KYMENLAAKSON AMMATTIKORKEAKOULU

Tietotekniikka / Ohjelmistotekniikka

Teppo Ristola

PELI APPLEN KOSKETUSNÄYTTÖLAITTEILLE

2014

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikka

RISTOLA, TEPPO	Peli Applen kosketusnäyttölaitteille
Opinnäytetyö	34 sivua
Työn ohjaaja	Opettaja Paula Posio
Toimeksiantaja	Kymenlaakson ammattikorkeakoulu
Toukokuu 2014	
Avainsanat	Peli, Unity, kosketusnäyttö, iOs

Opinnäytetyön aihe oli oman pelin suunnitteleminen ja toteuttaminen Unityn 4.3-version mukana tulleiden 2D-työkalujen avulla. Opinnäytetyöni tavoitteena oli toteuttaa toimiva ja pelattava pelin ideaa esittelevä pelidemo Applen kosketusnäyttölaitteille.

Opinnäytetyön aiheena olevan pelin suunnittelun aloitin miettimällä aikaisemmin pelaamiani kosketusnäyttöpelejä, joista etsin kosketusnäyttölaitteille sopivia ohjaussysteemejä. Pelin suunnitelman ollessa valmis siirryin valitsemaan työhön sopivat työkalut jo valitun Unity-pelimoottorin lisäksi. Valitsin työhön käytetyt työkalut kokemukseni perusteella. Opinnäytetyössä perehdytään tarkemmin valitsemiini työkaluihin ja pelidemon toteutukseen, sekä Applen kosketusnäyttölaiteversion tekemiseen.

Työn tuloksena syntyi Applen kosketusnäyttölaitteilla pyörivä ja pelattava pelidemo. Sain myös kokemusta Unityn käytöstä ja Applen käyttöjärjestelmien ohjelmointitoimenpiteistä.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information Technology

RISTOLA, TEPPO	Game for Apple Touchscreen Devices
Bachelor's Thesis	34 pages
Supervisor	Paula Posio, Principal lecturer
Commissioned by	Kymenlaakso University of Applied Sciences
May 2014	
Keywords	Game, Unity, touchscreen, iOs

The focus of this thesis was the design and creation of a game using 2D tools from Unity version 4.3. The objective was to create a working gameplay demo of a game for Apple iOs touchscreen devices.

Design process of the game was started by comparing existing touchscreen games to find a fitting touchscreen control scheme for the game. After the design process of the game was completed, it was time to choose fitting tools to go with the already chosen Unity game engine. The tools were chosen according to experience. The selected tools were examined and briefed on in the study. The creation of the gameplay demo and making of the iOs version were also documented and presented in the thesis in detail.

A playable gameplay demo working on an Apple iOs touchscreen device was made as a result of this thesis. The process also provided experience in working with Unity and the steps needed for programming for Apple operating systems.

SISÄLLYS

TERMIT JA LYHENTEET

2D / 3D	2D ja 3D tulevat englannin kielestä kaksi ja kolme ulotteinen. Pelien tekemisessä nämä ulottuvuudet tarkoittavat korkeutta leveyttä ja syvyyttä.
Build	Pelimoottorin rakentama pelattava versio pelistä.
Bundle identifier	Pakettitunniste. Applen ohjelmasertifikaatteihin vaadittava tunniste.
Indie-pelinkehittäjä	Yksityinen ilman julkaisijaa toimiva pelinkehittäjä.
iOs	Applen mobiililaitteiden käyttöjärjestelmä.
OS-X	Applen Mac-tietokoneiden käyttöjärjestelmä.
Proof of concept	Yksinkertainen toteutus jostain ilmiöstä, jonka tarkoituksena on hahmotella ilmiön toimintaa.
Pelimoottori	Pelimoottoriksi sanotaan ohjelmia, jotka on kehitetty pelienkehitystä varten.
Retina-näyttö	Retina-näyttö on nimitys Applen mobiililaitteiden näytöille.
Sprite	Spriteksi kutsutaan yksittäisiä kuvia 2D-peliohjelmoinnissa.
Xcode	Xcode on Applen ohjelmointiympäristö.

1 JOHDANTO

Opinnäytetyöni aiheena oli kehittää kosketusnäyttölaitteille sopiva pelidemo omasta peli-ideastani. Kymenlaakson ammattikorkeakoulussa toimiva Pelilabra eli Gamelab mahdollisti oman pelin toteuttamisen opinnäytetyön aiheena. Pelilabra on vuonna 2011 Kymenlaakson ammattikorkeakoulussa toimintansa aloittanut peliohjelmointiin ja -kehittämiseen suuntaava oppimisympäristö tietotekniikan koulutusohjelmassa.

Pelilabrassa opiskelu on hyvin vapaamuotoista ja mahdollistaa myös suuntautumisen normaaleihin ohjelmistotehtäviin perusopetusten kautta. Pelilabran kautta voi myös tutustua paikallisiin peliyrityksiin, koska koulu tekee yhteistyötä yritysten kanssa ja järjestää kursseja yritysten tiloissa.

Koska harrastuksenani on aina ollut videopelien pelaaminen, oli oman pelin tekeminen hyvin mieleinen aihe. Pelattaessa muita pelejä tulee välillä mieleen ideoita, kuinka joitakin eri peleissä käytettyjä pelimekaniikoita yhdistelemällä kyseiseen peliin saataisiin aikaiseksi jotakin uutta ja mielenkiintoista. Opinnäytetyössäni käytetty peli-idea syntyi myös tällä tavalla.

Kuva 1. RedLynx-peliyrityksen Drawrace.

Peli-idea syntyi kun pelasin Applen kosketusnäyttölaitteelle tehtyä peliä Drawrace, joka näkyy kuvassa 1. Drawrace pelin pääpelimekaniikkana on pelaajahahmolle piirrettävä reitti, jota pelaajahahmo seuraa kun reitin piirto on valmis (1.). Kyseinen pelimekaniikka on mielestäni paras kosketusnäyttölaitteille sopiva ohjaussysteemi, jota olen kokeillut. Peli-ideani oli yhdistää se ylhäältä kuvattuun räiskintäpeliin, jossa on tarkoitus väistellä suuria määriä vihollisen ampumia luoteja.

Muihin olemassa oleviin peleihin vertaaminen on myös pelin kehittämisessä hyödyllistä. Välillä jää pitkäksi aikaa miettimään ratkaisua jotain pelimekaniikkaa ohjelmoidessa ja ongelma saattaa ratketa heti kun näkee, että miten ongelma on ratkaistu muissa samantyyppisissä peleissä. Tämän tavan huomasin olevan myös käytössä peliyrityksessä, jossa työskentelin työharjoitteluni aikana.

Peli-idean synnyttyä oli valittava työhön sopiva pelimoottori. Tähän peliin valitsin Unity-ohjelman, koska lukemani mukaan pelimoottori toimii hyvin tehtäessä peliä usealle eri alustalle. Valintaa myös helpotti, että Unity on hyvin suosittu pelialan yri-

tyksissä, joten sen osaamisesta ja kokemuksesta on hyötyä. Pelimoottoriin on myös lähiaikoina lisätty 2D-ohjelmointirajapinta, jota halusin kokeilla.

2 TYÖKALUT

Ennen ohjelman tekemisen aloittamista oli valittava siihen sopivat työkalut. Suurin osa työssä käytetyistä työkaluista olivat minulle entuudestaan tuttuja, mutta mukaan mahtui myös täysin uusia ohjelmia, kuten Xcode. Työkalujen valintaan vaikutti myös hinta, koska kaikissa tapauksissa päädyin ilmaiseen ratkaisuun.

Työkaluina käytin myös pöytäkonettani ja kannettavaa tietokonettani. Pelin iOs-version kääntämiseen käytin Applen uusimmalla OS-X -käyttöjärjestelmällä varustettua tietokonetta. Kahden Windows 7 -koneen välillä työskentely oli yksinkertaista, koska käytin työssäni versionhallintaa. Kosketusnäyttöohjausten ja käännetyn ohjelman testaamiseen käytin Applen iPhone4S -puhelinta ja iPad -tablettia, joissa uusimmat versiot Applen iOs -käyttöjärjestelmästä.

2.1 Unity

Unity on Unity Technologies yrityksen kehittämä monille alustoille sopiva pelimoottori. Sitä voi käyttää pelinkehitykseen useille eri alustoille kuten selaimille, Windowsille, Applen OS-X -käyttöjärjestelmälle, yleisimmille konsoleille sekä useille eri mobiilialustoille.

Ensimmäinen Unity julkaistiin 2005 vain Applen OS-X -käyttöjärjestelmälle, mutta uusien julkaisujen myötä se ulottui usealle eri alustalle. Unity on ollut lähiaikoina erittäin suosittu peliyrityksissä ja indie-pelikehittäjien keskuudessa. Uusin virallinen julkaisu on 2014 tammikuussa tullut 4.3.4-versio. Version 4.3 mukana ohjelmaan lisättiin virallinen 2D-tuki, jota käytin pelini tekemiseen.

Unity on suurimmaksi osaksi täysin ilmainen kokeiltavaksi. Opiskelijana ei tarvitse välittää ohjelman maksullisista versioista.

Unityssä on myös Asset store. Se on nettikauppa, jossa on mahdollista ostaa muiden tekemiä pelimateriaaleja tai myydä omia.

Unity sopii erittäin hyvin aloittelevalle ohjelmoijalle, koska se on melko yksinkertainen ja Unity-skriptit on kirjoitettavissa usealla eri ohjelmointikielellä. Moottorin käytöstä löytyy myös helposti tutoriaaleja. Koska Unity on yleisesti käytetty moottori, ongelmiin löytyy vastauksia nopeasti. Unity-editorin ominaisuuksia voi myös päivittää muiden tekemillä lisäosilla. Näistä esimerkkinä oli 2D-ohjelmointia tukeva lisäosa ennen virallisten työkalujen julkaisua.

Kuva 2. Kuvakaappaus pelistä Tiny Troopers 2.

Pelimoottorilla on kehitetty useita tunnettuja pelejä. Näitä ovat esimerkiksi Kotkalaisen Kukour-yrityksen kuvassa 2 näkyvä Tiny troopers pelisarja (2.).

2.1.1 Peruselementit

Unityssa kehitetyt pelit koostuvat pääasiassa eri peruselementeistä. Kaikkia elementtejä voi rakentaa itse, käyttää Unityn mukana tulevia osia tai ostaa muiden tekemiä nettikaupan kautta.

Scenet ovat tasoja, joihin muut elementit lisätään. Scenejä pystyy vaihtamaan koodin kautta, mikä mahdollistaa pelikenttien rakentamisen eri sceneihin.

Gameobject eli peliobjekti on Unityn scenen rakennuksen peruselementti. Jokainen scenessä oleva esine tehdään peliobjektin sisään. Objektit ovat tyhjiä säiliöitä, joiden sisältö eli komponentit määräävät, mitä objekti tekee scenessä. Peliobjektilla on vakioominaisuutena vain sijainti ja orientaatio scenessä. Peliobjekteja pystyy myös lisäämään muiden peliobjektien lapsiksi, jolloin ne seuraavat isäntäobjektia scenessä. (3.)

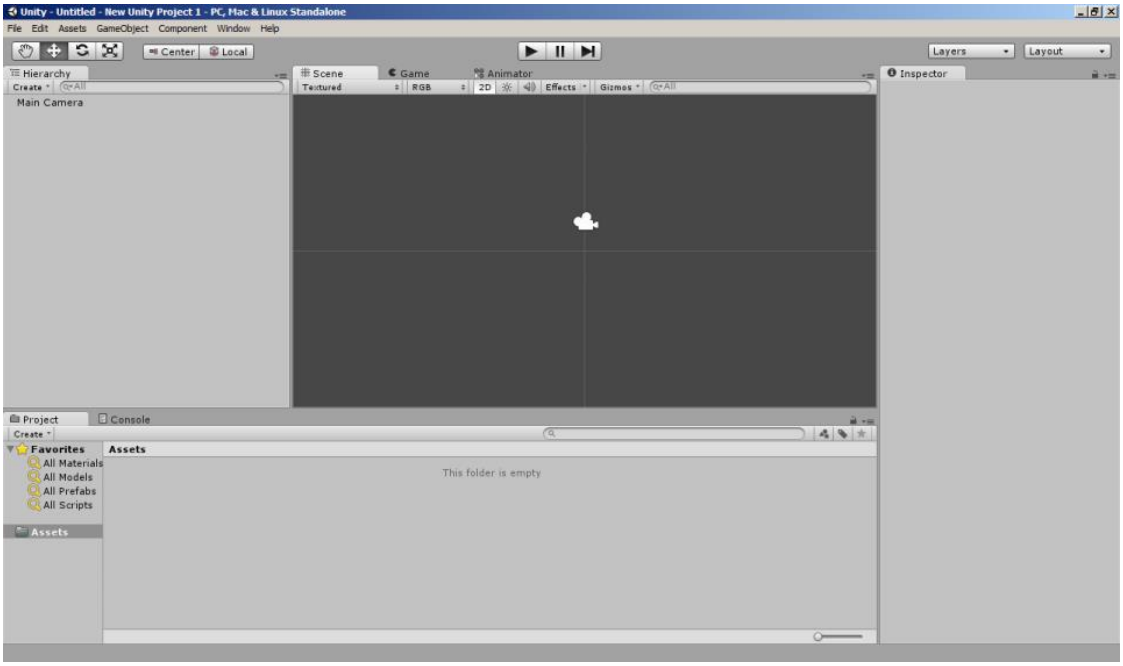
Komponentit ovat peliobjektin sisältöä. Tärkeimpänä on transform-komponentti, joka on vakiona kaikissa peliobjekteissa määräämässä sen sijainnin. Peliobjektin komponenttien hallitseminen käy helposti Unityn Inspector-ikkunan kautta, jossa näkyy scenessä valitun objektin komponenttivalikko. Valikon kautta tapahtuu komponenttien lisääminen poistaminen ja muokkaus. Komponentteihin lukeutuvat myös 3D-mallit ja spritet. Jos jotain ominaisuutta ei löydy Unityn perus-komponenteista, ominaisuuden voi rakentaa itse käyttäen skripti-komponenttia. Komponentille asetetaan skripti, jota kyseinen peliobjekti seuraa. (4.)

Skriptit ovat tekstitiedostoja, joihin sisällytetään kaikki itse kirjoitettava koodi. Unityn mukana tulee myös valmiita skriptitiedostoja. Skriptien päätehtävänä on toimia objektien tekoälynä, mutta niiden avulla voi myös tehdä objekteja, jotka pitävät muita pelissä olevia objekteja yllä. Esimerkkinä työhöni tekemä objekti GameManager, joka hoitaa pelin etenemisen, vaikka objektia ei näy scenessä pelin aikana.

Kun peliobjektiin on lisätty kaikki tarvittavat komponentit, voi siitä tehdä prefab-tiedoston. Prefab-tiedosto on klooni kokonaisesta peliobjektista, joka perii kaiken alkuperäiseltä objektilta nimeä myöten. Esimerkiksi yleisesti peleissä kaikista erilaisista hahmoista on omat prefabit, jotta niiden luominen skriptien avulla olisi helpompaa.

2.1.2 Käyttöliittymä

Unityä käytetään pääasiassa editorin kautta. Editorin Käyttöliittymä on vapaasti muokattavissa. Skriptien kirjoittamiseen voi käyttää mitä vain haluamaansa tekstieditoria, mutta ohjelman mukana tuleva Monodevelop on hyvin käytännöllinen ja toimii hyvin Unityn kanssa yhteistyössä.



Kuva 3. Unity-editorin peruskäyttöliittymä.

Uutta projektia avattaessa Unity-editorilla ruudulle tulee näkymään kuvan 3 mukainen tyhjä scene ja sen ympärille muita ikkunoita. Näitä ikkunoita ovat Scenen hierarkia, projektin sisältöluettelo sekä valitun peliobjektin ominaisuudet. Scenen hierarkiasta näkee sceneen laitetut objektit. Tyhjässä scenessä pitäisi olla vain Main Camera -objekti. Projektin sisältöluettelosta näkee ohjelmaan tuodut assetit eli materiaalit. Kun scenestä valitsee jonkin peliobjektin sen ominaisuudet tulevat näkymään Inspector-ikkunaan. Inspector-ikkunan kautta luodaan objektien sisältö käyttäen komponentteja, skriptejä, spritejä tai 3D-malleja. Perusikkunoiden lisäksi Unitystä löytyy tiettyihin toimenpiteisiin liittyviä ikkunoita, kuten animator, jossa luodaan animaatiotaulukoita. (5.)

2.1.3 Unity2D

Unity2D on Unity 4.3 -version mukana tullut uusi 2D-pelien kehittämiseen suunniteltu kehitysympäristö. Version ominaisuuksina tuli kehityksiä spritejen tuomiseen ja hallitsemiseen, helpotuksia 2D-animaatioiden luomiseen ja käyttämiseen editorin kautta, aikaisempien 3D-komponenttien pohjalta kehitetty 2D-pelifysiikkasysteemi, sekä uusi esimerkkiprojekti työkaluilla toteutetusta 2D pelistä. (6.)

2.2 Monodevelop

Monodevelop on Unityn mukana tuleva ohjelmoimiseen tarkoitettu tekstieditori. Ohjelma on ilmainen pääasiassa C# tai muille ”.NET” -ohjelmointikielille suunniteltu ohjelmointiympäristö (7.). Unityn tapauksessa ohjelmaa voi käyttää kaikkien tuettujen ohjelmointikielien kanssa. Ohjelma on saatavilla Windowsille ja OS-X -käyttöjärjestelmälle.

2.3 Versionhallinta

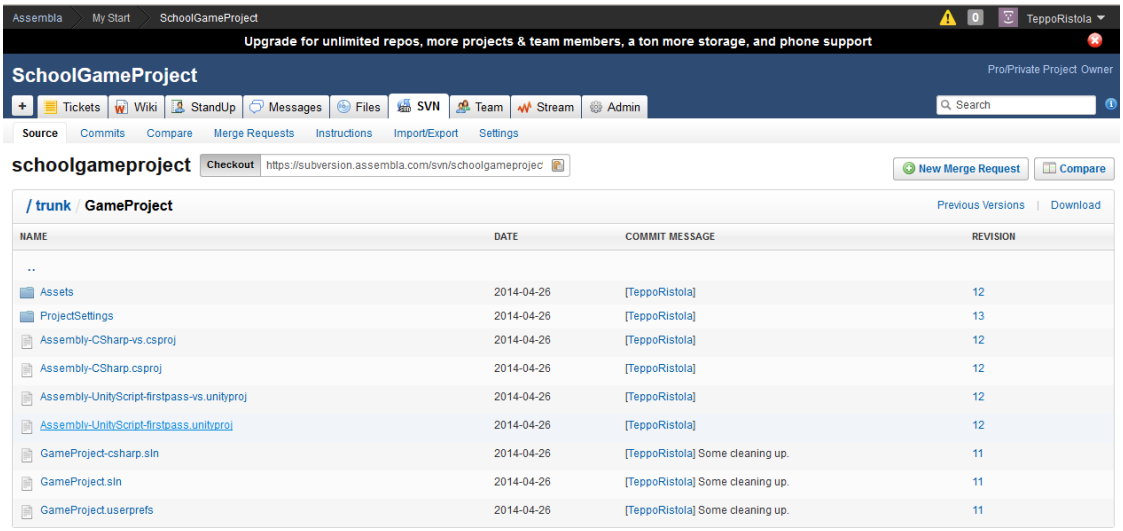
Versionhallinta on ohjelmointitekniikassa yleisesti käytetty menetelmä, jonka avulla hallitaan ohjelman eri versioita. Versionhallinta mahdollistaa ohjelman versioiden tallentamisen kehitysympäristön ulkopuolelle, sekä eri versioiden eroavaisuuksien tarkastelun. Versionhallinta mahdollistaa myös aikaisemman version palauttamisen ja uusien haarojen aloittamisen. Tärkein ominaisuus versionhallinnalla ohjelmistotyössä on, että se mahdollistaa useamman eri ohjelmoijan samanaikaisen kooditiedostojen muokkauksen.

2.3.1 TortoiseSVN

TortoiseSVN on helppokäyttöinen versionhallintaohjelmisto Windows käyttöjärjestelmille. Ohjelma on rajoittamattomasti täysin ilmainen kaikille, mukaan lukien myös kaupalliset yritykset. Ohjelman lähdekoodi on tarjolla ilmaisena, jos haluaa kehittää oman version ohjelmasta. (8.)

Ohjelma toimii Windows explorerin lisäosana ja kaikkia sen tärkeimpiä ominaisuuksia voi käyttää hiirivalikon kautta. SVN yhdistää järjestelmältä valitun kansion ulkoisella palvelimella tai paikallisella kiintolevyllä olevaan versionhallinnan alaiseen kansioon. Yhdistämisen jälkeen kansion kautta voi hallita versionhallinnassa olevaa versiota, tai päivittää kansion versionhallinnassa olevan version mukaiseksi. Kun versionhallinnan kansion sisältöä muokataan yhdellä koneella, siitä voi ottaa päivityksen muilla koneilla.

2.3.2 Assembla



Kuva 4. Lopullinen tilanne Assembla.com -sivustolta pelini SVN-serveristä.

Käytin versionhallintaan ilmaista Assembla-sivustoa. Assembla on versionhallintasivusto, joka mahdollistaa yksinkertaisten projektien siirtämisen useammalle koneelle internetin välityksellä. Versionhallintaan tehtyihin lisäyksiin voi myös laittaa lisäysviestin, joka näkyy kuvassa 4.

2.4 Paint.net ja Opengameart

Paint.net on ilmainen kuvankäsittelyohjelma Windows alustoille. Käytin ohjelmaa pelin spritejen muokkaamiseen.

Koska työni tarkoituksena oli tehdä pelistä vain demoversio, en käyttänyt turhaa aikaa spriteihin, jotka luultavasti vaihdan myöhemmin. Käytin työssäni kuvia Opengameart.org -sivustolta. Sivustolta löytyy ilmaista peligrafiikkaa tai äänimaailmaa, joita voi käyttää, kun mainitsee tekijät pelissä. Muokkasin spritejä tekemällä niistä kuvaatlaksia tai lisäämällä yksinkertaisia animaatioita.

Opengameart-sivusto avattiin vähentämään ohjelmoijataidetta ilmaisissa peleissä. Yleensä peliin tehdään lopullinen ulkonäkö vasta silloin, kun peli on viimeistelyvaiheessa, mutta useat ilmaiset peliprojektit eivät koskaan pääse tähän pisteeseen asti ja päätyvät käyttämään väliaikaiseksi tarkoitettua taidetta. Sivun tarkoituksena on tarjota ilmaisten pelien tekijöille laaja ja koko ajan kasvava korkean laadun taidekirjasto. Sivun taidetta voi myös käyttää kaupallisissa töissä, mutta on toteltava taiteen lisääjän määräämiä sopimuksia. (9.)

2.5 Xcode

Xcode on Applen ohjelmistotuotantoon tarkoitettu ohjelmointiympäristö, jota on käytettävä iOs-versioita käännettäessä. Xcode on ilmainen ohjelmisto, jonka voi ladata virallisen OS-X -käyttöjärjestelmän ohjelmakaupasta. Vaikka Xcode on ilmainen ohjelma, sen käyttöön tarvitsee ainakin iOs-pelejä tehtäessä Applen kehittäjätunnukset. Kehittäjätunnusten hankkiminen vaatii vuosimaksua, mutta sain työtäni varten tunnukset koululta.

Unity kääntää automaattisesti Xcode-projektitiedoston, kun ohjelmasta valitaan iOs-version kääntäminen.

3 OHJELMAKOODIN KIRJOITTAMINEN

Kaiken tuotettavan koodin pitää olla selvästi luettavaa ainakin koodin kirjoittajalle, vaikka koodin pariin palattaisiin paljon myöhemmin. Koodin luettavuutta auttaa eri muuttujien selvä nimeäminen. Omana tapanani on nimetä muuttujat pienellä alkukir-

jaimella tyypin mukaan ja isolla kirjoitettuna jokin selvä nimi. Esimerkkinä nimeäisin float-tyyppisen muuttujan, jonka tarkoituksena on pitää yllä aikalaskuria fTimer.

```
21.     private List<Vector3> path = new List<Vector3>();
22.     private bool isMoving = false;
23.     private float fDistance;
24.     private Vector3 destination = Vector3.zero;
25.     private Vector3 lastpathnode = Vector3.zero;
26.
27.     //line renderer handles path drawing
28.     private LineRenderer line;
29.     private int iCurrentline = 0;
30.
31.     //animator for changing animation states
32.     private Animator animator;
33.
34.     // Update is called once per frame
35.     void Update () {
36.         //update pause
37.         if (isMoving)
38.             GameManager.GetInstance().SetPause(false);
39.         else
40.             GameManager.GetInstance().SetPause(true);
41.
42.         UpdateControls();
43.     }
44.
45.     //function that gets controls for player ship movement
46.     void UpdateControls()
47.     {
48.         //shoot towards mouse if moving
49.         if (Input.GetMouseButtonUp(0) && isMoving)
50.             Shoot ();
51.
52.         //path drawing if mouse is held down go to moving otherwise
53.         if (Input.GetMouseButton(0) && !isMoving)
54.             DrawPath ();
55.         else
56.             UpdateMoving();
```

Kuva 5. Esimerkki kooditiedostostani

Koodin lukemista auttaa myös koodirivien oikea sisennys ohjelmarakenteiden perusteella. Kaikki vastakkain kuuluvat hakasulut on oltava samalla tasolla, jolloin koodista on helppo löytää kaikki ehtolausekkeet ja toistot. Peleissä luokan Update-funktio kannattaa pitää mahdollisimman siistinä ja siirtää kaikki ylimääräiset toiminnot omiin funktioihinsa. Koodin redundanssin poistamisesta on myös hyötyä. Jos koodissa käytetään samaa laskua useampaan kertaan, on hyvä tehdä kokonaan uusi funktio ja kutsua sitä kaikissa tapauksissa.

Hyvin tärkeää on myös koodin huolellinen kommentointi. Kommentointi tarkoittaa koodirivien väliin merkittyjä selväkielellä kirjoitettuja kommentteja. Kommenteista on helppo lukea, mitä mikäkin koodin kohta tekee ja se auttaa koodin korjausta, jos ohjelmassa ilmenee virheitä tai halutaan lisätä jokin uusi ominaisuus vanhan lisäksi. Myös lisäselvennykset muuttujia luodessa auttavat koodin selvyyttä.

4 TOTEUTUS

4.1 Suunnittelu

Halusin tehdä pelistäni kosketusnäytölle sopivan, joten aloitin peli-idean kehittämisen pelin ohjausmekaniikasta. Kokeiltuani useita erilaisia iOs-pelejä oli mieleeni jäänyt selvästi yksi ohjausmekaniikka. Kotimaisen RedLynx-peliyrityksen pelissä Drawrace käytetty reitin piirtäminen sopii mielestäni täydellisesti kosketusnäyttöpeleihin, joten halusin kehittää pelin samanlaisella ohjaussysteemillä. Päätin kuitenkin tehdä pelistä mahdollisimman omanlaisensa, joten mietin ohjauksille sopivan uuden peligenren. Lopulta päädyin tekemään ylhäältäpäin kolmannen persoonan kamerakulmalla kuvatun taktisen räiskintäpelin.

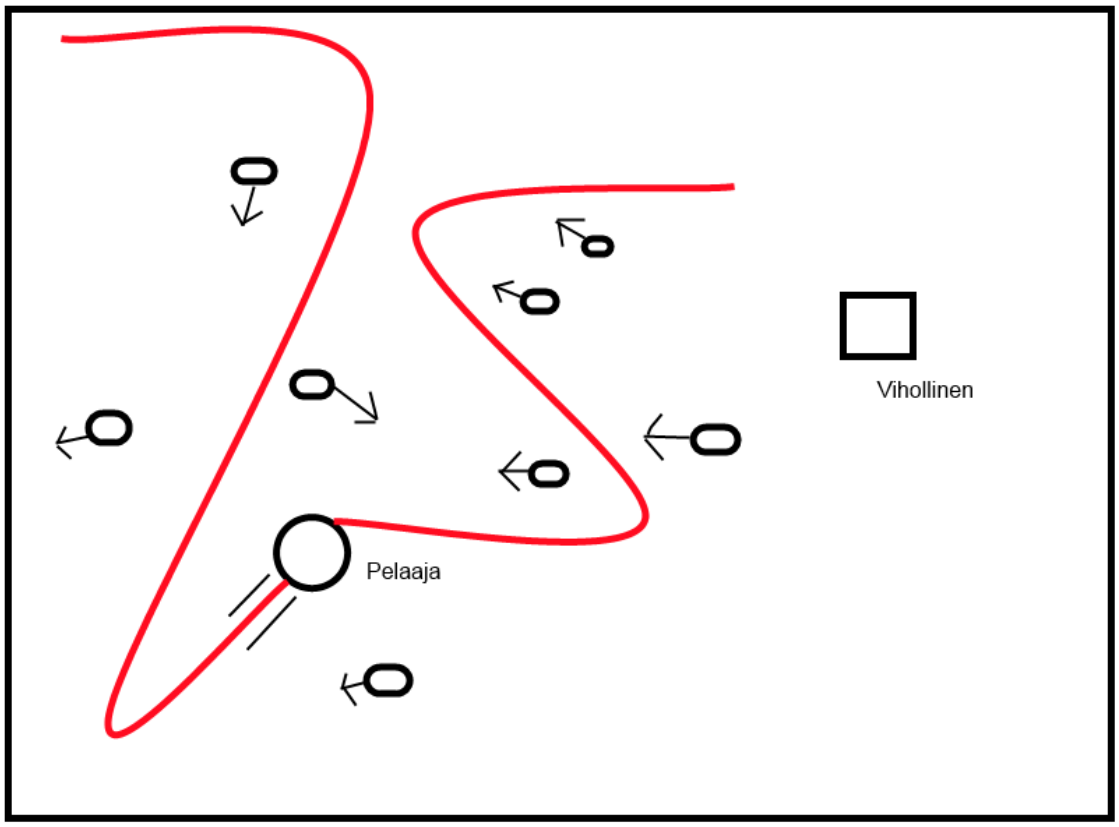
Pelinbudjetin päätin pitää ilmaisena, joten käytin työssä vain koululta ilmaiseksi saatavia, tai etukäteen omistamiani työkaluja. Pelin kohdeyleisöksi mietin yleisesti kaikenikäiset kosketusnäyttölaitteen omistajat, sillä suunnitelmanani oli pitää pelin ohjaukset simppelinä ja intuitiivisena. Pääosassa pelin kohdeyleisö luultavasti määräytyy pelin graafisen suunnittelun yhteydessä, mutta en aikonut tämän työn aikana kiinnittää siihen huomiota.

	A	B
1	**Pelaajana haluan pelihahmon**	
2		Pelaajalla on Sprite
3		Pelaajalla on scripti
4		Pelaajalla on animaatiota
5		Pelaajalla on prefab
6	**Pelaajana haluan liikkua**	
7		Pelaajahahmon liikutus tiettyyn pisteeseen
8		Reitin tekeminen
9		Reitin piirtäminen
10		Ohjaukset
11	**Pelaajana haluan ampua**	
12		Ammuksen luominen
13		Ammuksen spawnaus
14		Ammuksen ampuminen
15		Ammukset voi osua
16	**Pelaajana haluan vihollisia**	
17		Ainakin yksi vihollistyyppi jolla sprite, scripti ja prefab
18		Tekoäly
19	**Pelaajana haluan haastetta**	
20		Viholliset voi ampua
21		Pelaaja voi kuolla

Kuva 6. Peliini valitsemani käyttäjätarinat ja niiden tehtäväjako.

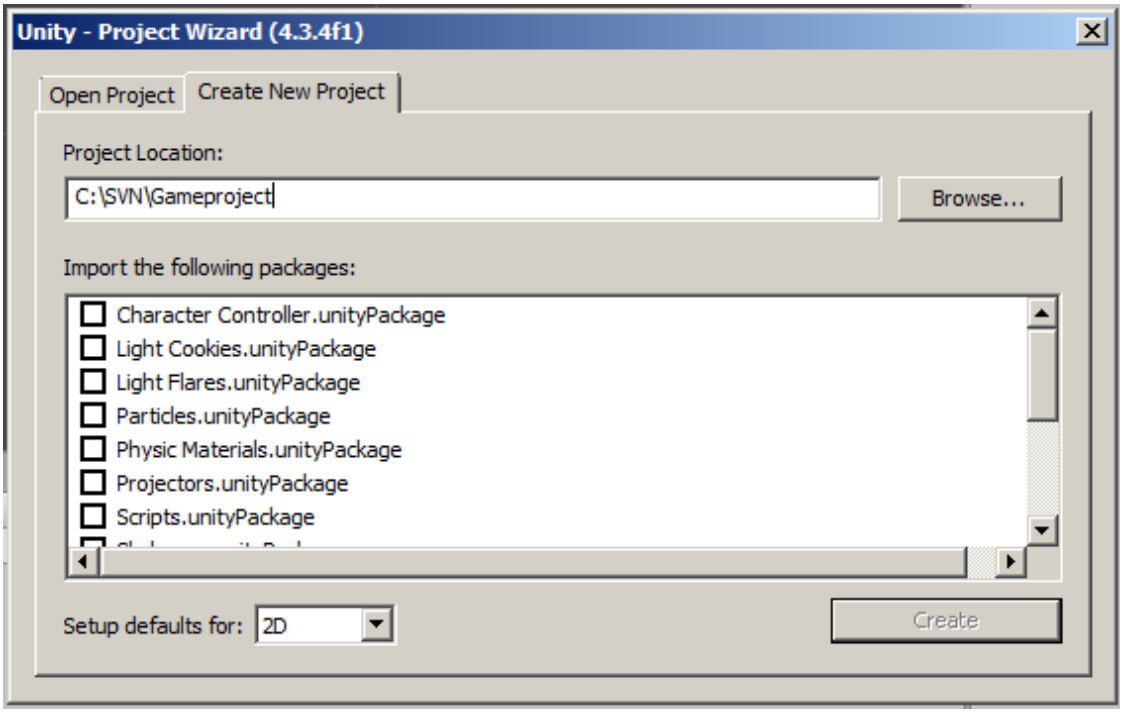
Aloitin pelin suunnittelemisen miettimällä, mitä kaikkia ominaisuuksia haluan peliin ja tekemällä käyttäjätarinalistan, jonka jaoin pienempiin tehtäviin. Lopullinen lista näkyy kuvassa 6. Käyttäjätarinalistan tekeminen auttaa selventämään kaikkia ominaisuuksia, jota haluan peliini. Tehtävälista ei ole lopullinen, vaan siihen voi lisätä tehtäviä tarpeen mukaan. Yhden miehen projekteissa siitä ei ole edes erityisemmin hyötyä, mutta se antaa suuntaa ohjelmoinnille ja auttaa jos ohjelmoinnissa ilmenee ongelmia.

En suunnitellut pelin graafista ilmettä etukäteen, koska en ole erityisen kiinnostunut artistin roolista ja pelin tekemisessä se ei ole yleensä ohjelmoijan tehtävä. Tästä syystä päädyin käyttämään Opengameart-sivustoa, josta löytyy eri artistien lisäämiä pelimateriaaleja. Pelin ulkonääksi päätin tehdä avaruuspelin, joten etsin sivustolta tähän aihealueeseen sopivia spritejä. Spritejen oli myös tarkoitus olla kaikenikäisille sopivia.



Kuva 7. Suunnitelman perusteella pelistä piirretty kuva.

Suunnittelun päätteeksi päädyin avaruuspeliin, jossa pelaaja-alus seuraa pelaajan piirtämää reittiä ja väistelee luoteja, joita pelin viholliset ampuvat. Pelin idea on arcademainen eli peli jatkuu kunnes pelaaja kuolee.



Kuva 8 Unityn projektin luonti.

4.2 Unity-asetukset

Asennettuani Unity 4.3.4-version aloitin ohjelman tekemisen luomalla uuden projektin Unityssä. Uuden projektin tekoikkuna Unityssa näkyy kuvassa 8. Valitsin projektin kansioksi ”C:\SVN\SchoolGameProject”, koska halusin muuttaa projektin myöhemmin versionhallintaan ja helposti löydettäväksi. En valinnut mitään Unityn valmispaketeista, koska niitä voi helposti tuoda projektiin myös luomisen jälkeen tarvittaessa. Lopuksi valitsin valikosta 2D-asetukset ja loin projektin.

Unityn latauduttua ruudulle ilmestyi tyhjä projekti. Tein projektin valikkoon uusia kansioita, joita arvioin tarvitsevani. Lopulta tallensin tyhjän scenen nimellä Mainscene. Projektiasetuksista vaihdoin 2D-fysiikoiden painovoiman nollaksi, jolloin ylhäältäpäin kuvatut peliobjektit eivät valu alas ruudusta painovoiman mukana. Lisäsin myös tuontiasetuksista peruspartikkelipaketin projektiini. Perusasetusten jälkeen siirryin asettamaan versionhallinnan toimimaan.

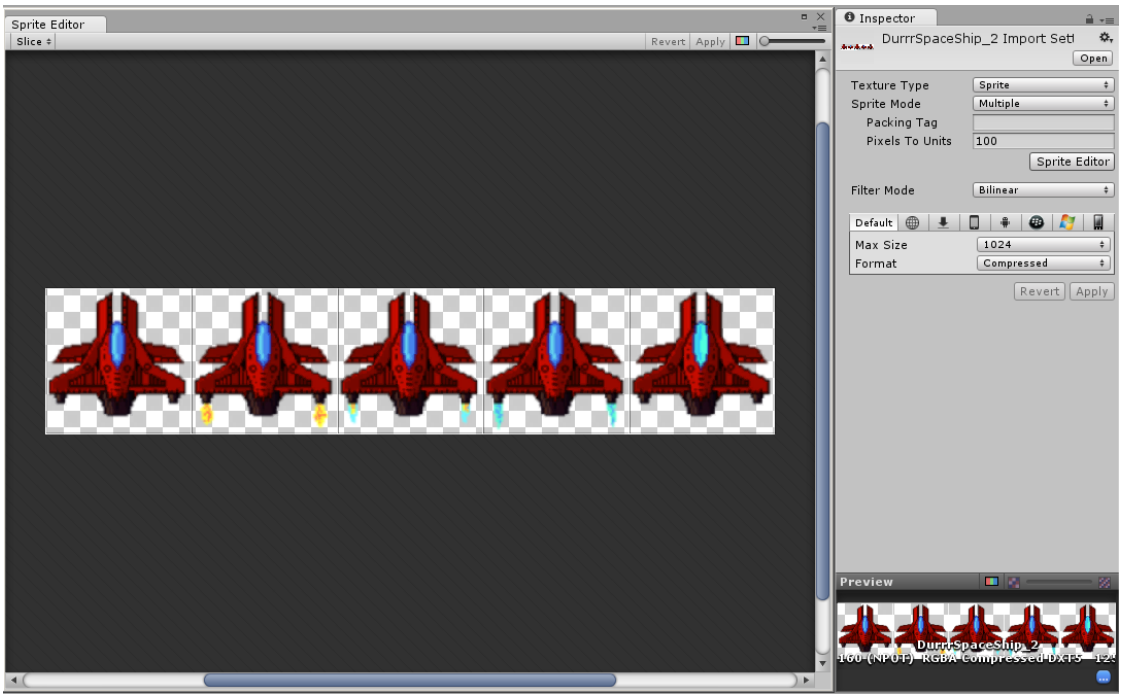
Versionhallinnan käyttäminen Unityssa edellyttää myös Unityn metatiedostojen siirtämistä. Unityssa metatiedostot pitävät yllä materiaalitiedoston uniikkia tunnistetta, jo-

ta Unity käyttää materiaalien tunnistamiseen tiedoston nimen tai sijainnin sijaan. Jos metatiedostoja ei ota huomioon ja siirtelee tiedostoja, Unity kadottaa tiedoston. Vakiona metatiedostot ovat näkymättömiä, mutta projektin editorin asetusten kautta ne voi määrätä näkyviksi. (10.)

Seuraavaksi avasin Assembla-sivuston, johon olin jo aikaisemmin kirjautunut. Käyttäen Assembla-sivustolta saatua osoitetta siirryin tekemäni projektikansion tortoiseSVN-asetuksiin, josta valitsin import. Seuraavaksi valitsin SchoolGameProject-kansiosta checkoutin, mikä tekee kansiosta versionhallinnan mukaisen aikaisempaa Assembla-osoitetta käyttäen. Versionhallinta mahdollistaa useammalla koneella työskentelemisen samaan projektiin. Se myös mahdollistaa muutosten palauttamisen, vaikka ne olisivat tapahtuneet jo päiviä aikaisemmin.

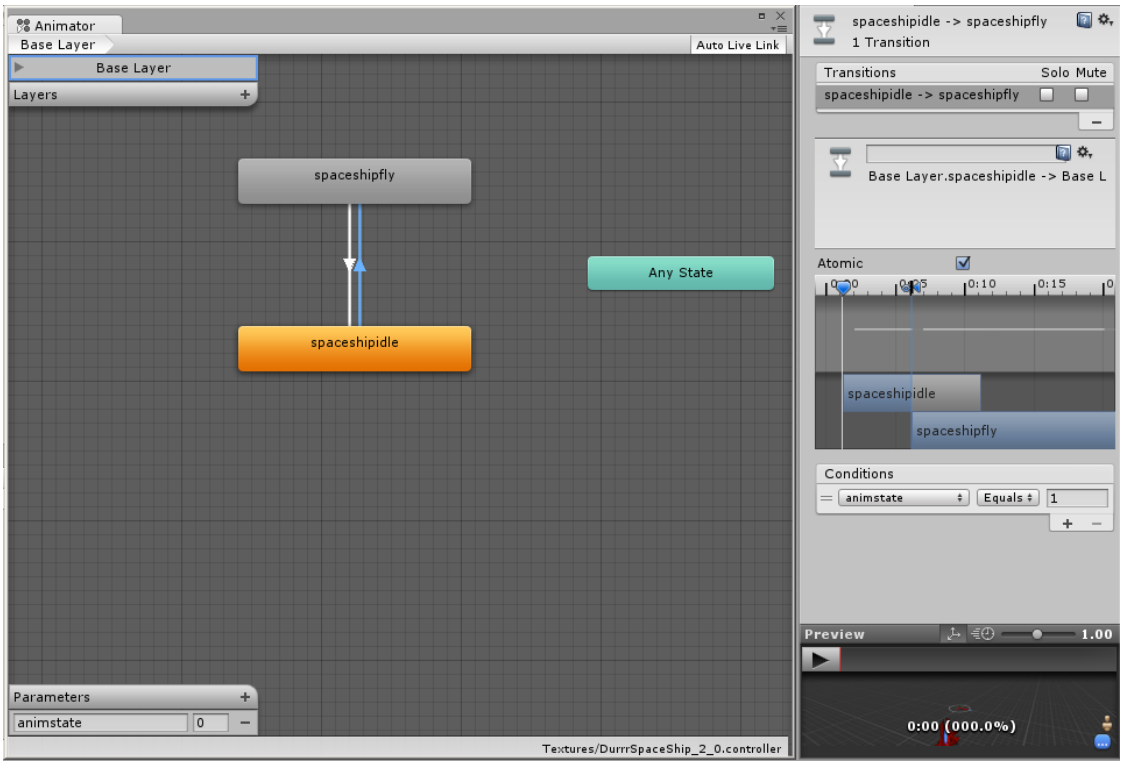
4.3 Spritet ja animaatio Unityssa

2D-spritejen ja animaatioiden luominen ja käyttäminen on Unityssa hyvin yksinkertaista. Spritet voi tuoda yksittäisinä tai isommissa kuva-atlaksissa, jolloin Unityssa on työkalut niiden erottelemiseen. Tein spriteistä kuva-atlaksia, joihin asettelin pieniä animaatioita. Kuvien taustat muokkasin läpinäkyviksi ja tallensin ne png-tiedostoina. Unity etsii projektin kansioistaan tiedostot automaattisesti ja lisää ne projektinäkymään.



Kuva 9. Unityn Sprite Editor -ikkuna.

Kuvassa 9 näkyy, kuinka olen muuttanut tiedoston tekstuurityypin spriteksi ja Sprite mode -asetuksen useammaksi eri spriteksi. Sprite editor -ikkunan kautta kuva-atlaksen jakaminen useampaan spriteen käy helposti ja kätevästi. Tässä tapauksessa tiesin jokaisen spriten olevan 160x160 pikselin kokoisia, jolloin pystyin käyttämään automaattista 160x160 pikselin taulukon tekemistä. Kuvaan 9 valitsemassani kuva-atlaksessa on 2 yksinkertaista animaatiota. Animaatiot ovat suihkumoottorein varustettu lentäminen, sekä ohjauskopin valoja välkyttävä paikallaan seisominen.



Kuva 10. Unity Animator -ikkuna.

Uusien animaatioiden luominen editorin kautta on yksinkertaista. Kun projektiluettelosta valitsee animaatioon halutut kuvat ja vie ne scenen objektiluetteloon, Unity tekee niistä automaattisesti objektin, jolla on kyseinen animaatio, ja ohjelma pyytää tallentamaan animaation. (11.)

Unityn spriteanimaatiot toimivat Animator komponentin kautta, jolle annetaan ”controller”-tiedosto. Controller-tiedosto on tilakone, jolle voi antaa useampia animaatioita ja ehdot niiden vaihtamiseen. Kuvan 10 tapauksessa tein controller-tiedostoon 2 tilaa paikallaan olo ja lento, joiden välillä on mahdolliset tilanvaihdot. Tilanvaihto tapahtuu koodin kautta muuttamalla controllerin muuttujaa animstate, jonka arvot asetetaan tilan vaihdoille. Esimerkiksi rivi ”animator.SetInteger("animstate",1);” vaihtaa luokan animaattorin animstate muuttujan arvoon 1, joka tässä tapauksessa tarkoittaa lento-animaatiota. (11.)

Controller-tiedostoon on mahdollista lisätä useita eri muuttujia tai animaatiokierroksia, joka takaa monimutkaisten ja käytännöllisten animaatioiden yksinkertaisen ja ripeän kehittämisen.

4.4 Skriptit

Unity tukee useita eri ohjelmointikieliä skriptien kirjoittamiseen. Mahdollisia ovat Javascript, C# tai Boo ohjelmointikielet. Valitsin pelini pääohjelmointikieleksi C#. Skripti-tiedostojen käyttäminen Unityssa tapahtuu liittämällä ne peliobjektiin skriptikomponentilla. Kun komponentin omistava peliobjekti on lisättynä sceneen, se alkaa kutsua skriptinsä Update-funktiota, joka on vakiona jokaisessa skriptissä. Update-funktioiden lisäksi skripteissä on myös Start-funktio, jota kutsutaan kerran ohjelman käynnistyessä. Muita vakioskriptifunktioita peliobjektit perivät isäntäobjektinsa komponenteilta. (12.)

Unity-editorin kautta aseteltavia skriptin muuttujia saa peliobjekteille lisäämällä skriptiin public-tyypin muuttujia, jolloin editori huomaa sen automaattisesti ja lisää sen peliobjektin skripti-komponenttiin. Public-muuttujaksi pystyy lisäämään myös prefab-tyyppisiä esineitä, jolloin niiden luominen koodin kautta on helppoa.

4.4.1 GameManager

Tein peliini GameManager-skriptin. Skriptin ideana on pitää peliä pyörimässä. Skriptin ansiosta pystyin jättämään pelin scenen miltei tyhjäksi ja välttämään kenttäsuunnittelun. Skripti pitää yllä itse tekemääni aikaskaalaa, joka määrää ajan kulkua pelissä tapahtuvien hidastusten aikana. Halusin tehdä uuden aikaskaalan, koska Unityn perusaikaluokka ei toiminut aivan halutulla tavalla. Pelin aikana skriptin perivä objekti tarkkailee, onko pelikentällä tarvittava määrä vihollisia ja luo uusia, jos niitä puuttuu. Vihollisten luomiseen on luokalla oma funktionsa. Luokalla on myös funktio GetInstance, joka palauttaa luodun peliobjektin instanssin. Funktion avulla pääsee käsiksi GameManagerin funktioihin muista kooditiedostoista.

Luokan Spawnenemy-funktio etsii pelikameran alueelta arvotun alueen. Tarkistettuaan alueen saatavuuden fysiikka2D-laskulla se luo uuden vihollisen paikalle. Vihollisen luonti koodin kautta tapahtuu Instantiate-funktion avulla, jolle annetaan parametreiksi vihollistyypin prefab, sekä koordinaatit johon vihollinen luodaan (13.).

4.4.2 PlayerShip

Playership-skriptin tehtävänä on liikuttaa pelaajan hahmoa scenessä, sekä kuunnella pelaajan antamia ohjausliikkeitä. Skripti muokkaa GameManagerille asetettua pelin pysäytysominaisuutta riippuen siitä, että onko pelaajan hahmo liikkeessä. Tämä tekee pelin kulusta sopivamman kosketusnäyttölaitteelle ja ohjauksilleni, koska niiden käyttäminen nopeasti on hankalaa. Skripti myös päivittää pelaajan animaatiotilaa riippuen pelaajan liikkumisesta.

Kuva 11. Pelikuva josta näkyy pelaajan ohjausviiva piirrettynä.

Pelaaja-skriptillä on piirtofunktio, joka on hyvin tärkeä osa pelidemoa. Se käyttää Unityn omaa LineRenderer-luokkaa ja -komponenttia, joka on asetettuna pelaajan peliobjektille. LineRenderer piirtää pelialueelle viivan kahden tai useamman pisteen välille. Funktio tarkastaa hiiren tai painetun sormen läheisyyden pelaajan hahmoon ja alkaa pistämään muistiin pisteitä reitiltä, jota pelaajan hiiri tai sormi piirtää. Funktio ei tallenna uusia pisteitä liian kaukaa, jonka takia liian nopea osoittimen liikuttaminen ei toimi, mutta lyhyillä pisteillä reitistä tulee paljon pyöreämmän näköinen suoran sijaan. Valmiissa pelissä piirretty viiva näkyy kuvassa 11.

Skripti myös päivittää pelaajan ampumista, joka on mahdollista vain pelaajan aluksen ollessa liikkeessä. Luodin ampuminen tapahtuu luomalla luodin aluksen vierelle pelaajan sormen painalluksen suuntaan käännettynä, jolloin luoti käy päivittämään itseään. Luodulle ammukselle annetaan myös pelaajan nimi, jolloin sitä ei oteta huomioon pelaajaan osuessa.

4.4.3 Enemyship

Enemyship-skripti ohjaa pelin vihollisaluksia. Koska GameManager määrää vihollisaluksen paikan, on sen tekoäly hyvin yksinkertainen. Vihollisaluksen päivitysfunktio laskee sopivaa aikaa ampumiselle arvotun numeron avulla, ja siirtyy ampumiseen tarkoitettuun funktioon, kun tilanne on sopiva.

Ampumafunktio etsii pelaajan aluksen ja luo sarjan ammuksia pelaajan suuntaa. Luodut ammukset nimetään vihollisaluksen mukaan, jolloin ne eivät ota vihollisaluksia huomioon osumissaan.

4.4.4 Projectile

Projectile-skripti ohjaa peliin luotuja ammuksia. Kun ammus on luotu, se alkaa laskea olemassaoloaikaansa. Jos aika ehtii kulua loppuun ennen osumaa, ammus tuhotaan automaattisesti. Pelissä tämä tapahtuu ruudun ulkopuolella. Tämä varmistaa, ettei pelissä ole loputtomasti objekteja liikkumassa ja käyttämässä laitteen tehoja. Skripti päivittää peliobjektin paikkaa scenessä eteenpäin ja tarkistaa mahdolliset osumat Unityn 2DCollider-komponenttien avulla.

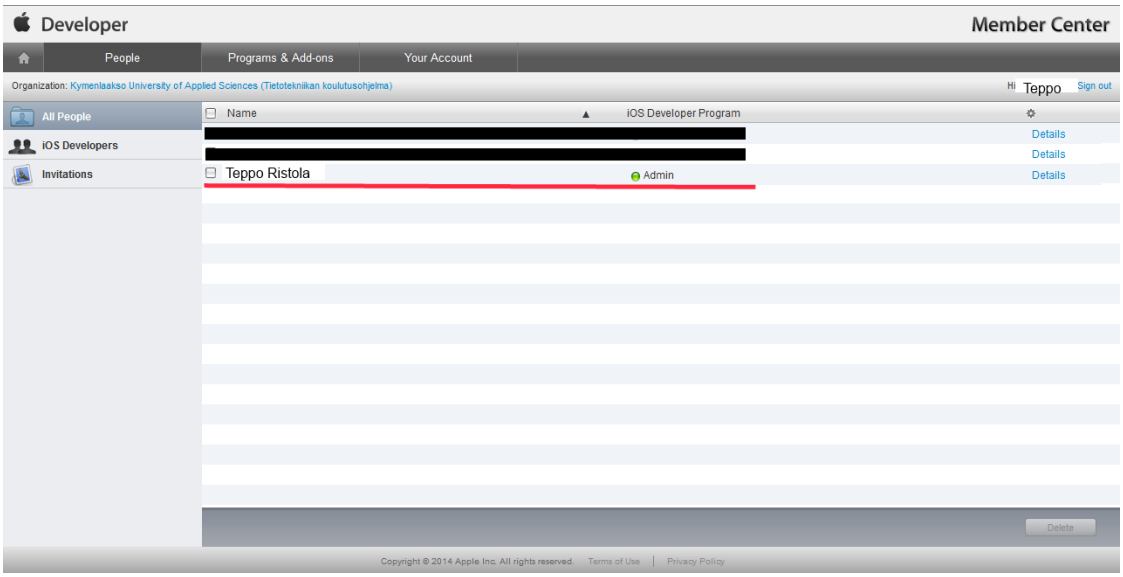
Kaikilla pelissä olevilla peliobjekteillani on komponenteissaan ”Collider”. Collider on komponentti, joka on tarkoitettu tarkastamaan eri peliobjektien kosketuksia scenessä. Kun ammus osuu pelin aikana toiseen collideriin, skripti ajautuu automaattisesti OnTriggerEnter2D funktioon, jossa osuneista peliobjekteista tarkistetaan tunniste ja kutsutaan objektin osumafunktiota tarvittaessa. Osuessaan ammus myös luo osumapisteeseen räjähdyspartikkelin prefabista ja tuhoaa itsensä.

4.5 Asettelu scenessä

Pelin asettelu scenessä on hyvin yksinkertainen, koska suurin osa pelin toiminnallisuudesta luodaan skriptien kautta. Alkutilanteessa pelin scenessä on vain pelaaja-alus, GameManager, sekä pääkamera.

4.6 iOs-versio pelistä

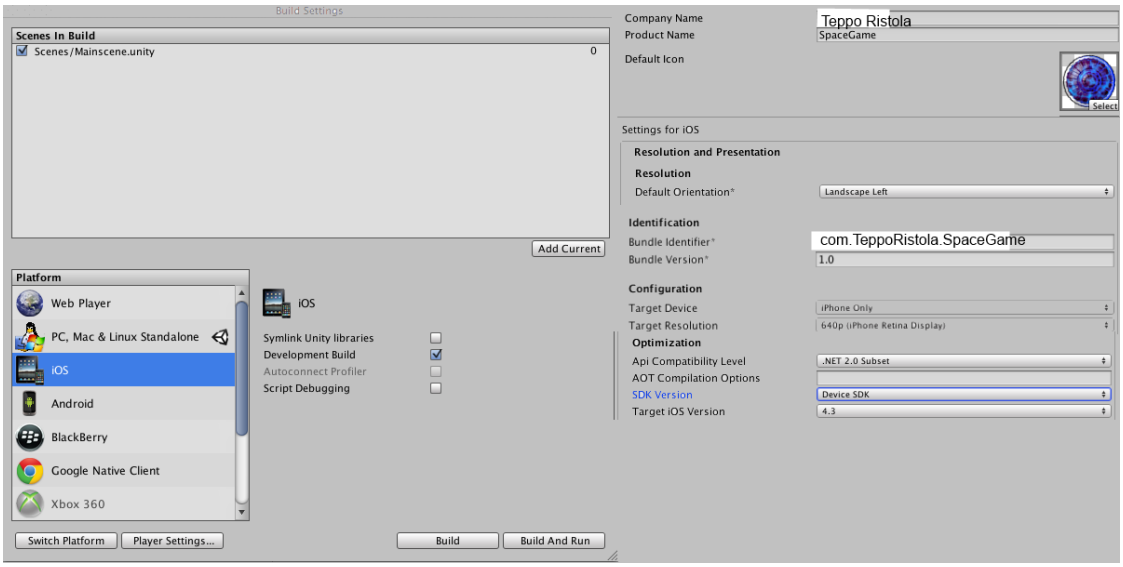
Pelien iOs-versioiden tekeminen edellyttää Applen käyttöjärjestelmien käyttämistä, joten tehtävään tarvitsee ainakin ylimääräisen Mac-tietokoneen, sekä jonkin laitteen missä testata peliä. Ennen iOs-version tekemisen aloittamista asensin Mac tietokoneelleni Applen Xcode-ohjelman uusimman version ja saman version Unitysta, jota olin pelin kehittämiseen käyttänyt. Koska Apple haluaa pitää ohjelmistonsa suljettuna ympäristönä, Apple-laitteille ohjelmoiminen edellyttää lisenssin hankkimista.



Kuva 12. Apple member center -sivusto.

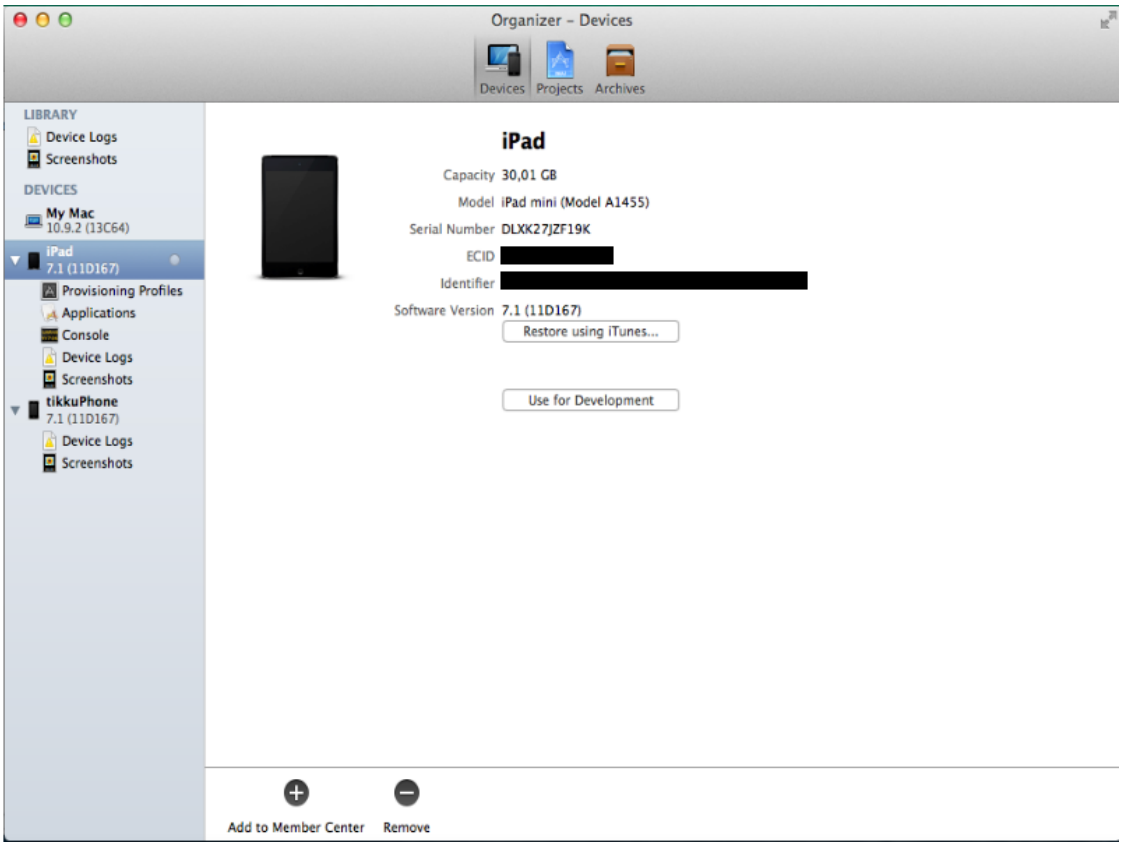
Lisenssin pystyin hankkimaan koulun kautta. Itse hankittuna siihen kuuluisi vuosimaksu. Lisenssin luominen oli yksinkertainen prosessi, koska koululla oli kaikki valmiina ja se vain edellytti lyhyttä nettilomakkeen täyttämistä. Lomakkeen täyttämistä auttoi myös se, että minulla oli jo entuudestaan Apple id käytössä puhelimessani. Kuvassa 12 näkyy Applen kehittäjäsivusto, josta näkyy kaikki ryhmän jäsenet. Nettisivu löysi tililtäni väärän nimen, mutta selvyyden takia muutin nimen omakseni tekstissä ja kuvissa.

Siirrettyäni peliprojektini tiedostot Mac-tietokoneelleni, käynnistin projektin Unitylla. Unityn käyttäminen OS-X -käyttöjärjestelmällä tapahtuu samalla tavalla, kuin Windowsillakin.

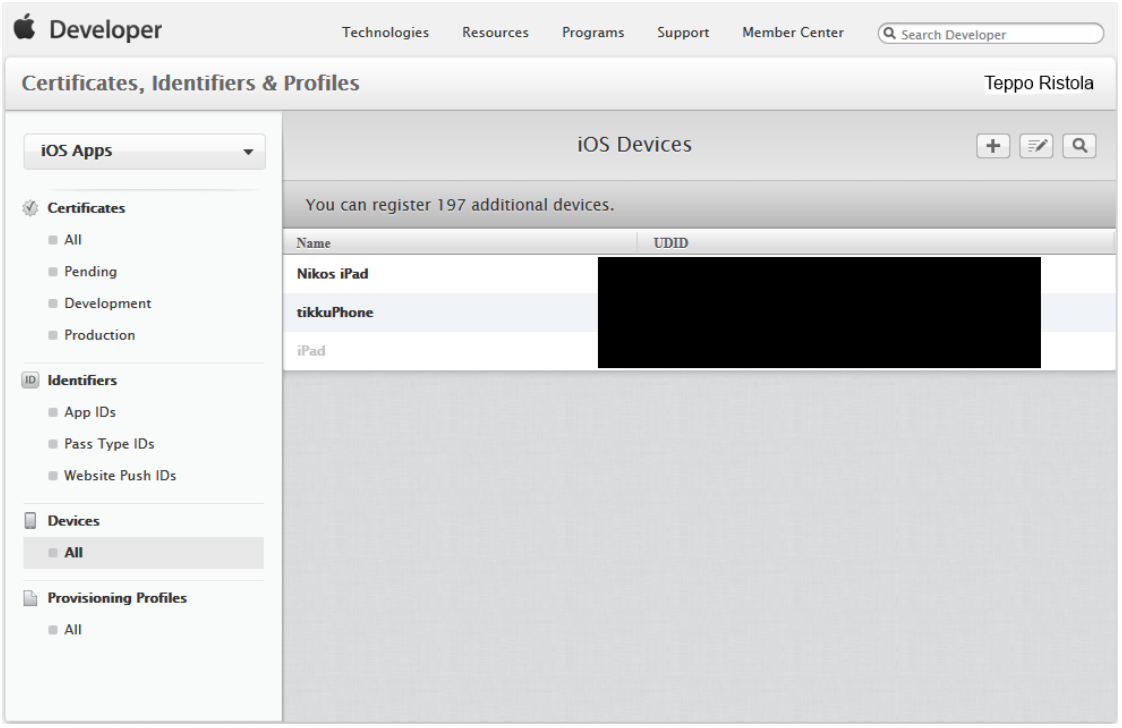


Kuva 13. Unity build ja Player Settings -valikko.

Tässä tapauksessa projektini oli valmis, joten siirryin kuvassa 13 näkyvään build settings -valikkoon. Unityn asetuksista valitsin iOs-buildin ja valitsin pelin Mainscene-tiedoston käännettäväksi. Player-asetuksista asetin tekijän nimeksi Apple kehittäjäprofiilini ja pelin nimeksi SpaceGame. Näistä asetuksista syntyy pelin pakettitunniste joka pelini tapauksessa oli com.TeppoRistola.SpaceGame. Pelin pakettitunniste pitää olla oikea, jotta pelin sertifikointi onnistuu. Lisäksi muokkasin pelille ikonin viholisalusspritesta. Asetin buildin luomaan iPhone retina-näytölle sopivan kuvan. Vaihdoin asetuksista myös laite ohjelmointiympäristön simulaattorin sijaan, jolloin Xcode osaa kääntää ohjelman oikealle laitteelle. Applen laitteille käännettäessä Unity tekee automaattisesti Xcode-projektitiedoston.

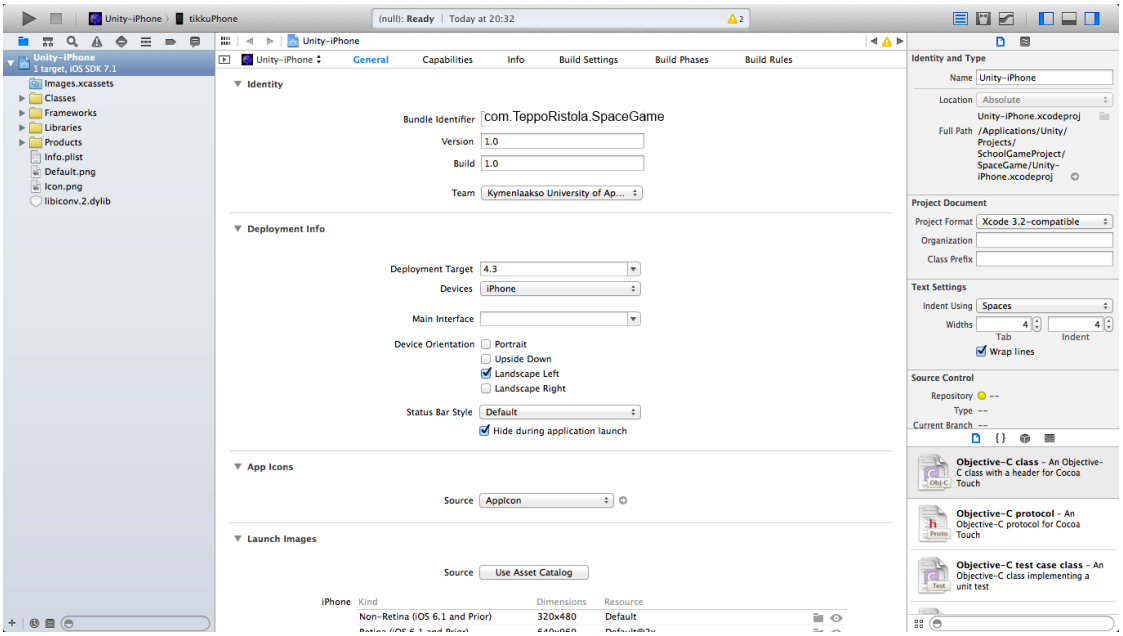


Kuva 14 Xcode-ohjelman Organizer-ikkuna.



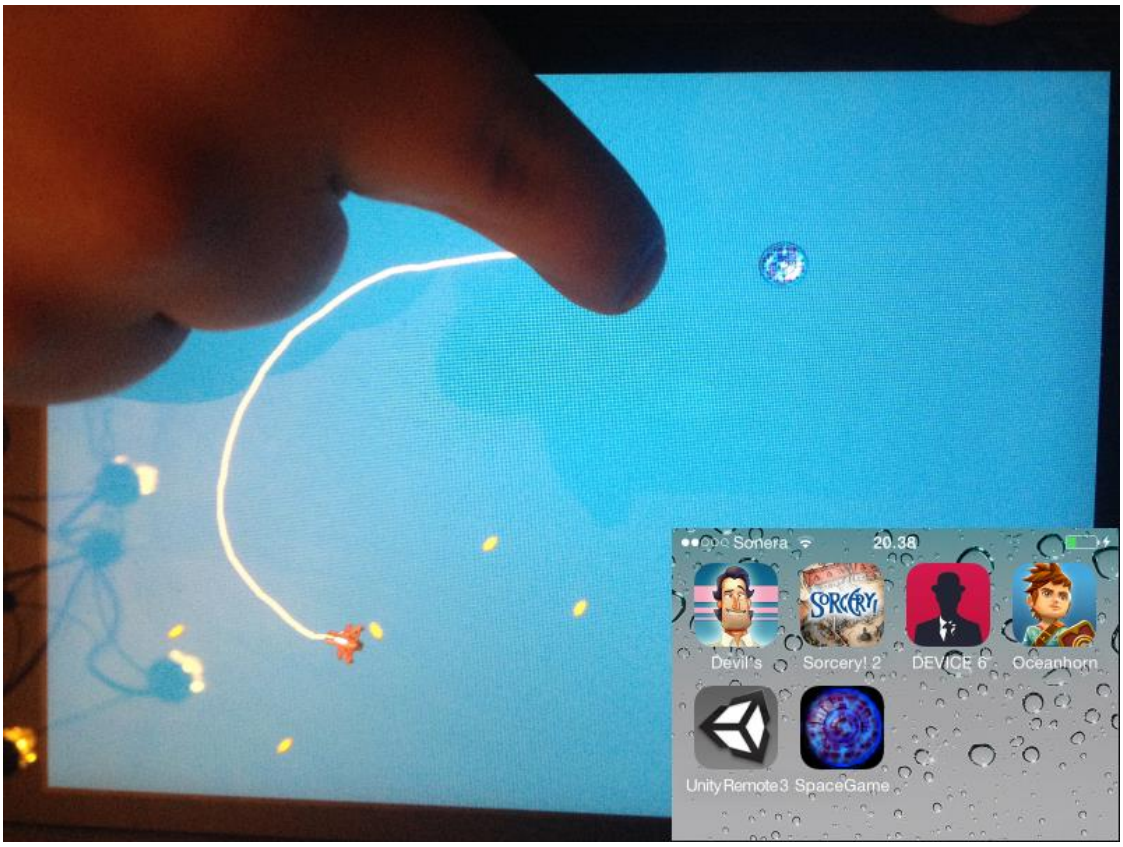
Kuva 15. Applen Developer-sivuston laitenäkymä.

Xcode-ohjelmassa siirryin kuvassa 14 näkyvään device organizer -ikkunaan. Ikkunassa lisätään member centeriin koneeseen kiinnitettyjä käytettäviä laitteita. Laitteitten lisääminen vaatii kehittäjälisenssiin ainakin admin-tason oikeuksia. Pelini tapauksessa lisäsin iPhone4S -puhelimen, sekä iPad mini -tabletin. Ohjelma asettaa laitteille automaattisesti sertifikaatit. Kun sertifikaatin lisääminen onnistuu, laite tulee näkyviin kuvassa 15 näkyvään kehittäjäsivustoon, josta tarkastin laitteen lisäyksen onnistuneen. (14.)



Kuva 16 Xcode-projektin perusnäkymä ja asetukset.

Xcode-projektin asetuksista vaihdoin pelin kehittäjätiimin koulun tiimin mukaiseksi ja muita laitekohtaisia asetuksia. Asetusten ollessa kohdallaan ohjelman kääntäminen onnistuu ja ohjelma siirtää pelin koneeseen kiinnitettyyn laitteeseen, joka näkyy kuvan 16 ylälaidasta. Kun kääntäminen onnistui, peli ilmestyi puhelimen valikkoon valittavaksi ohjelmaksi.



Kuva 17. Peli pelattavana iPad mini -tabletilla.

Kuvassa 17 näkyy peli käynnissä ja pelattavana kosketusnäyttölaitteessa, sekä pelin ikonin ulkonäkö puhelimen valikossa. Testauksen perusteella peli pyörii moitteettomasti iPhone4S -puhelimella, sekä pelin ohjaukset toimivat hyvin kosketusnäytöllä.

5 YHTEENVETO

Työni tekeminen eteni suunnitellun mukaisesti. Lopputuloksena työstäni tuli kosketusnäyttölaitteella pelattava pelimekaniikkaa esittelevä proof of concept demo. Opin työni aikana valmistamaan pelejä Unity-pelimoottorilla ja uskon siitä olevan hyötyä tulevaisuudessa. Sain myös kokemusta Applen laitteille ohjelmoitaessa tarvittavista toimenpiteistä.

5.1 Pelistä

Vastaisuudessa aion luultavasti jatkaa pelinkehitystä vapaa-ajallani. Pelin kehitys jatkuu uusien ominaisuuksien suunnittelemisella ja toteuttamisella. Jos aion tehdä pelin julkaistavaan kuntoon asti, on tärkeää kiinnittää enemmän huomiota pelin ulkokuoreen. Uniikki ja siisti graafinen ulkonäkö ja äänimaailma tekevät peleistä paljon parempia. Niiden ansiosta huonosti pelattavat pelit voivat tuntua hyviltä, tai hyvät pelit voivat tuntua vielä paremmilta. Tämän takia pelin kehittämistä auttaisi jonkun graafisesta suunnittelusta kiinnostuneen kaverin löytäminen

Peliä voisi mielestäni kehittää kahteen eri suuntaan. Yksi suunta on arcademainen peli, jossa tarkoituksena on vain selviytyä hengissä. Toinen suunta olisi rakentaa pelin ympärille isompi avaruusstrategiakokonaisuus ja kehittää pelin pohjalta taktinen taistelusysteemi.

En ollut aivan tyytyväinen pelaaja-aluksen liikkumisfunktioihin, joten luultavasti muokkaisin niitä jonkin verran. Pelaajan kääntymisestä voisi tehdä hitaampaa ja liikkumisesta enemmän painovoimaa tottelevaa, jolloin peli tuntuisi paljon aidommalta pelata.

5.2 Työkaluista

Valitsin työni tekemiseen hyvin käytännölliset ja joustavat työkalut. Unity oli pelimoottorina erittäin käytettävän tuntuinen ja helposti opittavissa, joten jos vastaisuudessa teen peliprojektia, on Unity erittäin todennäköinen valinta siihen. Applen ohjelmointisysteemit tuntuvat turhauttavan sekavilta, mutta niissä ei ole erityisemmin valinnan varaa, jos haluaa tehdä pelejä Applen kosketusnäyttölaitteille. Työkaluista vaihtaisin vain Paint.net tilalle jonkin paremman kuvankäsittelyohjelman kuten Adobe Photoshopin.

LÄHTEET

1. RedLynx. Kotisivut. Drawrace2. http://www.redlynx.com/games/ios/drawrace2 [Viitattu 26.4.2014]

2. Kukouri. Kotisivut. Tiny Troopers pelisarja. http://www.kukouri.com/games/tinytroopers [Viitattu 26.4.2014]

3. Unity. Peliobjekteista. 2010 http://docs.unity3d.com/Documentation/Manual/GameObjects.html [Viitattu 4.5.2014]

4. Unity. Komponenttien käytöstä. 2013. http://docs.unity3d.com/Documentation/Manual/UsingComponents40.html [Viitattu 4.5.2014]

5. Unity. Learning the interface. 2013. http://docs.unity3d.com/Documentation/Manual/LearningtheInterface.html [Viitattu 26.4.2014]

6. Unity. Unity 2D-power. 2013. http://unity3d.com/pages/2d-power [Viitattu 26.4.2014]

7. MonoDevelop FAQ. 2014. http://monodevelop.com/FAQ [Viitattu 26.4.2014]

8. About TortoiseSVN. 2014. http://tortoisesvn.net/about.html [Viitattu 26.4.2014]

9. Opengameart sivuston tarkoitus. 2014. http://opengameart.org/content/faq#q-proprietary [Viitattu 26.4.2014]

10. Oosten, J. Using version control with Unity. 2013. http://www.3dgep.com/?p=5105#Meta_Files [Viitattu 26.4.2014]

11. Cummings, M. Unity 4.3-version animoitujen spritejen luonti. 2013. http://michaelcummings.net/mathoms/creating-2d-animated-sprites-using-unity-4.3 [Viitattu 4.5.2014]

12. Unity. Skriptien tekeminen ja käyttö. 2013. http://docs.unity3d.com/Documentation/Manual/CreatingAndUsingScripts.html [Viitattu 4.5.2014]

13. Unity. Instantiate. 2013. http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html [Viitattu 2.5.2014]

14. Apple inc. Xcode laitteiden lisäys. 2013. https://developer.apple.com/library/ios/recipes/xcode_help-devices_organizer/articles/provision_device_for_development-generic.html [Viitattu 4.5.2014]