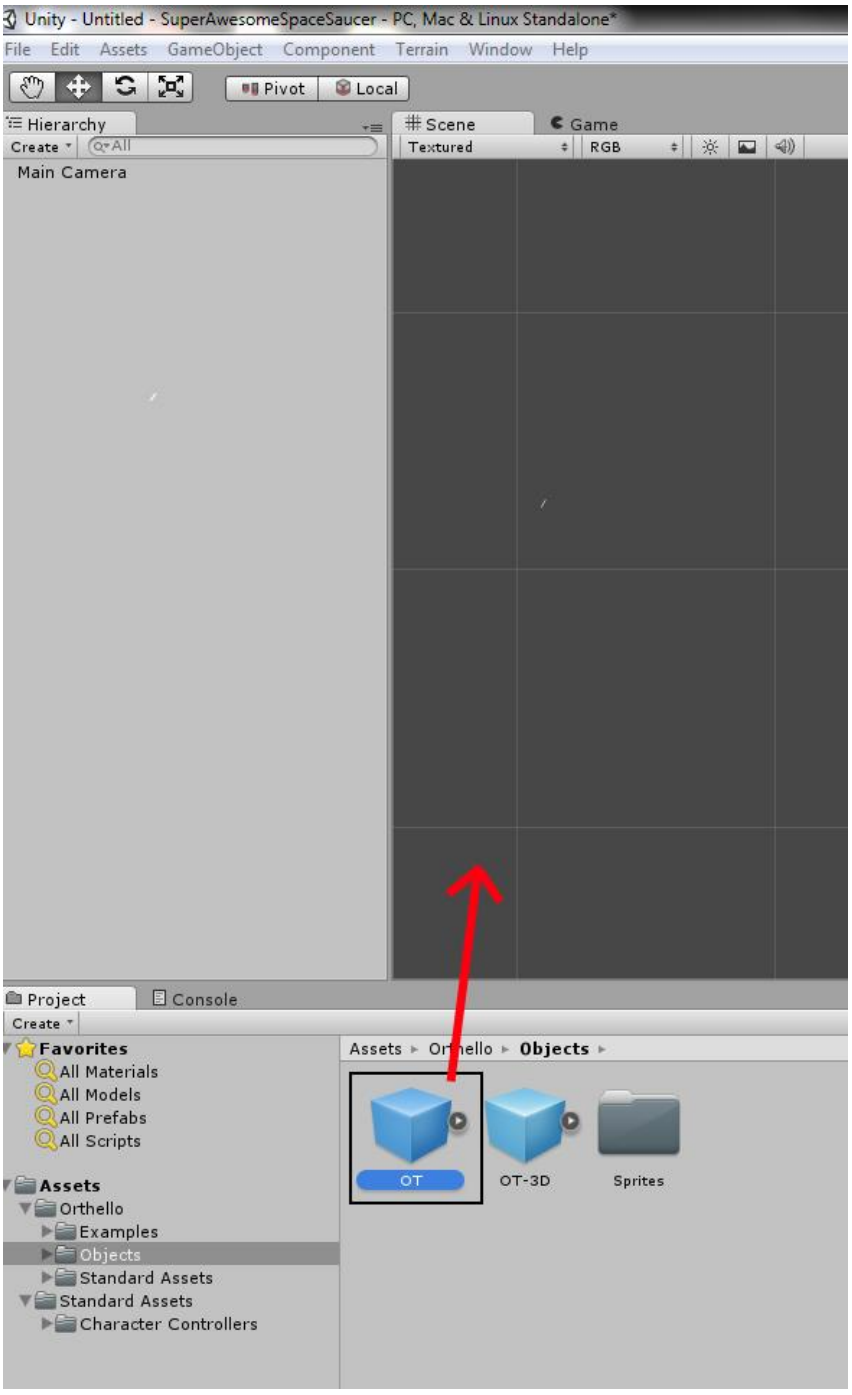
4.2 Orthello2d:n asennus ja käyttöönotto

Orthello2d asennetaan Unity3d:n oman Asset storen kautta. Asset store avataan Unityssä ”window”-valikon alta löytyvästä ”Asset Store”-kohdasta (tai pikakomennolla ctrl + 9). Kun aukeavan ikkunan hakukohtaan kirjoittaa orthello2d, löytyy useita vaihtoehtoja. Oikea vaihtoehto on nimeltään Orthello 2d Framework. Sitä täytyy klikata hiirellä ja painaa Download-painiketta. Unity pitää huolta varsinaisesta asennuksesta. Jos Orthellon haluaa ottaa käyttöön auki olevaan projektiin, pitää seuraavaksi aukeavasta valikosta vielä painaa Import-painiketta. Tämän jälkeen Orthello lisää kaikki tarvittavat tiedostot avoinna olleeseen projektiin.

Unityn projekti koostuu yhdestä tai useammasta scenestä. Yksi scene saattaa pitää sisällään esimerkiksi pelin päävalikon, jonkin tietyn tason, tai vaikka koko pelin. Näitä scenejä sitten ladataan peliin tarvittaessa. Jotta Unityyn saadaan luotua scene, jossa Orthello on käytössä, pitää seurata alla olevia vaiheita.

Ensin pitää luoda uusi scene valitsemalla ”File”-valikosta ”New Scene” vaihtoehto (tai vaihtoehtoisesta pikakomennolla Ctrl + N). Tällöin Unity luo tyhjän scenen, joka ei sisällä mitään muuta kuin ”Main Camera”-objektin. Tämä objekti on se kamera minkä kuva näkyy pelaajalle, ellei toisin määritellä. Main Cameran asetuksista ei tar-

vitse Orthelloa käytettäessä välittää, sillä Orthello laittaa ne kuntoon automaattisesti. Seuraavaksi sceneen pitää lisätä OT-objekti. Tämä objekti löytyy tiedostopolusta {juuri}/Orthello/Objects. Objektin lisääminen sceneen tapahtuu yksinkertaisesti raahaamalla se hiirellä avoinna olevaan scene-ikkunaan, kuten kuvassa 7.



Kuva 7. OT-objekti pitää raahata sceneen, jotta Orthello toimisi.

Kun OT-objekti on lisätty sceneen, se on nähtävissä scene hierarchy -ikkunassa. Kun kyseisessä ikkunassa OT-objektin päältä painaa hiirellä, avautuu sen alle kaikki sen lapsiobjektit. Näitä lapsiobjekteja ovat Animations, Containers, Prototypes ja View.

Animations-objekti on säilytyspaikka kaikille animaatio-objekteille, joita sceneen lisätään. Containers-objektiin lisätään scenen kaikki sprite sheetit. View-objektilla hallitaan kameran näkymää. Kyseisellä objektilla voi mm. liikuttaa, pyörittää ja zoomata kameraa tarvittaessa. (7)

Näiden vaiheiden jälkeen Orthello on valmis käyttöä varten kyseisessä scenessä. Opinnäytetyön peliprojektiakin varten käytiin läpi nämä vaiheet, ja luotu scene tallennettiin nimellä demoscene.

5 PERUSTEITA UNITYSTA

Jotta pelin toiminnallisuutta olisi helpompi ymmärtää, on hyvä tutustua hieman tarkemmin Unity3d:n perustoiminnallisuuteen. Siksi tässä osassa opinnäytetyötä kerrotaan hieman Unityn perusteista, kuitenkaan menemättä liian syvälle.

5.1 Unity3d:n perustoiminnallisuudesta

Unity3d on komponenttipohjainen ympäristö, mikä on huomattavissa lähes joka tasolla. Kuten opinnäytetyössä on jo aiemmin mainittu, koostuvat Unity3d:n projektit useasta pienemmästä kokonaisuudesta, joita kutsutaan sceneiksi. Nämä scenet ovat siis projektin komponentteja, jotka voi ladata käyttöön eri tilanteissa. Scenen vaihto onnistuu lähes milloin tahansa, mutta se kannattaisi suorittaa sellaisessa tilanteessa, ettei pelaaja sitä huomaa – esimerkiksi jo hetki ennen senhetkisen tason loppumista, sillä uuden scenen lataamiseen voi kulua hieman aikaa ja jos lataamisen aikana ei tapahdu mitään, voi pelaaja ihmetellä onko peli mennyt jumiin. (1)

Komponenttipohjaisuus näkyy myös scenen sisällä. Scenet koostuvat lukuisista eri peliobjekteista, joihin voidaan liittää eri komponentteja. Objektiin liitettäviä komponentteja ovat muun muassa 3d-malli, fysiikkaobjekti, joka mahdollistaa esineen törmäystarkistukset, sekä skripti, joka ohjaa objektin toiminnallisuutta. (8)

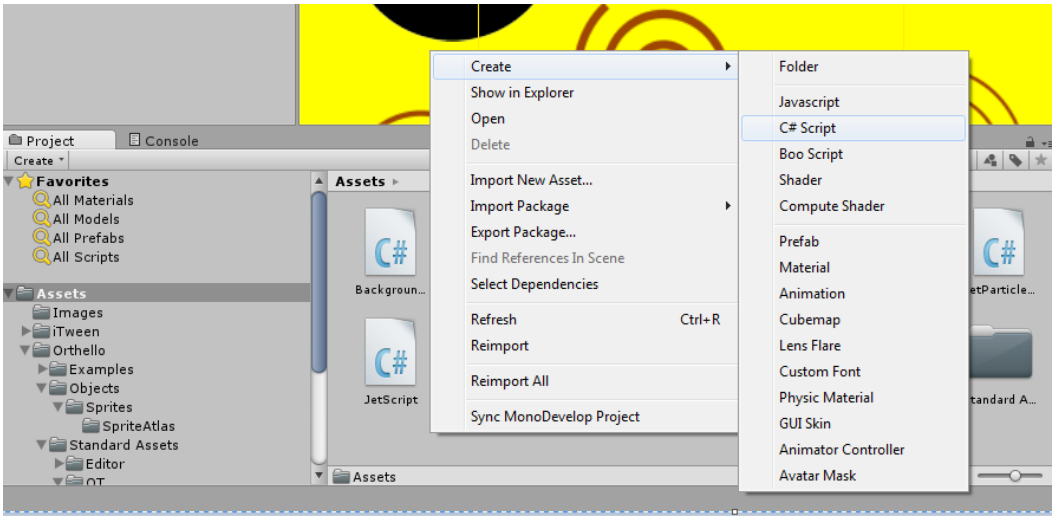
Jokaisesta objektista voi myös luoda oman prefabin, joka tallentuu oletuksena projektin Assets-kansioon. Kun tämän prefabin sitten lisää avoinna olevaan sceneen, muodostuu objektista täsmällinen kopio. Tätä ominaisuutta voi käyttää hyväksi niin, että usein tarvituista objekteista luodaan prefab, jolla on jo valmiiksi kaikki tarvittavat komponentit ja arvot. Tämä nopeuttaa huomattavasti uusien tasojen luontia. Lisäksi

prefab-objektien käyttö helpottaa useiden samanlaisten peliobjektien muuttamista, sillä kun tallennetun prefabin arvoja muuttaa, muuttuvat kaikkien sen kopioiden arvot samalla. (2)

5.2 Unity3d:n skriptauksen perusteista

Unity3d:ssä jokaiseen peliobjektiin on mahdollista liittää erillinen skriptitiedosto. Nämä skriptitiedostot kirjoitetaan jollakin Unityn tukemista ohjelmointikielistä (C#, Javascript ja Boo) ja ne kertovat peliobjektille mitä se milloinkin tekee, ja miten se käyttäytyy eri tilanteissa.(9)

Uuden skriptitiedoston voi luoda Unityssa painamalla projekti-ikkunassa hiiren oikeaa painiketta ja valitsemalla esiin ponnahtavista vaihtoehdoista Create, ja jonkin sen alta löytyvistä skriptityypeistä (kuten kuvassa 9.). Tämän opinnäytetyön skripteissä on käytetty C#-ohjelmointikieltä, joten valitaan C# Script. Tämän jälkeen skriptin voi vielä halutessaan nimetä uudelleen. Oletuksena skriptin nimi on NewBehaviourScript.



Kuva 9. Uuden skriptitiedoston luonti

Tämän jälkeen luodun skriptin voi avata esimerkiksi kaksoisklikkaamalla sitä hiirellä. Lisäksi skriptin voi liittää peliobjektiin raahaamalla ja tiputtamalla se halutun objektin components-välilehden päälle. Kun juuri luodun, muokkaamattoman skiptitiedoston avaa, näyttää se samalta kuin kuvassa 10.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class NewBehaviourScript : MonoBehaviour {
5
6      // Use this for initialization
7      void Start () {
8
9      }
10
11     // Update is called once per frame
12     void Update () {
13
14     }
15 }
16
```

Kuva 10. Skripti, johon ei ole vielä tehty mitään muutoksia

Skriptin kahdella ensimmäisellä rivillä määritellään mitä nimiavaruuksia käytetään, jotta niitä ei tarvitse määritellä erikseen joka kerta, kun niistä tarvitaan jotain (voidaan esimerkiksi merkitä suoraan MonoBehaviour eikä UnityEngine.MonoBehaviour).

Seuraavaksi rivillä 4 määritellään luokka joka perii MonoBehaviour luokan. MonoBehaviour on luokka, joka on UnityEngine nimiavaruudessa, ja skriptin tulee periä se, jotta voidaan käyttää Unityn valmiita ominaisuuksia ja metodeja peliobjektissa. Esimerkiksi Update-metodi ei toimi ilman MonoBehaviour luokan perintää (10).

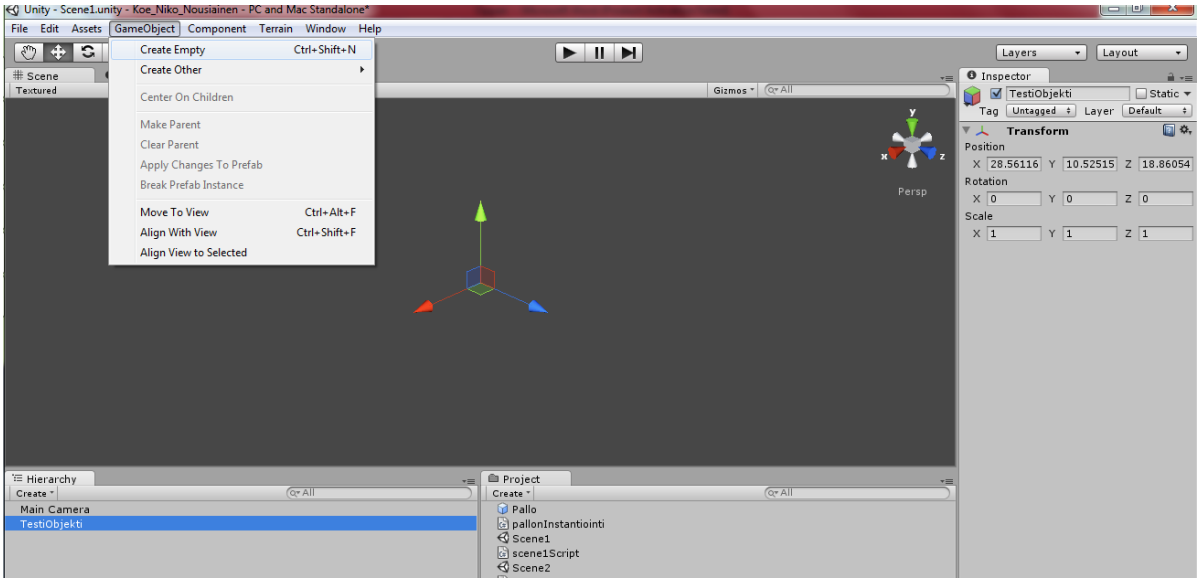
Seuraavaksi skriptissä ovat Start- ja Update-metodit. Nämä ovat ehkä Unityn tärkeimmät ja useimmiten käytetyt metodit. Start-metodissa olevat toiminnot suoritetaan kun objekti luodaan ja se suorittaa skriptinsä ensimmäisen kerran (11). Update-metodia kutsutaan kerran jokaisessa framessa. Tänne laitetaan usein suurin osa objektin toiminnallisuudesta, jos halutaan että se tekee jatkuvasti jotain (12).

Muita tärkeita Unityyn sisäänrakennettuja metodeja ovat mm. OnCollisionEnter, OnDisable, OnTriggerEnter jne. Metodeista kerrotaan tarkemmin sitä mukaa, kun niitä opinnäytetyössä käytetään.

5.3 Esimerkkiobjektin ja sen komponenttien luonti

Esimerkin vuoksi tässä kappaleessa näytetään, miten Unityssä luodaan objekti, kuinka objektiin lisätään komponentteja ja kuinka siitä tehdään prefab.

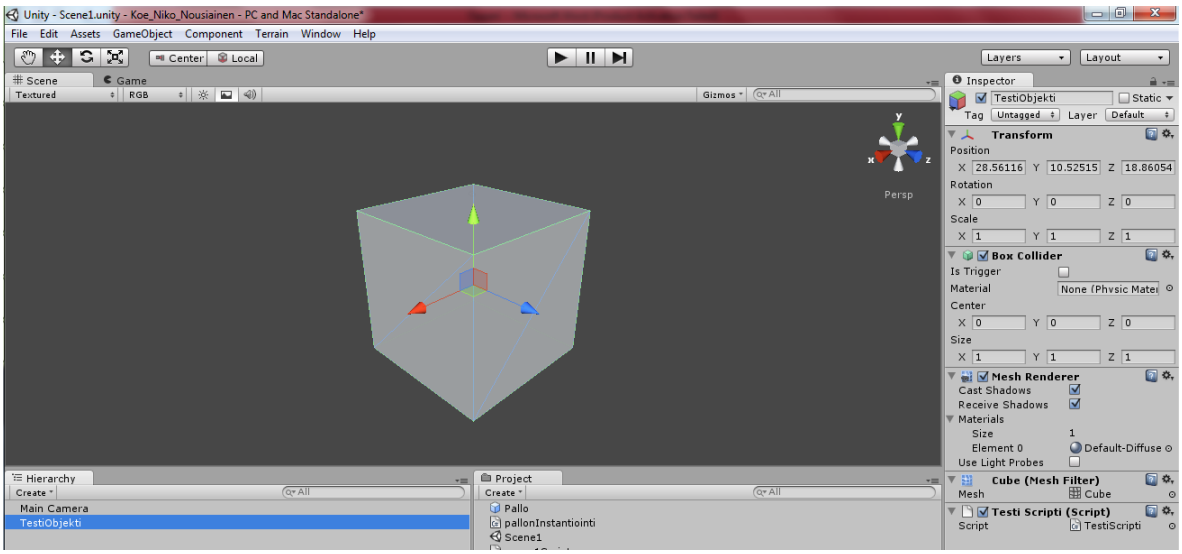
Aluksi luodaan tyhjään sceneen uusi peliobjekti. Tämä onnistuu valitsemalla ylävalikon ”GameObject”-kohdasta Create Empty (tai pikakomennolla CTRL + Shift + N). Tämä luo avoinna olevaan sceneen tyhjän objektin nimellä GameObject. Kuvassa 10. on näin luotu objekti, jonka nimeksi on annettu TestiObjekti.



Kuva 10. Kohdasta Create Empty pystyy luomaan uuden objektin sceneen. Hierarkia ikkunassa näkyy luomani uusi objekti: TestiObjekti

Jotta TestiObjekti ei olisi vain tyhjä peliobjekti, siihen lisättiin ensin Box Collider, joka luo sille kuution muotoisen fysiikkatarkastelun ja tämän jälkeen lisättiin vielä Mesh Filterin ja Mesh Rendererin. Mesh Filter on se komponentti, joka tuo halutun meshin assets-kansiosta Mesh Rendererille, ja Mesh Renderer piirtää sen näytölle valitulla materiaalilla. Komponentit lisättiin ”Components”-ylävalikosta.

Jotta objektin voisi nähdä, täytyy Mesh Renderer komponenttiin lisätä vielä materiaali (esimerkissä valittiin materiaaliksi ”Default-Diffuse”), sekä Mesh Filtteriin piti valita meshi (esimerkissä ”Cube”). TestiObjektiin lisättiin vielä skripti, joka nimettiin TestiSkriptiksi. Tämän jälkeen TestiObjekti oli näkyvä kuutio, joka sisälsi useita komponentteja. Testiobjekti, sen komponentit ja niiden asetukset näkyvät kuvassa 11.



Kuva 11. Kuvassa näkyy keskellä TestiObjekti ja oikealla kaikki siihen liitetyt komponentit.

Jotta TestiObjektilla olisi vielä jotain toiminnallisuutta, muutettiin TestiScriptiä siten, että kuutiota pystyi liikuttelemaan eri suuntiin W, A, S ja D painikkeilla. Kuvassa 12. näkyy, miltä tämä C#-ohjelmointikielellä tehty skripti näytti MonoDevelop-ohjelmointiympäristössä. Skriptiin tehdyt muutokset näkyvät ja toimivat Unityssä heti tallennuksen jälkeen. Kuvassa vihreällä näkyvät kommenttikohdat kertovat mitä missäkin kohtaa koodia tehdään. Kuutiota ja sen liikuttelua voi kokeilla käytännössä painamalla Unityn pääikkunan yläosassa olevaa play-painiketta (nuoli-ikoni, näkyy myös kuvassa 11.). Tätä play painiketta painamalla Unity käynnistää avoinna olevan scenen, ja kaikkea scenen toiminnallisuutta voi kokeilla jo editorissa, ilman että mitään täytyy kääntää.

TestiScripti.cs*

TestiScripti ▸ Update ()

```
using UnityEngine;
using System.Collections;

public class TestiScripti : MonoBehaviour {

    // Tätä kutsutaan kun objekti luodaan
    void Start ()
    {

    }

    // Updatea kutsutaan kerran framessa
    void Update ()
    {
        // Jos painetaan W-näppäintä
        if(Input.GetKey(KeyCode.W))
        {
            this.transform.Translate(new Vector3((1.0f*Time.deltaTime),0.0f,0.0f));
        }
        // Jos painetaan S-näppäintä
        if(Input.GetKey(KeyCode.S))
        {
            this.transform.Translate(new Vector3((-1.0f*Time.deltaTime),0.0f,0.0f));
        }
        // Jos painetaan A-näppäintä
        if(Input.GetKey(KeyCode.A))
        {
            this.transform.Translate(new Vector3(0.0f,(-1.0f*Time.deltaTime),0.0f));
        }
        // Jos painetaan D-näppäintä
        if(Input.GetKey(KeyCode.D))
        {
            this.transform.Translate(new Vector3(0.0f,(1.0f*Time.deltaTime),0.0f));
        }
    }
}

```

Kuva 12. Tämä lyhyt esimerkkiskripti kertoo peliobjektille, että sen pitää liikkua x ja y akselilla pelaajan painaessa W, A, S tai D näppäimiä.

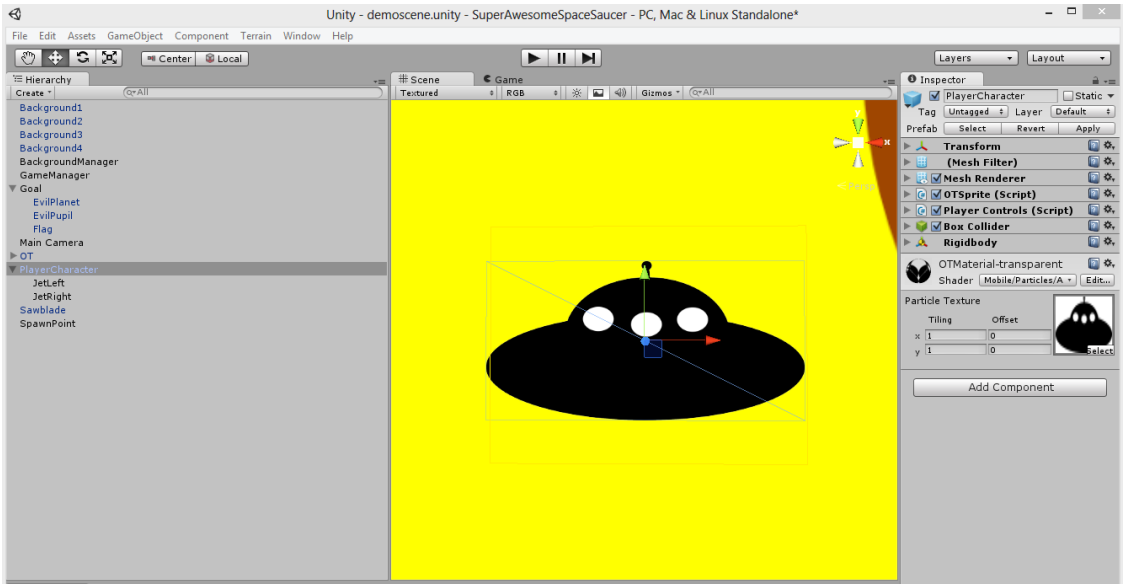
6 PELIN PROTOTYYPIN TOTEUTUS JA ORTHELLO 2D:N KÄYTTÖ

Tässä kappaleessa kerrotaan peliä varten luodun prototyypin toteutuksesta. Samalla näytetään esimerkkejä siitä, kuinka Orthello2d liitännäistä käytetään erilaisissa peliobjekteissa. Prototyyppiä lähdettiin rakentamaan jo aiemmin opinnäytetyössä luotuun demosceneen. Demoscenen tarkoitus on toimia testitasona, jossa on mahdollista kokeilla ja demonstroida pelin toiminnallisuutta ja eri ominaisuuksia. Seuraavaksi käydään läpi kaikki oleellisimmat demosceneen luodut peliobjektit, sekä niiden tärkeimmät komponentit.

6.1 PlayerCharacter ja OTSprite komponentin asetuksista

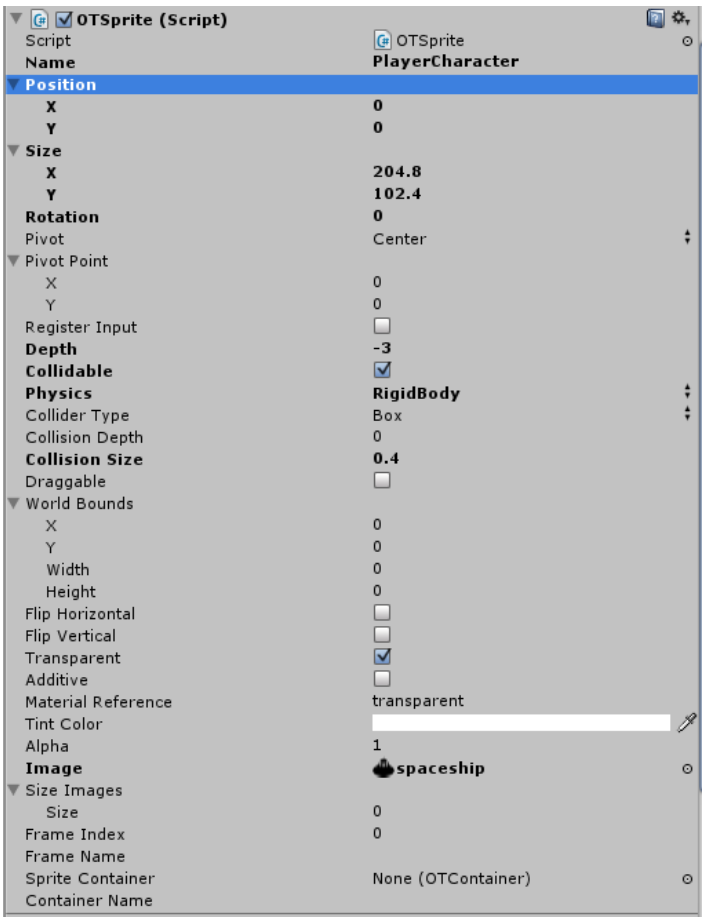
PlayerCharacter on koko demoscenen tärkein peliobjekti. Se on pelaajan ohjaama avaruusalus. Se sisältää seuraavat komponentit: Transform-, Mesh Filter-, Mesh Renderer-, Box Collider- ja Rigidbody-komponentit, sekä OTSprite- ja Player Controls-

skriptin. Lisäksi sillä on kaksi lapsiobjektia: JetLeft ja JetRight. PlayerCharacter peliobjekti ja sen komponentit näkyvät kuvassa 13.



Kuva 13. PlayerCharacter-peliobjekti ja sen komponentit (kuvassa oikealla).

Tämän työn kannalta tärkeimmät PlayerCharacterin komponenteista ovat OTSprite- ja PlayerControls-skriptit. OTSprite skripti on yksi Orthello2d:n mukana tulevista valmiista Sprite-komponenteista. Siitä löytyy kaikki tarvittavat asetukset toiminnallisen ja näkyvän kaksiulotteisen peliobjektin luomiseksi Orthello-sceneen. Nämä asetukset ovat nähtävillä kuvassa 14.



Kuva 14. OTSpriten asetukset.

OTSprite lisätään peliobjektiin raahaamalla se Assets/Orthello/Sprites hakemistosta minkä tahansa Orthello-scenessä olevan peliobjektin päälle. Näin kyseisestä peliobjektista voidaan helposti tehdä kaksiulotteinen sprite.

OTSprite-skriptistä voi muokata perusasetusten, kuten koon, nimen ja sijainnin lisäksi monia muita ehkä hieman vieraampia asioita. Depth-kentällä tarkoitetaan tässä sitä syvyystasoa, jolla objekti on muihin spriteihin nähden. Jos spriteillä on törmäystarkastuksia, vain samalla tasolla olevat spritet voivat törmätä toisiinsa. Collidable-kohdassa voidaan valita törmääkö objekti lainkaan muihin sprite objekteihin. Physics-kohdassa voidaan valita pelimoottorin tässä objektissa käyttämän fysiikkamallin tyyppi. Lisäksi tälle fysiikkamallille voidaan vielä erikseen valita perusmuoto (pallo tai laatikko) ja syvyys. Draggable-kohta määrittelee voiko objektiin tarttua esimerkiksi hiirellä. World Bounds -kohdassa rajataan alue, jolla peliobjektin on mahdollista scenessä liikkua. Kaikkien näiden asetusten lisäksi on tärkeää laittaa Image kohtaan se kuva, miltä peliobjektin halutaan näyttävän. Material Reference -kohdassa valitaan, mitä kuvan taustavärille tapahtuu. Prototyypin PlayerCharacter-objektissa kohtaan valittiin vaih-

toehto transparent (suom. läpinäkyvä), eli käytännössä kaikki Tint Color -kohdassa valittuna olevaa väriä olevat kohdat spriten kuvasta ovat läpinäkyviä scenessä. (13)

PlayerControls-skripti on kirjoitettu itse ja löytyy opinnäytetyön liitteenä (Liite 2). Skriptin alussa esitellään kaikki käytetyt muuttujat. Näistä mielenkiintoisin on ehkä julkinen GameObject tyyppiä oleva muuttuja explosion. Jotta skriptin sisällä päästään käsiksi muihin scenessä oleviin peliobjekteihin, on niitä varten tehtävä tämänkaltainen muuttuja. Tämä ei vielä yksin riitä, vaan muuttujalle on annettava editorissa se peliobjekti johon muuttujan halutaan viittaavan. Tämä tehdään Unityn editorissa kuvan 15. mukaisesti.

Kuva 15. Explosion muuttujalle on annettu arvoksi Explosion niminen prefab.

Tämän jälkeen koodissa pystytään luomaan Instantiate-funktion avulla kopioita olemassa olevasta Explosion prefabista. (14.) Esimerkiksi PlayerControls-skriptissä luodaan kyseisellä funktiolla kopioita Explosion prefabista pelaajan kuollessa.

Muuttujien esittelyn jälkeen PlayerControls-skriptissä on Start-funktio, jota kutsutaan kerran scenen käynnistyessä. Tässä funktiossa annetaan sellaiset arvot, joiden täytyy olla samat joka kerta kun PlayerCharacter-peliobjekti luodaan.

Seuraavaksi on PlayerControls-skriptin Update-funktio. Jokaisen skriptin Update-funktio käydään läpi kerran jokaisessa ruudunpäivityksessä. Tämän kyseisen skriptin Update-funktiossa pidetään huolta pelihahmon liikkumisesta (pelaajan syöte luetaan Unityn oman input rajapinnan avulla (15.)), pelinsisäisestä painovoimasta, pelihahmon törmäyksentarkistuksesta, sekä pelihahmon kuolemasta ja uudelleensyntymisestä. Uudelleensyntymiskohtaa merkitään demoscenessä erillisellä, tyhjällä SpawnPoint peliobjektilla.

Tämän lisäksi PlayerControls-skriptissä on vielä kaksi itse lisättyä funktiota: IsHit ja Respawn. Näitä kutsutaan vastaavasti silloin, kun pelaaja osuu vaaralliseen esteeseen ja kun pelihahmo syntyy uudestaan esimerkiksi kuolemisen, tai kentän udelleenaloittamisen jälkeen.

Kaksi PlayerCharacter-objektin lapsiobjektia, JetLeft ja JetRight, pitävät huolta kipinäsuihkuista, jotka syntyvät, kun pelaaja liikuttaa alusta. Ne on sijoitettu pelimaailmassa niihin kohtiin, joista kipinöiden tulisi lentää (aluksen vasen ja oikea alalaita) ja molemmat sisältävät saman, lyhyen JetScript-skriptin (Liite 3.). Tässä sciptissä luodaan erivärisiä kipinöitä aina, kun pelaaja aktivoi toisen, tai molemmat moottorit.

Skriptissä luodut kipinät ovat todellisuudessa scenen prototyyppeihin tallennettuja OTsprite tyyppisiä objekteja. OT-objektin Prototypes-lapsiobjektin alle on hyvä laittaa malli (eli prototyyppi) sellaisille spriteille, joita luodaan ajon aikana paljon. Niistä voi sitten luoda koodissa kopion seuraavanlaisella komennolla: ”OT.CreateSpriteAt(”PrototyypinNimi”, SijaintiVektori)”. Orthello pitää huolta siitä, ettei objekteja luoda liikaa, vaan hyödyttömät, jo kertaalleen luodut peliobjektit käytetään tarvittaessa uudestaan. Kuvassa 16. näkyy demoscenessä käytetyt prototyypit (aluksen räjähtäessä, sekä liikkuessa syntyvät kipinät) ja koodinpätkä jolla niitä voidaan luoda skriptissä.


```
OT.CreateSpriteAt("JetParticleOrange",new Vector2(this.transform.position.x,this.transform.position.y));
```

Kuva 16. Demescenen prototyypit ja esimerkki niiden käytöstä koodissa.

6.2 BackgroundManager

BackgroundManager on peliobjekti, joka pitää nimensä mukaisesti huolta pelin taustoista. Pelin tausta koostuu neljästä identtisestä OTSpriten sisältävästä peliobjektista (Background1, Background2, Background3 sekä Background4), joista jokainen on asetettu näyttämään kooltaan 2048*2048 pelimaailman yksikköä olevaa, toistensa kanssa identtistä taustakuvaa. BackgroundManager liikuttelee näitä taustakuvia niin, että lensipä pelaaja kuinka kauas hyvänsä, näkyy taustalla aina jotain. Näin ei tarvitse luoda uusia objekteja joka kerta, kun taustaa tarvitaan lisää, vaan voidaan käyttää hyväksi samoja peliobjekteja. Taustojen liikuttelu on toteutettu niin, ettei pelaaja sitä huomaa.

BackgroundManagerin ainut komponentti on BackgroundManagerScript (Liite 4.), joka on nimensä mukaisesti skriptitiedosto, jossa kaikki edellä mainittu toiminnallisuus toteutetaan. Skriptille täytyy määritellä editorissa neljä kuvan sisältävää peliobjektia, joita se käyttää taustoina. Demoscenessä nämä objektit ovat nimeltään Background1, Background2, Background3 ja Background4.

BackgroundManager myös liikuttaa taustoja jatkuvasti samaan suuntaan, kuin mihin PlayerCharacter-peliobjekti on menossa. Taustoja liikutetaan kuitenkin vain puolella siitä nopeudesta, millä PlayerCharacter-liikkuu. Tämä luo efektin jota kutsutaan pelinkehityksessä nimellä parallax scrolling. Lyhyesti selitettynä pelin taustat näyttävät pelaajalle liikkuvan hitaammin kuin pelihahmon kanssa samalla tasolla olevat objektit, jolloin pelaajalle tulee harhakuva syvyydestä (oikeasti kaksiulotteisessa maailmassa ei ole syvyyttä).

6.3 GameManager ja käyttöliittymän piirtäminen pelissä

GameManager-objekti pitää sisällään ainoastaan yhden, GameManagerScript nimisen komponentin (Liite 5.). Komponentti on nimensä mukaisesti skriptitiedosto, jossa kaikki GameManagerin toiminnallisuus tapahtuu.

GameManagerilla on demoscenessä kaksi tehtävää. Se pitää huolta tarvittavan käyttöliittymän (pelaajalle näkyvät nappulat, kuten esim. paussi- ja uudelleenaloitus-painike ja valikkotekstit) piirtämisestä näytölle, sekä pysäyttää tai lopettaa pelin tarvittaessa.

Pelaaja voi halutessaan pysäyttää pelin painamalla vasemmassa yläkulmassa näkyvää tauko-painiketta. Peli myös loppuu kun pelaaja osuu haitalliseen objektiin ja tuhoutuu, tai kun pelaaja saavuttaa maalin pelihahmolla. Kaikissa näissä tapauksissa GameManager pysäyttää pelin kellon niin, ettei mitään tapahdu ennen kuin pelaaja painaa pelaa- tai uudelleenaloitus-painiketta. Skriptissä Unityn sisäinen ajankulku voidaan pysäyttää komennolla Time.timeScale=0. Kun timeScale on asetettu nollaan, skriptien update-silmukkaa ei käydä läpi ollenkaan. Pelin saa taas käyntiin vastaavasti komennolla Time.timeScale=0.

Käyttöliittymän, josta käytetään pelinkehityksessä useammin termiä UI (tulee englanninkielen sanoista user interface), piirtämiseksi prototyypissä käytettiin Unityn omaa GUI (lyhenne sanoista graphical user interface) järjestelmää. Se mahdollistaa yksin-

kertaisten UI-elementtien, kuten painikkeiden ja tekstien piirtämisen ja toiminnallisuuden hallinnan. Unityssä UI-elementit piirretään aina OnGUI()-funktiossa. Kuvassa 17. näkyy esimerkki kolmesta OnGUI()-funktiossa piirretystä UI-elementistä (uudelleenkäynnistyspainike, menupainike ja level failed -teksti).


```
GUI.Label (new Rect(Screen.width/2-125, Screen.height/2-200, 250, 50), levelFailedTxt);
if (GUI.Button(new Rect(Screen.width/2-110,Screen.height/2-150,100,100), restartBtnTexture))
{
    timePlayed=0;
    playerCharacter.GetComponent<PlayerControls>().Respawn();
    isRunning=true;
}
if (GUI.Button(new Rect(Screen.width/2+10,Screen.height/2-150,100,100), menuBtnTexture))
{
}
```

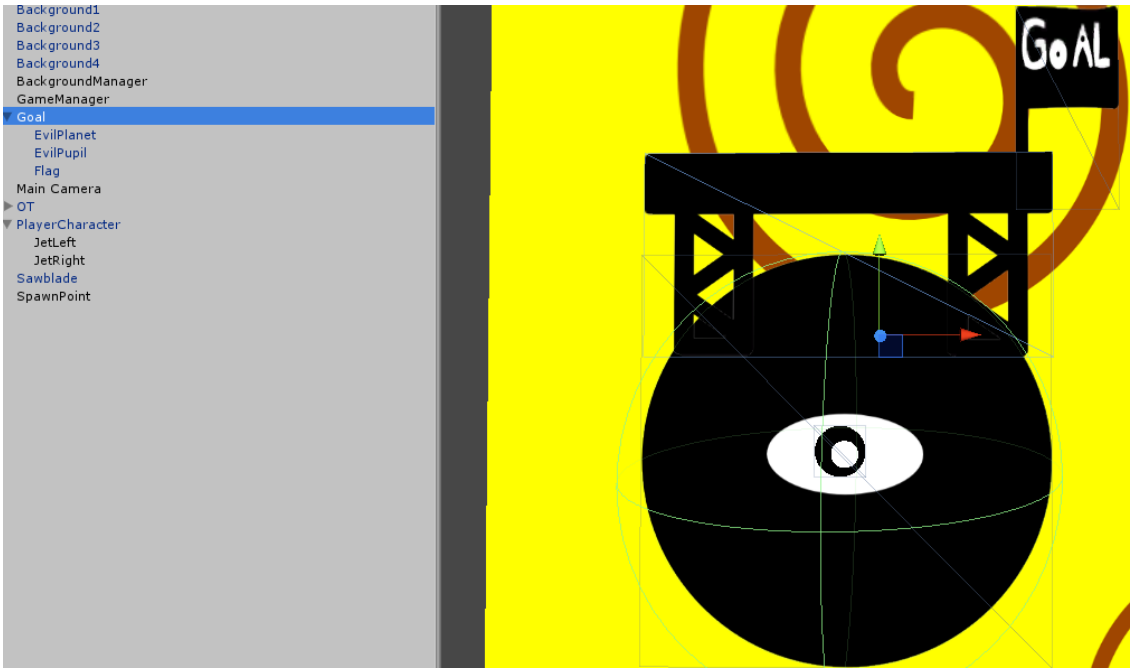
Kuva 17. Pelaajan kuollessa piirrettävät UI-elementit, sekä niiden piirtämiseen tarvittava pätkä skriptistä.

Kuten kuvassa 17, on demoscenessäkin käytetty ainoastaan kahdenlaisia UI-elementtejä: Label ja Button. Niitä voidaan käyttää yllä olevan kuvan esimerkin kaltaisesti. Kuvassa näkyvä Level Failed -teksti on oikeasti Label-elementti, jolle on määritelty sijainti, sekä näytettävä kuva (levelFailedTxt). Uudelleenkäynnistys ja valikkopainikkeet taas ovat Button-elementtejä. Ne luodaan if-lauseen ehtolauseessa ja kun niitä painaa, palauttaa painike if-lauseelle arvon true, jolloin suoritetaan if-lauseen koodi. Jos if-lauseen alla ei ole koodia, painike ei tee painettaessa mitään.

GameManager pitää huolta seuraavien elementtien piirtämisestä: Kun peli on käynnissä, vasemmassa ylänurkassa näkyy tauko-painike. Kun ollaan taukotilassa, näkyy vasemmassa ylänurkassa pelaa-painike, sekä ruudun keskellä olevat uudelleenaloitus-painike, valikko-painike ja ”Paused” -kuvan näyttävä Label. Valikko-painike ja uudelleenaloitus-painike näkyvät myös pelaajan kuollessa, tai maaliin tultaessa. Tällöin yläpuolella oleva Label on vastaavasti joko ”Level Failed” tai ”Level Complete”.

6.4 Goal ja sen lapsiobjektit

Goal-peliobjekti on maali, jolle pelaajan tulee yrittää laskeutua. Demoscenessä maali koostuu laskeutumisalustasta, maalilipusta, ”elävästä” planeetasta ja planeetan liikkuvasta pupillista. Todellisuudessa maalin eri osat ovat kuitenkin erillisiä peliobjekteja, joita käsitellään seuraavaksi. Goal ja sen lapsiobjektit näkyvät kuvassa 18.



Kuva 18. Goal ja sen lapsiobjektit scenessä.

6.4.1 Goal

Goal-peliobjekti on todellisuudessa vain planeetan päällä lepäävä laskeutumisteline. Se sisältää OTSprite-komponentin, jossa sille määritellään sen käyttämä kuva, sekä fyysiset ominaisuudet. Lisäksi se sisältää GoalScript-skriptin, jossa määritellään peliobjektin toiminnallisuus pelin maalina. Skripti tarkastelee jokaista goal-peliobjektiin törmäävää objektia. Jos törmäävä objekti sisältää PlayerControls-skriptin (ainoastaan pelaajan alus sisältää kyseisen skriptin), tarkastetaan vielä, että törmäävä objekti on tarpeeksi korkealla (näin varmistetaan, että vain laskeutumisalustan yläpintaa koskeva pelaaja saavuttaa maalin). Kuvassa 19. on GoalScript-skripti kokonaisuudessaan.

```
using UnityEngine;
using System.Collections;

public class GoalScript : MonoBehaviour {

    public bool playerHasReachedTheGoal=false;

    void OnCollisionEnter(Collision collision)
    {
        if(collision.collider.GetComponent<PlayerControls>()!=null && collision.collider.transform.position.y > this.transform.posi
        {
            playerHasReachedTheGoal=true;
        }
    }
}
```

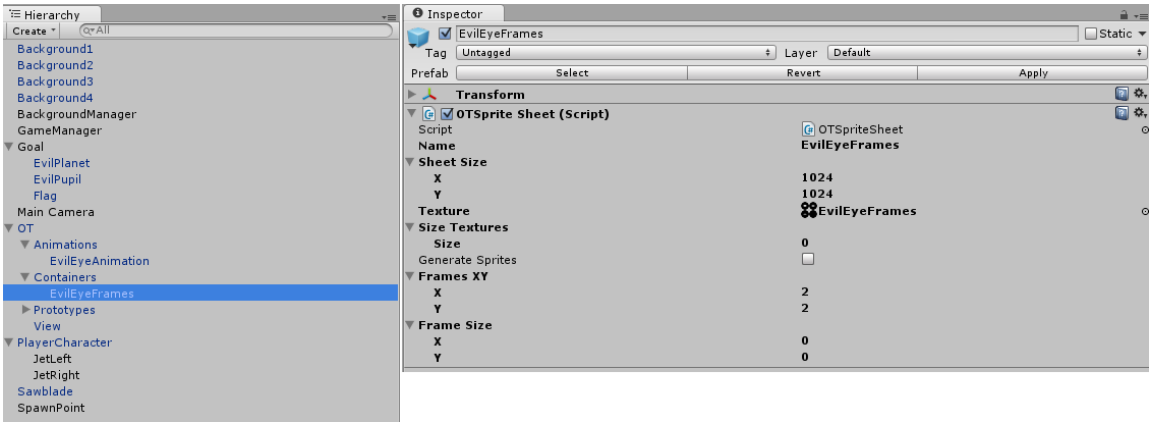
Kuva 19. GoalScript.

6.4.2 EvilPlanet ja animoidun spriten käyttö Orthello2d:ssä

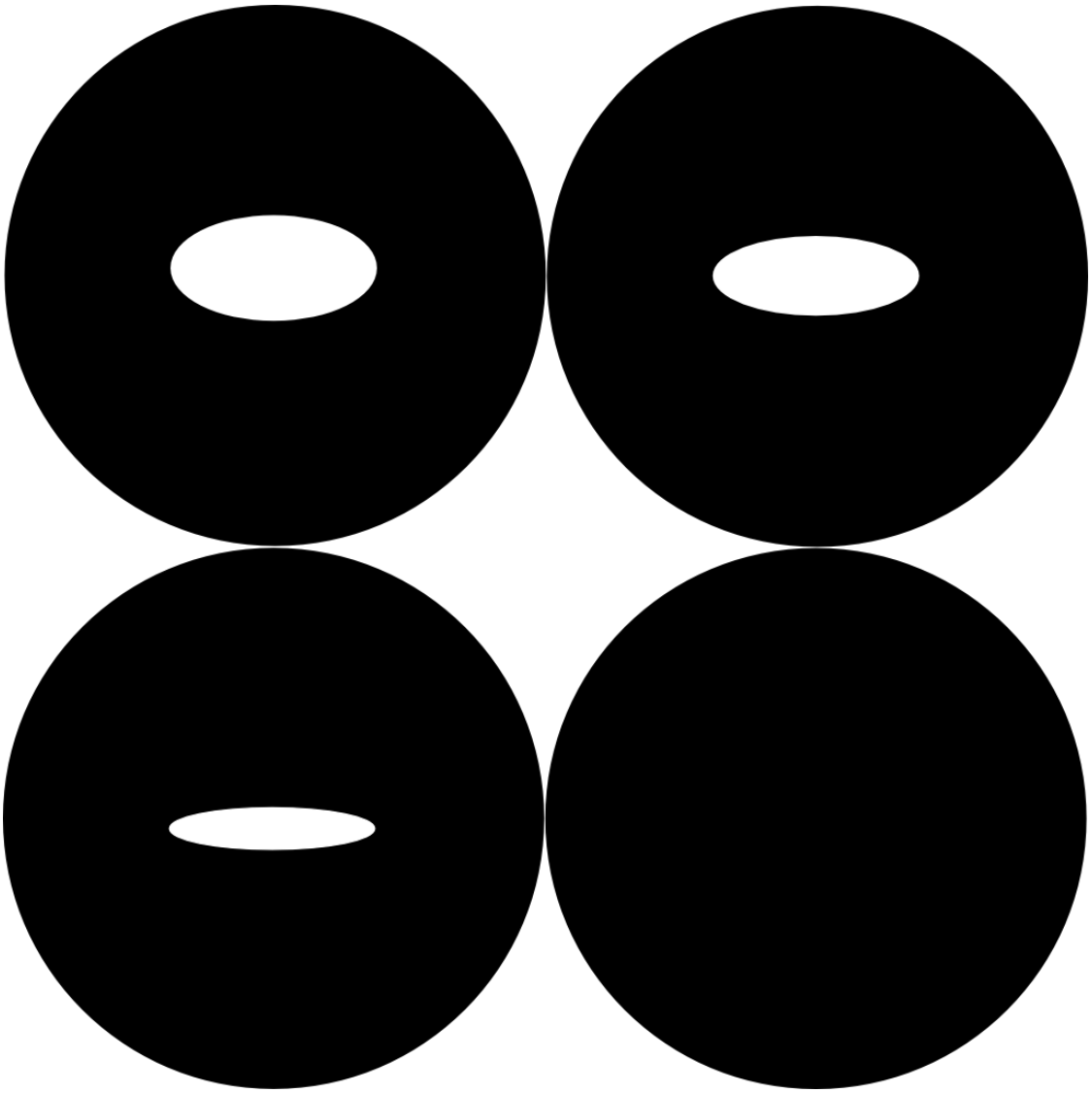
EvilPlanet on Goalin lapsiobjekti ja se on animoitu sprite. Jos halutaan, että spritekuva itsessään näyttää pelaajalle liikkuvalta (eli se on animoitu) pitää OTsprite komponentin sijasta käyttää Orthellon OTAnimatingSprite komponenttia (16.).

OTAnimatingSprite komponentin sisältävän peliobjektin voi luoda raahaamalla sceneen /Orthello/Objects/Sprites hakemistosta AnimatingSprite nimisen prefabin. Jotta AnimatingSprite-objekti toimisi, täytyy sille luoda myös tarvittavat OTSpriteSheet- ja OTAnimation-komponentit.

OTSpriteSheet sisältää nimensä mukaisesti sprite sheetin, joka on kokoelma samankokoisia kuvia, jotka on erotettu toisistaan tasaisesti. Näitä kuvia sitten käytetään animoidun spriten ruutuina, joita näytetään nopeasti peräkkäin ja näin käyttäjälle luodaan illuusio liikkeestä kuvassa. Sprite sheet luodaan raahaamalla sceneen /Orthello/Objects/Sprites hakemistosta SpriteSheet niminen prefab ja nimeämällä se uudestaan. Luotu peliobjekti tulee automaattisesti scenessä Containers-objektin lapsiobjektiksi. Containers-peliobjekti on taas puolestaan OT-objektin lapsiobjekti. Kuvassa 20. näkyy demoscenessä käytetyn sprite sheetin sijainti objektihierarkiassa, sekä sen asetukset. Demoscenessä EvilPlanetin animaatio käyttää EvilEyeFrames-nimistä sprite sheetiä, joka näkyy kuvassa 21.



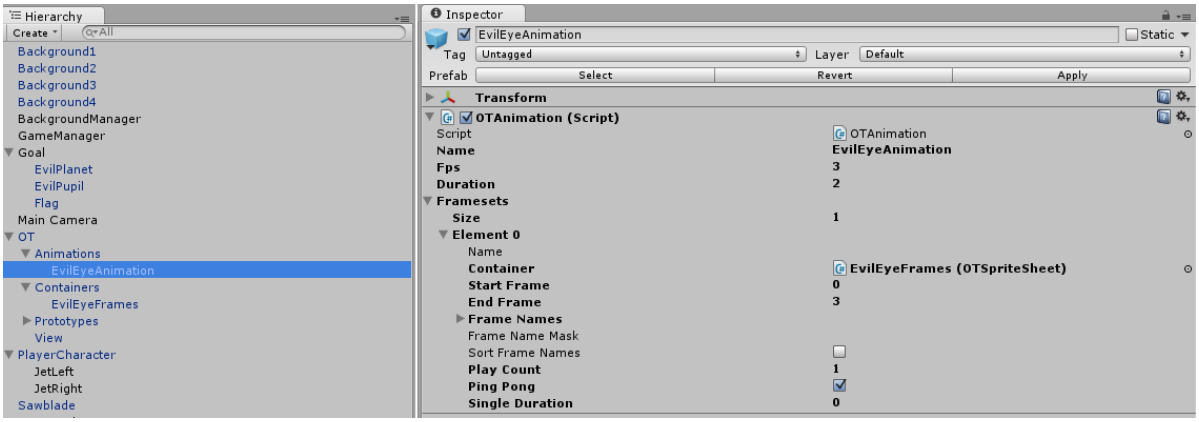
Kuva 20. EvilEyeFramesin sijainti objektihierarkiassa sekä asetukset.



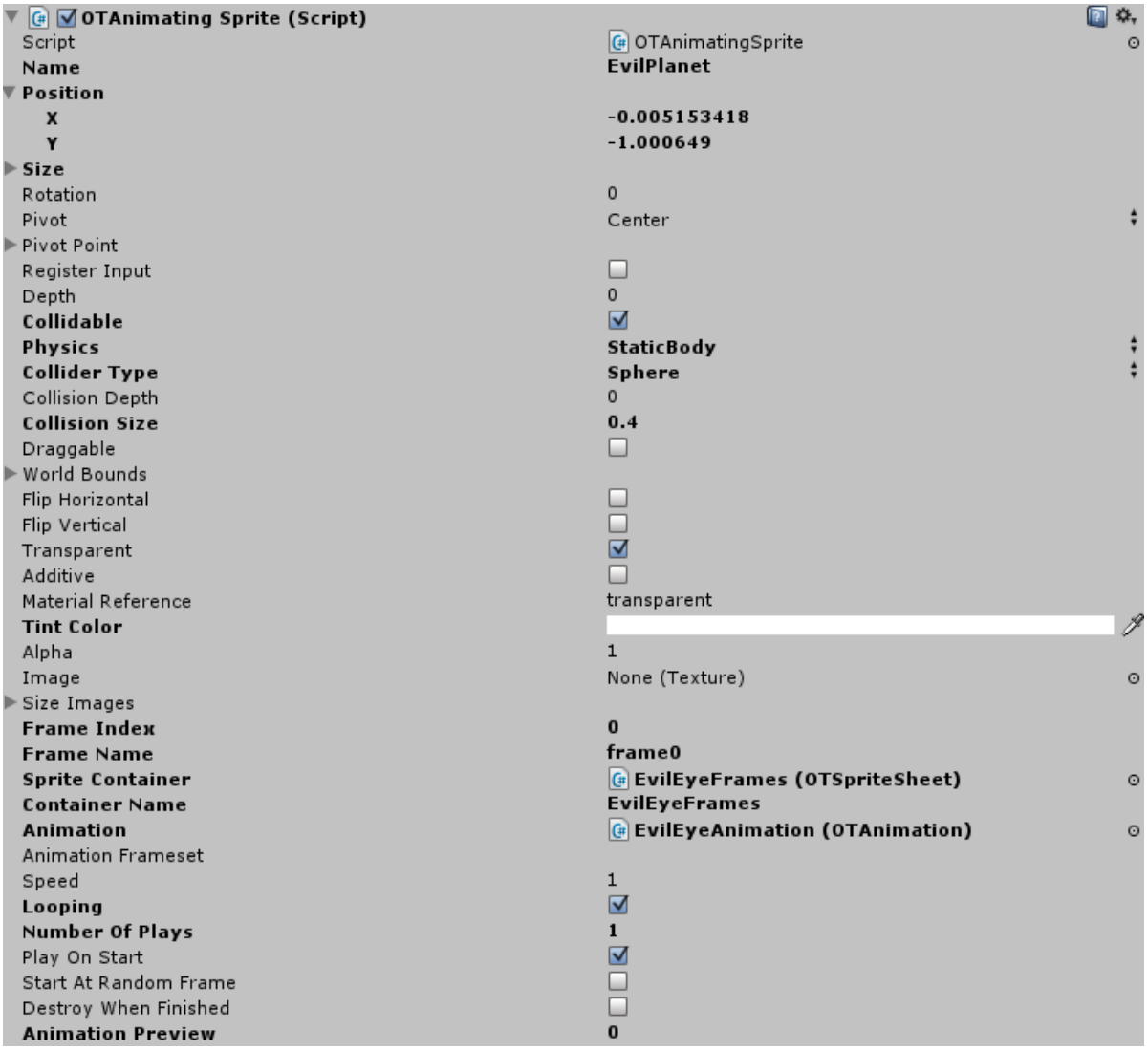
Kuva 21. EvilEyeFramesin käyttämä sprite sheet.

OTAnimation objekti lisätään sceneen raahaamalla /Orthello/Objects/Sprites hakemistosta Animation niminen prefab ja nimeämällä se uudestaan. Luotu peliobjekti tulee automaattisesti scenessä Animations-objektin lapsiobjektiksi. Animations-peliobjekti

on taas puolestaan OT-objektin lapsiobjekti. Demoscenessä EvilPlanetin animaatio käyttää EvilEyeAnimation-nimistä Animation-objektia. Kuvassa 22. näkyy EvilEyeAnimationin sijainti objektihierarkiassa ja sen asetukset. OTAnimation on käytännössä kokoelma asetuksia sille, miten halutun sprite sheetin kuvia käytetään animaatiossa.



Kuva 22. EvilEyeAnimationin sijainti objektihierarkiassa ja sen asetukset.



Kuva 23. OTAnimatingSpriten asetukset EvilPlanet-peliobjektissa.

OTAnimatingSpritessa on hieman enemmän asetuksia kuin jo aiemmin käytetyssä OTSpritessa. Asetukset näkyvät kokonaisuudessaan kuvassa 23. Näistä huomion arvoisimmat ovat Animation-kohta, johon laitetaan haluttu OTAnimation-komponentti, sekä SpriteContainer-kohta, johon laitetaan haluttu OTSpriteSheet-komponentti. Muita OTSpritestä puuttuvia asetuksia ovat mm. Speed (toiston nopeus), Looping (toistuuko animaatio loputtomasti) sekä Play On Start (animaatio toistetaan heti kun peliobjekti luodaan).

Kun kaikki tarvittavat peliobjektit oli luotu demosceneen ja asetukset oli laitettu kuvien osoittamalla tavalla kohdilleen, oli EvilPlanet-objekti animoitu. Animaatiossaan objekti räpyttelee toistuvasti kuvan keskellä olevaa ”silmää”.

6.4.3 EvilPupil

EvilPupil on ”elävän” planeetan silmän keskellä oleva, OTSprite-komponentilla toteutettu pupilli. Pupilli seuraa pelaajan liikettä, mikä luo vaikutelman siitä että planeetta seuraisi pelaajan alusta katseellaan. Pupillin liike on toteutettu PupilScript-skriptissä, joka näkyy kokonaisuudessaan kuvassa 24.

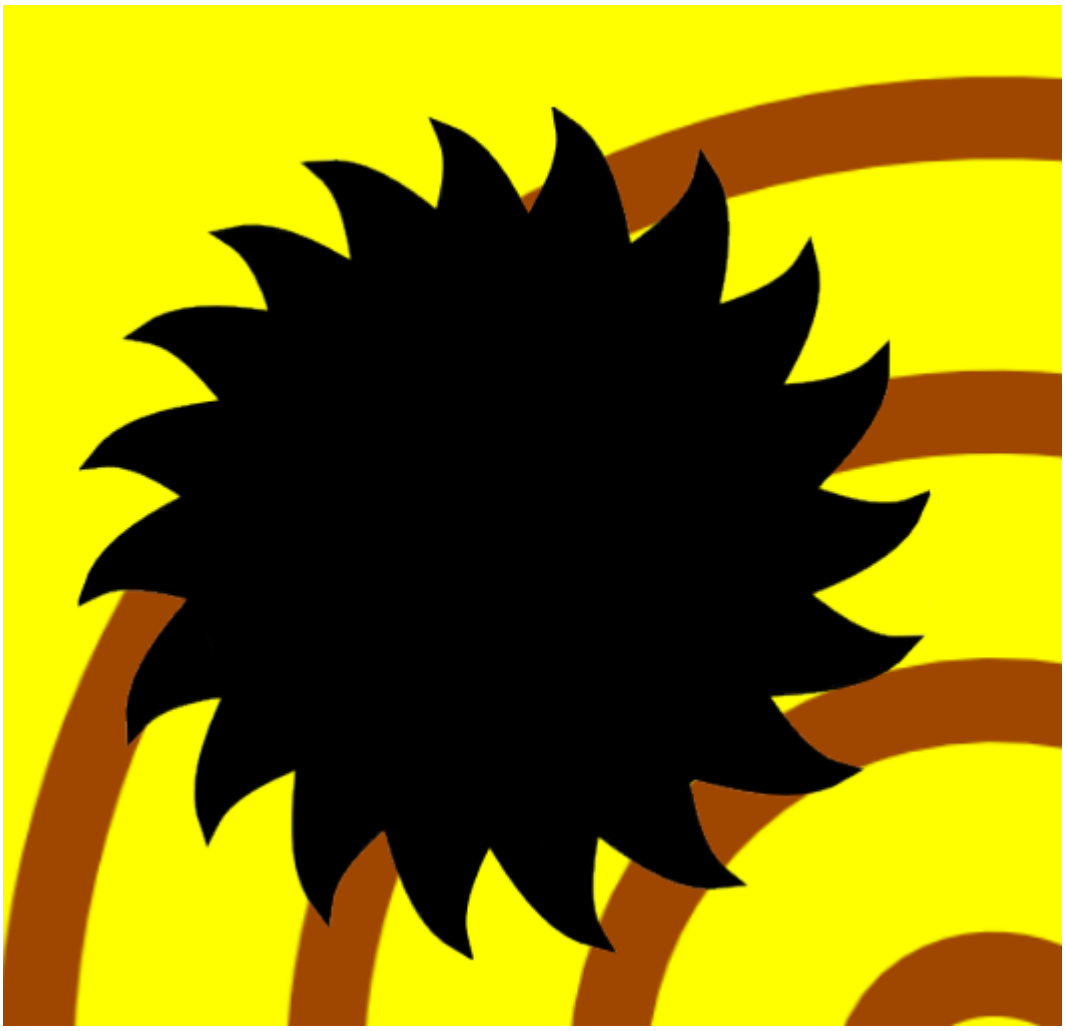
```
1  using UnityEngine;
2  using System.Collections;
3
4  public class PupilScript : MonoBehaviour {
5
6      private float range=20.0f;
7      public GameObject playerCharacter;
8      private Vector3 originalPosition;
9      private Vector3 playerDirection;
10
11     void Start ()
12     {
13         originalPosition=this.transform.position;
14     }
15
16     void Update ()
17     {
18         playerDirection = Vector3.Normalize(playerCharacter.transform.position-originalPosit
19         this.transform.position=originalPosition+playerDirection*range;
20     }
21 }
22
```

Kuva 24. PupilScript.

6.4.4 Flag

Flag on yksinkertainen objekti, jolla ei ole mitään toimintoa (koristeena toimimisen lisäksi). Se on maalin vieressä näkyvä maalilippu. Muista Goalin lapsiobjekteista poiketen pelaajan alus ei voi törmätä lippuun. Lippu on toteutettu OTSprite komponentin avulla.

6.5 Sawblade



Kuva 25. Sawblade peliobjekti.

Sawblade (näkyy kuvassa 25.) on ainut demoscenestä löytyvä pelaajalle haitallinen peliobjekti. Se on toteutettu OTSprite-komponentin avulla. Lisäksi siitä löytyy kaksi lyhyttä itse kirjoitettua skriptiä: Sawbladescript ja HarmfulObject. Sawbladescript on erittäin yksinkertainen ja pitää huolta ainoastaan siitä että sahanterä pyörii ympäri, jotta se näyttäisi pelaajalle uhkaavammalta. Sawbladecripti näkyy kokonaisuudessaan kuvassa 26. HarmfulObject puolestaan on skripti joka muuttaa jokaisen peliobjektin, johon se on kiinnitetty, pelaajalle haitalliseksi. Tällaiseen peliobjektiin osuminen tuhoaa aluksen kutsumalla PlayerControls-skriptin isHit()-metodia. HarmfulObject-skripti näkyy kokonaisuudessaan kuvassa 27.

```
using UnityEngine;
using System.Collections;

public class Sawbladescript : MonoBehaviour {
    private float rotationSpeed=125;

    // Update is called once per frame
    void Update ()
    {
        this.transform.Rotate(Vector3.forward*rotationSpeed*Time.deltaTime);
    }
}

```

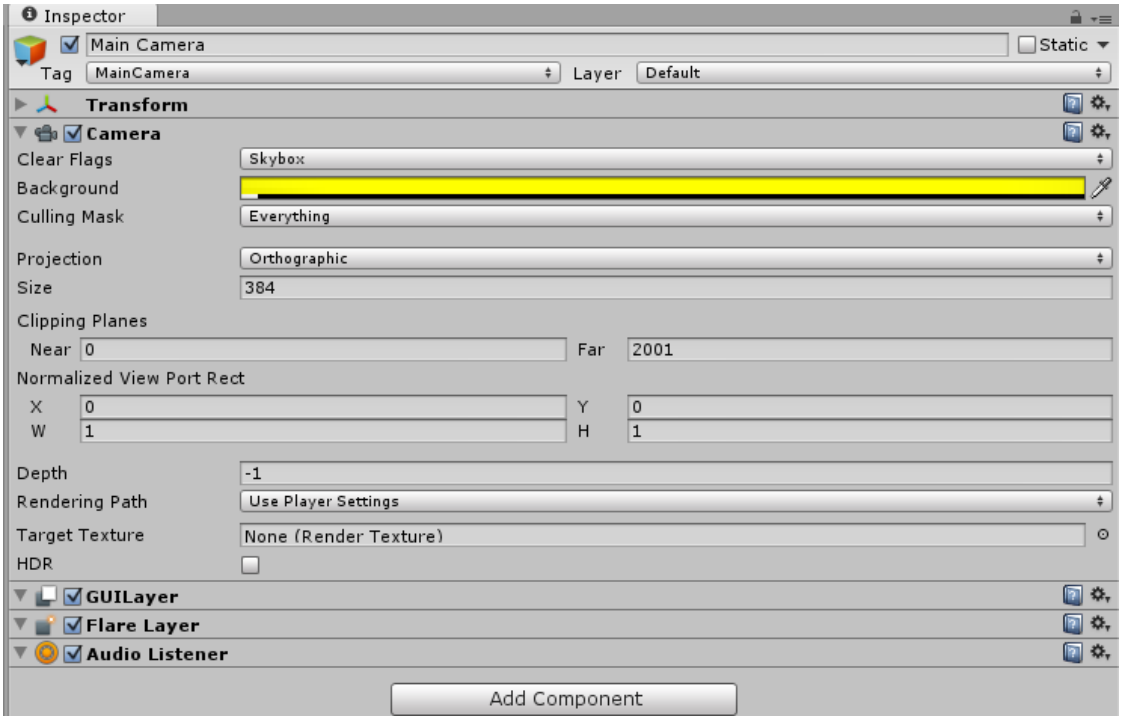
Kuva 26. Sawbladescript.

```
using UnityEngine;
using System.Collections;

public class HarmfulObject : MonoBehaviour {

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.GetComponent<PlayerControls>()!=null)
        {
            other.gameObject.GetComponent<PlayerControls>().IsHit(this.gameObject);
        }
    }
}

```

Kuva 27. HarmfulObject-skripti.

6.6 Main Camera

Main Camera -peliobjekti löytyy valmiiksi Unityn jokaisesta scenestä. Automaattisesti tämä on se kamera, jonka rajaama kuva näkyy pelaajalle. Kuvassa 28. näkyy Main Camera -peliobjektin asetukset opinnäytetyötä varten tehdyssä pelin prototyypissä.



Kuva 28. Main Camera -peliobjektin asetukset demoscenessä.

7 PELIN SIIRTÄMINEN IPAD-LAITTEELLE

Pelin siirtämiseksi iPad laitteelle tarvittiin hieman erilaiset työkalut, kuin mitä tähän mennessä oli käytetty (tähän mennessä käytössä on ollut pc-kone, jossa oli asennettuna Windows-käyttöjärjestelmä). Jotta unity peli voidaan kääntää iPadille, tarvitaan iPadin lisäksi jokin Applen työkoneista (Macbook tai iMac). Tässä työssä käytössä oli iPad 2 (jossa oli asennettuna iOS 7.0.4) ja iMac pöytäkone. iOS on Applen oma käyttöjärjestelmä yhtiön omille kosketusnäyttölaitteille.

7.1 Pelin siirtämiseen käytetyt ohjelmistot

Kääntämiseen käytetyllä iMac pöytäkoneella oli asennettuna OS X -käyttöliittymän uusin versio (10.9.1). Unity3d-ohjelmistosta asennettiin myös uusin versio iMacille (tekohetkellä 4.3.3f1). Unityn asennus Macille oli lähestulkoon vastaavanlainen kuin Windowsille (näytettiin opinnäytetyön kappaleessa 4.2), joten asennusprosessin yli hypätään opinnäytetyön selkeyden säilyttämiseksi.

Sovellukset iOS-alustalle käännetään Applen oman ohjelmointiympäristön Xcoden avulla. Xcoden uusimman version (opinnäytetyön tekohetkellä 5.0.2) saa ladattua il-

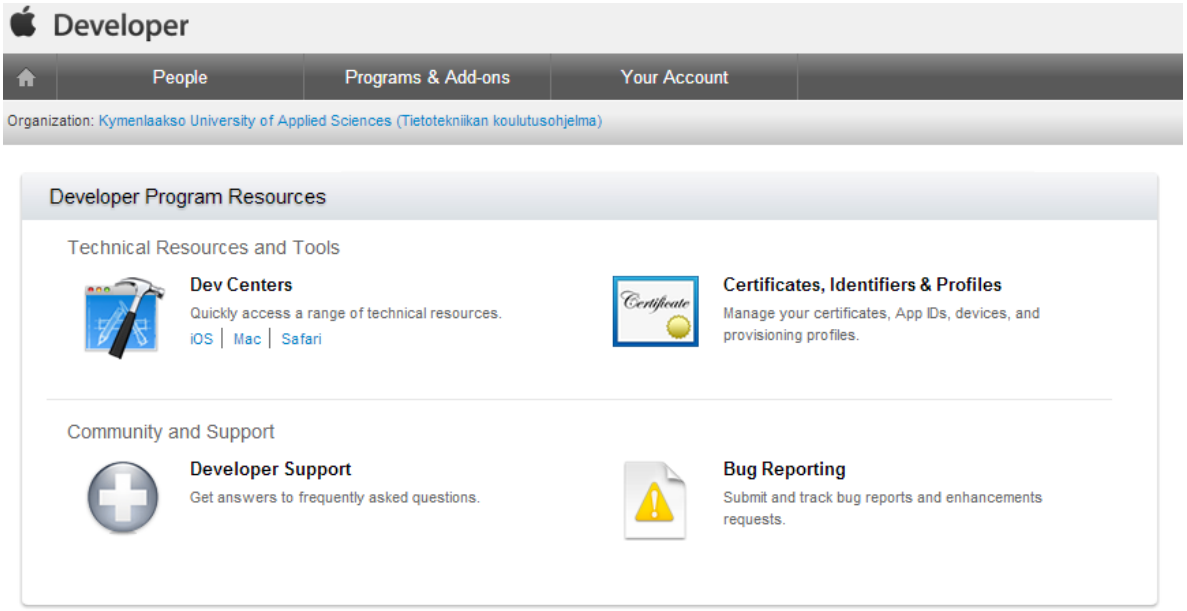
maiseksi Applen App Store -sovelluskaupasta. Xcodessa on sisäänrakennettuna mm. projektimanageri, sekä debuggerit ja kääntäjät useille eri ohjelmointikielille. Xcoden asennuksen jälkeen kaikki tarvittava oli asennettu.

7.2 Säännöstelyprofiilin luonti

Jotta prototyypin sai käännettyä, piti luoda tarvittava säännöstelyprofiili (englanniksi provisioning profile). Tämän luomiseksi täytyy olla vähintään yksi Apple kehittäjä-ohjelman jäsenyys. Jäsenyys maksaa tavallisesti 99 dollaria vuodessa. Opiskelija voi koulunsa kautta kuitenkin saada jäsenyyden opiskelunsa ajaksi ilmaiseksi. Säännöstelyprofiilin luomiseksi tarvitaan kolme vaihetta. Pitää rekisteröidä laite, luoda App ID (sovelluksen tunnus) ja hankkia tarvitut sertifikaatit. Kaikki yllämainitut asiat voi tehdä oman Apple kehittäjä -profiilin kautta (17.).

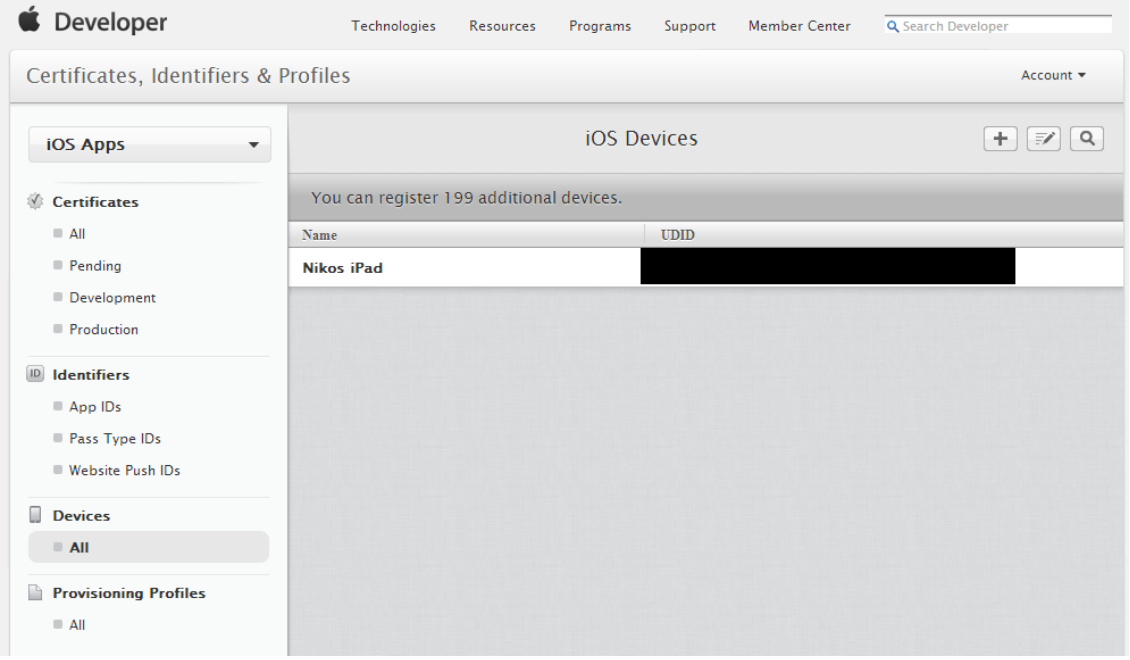
7.2.1 Laitteen rekisteröinti

Säännöstelyprofiilin luomista varten pitää ensin kirjautua omaan Apple kehittäjä -profiilin. Kirjautumisen jälkeen avautuvalla sivulla on neljä pääkohtaa: Dev Centers, Bug Reporting, Developer support ja Ceritficates, Identifier & Profiles. Näistä jälkimmäinen on ainoa mitä tarvittiin opinnäytetyössä. Kuvassa 28. näkyy kirjautumisen jälkeen avautuva sivu.



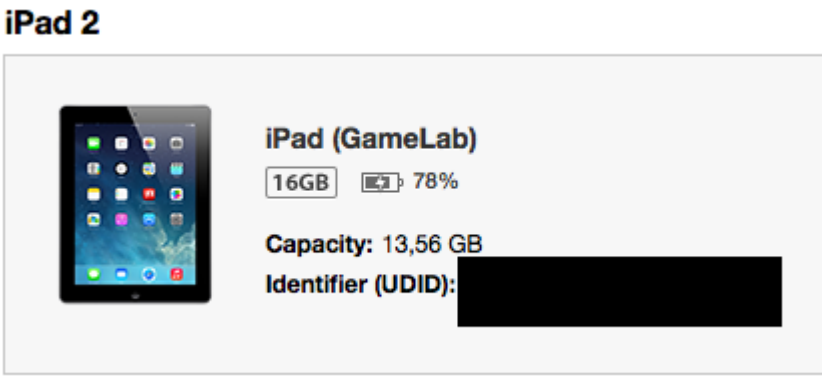
Kuva 28. Kuvassa on Apple Developer Member Center -sivuston pääkohdat.

Certificates, Identifiers & Profiles linkistä avautuvasta ikkunasta pitää valita kohta Devices (suom. laitteet). Avautuvassa ikkunassa (nähtävillä kuvassa 29.) näkyy kaikki sisään kirjautuneena olevaan tiliin liitetyt laitteet. Oikeassa yläkulmassa olevasta plus-painikkeesta saa lisättyä uuden laitteen.



Kuva 29. Apple Developer Member Centerin Devices -kohta.

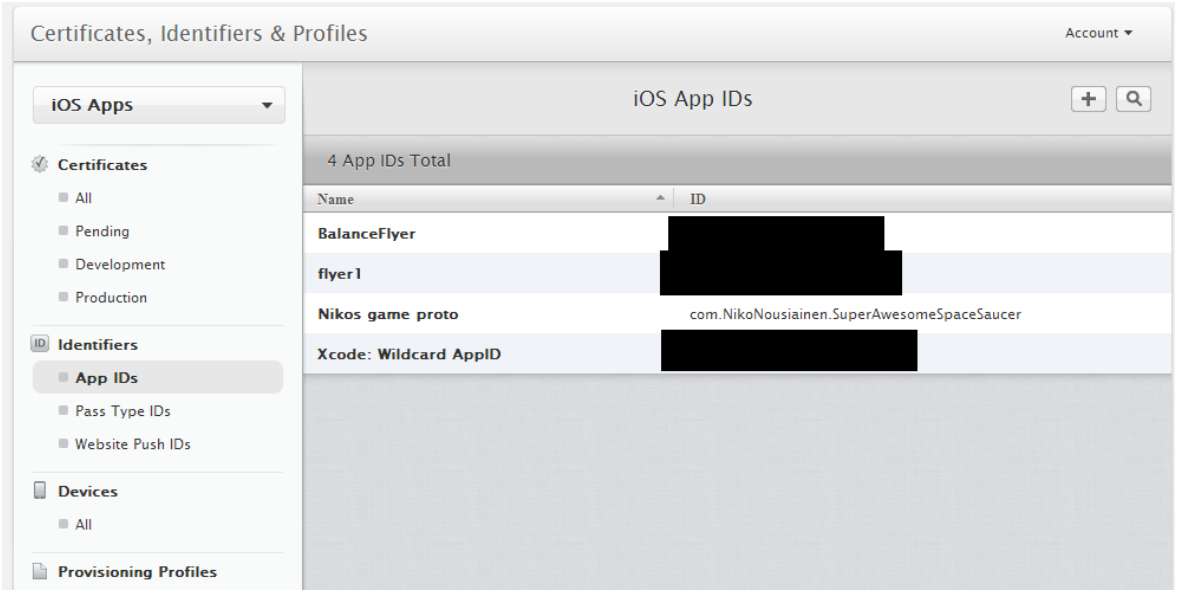
Kun tilille lisättiin uusi laite, piti täyttää kaksi kohtaa: nimi ja UDID (Unique Device Identifier). UDID on jokaiselle laitteelle uniikki tunnistuskoodi. Laitteen UDID-sarjan saa selville kiinnittämällä kyseinen laite kiinni esimerkiksi iMac pöytäkoneeseen ja avaamalla koneen iTunes ohjelmisto. iTunesin oikeasta yläkulmasta pitäisi löytyä painike jossa näkyy kiinni kytketyn laitteen nimi ja tyyppi. Tätä painiketta painamalla aukeavat laitteen tiedot. Tiedoissa näkyy kohta Serial Number. Kun tämän kohdalta klikkaa kerran, tulee näkyviin UDID, kuten kuvassa 30. Opinnäytetyötä varten loin laiteprofiilin nimellä Nikos iPad (näkyy kuvassa 29).



Kuva 30. UDID iTunesissa (tummennettu käytetyn laitteen suojaamiseksi).

7.2.2 App ID:n luonti

Seuraavaksi luotiin App ID käännettävälle sovellukselle (luotu pelin prototyyppi). Tämä tapahtui Certificates, Identifiers and Profiles ikkunan (kuva 29.) kohdasta App IDs. Uusi App ID luotiin painamalla oikeasta yläkulmasta löytyvää plus-painiketta. App IDs kohta on nähtävissä kuvassa 31.



Kuva 31. App ID -valikko.

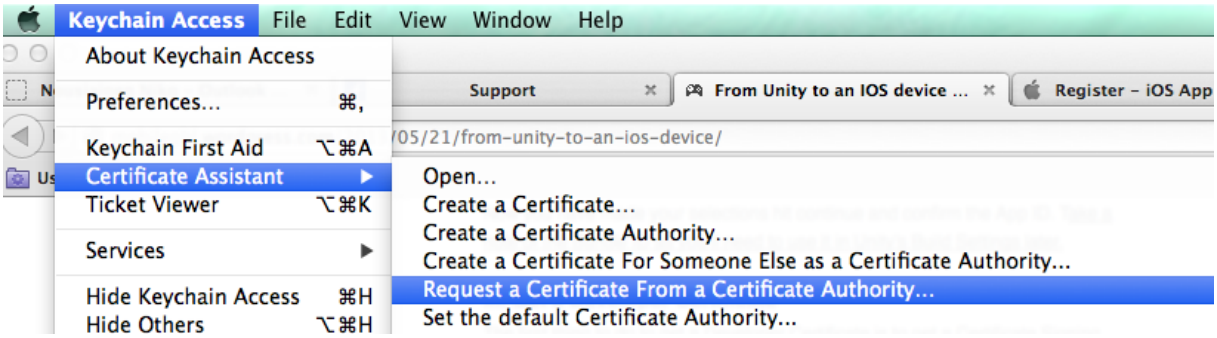
Uutta App ID:tä luodessa pitää ensin antaa sovelluksen kuvauksena toimiva nimi Name-kenttään. Tämä voi olla mitä vain, kunhan se auttaa muistamaan mihin kyseistä ID:tä käytetään. Tämän projektin nimeksi annettiin Nikos game proto.

Seuraavaksi asetetaan pakettitunniste (englanniksi Bundle Identifier). Pakettitunniste kirjoitetaan muotoon com.EtunimiSukunimi.PelinNimi. Tämän peliprojektin paketti-

tunniste oli siis com.NikoNousiainen,SuperAmazingSpaceSaucer. Sitten vielä valitaan mitä palveluita projekti pystyy käyttämään. Tähän projektiin jätettiin valituiksi vain pakolliset Game Center-toiminnot ja pelinsisäiset ostokset.

7.2.3 Kehittäjä sertifikaatin hankkiminen

Kehittäjä sertifikaatti hankittiin iMacilta löytyvän Keychain Access -ohjelmiston avulla. Kun Keychain Access on auki, valitaan Certificate Assistant ja Request Certificate From Certificate authority, kuten kuvassa 32.



Kuva 32. Sertifikaattipyynnön tekeminen.

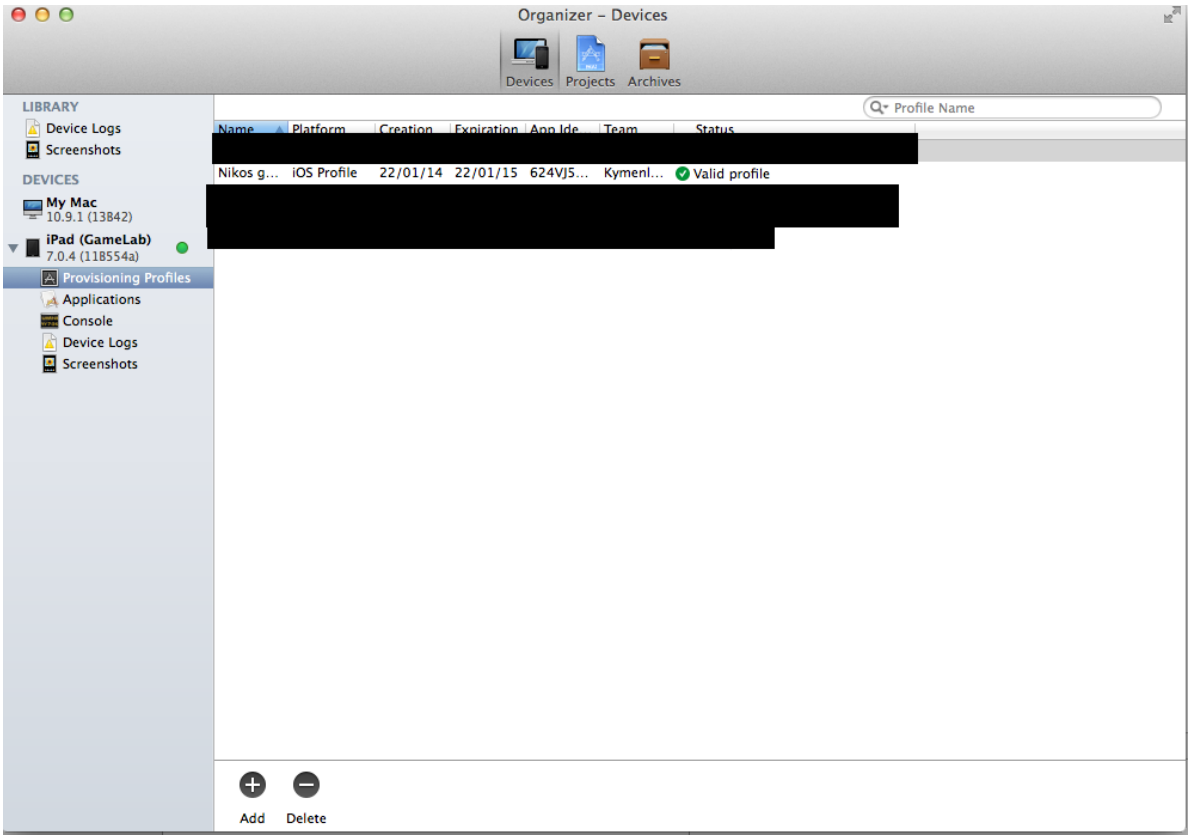
Avautuvaan ikkunaan annetaan samat tiedot, kuin mitä käytettiin Applen kehitysohjelmaan rekisteröityessä. Lisäksi pitää valita, että pyyntö tallennetaan kovalevylle. Tämän jälkeen ohjelma lataa .certSigningRequest tyyppisen tiedoston työpöydälle.

Tämän jälkeen palataan Apple Developer Member Centerin Certificates, Identifiers and Profiles kohtaan ja valitaan Certificates kohdasta Development. Jälleen oikeasta yläkulmasta löytyvästä plus-painikkeesta luodaan uusi sertifikaatti. Seuraavassa kohdassa pitää valita, minkä tyyppinen sertifikaatti halutaan. Peliprototyyppiä varten valittiin iOS App Development, koska kohdealusta oli iOS. Lisäksi pitää antaa aiemmin työpöydälle luotu .certSigningRequest tyyppinen tiedosto. Tämän jälkeen pystyy lataamaan kaksi uutta tiedostoa: WWDR sertifikaatin, sekä ios_developer.cer tiedoston. Tämän toimenpiteen jälkeen kyseiset kaksi sertifikaattitiedostoa löytyvät työpöydältä ja niitä pitää kaksoisklikata hiirellä, jotta ne asentuisivat Keychain Accessiin.

7.2.4 Säännöstelyprofiilin luonnin loppuvaiheet

Nyt kaiken pitäisi olla valmista säännöstelyprofiilin luomista varten. Jälleen Apple Developer Member Centerissä mennään Provision Profiles kohdan alta löytyvään All kohtaan. Uusi profiili luodaan oikeasta yläkulmasta löytyvän plus-painikkeen avulla. Tänne pitää valita aiemmin luodut App ID, laiteprofiili ja sertifikaatti. Säännöstely-profiilille pitää myös antaa nimi. Kun kaikki on valmista, säännöstelyprofiili ladataan työpöydälle. Säännöstelyprofiili on .mobileprovision tyyppinen tiedosto.

Säännöstelyprofiili pitää asentaa vielä XCode ohjelmistoon. Ensin pitää avata XCode, jonka jälkeen avataan Organizer-ikkuna Window ylävalikosta. Organizer ikkunasta, joka näkyy kuvassa 33, mennään kohtaan Provisioning Profiles. Tänne säännöstely-profiili lisätään alhaalta löytyvän pluspainikkeen avulla. Lisättävä säännöstelyprofiili on aiemmin työpöydälle tallennettu .mobileprovision tyyppinen tiedosto.

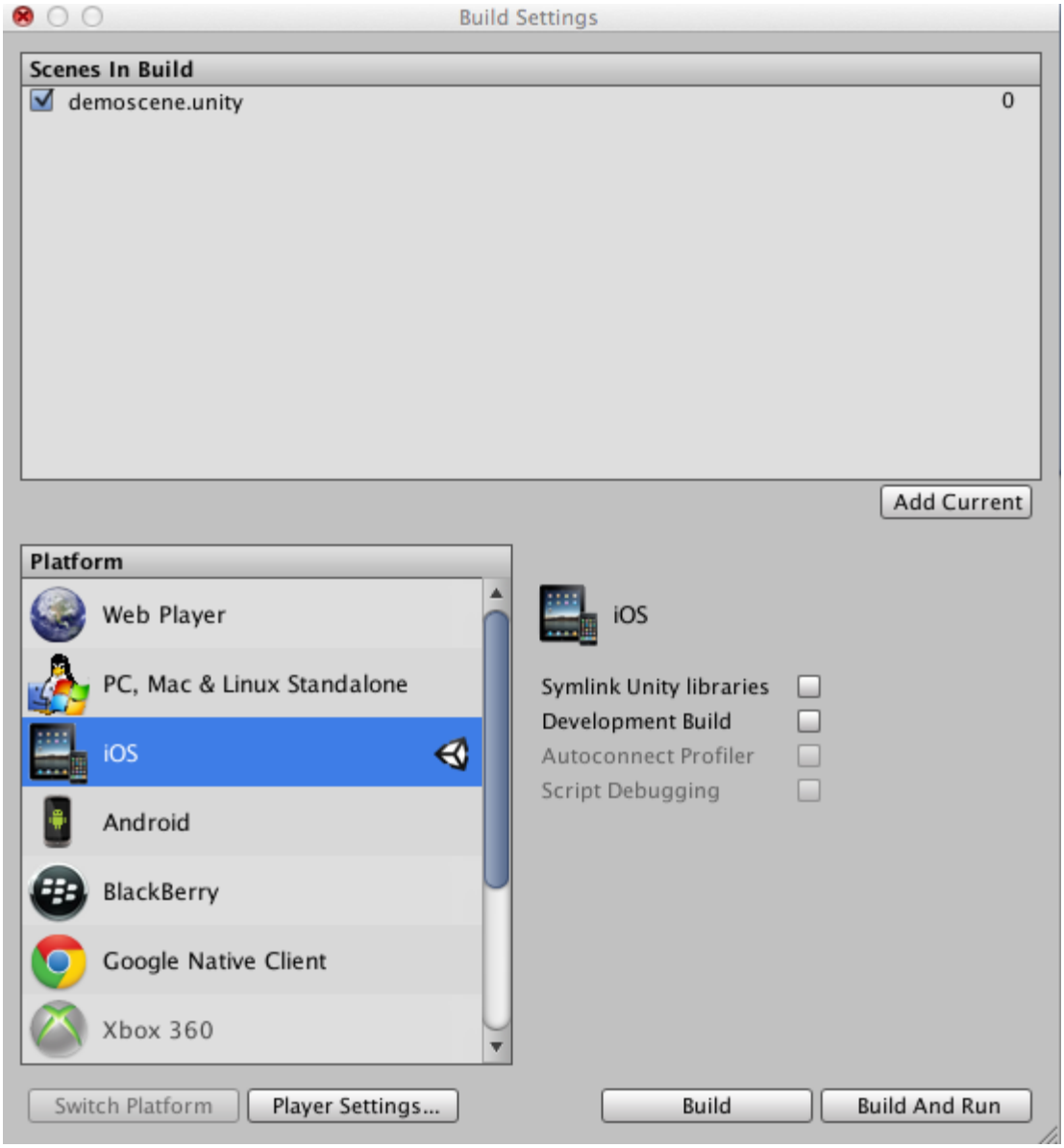


Kuva 33. XCoden Organizer ikkuna.

Organizer-ikkunan laitteet kohdassa on myös hyvä varmistaa, että laitteessa, jolle peli tai sovellus halutaan, on valittuna käyttö sovelluskehitykseen.

7.3 Unityn asetusten kuntoon laitto

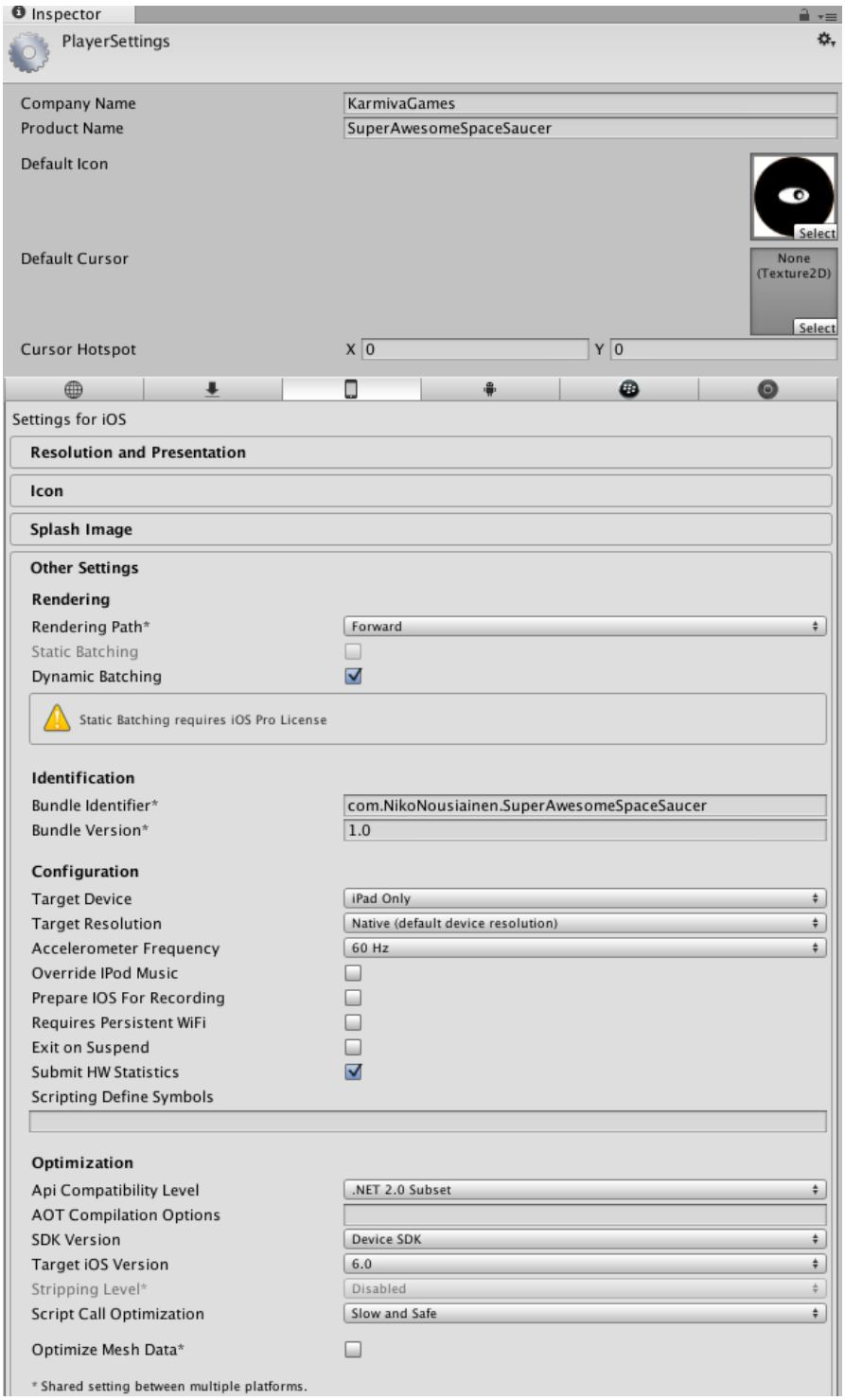
Ennen pelin siirtämistä iPadille, pitää vielä varmistaa että Mac-pöytäkoneelle asennetun Unityn asetukset ovat oikein. Tarvittavat asetukset löytyvät File-ylävalikon alta löytyvästä Build Settings kohdasta (näkyy kuvassa 34.).



Kuva 34. Unityn Build Settings -ikkuna.

Ennen kuin pelin kääntää, on hyvä varmistaa että halutut scenet ovat valittuina (tämän opinnäytetyön kohdalla valittuna oli vain aiemmin luotu demoscene). Pitää myös varmistaa, että valittuna on oikea kohdealusta (englanniksi platform) mille peli halutaan kääntää. Tässä työssä valittuna oli iOS.

Vasemmasta alakulmasta löytyvää Player Settings -painiketta painamalla aukeaa vielä lisää asetuksia. Prototyypin kääntämistä varten täältä muutettiin vielä muutamia asetuksia iOS asetuksien kohdalta (pieni iPad ikoni). Resolution and Presentation kohdan alta löytyvään Default Orientation -kohtaan vaihdettiin Landscape Left. Tämä tarkoittaa, että pelaajalle näytettävä kuva on aina horisontaalisesti, eikä se keikahda ympäri vaikka tablettia heiluttelisi kuinka. Other Settings kohdan alta löytyvään Bundle identifier kohtaan täytyi laittaa aiemmin määritelty pakettitunniste (com.NikoNousiainen.SuperAwesomeSpaceSaucer). Ilman tätä Xcode ei suostuisi kääntämään peliä laitteelle. Target Device -kohtaan valittiin haluttu kohdelaite (iPad) ja Target iOS Version -kohtaan valittiin se iOS versio, jolla pelin halutaan vähintään toimivan. Tähän kohtaan valittiin uusin mahdollinen, eli 6.0. Kuvassa 35 näkyy prototyypin kääntämisessä käytetyt asetukset.



Kuva 35. Build Settings valikosta aukeava Player Settings -ikkuna.

Nyt kaikki on valmista pelin kääntämistä varten. Ensin varmistetaan että iPad on kiinni Mac-pöytäkoneessa, jonka jälkeen Build settings ikkunasta painetaan Build and Run -painiketta. Tämä käynnistää automaattisesti myös Xcoden, joka suorittaa varsinaisen käännöstyön. Hetken kuluttua peli käynnistyy automaattisesti iPadilla ja on valmis kokeiltavaksi. Kuvassa 36 näkyy opinnäytetyön prototyyppi peli pyörimässä iPad2 laitteella.

Kuva 36. Peli pyörii ongelmitta iPadilla.

8 TULOSTEN TARKASTELU JA PÄÄTELMÄT

Tässä osiossa käydään lyhyesti läpi työstä opittua ja mieleen jäänyttä asiaa ja tehdään päätelmiä demoprojektin luoman kokemuksen pohjalta. Havainnot on lajiteltu huomionarvoisimpien asioiden mukaan nimettyjen otsikoiden alle.

8.1 Työkalujen valinnan tärkeys

Ehkä tärkein huomionarvoinen asia, joka demoprojektia tehdessä huomattiin, oli että hyvien ja tehtävään soveltuvien työkalujen valitseminen on erittäin tärkeää. Kun työkalujen valintaan käytetään heti alussa aikaa ja vaivaa, maksaa nähty vaiva lähes varmasti itsensä takaisin projektin myöhemmissä vaiheissa. Esimerkiksi ilman demoprojektissa käytettyä Orthello2D:tä oltaisiin kaikki 2d-objektien toiminnallisuudesta (aina piirtämisestä lähtien) tekemään itse, ja aikaa olisi mennyt huomattavasti enemmän. Ilman Unity3D:tä jouduttaisiin kehittämään mm. oma fysiikkamoottori ja projektien kääntäminen laitteelle vaikeutuisi huomattavasti. Vaikka joskus valmiiden työkalujen maksut tuntuvat liiallisilta, maksaa sama työ usein huomattavasti enemmän itse tehtynä.

Kannattaa myös valita ajan tasalla olevat uusimmat versiot käytetyistä työkaluista. Vaikka vanhemman version voi saada halvemmalla (tai ilmaiseksi), taataan uudella versiolla yhteensopivuus ajankohtaisten laitteiden ja muiden työkalujen kanssa. Jos jonkin työkaluista päivittää kesken projektin, saattaa osa tehdystä työstä lakata toimimasta. jolloin joudutaan tekemään turhaa työtä. Pahimmassa tapauksessa aiemmin tehty työ menee kokonaan hukkaan.

8.2 Teknologian nopea kehittyminen ja sen tuomat ongelmat

Tietotekniikka on tunnetusti ala, jonka työkalut ja tekniikka kehittyvät huimaa vauhtia. Projektin alussa valitut työkalut ja kohdelaitteet saattavat olla jo projektin lopussa vanhentuneita. Pelikehityksessä varsinkin uusien kohdealustojen valinnassa on se vaara, että alustan suosio on vain lyhytaikaista. Tällöin pelin valmistuessa alustalla ei ole enää käyttäjiä, eikä pelillä saavuteta haluttua yleisöä.

Tätä opinnäytetyötä tehdessä ongelma ilmentyi Unity3D:n nopean kehityksen myötä. Vain hetki opinnäytetyön aloittamisen jälkeen Unity julkaisi omat työkalunsa kaksiulotteisten pelien luontiin. Nämä työkalut tekivät Orthello2D:stä lähes tarpeettoman. Tämä heikensi myös motivaatiota Orthello2D:n opetteluun. Demoprojekti päädyttiin kuitenkin jatkamaan loppuun valituilla työkaluilla, sillä Unityn omat työkalut olivat vielä varhaisessa vaiheessa, ja saattavat sisältää puutteita, kun taas Orthello2D:tä on käytetty jo lukuisissa projekteissa ja se on todistettu hyväksi ja toimivaksi.

8.3 Suunnittelu ja saatavilla olevien resurssien arviointi

Sitä ei voi painottaa tarpeeksi, miten tärkeää on hyvä suunnittelu, ja erityisesti se että saatavilla olevat resurssit (mm. aika, raha, käytössä oleva osaaminen ja miestyövoima) otetaan jo alussa huomioon. Näin projekti ei veny tarpeettoman pitkäksi, ja osataan valmistautua niin, että ainakin kaikkein tärkeimmät asiat saadaan tehtyä. Jos resurssit arvioidaan väärin, on koko projekti vaarassa jäädä kesken.

Opinnäytetyössä tehdyn projektinkin oli alun perin tarkoitus olla täysin valmis, kokonainen peli. Ajanpuutteen vuoksi jouduttiin kuitenkin tyytymään vain toimivaan prototyyppiin, jolla pystyttäisiin esittelemään pelin keskeistä toiminnallisuutta. Tavallaan tämä olikin hyvä asia, sillä tehdyn prototyypin avulla huomattiin pelin suunnittelussa

olevat virheet, ja projektia pystyttään tulevaisuudessa yksinkertaistamaan ja hiomaan paremmaksi.

8.4 Lopetus

Valmiiksi saatu demoprojekti oli tarvittavan iso peli-idean toimivuuden kokeilemiseen ja tarvittavien muutosten suunnitteluun. Projektin pohjalta oli hyvä ruveta tekemään jatkosuunnitelmia.

Opinnäytetyöprosessi oli opettavainen, ja kehitti taitoja pelisuunnittelussa, ohjelmoinnissa ja projektinhallinnassa. Erityisesti peliohjelmointi on sen mielenkiintoisuuden takia monille mielekäs, hyvä ja monipuolinen tapa opetella ohjelmointia ja ohjelmistotuotantoa. Pelien tekemistä voikin suositella kaikille jotka haluavat harjoittaa taitojaan ohjelmisto-alalla, ja ovat kiinnostuneita peleistä.

LÄHTEET

1. Unity Prefabs. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Prefabs.html [viitattu: 7.10.2013]

2. Unity Creating Scenes Manual. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Manual/CreatingScenes.html [viitattu: 7.10.2013]

3. Rodriguez, E. 2007. Computer Graphic Artist. Saatavissa: https://kymi.amkit.fi/vwebv/search?sk=fi_FI&searchArg=computer+graphic+artist&searchCode=GKEY%5E*&searchType=0 [viitattu: 8.10.2013]

4. Rogers, S. 2012. Swipe This! The Guide to Great Touchscreen Game Design. Saatavilla: http://www.ellibs.com/fi/book/9781119940524 [viitattu: 8.10.2013]

5. Rutledge, P. 2012. The positive side of video games part III. Saatavissa: http://mprcenter.org/blog/2012/08/the-positive-side-of-video-games-part-iii/ [viitattu: 31.3.2013]

6. Unity Script Reference. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/index.html [viitattu: 7.10.2013]

7. Starting a Orthello scene. Saatavissa: http://www.wyrmtale.com/orthello/starting-a-scene [viitattu:7.10.2013]

8. Unity GameObjects. 2010. Saatavissa: http://docs.unity3d.com/Documentation/Manual/GameObjects.html [viitattu: 7.10.2013]

9. Creating and Using Scripts. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Manual/CreatingAndUsingScripts.html

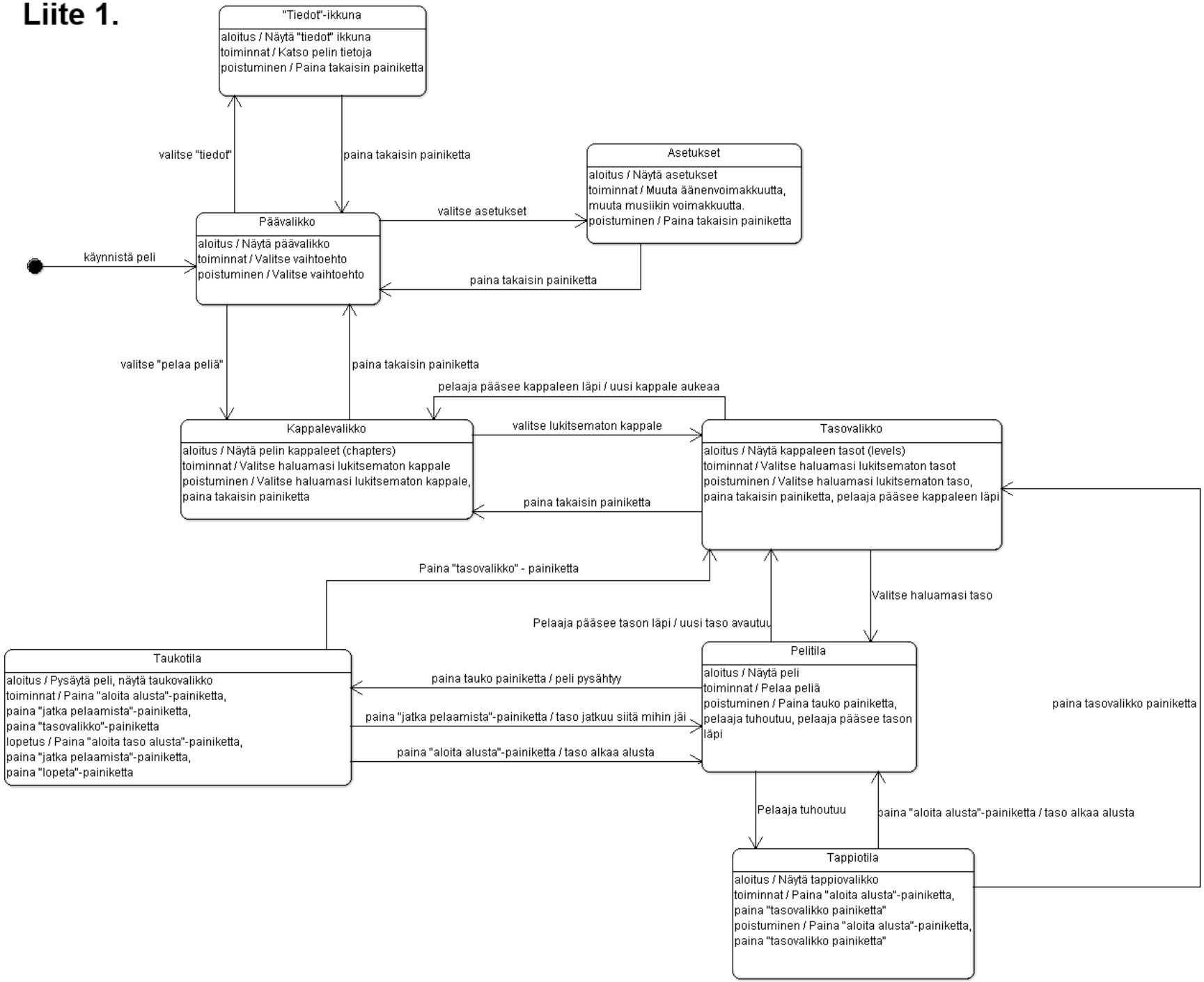
[viitattu:7.10.2013]

10. MonoBehaviour. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html [viitattu: 7.10.2013]

11. Start Method. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Start.html [viitattu: 7.10.2013]

12. Update Method. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Update.html [viitattu: 7.10.2013]
13. OTSprite. Saatavissa: http://www.wyrmtale.com/orthello/sprites#sprite [viitattu: 15.12.2013]
14. Unity Instantiate. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html [viitattu: 18.12.2013]
15. Input rajapinta. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Input.html [viitattu:18. 12.2013]
16. AnimatingSprite. Saatavissa: http://www.wyrmtale.com/orthello/sprites#animatingsprite [viitattu:6.1.2014]
17. Apple Developer Site. Saatavissa: http://developer.apple.com [viitattu:24.1.2014]

Liite 1.


"Tiedot"-ikkuna
aloitus / Näytä "tiedot" ikkuna
toiminnat / Katso pelin tietoja
poistuminen / Paina takaisin painiketta
valitse "tiedot"
paina takaisin painiketta
Asetukset
aloitus / Näytä asetukset
toiminnat / Muuta äänenvoimakkuutta,
muuta musiikin voimakkuutta.
poistuminen / Paina takaisin painiketta
Päävalikko
aloitus / Näytä päävalikko
toiminnat / Valitse vaihtoehto
poistuminen / Valitse vaihtoehto
valitse asetukset
käynnistä peli
paina takaisin painiketta
valitse "pelaa peliä"
paina takaisin painiketta
pelaaja pääsee kappaleen läpi / uusi kappale aukeaa
Kappalevalikko
aloitus / Näytä pelin kappaleet (chapters)
toiminnat / Valitse haluamasi lukitsematon kappale
poistuminen / Valitse haluamasi lukitsematon kappale,
paina takaisin painiketta
valitse lukitsematon kappale
paina takaisin painiketta
Tasovalikko
aloitus / Näytä kappaleen tasot (levels)
toiminnat / Valitse haluamasi lukitsematon tasot
poistuminen / Valitse haluamasi lukitsematon taso,
paina takaisin painiketta, pelaaja pääsee kappaleen läpi
Paina "tasovalikko" - painiketta
Valitse haluamasi taso
Pelaaja pääsee tason läpi / uusi taso avautuu
Taukotila
aloitus / Pysäytä peli, näytä taukovalikko
toiminnat / Paina "aloita alusta"-painiketta,
paina "jatka pelaamista"-painiketta,
paina "tasovalikko"-painiketta
lopetus / Paina "aloita taso alusta"-painiketta,
paina "jatka pelaamista"-painiketta,
paina "lopeta"-painiketta
paina tauko painiketta / peli pysähtyy
paina "jatka pelaamista"-painiketta / taso jatkuu siitä mihin jäi
paina "aloita alusta"-painiketta / taso alkaa alusta
Pelitila
aloitus / Näytä peli
toiminnat / Pelaa peliä
poistuminen / Paina tauko painiketta,
pelaaja tuhoutuu, pelaaja pääsee tason
läpi
paina tasovalikko painiketta
Pelaaja tuhoutuu
paina "aloita alusta"-painiketta / taso alkaa alusta
Tappiotila
aloitus / Näytä tappiovalikko
toiminnat / Paina "aloita alusta"-painiketta,
paina "tasovalikko painiketta"
poistuminen / Paina "aloita alusta"-painiketta,
paina "tasovalikko painiketta"

LIITE 2/1 PlayerControls-skripti

```
using UnityEngine;
using System.Collections;

public class PlayerControls : MonoBehaviour {


    //These variables are related to ship movement and turning
    private float rotationSpeed = 150.0f;
    private Vector3 movementDirection = new Vector3(0.0f,0.0f,0.0f);
    public Vector2 movement = new Vector2(0.0f,0.0f);
    private float maxMovementSpeed=350.0f;
    private float acceleration=0.0f;
    private float accelerationIncreaseInSecond=20.0f;
    private float maxAcceleration=40.0f;
    private float friction=125.0f;

    //These variables are related to gravitation of the ship
    private Vector3 gravityDirection = new Vector3(0.0f,0.0f,0.0f);
    private Vector3 nearestPointOfGravity = new Vector3(0.0f,0.0f,0.0f);
    private Vector3 gravityVector = new Vector3(0.0f,0.0f,0.0f);
    private float gravity=0.0f;
    private float maxFallSpeed=350.0f;
    private float fallSpeed=0.0f;

    //These variables are related to the surfmode
    private bool isSurfing;

    //These variables are related to respawn and hit system
    public bool isHit;
    public bool isDead;
    private bool isShakeRotationDirectionLeft;
    Vector3 hitObjectPosition;
    Vector3 damagedShipFloatDirection;
    private float totalShakeTime=1.5f;
    private float shakeTime=0.2f;
    private float maxShakeTime=0.3f;
    private float lastShakeDirectionSwitch=0.0f;
    private float hitTime=0.0f;
    public GameObject explosion;


    //This is the starting position in level
    public Vector3 startPosition;




    // Start function is called when the object script is first run in scene
    void Start ()
    {
        isSurfing=false;
        nearestPointOfGravity = new
Vector3(this.transform.position.x,this.transform.position.y-
20,this.transform.position.z);
        gravityDirection = nearestPointOfGravity - this.transform.position;
        gravityDirection.Normalize();
        this.transform.Translate(startPosition);
        OT.view.movementTarget = this.gameObject;
        gravity=200;
```

LIITE 2/2

```
        isHit=false;
        isDead=false;
    }

    // Update is called once per frame
    void Update ()
    {
        if(isHit==false)
            {
            this.rigidbody.velocity=new Vector3(0.0f,0.0f,0.0f);
            nearestPointOfGravity = new
Vector3(this.transform.position.x,this.transform.position.y-
20,this.transform.position.z);
            gravityDirection = nearestPointOfGravity - this.transform.position;
            gravityDirection.Normalize();

            //Here the movement of the player is handled
            if(Input.anyKey==true)
            {
                //Left engine is active
                if(Input.GetMouseButton(0)==true &&
Input.GetMouseButton(1)==false)
                {

this.transform.Rotate(Vector3.back*rotationSpeed*Time.deltaTime);
                    acceleration=0.0f;
                }
                //Right engine is active
                else if(Input.GetMouseButton(1)==true &&
Input.GetMouseButton(0)==false)
                {

this.transform.Rotate(Vector3.forward*rotationSpeed*Time.deltaTime);
                    acceleration=0.0f;
                }
                //Both enginees active
                else if(Input.GetMouseButton(1)==true &&
Input.GetMouseButton(0)==true)
                {
                    movementDirection=this.transform.up;

if(Vector3.Distance(this.transform.position+this.transform.up,nearestPointOfGrav
ity)>(Vector3.Distance(this.transform.position,nearestPointOfGravity)+0.2f))
                    {
                        fallSpeed-=acceleration;
                    }
                    if(acceleration<maxAcceleration)
                    {

acceleration+=accelerationIncreaseInSecond*Time.deltaTime;
                    }
                    if(movement.x<maxMovementSpeed && movement.x>-
maxMovementSpeed)
                    {
                        movement.x+=acceleration*movementDirection.normalized.x;
                    }
                    if(movement.y<maxMovementSpeed && movement.y>-
maxMovementSpeed)
                    {
                        movement.y+=acceleration*movementDirection.normalized.y;
```

LIITE 2/3

```
                    }

                }
            }
            //Nothing is pressed
            else
            {
                this.rigidbody.freezeRotation=true;
                acceleration=0.0f;
            }
            //Here gradually slow down the movement.
            if (movement.x>0.0f)
            {
                movement.x-=friction*Time.deltaTime;
            }
            else if(movement.x<0.0f)
            {
                movement.x+=friction*Time.deltaTime;
            }
            if (movement.y>0.0f)
            {
                movement.y-=friction*Time.deltaTime;
            }
            else if (movement.y<0.0f)
            {
                movement.y+=friction*Time.deltaTime;
            }

            //Here gravititation of the ship is applied
            if(fallSpeed<maxFallSpeed)
            {
                fallSpeed+=gravity*Time.deltaTime;
            }
            if(fallSpeed<0.0f)
            {
                fallSpeed=0.0f;
            }
            gravityVector=gravityDirection*fallSpeed;
            if(movement.x>-fallSpeed&&movement.x<fallSpeed)
            {
            movement.x+=gravityVector.x*Time.deltaTime;
            }
            if(movement.y>-fallSpeed&&movement.y<fallSpeed)
            {
            movement.y+=gravityVector.y*Time.deltaTime;
            }
            if(movement.x!=0.0f || movement.y!=0.0f)
            {
                this.transform.Translate(new
Vector3(movement.x,movement.y,0)*Time.deltaTime,Space.World);
            }
        }
        else if(isHit==true)
        {
            if(Time.time-hitTime<=totalShakeTime)
            {
                if(shakeTime<=0.0f)
                {
                    if(isShakeRotationDirectionLeft==true)
```

```
                    {
                        isShakeRotationDirectionLeft=false;
                    }
                    else
                    {
                        isShakeRotationDirectionLeft=true;
                    }
                    shakeTime=maxShakeTime;
                }
                else
                {
                    shakeTime-=Time.deltaTime;
                }
                if(isShakeRotationDirectionLeft==true)
                {

this.transform.Rotate(Vector3.back*rotationSpeed*Time.deltaTime);
                }
                else
                {

this.transform.Rotate(Vector3.forward*rotationSpeed*Time.deltaTime);
                }

                this.transform.Translate(new
Vector3(damagedShipFloatDirection.x,damagedShipFloatDirection.y,0)*Time.deltaTim
e,Space.World);
            }
            else
            {
                GameObject Explosion;
                GameObject Explosion2;
                GameObject Explosion3;
                GameObject Explosion4;

                Explosion = Instantiate(explosion,new
Vector3(this.transform.position.x+Random.Range(-
100,100),this.transform.position.y+Random.Range(-
100,100),0),Quaternion.identity) as GameObject;
                Explosion2 = Instantiate(explosion,new
Vector3(this.transform.position.x+Random.Range(-
100,100),this.transform.position.y+Random.Range(-
100,100),0),Quaternion.identity) as GameObject;
                Explosion3 = Instantiate(explosion,new
Vector3(this.transform.position.x+Random.Range(-
100,100),this.transform.position.y+Random.Range(-
100,100),0),Quaternion.identity) as GameObject;
                Explosion4 = Instantiate(explosion,new
Vector3(this.transform.position.x+Random.Range(-
100,100),this.transform.position.y+Random.Range(-
100,100),0),Quaternion.identity) as GameObject;
                this.gameObject.SetActive(false);
                this.transform.Translate(startPosition);
                isHit=false;
                isDead=true;
            }
        }
        }
```

LIITE 2/5

```
    public void IsHit(GameObject other)
    {

        isHit=true;
        hitTime=Time.time;
        shakeTime=0.0f;
        hitObjectPosition=other.transform.position;
        damagedShipFloatDirection=this.transform.position-hitObjectPosition;
        Vector3.Normalize(damagedShipFloatDirection);
    }

    public void Respawn()
    {
        this.transform.position= startPosition;
        this.movement=new Vector2 (0,0);
        this.gameObject.SetActive(true);
        isDead=false;
    }

}
```

LIITE 3/1 JetScript-skripti

```
using UnityEngine;
using System.Collections;

public class JetScript : MonoBehaviour {

    private float particleRate=0.025f;
    private float timeSinceLastParticle=0.0f;
    private int whichParticleColorToUse;
    private Vector2 touch1 = new Vector2(0,0);

    void Start ()
    {
    }

    void Update ()
    {

        if(Input.touchCount > 0)
        {
            touch1=Input.GetTouch(0).position;
        }

        timeSinceLastParticle+=Time.deltaTime;
        if(this.gameObject.name=="JetLeft")
        {
            if((touch1.x<=512  && Input.touchCount > 0) || Input.touchCount >=2)
            {
                if(timeSinceLastParticle>=particleRate)
                {
                    whichParticleColorToUse= Random.Range(1,4);
                    if(whichParticleColorToUse==1)
                    {
                    OT.CreateSpriteAt("JetParticleRed",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    else if(whichParticleColorToUse==2)
                    {
                        OT.CreateSpriteAt("JetParticleOrange",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    else if(whichParticleColorToUse==3)
                    {
                        OT.CreateSpriteAt("JetParticleYellow",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    timeSinceLastParticle=0.0f;
                }
            }
        }
        else if(this.gameObject.name=="JetRight")
        {
            if((touch1.x>512  && Input.touchCount > 0) || Input.touchCount >=2)
            {
                if(timeSinceLastParticle>=particleRate)
                {
                    whichParticleColorToUse= Random.Range(1,4);
                    if(whichParticleColorToUse==1)
                    {
```

LIITE 3/2

```
                    OT.CreateSpriteAt("JetParticleRed",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    else if(whichParticleColorToUse==2)
                    {
                        OT.CreateSpriteAt("JetParticleOrange",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    else if(whichParticleColorToUse==3)
                    {
                        OT.CreateSpriteAt("JetParticleYellow",new
Vector2(this.transform.position.x,this.transform.position.y));
                    }
                    timeSinceLastParticle=0.0f;
                }
            }
        }
    }
}
```

LIITE 4/1 BackgroundManagerScript

```
using UnityEngine;
using System.Collections;

public class BackgroundManagerScript : MonoBehaviour {

    public GameObject playerCharacter;
    public GameObject background1;
    public GameObject background2;
    public GameObject background3;
    public GameObject background4;
    public GameObject currentBackgroundUnderCharacter;
    public GameObject lastBackgroundUnderCharacter;
    public bool backgroundSwitched=true;
    public bool isOnLeftSide;
    public bool isOnTop;

    private float backgroundWidth;
    private bool hasChangedSides=false;

    void Start ()
    {
        //Here check the starting position of the ship related to background1

if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
        {
            isOnLeftSide=false;
            isOnTop=true;
        }
        else
if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
        {
            isOnLeftSide=false;
            isOnTop=false;
        }
        else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
        {
            isOnLeftSide=true;
            isOnTop=true;
        }
        else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
        {
            isOnLeftSide=true;
            isOnTop=false;
        }

        backgroundWidth=background1.transform.localScale.x;
        background1.transform.position = new Vector3(0.0f,0.0f,1.0f);
        background2.transform.position = new Vector3(backgroundWidth,0.0f,1.0f);
        background3.transform.position = new Vector3(0.0f,backgroundWidth,1.0f);
```

LIITE 4/2

```
        background4.transform.position = new
Vector3(backgroundWidth,backgroundWidth,1.0f);

    }

    void Update ()
    {
        if(backgroundSwitched==true)
        {
        //check if you change the quarter of backgound1 youre in

if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
            {
                if(isOnLeftSide==true)
                {
                    isOnLeftSide=false;
                    hasChangedSides=true;
                }
                if(isOnTop==false)
                {
                    isOnTop=true;
                    hasChangedSides=true;
                }

        }
            else
if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
            {
                if(isOnLeftSide==true)
                {
                    isOnLeftSide=false;
                    hasChangedSides=true;
                }
                if(isOnTop==true)
                {
                    isOnTop=false;
                    hasChangedSides=true;
                }
            }
            else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
            {
                if(isOnLeftSide==false)
                {
                    isOnLeftSide=true;
                    hasChangedSides=true;
                }
                if(isOnTop==false)
                {
                    isOnTop=true;
                    hasChangedSides=true;
                }
            }
```

LIITE 4/3

```
            else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
            {
                if(isOnLeftSide==false)
                {
                    isOnLeftSide=true;
                    hasChangedSides=true;
                }
                if(isOnTop==true)
                {
                    isOnTop=false;
                    hasChangedSides=true;
                }
            }
        }
        //here if the background where  the player was at is switched
        if(backgroundSwitched==false)
        {
            background1.transform.position =
currentBackgroundUnderCharacter.transform.position;
            currentBackgroundUnderCharacter = background1;

if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x+backgroundWidth,background1.transform.p
osition.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y+backgr
oundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x + backgroundWidth,
background1.renderer.bounds.center.y + backgroundWidth,1.0f);
            }
            else
if(playerCharacter.renderer.bounds.center.x>background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x+backgroundWidth,background1.transform.p
osition.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y-
backgroundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x + backgroundWidth,
background1.renderer.bounds.center.y - backgroundWidth,1.0f);
            }
            else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y>background1.renderer.bounds.center.y)
            {
```

```
                background2.transform.position = new
Vector3(background1.transform.position.x-
backgroundWidth,background1.transform.position.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y+backgr
oundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x - backgroundWidth,
background1.renderer.bounds.center.y + backgroundWidth,1.0f);
            }
            else
if(playerCharacter.renderer.bounds.center.x<background1.renderer.bounds.center.x
&&
playerCharacter.renderer.bounds.center.y<background1.renderer.bounds.center.y)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x-
backgroundWidth,background1.transform.position.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y-
backgroundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x - backgroundWidth,
background1.renderer.bounds.center.y - backgroundWidth,1.0f);
            }
            backgroundSwitched=true;
        }
        //if the player has changed the the part of bacground1 hes in
        if(hasChangedSides==true)
        {
            if(isOnLeftSide && isOnTop)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x-
backgroundWidth,background1.transform.position.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y+backgr
oundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x - backgroundWidth,
background1.renderer.bounds.center.y + backgroundWidth,1.0f);
            }
            if(!isOnLeftSide && isOnTop)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x+backgroundWidth,background1.transform.p
osition.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y+backgr
oundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x + backgroundWidth,
background1.renderer.bounds.center.y + backgroundWidth,1.0f);
            }
            if(isOnLeftSide && !isOnTop)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x-
backgroundWidth,background1.transform.position.y,1.0f);
```

```
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y-
backgroundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x - backgroundWidth,
background1.renderer.bounds.center.y - backgroundWidth,1.0f);
            }
            if(!isOnLeftSide && !isOnTop)
            {
                background2.transform.position = new
Vector3(background1.transform.position.x+backgroundWidth,background1.transform.p
osition.y,1.0f);
                background3.transform.position = new
Vector3(background1.transform.position.x,background1.transform.position.y-
backgroundWidth,1.0f);
                background4.transform.position = new
Vector3(background1.renderer.bounds.center.x + backgroundWidth,
background1.renderer.bounds.center.y - backgroundWidth,1.0f);
            }
            hasChangedSides=false;
        }
    }
}
```

LIITE 5/1 GameManagerScript

```
using UnityEngine;
using System.Collections;

public class GameManagerScript : MonoBehaviour {

    private float timePlayed;
    public bool isRunning;
    public bool isPaused;
    public GameObject playerCharacter;
    public GameObject goal;
    public Texture2D pauseBtnTexture;
    public Texture2D restartBtnTexture;
    public Texture2D playBtnTexture;
    public Texture2D menuBtnTexture;
    public Texture2D levelFailedTxt;
    public Texture2D levelCompleteTxt;
    public Texture2D pauseTxt;

    void Start ()
    {
        isRunning=true;
        isPaused=false;
    }

    void Update ()
    {
        if(isRunning=true)
        {
        timePlayed+=Time.deltaTime;
        }

        if(playerCharacter.GetComponent<PlayerControls>().isDead==true ||
goal.GetComponent<GoalScript>().playerHasReachedTheGoal==true)
        {
            isRunning=false;
        }

    }

    void OnGUI()
    {
        GUI.Label (new Rect(Screen.width/2, Screen.height/20, 200, 50), "Time:"
+ (int)timePlayed );
        if(isPaused==false &&
playerCharacter.GetComponent<PlayerControls>().isDead==false &&
goal.GetComponent<GoalScript>().playerHasReachedTheGoal==false )
        {
            if(GUI.Button(new Rect(50,50,100,100), pauseBtnTexture))
            {
                Time.timeScale=0;
                isPaused=true;
            }
        }
        else if(isPaused==true &&
playerCharacter.GetComponent<PlayerControls>().isDead==false &&
goal.GetComponent<GoalScript>().playerHasReachedTheGoal==false)
        {
            if(GUI.Button(new Rect(50,50,100,100), playBtnTexture))
            {
```

LIITE 5/2

```
                Time.timeScale=1;
                isPaused=false;
            }
            GUI.Label (new Rect(Screen.width/2-100, Screen.height/2-200, 200,
50), pauseTxt);
            if (GUI.Button(new Rect(Screen.width/2-110,Screen.height/2-
150,100,100), restartBtnTexture))
            {
                timePlayed=0;
                playerCharacter.GetComponent<PlayerControls>().Respawn();
                isRunning=true;
                isPaused=false;
                Time.timeScale=1;
                playerCharacter.GetComponent<PlayerControls>().isDead=false;
                playerCharacter.GetComponent<PlayerControls>().isHit=false;
            }
            if (GUI.Button(new Rect(Screen.width/2+10,Screen.height/2-
150,100,100), menuBtnTexture))
            {
            }
        }
        if(playerCharacter.GetComponent<PlayerControls>().isDead==true)
        {
            GUI.Label (new Rect(Screen.width/2-125, Screen.height/2-200, 250,
50), levelFailedTxt);
            if (GUI.Button(new Rect(Screen.width/2-110,Screen.height/2-
150,100,100), restartBtnTexture))
            {
                timePlayed=0;
                playerCharacter.GetComponent<PlayerControls>().Respawn();
                isRunning=true;
            }
            if (GUI.Button(new Rect(Screen.width/2+10,Screen.height/2-
150,100,100), menuBtnTexture))
            {

            }
        }
        if(goal.GetComponent<GoalScript>().playerHasReachedTheGoal==true)
        {
            GUI.Label (new Rect(Screen.width/2-125, Screen.height/2-200, 250,
50), levelCompleteTxt);
            if (GUI.Button(new Rect(Screen.width/2-110,Screen.height/2-
150,100,100), restartBtnTexture))
            {
                timePlayed=0;
                playerCharacter.GetComponent<PlayerControls>().Respawn();
                isRunning=true;
                goal.GetComponent<GoalScript>().playerHasReachedTheGoal=false;
            }
            if (GUI.Button(new Rect(Screen.width/2+10,Screen.height/2-
150,100,100), menuBtnTexture))
            {

            }
        }
    }
}
```