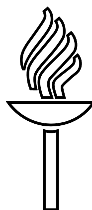
Jani Ylönen

VIDEOPELIEN HISTORIA JA PELINKEHITYS

2D-PELIMOOTTOREIDEN VERTAILU



JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2014

TIIVISTELMÄ

Ylönen, Jani
Videopelien historia ja pelinkehitys – 2D-pelimoottoreiden vertailu
Jyväskylä: Jyväskylän yliopisto, 2014, 92 s.
Tietojärjestelmätiede, Pro Gradu -tutkielma
Ohjaaja: Puuronen, Seppo

Videopelien historia alkoi 1940-luvun lopulta ja on 2010-luvulla nopeimmin kasvava viihdeteollisuuden ala, niin Suomessa kuin maailmanlaajuisestikin. Tekniikan kehittymisen myötä myös pelit ja niiden kehittäminen ovat muuttuneet. Peleistä on tullut entistä laajempia ja näyttävämpiä, samalla kuitenkin kehityskustannukset ja kehitysajat ovat kasvaneet. Mobiililaitteet kuten älypuhelimet ja tabletit, sekä digitaalinen jakelu ovat muuttaneet alaa 2000-luvulla, ja mahdollistaneet jälleen pienten studioiden menestymisen yksinkertaisilla peli-ideoilla. Pelinkehitysvälineiden kehittyminen on helpottanut ja nopeuttanut videopelien tekemistä, ja yksinkertaisimmilla pelimoottoreilla voidaan toteuttaa pelejä jopa ilman ohjelmointia. Tässä teoreettis-käsitteellisessä tutkielmassa perehdytään kirjallisuuden pohjalta videopelien historiaan, niiden kehityksen muutoksiin sekä yleiskäyttöisiin pelinkehitysvälineisiin. Tutkimus selvittää kehityksessä käytettävien rajapintojen ja pelimoottoreiden käyttötarkoituksen, ja esittelee vuonna 2014 pelinkehittäjien keskuudessa viisi suosituinta pelimoottoria. Tarkempaan tarkasteluun valikoituneissa kehitysvälineissä on kriteerinä käytetty kykyä alustariippumattomaan kehitykseen, erityisesti mobiililaitteille. Tuloksena saatua tietämystä on käytetty empiirisessä osuudessa, jossa on vertailtu käytännössä kahden vaikeustasoltaan erilaisen pelimoottorin ominaisuuksia. Lisäksi Space Invaders pelistä on toteutettu prototyypit.

Asiasanat: videopeli, pelimoottori, luokkakirjasto, rajapinta, pelinkehitys, mobiililaitteet, Cocos2d, Unity

ABSTRACT

Ylönen, Jani
History of video games and game development – 2D game engine comparison
Jyväskylä: University of Jyväskylä, 2014, 92 p.
Information Systems, Master's Thesis
Supervisor: Puuronen, Seppo

History of video games began in the late 1940s and it's the fastest growing business in entertainment industry in Finland and globally in 2010s. Along with development of technology, games and their development has also changed. Games have become larger and more impressive, though at the same time the development costs and times have increased. Mobile devices, such as smartphones and tablets, as well as digital distribution have changed the business after the year 2000 and made it possible for small game studios to success again with simple game ideas. Evolution of game development tools has made video game development easier and faster and with simplest game engines it's possible to implement games even without programming. This theoretical-conceptual thesis studies the history of video games, the evolution of their development and general-purpose game development tools on the basis of literature. The thesis examines the purpose of interfaces and game engines used in video games and presents five most popular game engines in 2014 among developers. The selection of the development tools for a more detailed examination is based on their ability to offer platform independent development, especially for mobile devices. The resulting knowledge will be subsequently used in empirical part, where two game engines were compared and a prototype of Space Invaders video game was implemented.

Keywords: video game, game engine, programming library, programming interface, game development, mobile devices, Cocos2d, Unity

KUVIOT

KUVIO 1 OXO, Tennis for Two ja Spacewar! 13
KUVIO 2 Brown Box, Pong ja Computer Space 14
KUVIO 3 Space Invaders, Pac-Man, Donkey Kong Game & Watch 15
KUVIO 4 Commodore 64 ja ZX Spectrum 16
KUVIO 5 Atari VCS ja E.T. 17
KUVIO 6 Pinball Construction Set ja Shoot 'em Up Construction Kit 18
KUVIO 7 Amiga, Intro, IBM PC 20
KUVIO 8 Maniac Mansion ja Doom 21
KUVIO 9 Matopeli, Angry Birds ja iPhone 22
KUVIO 10 Pelisilmukka 28
KUVIO 11 Työtuntimäärien vertailu ilman kehystä ja kehyksen kanssa 30
KUVIO 12 Pelimoottorin uudelleenkäytettävyyden kirjo 31
KUVIO 13 Space Invaders 35
KUVIO 14 Pelin vaikeustason kasvu pelin edetessä 35
KUVIO 15 Space Invadersin sprite sheet 38
KUVIO 16 Juoksuanimaation sprite sheet pelistä Prince of Persia 39
KUVIO 17 Syvyysvaikutelma 3D- ja 2D-näkymissä 46
KUVIO 18 Unityn editori 47
KUVIO 19 Cocos2d-x ja Visual Studio Express 2012 48
KUVIO 20 Texture Packerilla luotu sprite sheet 53
KUVIO 21 Unity – Sprite Editor 54
KUVIO 22 Texture Packer 54
KUVIO 23 Unity – Pelin kuvasuhteen valinta 56
KUVIO 24 Unity – Peliobjektin komponentit 57
KUVIO 25 Unity – Fysiikkamoottorin asetukset 65
KUVIO 26 Unity – Vihollisen hitboxin määritys 68
KUVIO 27 Unity – Äänitiedostojen liittäminen skriptiin 70
KUVIO 28 Unity – GUISkin 73
KUVIO 29 Unity – Valikko 74
KUVIO 30 Cocos2d-x – Valikko 77
KUVIO 31 Unity – Valmis taso editorissa 78
KUVIO 32 Unity – Pelin käännösikkuna 79

TAULUKOT

TAULUKKO 1 Pelimoottoreiden kolme tasoa 31
TAULUKKO 2 Cocos2d:n kehityshaarat 37
TAULUKKO 3 Cocos2d-x:n ja Unityn tukemat tietokone- ja mobiilialustat 37
TAULUKKO 4 Lähdekoodin syntaksin väritys 49
TAULUKKO 5 Unity – Pelissä käytettävät assetit 49
TAULUKKO 6 Unity – Skriptien tavallisimmat metodit 52
TAULUKKO 7 Cocos2d-x – Action olion perustoiminnot 61
TAULUKKO 8 Unity – Törmäysten aiheuttamat tapahtumat 68

KOODIT

LÄHDEKOODI 1 Cocos2d-x – Projektitiedostojen luonti Python-skriptillä 50
LÄHDEKOODI 2 Cocos2d-x – MenuScenen luonti ja käynnistys 50
LÄHDEKOODI 3 Cocos2d-x – Update metodin tahdistuksen asetus 51
LÄHDEKOODI 4 Unity – C#-skriptin pohja 52
LÄHDEKOODI 5 Cocos2d-x – Spritejen lataus kuvatiedostosta 53
LÄHDEKOODI 6 Cocos2d-x – Sprite sheetin luku ja siitä spriten luonti 55
LÄHDEKOODI 7 Cocos2d-x – Suunnitteluresoluution asetus 55
LÄHDEKOODI 8 Unity – Pelaajan syötteen luku ja lasertykin siirto 59
LÄHDEKOODI 9 Cocos2d-x – Pelaajan syötteen kuuntelijat 60
LÄHDEKOODI 10 Cocos2d-x – Spriten siirto 61
LÄHDEKOODI 11 Cocos2d-x – Pelaajan tykin kontrollointi 62
LÄHDEKOODI 12 Unity – Vihollislaivaston siirtoskripti 63
LÄHDEKOODI 13 Coco2d-x – Vihollislaivaston siirto 64
LÄHDEKOODI 14 Cocos2d-x – Fysiikat pelimaailmalle 66
LÄHDEKOODI 15 Cocos2d-x – Törmäysten kuuntelijoiden määrittely 67
LÄHDEKOODI 16 Cocos2d-x – Spritelle kappaleen fyysisen muodon asetus 67
LÄHDEKOODI 17 Cocos2d-x – Osumien käsittelyä 69
LÄHDEKOODI 18 Unity – Ääniefektien soittoskripti 71
LÄHDEKOODI 19 Unity – Ääniefektin soitto tulituksessa 71
LÄHDEKOODI 20 Cocos2d-x – Ääniefektien ja musiikin lataus ja soitto 72
LÄHDEKOODI 21 Unity – Pelinaikainen käyttöliitymä 73
LÄHDEKOODI 22 Cocos2d-x – Käyttöliittymätekstien määrittely ja päivit 74
LÄHDEKOODI 23 Unity – MenuScript piirtää valikon napit 75
LÄHDEKOODI 24 Cocos2d-x – Valikon napin luonti spriteä käyttäen 76
LÄHDEKOODI 25 Cocos2d-x – Valikon luonti 76
LÄHDEKOODI 26 Cocos2d-x – Siirtyminen skenejen välillä 77

SISÄLLYS

TIIVISTELMÄ
ABSTRACT
KUVIOT
TAULUKOT
LÄHDEKOODIT
SISÄLLYS

1 JOHDANTO 9

2 VIDEOPELIEN HISTORIA 11
2.1 Videopelien aamunkoitto 11
2.2 Kulta-aika 14
2.3 Suuri videopeliromahdus Pohjois-Amerikassa 1983 16
2.4 Kenttäeditorit 17
2.5 Piratismi 18
2.6 Ensimmäiset pelimoottorit ja pelien modaus 19
2.7 Mobiilipelit 21
2.8 Yhteenveto 22

3 VIDEOPELIEN KEHITYS 24
3.1 Kehityksen evoluutio 24
3.2 Kehitystiimin tehtävät 26
3.3 Kehitysprosessi 26
3.4 Peli ohjelmana 27
3.5 Luokkakirjastot 28
3.5.1 Grafiikkakirjastot 29
3.5.2 Audiokirjastot 29
3.5.3 Fysiikkakirjastot 29
3.5.4 Multimediakirjastot 29
3.6 Pelimoottori 29
3.7 Suosittuja usean alustan pelimoottoreita 2014 32
3.7.1 App Game Kit – The Game Creators 32
3.7.2 Cocos2d-x – Open Source 32
3.7.3 GameMaker: Studio – YoYo Games 32
3.7.4 Unity – Unity Technologies 33
3.7.5 Unreal Engine 4 – Epic Games 33
3.8 Yhteenveto 33

4 SUUNNITELMA PELIN TOTEUTUKSESTA 34
4.1 Toteutettava peli 34
4.2 Ympäristö 36

4.2.1 Kohdeympäristöt 36
4.2.2 Kehitysvälineet 36
4.3 Kehitysvaiheet 38
4.3.1 Resurssien lataus 38
4.3.2 Grafiikka ja animaatio 39
4.3.3 Peliobjektien ja pelimaailman määrittely 40
4.3.4 Pelaajan syötteen käsittely 40
4.3.5 Peliobjektien liikuttaminen 41
4.3.6 Törmäystarkistukset ja käsittely 41
4.3.7 Äänet 42
4.3.8 Käyttöliittymä 42
4.3.9 Valikot 42
4.4 Tiedon keruu eri vaiheissa 42
4.5 Yhteenveto 43

5 PELIN TOTEUTUS 44
5.1 Kohdelaitteiden suuri kirjo 44
5.2 Pelimoottoreiden dokumentaatio 45
5.3 Peruskäsitteet 45
5.4 Kehitysympäristöjen käyttöönotto 45
5.5 Pelimoottoreiden editorit 47
5.6 Kehitys vaiheittain 49
5.6.1 Alustavat toimet 49
5.6.2 Skriptit 52
5.6.3 Resurssien lataus 53
5.6.4 Grafiikka ja animaatio 53
5.6.5 Peliobjektien ja pelimaailman määrittely 56
5.6.6 Pelaajan syötteen käsittely 58
5.6.7 Peliobjektien liikuttaminen 60
5.6.8 Törmäystarkistukset ja käsittely 65
5.6.9 Äänet 70
5.6.10 Käyttöliittymä 72
5.6.11 Valikot 73
5.7 Asennuspaketin luonti eri alustoille 77
5.8 Yhteenveto 79

6 TULOKSET JA JOHTOPÄÄTÖKSET 80

7 YHTEENVETO 83

LÄHTEET 84

LIITE 1 KONSOLISUKUPOLVET 88

LIITE 2 KOTIMIKROT 90

LIITE 3 UNITY-PELIN KUVANKAAPPAUKSET 91

LIITE 4 COCOS2D-X-PELIN KUVANKAAPPAUKSET 92

1 JOHDANTO

Ajatus videopeleistä syntyi ensimmäisen kerran jo 1940-luvun lopulla, mutta kaikkien ulottuville ne saatiin vasta 1970-luvun alussa. Pelit ovat parantuneet kaikilla osa-alueilla. Pelien kehitys on muuttunut laitteiden kehittymisen myötä, ja samalla isompien kehitystiimien kustannuksetkin ovat nousseet. Mobiilipelien tulo 2000-luvulla on kuitenkin muuttanut alaa ja pienten kehitystiimien pikkupeleillä on jälleen kysyntää.

Tutkimuksen motiivina on tutustua kirjallisuuskatsauksen avulla yleisesti pelien kehittämiseen ja löytää sopivia kehitystyökaluja, joilla videopelien teko onnistuu mahdollisimman alustariippumattomasti. Sen lisäksi motiivina on myös henkilökohtainen tiedontarve löytää omiin tarpeisiin sopivin mahdollinen kehitystyökalu 2D-videopelien tekoon mobiililaitteille.

Alan nopean kehityksen vuoksi aikaisempaa tutkimusta, jossa monialustapelimoottoreita vertaillaan, ei oikeastaan löydy. Jo pari vuotta vanhoissa dokumenteissa mainitaan monia kuolevia alustoja kuten Symbian OS, Palm OS, Blackberry OS ja Flash. Tämän päivän mobiilialustoja ovat kuitenkin käytännössä iOS, Android ja Windows Phone. Myös pelimoottorit on sivuutettu akateemisissa tutkimuksissa, vaikka nykyisin erityisesti Unitya käytetään laajalti suomalaisessa yliopistomaailmassa ja suomalaisessa peliteollisuudessa.

Tutkimusongelmana tässä Pro Gradu tutkielmassa on selvittää miten peliteollisuus on historiansa aikana muuttunut ja millaisia kehitysvälineitä videopelien tekemiseen mobiililaitteille on olemassa. Lisäksi ongelmana on yleiskäyttöisten kehitysvälineiden vertailu, jossa selvitetään niiden ominaisuuksia, ja miten käyttöjärjestelmien ja laitteiden eroja on kehitysvaiheessa otettava huomioon.

Tutkielman toisessa luvussa käsitellään videopelien historiaa ja käydään läpi alan kehittymistä. Lisäksi kuvataan kuinka pelien kehittäminen itsessään on muuttunut alkuajoista tähän päivään, sekä minkälaisia mahdollisuuksia pelien kehittäjät ovat tarjonneet pelaajille toimia itse pelisuunnittelijoina.

Kolmannessa luvussa perehdytään pelien kehitykseen ja yleiskäyttöisiin pelinkehitysvälineisiin. Tässä niitä ovat luokkakirjastot ja pelimoottorit, jotka mahdollistavat monialustakehityksen mobiililaitteille ja PC:lle.

Neljännessä luvussa esitellään peli, josta tehdään prototyyppi Cocos2d-x- ja Unity-pelimoottoreilla. Lisäksi esitellään osat, joista peli koostuu ja käydään läpi suunnitelma vaiheista, joissa kehitys toteutetaan. Lopuksi esitellään millaista tietoa eri vaiheista kerätään myöhempää arviointia varten.

Viidennessä luvussa käydään läpi pelin toteutus vaiheittain valituilla pelimoottoreilla. Toteutuksen vaiheita arvioidaan pelimoottoreiden käytettävyyden ja niiden välisten erojen kannalta. Lisäksi vaiheita havainnollistetaan lähdekoodeilla ja kuvankaappauksilla.

Kuudennessa luvussa esitellään lopputulokset ja pohditaan tarkemmin pelimoottoreiden soveltuvuutta useammalle alustalle tapahtuvaan pelinkehitykseen.

Seitsemäs luku on tutkielman yhteenveto.

2 VIDEOPELIEN HISTORIA

Videopeleillä tarkoitetaan elektronisia pelejä, joiden pelaamisessa käyttöliittymänä ovat näyttö ja ohjaimet. Ohjainlaitteina voivat olla mm. joystick, padi, ratti ja polkimet, lento-ohjaimet, näppäimistö, kosketusnäyttö, liiketunnistin jne. Videopelit voidaan jakaa tietokone-, konsoli-, arcade-, käsikonsoli- ja mobiilipelehin.

Termi, videopeli, vakiintui käytössä 1970-luvun lopulla. Sitä ennen käytettiin useimmiten termiä TV-peli, joka kuitenkin jäi pois käytöstä 80-luvun alussa. Toisaalta termiä tietokonepeli käytettiin 70-luvulla, mutta se oli vielä silloin hieman harhaanjohtava koska mikroprosessoreita alettiin yleisesti käyttää videopeleissä vasta 70-luvun lopulla (Donovan, 2010, 21). Erityisesti 80-luvulla käyttäjien keskuudessa puhuttiin tietokone- ja videopeleistä, vaikka todellisuudessa tarkoitettiin tietokone- ja konsolipelejä, jotka ovat kumpikin videopelien alalajeja. Tätä jakoa artikkeleissa näkee kuitenkin vielä tänäkin päivänä. Suomalaisessa yliopistomaailmassa käytetään myös termiä digitaalinen peli.

Pelejä pidettiin vielä 80-luvulla nuorten poikien harrastuksena, mutta 90-luvun alussa PC ja erityisesti Sonyn Playstation tekivät pelaamisesta yleisesti hyväksytyn ja suositun harrastuksen.

Tässä luvussa käydään läpi videopelien historian tärkeimmät tapahtumat kronologisessa järjestyksessä 1940-luvun lopulta 2010-luvun alkuun eurooppalaisesta ja pohjois-amerikkalaisesta näkökulmasta.

2.1 Videopelien aamunkoitto

Videopelien historia alkaa vuodesta **1947**, jolloin Thomas T. Goldsmith Jr. ja Estle Ray Mann rakensivat yksinkertaisen elektronisen pelin, johon he olivat saaneet innoituksen toisen maailmansodan tutkista. He liittivät katodisädeputken oskilloskooppiin ja ohjaimena toimivaan säätönuppiin. Säätimellä voitiin muuttaa oskilloskoopin näytöllä olevan valopisteen liikerataa. Pelissä ammuttiin ohjuksia eri kohteisiin, ja mikäli valopiste osui ennalta määrättyihin koor-

dinaatteihin, kohde tuhoutui räjähtämällä oskilloskoopin näytöltä. Pelin nimi oli **Cathode-Ray Tube Amusement Device**, ja se on ensimmäinen elektroninen peli. Se on myös ensimmäinen elektroninen peli, jolle on myönnetty patentti. Siitä tehtiin kuitenkin ainoastaan prototyyppi, eikä peliä koskaan tuotu markkinoille kalliin hinnan vuoksi. (Cohen, 2014).

Samana vuonna englantilainen matemaatikko Alan Turing sai ajatuksen tietokoneella pelattavasta **shakkipelistä**. Hänen kirjoittamansa koodi olisi kuitenkin vaatinut sen aikaisia kehittyneemmän tietokoneen, eikä monimutkaista peliä voitu ohjelmoida käytännössä. Turing testasi peli-ideaa kollegansa kanssa vuonna 1952, tekeytyen itse shakkikoneeksi ja tulkiten koodia pelin edetessä. Lopulta tunteja kestäneen pelin kuitenkin voitti ihminen. (Donovan, 2010, 4.)

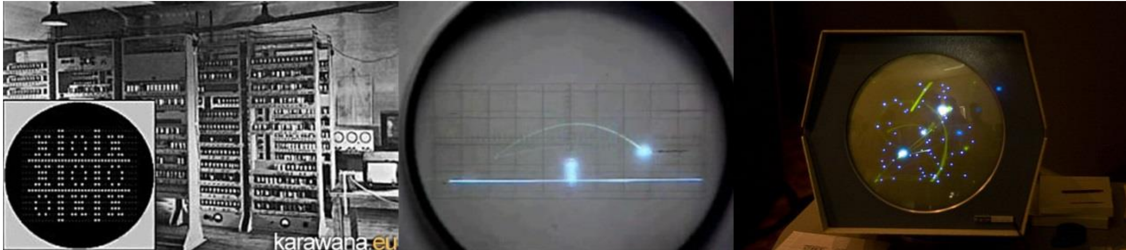
Brittiläinen tietokoneyritys Ferranti otti vuonna **1951** osaa kansallisen Festival of Britain -tapahtuman tiedenäyttelyyn South Kensingtonissa, Lontoossa. Näyttelyn lähestyessä Ferrantilta ei kuitenkaan löytynyt mitään näytteille asetettavaa, jolloin firman työntekijä John Bennett ehdotti, että rakennetaan tietokone, jolla voi pelata Nimiä, yksinkertaista matemaattista strategiapeliä. Idean **Nim**-peliin Bennett oli saanut sähkömekaanisesta koneesta nimeltään Nimatron, joka esiteltiin New Yorkin maailmannäyttelyssä 1940. Laitteen tarkoitus oli esitellä tietokoneen laskentakykyä, eikä suinkaan viihdyttää suurta yleisöä, jota ei kiinnostanutkaan matematiikka tai tiede taustalla vaan he halusivat ainoastaan pelata. (Donovan, 2010, 5.).

Englantilainen Alexander S. Douglas kirjoitti oman versionsa ristinollapelistä vuonna **1952** osana Cambridgen yliopistolla tekemäänsä tohtorinväitöskirjaa ihmisen ja tietokoneen vuorovaikutuksesta. Tämä EDSAC-tietokoneelle tehty peli sai nimekseen **OXO** (kuvio 1) ja se on vanhin tunnettu peli, joka käyttää näyttöä (35x16 pikseliä) pelin esittämiseen. (Donovan, 2010, 6.).

Vuonna **1958** amerikkalainen William A. Higinbotham päätti tehdä jotain hauskaa New Yorkin Long Islandilla vuosittain järjestettäville Brookhaven National Laboratoryn avoimien ovien päiville, jossa yleisölle esitellään laboratoriossa tehtyä työtä. Hän päätyi ideaan tenniksestä, jota pelattaisiin oskilloskoopin ruudulla. Brookhavenin insinööri Robert Dvorakin kanssa he rakensivat peliohjaimet, joilla liikuteltiin mailoina toimivia viivoja ja lyötiin palloa nappia painamalla pelikentällä, jonka keskellä oli verkko. **Tennis for Two** (kuvio 1) oli suosittu kävijöiden keskuudessa, mutta edelleenkin pelejä pidettiin lähinnä ajanhukkana, eikä niitä nähty muuta kuin hetken hupina. Tennis for Twon laitteisto purettiin myöhemmin muita projekteja varten. (Donovan, 2010, 8-9.).

Keväällä 1959 Massachusettsin teknillisessä korkeakoulussa MIT:ssä tarjottiin ensimmäinen ohjelmoinnin kurssi. Samalla osa korkeakoulun opiskelijoista koostuvan pienoisrautatiekerhon jäsenistä innostui tietokoneista ja niiden ohjelmoinnista, ja osallistuivat kyseiselle kurssille. Digital Equipment Corporation lahjoitti MIT:lle PDP-1 -tietokoneen loppuvuonna **1961**. Sen seurauksena kerhon jäsenet Steven Russell, Martin Graetz ja Wayne Wiitanen intoutuivat miettimään, mitä sillä voisi tehdä, ja päätyivät tekemään pelin. Avaruustaistelu kuulosti mahtavalta idealta, ja tuloksena oli kaksinpelattava kaksintaistelupeli **Spacewar!** (kuvio 1), jossa oli aidosti mallinnetut fysiikat ja jossa keskellä ruu-

tua oli tähti, jonka painovoima veti aluksia puoleensa. Peli esiteltiin toukokuussa 1962 vuosittaisessa Science Open House -tapahtumassa. Tekijät päätyivät jakamaan peliä ilmaiseksi kaikille halukkaille PDP-1-tietokoneiden käyttäjille, koska sille ei uskottu löytyvän markkinoita kalliin 120 000 dollaria maksavan tietokoneen omistajista. (Levy, 2010, 3-59.).



KUVIO 1 OXO, Tennis for Two ja Spacewar! (Anonimowy, 2010, Anonimowy, 2010 ja Lasar, 2011)

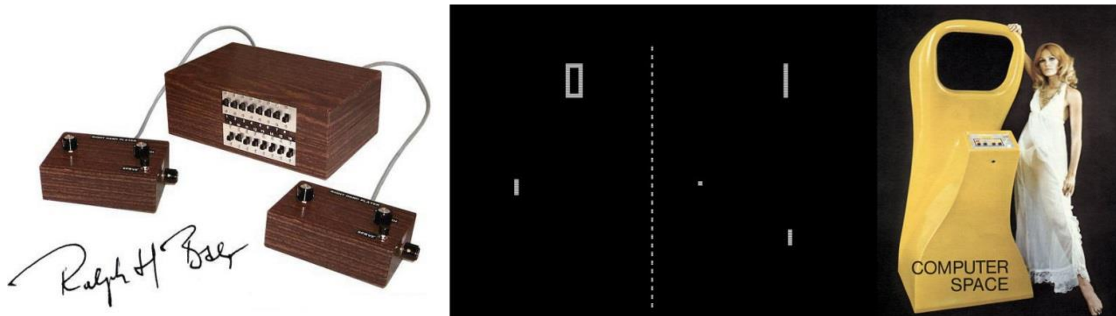
Elokuussa 1966 Ralph Baer oli työmatkalla New York Cityssä ja sai linja-autoasemalla paluukyytiä odotellessaan ajatuksen peleistä. Seuraavana aamuna hän oli kirjoittanut nelisivuisen ehdotuksen halvasta pelikoneesta, jonka saisi liitettyä televisioon. Baer toimi osaston johtajana elektroniikkayritys Sanders Associatesilla, joka toimi alihankkijana sotateollisuudelle. Baer ei halunnut herättää huomiota hankkeellaan ja alkoi käyttää siitä nimeä Channel LP (Let’s Play), joka kuulosti sotilassanastolta. Yhdessä teknikko Bill Harrisonin ja pääinsinööri Bill Ruschin kanssa Baer sai maaliskuussa **1967** valmiiksi toimivan laitteen ja muutamia peli-ideoita, mm. Ping-Pongin, jossa pelaajat pomputtelisivat mailoillaan palloa, ja ammuntapelin, jossa ammuttiin muovisella kiväärillä ruudulla näkyviä kohteita. Laitetta esiteltiin yrityksen johdolle nimellä **Brown Box** (kuvio 2), mutta he eivät osoittaneet sitä kohtaan minkäänlaista kiinnostusta. Loppuvuodesta 1967 TelePrompter niminen yritys oli kiinnostunut aloittamaan laitteen valmistuksen, mutta ajautui pari kuukautta myöhemmin niin pahoihin taloudellisiin vaikeuksiin, että heidän oli luovuttava suunnitelmistaan. (Donovan, 2010, 11-13.). Tammikuussa 1971 tv-valmistaja Magnavox kiinnostui Brown Boxista ja tehtiin alustava sopimus, jonka perusteella siitä alettiin tehdä oikeaa tuotetta. Elokuussa **1972** maailman ensimmäinen pelikonsoli **Magnavox Odyssey** tuli myyntiin Magnavoxin omien jälleenmyyjien kautta, mutta jäi luultavasti ylihinnoittelunsa vuoksi myöhempien kilpailijoidensa jalkoihin. Suunnitteluvaiheessa myyntihinnaksi kaavailtiin 19,95 dollaria, mutta lopulliseksi hinnaksi kuitenkin asetettiin 99,95 dollaria. (Donovan, 2010, 21-22.).

Bill Pitts ja Hugh Tuck innostuivat opiskeluaikoinaan Spacewar!ista ja rakensivat vuonna **1971** ensimmäisen kolikkopelin **Galaxy Gamen**, joka oli kopio Spacewar!-pelistä. Kaikki peliä pelanneet olivat siitä innoissaan, mutta 65 000 dollarin kehityskustannukset olivat kuitenkin aivan liian korkeat, eikä kolikkopeliä ollut mahdollista saattaa sarjatuotantoon. (Donovan, 2010, 15-19.).

Myös Nolan Bushnell innostui samoihin aikoihin kuin Pitts ja Tuck Spacewar!ista ja lyöttäytyi yhteen Ted Dabneyn kanssa aikomuksenaan tehdä avaruustaistelusta kolikkopeli. Toisin kuin Galaxy Game, Bushnellin tavoitteena oli

pitää kustannukset pieninä suoran kopion kustannuksella. Pari kuukautta Galaxy Gamen julkaisun jälkeen marraskuussa **1971** Bushnell vei **Computer Spacen** (kuvio 2) Stanfordin yliopiston kampuksen lähellä olevaan Dutch Goose -baariin testikäyttöön. Se oli välittömästi menestys baarin kävijöiden keskuudessa, ja Nutting Associates tekikin samantien peliä 1 500 kappaletta. Muualla ihmiset eivät pelanneetkaan sitä, vaan peliä pidettiin liian monimutkaisena. Kukaan ei ollut ottanut huomioon, että Dutch Goosessa asiakaskunta koostui lähes ainoastaan opiskelijoista, kun muissa baareissa kävijät olivat työläisiä, jotka tulivat työpäivän jälkeen kaljalle, eivätkä innostuneet niin monimutkaisista peleistä. (Donovan, 2010, 17-21.).

Computer Space menestyi sen verran hyvin, että vuonna **1972** Bushnell ja Dabney perustivat Syzygy Engineeringin. Nimi oli kuitenkin varattu, joten se muutettiin Go-pelistä tuttuun termiin **Atari**. Samana päivänä Al Alcorn palkattiin taloon, ja hänen ensimmäisenä työnään oli tutustua Bushnellin kasaamaan laitteistoon ja tehdä harjoituksena Ping-Pong-klooni. Siitä tuli kuitenkin niin hauska ja koukuttava peli, että Bushnell päätti kokeilla, kiinnostuisivatko myös muut pelistä. (Donovan, 2010, 23-24.). Pelin nimi muutettiin muotoon **Pong** (kuvio 2), ja kolikkopeli vietiin Andy Capp's Tavernvaan Sunnyvaleen, Kaliforniaan. Legendan mukaan sieltä kuitenkin soitettiin pian Atarille ja kerrottiin koneen rikkoutuneen. Alcorn saapui paikalle tutkimaan ongelmaa, ja hänen avatessaan koneen kolikkolaatikon kolikot lensivät pitkin kapakan lattiaa. Peli oli ollut niin suosittu, että kolikkolaatikko oli täyttynyt ja aiheuttanut pelin jumiutumisen. Atari huomasi käsissään olevan menestyksen avaimet. (Goldberg & Vendel, 2012, 72-74.).



KUVIO 2 Brown Box, Pong ja Computer Space (The Game Console, 2014, Wikipedia, 2014i ja CED Magic, 2014)

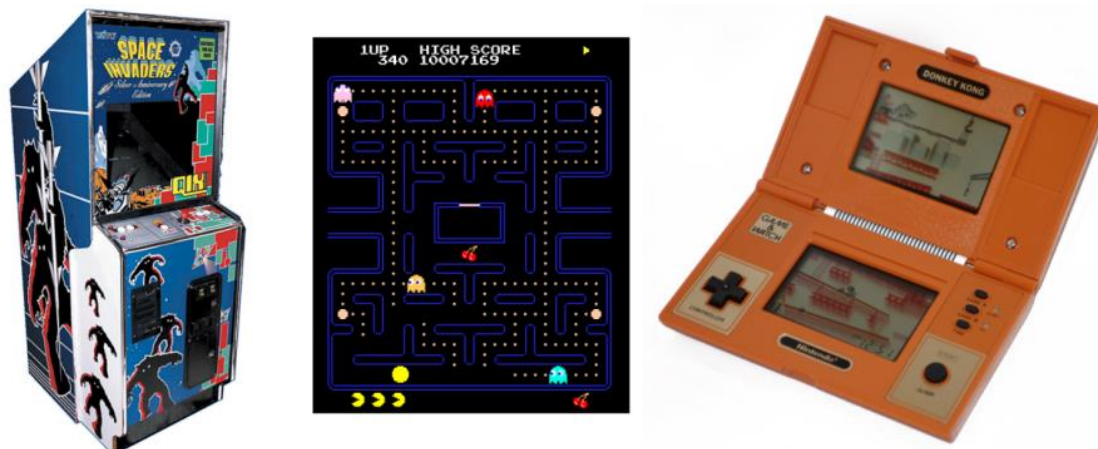
2.2 Kulta-aika

Videopelien kulta-aikana pidetään ajanjaksoa joka alkoi 1970-luvun lopulla ja päättyi 1980-luvun puolivälissä. Se oli suurten teknisten harppausten ja luovuuden aikaa, jolloin kolikkopelit levisivät ympäri maailman. 1970-luvun lopulla perustettiin ensimmäiset arcade-hallit, joissa pääsi pelaamaan useita kolikkopelejä ja tapaamaan kavereita. Suomessa kolikkopelejä löytyi mm. baareista, keilahalleista, huoltoasemilta ja huvipuistoista.

Heinäkuussa **1978** julkaistiin Japanissa ensimmäinen todellinen kolikkopelihitti Taiton **Space Invaders** (kuvio 3). Se myi maailmanlaajuisesti yli 160 000 kappaletta, ja toi videopeleihin monia asioita ensimmäistä kertaa, kuten animoidut hahmot ja high score –listan. (Eddy, 2012, 13.). Pelin suosion myötä Japanin rahaliikenteessä näkyi puute 100 jenin kolikoista, koska huomattava osa niistä päätyi kolikkopeleihin. Se pakottikin Japanin lyömään kolikoita lisää markkinoille. (Mott, 2010, 26). Space Invadersia seurasi lukuisia kopioita, mutta myös hieman erilaisia räiskintäpelejä julkaistiin, kuten **1979** Atarin **Asteroids** ja Namcon **Galaxian**. (Eddy, 2012, 14).

Kaikkien aikojen menestynein kolikkopeli Namcon **Pac-Man** (kuvio 3) julkaistiin **1980**, myyden yli 350 000 kappaletta. Se oli ensimmäinen peli, joka vetosi kumpaankin sukupuoleen, tuoden peleille kokonaan uuden kohderyhmän. Pac-Man on luultavasti suurin syy, miksi videopelikulttuurista muodostui ilmiö 1980-luvun alussa. Pac-Man näkyi kaikkialla. Se oli vaatteissa ja leluissa, ja siitä kirjoitettiin lukuisia kappaleita, joista kuuluisin on Bucknerin ja Garcian Pac-Man Fever. Se on myös tunnetuin pelihahmo videopelien historiassa, jonka tunnistaa 94 %:ia amerikkalaisista kuluttajista. (Eddy, 2012, 21-22.).

Kun Nintendon leluosaston pomo Gunpei Yokoi näki miehen, joka kulutti aikaa leikkimällä taskulaskimellaan jossa oli LCD näyttö, Yokoi sai ajatuksen kannettavasta videopelistä ja tuloksena oli **Game & Watch** (kuvio 3) pelien sarja. Ensimmäinen niistä julkaistiin huhtikuussa 1980 ja siitä tuli välitön menestys. Seuraavan 11 vuoden aikana Nintendo julkaisi kymmeniä Game & Watch pelejä, joita myytiin yhteensä yli 30 miljoonaa kappaletta maailmanlaajuisesti. Suomessa pelit tunnettiin paremmin elektroniikkapeleinä. (Donovan, 2010, 154-155.).



KUVIO 3 Space Invaders, Pac-Man, Donkey Kong Game & Watch (Firebox, 2014, Namco, 1980 ja Wikipedia, 2014g)

1981 videopeliteollisuuden liikevaihto Yhdysvalloissa oli jo 5 miljardia dollaria. Samana vuonna Nintendo iski kultasuoneen Shigeru Miyamoton hitillä **Donkey Kong**, jonka hahmo Jumpman sai myöhemmin vuonna **1983** oman pelin **Mario Brothers**. (Eddy, 2012, 27-28).

Vuosi 1982 oli kulta-ajan huippu. Silloin pelejä tuotettiin eniten, ne myivät eniten ja yleisön kiinnostus oli huipussaan. Videopelit pääsivät jopa Time-lehden kansitarinaksi tammikuun numeroon. (Eddy, 2012, 35.). Arcade-halleja perustettiin joka puolelle, ja mm. Nevadassa kasinot tyhjensivät perinteisiä uhkapelejään kolikkopelien tieltä. (Kent, 2001, 167).

2.3 Suuri videopeliromahdus Pohjois-Amerikassa 1983

Yhdysvaltoja koetteli taantuma **1982**, ja jouluun mennessä työttömyys oli noussut jo 10 %:iin. Samana vuonna Commodore toi markkinoille **Commodore 64** –kotimikron (kuvio 4), jota se myi halvalla massoille. Englannissa samaan aikaan Sinclair toi edullisen **ZX Spectrum**in (kuvio 4) myyntiin.

KUVIO 4 Commodore 64 ja ZX Spectrum (Wikipedia, 2014c ja Wikipedia, 2014k)

Arcade-halleja syntyi kuin sieniä sateella ja pienissä kaupungeissakin niitä saattoi olla useita. Kilpailu asiakkaista oli kovaa, joten kukaan ei enää ansainnut arcade-halleilla, vaan konkurssiaalto pyyhkäisi koko USA:n läpi. Pelaajat hioivat taitojaan ja kehittyivät aina vain paremmiksi. Neljännesdollarilla pystyi pelaamaan entistä pidempään, ja pelien kehittäjien oli pakko reagoida tekemällä peleistä yhä vaikeampia, jotta pelit eivät kestäisi liian pitkään vaan rahavirta pysyisi tasaisena. Koko 80-luvun kolikkopelien vaikeustaso pysyi varsinkin tietyissä peligenreissä jopa epäinhimillisen korkeana. Seurauksena massat hylkäsivät kolikkopelit ja siirtyivät konsoleiden pariin. (Donovan, 2010, 97.).

Activision jalanjäljissä syntyi uusia pelitaloja, jotka tekivät pelejä Atari VCS:lle (kuvio 5). Tekijöillä ei kuitenkaan välttämättä ollut aiempaa kokemusta tai ylipäätään käsitystä peleistä saati niiden tekemisestä, joten pelien heikko taso ja ylituotanto jättivät pelejä kauppojen hyllyille, ja romahdus oli edessä myös konsoleissa. (Donovan, 2010, 98.). Kauppojen varastot olivat täynnä pelejä, jotka eivät menneet kaupaksi, joten kaupat alkoivat tyhjentää varastojaan alehintaan eivätkä enää ostaneet uusia pelejä tilalle. Activisionkin kuvitteli heikompien pelitalojen kaatuvan hetkessä ja laadun taas ratkaisevan, mutta he eivät olleet ottaneet huomioon, että pelitalot olivat jo teettäneet peleistään miljoonan pelikasetin varastoja kuvitellen kaiken menevän kaupaksi. Tästä seurasi pelien myyminen kaupoille lähes omakustannehintaan. Pelitalolle kasetin (cart-

ridge) valmistus maksoi 3 dollaria, ja se myytiin kaupoille 4 dollarilla, jotka taas kauppasivat kuluttajille 5 dollarilla. Activisionin täysihintaiset pelit maksoivat 40 dollaria, joten täysihintaiset laatupelit eivät yllättäen enää menneetkään kaupaksi. (Donovan, 2010, 99.). Activision ja Atari joutuivat leikkaamaan omien peliensä hintoja, etteivät ne jääneet varastoihin pölyttymään, mutta samalla kadotettiin pelien katteet.

Steven Spielbergin elokuvaan **E.T.** pohjautuva peli (kuvio 5) kuvastaa hyvin miten vakavaan tilanteeseen oli ajauduttu. Tätä pidetään yleisesti videopelihistorian huonoimpana sopimuksena, ja lopulta se sinetöi Atarin kohtalon. Atari teki lisenssistä sopimuksen heinäkuussa 1982, ja maksoi pelistä lisenssimaksuja 25 miljoonaa dollaria. Atarin Ray Kassar kiinnitti projektiin Howard Scott Warshawn, joka oli nuori ohjelmoija ja käski hänen tehdä pelin valmiiksi syyskuun alkuun mennessä, jotta peli ehtisi kauppoihin ennen joulua. Kiireessä tehty peli oli buginen, eikä siinä ollut oikeastaan mitään pelattavaa. Peliä päätettiin kuitenkin valmistaa 5 miljoonaa kappaletta, joista peli myi ainoastaan E.T.:n nimellä 1,5 miljoonaa kappaletta. Myymättä jääneet kasetit haudattiin New Mexicon aavikolle. (Dillon, 2011, 73-74.).

KUVIO 5 Atari VCS ja E.T. (Wikipedia, 2014a ja Wikipedia, 2014f)

2.4 Kenttäeditorit

Ohjelmointiharrastus kotimikrojen aikana oli suhteellisesti yleisempää kuin tänä päivänä. Siitä huolimatta pelien tekeminen oli vaikeaa, eikä suurinta osaa harrastajista ohjelmointi kiinnostanutkaan. Useat kehittäjät kuitenkin halusivat antaa pelaajille mahdollisuuden käyttää luovuuttaan ja toimia pelisuunnittelijoina. Pelien sisään rakennettiin editoreita, joilla pystyi muokkaamaan pelien kenttiä (level) ja pelaamaan niitä. Samoihin aikoihin tuli myös ohjelmia, joilla pystyi suunnittelemaan ja rakentamaan yksinkertaisia, yleensä tietyn genren pelejä alusta loppuun, tallentamaan tuloksen levylle tai kasetille ja jakamaan kavereilleen.

Doug Smithin **Lode Runner** vuodelta **1983** oli ensimmäinen tasohyppely, jossa oli sisäänrakennettu kenttäeditori (Donovan, 2010, 142). Julkaisija **Brøderbund** päätti jättää editorin myytävään versioon, ja sen seurauksena pelaajat lä-

hettivät lukuisia omatekemiään kenttiä postitse takaisin julkaisijalle. Kenttiä tuli niin paljon, että niistä vaikeimmat päätyivät jatko-osaan Championship Lode Runneriin. Siitä kenttäeditori kuitenkin jätettiin pois. (Grannel, 2013, 24).

Samana vuonna Bill Budgen **Pinball Construction Set** (kuvio 6) mahdollisti flipperipöytien suunnittelun ja jopa pelifysiikalla leikkimisen. Kentät pystyttiin tallentamaan levylle myöhempää pelaamista varten. (Grannel, 2013, 23). Nämä kaksi peliä ovatkin edelläkävijöitä uudessa tavassa, jossa pelaajat voivat itse luoda sisältöä peliin.

Vuosien varrella useat pelit ovat antaneet pelaajille mahdollisuuden toimia pelisuunnittelijoina. Mattel Intellivisionin autopeli **Rally Speedwayn** (Wikipedia, 2014j) paranneltu versio C64:lle mahdollisti ratojen tekemisen ja niiden kaksinpelaamisen **1984**. Motocrossratojen tekeminen onnistui Nintendon **Excitebikessa** vuonna **1984** ja C64:llä Shaun Southernin **Kickstart II – The Construction Setissä 1987**. Nintendolla tosin ratoja ei voinut tallentaa mihinkään ilman lisälaitteita ja virtojen katkaisu hävitti vaivalla tehdyn työn. Stuart Smithin **Adventure Construction Setillä** sai **1984** C64:lle luotua seikkailupelejä tiiligrafiikalla. Sensible Softwaren **Shoot 'em Up Construction Kitillä 1987** (kuvio 6) puolestaan pystyi tekemään räiskintäpelejä C64:lle.

2000-luvulla merkittäviä pelejä, joissa on helppokäyttöiset työkalut kenttien suunnitteluun ovat mm. Bungien **2007** Xbox 360:lle tekemä **Halo 3**, Media Moleculen **LittleBigPlanet** Playstation 3:lle **2008** ja kaikkien aikojen suosituin kenttäeditori, Markos Perssonin PC:lle kehittämä **Minecraft** vuodelta **2009**. (Grannel, 2013, 23.). Syksyllä **2014** Microsoft julkaisi PC:lle ja Xbox Onelle pelintekopeli **Project Sparksin**.

KUVIO 6 Pinball Construction Set ja Shoot 'em Up Construction Kit (BudgeCo, 1983 ja Sensible Software, 1987)

2.5 Piratismi

Pelien siirryttyä kotimikroille ja niiden tallennusmedian vaihtuminen pelikoneisiin kovakoodaamisen sijasta kaseteille ja levykkeille, synnytti samalla myös piratismin. Pelien kopioiminen kävi helpoimmillaan kaksipesäisellä kasettinauhurilla ja tyhjien kasettien ja levykkeiden halpa hinta verrattuna alkuperäisiin

peleihin edesauttoi piratismin leviämistä. Pelejä vaihdeltiin 80-luvulla mm. koulujen pihoilla ja vain harva edes omisti useampaa alkuperäistä peliä. Yksiä ensimmäisiä taltioituja tapauksia piratismista on 70-luvun puolivälistä kun Bill Gates huomasi MITS Altair koneen Micro-Soft BASICin kopioita liikkuvan siellä täällä. (Railton, 2005, 180-181.).

Peliteollisuus reagoi piratismin riehumiseen ja alkoi lisätä peleihin kopiosuojauksia. Ensimmäisenä tulivat salasanat, joita kysyttiin pelin latauduttua, ja joihin piti löytää vastaus pelin manuaalista. Se oli käytännössä melko turhaa, koska myös manuaalit pystyttiin kopioimaan kopiokoneilla. Tallennusmedian muuttuessa kaseteista levyihin, alkuperäiset pelit tallennettiin levylle yhdellä ylimääräisellä raidalla. Sen piti toimia kopiosuojauksena koska levyn kopiointi keskeytyi siihen, mutta piraatit saivat tämänkin nopeasti selville. Peli kopioitui normaalisti jättämällä viimeisen raidan kopioimatta. Peliteollisuus kehitti uudenlaisia kopiosuojauksia ja samaan tahtiin piraatit mursivat eli kräkkäsivät ja poistivat ne. Piraatit keksivät myös aloittaa pelien pakkaamisen, jolloin yhdelle levykkeelle tai kasetille mahtui useita pelejä. Tämän lisäksi pakattuihin peleihin lisättiin tervehdys toisille crack-ryhmille jonkinlaisen intron muodossa (kuvio 7). (Railton, 2005, 182-183.).

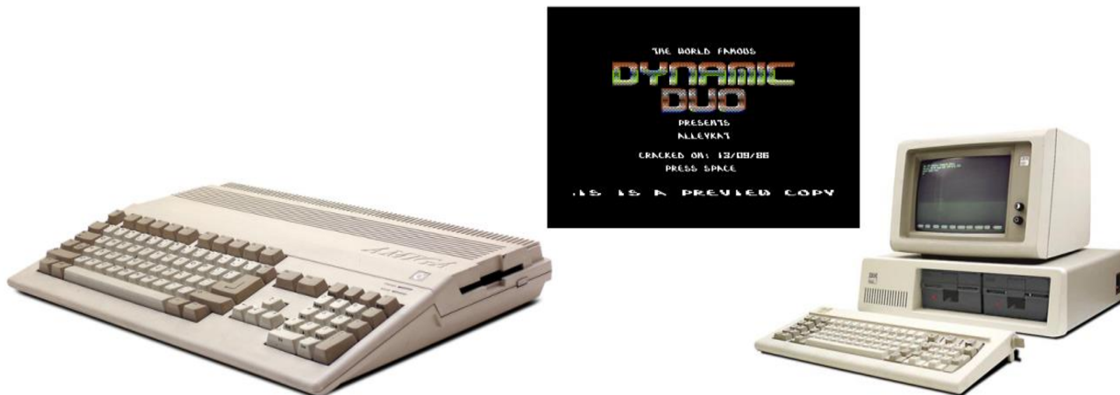
Introt eli demot ovat digitaalisen taiteen reaaliaikaisia esityksiä, joissa yhdistyvät grafiikka, musiikki ja algoritmit. Niillä haluttiin esitellä oman ryhmän ohjelmointitaitoa ja ajan myötä ne kehittyivät entistä näyttävämmiksi. Jossain vaiheessa piraatit ja introjen tekijät erkanivat omille teilleen ja demoskene oli syntynyt. Suomessa erityisen vahva demoskene on pohjana tämän päivän peliteollisuudelle, ja josta moni suomalainen pelitalo on ponnistanut, kuten 1990-luvulla Terramarque, Bloodhouse, Remedy ja Bugbear. (Kuorikoski, 2014, 36-37.).

16-bittiset laitteet olivat jo 80-luvun lopulla tulleet 8-bittisten tilalle. Konsoleilla ja PC:llä (kuvio 7) piratismi ei ollut paha ongelma, mutta 80-/90-luvun taitteen suosituin kotimikro Commodore Amiga (kuvio 7) kärsi siitä voimakkaasti. Pelit julkaistiin mieluummin PC:lle kuin Amigalle koska Amiga versiot eivät yksinkertaisesti käyneet kaupaksi. Euroopan pahimmissa piraattipesäkkeissä Italiassa ja Espanjassa jopa lopetettiin Amiga-pelien maahantuonti, koska piratismi oli tappanut liiketoiminnan kannattamattomana. Näissä kahdessa maassa pelit olivat jo pari viikkoa aikaisemmin piraattien levityksessä ennen kuin maahantuojat saivat niitä lopulta varastoihinsa (Pelikirja Special Operations Group, 1991, 24). Lopulta Amigankin pelien tekeminen lopetettiin ja voimavarat siirrettiin kehitykseen uusille alustoille. Kotimikrojen aika oli päättynyt.

2.6 Ensimmäiset pelimoottorit ja pelien modaus

Vuonna **1977** tekstiseikkailupeli **Zorkiin** kehitettiin Z-machine niminen virtuaalikone, johon rakennettiin peli käyttämällä omaa ZIL-kieltä eli Zork Implementation Languagea. Kun peli julkaistiin uusille alustoille, piti sitä varten tehdä

ainoastaan tulkkiohjelma kyseiselle laitteelle. (Bevan, 2012). Käsitettä pelimoottori ei kuitenkaan tunnettu vielä edes 1980-luvulla, mutta mm. Sierra Online ja Lucasfilm Games, sittemmin LucarArts, kehittivät kumpikin seikkailupeleihinsä talon sisällä käytettäviä ohjelmistoja, jotka toimivat kuten pelimoottorit. Kummallakin ne olivat moottoreita, jotka tulkkasivat yritysten omilla skriptikielillä kirjoitettua koodia. Sierran ensimmäinen moottori AGI eli Adventure Game Interpreter tulkkasi omaa proseduraalista skriptikieltä, ensimmäisenä pelissä **King's Quest** vuonna **1984**. (ScummVM, 2006). Tekniikan kehittyessä ja uusien 16-bittisten koneiden tullessa kehitettiin uusi tulkki, **SCI** eli Sierra's Creative Interpreter, jolla kyettiin jo tulkkaamaan oliopohjaista koodia. Ensimmäinen peli, jossa SCI korvasi **AGIn** oli **King's Quest IV** vuonna **1988**. (ScummVM, 2009). LucasArtsin SCUMM eli Script Creation Utility for Maniac Mansion kehitettiin vuonna **1987** nimensä mukaisesti peliin **Maniac Mansion** (kuvio 8). (ScummVM, 2001). Uusien tietokoneiden myötä myös pelimoottoreita kehitettiin eteenpäin ja muokattiin toimimaan usealla eri laitteistolla.



KUVIO 7 Amiga, Intro, IBM PC (Obsolete Technology, 2014, Dynamic Duo, 1986 ja Obsolete Technology, 2014)

Yleisesti ensimmäisenä FPS (First Person Shooter) eli ensimmäisen persoonan ammuntapelinä, jossa pelimaailma esitetään pelihahmon näkökulmasta, pidetään **id Softwaren Wolfenstein 3Dtä** vuodelta **1992**. (Gregory, 2009, 25). Samaisen id Softwaren seuraavan pelin yhteydessä termi pelimoottori tuli yleiseen tietoisuuteen, kun erittäin suosittu **Doom** (kuvio 8) julkaistiin vuonna **1993**. John Carmack kehitti peliä varten Doom Enginen, jonka arkkitehtuurissa moottorin eri osat olivat selkeästi rajattu tekemään ainoastaan omat tehtävänsä. Sen ydinohjelmistokomponentit, kuten 3D renderöinti, törmäysten tutkiminen ja äänijärjestelmä oli erotettu pelin taiteesta, pelimaailmasta ja pelin säännöistä. Arkkitehtuuriratkaisun arvo nähtiin nopeasti ja muut kehittäjät alkoivat luoda omia pelejään uusilla grafiikoilla ja pelimaailmoilla, lisensoimalla pelimoottoria ja sitä varten tehtyjä kehitystyökaluja. Koska pelimoottorin ytimeen ei tarvinnut koskea tai sitä jouduttiin muokkaamaan ainoastaan hieman, myös pelaajaryhmät ja pienet pelistudiot ryhtyivät muokkaamaan olemassa olevien pelien sisältöä ilmaisilla työkaluilla. Se merkitsi modauskulttuurin syntymistä, jossa kuka tahansa sai mahdollisuuden modata (modding) eli muokata (modify) FPS-

pelejä mieleisekseen. (Gregory, 2009, 11.). Luultavasti suosituin modi (mod) on Valve Softwaren Half-Lifen pohjalta tehty **Counter Strike** vuodelta **1999**.

KUVIO 8 Maniac Mansion ja Doom (Lucasfilm Games, 1987 ja id Software, 1993)

2.7 Mobiilipelit

Ensimmäinen mobiilipeli on Tetris, joka löytyy tanskalaisesta Hagenuk MT-2000 matkapuhelimesta vuodelta 1994. Nokia toi vuonna **1997** uuden puhelinmallinsa 6110:n markkinoille. Se oli ensimmäinen Nokia matkapuhelin, jossa oli pelejä ja niistä suosituimmaksi käyttäjien keskuudessa nousi Taneli Armannon kehittämä **Matopeli** (kuvio 9). Se on sittemmin päätynyt 350 miljoonaan laitteeseen ympäri maailman. Peli valittiin yhtenä neljästäkymmenestä pelistä New Yorkin modernintaiteen museoon marraskuussa 2012. (Kuorikoski, 2014, 42-43).

Nokia tavoitteli Nintendon Game Boy Advancen menestystä ja toi markkinoille ensimmäiset pelipuhelimet **N-Gagen 2003** ja puoli vuotta myöhemmin **2004 N-Gage QD:n**. Niistä tuli kuitenkin taloudelliset katastrofit Nokialle huonojen teknisten ratkaisujen ja heikon pelitarjonnan vuoksi. Lisäksi pelit olivat helposti kopioitavissa, ja Nintendon edullisempi käsikonsoli Game Boy Advance SP tuli samaan aikaan markkinoille kattavammalla pelitarjonnalla. (Kuorikoski, 2014, 70).

Koko matkapuhelinalaa ravisteltiin **2007**, kun Apple toi ensimmäisen **iPhone** (kuvio 9) älypuhelimensa markkinoille. Sen vaikutus oli niin valtava, että mm. entinen markkinajohtaja Nokia, joutui luopumaan matkapuhelintoiminnoistaan 7 vuotta myöhemmin keväällä 2014, ja myymään kannattamattoman puhelintoimintansa Microsoftille. iPhonen julkistustilaisuudessa San Fransiscossa tammikuussa 2007, Steve Jobs sanoi ”Silloin tällöin markkinoille tulee vallankumouksellinen tuote, joka muuttaa kaiken” (Isaacson, 2011, 504). Teknisesti laadukas kosketusnäytöllinen puhelin haastoi kerralla myös käsikonsolit pelialustana. Todellinen vallankumous tapahtui kuitenkin seuraavana vuonna kun Apple lanseerasi App Storen, mobiilisovellusten ja -pelien digitaalisen kauppapaikan. Pelien julkaisu ja niillä ansainta helpottui ja uusia mobiilipeleihin keskittyneitä pelifirmoja alkoi syntyä. App Storen jälkeen Google ja Microsoft julkistivat omille alustoilleen vastaavat kauppapaikat Google Playn, tun-

nettiin alun perin Android Markettina ja Microsoft Storen. Apple toi myös ensimmäisen tabletin, **iPadin**, markkinoille huhtikuussa **2010**.

Rovio julkaisi **Angry Birds** (kuvio 9) pelinsä **2009** iPhonelle ja sen menestyksen siivittämänä koko suomalainen peliala on kasvanut kiihtyvällä vauhdilla koko 2010-luvun alun. Suomalaisen pelialan liikevaihto vuonna 2004 oli 40 miljoonaa euroa (Hiltunen, Latva & Kaleva, 2014, 44). Vuonna 2012 se oli 250 miljoonaa, ja vuonna 2013 sen arvioidaan olleen jo 800 miljoonaa euroa. Maailmanlaajuisesti peliteollisuuden arvo vuonna 2014 lähenee 100 miljardia euroa. (Hiltunen ym, 2014, 6).

Suurten julkaisijoiden pitkään jatkunut varman päälle peluu vaikutti, paitsi pelinkehitystiimien kokoon ja kehitysbudjetteihin, myös uusien peli-ideoiden köyhtymiseen. Eräällä tavalla ympyrä on sulkeutunut, kun yksittäiset kehittäjät voivat jälleen kokeilla omaperäisiä ideoitaan, ja saada ne maailmanlaajuiseen jakeluun ilman suuria budjetteja. Tekninen ylivertaisuus ja graafinen loisto ei enää olekaan menestyksen tae, vaan koukuttava peli-idea ja uudenlaiset ansaintamallit.

KUVIO 9 Matopeli, Angry Birds ja iPhone (Taskumuro, 2010, Rovio, 2009 ja Amazon, 2014)

2.8 Yhteenveto

Luvussa käytiin läpi lyhyesti videopelien historian tärkeimmät tapahtumat pohjois-amerikkalaisesta ja eurooppalaisesta näkökulmasta. Hitaan alun jälkeen videopelit tulivat kaikkien tietoisuuteen 1970-luvulla, ja ovat nykyisin suositumpia kuin koskaan aiemmin. 1970-luvulta saakka jatkunut kehityskustannusten kallistuminen muuttui 2000-luvulla älypuhelinten tullessa markkinoille. Yksinkertaiseen ideaan perustuvat pikkupelit kelpailevat suosiosta suurten ja kalliiden AAA-luokan pelien kanssa.

Lisäksi luvussa painotettiin erityisesti harrastajapelinkehittäjän näkökulmaa. Sitä kuinka ohjelmointia osaamattomille pelaajille tarjottiin 1980-luvulla mahdollisuutta toimia pelisuunnittelijoina peleihin sisäänrakennettujen kenttäeditorien ja pelinteko-ohjelmien avulla. Myöhemmin 1990-luvulla pelien yhtey-

dessä julkaistujen pelimoottoreiden ja niitä varten tehtyjen kehitysvälineiden avulla pelien modaus synnytti kokonaan uusia pelejä.

3 VIDEOPELIEN KEHITYS

Videopelit ovat muuttuneet ajan myötä huomattavasti. Tekniikan kehittyminen on tehnyt peleistä monipuolisempia ja syventänyt pelikokemuksia. Samalla kehityskustannukset ja työmäärä ovat kuitenkin kasvaneet, ja isoimmissa julkaisuissa voi olla mukana satoja tekijöitä. Pelintekijöiden ei kuitenkaan nykyisin enää tarvitse rakentaa kaikkea alusta alkaen itse, eikä ottaa huomioon erilaisia laitteistokokoonpanoja, vaan pelinkehitykseen on kehitetty lukuisa määrä apuvälineitä. Tässä luvussa käydään läpi yleisesti pelinteon työkaluja.

> Pelinkehittäjälle siistein juttu on se taianomainen hetki, kun kasa pikseleitä herää eloon, ja muuttuu peliksi ensimmäistä kertaa. –Eugene Jarvis, Robotron: 2084:n tekijä. (Retro Gamer, 2012, 26).

3.1 Kehityksen evoluutio

Ensimmäisessä kaupallisessa videopelissä, Computer Spacessa, vuodelta 1971 ei ollut prosessoria tai RAM-muistia vaan siinä käytettiin transistoreja tilakoneina ja pelin logiikka toteutettiin kokonaan laitteistotasolla. (Madhav, 2013, 2.).

1970-luvun alkupuolen laitteisto ja ohjelma suunniteltiin yhdelle tai riittävän samanlaisille peleille, koska pelit eivät olleet vaihdettavissa. Pong konsoli julkaistiin 1975, kun mikrosirut halpenivat ja kotikonsoli oli kannattavaa tehdä. (Donovan, 2012, 36-37.).

Intel julkaisi 1975 mikroprosessorin 8080, jossa riitti tehoja näyttää kuvaa tv:ssä, toisin kuin sen edeltäjässä 4040:ssä. Se tiputti tulevien pelilaitteiden hintoja huomattavasti, koska mikroprosessorille voitiin antaa ohjelmallisesti tehtäväksi asioita, jotka aiemmin oli jouduttu tekemään laitteistolla. (Donovan, 2012, 40-41.).

Atari toi vuonna 1977 Atari VCS-konsolin markkinoille, jossa oli vaihdettavat pelikasetit (cartridge), ja yhtenäinen alusta johon pelit kehitettiin ohjelmoimalla. 1970-luvun puolivälin jälkeen myös muilta laitevalmistajilta tuli pelikonsoleita, joissa käytettiin vaihdettavia kasetteja. Laitteiden tehot olivat kui-

tenkin vielä silloin niin olemattomat, että kehittäjien oli tunnettava perinpohjaisesti koneiden laitteisto jota ohjelmoitiin, ja kehityksessä oli käytettävä matalan tason ohjelmointiin konekieltä, koska korkeamman tason kielillä ei saatu riittävän tehokasta koodia aikaiseksi. Pelejä tekivät yksittäiset kehittäjät, jotka suunnittelivat, ohjelmoivat ja tekivät grafiikat sekä äänet. Kehitysaika peleissä oli lyhyt, korkeintaan muutama kuukausi. (Madhav, 2013, 2-3.).

Seuraavan sukupolven konsoleissa, kuten Nintendon NESissä, kehittäjiä oli jo enemmän kuin yksi, mutta tiimit pysyivät kuitenkin pieninä. Koodin kirjoitusta oli myös jaettu pelien sisällä, jolloin ohjelmoijilla oli omat vastuualueensa. Ohjelmointi tapahtui edelleen konekielellä, koska laitteiden suorituskyky ei vielä ollut riittävällä tasolla kehittyneemmille ohjelmointikielille. Nintendo tosin tarjosi jo 1980-luvun lopulla kehitystyökaluja, joilla pystyttiin ohjelmointivirheiden jäljitykseen, ja siten parantamaan pelien toiminnallisuutta. (Madhav, 2013, 3.).

1990-luvun puolivälissä Playstationin ja Nintendo 64:n myötä myös kehitysvälineet kehittyivät, ja pelejä pystyttiin jo tekemään pääasiassa c-kielellä, vaikka tehokkuuden kannalta kriittisissä paikoissa jouduttiin edelleen koodaamaan konekielellä. Kehittäjien määrä pysyi edelleen suhteellisen pieninä, 8 – 10 hengen tiimeinä. Suurimmillaankin Playstation 2:n tiimikoot pysyivät 15-20 ohjelmoijan kokoisina konsolisukupolven elinkaaren loppupäässä. (Madhav, 2013, 3.).

Tiimikoot lähtivät kasvuun 2000-luvulla kun seuraavan sukupolven konsolit, Playstation 3, Xbox 360 ja Nintendo Wii tulivat markkinoille. Suurimmissa AAA-luokan peleissä jo pelkkien ohjelmointitiimien koot olivat moninkertaisia verrattuna edelliseen sukupolveen. Mm. vuonna 2011 julkaistussa Assassin's Creed: Revelationsissa ohjelmointitiimi oli yli 75 hengen kokoinen. (Madhav, 2013, 4.).

Syyskuussa 2014 julkaistiin tähän mennessä kaikkien aikojen kallein peli, Halo-pelisarjan tekijän Bungien Destiny. Sen budjetti, joka sisältää kehityskulut, markkinoinnin ja käyttäjätuen, on arvioitu olevan 500 miljoonaa dollaria. (Stuart, 2014). AAA-luokan pelien budjetit ovat siis ohittaneet jo reilusti elokuvat kun vertaa kalleimpaan tähän mennessä julkaistuun elokuvaan Pirates of the Caribbean: At World's Endiin vuodelta 2007, jonka kokonaisbudjetti oli 341,8 miljoonaa dollaria. (Acuna, 2014). Huhujen mukaan Rockstar Northin Grand Theft Auto V:ssä vuodelta 2013 työskenteli 1000 kehittäjää, mutta Rockstar Northin toimitusjohtaja ja GTA V:n tuottaja Leslie Benzies paljasti haastattelussa lokakuussa 2013 luvun olevan huomattavasti suurempi. (French, 2013). GTA V:n budjetin arvioidaan olleen 137:n ja 265:n miljooonan dollarin välillä. Peli rikkoi seitsemän Guinessin maailman ennätystä mm. 800 miljoonan dollarin myynnillään ensimmäisen vuorokauden aikana. (Levy, 2014).

Samaan aikaan kuitenkin indie- ja mobiilipelintekijät kehittävät yksin tai pienillä tiimeillä pienen budjetin pelejä. Digitaalinen jakelu on avannut maailmanlaajuiset markkinat, ja tavallaan ympyrä on sulkeutunut kun menestykseen ei välttämättä enää vaadita valtavaa koneistoa.

3.2 Kehitystiimin tehtävät

Peliprojektin kehitystiimissä on useita tehtäviä, joissa pelinkehittäjät toimivat. Jokaisella pelillä pitäisi olla yksi visionääri, jolla on selkeä kuva siitä millaista peliä ollaan tekemässä. Yleensä roolin täyttää tuottaja (producer), tekninen johtaja (tech lead) tai taiteellinen johtaja (art director). (Bates, 2004, 152).

Tuotannosta voidaan erottaa sisäinen (internal producer) ja ulkoinen tuottaja (external producer). Sisäisen tuottajan tehtävä vastaa muissa ohjelmistoprojekteissa projektipäällikön (project manager) roolia. Se koostuu projektinhallinnasta ja yhteydenpidosta yrityksen johtoon tai ulkopuoliseen julkaisijaan (publisher). Ulkoinen tuottaja on julkaisijan edustaja, joka vastaa pelin valmistumisesta ajallaan ja pysymisestä budjetissa. (Bates, 2004, 153-159).

Suunnittelutiimi koostuu pelisuunnittelijasta (game designer), tasosuunnittelijasta (level designer) ja kirjoittajasta (writer). Pelisuunnittelija vastaa pelin mekaniikasta ja siitä, että pelistä tulee viihdyttävä pelata. Tasosuunnittelija suunnittelee nimensä mukaisesti pelin tasot. Kirjoittaja vastaa pelin juonesta, hahmojen dialogista, pelin manuaalista ja muista kirjoittamista vaativista tehtävistä. (Bates, 2004, 159-165).

Ohjelmointitiimissä on yleensä pääohjelmoija (tech lead tai lead programmer) ja ohjelmoijat (programmers). Pääohjelmoija on projektissa mukana alusta lähtien, ja hänellä pitäisi olla realistinen kuva asioista, jotka ovat toteutettavissa. Pääohjelmoija vastaa pelin ohjelmistoarkkitehtuurista, ja yhdessä muiden ohjelmoijien kanssa pelin teknisestä toteutuksesta. (Bates, 2004, 165-171).

Kuvataiteesta vastaavat taiteellinen johtaja (art lead) ja taiteilijat (artists). Taiteellinen johtaja vastaa pelin ilmeestä ja koordinoi taidetiimiä. Taiteilijoiden tehtävät voidaan jakaa konseptitaiteeseen, hahmojen mallinnukseen, animaatioon, taustojen mallinnukseen ja tekstuureihin. (Bates, 2004, 171-176).

Testiryhmään kuuluvat testauksen johtaja (test lead) ja testaajat (testers). Pelin testaaminen aloitetaan jo projektin varhaisessa vaiheessa, ja sitä jatketaan kunnes peli on valmis. Pelien koon kasvaessa testaus on entistä tärkeämmässä roolissa pelin menestyksen kannalta. Testaajat ovat vastuussa pelin toimivuudesta ja siitä että peli on myös viihdyttävä pelata. (Bates, 2004, 176-180).

Pelinkehitykseen kuuluu useita muita tehtäviä, joihin ei välttämättä pienemmissä pelitaloissa ole asiantuntemusta, ja sen vuoksi niitä on usein ulkoistettu. Sellaisia on musiikin säveltäminen (composer), äänitehosteiden teko (sound designer), vuorosanojen ääninäyttely (voice acting), erilaiset videot, liikekaappaukset (motion capture), kielilokalisaatiot (language localization) jne. (Bates, 2004, 183-200).

3.3 Kehitysprosessi

Pelinkehitysprosessin vaiheet vaihtelevat hieman riippuen lähdemateriaalista. Olli Sinerman mukaan kehitysprosessi voidaan jakaa viiteen osaan. Osia ovat

konseptointi, esituotanto, tuotanto, laadunvarmistus ja ylläpito. (Sinerma, 2009, 16-21).

Konseptointivaiheessa pelin idea kehitetään. Tällöin keksitään pelin asetelma, hahmot, pelimaailma ja kohdealustat. Pelistä voidaan tässä vaiheessa tehdä prototyyppi, jota käytetään idean myymisessä julkaisijalle.

Esituotantoon siirrytään, kun julkaisija on hyväksynyt pelin tuotantoon. Tässä vaiheessa projektitiimin kokoaminen aloitetaan, ratkotaan juridiset ja taloudelliset kysymykset, sekä asetetaan projektitavoitteet. Esituotannossa pelistä toteutetaan esimerkiksi ensimmäinen pelattava kenttä, josta voidaan päätellä onko peli toteuttamisen arvoinen. Mikäli rahoittaja ei siitä vakuutu, projekti voidaan lopettaa esituotantovaiheessa vielä kohtuullisiin kustannuksiin.

Tuotanto jakautuu kolmeen osaan, jotka ovat suunnittelu- toteutus- ja testausvaiheet. Tuotannon tavoitteena on kehittää laadukas peli, tämä on pitkäkestoisin ja kallein vaihe pelinkehityksessä. Suunnittelussa pelin ominaisuudet määritellään ja dokumentoidaan. Seuraavaksi suunnitelmasta toteutetaan peli tuottamalla sen lähdekoodi, grafiikka, äänet, musiikki ja muut pelin vaatimat osat. Testauksessa pelin toteutus testataan, toteutusta verrataan suunnitelmaan, ja analysoidaan pelin viihdyttävyyttä. Nykyaikaisessa pelinkehityksessä tuotannon vaiheet toteutetaan rinnakkain, ketteriä menetelmiä käyttäen.

Laadunvarmistus tarkistaa laitealustan pelille asettamat vaatimukset. Niitä ovat muun muassa valikoiden ulkoasu, tai manuaalin sisältö. Laadunvarmistuksesta läpi päässyt peli on valmis levitettäväksi.

Ylläpitovaiheessa peliin julkaistaan päivityksiä, joilla pelin virheitä korjataan, lisätään sisältöä tai muokataan pelin tasapainotuksia.

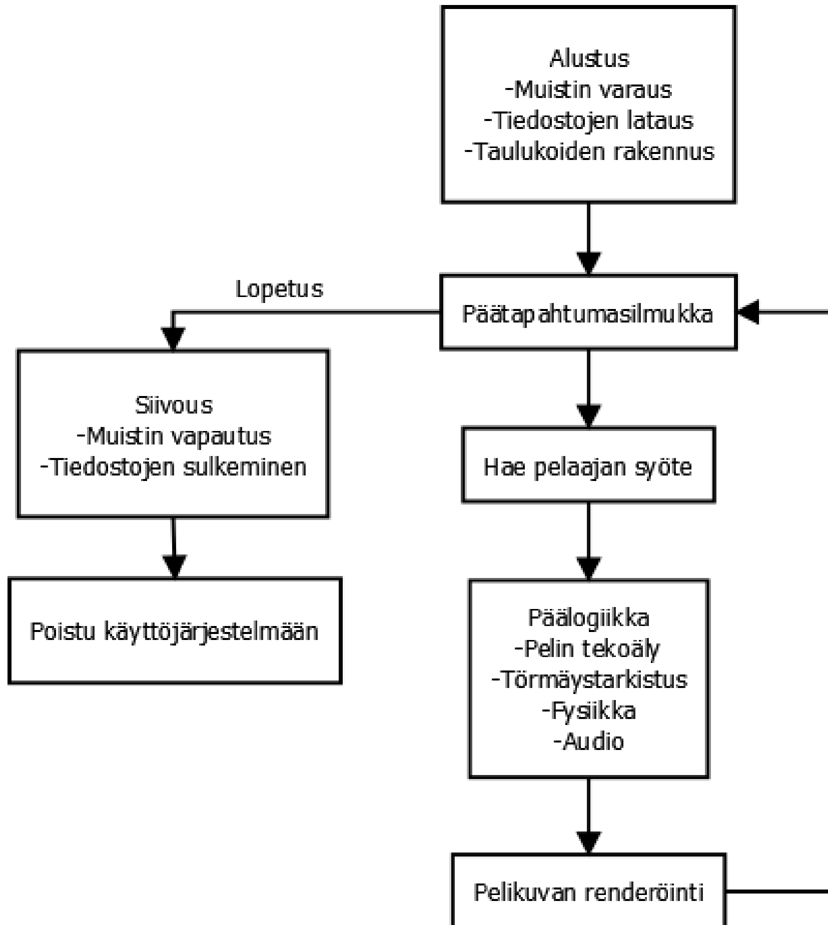
3.4 Peli ohjelmana

Perusajatus videopelistä ohjelmana ei ole juurikaan muuttunut vuosien saatossa. Edelleenkin periaate on sama kuin ennen, vaikka pelit ovat ulkoisesti muuttuneet, ja ohjelmointikielet ja työkalut vaihtuneet. Nykyisin pelintekijän ei tarvitse enää tietää mitä pinnan alla tapahtuu, vaan kehityksessä käytetään korkean tason ohjelmointikieliä, luokkakirjastoja ja pelimoottoreita.

Tässä käydään läpi yksinkertaistetusti pelin vaiheet ohjelman suorituksen aikana (kuvio 10). Kaikki alkaa alustuksista, jolloin ladataan muistiin pelimaailman määrittelyitä, kuten hahmojen ominaisuuksia ja maailman rakenteita sekä ladataan muistiin grafiikkaa, ääniä jne. Tämä tehdään joko peliä käynnistettäessä tai tasolta toiselle siirryttäessä. Yleensä aina pelissä on myös valikko, josta pelimoodit voidaan käynnistää tai asetuksia muuttaa.

Kun kaikki pelin aloitusta varten on tehty, hypätään varsinaiseen pelisilmukkaan, jota käydään läpi niin usein kuin ehditään tai se voi myös olla kiinteästi määritelty esimerkiksi 60:een kertaan sekunnissa eli 60 fps:ään. Yksi frame on tapahtumaikkuna, jonka aikana haetaan käyttäjien syötteet eli se mitä pelaaja haluaa vaikkapa joystickin välityksellä pelissä tehdä. Annetaan pelin tekoälyn siirtää tietokoneen ohjaamia hahmoja, kuten vihollisia. Suoritetaan törmäystar-

kistukset. Suoritetaan fysiikan mallinnukset, esimerkiksi pelihahmon lentoradan laskenta. Reagoidaan törmäyksiin ja soitetaan tarvittaessa äänitehosteita. Lopuksi piirretään näytölle sen hetkinen pelimaailman tila, joka halutaan pelaajalle näyttää. Tätä silmukkaa toistetaan pelin loppuun saakka, jonka jälkeen tehdään normaalit siivoustoimenpiteet ja poistutaan takaisin valikkoon tai suoraan käyttöjärjestelmään.



KUVIO 10 Pelisilmukka

3.5 Luokkakirjastot

Ohjelmointirajapinta eli API tarjoaa kommunikointitavan johonkin ohjelmistokomponenttiin kuten ulkoisiin luokkakirjastoihin tai käyttöjärjestelmän rutiineihin.

Luokkakirjastot ovat kokoelma luokkia, jotka on suunniteltu johonkin tiettyyn tarkoitukseen. Kirjaston käyttäjän ei tarvitse tietää miten kirjasto toteuttaa esimerkiksi eri näytönohjaimilla halutun toiminnon. Tässä käydään läpi muutamia erilaisia kirjastoja, joita pelinkehityksessä voidaan käyttää. Useat kirjastot on toteutettu eri ympäristöihin, jonka etuna on muutostarpeiden vähäisyys koodia siirrettäessä uusille alustoille.

3.5.1 Grafiikkakirjastot

Grafiikkakirjastot tarjoavat työkalut grafiikan renderöintiin ja näytölle piirtämiseen. Renderöinti voidaan toteuttaa joko kokonaan ohjelmallisesti tietokoneen prosessoria käyttäen tai laitteistokiihdytettynä näytönohjaimella.

Esimerkkinä Silicon Graphicsin 1992 julkaisema OpenGL eli Open Graphics Library (OpenGL, 2014), ja sen alijoukko sulautetuille järjestelmille OpenGL ES (Khronos Group, 2014).

3.5.2 Audiokirjastot

Audiokirjastot on tarkoitettu ääniohjelmointiin. Niillä toteutetaan pelin 2D- tai 3D-äänimaailma erilaisine efekteineen. , jossa äänet voivat kuulua eri suunnista.

Esimerkkinä Loki Softwaren alun perin vuonna 2000 kehittämä OpenAL eli Open Audio Library (Wikipedia, 2014h).

3.5.3 Fysiikkakirjastot

Fysiikkakirjastot tarjoavat reaaliaikaista fysiikan mallinnusta peleihin. Niissä on yleensä myös törmäysten tarkistukset mukana.

Esimerkkinä Erin Catton vuonna 2007 julkaisema Box2D (Box2D, 2014), joka soveltuu 2D-pelien fysiikan mallinnukseen.

3.5.4 Multimediakirjastot

Multimediakirjastoiksi kutsutaan kokoelmia, jotka voivat sisältää kaikkia edellä esiteltyjä yksittäisiä kirjastoja. Niiden lisäksi tarjotaan usein rajapintoja mm. käyttäjän syötteeseen eli näppäimistöön, joystickiin ja hiireen, säikeisiin, ikkunoiden hallintaan ja muihin peleissä tarvittaviin toimintoihin.

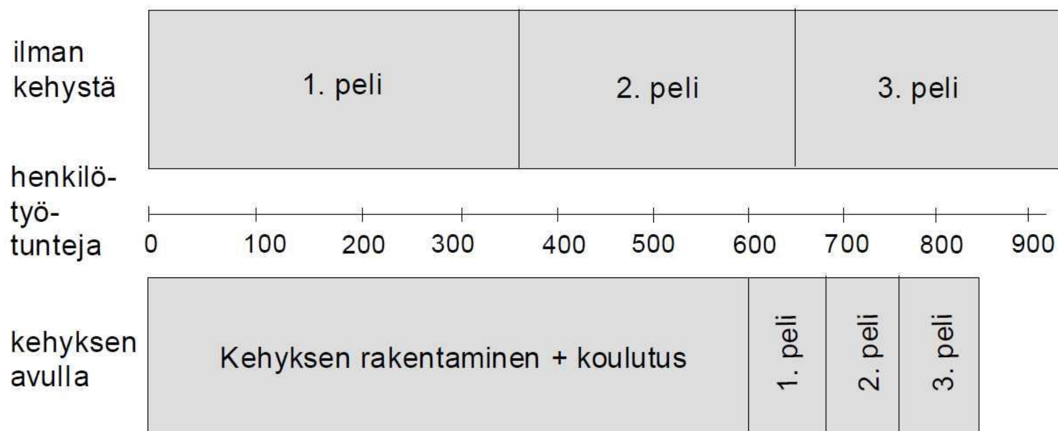
Esimerkkinä Sam Lantingan 1998 julkaisema SDL (SDL, 2014) ja Microsoftin alun perin Windows 95:n mukana vuonna 1994 julkaistu DirectX (Wikipedia, 2014d).

3.6 Pelimoottori

Ohjelmistokehys on luokka-, komponentti ja/tai rajapintakokoelma, joka toteuttaa jonkin ohjelmistojoukon yhteisen arkkitehtuurin ja perustoiminnallisuuden.

Sen sijaan että peli koodattaisiin jokaisen pelin kohdalla alusta asti, kehyksen tekeminen samantyylisiin peleihin nopeuttaa seuraavien pelien kohdalla kehitysaikoja huomattavasti (kuvio 11). Ohjelmistojen uudelleen käytön haittapuolena on kuitenkin tehokkuuden menetys, joten kehyksiä ei välttämättä kan-

nata käyttää sovelluksissa, joissa suorituskyky on kriittinen tekijä. (Koskimies, 2005, 187-189).

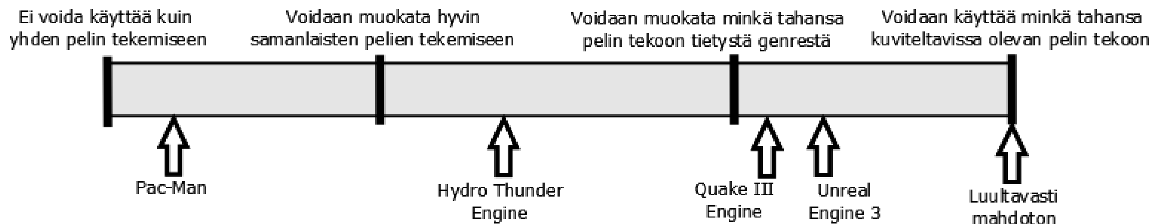


KUVIO 11 Työtuntimäärien vertailu ilman kehystä ja kehyksen kanssa pelisovelluksen tapauksessa (Santelices & Nussbaum, 2001, 1103)

Pelimoottori on videopelien tekoon tarkoitettu ohjelmistokehys. Se tarjoaa uudelleenkäytettäviä komponentteja, joiden avulla kehittäjät voivat rakentaa pelinsä. Moottorin osat suorittavat yleisiä peliin liittyviä tehtäviä, joita voivat olla tiedostojen lataukset, pelikuvan renderöinti, animointi, objektien välisten törmäysten tutkiminen, fysiikan mallinnus, käyttäjän syötteen lukeminen, tekoäly jne. Varsinaisen pelin erottaa moottorista kuitenkin sen sisältö, kuten kehittäjän näkemys pelin logiikasta ja säännöistä. Se miten hahmot pelimaailmassa liikkuvat ja kuinka törmäyksiin reagoidaan, tai käytettävät grafiikat ja äänet.
Siksi saman pelimoottorin päälle tehdyt pelit voivat erota toisistaan niin tyylin kuin genrenkin puolesta. Pelimoottoreiden mukana on yleensä myös editoreita, joilla kehitystä voidaan tehdä.

Toisinaan pelin ja pelimoottorin välinen ero ei välttämättä ole kovin selkeä. Datavetoinen arkkitehtuuri kuitenkin erottaa pelimoottorin ohjelmasta, joka on peli, mutta ei kuitenkaan pelimoottori. Jos pelin logiikka tai säännöt on kirjoitettu ohjelmakoodin sekaan, sen koodin uudelleenkäyttö muissa peleissä on hyvin vaikeaa tai käytännössä mahdotonta. Kun ne sen sijaan ovat erillään alla olevasta moottorista, voidaan uudet pelit rakentaa joutumatta muokkaamaan pelimoottorin koodia. (Gregory, 2009, 11).

Useimmiten peliä varten tehty pelimoottori on viilattu juuri tiettyä peliä varten toimimaan tietyssä ympäristössä niin kuin Pac-Man (kuvio 12). Vaikka pelimoottori olisi tehty kuinka yleiskäyttöiseksi monelle alustalle, on se yleensä käyttökelpoinen vain tietyn genren peleille. Mitä yleiskäyttöisempi pelimoottori on, sitä vähemmän se on optimaalinen tiettyyn peliin, tietylle alustalle. (Gregory, 2009, 12.).



KUVIO 12 Pelimoottorin uudelleenkäytettävyyden kirjo (Gregory, 2009, 12)

Useiden vuosien ajan pelitalojen sisäisten pelimoottoreiden kehittämiskustannukset ovat nousseet, ja samalla yhä useammat yhtiöt ovat erikoistuneet tekemään joko täysiä pelimoottoreita tai pelimoottoreiden komponentteja, joita ne myyvät muille yrityksille. Se on saanut pelitalot pohtimaan miksi tehdä usealla ohjelmoijalla pelimoottoreita, kun on mahdollista ostaa valmista ja toimivaksi testattua teknologiaa huomattavasti halvemmalla.

Pelimoottorit voidaan jakaa kolmeen tasoon niiden käytettävyyden perusteella (taulukko 1).

TAULUKKO 1 Pelimoottoreiden kolme tasoa

Taso	Ominaispiirteitä	Esimerkkejä
Ylin taso	Helppo käyttää. Ei vaadi ohjelmointia. Rajoittunut.	GameMaker, Unity
Keskitaso	Suurelta osin valmis moottori. Vaatii jonkin verran ohjelmointia. Rajoittuneempi kuin alimman tason.	Cocos2D, Unreal Engine
Alin taso	Moottori rakennetaan valmiiden rajapintojen päälle. Vaatii paljon ohjelmointia. Joustava.	OpenGL, DirectX, SDL

Alimmalla tasolla moottori tehdään kokonaan itse valmiiden rajapintojen, kuten OpenGL, DirectX, SDL jne. päälle. Tällainen moottori on hyvin joustava ja sen voi rakentaa täysin vaatimusten mukaiseksi. Se kuitenkin vaatii pisimmän kehitysajan lisäksi paljon asiantuntemusta ja ohjelmointia.

Keskitason moottori on suurelta osin valmis. Sellaisessa on renderöinti, syötteenkäsittely, käyttöliittymät, fysiikat jne. valmiina, mutta se vaatii kuitenkin jonkin verran ohjelmointia, ennen kuin saadaan toimiva peli. Tällaiset moottorit ovat hieman rajoittuneempia kuin alimman tason kokonaan itse tehdyt, mutta ne tarjoavat parempaa suorituskykyä, pienemmällä vaivalla.

Ylimmällä tasolla ovat ns. point-and-click moottorit. Ne vaativat koodausta niin vähän kuin mahdollista, ja monessa onkin ongelmana, että ne voivat olla jopa liian rajoittuneita, jolloin niillä pystytään tekemään ainoastaan johonkin tiettyyn peligenreen pelejä. Tällaisilla moottoreilla saa kuitenkin kehitettyä pelejä nopeasti, ilman suurta työmäärää. (Ward, 2008.).

Pelimoottori koostuu useammista alimoottoreista eli komponenteista. Nykyaikainen pelimoottori sisältää ainakin seuraavia osia:

- Renderöinti
- Animointi
- Törmäysten tarkistus
- Fysiikka
- Käyttäjän syöte
- Käyttöliittymä
- Tekoäly
- Audio
- Verkkoliikenne
- Välianimaatiot
- Skriptaus

3.7 Suosittuja usean alustan pelimoottoreita 2014

Tähän on valittu indie- ja modikehittäjien arviointien perusteella valikoidusti tällä hetkellä suosittuja pelimoottoreita. Lähteinä on käytetty Indie DB (Indie DB, 2014), Mod DB (Mod DB, 2014), Pixel Prospector.com – The Indie Goldmine ja The top 16 game engines for 2014 (Chapple, 2014) sivustoja. Liian samanlaisista pelimoottoreista on otettu ainoastaan yksi, ilman sen kummempaa analysointia paremmuudesta. Seuraavissa luvuissa käytettävien kahden pelimoottorin valinnassa merkitsivät niiden eri käytettävyystasot ja niiden suosio 2D-pelien kehittäjien keskuudessa.

3.7.1 App Game Kit – The Game Creators

App Game Kit on kehitysväline, joka on alun perin tarkoitettu pelien tekoon mobiililaitteille, mutta se tukee myös PC:tä ja Macia. Koodi kirjoitetaan käyttäen joko välineen omaa AGK Basicia tai C++aa.

3.7.2 Cocos2d-x – Open Source

Cocos2D on avoimen lähdekoodin pelimoottori 2D-pelien tekoon. Kehitykseen voi käyttää vaihtoehtoisesti useita kieliä mukaan lukien Ruby, Java, C++, C# ja JavaScript. Cocos2d:stä on tullut hyvin suosittu mobiilipelien kehitysympäristö.

3.7.3 GameMaker: Studio – YoYo Games

GameMaker julkaistiin 1999 ja on suosittu niin kaupallisten kuin inde-studioidenkin keskuudessa. Alun perin 2D-animaatio-ohjelmaksi tarkoitetusta

ohjelmasta päädyttiinkin tekemään pelinteko-ohjelma, jota käytettäessä ei tarvitse osata korkeantason ohjelmointikieliä. Se sopii hyvin myös sellaisille pelintekijöille, joilla ei ole kiinnostusta opiskella ohjelmointia syvällisemmin. Pelien toiminnallisuus toteutetaan ns. raahaa ja pudota tekniikalla sekä käyttämällä pelimoottorin omaa skriptikieltä Game Maker Languagea (GML), jota tulkataan ajon aikana. Sen haittapuolena voi olla hitaus verrattuna käännettyihin peleihin. Nykyisin sillä pystyy tekemään myös 3D-pelejä.

3.7.4 Unity – Unity Technologies

Unity on tällä hetkellä yksi markkinoiden suosituimmista pelimoottoreista, se tukee suurta määrää alustoja. Vuonna 2005 alkunsa saaneesta pelimoottorista on kehittynyt jo niin monipuolinen, että useat pelifirmat ovat alkaneet käyttää sitä myös kaupallisiin peleihin, kuten Richard Garriotin Ultiman perillinen Shroud of the Avatar – Forsaken Virtues. Myös monet suomalaiset pelistudiot käyttävät Unityä. Versiosta 4.3 lähtien Unity on tukenut myös natiivia 2D:tä, vaikka alkujaan se on ollut tarkoitettu 3D-pelien tekoon.

3.7.5 Unreal Engine 4 – Epic Games

Unreal Engine kehitettiin alun perin vuonna 1998 FPS-pelien tekoon, mutta sitä on sittemmin käytetty muidenkin genrejen peleihin. Uusin versio Unreal Engine 4 julkaistiin toukokuussa 2012.

3.8 Yhteenveto

Luvussa perehdyttiin videopelien kehitykseen, miten se on muuttunut ajan myötä, millaisia tehtäviä nykyisin kehitystiimillä on, ja millaisista vaiheista pelinkehitysprosessi koostuu.

Luvussa käytiin läpi yksinkertaisen pelin vaiheet ohjelman suorituksen aikana, millaisia luokkakirjastoja pelinteossa voidaan käyttää, sekä esiteltiin pelimoottori, sen hyödyt ja käytettävyystasot. Lisäksi esiteltiin lyhyesti viisi suosittua pelimoottoria 2D-pelien kehityksessä.

4 SUUNNITELMA PELIN TOTEUTUKSESTA

Tutkielman empiirisessä osuudessa toteutetaan yksinkertainen 2D-videopeli kahdella pelimoottorilla, jotka ovat käytettävyydeltään eri tasolla, ja vertaillaan niiden välisiä eroja pelinkehityksessä. Tässä luvussa esitellään millainen peli on kyseessä, ja millaisissa vaiheissa toiminnallisuus toteutetaan. Erityisenä mielenkiinnon kohteena on pelimoottoreiden helppokäyttöisyys, ja millä työmäärällä kehitys eri alustoille onnistuu.

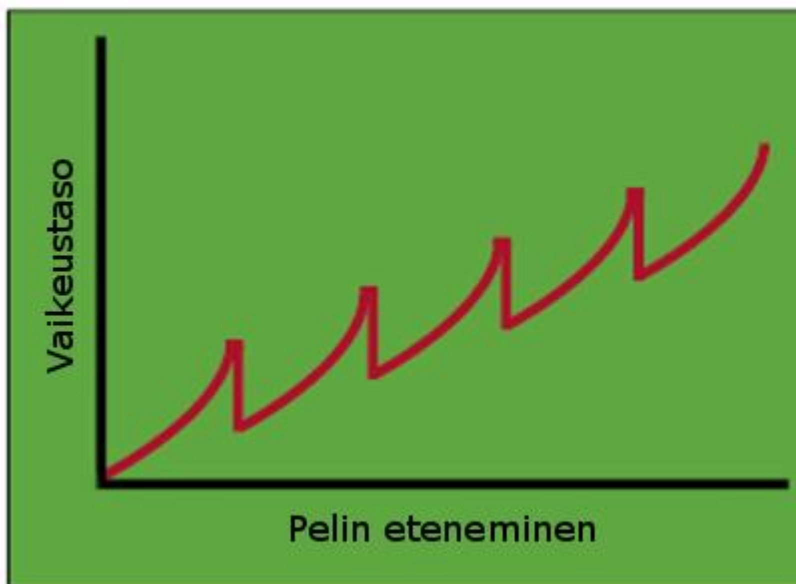
4.1 Toteutettava peli

Toteutettava peli on klooni Taiton suositusta kolikkopelistä, Space Invaders (kuvio 13), vuodelta 1978. Siinä pelaajan tehtävänä on puolustautua lasertykillä hyökkääviä muukalaisia vastaan. Space Invaders käytännössä esitteli videopelit kokonaiselle sukupolvelle, joka ei ollut pelannut koskaan aiemmin. Pelin suosio aikanaan oli jotain ennen näkemätöntä, ja sen seurauksena 2D shoot 'em up genre koki kukoistuksensa 1980-luvulla. 1990-luvun alussa genre oli jalostunut ns. luotihelvetiksi kilpailemaan pelaajien suosiosta uusien 3D-pelien rinnalle.

Pelinä Space Invaders on yksinkertainen, mutta siinä on elementit, jotka löytyvät monimutkaisemmistakin peleistä. Niitä ovat resurssien lataukset, grafiikan piirto ruudulle, yksinkertainen animaatio, pelaajan syötteen luku, äänien toisto, peliobjektien siirrot, törmäysten tarkistukset ja käyttöliittymän piirto.

Alkuperäisen pelin laitteistossa olleen pienen vian vuoksi, vihollisten nopeus kasvoi sitä mukaa, mitä vähemmän niitä oli ruudulla jäljellä. Tämän sattuman seurauksena jokainen taso vaikeutui loppua kohden. Se ei ollut pelintekijän Toshihiro Nishikadon alkuperäinen suunnitelma, mutta koska se lisäsi pelin kiinnostavuutta, vihollisten kiihtyvä nopeus säilytettiin lopullisessa pelissä. Sen lisäksi jokainen taso aloitettiin hieman vaikeampana kuin edellinen, siirtämällä vihollisia aina yhtä riviä lähemmäksi pelaajaa. Space Invadersin jälkeen vaikeustason kasvattaminen otettiin käyttöön lähes kaikissa videopeleissä jollakin tavalla (kuvio 14). (The Game Design Forum, 2012).

KUVIO 13 Space Invaders



KUVIO 14 Pelin vaikeustason kasvu pelin edetessä (The Game Design Forum, 2012)

4.2 Ympäristö

Kehitysympäristönä on 64-bittinen Windows 7. Kohdeympäristöiksi valikoitui sen perusteella PC ja Android, koska iOS ja Windows Phone 8 vaatisivat kumpikin käännösympäristökseen joko Macintoshin tai Windows 8:n.

Käytettävät pelimoottorit ovat Cocos2d-x ja Unity. Valituista moottoreista Cocos2d-x on avoimen lähdekoodin pelimoottori. Unitystä on käytettävissä ilmainen versio, jonka toiminnallisuutta on jonkin verran rajattu, mutta jolla on mahdollista toteuttaa halutunlainen peli.

4.2.1 Kohdeympäristöt

Kohdeympäristöinä ovat Windows tietokoneet sekä Android puhelimet ja tabletit.

Androidille kehitettäessä on huomioitava Android-laitteiden suuri kirjo. Puhelimien ja tablettien tehot, muistien määrät ja näyttöjen resoluutiot vaihtelevat suuresti. Näyttöjen kosketustunnistimien ominaisuudetkin vaihtelevat mallikohtaisesti, joka on otettava huomioon suunniteltaessa pelin kontrolleja.

PC:lle kehitettäessä voidaan olettaa pelaajalla olevan käytettävissä ohjaamiseen ainakin näppäimistö ja hiiri. Nykyisissä näytönohjaimissa on riittävästi tehoja 2D-pelien perusgrafiikkaan. Shadereiden, eli varjostimien käyttö erikoistehosteiden piirtämisessä voi kuitenkin olla vanhemmille näytönohjaimille liian raskasta sujuvan kuvan piirtoon.

4.2.2 Kehitysvälineet

Käytettävinä pelimoottoreina ovat Cocos2d-x 3.3rc0 ja Unity 4.5.5f1. Unityssä on oma kehitysympäristö, mutta C#-skriptejä kirjoitettaessa joudutaan käyttämään jotain muuta editoria, kuten Visual Studiota tai Unityn mukana tulevaa MonoDevelopia. Cocos2d-x:ää käytetään suoraan jollain C++ editorilla, kuten Visual Studiolla tai Eclipsellä. Cocos2d-x tarjoaa myös Lua ja JavaScript rajapinnat, ja silloin editorina voidaan käyttää Cocos Code IDEä.

Developer Economicsin mobiilikehittäjille vuonna 2014 tekemän kyselytutkimuksen mukaan Unity on tällä suosituin ja Cocos2d toiseksi suosituin pelimoottori mobiilipelinkehityksessä (Wilcox, 2014). Cocos2d on avoimen lähdekoodin 2D-pelimoottori, joka kehitettiin alun perin Pythonilla. Pian ensimmäisen version jälkeen Apple julkaisi AppStore kauppapaikan, ja sille sovelluskehitysympäristön. Sen seurauksena Cocos2d uudelleen kirjoitettiin Objective-C:llä, ja julkaistiin nimellä Cocos2d for iPhone v0.1. Tämän jälkeen Cocos2d:n pohjalta on tehty useita kehityshaaroja, joilla eri kohdealustoille on rajapintoja, joilla voidaan kehittää pelejä eri ohjelmointikielillä (taulukko 2).

Valituista pelimoottoreista Cocos2d:n Cocos2d-x haara on ainut, jossa kehitys voidaan toteuttaa kokonaan C++:lla. C++ on yksi suosituimmista ohjelmointikielistä pelinkehityksessä. Sen lisäksi kehitettäessä peliä usealle alustalle,

Cocos2d-x:n lähdekoodi toimii suurimmalta osin sellaisenaan alustasta riippumatta. Cocos2d-x:n tukemia mobiilialustoja ovat iOS, Android, Windows 8 ja Windows Phone 8, työpöytäalustoja Windows, Linux ja Mac OS X. Muiden Cocos2d-ympäristöjen tukemia alustoja ovat Windows Phone 7, Windows 7, Xbox 360 ja HTML5 selaimet. Cocos2d-xn kanssa voidaan käyttää CocoStudiota, jossa on työkalut pelin käyttöliittymän suunnitteluun, animaatioeditori, datanmuokkaaja ja tasoeditori.

TAULUKKO 2 Cocos2d:n kehityshaarat (Wikipedia, 2014b)

Kehityshaara	Kohdealusta	API kieli
Cocos2d	Windows, Mac OS X, Linux	Python 2.6, 2.7, 3.0+
Cocos2d-x	iOS, Android, Windows 8, Windows Phone 8, Windows, Linux, Mac OS X	C++, Lua, JavaScript
Cocos2d-JS	HTML5 (Selaimet)	JavaScript
Cocos2d-Swift	iOS, Mac OS X, Android	Objective-C, Swift
Cocos2d-xna	Windows Phone 7 & 8, Windows 7 & 8, Xbox 360	C#

Unity on hyvin suosittu pelimoottori, jossa on ollut natiivi 2D tuki versiosta 4.3 lähtien. Siinä pelin toiminnallisuus toteutetaan ”raahaa ja pudota”-tekniikan lisäksi joko C# tai JavaScript kieliä käyttävillä skripteillä. Unityllä voi yleisimpien mobiilikäyttöjärjestelmien ja Windowsin lisäksi kehittää pelejä myös muille suosituimmille käyttöjärjestelmille ja nykykonsoleille mukaan lukien PS3, PS4, PS Vita, Xbox 360, Xbox One ja Wii U. Tällä hetkellä Unityllä voidaan tehdä selaimessa toimivia pelejä, jotka käyttävät Unityn omaa selainliitännäistä, mutta tulevaisuudessa selaimen HTML5 tuki riittää.

Cocos2d-x ja Unity tukevat kumpikin pelinkehitystä suosituimmille tietokone- ja mobiilialustoille (taulukko 3). Natiivien pelisovellusten lisäksi, HTML5:ttä tukeviin selaimiin voidaan peli toteuttaa Cocos2d-JS:llä, ja tulevaisuudessa myös Unityllä. Tällä hetkellä Unity vaatii erikseen asennettavan liitännäisen selaimeen.

TAULUKKO 3 Cocos2d-x:n ja Unityn tukemat tietokone- ja mobiilialustat

	Android	iOS	Windows Phone 8	Windows	Mac	Linux	HTML5	Selain liitän.
Cocos2d-x	x	x	x	x	x	x	*	-
Unity	x	x	x	x	x	x	**	x

x tukee, * Cocos2d-JS, ** tulossa

Molemmat pelimoottorit käyttävät avoimen lähdekoodin Box2Dtä fysiikkamoottorinaan. Cocos2d-x:ssä fysiikkamoottorina on kuitenkin oletuksena Chipmunk.

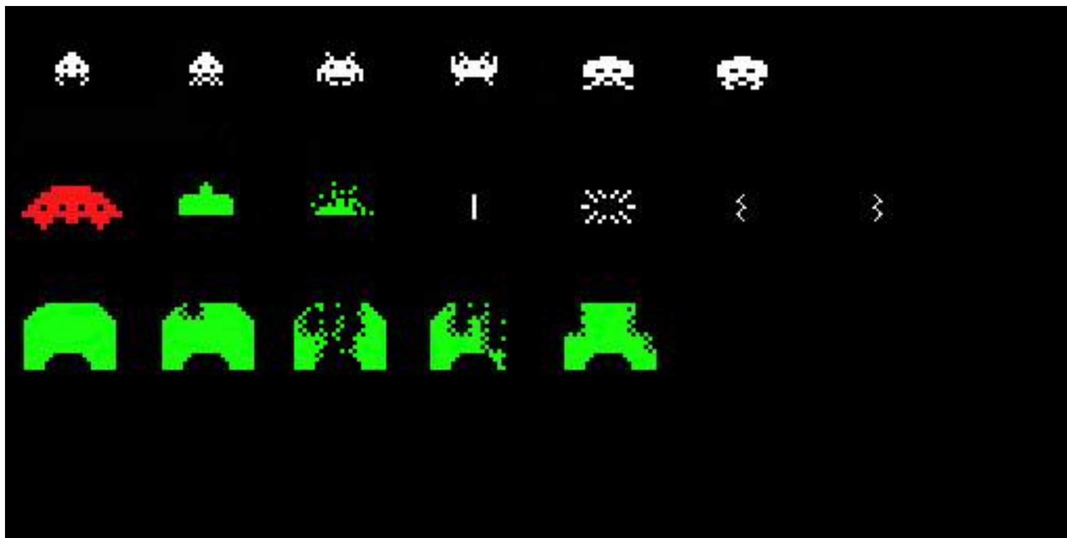
4.3 Kehitysvaiheet

Pelinkehityksen eri vaiheet toteutetaan molemmilla pelimoottoreilla samanaikaisesti ennen kuin siirrytään seuraavaan vaiheeseen. Se mahdollistaa pelimoottoreiden paremman vertailun keskenään. Tässä aliluvussa käydään kehitysvaiheet läpi.

4.3.1 Resurssien lataus

Videopelit koostuvat erilaisista resursseista, kuten pelin grafiikat, animaatiot, äänet, musiikit jne. Resurssit ladataan keskusmuistiin tai näytönohjaimen muistiin peliä käynnistettäessä, tai pelin tasoa ladattaessa. Ne voidaan myös ladata kesken peliä.

Videopeleissä grafiikalla on iso osa pelikokemuksessa. Grafiikan lataukseen kuuluvat taustakuvien ja peliobjektien kuvien lataukset. 2D-peliobjektien kuvat ovat nimeltään spritejä ja ne ovat yleensä sijoitettu kuvaan tai kuviin, joita kutsutaan sprite sheeteiksi (kuvio 15).



KUVIO 15 Space Invadersin sprite sheet

Ääniresursseja ovat ääniefektit, musiikki ja ääninäyttely. Ääniefekteillä luodaan peliin äänimaailma, jossa aseet laukeavat ja auton moottorit jylisevät. Ääniefektien lisäksi pelin tunnelmaa luodaan taustamusiikilla, joka peleissä usein muuttuu kohtauksen mukaan. Ääninäyttely on näyttelijöiden puhumaa dialogia, jota pelihahmot puhuvat.

Fonttitiedostoja käytetään pelissä käytettävien erilaisten tekstien ulkoasun määrittelytiedostoina. Ne voivat myös olla sprite sheetejä, joissa käytettävät merkit ovat omina kuvinaan.

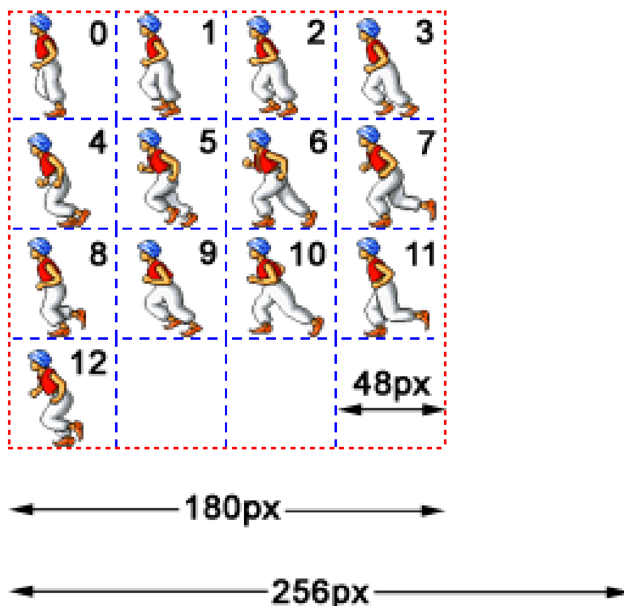
Skripteillä pelin toiminnallisuus on erotettu erillisiin tiedostoihin, joita pelimoottorit käyttävät kovakoodaamisen sijasta. Niissä voidaan esimerkiksi mää-

ritellä pelin sääntöjä tai miten peliobjekti reagoi toisen peliobjektin kanssa törmäämiseen.

4.3.2 Grafiikka ja animaatio

Peliobjektien grafiikoita kutsutaan 2D-peleissä spriteiksi. Aikoinaan monissa kotimikroissa spritet oli toteutettu laitteistotasolla, ja vaikka nykyisin ne toteutetaankin ohjelmallisesti, niistä käytetään silti samaa nimitystä. Taustakuvat eivät varsinaisesti ole spritejä, mutta 2D-pelimoottoreiden termistössä niistä käytetään yhtenäisyyden vuoksi samaa nimeä. Peleissä voidaan käyttää myös partikkeleita, jotka koostuvat useista pienistä spriteistä tai muista graafisista objekteista. Niiden avulla voidaan esittää vaikkapa pakokaasuvanaa tai räjähdyksiä.

Animaatioilla saadaan pelihahmoille elävyyttä, joka ei pelkillä kuvilla olisi mahdollista. Pelihahmoilla on yleensä paikallaan seisomiseenkin oma animaationsa. Animaatioiden perusosat piirretään jollain piirto-ohjelmalla, jonka jälkeen niiden liikeanimaatiot voidaan toteuttaa suoraan pelimoottoreiden mukana tulevilla animaatioeditoreilla. Unityssä editori löytyy suoraan, ja Cocos2d-x:n kanssa voidaan käyttää CocoStudiota. Vaihtoehtoisesti on mahdollista piirtää liikeanimaation yksittäiset kuvat valmiiksi, ja siirtää ne sprite sheetiin, josta ne voidaan ladata animaatioiksi pelin käyttöön. Esimerkkikuvassa pelaajahahmon juoksuanimaation kuvat on asetettu samaan isoon kuvaan, josta ne näytetään peräjälkeen, jolloin pelihahmo näyttää juoksevan (kuvio 16). Pelihahmon animaatio vaihtuu sen perusteella, mitä hahmo milläkin hetkellä on tekemässä.



KUVIO 16 Juoksuanimaation sprite sheet pelistä Prince of Persia. (Patricios, 2011).

Space Invadersin grafiikat ovat hyvin yksinkertaiset. Pelissä on kolme erilaista muukalaishahmoa, pelaajan lasertykki, suojamuurit ja emoalus. Animaatio on

minimalistista. Vihollisten jalkojen liikkeitä on kaksi, joita vaihdellaan jokaisella siirrolla, jolloin jalat näyttävät liikkuvan. Lisäksi ainoastaan vihollisen ammuksella on kahden kuvan animaatio. Toteutettavaan peliin ei kuitenkaan tehdä animaatioita.

4.3.3 Peliobjektien ja pelimaailman määrittely

Peliobjektit ovat pelissä olevia elementtejä, jotka voivat olla staattisia, dynaamisia tai partikkeleita. Niille voidaan määritellä ominaisuuksia, kuten sijainti, nopeus, liikkeen suunta, osumien kesto, massa, tyyppi, nimi jne. Objektit näytetään yleensä ruudulla, jolloin niille on määritelty näytettäväksi jokin kuva tai animoitu sprite. Useimmiten objekteilla on jokin näytettävästä kuvasta riippumaton muoto, jolloin muut pelimaailman objektit voivat reagoida niiden kanssa. Yksinkertaisimmillaan se voi olla 2D-peleissä suorakaide tai ympyrä. Kahden peliobjektin törmäyksen tutkiminen ja siihen reagointi on silloin hyvin helppoa, eikä suurta laskentatehoa tarvita.

Dynaamisia objekteja liikuttavat pelaajat, pelin tekoäly tai fysiikkamoottorin voimat. Staattisia objekteja ovat esimerkiksi tasohyppelyn tasot, jotka eivät välttämättä reagoi pelaajan hahmoon mitenkään, mutta pelaajan hahmo voi reagoida niihin. Edellä mainittujen lisäksi pelissä voi olla partikkeleita. Ne ovat esimerkiksi raketin jälkipolttimen liekit tai leijailevat lumihiutaleet, jotka eivät ole vuorovaikutuksessa muiden peliobjektien kanssa. Partikkeleiden elinaika on yleensä suhteellisen lyhyt.

Space Invadersissa peliobjekteina ovat pelaajan liikuttama lasertykki, hyökkäävät muukalaiset, vihollisen emoalus, suojamuurit sekä vihollisen ja pelaajan laserit.

4.3.4 Pelaajan syötteen käsittely

Pelaajan tai pelaajien syötteet voivat tulla näppäimistöltä, hiireltä, peliohjaimelta, kosketusnäytöltä, kiihtyvyysanturilta tms. Syötteen perusteella pelaajan hahmoa liikutetaan, tehdään valintoja pelissä tai liikutaan valikoissa.

Pelaajan syöte voidaan lukea pelisilmukan jokaisella kierroksella. Tällöin esimerkiksi siirretään pelaajan hahmoa oikealle niin pitkään kuin pelaaja kääntää ohjainta oikealla. Vaihtoehtoisesti syötteen luku toteutetaan tapahtumapohjaisesti. Silloin peliin on kirjoitettu käsittelijä, joka suoritetaan kun pelaaja aloittaa peliohjaimen kääntämisen, ja kun kääntö loppuu.

Space Invadersissa pelaajan kontrollit ovat yksinkertaiset. Pelaaja liikuttaa lasertykkiä sivusuunnassa ja ampuu nappia painamalla. Valikoissa liikkuminen tapahtuu joko hiirellä tai kosketusnäytöltä valitsemalla. Toteutusvaiheessa täytyy selvittää miten näppäimistön ja kosketusnäytön käskyjä luetaan eri pelimoottoreilla, ja eri laitteilla.

4.3.5 Peliobjektien liikuttaminen

Pelaajan pelihahmoa liikutetaan edellisessä aliluvussa kerrotulla tavalla. Vihollisen pelihahmojen liikuttamisen hoitaa tietokoneen tekoäly tai valmiit mallit, joiden mukaan pelihahmot käyttäytyvät. Pelien tekoälyllä ei välttämättä tarkoiteta kuitenkaan samanlaista älykkyyttä mitä tekoälytutkimuksessa. Peleissä laskenta-aika tekoälylle on yleensä lyhyt, ja mikä tärkeintä, pelin tekoälyn ei ole välttämättä tarpeen olla edes kovin älykäs. Pelin on oltava tarpeeksi haastava, mutta tekoälyn on myös tehtävä virheitä, jotta peli on hauska pelata. Siksi usein riittääkin, että tietokoneen ohjaamat pelihahmot toistavat vain jotain ennalta määriteltyä kaavaa.

Varsinaista tekoälyä Space Invadersissa ei siis ole, vaan vihollislaivasto hyökkää pelaajaa kohden tietyn kaavan mukaan. Koko laivue siirtyy vasemmasta reunasta oikeaan ja takaisin, ja aina ruudun reunassa siirretään kaikkia vihollisrivejä lähemmäksi, kohti pelaajan puolustamaa planeettaa. Aika ajoin jokin lähimpänä olevista vihollisista ampuu kohti pelaajaa.

4.3.6 Törmäystarkistukset ja käsittely

Törmäystarkistukset ja niiden käsittely ovat olennainen osa videopelejä. Ilman törmäyksiä suurin osa peligenreistä olisi turhia. 2D-peleissä nimensä mukaisesti tutkitaan törmäyksiä kahdessa ulottuvuudessa. 3D-peleissä ulottuvuuksia on kolme eli niissä on otettava syvyys mukaan laskuihin. Jotta kahden kappaleen välinen törmäys voidaan havaita, on kummallakin oltava jokin muoto, jotka leikkaavat keskenään. Yksinkertaisimmillaan kappaletta voidaan ajatella ympyränä, josta tiedetään sen keskipisteen sijainti tasolla sekä ympyrän säde. Kahden ympyrän välinen törmäys saadaan selville laskemalla pisteiden välinen etäisyys, josta vähennetään ympyröiden säteet. Mikäli tulos on negatiivinen, kappaleiden välillä on törmäys, johon reagoidaan. Reaktio riippuu pelistä, ja se voi olla ympyröiden kimpoaminen toisistaan pois, toisen räjähtäminen tai jotain muuta.

Kappaleiden muodot, joiden törmäystä muiden pelimaailman kappaleiden kanssa tutkitaan, voivat vaihdella yksinkertaisesta pisteestä hyvinkin monimutkaisiin 3D-malleihin. Nykyisissä pelimoottoreissa on fysiikkamoottori, jossa on sisäänrakennettuna törmäystarkistukset. Fysiikkamoottorin avulla pelimaailmalle voi antaa ominaisuuksia, kuten liukas tie tai planeetalle suuren gravitaation, jolloin pelihahmot reagoivat pelimaailman olosuhteisiin realistisesti. Mikäli törmäystarkistus suoritetaan kappaleiden välillä ainoastaan niiden hetkellisten sijaintien perusteella, voivat nopeasti liikkuvat kappaleet mennä toistensa läpi, koska törmäystä ei havaita. Ongelmaa kutsutaan tunneloinniksi. Se voidaan kiertää tutkimalla nopeasti liikkuvien kappaleiden väliset törmäykset yhden kerran sijaan useita kertoja saman pelisilmukan kierroksella eli jatkuvalla törmäystarkistuksella.

Space Invadersissa ei kuitenkaan tarvita fysiikkamoottoria, ja törmäysten tarkistuksessakin voidaan peliobjekteja ajatella suorakaiteen muotoisina kappaleina. Pelaajan ammuksen osuessa viholliseen, vihollinen kuolee ja pelaaja saa

pisteen. Pelaajan osuessa suojamuuriin, ensimmäisestä osumasta suojamuuri hajoaa ja toisesta osumasta se tuhoutuu lopullisesti. Vihollisten ammuksista pelaaja kuolee. Mikäli vihollislaivue pääsee osumaan pelaajaan, pelaaja kuolee ja peli loppuu.

4.3.7 Äänet

Äänet ja musiikki ovat osa pelimaailmaa, jotka luovat siihen omalta osaltaan tunnelman.

Space Invadersissa ei musiikkia käytetä. Ääniefekteinäkin on ainoastaan pelaajan lasertykin laukaisu, vihollisen tulitus, emoaluksen läsnäolon ilmoittama humina ja osuman aiheuttama räjähdys.

4.3.8 Käyttöliittymä

Pelinaikaisesta käyttöliittymästä pelaaja näkee tarpeellista informaatiota. Käyttöliittymä näytetään varsinaisten pelitapahtumien edustalla, ja se voi sisältää muun muassa pisteet, elämät, panosten määrän, valitut aseet ja niin edelleen.

Space Invadersissa näytettäviä asioita ovat pisteet, ajantasainen high score ja pelaajan elämien määrää kuvastavat pienemmät lasertykit alalaidassa. Toteutettavassa pelissä kuitenkin riittää näytettäväksi ainoastaan pelaajan pisteet.

4.3.9 Valikot

Peleissä on yleensä ainakin päävalikko, josta peli käynnistetään ja siitä poistutaan takaisin käyttöjärjestelmään. Useimmiten peleissä on myös valikot, joissa voi muuttaa pelin asetuksia, kuten vaihtaa näytön resoluutiota ja määritellä peliohjaimen nappeja. Valikoissa komponentteina on yleensä aina tekstiä, kuvia ja jonkinlaisia painonappeja, joiden avulla valikoiden ja pelin välillä voidaan navigoida.

Space Invadersin valikko on yksinkertainen. Siinä näytetään ainoastaan pelin nimi, jonka lisäksi ruudulla on kaksi painonappia, joista toisesta peli käynnistyy, ja toisesta pelistä poistutaan.

4.4 Tiedon keruu eri vaiheissa

Tiedon keruu toteutetaan pelien kehitysten edetessä. Erityisenä mielenkiinnon kohteena ovat eri vaiheissa toteutuksen vaikeus ja loogisuus, eri alustojen vuoksi huomioon otettavien asioiden määrä, sekä käytettävissä olevan dokumentaation määrä ja laatu.

Tuloksena on kehityksen eri vaiheissa syntyvää tietämystä kyseiseen ongelmaan, lähdekoodia, kuvankaappauksia ja vertailua pelimoottoreiden välisis-

tä eroista. Kehityksen vaiheista saatu tieto on arvokasta kehitykseen käytettävän pelimoottorin valintaa tehtäessä.

4.5 Yhteenveto

Luvussa esiteltiin tutkielman empiirisessä vaiheessa toteuttava Space Invaders peli. Tarkasteltiin millaisessa ympäristössä, mille alustoille ja millä kehitysvälineillä peli toteutetaan. Lisäksi esiteltiin osat, joista peli koostuu. Ne ovat samalla kehitysvaiheet, joissa peli toteutetaan. Lopuksi kuvailtiin millaista tietoa kehityksen eri vaiheista halutaan kerätä.

5 PELIN TOTEUTUS

Valitut pelimoottorit ovat pelinkehittäjän kannalta käytettävyydeltään kahta tasoa. Unity edustaa ylintä tasoa, jossa ohjelmointia tarvitaan melko vähän tai ei lainkaan. Pelin toiminnallisuus joudutaan kuitenkin toteuttamaan skriptikielillä, jotka ovat C#, JavaScript tai Boo. Cocos2d-x taas on keskitason moottori, ja se vaatii jo C++ ohjelmoinnin osaamista. Vaihtoehtoisesti Cocos2d-x tarjoaa Lua ja JavaScript rajapinnat ohjelmointiin. Tässä luvussa käydään läpi Space Invaders kloonin kehitys vaiheittain eri pelimoottoreilla, ja miten niiden lähestyminen ongelmaan eroaa toisistaan. Käytettävinä kielinä ovat C++ ja C#. Pelistä ei toteuteta loppuun asti viimeisteltyjä versioita, vaan pääpaino on pelimoottoreiden arvioimisessa 2D-pelien kehityksessä.

5.1 Kohdelaitteiden suuri kirjo

Android on tällä hetkellä suosituin mobiililaitteiden alusta. Sen markkinaosuus uusien puhelimien ja tablettien käyttöjärjestelmänä toisella kvartaalilla 2014 oli 84,7%. (IDC, 2014). Net Market Share seuraa internetissä käytettyjä teknologioita. Sen mukaan lokakuussa 2014 Androidin markkinaosuus kaikista käytetyistä mobiililaitteiden käyttöjärjestelmistä oli 46,38% ja Windowsin eri versioiden yhteenlaskettu osuus tietokoneiden käyttöjärjestelmistä oli 91,75% (Net Market Share, 2014).

Android-laitteita on markkinoilla eri käyttöjärjestelmäversioiden lisäksi paljon eri tehoisia, ja erilaisilla näyttöjen resoluutioilla ja ominaisuuksilla varustettuina. Laitteiden suuri kirjo tekee kehityksen jossain määrin ongelmalliseksi, verrattuna kehitykseen Windows tietokoneille tai Applen iOSlle.

5.2 Pelimoottoreiden dokumentaatio

Molemmilla pelimoottoreilla on kattavat kotisivut. Ohjelmointirajapinnat on dokumentoitu selkeästi, ja tarvittava tieto löytyy helposti. Cocos2d-x tiimi on aloittanut ohjelmointioppaan kirjoittamisen, mutta joulukuussa 2014 se ei vielä kata kaikkea tarpeellista tietoa pelinkehitykseen pelimoottorin avulla. Unityn kehitystiimi taas on panostanut huomattavasti enemmän erilaisiin kehitysoppaisiin. Niihin kuuluu oppikirjamaisten oppaiden lisäksi myös videoituja oppaita. Dokumentaation määrässä avoimen lähdekoodin Cocos2d-x häviää kaupalliselle Unitylle. Molempien pelimoottoreiden opiskeluun on julkaistu useita kirjoja, mutta kokeneelle pelinkehittäjälle löytyy riittävästi hyvätasoista dokumentaatiota ilmaiseksikin.

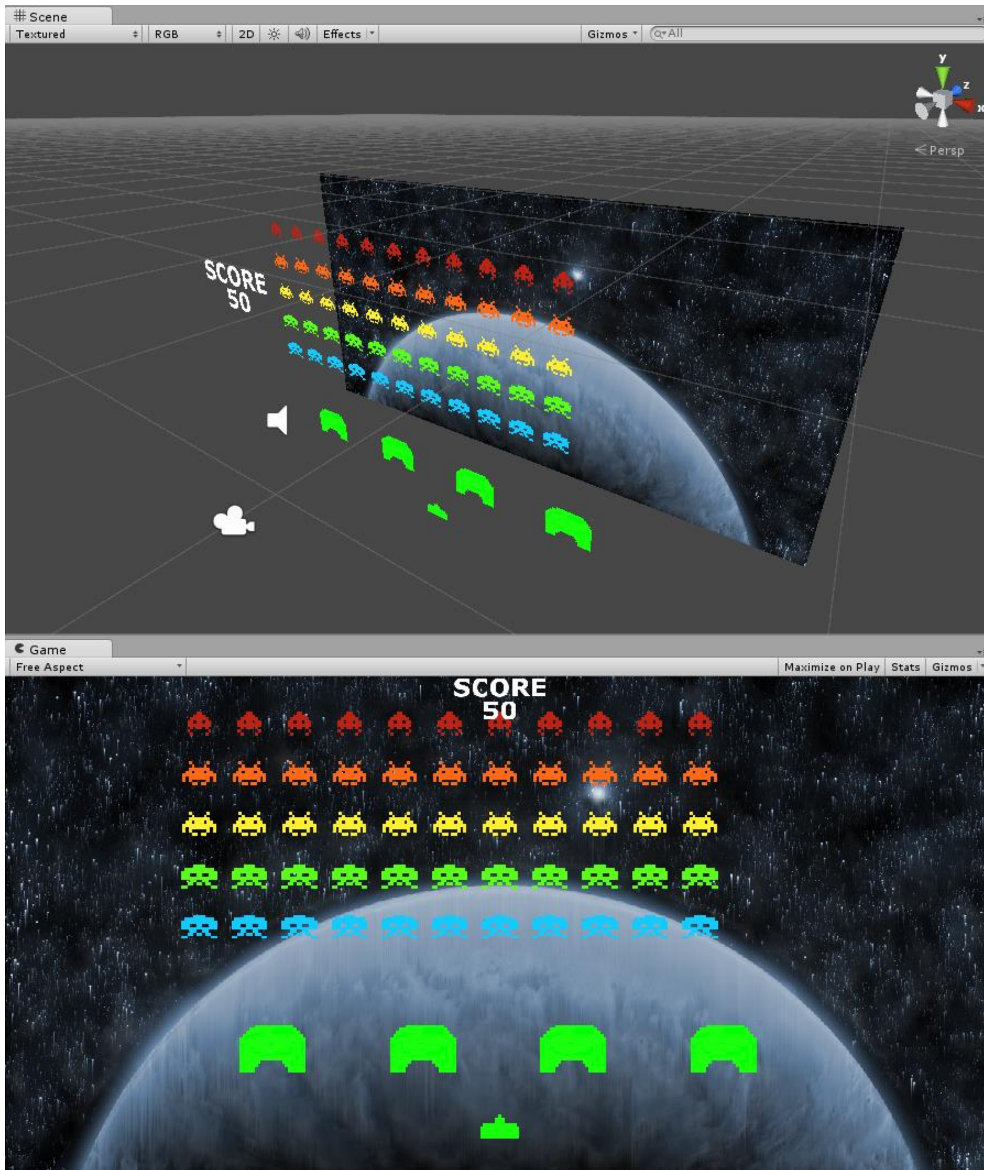
5.3 Peruskäsitteet

Pelimoottoreiden käyttämä sanasto ei välttämättä ole kovinkaan yhtenäinen. Cocos2d-x:n ja Unityn käyttämät termit ovat kuitenkin melko samankaltaiset. Tässä aliluvussa selitetään niistä tarpeelliset.

Asseteilla tarkoitetaan pelin resursseja, kuten kuvia, ääniä ja skriptejä. Pelit rakennetaan asseteilla näyttämöille eli skeneille. Skene voi olla valikko tai pelikenttä. Skenet koostuvat yhdestä tai useammasta päällekkäisestä kerroksesta. 2D-peleissä syvyysvaikutelmaa ei kuitenkaan ole (kuvio 17), joten eri kerroksissa olevien spritejen sijainnin syvyyssuunnassa voi päätellä ainoastaan, mikäli ne jäävät toisten spritejen taakse. Peliobjektit ovat pelin elementtejä, kuten pelaaja tai kamera. Peliobjekteilla voi olla jokin hahmo, joka on 2D-peleissä sprite. Spritet ovat kuvia joita pelissä voi siirtää, kääntää ja niin edelleen. Usein ne esitetään animaationa. Fysiikkamoottoria käytettäessä peliobjektille on määriteltävä jäykkäkappale, jota käytetään myös objektien välisten törmäysten tarkistuksissa.

5.4 Kehitysympäristöjen käyttöönotto

Pelimoottoreiden erot näkyvät jo kehitysympäristöjen käyttöönotossa. Unityn asennus on suoraviivainen, eikä asennusvaiheessa tarvitse valita muuta kuin Unityn asennushakemisto. Pelin kehityksen voikin aloittaa suoraan luomalla uuden projektin, jonka jälkeen peli on käännettävissä PC:lle. Androidille kehitettäessä on koneelle asennettava vielä Android SDK, ja sen vaatima Java Development Kit.



KUVIO 17 Syvyysvaikutelma 3D- ja 2D-näkymissä

Cocos2d-x ei vaadi ylimääräisten kirjastojen asentamista kun peliä kehitetään ainoastaan PC:lle, mutta aivan kuten Unityllä, Android kehitys vaatii Cocos2d-x:lläkin Android SDKn asennuksen. Pelkän SDKn sijaan on syytä asentaa Android Development Tools, jossa tulee mukana myös Eclipse-kehitysympäristö. Edellisten lisäksi tarvitaan Android NDK, joka mahdollistaa sovelluskehityksen natiivikielillä, kuten C:llä ja C++:lla. Lisäksi vaaditaan Apache ANT, joka on Java-kirjasto pelin kääntämisen helpottamiseksi.

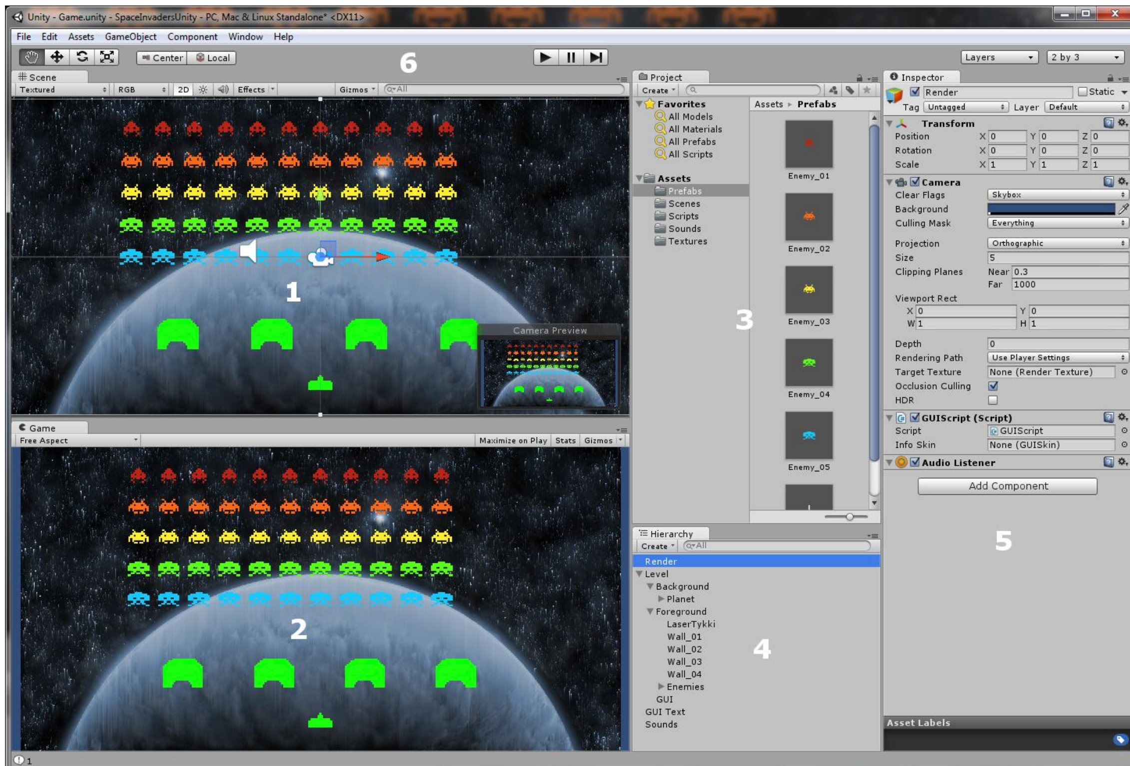
Cocos2d-x:ää käyttöönotettaessa on koneella oltava asennettuna jokin kehitysympäristö. Tässä tapauksessa se oli Microsoftin Visual Studio Express 2012. Android kehityksessä Eclipse on kuitenkin tällä hetkellä parempi vaihtoehto. Lisäksi vaaditaan Python, jota tarvitaan uusien projektien luomisessa, koska nämä toteutetaan python skripteillä.

Unityn kehitysympäristön käyttöönotto on huomattavasti helpompaa ja nopeampaa kuin Cocos2d-x:n.

5.5 Pelimoottoreiden editorit

Unityn editori (kuvio 18) mahdollistaa lähes koko pelin kehityksen graafisen käyttöliittymän avulla. Editori koostuu useista näkymistä, joiden sijaintia ruudulla voi muokata mieleisekseen. Näkymät ovat:

1. Scenenäkymä on interaktiivinen hiekkalaatikko, jossa muokataan ”raahaa ja pudota”-tekniikalla skenejä, eli pelin tasoja ja valikkoja. Tässä näkymässä voi luoda tasot alusta loppuun lisäämällä pelaajat, viholliset, taustat, tasot, kamerakulmat ja muut peliobjektit halutuille paikoilleen.
2. Pelinäkymä on kameran renderöimä kuva pelistä. Se näyttää pelikuvaa, jonka pelaaja näkee valmiissa pelissä.
3. Projektinäkymästä voi hallita kaikkia assetteja, jotka kuuluvat projektille.
4. Hierarkia sisältää kaikki valitun skenen peliobjektit.
5. Inspector näkymä esittää valitun peliobjektin yksityiskohtaisen informaation, sisältäen sille liitetyt komponentit ominaisuuksineen.
6. Työkalupalkki sisältää editorin peruskontrollit.

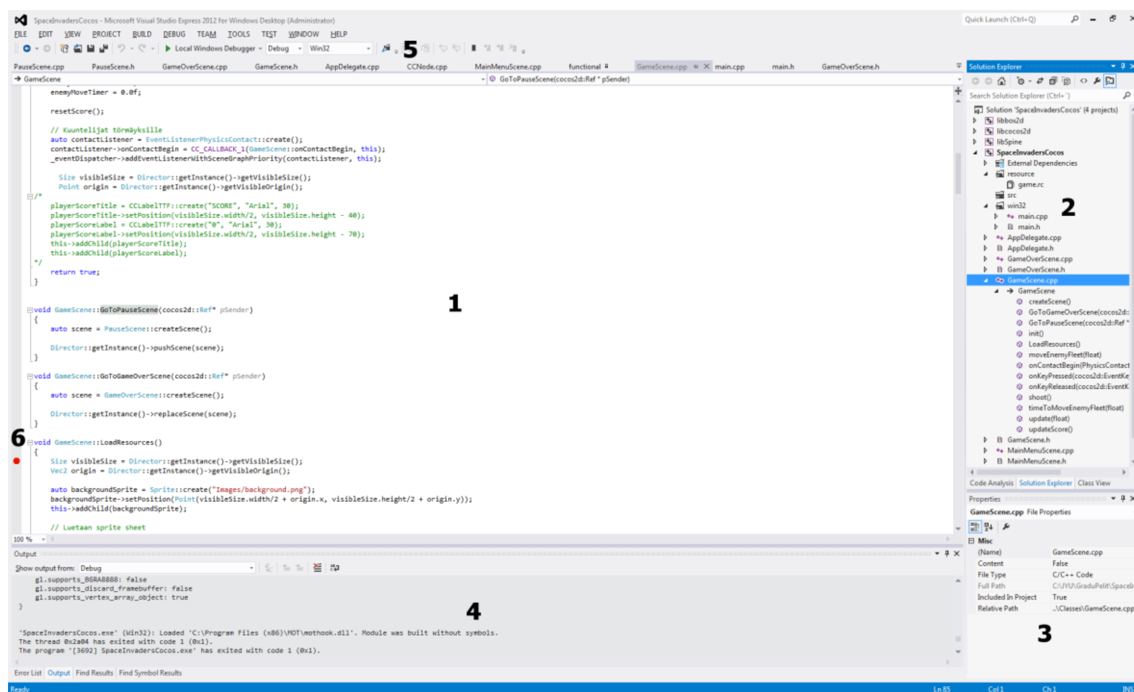


KUVIO 18 Unityn editori

Toisin kuin Unity, Cocos2d-x ei tarjoa C++ kehittäjälle editoria, ja käyttäjä voikin käyttää haluamaansa koodieditoria, kuten Visual Studio Expressiä (kuvio 19) tai Eclipseä. Lua ja JavaScript käyttäjille sen sijaan on tarjolla Cocos Code IDE, jolla kehitys, virheenjäljitys ja julkaisu onnistuvat. Cocos2d-x:n apuna voidaan käyttää CocoStudiota, joka on neljään eri tehtävään tarkoitettu työkalu. Siinä on käyttöliittymäeditori, animaatioeditori, datanmuokkaaja ja tasoeditori. CocoStudiota ei kuitenkaan käytetä apuna tässä tutkielmassa.

Visual Studio Expressin koodieditorin näkymät ovat myös muokattavissa käyttäjän mieleiseksi. Näkymät ovat:

1. Koodipaneelissa voi editoida lähdekoodia. Se tarjoaa myös IntelliSensen, joka avustaa ohjelmointikielen lauseiden kirjoituksessa.
2. Solution explorer ja luokkanäkymässä ovat selattavissa solutioniin kuuluvat projektit, niiden resurssit, kuten lähdekooditiedostot ja luokat metodeineen ja ominaisuuksineen.
3. Ominaisuusikkunassa voi hallita kontrollien ominaisuuksien arvoja.
4. Tulostusikkuna näyttää useita kehitysympäristön tulosteita.
5. Työkalupalkki on räätälöitävissä halutunlaiseksi.
6. Indikaattorimarginaalissa ovat indikaattorit virheenjäljityksen katkaisupisteille ja kirjanmerkeille.



KUVIO 19 Cocos2d-x ja Visual Studio Express 2012

Lähdekoodissa eri väreillä (taulukko 4) selkiytetään koodin rakennetta, jolloin kehittäjän on helpompi hahmottaa käsiteltävää kokonaisuutta. Syntaksin väritys on ominaisuus, joka löytyy kaikista koodieditoreista, ja se on muokattavissa

kehittäjän mieltymyksen mukaiseksi. Tämän luvun lähdekoodeissa on käytetty Visual Studion oletusvärejä.

TAULUKKO 4 Lähdekoodin syntaksin väritys

Elementti	Väri	Esimerkki
Varattu sana / Muuttuja	sininen	`float`
Kommentti	vihreä	`// Kommentti`
Merkkijono	punainen	`"SpaceInvaders.png"`
Luokka	turkoosi	`GameScene::init();`
Makro	purppura	`CC_CALLBACK_1`
Muut	musta	`LoadResources();`

5.6 Kehitys vaiheittain

Tässä aliluvussa käydään läpi pelin kehityksen vaiheet kummallakin pelimoottorilla. Jokaisessa vaiheessa tehdään samankaltaiset toteutukset, jotka dokumentoidaan kuvankaappausten ja lähdekoodien avulla.

5.6.1 Alustavat toimet

Videopeleissä resurssien määrä on yleensä melko suuri, ja käytettävät resurssit kannattaakin jakaa käyttötarkoituksen mukaan. Resursseja kutsutaan pelimoottoreissa asseteiksi, ja niitä ovat muun muassa pelissä käytettävät kuvat, äänet, skriptit ja tasomäärittelyt.

Unityssä projektin asseteihin luodaan kansioita helpottamaan niiden käsittelyä (taulukko 5). Käytettävyyden kannalta kannattaa luoda ainakin Prefabs, Scenes, Scripts, Sounds ja Textures kansiot. Unityssä projektin kaikki tiedostot ovat projektihakemiston ja sen alihakemistojen alla loogisesti löydettävissä. Varmuuskopion ottaminen ja projektin siirto muille koneille käy helposti kopioimalla projektihakemisto kokonaisuudessaan.

TAULUKKO 5 Unity – Pelissä käytettävät assetit

Asset	Käyttötarkoitus
Prefabs	Uudelleen käytettävät peliobjektit. Esim. viholliset, luodit jne.
Scenes	Pelin taso (level) tai valikko.
Scripts	Skriptit. Pelin ohjelmoitu toiminnallisuus.
Sounds	Äänet ja musiikki.
Textures	Kuvat. 2D-pelissä spritejä ja taustakuvia.

Cocos2d-x:ssä uuden C++ projektin tiedostot luodaan Python-skriptillä (lähdekoodi 1).

```
cocos new SpaceInvaders -p com.JYU.SpaceInvaders –l cpp -d C:\JYU\SpaceInvaders

cocos        skripti
new          uusi
SpaceInvader projektin nimi
-p           paketin nimi projektille
-l           ohjelmointikieli (cpp=C++, lua=Lua, js=JavaScript)
-d           hakemisto, johon projekti generoidaan
```

LÄHDEKOODI 1 Cocos2d-x – Projektitiedostojen luonti Python-skriptillä

Uuden projektin luonnin jälkeen, helpoin tapa jatkaa pelin kehitystä Cocos2d-x kehitysympäristössä on muokata HelloWorldScene.h ja HelloWorldScene.cpp omaan käyttöön. HelloWorldScene tiedostoja voi käyttää pohjana oikeastaan kaikkiin pelissä tarvittaviin skeneihin. Silloin muutetaan otsikko- ja lähdekooditiedostojen nimet halutuiksi, ja vaihdetaan muutetuista tiedostoista kaikki HelloWorld tekstit uuden skenen nimisiksi. Lisäksi AppDelegate.cpp tiedostoa täytyy muuttaa, niin että ensimmäisenä käynnistettävän skenen otsikkotiedosto sisällytetään siihen, ja HelloWorldScenen tilalla luodaan oma skene (lähdekoodi 2).

```
AppDelegate.cpp

#include "MainMenuScene.h"


// Pelin käynnistyessä, avataan päävalikko
bool AppDelegate::applicationDidFinishLaunching()
{
    // Luo päävalikko skenen
    auto scene = MainMenu::createScene();

    // ja ajaa ja näyttää sen pelin käynnistyessä
    director->runWithScene(scene);

    return true;
}
```

LÄHDEKOODI 2 Cocos2d-x – MenuScenen luonti ja käynnistys

Cocos2d-x:ssä projektihakemiston alla on kohdealustakohtaisesti projektitiedostot omissa hakemistoissaan. Esimerkiksi proj.android- ja proj.win32-hakemistoissa. Lähdekoodit sijaitsevat projektihakemiston Classes-alihakemistossa. Resurssit ovat Resources-alihakemistossa, johon tehdään halutut alihakemistot resurssien tyypin mukaan, kuten esimerkiksi Images ja Sounds.

Pelisilmukat on nykyisin tahdistettu yleensä 60 kertaan sekunnissa eli 60 FPSään, mikä on yleinen näytön virkistystaajuus. Osassa edullisempia Android-laitteita on kuitenkin myös näyttöjä, joiden virkistystaajuus on vain 30 FPS. Ellei tahdistusta ole tehty, peli pyrkii suorittamaan näytölle piirron niin nopeas-

ti kuin laitteisto siihen kykenee, ja peli voi laitteen tehoista riippuen pyöriä joko liian nopeasti tai liian hitaasti. Kehitysvaiheessa on syytä varautua ongelmaan, ja tahdistaa peli johonkin tiettyyn kuvanopeuteen. Se pakottaa pelisilmukan pyörimään maksimissaan tiettyyn tahtiin, mutta siitä huolimatta peli ei välttämättä pyöri riittävän nopeasti. Se voi johtua tehottomasta laitteesta, liian raskaasta laskennasta tai jostain muusta, kuten koneessa taustalla pyörivistä prosesseista. Myös näytönohjaimen Vsync pakottaa tahdistuksen näytön virkistystaajuuteen. Kehitysvaiheessa on siis otettava huomioon myös, ettei pelisilmukka pyöri kaikissa laitteissa samalla nopeudella. Sulavan pelikokemuksen aikaansaamiseksi, pelihahmojen liikkeiden ja animaatioiden laskemisessa onkin otettava huomioon yksittäisten kuvien välillä kulunut aika. Siitä käytetään nimitystä delta time.

Unityssä pelimoottori kutsuu skriptien Update-metodeja pelisilmukan jokaisella kierroksella. Sitä voidaan käyttää muun muassa objektien siirtoihin, ajastuksiin ja käyttäjän syötteen lukuun. Sen lisäksi skripteille voidaan kirjoittaa FixedUpdate metodi, joka on tahdistettu näytön päivitykseen, ja toisin kuin Update, se suoritetaan tasaisin väliajoin. Mikäli fysiikkamoottori on käytössä, FixedUpdatea täytyy käyttää Updaten sijaan, koska fysiikan laskenta käyttää FixedUpdatea.

Cocos2d-x:ssä kuvanopeus on oletuksena 60 FPS. Se voidaan vaihtaa halutuksi muuttamalla AppDelegate::applicationDidFinishLaunching() metodissa riviä `director->setAnimationInterval(1.0 / 60);`. Cocos2d-x pyörii tapahtumapohjaisesti. Mikäli jotain toiminnallisuutta varten halutaan suorittaa tehtäviä perinteisen pelisilmukan tapaan, täytyy sitä varten aktivoida Update metodi (lähdekoodi 3). Se onnistuu kutsumalla scheduleUpdate() metodia skenen alustuksissa, ja kirjoittamalla skenelle metodi update, jota pelimoottori kutsuu pelisilmukassa ennen ruudulle piirtämistä. Se saa parametrinaan edellisestä update kutsusta kuluneen ajan dt:n eli delta timen.

```
bool GameScene::init()
{
  // Ajastaa update metodin GameScenelle
  this->scheduleUpdate();

  return true;
}

// Updatea kutsutaan joka framella
void GameScene::update(float dt)
{

}
```

LÄHDEKOODI 3 Cocos2d-x – Update metodin tahdistuksen asetus

Cocos2d-x:llä projektitiedostot luodaan Python-skriptillä. Molempia pelimoottoreita käytettäessä on syytä jakaa assetit eli pelin resurssit niiden tyypin mu-

kaan omiin alihakemistoihinsa. Unityllä uuden peliprojektin aloittaminen on hieman yksinkertaisempaa kuin Cocos2d-x:llä.

5.6.2 Skriptit

Skriptit Unityssä liitetään aina johonkin peliobjektiin. Se voi olla myös kamera tai pelin taso. Yhdellä peliobjektilla voi olla komponentteina useita skriptejä, joilla sen toiminnallisuus toteutetaan. Skriptit voidaan myös liittää useille eri peliobjekteille. Mikäli skriptit ovat aktiivisia, pelimoottori kutsuu niiden metodeja tietyissä paikoissa. Kun uusi skripti luodaan, sillä on valmiina Start ja Update metodit (lähdekoodi 4). Start-metodi suoritetaan objektin luonnin yhteydessä, mikäli objekti on aktivoituna. Kaikkien aktiivisten skriptien Update suoritetaan pelisilmukan jokaisella kierroksella.

```
using UnityEngine;
using System.Collections;

public class Test : MonoBehaviour
 {
    // Use this for initialization
    void Start () {

    }

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

    }
}
```

LÄHDEKOODI 4 Unity – C#-skriptin pohja

Edellisten lisäksi skripteissä usein käytettyjä metodeja (taulukko 6) ovat FixedUpdate, joka on tahdistettu ruudunpäivitykseen, ja sitä tulee käyttää Updaten sijaan fysiikkamoottoria käytettäessä, Awake, joka suoritetaan heti siinä vaiheessa kun peliobjekti luodaan, ja Destroy, joka suoritetaan kun objekti tuhotaan.

TAULUKKO 6 Unity – Skriptien tavallisimmat metodit

Metodi	
Awake()	Suoritetaan kun objekti luodaan.
Start()	Suoritetaan Awaken jälkeen. Erona Awakeen on, että startia kutsutaan vain jos skripti on enabloituna.
Update()	Suoritetaan pelisilmukan jokaisella kierroksella.
FixedUpdate()	Suoritetaan kuvataajuuden perusteella tasaisin väliajoin.
Destroy()	Suoritetaan kun objekti tuhotaan.

Skriptejä käytetään Unityssä pelin toiminnallisuuden ohjelmointiin. Tässä tutkielmassa ei Cocos2d-x pelin toiminnallisuutta ohjelmoida skripteillä lainkaan.

5.6.3 Resurssien lataus

Unityssä käytettävät resurssit kopioidaan tiedoston käyttötarkoituksen mukaan omiin hakemistoihinsa projektin Assets kansion alle, jonka jälkeen ne ovat käytettävissä suoraan editorista. Unityssä on myös suora linkki kauppapaikkaan, Asset Storeen, josta voi hankkia ilmaiseksi tai pienestä maksusta erilaisia pelissä käytettäviä asseteja.

Cocos2d-x:n resurssitiedostot ovat projektihakemistossa Resourses alihakemiston alla. Cocos2d-x:ssä resurssit joudutaan lataamaan ennen käyttöä. Esimerkkinä taustakuvan lataus spriteksi (lähdekoodi 5).

```
auto backgroundSprite = Sprite::create("Images/background.png");
```

LÄHDEKOODI 5 Cocos2d-x – Spritejen lataus kuvatiedostosta

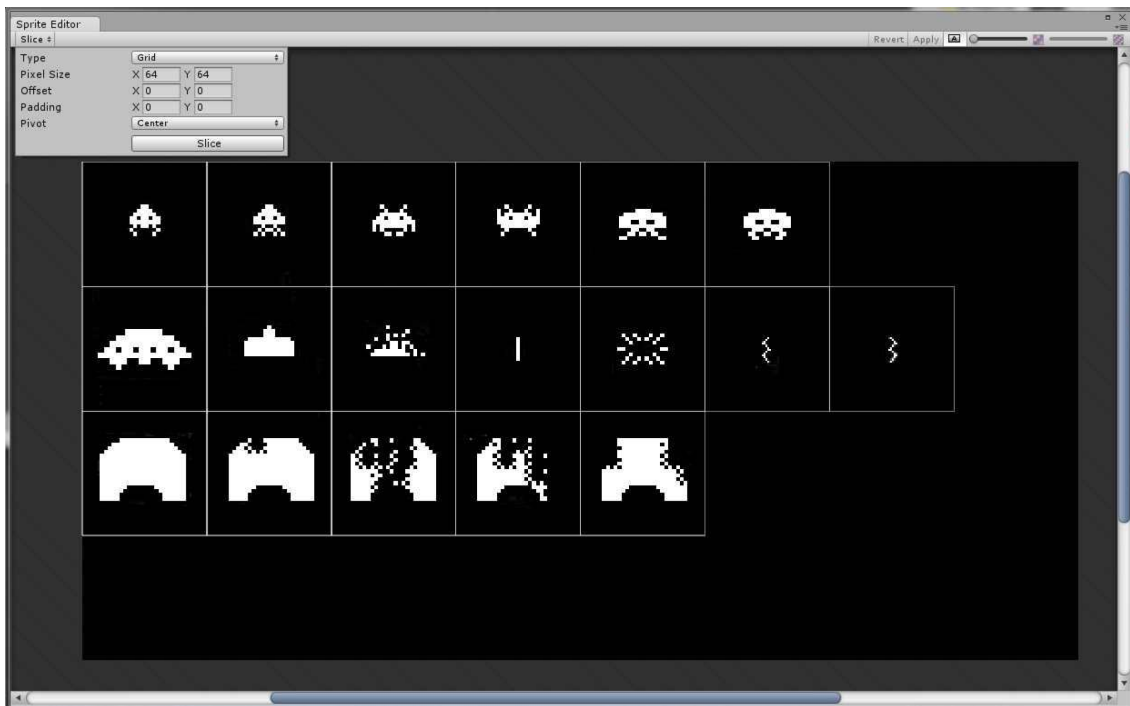
Resurssien käsittelyt on tehty käyttäjän kannalta kummassakin pelimoottorissa helpoiksi.

5.6.4 Grafiikka ja animaatio

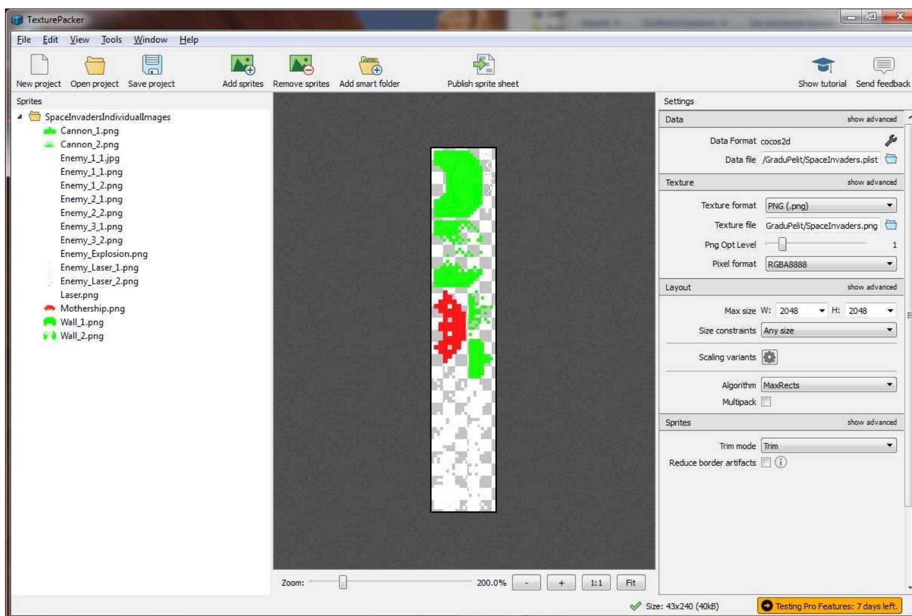
Kuten edellisessä aliluvussa mainittiin, Unityssä kuvatiedostot kopioidaan sellaisenaan projektin resurssien alihakemistoon Textures, jolloin ne näkyvät projektin asseteissa. 2D-pelissä kuvia kutsutaan spriteiksi. Kuvat voidaan käyttää pelissä näyttämällä ne kokonaisina, kuten taustakuva, tai näyttämällä ainoastaan osia suuremmasta kuvasta. Useampia pienempiä kuvia eli spritejä sisältävistä kuvista käytetään nimitystä sprite sheet. Sprite sheet kuvien sprite moodiksi täytyy valita singlen sijaan multiple, jonka jälkeen kuva voidaan pilkkoa pienempiin osiin. Unityssä on tehtävään soveltuva työkalu nimeltään Sprite Editor (kuvio 21). Unityssä kuvatiedostot ovat Textures kansion alla.

Kolmannen osapuolen Texture Packer (kuvio 22) työkalulla sprite sheetien teko on helppoa. Yksittäisiä kuvia raahataan ohjelmassa uuteen sprite sheetiin, joka asettelee ne vierekkäin uuteen kuvaan (kuvio 20). Kun halutut kuvat on lisätty, julkaistaan se uutena sprite sheetinä ja sen yksittäisten kuvien tiedot sisältävänä data tiedostona. Ohjelma osaa tehdä useiden pelimoottoreiden kanssa yhteensopivia data tiedostoja, mukaan lukien Cocos2d-x:n plistin ja Unityn tpsheetin.

KUVIO 20 Texture Packerilla luotu sprite sheet



KUVIO 21 Unity – Sprite Editor



KUVIO 22 Texture Packer

Cocos2d-x:ssä sprite sheetit ladataan SpriteBatchNodeihin ja sprite sheetien yksityiskohdat sisältävät datatiedostot SpriteFrameCacheihin (lähdekoodi 6). SpriteBatch lisätään skenelle, ja siitä luodut spritet ovat käytettävissä kuten yksittäisistä kuvistakin luodut spritet. SpriteBatchNodea käytettäessä on myös suori-

tuskykyetu, jossa kaikki samaan SpriteBatchNodeen kuuluvat spritet piirretään yhdellä OpenGL kutsulla, eli niin sanotulla batch drawlla.

```
// Luetaan sprite sheet ja sen .plist datatiedosto
SpriteBatchNode* spriteBatch = SpriteBatchNode::create("SpaceInvaders.png");
SpriteFrameCache* cache = SpriteFrameCache::getInstance();
cache->addSpriteFramesWithFile("SpaceInvaders.plist");

// Yhden muukalaisspriten luonti spritesheetistä nimen perusteella
auto spriteEnemy = Sprite::createWithSpriteFrameName("Enemy_1_1.png");
// Asetetaan keskelle ruutua
spriteEnemy1->setPosition(Point(visibleSize.width/2 + origin.x,
                                visibleSize.height/2 + origin.y));

// Muutetaan muukalaisen väri punaiseksi
spriteEnemy->setColor(Color3B::RED);
// Asetetaan spritelle nimi ”Enemy”. Tietoa käytetään mm. tutkittaessa osumia.
spriteEnemy->setName("Enemy");

// Lisää spriten vihollisvektoriin, jossa on kaikki kentässä olevat muukalaiset
enemies.push_back(spriteEnemy);
// Lisää sprite spritebachiin
spriteBatch->addChild(spriteEnemy);
```

LÄHDEKOODI 6 Cocos2d-x – Sprite sheetin luku ja siitä spriten luonti

Näyttöjen resoluutiot ja kuvasuhteet vaihtelevat jonkin verran. Kehitysvaiheessa ei voida tietää, minkälaisilla laitteilla peliä tullaan pelaamaan. Kehittäjä voi varautua ongelmaan eri tavoin.

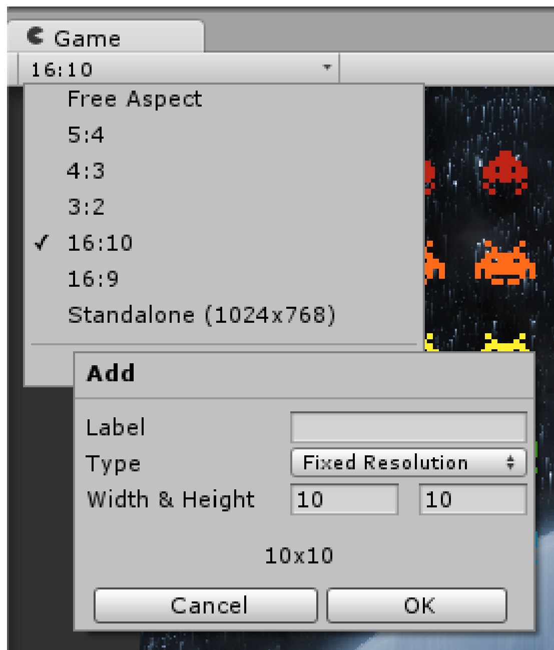
Pelin grafiikoista voidaan tehdä useampi versio, joissa kuvien resoluutiot ovat eri tarkkuudella, ja tallennetaan ne eri hakemistoihin. Pelin käynnistyessä tutkitaan näytön resoluutio, ja sen perusteella valitaan hakemisto, jonka kuvatiedostoja pelissä käytetään.

Cocos2d-x:ssä on mahdollista asettaa suunnitteluresoluutio (lähdekoodi 7). Tällöin näytön resoluutio on kehittäjän kannalta aina vakio, riippumatta käyttävän laitteen todellisesta näytön resoluutiosta. Tietoa voidaan hyödyntää sijoitettaessa spritejä pelikentälle, ja käyttää absoluuttisia koordinaatteja. Asetusmetodi saa kolmantena parametrinaan ohjeen, miten pelikuva näytetään näytöllä. Esimerkiksi näytön todellisen kuvasuhteen poiketessa suunnittelussa käytetystä kuvasuhteesta, voidaan näytön reunat näyttää mustana, jolloin pelin kuvasuhde säilyy kaikilla laitteilla samana. Vastaavasti näytön todellisen resoluution ollessa suurempi kuin suunnitteluresoluutio, pelikuva voidaan venyttää näkymään koko ruudulla.

```
// Asettaa suunnitteluresoluution
glview->setDesignResolutionSize(1024, 768, ResolutionPolicy::SHOW_ALL);
```

LÄHDEKOODI 7 Cocos2d-x – Suunnitteluresoluution asetus

Unityssä pelinäkymän yläkulmasta voi valita käytettävän kuvasuhteen (kuvio 23). Peli skaalautuu sen mukaan eri näytöille. Vaihtoehtoisesti peliin voi asettaa kiinteän resoluution. Kehitysvaiheessa pelissä voidaan käyttää haluttua koordinaatistoa, joka vastaa näytön resoluutiota.



KUVIO 23 Unity – Pelin kuvasuhteen valinta

Unityssä on sprite-editori, jolla pelin grafiikat saadaan käyttöön sprite sheeteistä ja muista kuvatiedostoista graafisen käyttöliittymän avulla. Cocos2d-x:ssä voidaan lukea yksittäisiä kuvia tai sprite sheetejä, joissa on määrittelytiedosto mukana. Kummallakin välineellä kuvatiedostojen lataaminen ja grafiikan käyttöönotto pelissä on helppoa.

Molemmissa pelimoottoreissa on myös yksinkertainen tapa, jolla voidaan varautua kohdelaitteiden erilaisiin resoluutioihin ja kuvasuhteisiin.

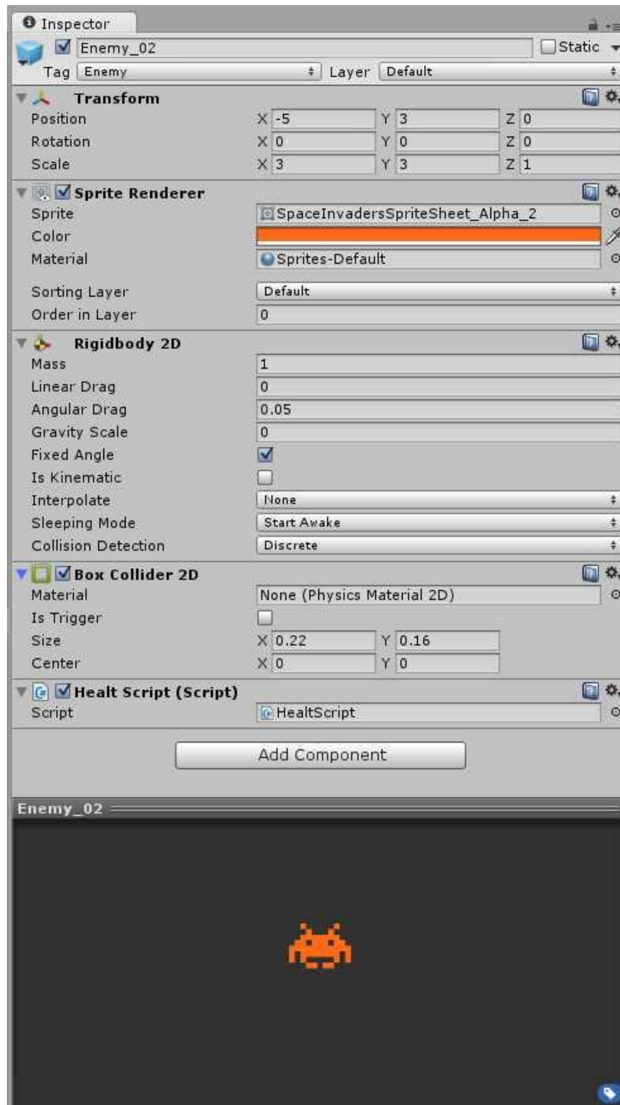
5.6.5 Peliobjektien ja pelimaailman määrittely

Peliobjektien luominen ja ominaisuuksien määrittely voidaan toteuttaa Unityssä lähes kokonaan graafisen käyttöliittymän avulla (kuvio 24). Peliobjektilla on oletuksena Transform-komponentti, joka sisältää tiedon peliobjektin sijainnista, rotaatiosta ja skaalauksesta.

Peliobjekti saadaan näkymään pelissä liittämällä siihen Sprite Renderer -komponentti. Komponentille voidaan valita näytettävä sprite, jonka se piirtää ruudulle. Spritejen värisävyjä pystytään muuttamaan suoraan komponentin ominaisuuksista, ja Space Invadersin yksiväristen valkoisten hahmojen värin vaihto onnistuu helposti valitsemalla haluttu RGB väri.

Uudelleen käytettävät peliobjektit tehdään Prefabs Assettien alle. Silloin niistä voidaan luoda pelissä rajattomasti ilmentymiä, eikä jokaista uutta objek-

tia tarvitse määritellä aina uudestaan. Sellaisia ovat esimerkiksi viholliset ja luodit. Pelikenttä voidaankin määritellä raahaamalla peliobjekteja Prefabseista suoraan Scene ikkunassa olevalle pelikentälle.



KUVIO 24 Unity – Peliobjektin komponentit

Yksinkertaisessa pelissä kuten Space Invadersissa voidaan käyttää Cocos2d-x:n spriteä peliobjektina. Normaalisti kannattaa kuitenkin tehdä peliobjektille oma luokka, josta pelihahmojen ilmentymät luodaan.

Space Invadersin peliobjekteina ovat pelaajan ohjaama lasertykki, vihollislaivaston muukalaiset, suojamuurit, ammukset ja vihollisten emoalus.

Peliobjekteille annetaan myös ominaisuutena osumien kesto, joka on vihollisille yksi, mutta muureille kaksi, jolloin ne ainoastaan särkyvät ensimmäisestä osumasta, ja tuhoutuvat toisesta. Särkyminen toteutetaan vaihtamalla ensimmäisen osuman jälkeen kuva sortuneesta muurista, ja vähentämällä muurin osumapisteitä.

5.6.6 Pelaajan syötteen käsittely

Pelaaja liikuttaa lasertykkiä pelikentällä sivusuunnassa, ja tulittaa vihollisia. Ainoastaan yksi pelaajan ammus voi olla kerrallaan ruudulla.

Pelaajan syötteen käsittelyä varten Unityssä liitetään lasertykille skripti (lähdekoodi 8), joka lukee pelisilmukan jokaisella kierroksella pelaajan näppäinten painallukset tai kosketusnäytöllä sormenliikkeet ja näpäytykset. Niiden perusteella tykkiä siirretään sivusuunnassa ja se laukaistaan. Tykin liikkeen hoitaa fysiikkamoottori, kun tykin rigidbody2D-komponentille annetaan nopeus-vektori. Vektori sisältää tiedon siirrettävän matkan pituudesta ja suunnasta. Koska muut voimat eivät tässä pelissä vaikuta siirrettävään objektiin, fysiikkamoottori siirtää tykkiä asetetun nopeuden verran sekunnissa. Fysiikkamoottorille siirrettävien tehtävien, kuten kappaleiden liikuttamisen, vuoksi nopeuden syöttö rigidbody2D-komponentille täytyy tehdä FixedUpdate() metodissa.

Unityssä virtuaaliohjaimen liikkeet luetaan Input.GetAxis-metodilla. Virtuaaliohjain voi olla näppäimistö tai peliohjain, joiden akselit ja napit on nimetty yhtenäisesti ohjelmoinnin helpottamiseksi. Luettaessa virtuaaliohjainten liikkeitä, sivusuunnassa tapahtuva liike saadaan parametrilla *Horizontal* ja pystysuuntainen parametrilla *Vertical* eli esimerkiksi Input.GetAxis(”Horizontal”). Vastaavat kutsut hiirtä käytettäessä ovat Input.GetAxis(”Mouse X”) ja Input.GetAxis(”Mouse Y”). Napin painallukset luetaan Input.GetButton(”Fire#”)-metodilla.

Kosketusnäytöllä kontrollien toteuttaminen on mahdollista siten kuin kehittäjät haluavat. On kuitenkin otettava huomioon kosketusnäytön ominaisuudet. Esimerkiksi samanaikaisten kosketusten määrä voi vaihdella. Tässä pelissä tykkiä liikutetaan sivusuunnassa siirtämällä sormea näytön vasemmalla puoliskolla, ja tulitetaan näpäyttämällä ruutua näytön oikealla puoliskolla. Se on toteutettu käymällä läpi kaikki näytön kosketukset, ja lukemalla niiden sijainnit. Mikäli kosketus on näytön vasemmalla puolella, ja sormea on liikutettu sivusuunnassa, siirretään tykkiä. Jos taas kosketus on näytön oikealla puolella, tulitetaan.

Cocos2d-x:ssä käyttäjän syötteet aiheuttavat tapahtumia. Sen vuoksi niille täytyy kirjoittaa pelinäkymään kuuntelijat ja käsittelijät (lähdekoodi 9). Näppäimistöä käytettäessä riittää tieto tietyn näppäimen pohjaan painamisesta, ja sen jälleen vapauttamisesta.

Syötteiden käsittelyssä merkitään pelaajan ohjausnappi painetuksi pohjaan tai ei, sen perusteella kumman käsittelijän tapahtuma laukaisi. Välilyönnin painallus aiheuttaa tulitusmetodin kutsumisen.

```
// Fixed Updatea kutsutaan vakio välein, esim. 60 kertaa sekunnissa
void FixedUpdate()
{
  float inputX = 0.0f;
  bool shoot = false;

  // HAETAAN PELAAJAN SYÖTE
  // Android - Kosketusnäyttö
  #if UNITY_ANDROID
    // Käy kaikki kosketukset läpi
    foreach (Touch touch in Input.touches)
    {
      // Kosketusnäytön vasemmassa laidassa kosketus
      if (touch.position.x < (float)Screen.width / 2.0f)
      {
        // Kosketus liikkuu sivusuunnassa
        if (touch.phase == TouchPhase.Moved)
        {
          Vector2 touchDeltaPosition = touch.deltaPosition;
          inputX = touchDeltaPosition.x;
        }
        // Kosketus päättyy
        else if (touch.phase == TouchPhase.Ended)
        {
          inputX = 0.0f;
        }
      }
      else // Kosketusnäytön oikeassa laidassa kosketus
      {
        // Tulituskäsky
        shoot = true;
      }
    }
  // PC - Näppäimistö
  #else
    // Nuolinäppäimillä painettu vasen/oikea
    inputX = Input.GetAxis ("Horizontal");

    // Tulitus - Painettu 1. tai 2. tulitusnäppäintä
    shoot = Input.GetButtonDown("Fire1");
    shoot |= Input.GetButtonDown("Fire2");
  #endif

  if (shoot)
  {
    // Tulituksen käsittely siirretään WeaponScriptille
    WeaponScript weapon = GetComponent<WeaponScript>();
    if (weapon != null)
    {
      weapon.Shoot();
    }
  }

  // Lasketaan tykin liike ja annetaan sen käsittely fysiikkamoottorin hoidettavaksi
  cannonMovement = new Vector2(cannonSpeed.x * inputX, 0);
  rigidbody2D.velocity = cannonMovement;
}
```

LÄHDEKOODI 8 Unity – Pelaajan syötteen luku ja lasertykin siirto

GameScene.h

```
// Näppäimistön kuuntelija
void onKeyPressed(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event);
void onKeyReleased(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event);
```

GameScene.cpp

```
bool GameScene::init()
{
  // Lisätään näppäimistön kuuntelijat
  auto keyboardListener = cocos2d::EventListenerKeyboard::create();
  keyboardListener->onKeyPressed = CC_CALLBACK_2(GameScene::onKeyPressed, this);
  keyboardListener->onKeyReleased = CC_CALLBACK_2(GameScene::onKeyReleased, this);

  _eventDispatcher->addEventListenerWithSceneGraphPriority(keyboardListener, this);

    return true;
}

// Näppäintä painettu
void GameScene::onKeyPressed(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event)
{
  if (keyCode == EventKeyboard::KeyCode::KEY_SPACE)
    shoot();
  if (keyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW)
    isPressedLeft = true;
  if (keyCode == EventKeyboard::KeyCode::KEY_RIGHT_ARROW)
    isPressedRight = true;
}

// Näppäin vapautettu
void GameScene::onKeyReleased(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event)
{
  if (keyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW)
    isPressedLeft = false;
  if (keyCode == EventKeyboard::KeyCode::KEY_RIGHT_ARROW)
    isPressedRight = false;
}
```

LÄHDEKOODI 9 Cocos2d-x – Pelaajan syötteen kuuntelijat

Unityssä on hieman yksinkertaisempi tapa lukea käyttäjän syötteet kuin Cocos2d-x:ssä. Unityssä syöte voidaan lukea joka kierroksella Update-metodissa, kun taas Cocos2d-x:ssä käyttäjän syötteestä syntyy tapahtumia, joille on kirjoitettava käsittelijät.

5.6.7 Peliobjektien liikuttaminen

Pelintekijä voi itse liikuttaa pelihahmoja pelissä paikasta toiseen tai antaa fysiikkamoottorin hoitaa siirrot. Tämän tutkielman peleissä testattiin molempia tapoja. Cocos2d-x:llä siirrettiin spritejä suoraan (lähdekoodi 10), ja Unityllä annettiin peliobjektien siirtäminen tiettyyn paikkaan fysiikkamoottorin tehtäväksi.

Fysiikkamoottoria käytettäessä tulisi fysikaalisten voimien antaa aiheuttaa peliobjektien siirrot, tai lopputuloksena voi olla outoja tai kokonaan havaitsematta jääviä törmäyksiä.

Cocos2d-x käyttää Action-olioita spritejen liikuttamiseen (lähdekoodi 10).

```
// MoveBy siirtää spriteä kahdessa sekunnissa 20 pikseliä oikealle
auto moveBy = MoveBy::create(2, Vec2(20, 0));
sprite->runAction(moveBy);

// MoveTo siirtää spriten kolmessa sekunnissa koordinaatteihin 500,180
auto moveTo = MoveTo::create(3, Vec2(500, 180));
sprite->runAction(moveBy);
```

LÄHDEKOODI 10 Cocos2d-x – Spriten siirto

Action-olioilla spriteä voidaan muun muassa siirtää, kääntää, skaalata ja näyttää se animaationa (taulukko 7). Erilaisia toimintoja voidaan myös ketjuttaa spritelle ajettavaksi peräkkäin tai samanaikaisesti.

TAULUKKO 7 Cocos2d-x – Action olion perustoiminnot

Toiminto	Selitys
Move	Siirto
Rotate	Kääntö
Scale	Skaalaus
Fade In/Out	Muuttaa näkyvästä näkymättömäksi ja päinvastoin
Tint	Muuttaa RGB värisävyä
Animate	Animaatio. Voi tehdä ohjelmallisesti useista yksittäisistä kuvista
Easing	Erilaisia kiihtyvyyden muutoksia pehmeämmän liikkeen aikaansaamiseksi

Cocos2d-x versiossa fysiikkamoottoria ei ole käytetty, vaan tykki siirtyy ruudulla, mikäli pelaaja on painanut nuolinäppäimiä, ja ampuu jos pelaaja on painanut välilyöntiä (lähdekoodi 11).

Pelissä vihollislaivasto siirtyy kiihtyvällä vauhdilla kohti pelaajaa. Laivastoa siirretään, mikäli pääsilmukassa todetaan olevan siirron aika. Vihollisen spritet on sijoitettu tasaisin välein riveille ja sarakkeisiin.

Unityssä pelikentän tasolle itselleen liitetään skripti (lähdekoodi 12), joka käy Enemy tagilla varustetut peliobjektit läpi, ja siirtää niitä kaikkia siirron aikana samaan suuntaan.

```
void GameScene::update(float dt)
{
  // Pelaaja painanut ohjainta vasemmalle tai oikealle
  if (isPressedLeft || isPressedRight)
  {
    float x = 0.0f;
    float y = 0.0f;

    // Riippuen suunnasta, asettaa sivusuunnassa siirron määrän
    if (isPressedLeft)
      x = -cannonSpeed;
    if (isPressedRight)
      x = cannonSpeed;

    // Action objektia käytetään muiden peliobjektien siirtoon
    auto action = MoveBy::create(0.0f, Point(x, y));
    laserCannon->runAction(action);
  }

  Size visibleSize = Director::getInstance()->getVisibleSize();

  // Ainoastaan yksi laser voi olla ruudulla kerrallaan,
  // sitä siirretään vain jos se on näkyvissä
  if (laser->isVisible())
  {
    // Ruudun ulkopuolella -> muutetaan näkymättömäksi -> tykillä voi ampua taas uuden
    if (laser->getPositionY() > visibleSize.height)
      laser->setVisible(false);
    else // Muuten tehdään normaali siirto
    {
      auto action = MoveBy::create(0.0f, Point(0.0f, 20.0f));
      laser->runAction(action);
    }
  }

  if (timeToMoveEnemyFleet(dt))
    moveEnemyFleet(dt);
}
```

LÄHDEKOODI 11 Cocos2d-x – Pelaajan tykin kontrollointi

```
// Huom! Ajastimeen liittyvien metodien rungot jätetty pois selkiyttämään isompaa kuvaa
public class EnemyAttack : MonoBehaviour {
  // Viholliset
  GameObject[] enemies;
  // Siirtymä
  public Vector2 movement;

  // Skripti luodaan
  void Start () {
    ResetEnemyMoveTimer();
  }

  // Updatea kutsutaan jokaisella framella
  void Update () {
    // Päivitetään ajastinta eli vähennetään edellisen ja tämän kierroksen välinen aika
    UpdateEnemyMoveTimer();
  }
```

jatkuu seuraavalla sivulla…

…jatkoa edelliseltä sivulta

```
  void FixedUpdate() {
    // Aika siirtää vihollisia?
    if (TimeToMoveEnemies())
      MoveEnemies();
  }

  // Palauttaa sijainnin johon vihollinen siirretään
  private Vector2 MoveToPosition(Vector2 currentPosition)
  { // Poistettu koodia selkeyden vuoksi
    // Siirtää vihollista joko sivusuunnassa sarakkeen verran tai
    // jos vihollislaivasto reunassa, niin pystysuunnassa rivin verran
    return newPosition;
  }

  // Siirtää koko vihollislaivastoa
  void MoveEnemies()
  {
    // Hakee taulukkoon kaikki Enemy-tagilla varustetut peliobjektit
    enemies = GameObject.FindGameObjectsWithTag("Enemy");

    // Käy läpi kaikki viholliset ja siirtää niitä kentällä
    foreach(GameObject enemy in enemies)
    {
      Rigidbody2D rigidbody2d = enemy.GetComponentInChildren<Rigidbody2D>();
      Vector2 newPos = MoveToPosition(rigidbody2d.position);

      rigidbody2d.MovePosition(newPos);
    }

    // Päivitetään timer
    ResetEnemyMoveTimer();
  }
}
```

LÄHDEKOODI 12 Unity – Vihollislaivaston siirtoskripti

Cocos2d-x versiossa siirretään vihollislaivastoa update-metodissa (lähdekoodi 13). Siirtojen väliset ajat lyhenevät alkuperäiselle Space Invadersille uskollisesti aina, kun vihollisia on siirretty yhtä riviä lähemmäs pelaajaa, tai pelaaja ampuu vihollisen.

```
void GameScene::update(float dt)
{
  if (timeToMoveEnemyFleet(dt))
    moveEnemyFleet(dt);
}
```

jatkuu seuraavalla sivulla…

```
…jatkoa edelliseltä sivulta
// Palauttaa tiedon, onko vihollislaivastoa aika siirtää
bool GameScene::timeToMoveEnemyFleet(float dt)
{
  // Vähentää edellisestä kerrasta kuluneen ajan. Toimii timerina.
  enemyMoveTimer -= dt;

  // Aikaa on kulunut vihollisten siirtotaajuden verran
  if (enemyMoveTimer < 0.0f)
  {
    // Asetetaan ajastin
    enemyMoveTimer = enemyMoveRate;
    return true;
  }
  else    return false;
}

// Siirtää koko vihollislaivastoa
void GameScene::moveEnemyFleet(float dt)
{
  static int enemyColumn = 0;
  static int direction = 1; // -1=vasen, 0=alas, 1=oikea
  float x = 0.0; float y = 0.0;

  // Vihollislaivasto reunassa -> siirretään rivillä alas
  if (enemyColumn < 0 || enemyColumn > 11)
  {
    // Ollaan siirtämässä sivusuunnassa
    if (direction)
    {
      direction = 0;
      y = -10.0f;
    }
    // Siirrettiin alas -> vaihdetaan suunta
    else
    {
      if (enemyColumn < 0)
        direction = 1;
      else
        direction = -1;
    }
  }

  // Muuten siirretään sarakkeen verran sivusuunnassa
  enemyColumn += direction;
  x = (float)direction * 50;

  for (std::vector<Sprite*>::iterator it = enemies.begin(); it != enemies.end(); ++it)
  {
    auto action = MoveBy::create(0.0f, Point(x, y));
    (Sprite*)(*it)->runAction(action);
  }
}
```

LÄHDEKOODI 13 Coco2d-x – Vihollislaivaston siirto

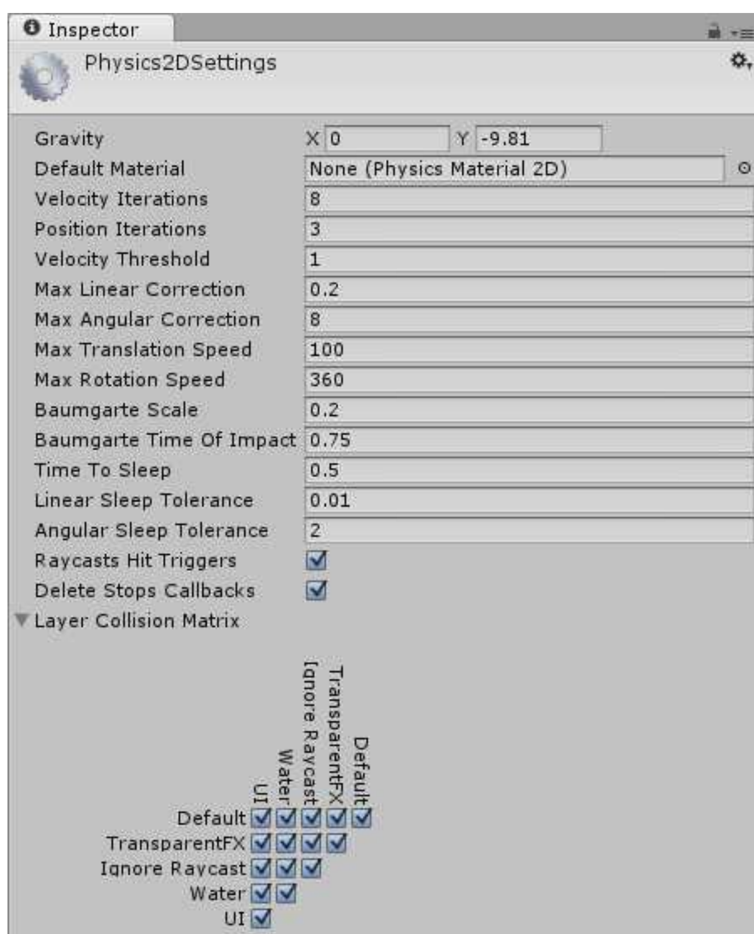
Peliobjektien liikuttaminen on 2D-peleissä Cocos2d-x:llä yksinkertaisempaa kuin Unityllä.

5.6.8 Törmäystarkistukset ja käsittely

Molemmissa pelimoottoreissa voidaan käyttää törmäysten tarkistuksia, mutta niitä varten on käytettävä fysiikkamoottoria, vaikka fysiikan mallinnusta ei muuten pelissä käytettäisikään. Fysiikkamottoreina Unity käyttää Box2Dtä ja Cocos2d-x oletuksena Chipmunk engineä, mutta Cocos2d-x:ssä voi käyttää vaihtoehtoisesti myös Box2Dtä.

Fysiikkojen mallinnusta käytettäessä pelimaailman olosuhteita voi muuttaa (kuvio 25). Esimerkiksi gravitaatiota voi käyttää vastaamaan planeetan todellista gravitaatiota, jolloin esineet putoavat aidon tuntuisesti. Peliobjektien massojen ja materiaalien muuttaminen muuttaa niiden käyttäytymistä muiden peliobjektien ja maailman kanssa. Tästä esimerkkinä tien pinta, joka voi olla jäinen tai kuiva. Riippuen tien pinnasta, kitkan määrä on erilainen, ja pelaajan auto voi sen perusteella joko luisua tai pysyä hallinnassa mutkaan kovaa ajettaessa.

Unityssä ei tarvitse erikseen ottaa fysiikoita käyttöön, vaan ne ovat valmiina kunhan peliobjekteille lisää Rigidbody2D-komponentin. Sen jälkeen peliobjektit, joilla on Rigidbody2D-komponentti, reagoivat virtuaalimaailman voimiin.



KUVIO 25 Unity – Fysiikkamoottorin asetukset

Cocos2d-x:ssä peliskene luodaan normaalin createn sijaan createWithPhysicsillä (lähdekoodi 14). Layerille annetaan skenen physicsWorld, jolloin se käyttää samoja fysiikkaominaisuuksia kuin skenekin, eli tässä tapauksessa pelimaailmassa eivät vaikuta mitkään voimat, koska gravitaatio on asetettu nollaan.

GameScene.h

```
class GameScene : public cocos2d::Layer
{
public:
  // Fysiikkojen asetus pelimaailmalle
  void setPhysicsWorld(cocos2d::PhysicsWorld* world)
  {
    _world = world;
    _world->setGravity(cocos2d::Vect(0, 0));
  }

private:
  cocos2d::PhysicsWorld* _world;
};
```

GameScene.cpp

```
Scene* GameScene::createScene()
{
  // Skene luodaan fysiikkamoottorin kanssa
  auto scene = Scene::createWithPhysics();

  // Skenen tasolle annetaan skenen kanssa samat fysiikkaominaisuudet
  auto layer = GameScene::create();
  layer->setPhysicsWorld(scene->getPhysicsWorld());

  // Lisää tason skenelle
  scene->addChild(layer);

  // Palauttaa skenen
  return scene;
}
```

LÄHDEKOODI 14 Cocos2d-x – Fysiikat pelimaailmalle

Peliskenelle täytyy myös luoda törmäyskuuntelijat (lähdekoodi 15). Halutut kuuntelijat luodaan skenen alustuksilla, jonka lisäksi kirjoitetaan onContactBegin metodi, jota fysiikkamoottori kutsuu silloin kun kosketus kahden peliobjektin välillä alkaa. Muita tarpeellisia kuuntelijoita objektien välisten törmäysten käsittelyyn ovat onContactPreSolve, onContactPostSolve ja onContactSeperate;

GameScene.cpp

```
bool GameScene::init()
{
  // Luo kuuntelijat törmäyksille
  auto contactListener = EventListenerPhysicsContact::create();
  contactListener->onContactBegin = CC_CALLBACK_1(GameScene::onContactBegin, this);
  _eventDispatcher->addEventListenerWithSceneGraphPriority(contactListener, this);

  return true;
}

// Törmäys alkaa
bool GameScene::onContactBegin(PhysicsContact& contact)
{

  return false;
}
```

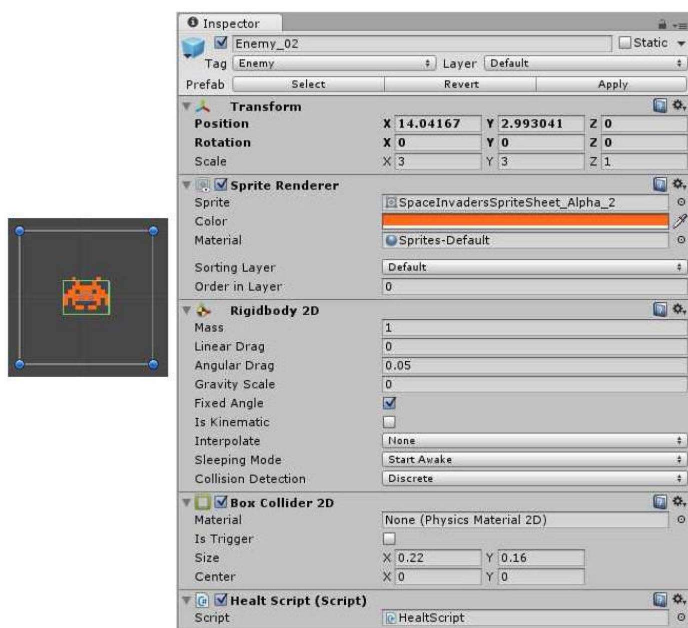
LÄHDEKOODI 15 Cocos2d-x – Törmäysten kuuntelijoiden määrittely

Spritelle voidaan Cocos2d-x:ssä antaa fyysinen kappaleen muoto, joka voi olla ympyrä, suorakaide tai monikulmio (lähdekoodi 16). Fyysisen kappaleen kokoa ja muotoa käytetään törmäysten tutkimisessa toisten kappaleiden kanssa.

```
// Luo spritelle fyysisen ”muodon”, joka on tässä saman kokoinen kuin spriten kuva.
// Sitä käytetään törmäysten tarkistuksissa
auto body = PhysicsBody::createBox(cocos2d::Size(laserCannon->getContentSize().width,
                                                 laserCannon->getContentSize().height));

body->setContactTestBitmask(0x01);
body->setDynamic(true);
laserCannon->setPhysicsBody(body);

this->addChild(laserCannon);
```

LÄHDEKOODI 16 Cocos2d-x – Spritelle kappaleen fyysisen muodon asetus

Unityssä peliobjektille lisätään Rigidbody2D ja Box Collider2D-komponentit. Box Collider 2Dn ominaisuuksista määritellään osumalaatikko (kuvio 26), jota käytetään törmäysten tutkimisessa. Jos ei haluta fysiikkamoottorin hoitavan kappaleiden välisten törmäysten käsittelyä, kuten kimmotuksia, niin Box Collider 2Dn Is Trigger valinta poistaa kappaleen reagoinnin muiden fyysisten kappaleiden kanssa. Törmäyksistä saadaan kuitenkin tieto OnTriggerXXX tapahtumista (taulukko 8), ja niihin voidaan reagoida itse skripteissä, esimerkiksi törmäyksen sattuessa räjäyttämällä peliobjektit.



KUVIO 26 Unity – Vihollisen hitboxin määritys

TAULUKKO 8 Unity – Törmäysten aiheuttamat tapahtumat

Tapahtuma	Aiheuttaja
OnCollisionEnter2D	Lähetetään kun toisen objektin kanssa kosketus alkaa
OnCollisionExit2D	Lähetetään kun toisen objektin kanssa kosketus päättyy
OnCollisionStay2D	Lähetetään jokaisella framella, kun toinen objekti koskettaa
OnTriggerEnter2D	Lähetetään kun toinen objekti aloittaa kosketuksen triggeriksi merkityn colliderin kanssa
OnTriggerExit2D	Lähetetään kun toinen objekti päättää kosketuksen triggeriksi merkityn colliderin kanssa
OnTriggerStay2D	Lähetetään jokaisella framella kun toinen objekti koskettaa triggeriksi merkityn colliderin kanssa

Cocos2d-x pelissä kahden spriten välinen törmäys aiheuttaa onContactBegin metodin kutsun (lähdekoodi 17). Spriteille niiden luonnin yhteydessä annettujen nimien perusteella selviävät törmäyksen osapuolet, ja ohjelmassa voidaan reagoida oikealla tavalla. Laserin osuessa viholliseen, pelaajan pisteet lisääntyvät, vihollinen poistetaan kentältä ja muun laivaston nopeus kiihtyy. Muuriin osunut laser aiheuttaa muurin murtumisen tai lopullisen tuhoutumisen.

Törmäysten tarkistuksiin käytetään kummassakin pelimoottorissa fysiikkamoottoria. Yksinkertaisissa peleissä, kuten Space Invadersissa ei kuitenkaan tarvita fysiikanmallinnusta, vaan törmäysten havaitseminen riittää. Molemmissa pelimoottoreissa ne ovat toteutettu samalla periaatteella.

```
bool GameScene::onContactBegin(PhysicsContact& contact)
{
  auto spriteA = (Sprite*)contact.getShapeA()->getBody()->getNode();
  auto spriteB = (Sprite*)contact.getShapeB()->getBody()->getNode();

  // Laser osuu viholliseen ->
  // räjähdys, lisätään pelaajan pisteitä ja poistetaan vihollinen pelistä
  if (spriteA->getName().compare("Laser") == 0 &&
      spriteB->getName().compare("Enemy") == 0)
  {
    if (spriteA->isVisible() && spriteB->isVisible())
    {
      // Ääniefektin soitto
      CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("Sounds/Explosion.mp3");

      // Poistetaan spritet
      spriteB->setVisible(false);
      spriteA->setVisible(false);

      // Lisätään pelaajalle piste
      addScore();
      updateScore();

      // Vihollislaivaston vauhtia kiihdytetään
      enemyMoveRate -= 0.01;
    }
  }

  // Laser osuu muuriin ->
  // soitetaan räjähdys, jos ensimmäinen kerta, niin hajotetaan muuria
  // toisella kerralla poistetaan
  if (spriteA->getName().compare("Laser") == 0 &&
      spriteB->getName().compare("Wall") == 0)
  {
    if (spriteA->isVisible() && spriteB->isVisible())
    {
      // Ääniefektin soitto
      CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("Sounds/Explosion.mp3");

      // Poistetaan laser
      spriteA->setVisible(false);

      // Jos muurilla ei vielä osumia -> hajoitetaan sitä
      if (spriteB->getTag() > 1)
      {
        SpriteFrameCache* cache = SpriteFrameCache::getInstance();
        spriteB->setDisplayFrame(cache->spriteFrameByName("Wall_2.png"));
        spriteB->setTag(spriteB->getTag() - 1);
      }
      else // muuten tuhotaan lopullisesti
        spriteB->setVisible(false);
    }
  }

  return false;
}
```

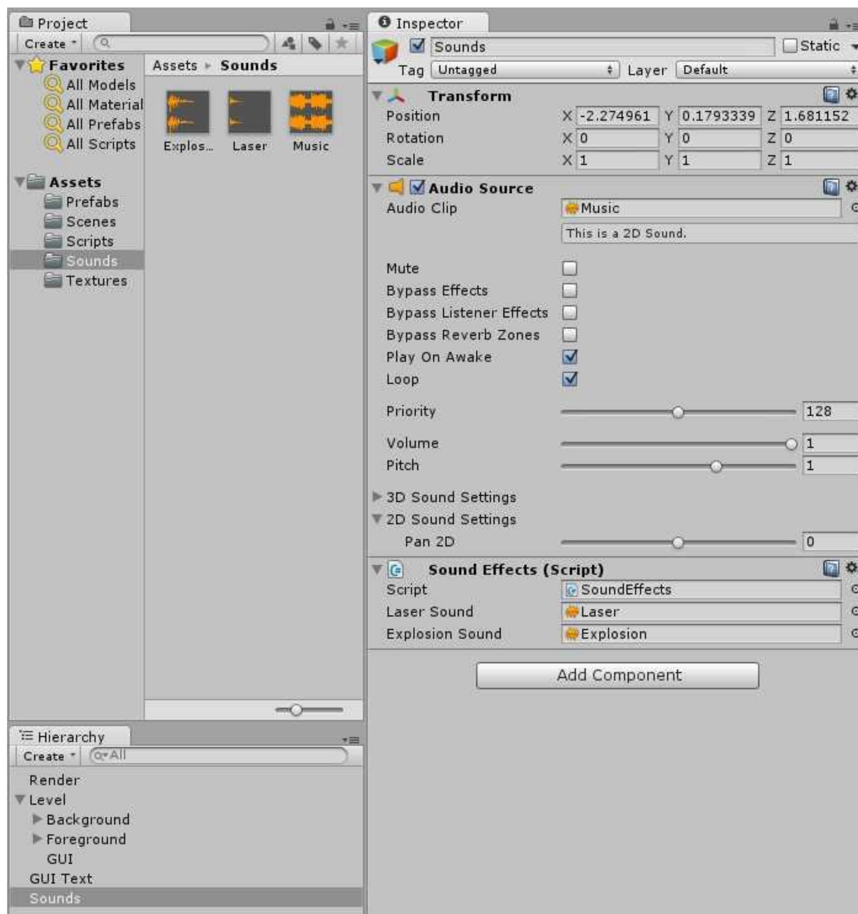
LÄHDEKOODI 17 Cocos2d-x – Osumien käsittelyä

5.6.9 Äänet

Unityssä eri peliobjekteilla voi olla omat äänilähteensä. Niitä voi ajatella pelikentällä olevina kauittimina, jotka on kiinnitetty peliobjekteihin. Esimerkiksi autoon voidaan liittää moottorin ääni tai koiraan haukkuminen. Yleensä äänikuuntelija, audio listener, liitetään kameraan. Se kuuntelee eri äänilähteitä, ja 3D-maailmassa äänilähteet kuuluvat niiden sijainnin perusteella eri voimakkuuksilla, eri suunnista.

2D-peleissä tilavaikutelmaa ei tarvita, eikä eri äänilähteitä tarvitse sijoitella eri puolille pelikenttää, vaan yksi audiolähde riittää. Skenelle voidaan luoda uusi peliobjekti, Sounds, johon liitetään audio source komponentti (kuvio 27). Assettien äänikansiossa olevien äänitiedostojen 3D ominaisuudet täytyy vaihtaa 2Dksi, jolloin niiden sijainnista ei välitetä ääntä soitettaessa.

Tässä pelissä käytetään musiikkia aina pelitilassa ollessa. Se tehdään raahaamalla musiikkiassetti audio sourcen clipiksi, ja asettamalla play on awake ja loop ominaisuudet päälle. Näin tehtäessä musiikki alkaa, kun audio source -komponentti luodaan, ja sitä soitetaan jatkuvassa luupissa.



KUVIO 27 Unity – Äänitiedostojen liittäminen skriptiin

Pelissä käytettäviä äänitehosteita varten tehdään skripti (lähdekoodi 18), jolla audiotiedostot soitetaan. Skripti liitetään Sounds objektille, jolloin sen ominai-

suuksissa olevia audiotiedostoja voi vaihtaa helposti käyttöliittymän kautta. Tässä käytetty ratkaisu ei kuitenkaan sovellu paljon eri ääniä käyttäviin peleihin.

Ääniefektejä soitetaan peleissä eri syistä. Tässä pelissä soitetaan efektejä tykin tulittaessa (lähdekoodi 19), ja laserin osuessa kohteeseen. Osumaan reagoidaan tapahtumia käsittelevissä OnCollisionEnter2D tai OnTriggerEnter2D metodeissa.

```
using UnityEngine;
using System.Collections;

public class SoundEffects : MonoBehaviour
{
  public static SoundEffects Instance;

  public AudioClip laserSound;
  public AudioClip explosionSound;

  void Awake()
  {
    Instance = this;
  }

  public void PlayLaserFX()
  {
    PlaySound(laserSound);
  }

  public void PlayExplosionFX()
  {
    PlaySound (explosionSound);
  }

  private void PlaySound(AudioClip clip)
  {
    AudioSource.PlayClipAtPoint(clip, transform.position);
  }
}
```

LÄHDEKOODI 18 Unity – Ääniefektien soittoskripti

```
public void Shoot()
{
  // Soitetaan laserin ääniefekti
  SoundEffects.Instance.PlayLaserFX();
}
```

LÄHDEKOODI 19 Unity – Ääniefektin soitto tulituksessa

Cocos2d-x:ssä äänitiedostot voidaan ladata valmiiksi muistiin peliä käynnistettäessä. Space Invaders kloonissa pelikentän käynnistys aloittaa myös taustamusiikin soiton, ja soittaa sitä taustalla kunnes peli lopetetaan. Esimerkissä on myös laserin ääniefekti (lähdekoodi 20). Musiikin ja ääniefektien soitot on to-

teutettu Cocos2d-x:ssä yhtä jaettua singleton oliota käyttäen. Rajapinta ja toiminnallisuus ovat hyvin yksinkertaiset ja selkeät, mutta 2D-peleihin aivan riittävät.

```
#include "SimpleAudioEngine.h"

bool GameScene::init()
{
  // Ääniefektien lataus
  CocosDenshion::SimpleAudioEngine::getInstance()->preloadEffect("Sounds/Laser.mp3");
  CocosDenshion::SimpleAudioEngine::getInstance()->preloadEffect("Sounds/Explosion.mp3");

  // Musiikin lataus ja soitto taustalla
  if (!CocosDenshion::SimpleAudioEngine::getInstance()->isBackgroundMusicPlaying())
  {
    CocosDenshion::SimpleAudioEngine::getInstance()->
      preloadBackgroundMusic("Sounds/Music.mp3");
    // Soitetaan taustamusiikkia luupissa (2. parametri=true)
    CocosDenshion::SimpleAudioEngine::getInstance()->
      playBackgroundMusic("Sounds/Music.mp3", true);
  }
}

// Pelaaja tulittaa
void GameScene::shoot()
{
  // Ääniefektin soitto
  CocosDenshion::SimpleAudioEngine::getInstance()->playEffect("Sounds/Laser.mp3");
}
```

LÄHDEKOODI 20 Cocos2d-x – Ääniefektien ja musiikin lataus ja soitto

Unityssä on Cocos2d-x:ää monipuolisemmat ääntenkäsittelymahdollisuudet.

5.6.10 Käyttöliittymä

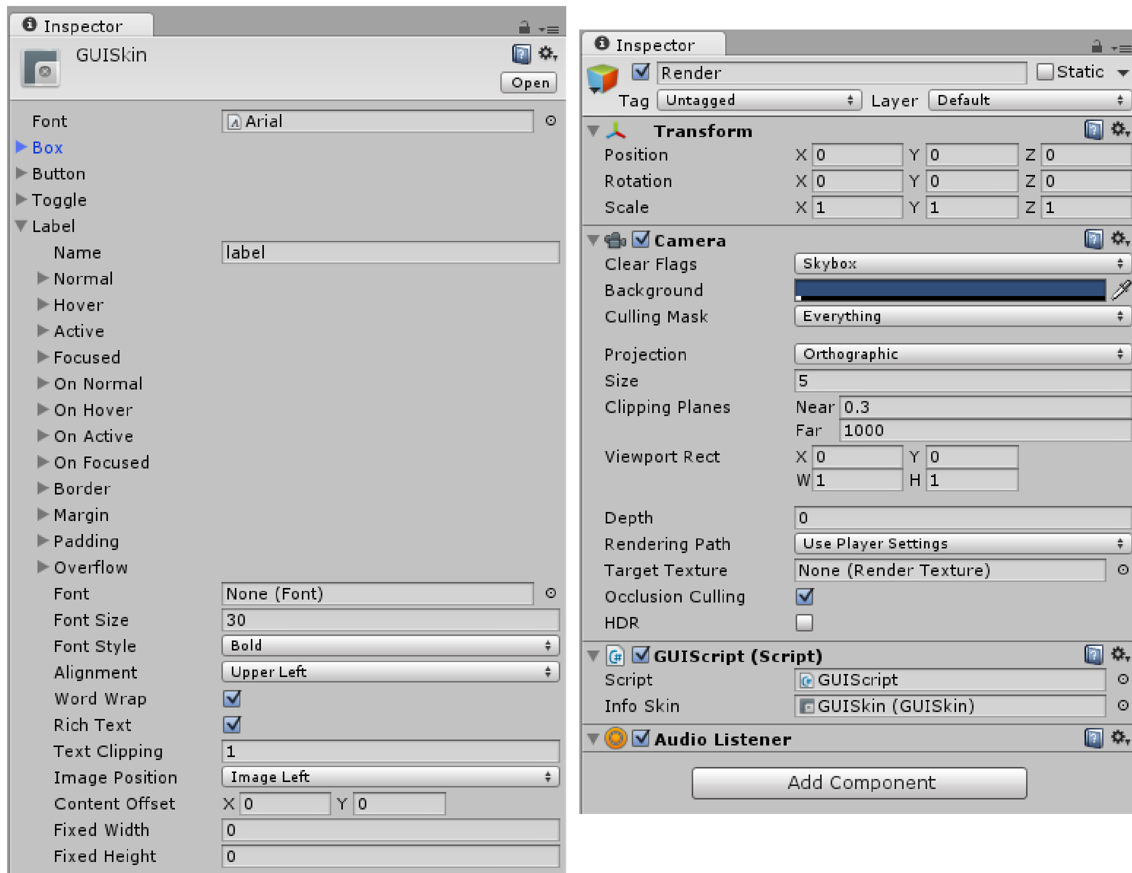
Pelaamisen aikana näkyvä käyttöliittymä on pelkistetty, ja siinä näkyvät ainoastaan pelaajan pisteet.

Unityssä yksinkertainen käyttöliittymä voidaan piirtää vanhalla tavalla skriptin OnGUI metodissa. Unityn versiossa 4.6 käyttöliittymien tekoon on tullut uusi tapa, jolla niistä saadaan rakennettua näyttävämpiä ja helpommin hallittavia. Esimerkissä luodaan GUIScript (lähdekoodi 21) skripti, joka liitetään kameraobjektiin. Sen OnGUI metodissa piirretään teksti Score ja pelajaan pisteet näkymättömän laatikon sisään ruudun yläreunaan.

```
void OnGUI()
{
  GUI.skin = infoSkin;
  GUI.Label(new Rect(Screen.width/2-100, 0, 200, 80), "SCORE " + Score.ToString());
  GUI.skin = null;
}
```

LÄHDEKOODI 21 Unity – Pelinaikainen käyttöliittymä

Käyttöliittymäkomponenttien kanssa on mahdollista käyttää GUISkiniä, jossa eri komponenttien ulkoasu määritellään (kuvio 28). Edellinen esimerkki käyttää GUISkiniä, jossa label komponentin tyyli on määritelty uudestaan.



KUVIO 28 Unity – GUISkin

Cocos2d-x:ssä käyttöliittymän teko vastaa Unityn vanhaa tapaa. Tekstilaput luodaan, ne sijoitetaan haluttuun kohtaan ruudulla, ja lopuksi lisätään skenelle (lähdekoodi 22). Sen jälkeen ne piirtyvät ruudulle. Tekstien päivitys onnistuu vaihtamalla tekstilappujen merkkijonoja setString metodilla.

Käyttöliittymien teko molemmilla pelimoottoreilla on helppoa, ja periaate samankaltainen.

5.6.11 Valikot

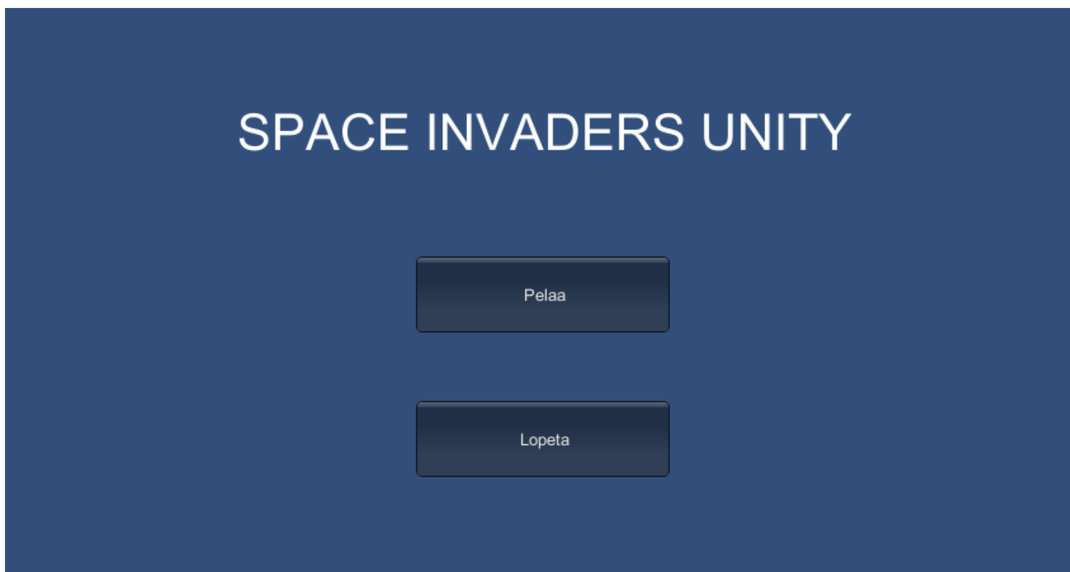
Valikot rakennetaan kummassakin pelimoottorissa skeneihin. Valikolle voidaan siten asettaa taustakuva, ja erilaisia valikkokomponentteja, kuten tekstejä, painonappeja, kuvia, animaatioita ja niin edelleen.

Unityssä valikko (kuvio 29) tehdään luomalla uusi skene. Siihen lisätään tyhjä peliobjekti, johon liitetään C#-skripti (lähdekoodi 23). Skriptin OnGUI-

metodissa voidaan luoda ja sijoittaa valikkoon käyttöliittymäkomponentteja. Edellisessä aliluvussa mainittu uusi tapa rakentaa käyttöliittymiä Unityn versiosta 4.6 alkaen koskee myös valikoiden tekoa. Tässä toteutuksessa käytetty vanha tapa toimii kuitenkin vielä toistaiseksi valikoita rakennettaessa. Valikoissa liikkuminen, eli skenestä toiseen siirtyminen, toteutetaan Unityssä Application.LoadLevel-metodilla, jolle annetaan parametrina vaihdettavan skenen nimi.

```
// Pisteiden otsikko ja varsinaiset pisteet
cocos2d::CCLabelTTF* playerScoreTitle;
cocos2d::CCLabelTTF* playerScoreLabel;

// Näytön näkyvän osan koko
Size visibleSize = Director::getInstance()->getVisibleSize();
Point origin = Director::getInstance()->getVisibleOrigin();

// Luodaan pisteiden otsikko ”SCORE” ja asetetaan se ruudulle
playerScoreTitle = CCLabelTTF::create("SCORE", "Arial", 30);
playerScoreTitle->setPosition(visibleSize.width/2, visibleSize.height - 30);
// Luodaan tekstikenttä, johon varsinaisia pisteitä päivitetään
playerScoreLabel = CCLabelTTF::create("0", "Arial", 30);
playerScoreLabel->setPosition(visibleSize.width/2, visibleSize.height - 60);
// Lisätään kumpikin skenelle
this->addChild(playerScoreTitle);
this->addChild(playerScoreLabel);


// Tekstin vaihto käyttöliittymään (pisteiden päivitys)
playerScoreLabel->setString(std::to_string(getScore()));
```

LÄHDEKOODI 22 Cocos2d-x – Käyttöliittymätekstien määrittely ja päivittäminen



KUVIO 29 Unity – Valikko

```
public class MenuScript : MonoBehaviour
{
  // Piirtää valikon käyttöliittymän
  void OnGUI()
  {
    // Tekstin tyylit
    GUIStyle style = new GUIStyle();
    style.fontSize = 40; // Fonttikoko
    style.normal.textColor = Color.white; // Fontin väri
    style.alignment = TextAnchor.MiddleCenter; // Keskitetty

    // Tekstilaatikko, jonka sisään otsikko tulee
    Rect title = new Rect(Screen.width / 2 - 200, 50, 400, 100);
    GUI.Box(title, "SPACE INVADERS UNITY", style);

    // Vakiokokoiset napit
    const int buttonWidth = 200;
    const int buttonHeight = 60;

    // Määrittelee Pelaa-napin sijainnin ruudulla
    Rect buttonPelaa = new Rect(Screen.width / 2 - buttonWidth / 2,
                            2 * Screen.height / 4 - buttonHeight  / 2,
                             buttonWidth,
                             buttonHeight);

    // Piirtää Pelaa-napin
    if (GUI.Button(buttonPelaa, "Pelaa"))
    {
      // Nappia klikattaessa, siirtyy peliin (Game sceneen)
      Application.LoadLevel("Game");
    }

    // Määrittelee Exit-napin sijainnin ruudulla
    Rect buttonExit = new Rect(Screen.width / 2 - buttonWidth / 2,
                           3 * Screen.height / 4 - buttonHeight  / 2,
                           buttonWidth,
                           buttonHeight);

    // Piirtää Exit-napin
    if (GUI.Button(buttonExit, "Exit"))
    {
      // Exittiä klikattaessa, lopettaa pelin
      Application.Quit();
    }
  }
}
```

LÄHDEKOODI 23 Unity – MenuScript piirtää valikon napit

Cocos2d-x:ssä yksinkertaiset valikkokomponentit voivat olla tekstejä ja kuvia (kuvio 30). Valikkokomponentit voivat toimia painonappeina tai muunlaisina käyttöliittymäkomponentteina, jolloin niitä voidaan käyttää valikoissa navigointiin ja valintojen tekoon. Dynaamisille komponenteille täytyy kirjoittaa takaisinkutsufunktiot, jolloin käyttäjän painaessa käyttöliittymäkomponenttia, kutsutaan haluttua metodia (lähdekoodi 24). Valikko luodaan antamalla sille

yksittäiset valikkokomponentit, ja lisäämällä luotu valikko skeneen (lähdekoodi 25).

```
// Spriten käyttö valikon nappina
auto menuPlaySprite = MenuItemImage::create("Pelaa_nappi.png",
                                             "Pelaa_nappi_klikattu.png",
                                            CC_CALLBACK_1(MainMenu::GoToGameScene, this));
```

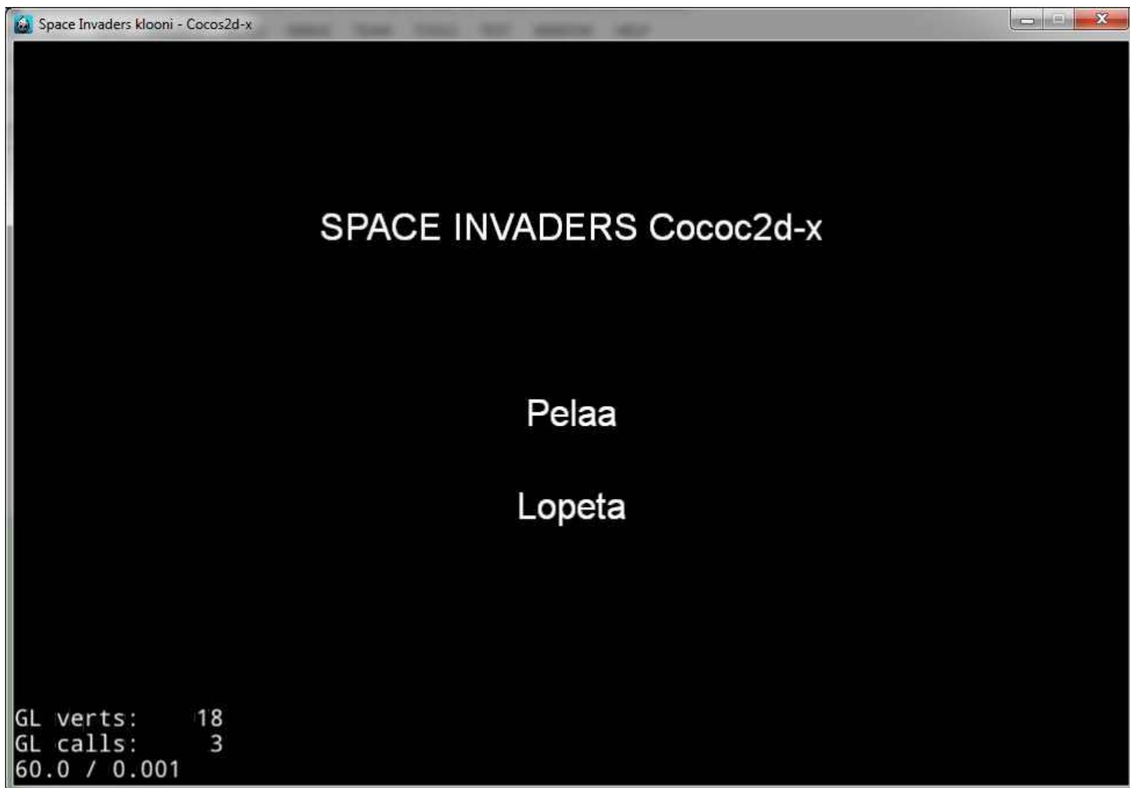
LÄHDEKOODI 24 Cocos2d-x – Valikon napin luonti spriteä käyttäen

```
bool MainMenu::init()
{
  // Kerroksen alustus
  if ( !Layer::init() )
  {
    return false;
  }

  Size visibleSize = Director::getInstance()->getVisibleSize();
  Vec2 origin = Director::getInstance()->getVisibleOrigin();

  // Menun otsikko
  auto menuTitle = MenuItemFont::create("SPACE INVADERS Cococ2d-x");
  menuTitle->setPosition(Point(visibleSize.width/2.0f, (visibleSize.height/4.0f) * 3.0f));

  // Pelaa - Käynnistää pelin
  auto menuPlay = MenuItemFont::create("Pelaa", CC_CALLBACK_1(MainMenu::GoToGameScene, this));
  menuPlay->setPosition(Point(visibleSize.width/2.0f, (visibleSize.height/4.0f) * 2.0f));

  // Lopeta - Lopettaa pelin
  auto menuExit = MenuItemFont::create("Lopeta", CC_CALLBACK_1(MainMenu::Exit, this));
  menuExit->setPosition(Point(visibleSize.width/2.0f, (visibleSize.height/4.0f) * 1.5f));

  // Luo valikon menuItemeista ja lisää sen skeneen
  auto menu = Menu::create(menuTitle, menuPlay, menuExit, NULL);
  menu->setPosition(Point(0, 0));
  this->addChild(menu);

  return true;
}
```

LÄHDEKOODI 25 Cocos2d-x – Valikon luonti

Siirtyminen skenejen välillä toteutetaan kutsumalla replaceScene-metodia (lähdekoodi 26).

Suorituksessa olevan skenen tilalle on myös mahdollista asettaa toinen skene pushScene-metodilla. Tällöin suoritettava skene asetetaan pinoon, ja sen tilalle käynnistetään parametrina annettava skene. Suorituksessa oleva skene voidaan lopettaa popScene-metodilla, joka tuhoaa skenen ja siirtyy päällimmäisenä pinossa olevaan skeneen tai parametrina saatavaan skeneen. Esimerkkinä edellisten metodien käytöstä on pelin tauko-tila.



KUVIO 30 Cocos2d-x – Valikko

```
// Siirtyy peliin
void MainMenu::GoToGameScene(Ref* pSender)
{
  auto scene = GameScene::createScene();
  Director::getInstance()->replaceScene(scene);
}

// Poistuu pelistä
void MainMenu::Exit(Ref* pSender)
{
  exit(0);
}
```

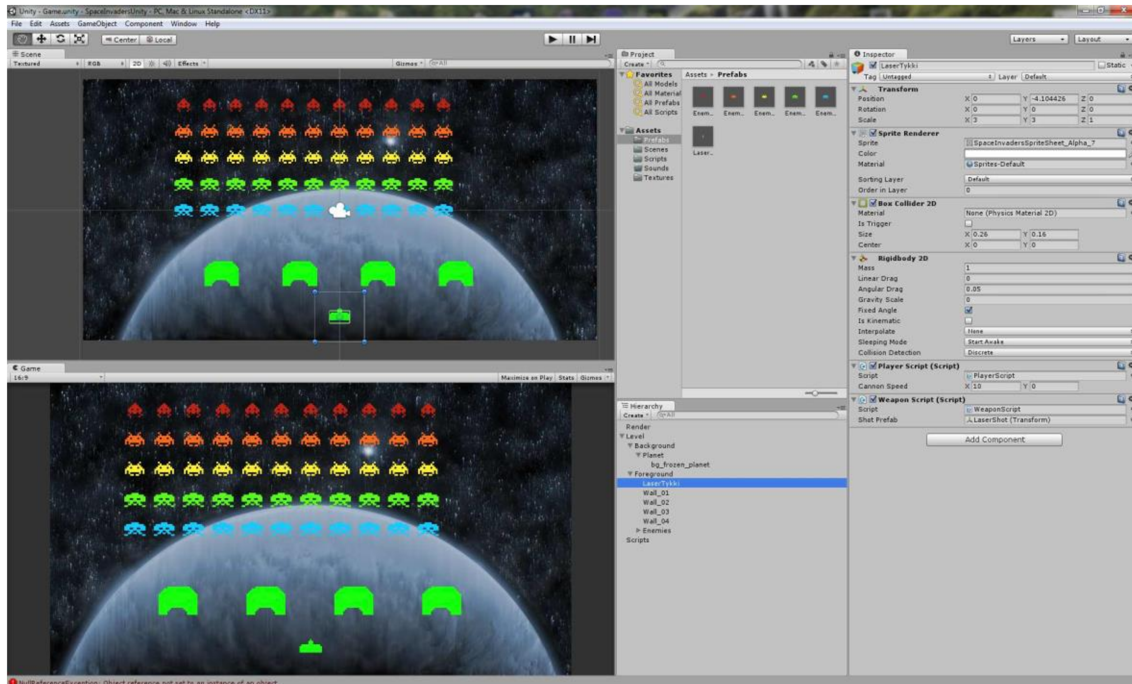
LÄHDEKOODI 26 Cocos2d-x – Siirtyminen skenejen välillä

Valikoiden teko on molemmilla pelimoottoreilla helppoa. Tässä tutkielmassa käytettiin Unityn vanhaa tapaa valikoiden tekoon, jolloin molempien pelien valikot luotiin lisäämällä näytettävät käyttöliittymäelementit koodilla.

5.7 Asennuspaketin luonti eri alustoille

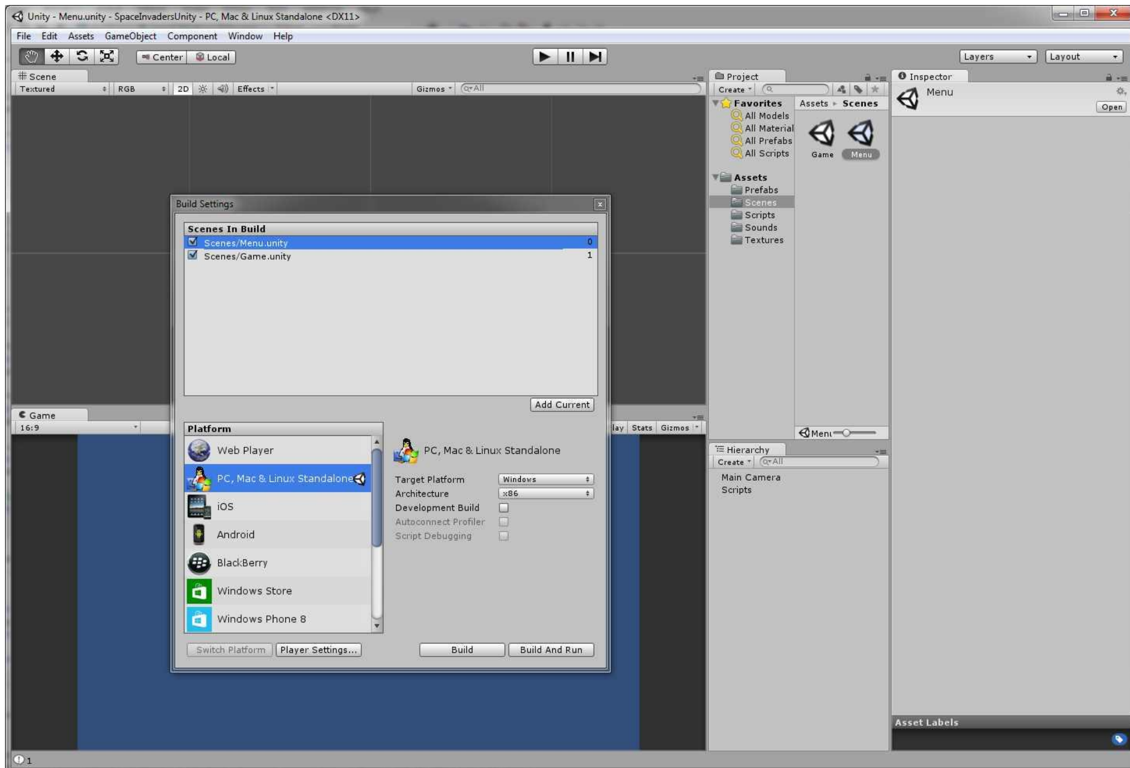
Valmis peli (kuvio 31) täytyy kääntää erikseen kaikille kohdealustoille. Pelin kääntämisellä tarkoitetaan tekstimuotoisen lähdekoodin kääntämistä objekti-

tiedostoiksi, jonka jälkeen linkittäjä liittää objektitiedostojen kirjastorutiinikutsut varsinaisiin kirjastorutiineihin, ja tuloksena saadaan ajokelpoinen konekielinen ohjelma. (Lappalainen, 2004, 66). Tässä tutkielmassa pelin kohdealustoina olivat Windows ja Android laitteet, koska iOS:lle kääntäminen olisi vaatinut Applen Macintoshin ja Windows Phone 8:lle käännös Windows 8:n.



KUVIO 31 Unity – Valmis taso editorissa

Unityssä peli rakennetaan eri kohdealustoille omassa ikkunassaan (kuvio 32). Rakentamisessa pelimoottori kääntää ja luo asennuspaketin, joka sisältää kaikki valmiissa pelissä tarvittavat resurssit, kuten kuvat ja äänet. Androidille pakettia luotaessa täytyy lisäksi määritellä Bundle identifier, joka on yksilöivä tunnus Google Playhyn ladattaville sovelluksille. Se on muotoa com.Yritys.Tuotenimi, mutta voi olla muuten mikä vaan, kunhan sovellusta ei julkaista. Lisäksi valitaan Androidin rajapinnan versio, joka peliä ajavassa laitteessa täytyy vähintään olla.



KUVIO 32 Unity – Pelin käännösikkuna

5.8 Yhteenveto

Luvun alussa tehtiin huomioita kohdealustojen suuresta kirjosta, pelimoottoreiden dokumentaatiosta ja peruskäsitteistä. Seuraavaksi käytiin läpi valittujen pelimoottoreiden käyttöönotto ja kehitysvälineiden perusnäkymät.

Luvussa toteutettiin suunnitelman mukaan vaiheittain Space Invaders pelin klooni käyttäen kahta käytettävyydeltään eri tason pelimoottoria, Cocos2d-x:ää ja Unityä. Eri vaiheissa kerrottiin havaintoja, siitä kuinka toteutus pelimoottoreilla tehtiin, ja niitä selvennettiin kuvankaappauksilla ja lähdekoodeilla. Kehitys vaiheittain aliluku jakautui yhteentoista vaiheeseen. Vaiheessa 1 käytiin läpi alustavat toimet, joita peliprojektia aloitettaessa on tehtävä. Vaiheessa 2 kerrottiin lyhyesti Unityn skripteistä. Vaiheessa 3 havainnollistettiin resurssien latausta esimerkin avulla. Vaiheessa 4 käytiin läpi kuinka pelin grafiikat otetaan pelimoottoreissa käyttöön, sekä miten suunnitteluvaiheessa otetaan huomioon erilaiset näyttöjen kuvasuhteet ja resoluutiot. Vaiheessa 5 tarkasteltiin peliobjekteja. Vaiheessa 6 esiteltiin kuinka pelaajan syöte luetaan kosketusnäytöltä ja näppäimistöltä. Vaiheessa 7 liikutettiin pelaajan ohjaamaa lasertykkiä ja tietokoneen hallitsemaa vihollislaivastoa. Vaiheessa 8 tutkittiin törmäyksiä ja kuinka ne käsitellään pelissä. Vaiheessa 9 käytiin läpi kuinka pelin ääniefektit ja musiikki soitetaan. Vaiheissa 10 ja 11 esitellään käyttöliittymän ja valikoiden tekoa.

Lopuksi peli käännettiin ja luotiin asennuspaketit eri alustoille.

6 TULOKSET JA JOHTOPÄÄTÖKSET

Tutkimusongelmana tässä tutkielmassa oli selvittää miten peliteollisuus on historiansa aikana muuttunut, ja millaisia kehitysvälineitä videopelien tekemiseen mobiililaitteille on olemassa. Lisäksi ongelmana oli yleiskäyttöisten kehitysvälineiden vertailu, niiden ominaisuuksien selvittäminen, sekä kehitysvaiheessa huomioon otettavat asiat käyttöjärjestelmien ja laitteiden eroissa.

Tutkielman teoreettis-käsitteellisessä osuudessa käytiin läpi videopelien historia, erityisesti harrastajapelinkehittäjän näkökulmasta. Videopelit tulivat ihmisten tietoisuuteen 1970-luvulla, jolloin kolikkopelit ja pelikonsolit levisivät ympäri maailman. 1980-luvulla pelaajille annettiin mahdollisuus toimia pelisuunnittelijoina, kun peleihin lisättiin sisäänrakennettuja kenttäeditoreja. Pian niiden jälkeen ilmestyivät ensimmäiset pelinteko-ohjelmat, joilla pystyi tekemään tietyn genren pelejä. 1990-luvun alussa Doom pelissä pelimoottori ja pelin sisältö olivat selkeästi erotettu toisistaan. Se mahdollisti pelien modauksen, jossa kuka tahansa sai mahdollisuuden muokata pelejä mieleisekseen. Videopeliteollisuus on muuttunut 2000-luvulla älypuhelinten myötä, ja yksittäiset pelinkehittäjät ovat taas voineet menestyä omaperäisillä peli-ideoillaan.

Ensimmäisissä videopeleissä ei ollut prosessoria tai RAM-muistia, vaan pelien logiikka toteutettiin laitteistotasolla. Intelin julkaistua 8080-mikroprosessorin, pelilaitteiden hinnat tippuivat huomattavasti, koska monet siihen asti laitteistotasolla hoidetut tehtävät voitiin toteuttaa ohjelmallisesti. Yksittäiset pelinkehittäjät ohjelmoivat pelejä vielä 1980-luvulla konekielellä. Laitetehojen kasvaessa, myös pelien vaatimukset ja kehitystiimien koot kasvoivat. Tehokkaampien laitteiden myötä, myös kehitysvälineet kehittyivät ja 1990-luvulla pelejä voitiin ohjelmoida korkeamman tason C-kielellä. 2000-luvulla suurimpien pelinkehitystiimien koot ovat kasvaneet satoihin henkilöihin. Videopelien kehitykseen vaaditaan työkaluja, joilla kehitys on helpompaa ja nopeampaa. Kehittäjät voivat käyttää luokkakirjastoja, jotka tarjoavat ohjelmointirajapinnan muun muassa grafiikkojen, äänien, fysiikkojen ja käyttäjän syötteiden ohjelmointiin. Nykyisin on kuitenkin kaupallisia ja avoimen lähdekoodin pelimoottoreita, joilla pelinkehitys on muuttunut huomattavasti. Pelimoottorit voidaan jakaa kolmeen tasoon, niiden käytettävyyden mukaan. Alimman tason

moottorit kehitetään kokonaan itse valmiiden rajapintojen päälle. Keskitason moottorit ovat suurimmalta osalta valmiita, mutta vaativat kuitenkin ohjelmointia. Ylimmän tason moottorit ovat ns. point-and-click moottoreita, jotka eivät vaadi ohjelmointia. Kehittäjän vaatimusten ja osaamisen pohjalta, markkinoilta löytyy pelimoottoreita, joilla kyetään alustariippumattomaan pelinkehitykseen. Mobiilipelinkehitykseen on lukuisia hyviä pelimoottoreita, joilla peli voidaan toteuttaa kaikille kolmelle, tällä hetkellä, suosituimmalle mobiilialustalle.

Tutkielman empiirisessä osuudessa vertailtiin kahta eri pelimoottoria käytännössä, ja niillä toteutettiin vaiheittain Space Invaders -pelin klooni Windows-tietokoneille ja Android-mobiililaitteille. Pelimoottoreiden lähestymistapa pelinkehitykseen eroaa hieman toisistaan. Molemmilla pelinkehitys osoittautui kuitenkin suhteellisen helpoksi verrattuna kehitykseen ilman valmista moottoria. Vaatimustasoltaan kaupallinen Unity on helppokäyttöisempi kuin avoimen lähdekoodin Cocos2d-x. Unityllä peli voidaan kehittää muuten kokonaan graafisen käyttöliittymän avulla, mutta skriptien ohjelmointia vaaditaan pelin toiminnallisuuden toteuttamiseen. Skriptaukseen käytettäviä kieliä ovat C#, JavaScript ja Boo. Cocos2d-x on vaatimustasoltaan keskitasoa, ja sillä kehitykseen vaaditaan C++, Lua tai JavaScript ohjelmointikielten hallintaa. Kehityksen aikana pelimoottoreissa ei ilmennyt virheellistä toimintaa, mikä antoi kuvan testatuista työkaluista.

Yhdeksi tutkimusongelmaksi oli asetettu kuinka paljon käyttöjärjestelmien ja laitteiden välisiä eroja on otettava huomioon kehitysvaiheessa. Tutkimuksen aikana selvisi, että molemmissa pelimoottoreissa on varauduttu hyvin mahdollisiin ongelmiin, kuten näyttöjen vaihteleviin resoluutioihin ja kuvasuhteisiin. Kehittäjän on kuitenkin tiedostettava asia, jotta siihen voidaan varautua, ja välttää ongelmatilanteet. Lisäksi kehittäjän on varauduttava laitteiden tehojen eroihin, sekä tietokoneiden ja mobiililaitteiden erilaisiin ohjaustapoihin. Kummallakin pelimoottorilla kehittäjältä on piilotettu alustakohtaiset yksityiskohdat ja ainoastaan projektitiedostot ja pääohjelmaan siirtyminen eroavat toisistaan, riippuen kohdealustasta. Ohjelmointi voidaan toteuttaa käyttäen samaa kieltä kehitettäessä kaikille pelimoottoreiden tukemille alustoille.

Vertailluista pelimoottoreista ei voi nostaa toista paremmaksi, niiden erilaisuuden vuoksi. Molemmilla 2D-pelien kehitys on nopeaa ja ohjelmointitaitoiselle helppoa. Yksinkertaisen pelin, kuten tässä Space Invadersin kehitys onnistuu kummallakin välineellä päivässä. Unityssä on kehittyneet graafiset editorit kehityksen eri vaiheisiin, ja pelin tasot voidaan suunnitella niiden avulla toiminnallisuutta lukuun ottamatta. Cocos2d-x:n rajapinta on ohjelmoijan kannalta selkeä ja looginen. Se on suunniteltu perinteisten 2D-pelien kehitykseen, kun Unity on alun perin 3D-pelimoottori. Unityn käyttöönotto on huomattavasti yksinkertaisempaa kuin Cocos2d-x:n. C++ on ollut yksi suosituimmista ohjelmointikielistä pelinkehityksessä jo pitkään. Cocos2d-x:n tukema kehitys C++:lla helpottaa olemassa olevan koodin siirrettävyyttä ja mahdollistaa vanhojen pelien julkaisun pienellä vaivalla uusille alustoille.

Pelin toteutusvaiheen tuloksena saatiin pelattavat prototyypit Space Invaders pelistä. Lähdekoodeista on jätetty pois sellaiset esimerkit, jotka ovat lähes samanlaisena jossain muussa yhteydessä. Tällaisia ovat vihollisen tulitus, joka vastaa pelaajan tulitusta. Vihollisen osuma pelaajaan, joka vastaa pelaajan osumista vihollliseen. Sekä pelin loputtua uusintapelin kysymysvalikko, joka vastaa päävalikkoa.

7 YHTEENVETO

Videopelit syntyivät kun ihmiset halusivat tehdä uusilla koneilla jotain hauskaa. Pelit ovat tulleet pitkän matkan laboratorioista kaikkien huviksi. Aikanaan nuorten poikien ylenkatsotusta harrastuksesta on tullut viihdeteollisuuden veturi, jonka vauhti uudella vuosikymmenellä mobiililaitteiden ja digitaalisen jakelun myötä näyttää vain kiihtyvän. Alkuaikojen rujoista ja yksinkertaisista peleistä on päästy jo elokuvamaisiin virtuaalitodellisuuksiin, joiden kehityskustannukset kasvavat konetehojen ja pelaajien vaatimusten mukana. Samaan aikaan kuitenkin myös yksinkertaiset pikkupelit elättävät yhä kasvavaa pelialaa, ja ammentavat ideansa videopelien kulta-ajalta. Eivätkä pelit ole ainoastaan retroharrastajien, vaan myös kasuaalipelaajien suosiossa.

Pelinkehitys on muuttunut konekielisestä ohjelmoinnista korkean tason kielten kautta nykyiseen, jopa ilman ohjelmointikieliä tapahtuvaan kehitykseen, jossa peli kootaan editoreilla erilaisista komponenteista. Laitekirjon kasvaessa, myös uusia ja parempia kehitysvälineitä on tullut kaikkien ulottuville. Hyviä luokkakirjastoja pelin kaikkiin osa-alueisiin löytyy valmiina, ja niiden avulla pelintekijä pääsee jo pitkälle. Niiden lisäksi, valmiita ja testattuja pelimoottoreita on useimpiin tarpeisiin, eikä pelinkehittäjän välttämättä tarvitse enää tehdä sellaista itse. Suurempi ongelma onkin löytää omiin tarkoituksiin sopivin vaihtoehto.

Kahdella eritasoisella pelimoottorilla 2D-pelin tekeminen osoitti käytännössä pelimoottoreiden arvon. Hyvin dokumentoiduilla, helppokäyttöisillä ja testatuilla työkaluilla pelinkehitys onnistui kohtuullisessa ajassa Windows- ja Android-laitteille. Pelinkehitys alustariippumattomasti ei vaadi kehittäjältä kohdelaitteiden tarkempaa tuntemusta, vaan pelimoottori piilottaa suurimman osan yksityiskohdista konepellin alle. Käytetyistä pelimoottoreista avoimen lähdekoodin Cocos2d-x vaatii kehittäjältä ohjelmointiosaamista, mutta kaupallinen Unity tarjoaa hieman helpomman lähestymistavan graafisella käyttöliittymällä.

LÄHTEET

Acuna, K. (2014, 18. kesäkuuta). The 30 Most Expensive Movies Ever Made. Haettu 19.9.2014 osoitteesta http://www.businessinsider.com/most-expensive-movies-2014-6?op=1

Amazon (2014). Apple iPhone. Haettu 1.12.2014 osoitteesta http://www.amazon.de/Apple-Smartphone-Touchscreen-Megapixel-Speicher-schwarz/dp/B001AXA056

Anonimowy, A. (2010, 29. kesäkuuta). Historia elektronicznej rozrywki. Haettu 1.12.2014 osoitteesta http://karawana.eu/index.php/historia-elektronicznej-rozrywki

Bates, B. (2004). *Game Design, Second Edition*. Boston: Thomson Course Technology PTR.

Bevan, M. (2012). SCUMM Origins. *Retro Gamer, 110*, 72-79.

Box2D (2014). Box2D – A 2D Physics Engine for Games. Haettu 10.12.2014 osoitteesta http://box2d.org/

CED Magic. (2014). Computer Space Arcade Game Advertisement. Haettu 1.12.2014 osoitteesta http://www.cedmagic.com/history/computer-space-ad.html

Chapple, C. (2014, 30. huhtikuuta). The top 16 game engines for 2014. Haettu 11.5.2014 osoitteesta http://www.develop-online.net/tools-and-tech/the-top-16-game-engines-for-2014/0192302

Cohen, D.S. (2014). Cathode-Ray Tube Amusement Device – The Electronic Game. Haettu 20.12.2014 osoitteesta http://classicgames.about.com/od/classicvideogames101/p/CathodeDevice.htm

Dillon, R. (2011). *The Golden Age of Video Games*. Florida: A K Peters/CRC Press.

Donovan, T. (2010). *Replay, The History of Video Games.* Great Britain: Yellow Ant.

Eddy, B. (2012). *Classic Video Games – The Golden Age, 1971-1984*. Oxford: Shire Publications Ltd.

Encyclopedia Gamia (2014). Video game console generations. Haettu 1.12.2014 osoitteesta http://gaming.wikia.com/wiki/Video_game_console_generations

Firebox (2014). Space Invaders Arcade Machine. Haettu 1.12.2014 osoitteesta http://www.firebox.com/product/847/Space-Invaders-Arcade-Machine

French, M. (2013, 4. lokakuuta). Inside Rockstar North – Part 2: The Studio. Haettu 19.9.2014 osoitteesta http://www.develop-online.net/studio-profile/inside-rockstar-north-part-2-the-studio/0184061

Goldberg, M. & Vendel, C (2012). *Atari Inc. Business Is Fun*. New York: Syzygy Company Press.

Grannell, C. (2013). Lode Runner. *Retro Gamer, 111.* 20-27.

Gregory, J. (2009). *Game Engine Architecture*. Florida: A K Peters/CRC Press.

Hiltunen, K., Latva, S. & Kaleva, J-P. (2014). *Peliteollisuus – kehityspolku*. (2. päivitetty painos). Helsinki: Tekes.

IDC (2014, Q2). Smartphone OS Market Share, Q2 2014. Haettu 23.11.2014 osoitteesta http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Indie DB (2014). 100 Most Popular Engines Today. Haettu 12.5.2014 osoitteesta http://www.indiedb.com/engines/top

Isaacson, W. (2011). *Steve Jobs.* Keuruu: Otavan Kirjapaino Oy.

Jarvis, E. (2012). Robotron: 2084. *Retro Gamer*, *111*. 26.

Kent, S. (2001). *The Ultimate History of Video Games.* New York: Three Rivers Press.

Khronos Group (2014). OpenGL ES. The Standard for Embedded Accelerated 3D Graphics. Haettu 10.12.2014 osoitteesta https://www.khronos.org/opengles/

Knotts, M. (2013, 22. marraskuuta). Gamechanger – Bill Budge. Haettu 1.12.2014 osoitteesta http://geekometry.com/2013/11/gamechanger-bill-budge/

Koskimies, K. & Mikkonen, T. (2005). *Ohjelmistoarkkitehtuurit*. Helsinki: Talentum Oyj.

Kuorikoski, J (2014). *Sinivalkoinen pelikirja – Suomen pelialan kronikka 1984-2014*. Saarijärvi: Fobos.

Lappalainen, V. (2004). *Ohjelmointi 2*. Jyväskylä: Jyväskylän yliopistopaino.

Lasar, M. (2011, 25 lokakuuta). Spacewar!, the first 2D top-down shooter, turns 50. Haettu 1.12.2014 osoitteesta http://arstechnica.com/gaming/2011/10/spacewar-the-first-2d-top-down-shooter-turns-50/

Levy, K. (2014, 7. heinäkuuta). The Most Expensive Video Games Ever Made. Haettu 19.9.2014 osoitteesta http://www.businessinsider.com/the-most-expensive-video-games-ever-made-2014-7?op=1

Levy, S. (2010). *Hackers, Heroes of the Computer Revolution*. California: O'Reilly.

Madhav, S. (2013). *Game Programming Algorithms and Techniques: A Platform-Agnostic Approach*. Indianapolis: R.R. Donnelley.

Mod DB (2014). 100 Highes Rated Engines. Haettu 12.5.2014 osoitteesta http://www.moddb.com/engines/rated

Mott, T. (2010). *1001 Video Game You Must Play Before You Die*. London: Cassell Illustrated.

Net Market Share (2014, lokakuu). Operating system market share reports. Haettu 23.11.2014 osoitteesta http://www.netmarketshare.com

Obsolete Technology (2013). IBM Personal Computer (PC). Haettu 1.12.2014 osoitteesta http://oldcomputers.net/ibm5150.html

Obsolete Technology (2014). Commodore Amiga 500. Haettu 1.12.2014 osoitteesta
http://oldcomputers.net/amiga500.html

OpenGL (2014). OpenGL. The Industry's Foundation for High Performance Graphics. Haettu 10.12.2014 osoitteesta
https://www.opengl.org/

Patricios, P. (2011, marraskuuta). Extending Texture2D. Haettu 1.12.2014 osoitteesta
http://www.mpatric.com/2011-10-31-extending-texture2d-part-2-sprite-sheets

Pelikirja Special Operations Group (1991). Piratismi - Pelaamisen pimeä puoli. *Pelit vuosikirja 1991 kevät,* 22-24.

Pixel Prospector.com (2014). The Big List of Game Making Tools. Haettu 30.9.2014 osoitteesta
http://www.pixelprospector.com/the-big-list-of-game-making-tools

Railton, J. (2005). *The A-Z of Cool Computer Games*. London: Allison & Busby Limited.

Santelices, R. & Nussbaum, M. (2001). A framework for the development of videogames. *Software: Practice and Experience, 31(11)*, 1091-1107.

ScummVM. (2006, 23. toukokuuta). AGI. Haettu 21.9.2014 osoitteesta
http://wiki.scummvm.org/index.php/AGI

ScummVM. (2009, 14. helmikuuta). SCI. Haettu 21.9.2014 osoitteesta
http://wiki.scummvm.org/index.php/SCI

ScummVM. (2001, 8. lokakuuta). SCUMM. Haettu 21.9.2014 osoitteesta
http://wiki.scummvm.org/index.php/SCUMM

SDL (2014). About SDL. Haettu 10.12.2014 osoitteesta
https://www.libsdl.org/

Sinerma, O. (2009). Tietokonepelit ohjelmistotuotantoprojektin aiheena pelialan orientoivassa korkeakouluopetuksessa. Tietojenkäsittelyn pro gradu – tutkielma. Helsingin yliopisto.

Stuart, K. (2014, 1. syyskuuta). Destiny: behind the scenes of the world's most expensive video game. Haettu 19.9.2014 osoitteesta
http://www.theguardian.com/artanddesign/2014/sep/01/destiny-behind-scenes-most-expensive-video-game-ever

Taskumuro (2010, 25. tammikuuta). Matopeli on kaikkien aikojen pelatuin videopeli. Haettu 1.12.2014 osoitteesta
http://taskumuro.com/matopeli-on-kaikkien-aikojen-pelatuin-videopeli

The Game Console.com (2014). Ralph H. Baer Brown Box Prototype Game Console. Haettu 1.12.2014 osoitteesta
http://www.thegameconsole.com/ralph-baer-brown-box/

The Game Design Forum (2012). An Introduction to Videogame Design History. Haettu 17.11.2014 osoitteesta
http://thegamedesignforum.com/features/GDH_1.html

Ward, J. (2008, 29 huhtikuuta). What is a Game Engine. Haettu 11.5.2014 osoitteesta

http://www.gamecareerguide.com/features/529/what_is_a_game.php?print=1
Wikipedia (2014 a). Atari 2600. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/Atari_2600
Wikipedia (2014 b). Cocos2d. Haettu 15.12.2014 osoitteesta http://en.wikipedia.org/wiki/Cocos2d
Wikipedia (2014 c). Commodore 64. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/Commodore_64
Wikipedia (2014 d). DirectX. Haettu 10.12.2014 osoitteesta http://en.wikipedia.org/wiki/DirectX
Wikipedia (2014 e). Home Computer. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/Home_computer
Wikipedia (2014 f). E.T. the Extra-Terrestrial (video game). Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/E.T._the_Extra-Terrestrial_(video_game)
Wikipedia (2014 g). List of Mario LCD game. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/List_of_Mario_LCD_games
Wikipedia (2014 h). OpenAL. Haettu 10.12.2014 osoitteesta http://en.wikipedia.org/wiki/OpenAL
Wikipedia (2014 i). Pong. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/Pong
Wikipedia (2014 j). Rally Speedway. Haettu 15.5.2014 osoitteesta http://www.c64-wiki.com/index.php/Rally_Speedway
Wikipedia (2014 k). ZX Spectrum. Haettu 1.12.2014 osoitteesta http://en.wikipedia.org/wiki/ZX_Spectrum
Wilcox, M. (2014, 16. syyskuuta). Top Game Development Tools : Pros and Cons. Haettu 15.12.2014 osoitteesta http://www.developereconomics.com/top-game-development-tools-pros-cons/

LIITE 1 KONSOLISUKUPOLVET (ENCYCLOPEDIA GAMIA, 2014)

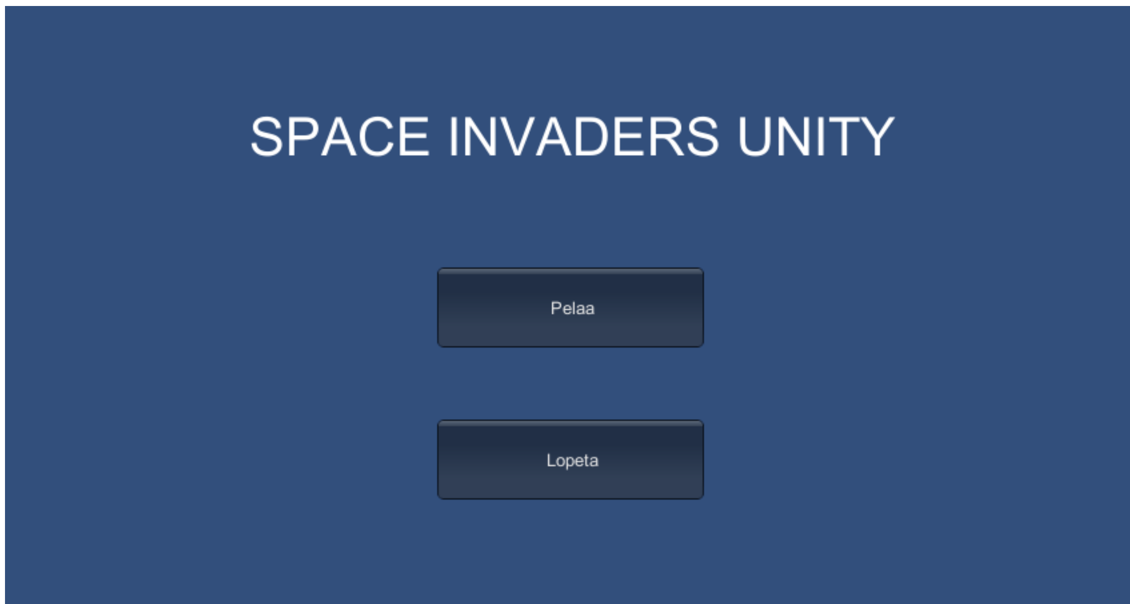
Sukupolvi	Konsoli	Julkaistu
1.	Magnavox Odyssey	1972
1972 -1977	Atari Pong	1975
	Coleco Telstar	1976
	Nintendo Color TV Game	1977
2.	Fairchild Channel F	1976
1976-1984	RCA Studio II	1977
	Atari VCS (Video Computer System) / Atari 2600	1977
	Bally Astrocade	1977
	Magnavox Odyssey²	1978
	Mattel Intellivision (Ensimmäinen 16-bittinen)	1980
	Emerson Arcadia 2001	1982
	Coleco Vision	1982
	Atari 5200	1982
	Vectrex	1982(US)
		1983 (UK, Jp)
	Nintendo Game & Watch	1980-1991
	Sega SG-1000	1983
3.	Nintendo Famicom /	1983 (Jp)
1983-1992	Nintendo Entertainment System (NES)	1985 (US),
8-bittiset		1986-1987 (EU)
	Sega SG-1000 Mark III /	1985 (Jp),
	Sega Master System	1986 (US), 1987 (EU)
	Atari 7800	1986
	Nintendo Game Boy	1989 (Jp),
		1990 (EU)
	Sega Game Gear	1990 (Jp), 1991 (EU)
4.	NEC PC-Engine	1987
1987-1996	TurboGrafx 16 (NEC)	1989
16-bittiset	Sega Mega Drive / Sega Genesis	1989
	PC-Engine SuperGrafx (NEC)	1989
	Neo Geo AES	1990
	Super Famicom / Super Nintendo (SNES)	1990 (Jp), 1991 (US)
	Philips CD-i	1991
	Atari Jaguar	1993
	Atari Lynx	1989
	Sega Game Gear	1990
5.	3DO	1993
1993-2006	PC-FX (NEC)	1994
32- ja	Sega Saturn	1994
64-bittiset	Sony Playstation	1994 (Jp), 1995 (EU)
	Nintendo 64	1996
	Apple Bandai Pippin	1996
	Nintendo Virtual Boy	1995
	Nintendo Game Boy Color	1998

6.	Sega Dreamcast	1998 (Jp), 1999 (EU)
1998-2012	Sony Playstation 2	2000
	Microsoft Xbox	2001 (US), 2002 (Jp)
	Nintendo Game Cube	2001 (Jp), 2002 (EU)
	Neo Geo Pocket	1998
	NeoGeo Pocket Color	1999 (UK)
	Nokia N-GAGE	2003
7.	Microsoft Xbox 360	2005
2004-	Sony Playstation 3	2006 (Jp), 2007 (EU)
	Nintendo Wii	2006
	Evo: Phase One	2008
8.	Nintendo Wii U	2012
2011-	Sony Playstation 4	2013 (EU), 2014 (Jp)
	Microsoft Xbox One	2013

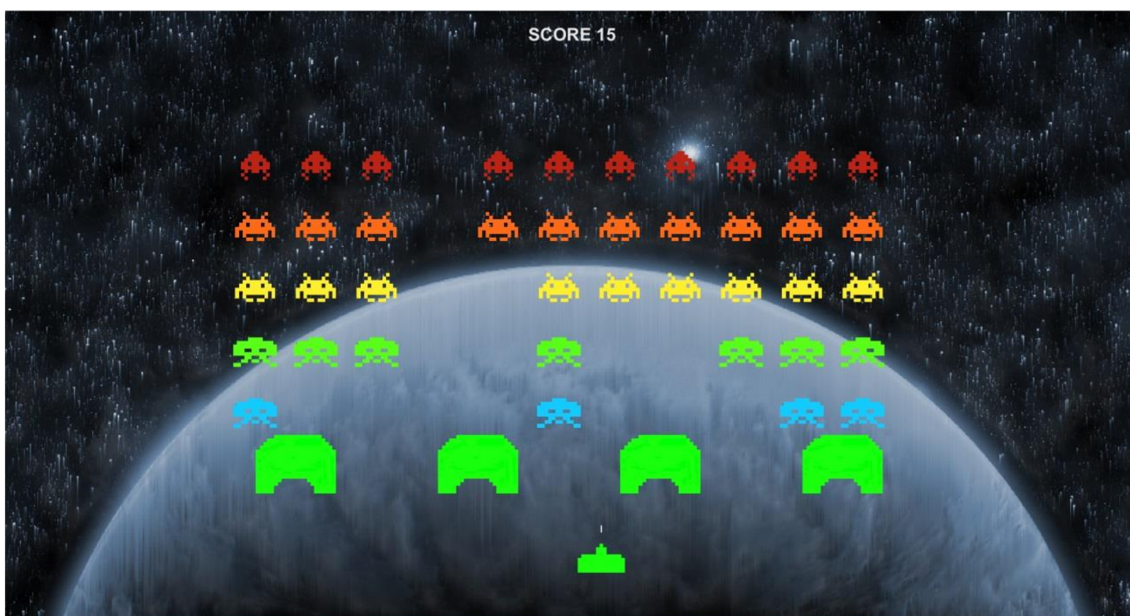
LIITE 2 KOTIMIKROT (WIKIPEDIA, 2014E)

Kotimikro	Julkaistu
Apple II	1977 (kesäkuu)
Tandy Radio Shack TRS-80	1977 (elokuu)
Commodore PET	1977 (lokakuu)
TI-99/4	1979 (kesäkuu)
Atari 400/800	1979 (marraskuu)
Commodore VIC-1001 / VIC-20	1980 (lokakuu) / 1981 (toukokuu)
TRS-80 Color Computer (Tandy Color Computer)	1980 (heinäkuu)
Texas Instruments TI-99/4A	1981 (kesäkuu)
Sinclair ZX81	1981 (maaliskuu)
BBC Micro	1981 (joulukuu)
MicroBee	1982 (helmikuu)
Sinclair ZX Spectrum	1982 (huhtikuu)
Dragon 32	1982 (elokuu)
Commodore 64	1982 (elokuu)
Apple IIe	1983 (tammikuu)
Oric-1 (Oric Atmos 1984, Oric Stratos 1985, Oric Telestrat 1986)	1983 (tammikuu)
MSX (MSX2 1986, MSX2+ 1988, MSX Turbo R 1990)	1983 (kesäkuu)
Acorn Electron	1983 (elokuu)
Coleco Adam	1983 (lokakuu)
VTech Laser 200	1983 (marraskuu)
Apple Macintosh	1984 (tammikuu)
Apple IIc	1984 (huhtikuu)
Tiki 100	1984 (huhtikuu)
Amstrad CPC 464	1984 (kesäkuu)
Commodore 128	1985 (tammikuu)
Atari ST	1985 (kesäkuu)
Commodore Amiga 1000	1985 (heinäkuu)
Apple IIGS	1986 (syyskuu)
Acorn Archimedes	1987 (kesäkuu)
Commodore Amiga 2000 (Amiga 3000 1990, Amiga 4000 1992)	1987 (maaliskuu)
Commodore Amiga 500 (Amiga 600 1992, Amiga 1200 1992)	1987 (lokakuu)

LIITE 3 UNITY-PELIN KUVANKAAPPAUKSET

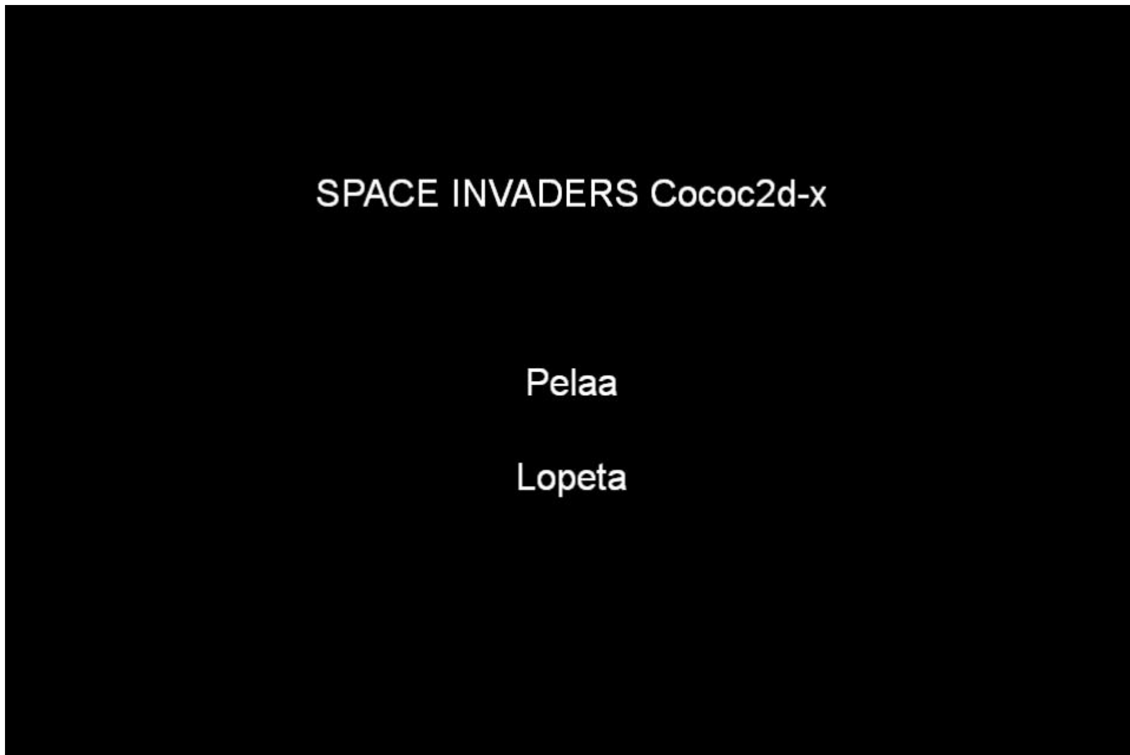


Valmiin pelin valikko Unityllä

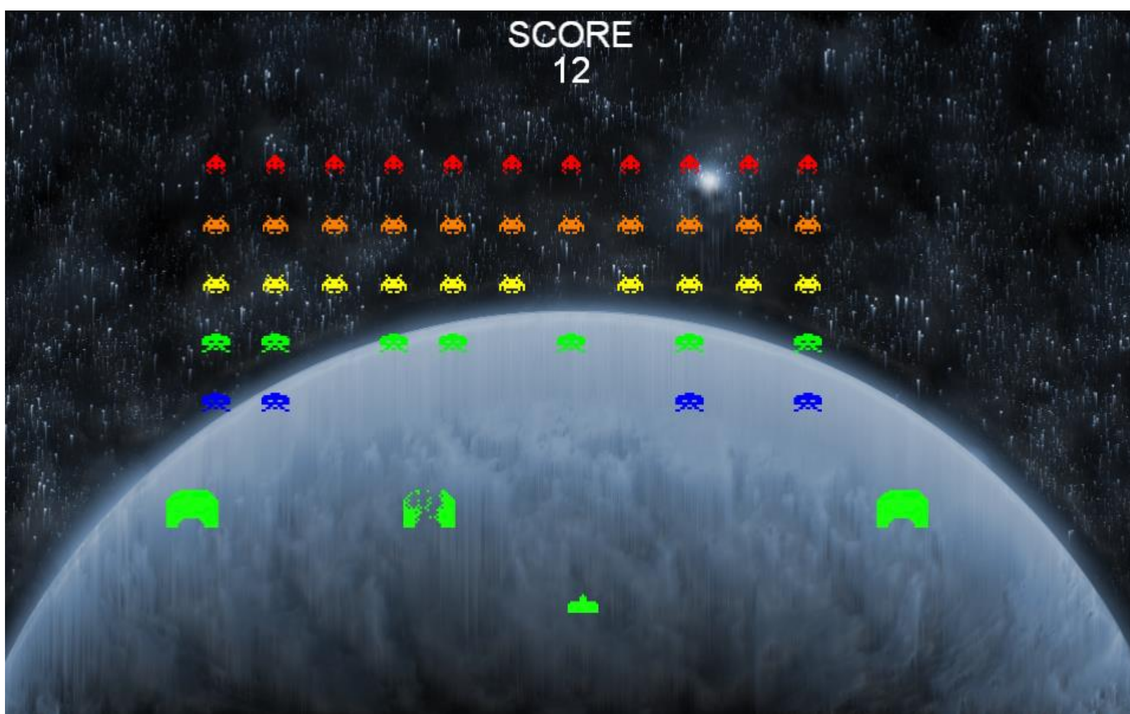


Valmis peli Unityllä

LIITE 4 COCOS2D-X-PELIN KUVANKAAPPAUKSET



Valmiin pelin valikko Cocos2d-x:llä



Valmis peli Cocos2d-x:llä