Samuli Salonen

Pelinkehitys prosessina

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

29.11.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Samuli Salonen Pelinkehitys prosessina 46 sivua + 1 liite 29.11.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	yliopettaja Kari Aaltonen

Insinöörityön tavoitteena oli toimivan pelisovelluksen tuottaminen. Tähän käytettiin lukuisia eri ohjelmia, mutta kokoava työ tehtiin XNA-kehitystyökaluilla ja C#-ohjelmointikieltä käyttäen. Toisena tavoitteena oli tietokonepelien kehitysprosessin tarkasteleminen.

Pelialan viime vuosien muutokset ovat parantaneet pienempien pelinkehittäjien mahdollisuuksia toimia. Aiempaa kehittyneemmät kehitysympäristöt esimerkiksi vähentävät resurssien tarvetta ja vaadittavaa työmäärää, joten myös muutaman tai jopa vain yhden hengen tiimi voi saada jotain aikaan. Insinöörityössä kokeiltiin jälkimmäistä eli yhden hengen vaihtoehtoa.

Pelien ja pelien tekemisen historiaan ja nykypäivään tutustuttiin myös. Tekniikan kehittymisestä huolimatta moni vuosikymmenten takainen peli löytää vastineensa nykypelien joukosta. Moni pelinkehityksessä toimivaksi havaittu periaate, kuten pikainen toimivan prototyypin tuottaminen sekä jatkuva testaaminen ja iterointi, todettiin toimivaksi myös tämän projektin kohdalla. Pelinkehityksen teoriasta suuri osa käsittelee prosessin eri toimijoiden välistä yhteistyötä, mutta sitä ei tässä työssä työryhmän pienen koon vuoksi päästy juuri soveltamaan.

Tuotetun pelisovelluksen nimeksi valikoitui Nopea Kuha, ja sen vaiheita käydään tässä työssä läpi etenkin suunnittelun sekä graafisen että ohjelmointiteknisen toteutuksen osalta.

Nopean Kuhan valmistuttua sen testasivat ulkopuoliset testaajat, ja peli todettiin toimivaksi. Monet osa-alueet, kuten audiovisuaalinen ilme ja yleinen toteutus, keräsivät kiitosta, mutta toisaalta löytyi myös parantamisen varaa. Pelin kontrolleja esimerkiksi moitittiin hieman huteriksi. Kokonaisuus todettiin kuitenkin onnistuneeksi ja tarvittaessa jatkokehityskelpoiseksi. Myös käytetyt kehitystyökalut osoittautuivat tämänkaltaisiin projekteihin sopiviksi.

Avainsanat	pelinkehitys, väliohjelmistot

Author Title Number of Pages Date	Samuli Salonen Game Development as a Process 46 pages + 1 appendix 29 November 2013
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Kari Aaltonen, Principal Lecturer

The objective of this thesis was producing a working game application. Several programs were used to achieve this, but everything was assembled together by using XNA development tools and C# programming language. Another objective for this study was surveying the development process of computer games.

Changes in the games industry have improved the chances and possibilities of smaller actors during recent years. More advanced development environments have resulted in reduced need for resources and smaller labor intensity. Now even very small teams of even only one member can hope to achieve something. Such feat was attempted as a part of this study.

The history and the present day situation of games and game development were investigated in this study. Despite the technical advances many a game of past decades finds its counterpart from amongst the games of today. Certain widespread practices of the industry, such as fast prototyping and constant testing and iteration, were also found to work remarkably well in this project. A substantial part of game development theory focuses on the interplay of various actors in the development process. Its application in practice was fairly limited due to the small team size in this project.

The game that was produced ended up with a name Nopea Kuha and the phases of its development will be detailed here.

As it was finished, Nopea Kuha was tested and found satisfactory by the outside testers. Many features, such as the game's audio-visual look and its overall implementation, received compliments, although some areas were found to still have room for improvement. For example the game's controls were thought to be a bit shaky. All in all the project was considered successful and if need be a good basis for further development. Also all the development tools used were found suitable for similar projects.

Keywords	game development, middleware

Sisällys

Lyhenteet

C#	Ohjelmointikieli, joka on tarkoitettu yhdistämään Javan ja C++:n parhaat puolet. Luetaan C sharp.
GDD	Game design document. Yksityiskohtainen suunnitteludokumentti, joka määrittelee, mitä kaikkea kehitettävää peliä varten pitää tuottaa.
.NET	Microsoftin luokkakirjastot, joita XNA tarvitsee toimiakseen.
TDD	Technical design document. Tekninen dokumentti, joka määrittelee, mitä teknologioita GDD:ssä määriteltyjen asioiden toteuttamiseen käytetään.
XACT	Cross-platform Audio Creation Tool. XNA:n työkalu ääniresurssien hallintaan.
XNA	XNA's Not Acronymed. Microsoftin pelinkehitykseen tarkoittama kehitysympäristö. Rekursiivinen lyhenne, joten auki kirjoittaminen on mahdotonta.

1 Johdanto

Insinöörityön tavoitteena on luoda pienimuotoinen pelisovellus ja samalla tarkastella tietokone- ja videopelien kehitysprosessia: mikä tämä prosessi on ja millä tavalla se ilmenee alalla niin sanotusti tuotantokäytössä sekä miten sama prosessi soveltuu ja suhtautuu pienen pelisovelluksen tuottamiseen. Kontrastia näiden kahden välillä yritetään rajoittaa keskittymällä ennemmin pelialan pieniin ja keskisuuriin toimijoihin kuin kymmenien miljoonien eurojen budjeteilla pelaaviin jättiläisiin. Työn aluksi määritellään, mitä tietokone- ja videopelit oikein ovat, sekä tutustutaan niiden ja niiden kehittämisen historiaan ja nykytilaan. Pelinkehityksen eri osa-alueita käydään myös läpi ja tarkastellaan tuotantoryhmien sisäisiä rooleja.

Pelialan viime vuosien muutokset ovat monella tapaa parantaneet pienempien pelinkehittäjien mahdollisuuksia toimia. Aiempaa kehittyneemmät kehitysympäristöt ja -työkalut esimerkiksi vähentävät resurssien tarvetta ja vaadittavaa työmäärää. Tässä työssä pelisovelluksen tuottamiseksi käyttöön otetaan yksi näistä teknologioista, joka on ollut osaltaan tuomassa pelialan pienempiä toimijoita takaisin marginaalista, eli Microsoftin XNA-nimellä kulkeva pelinkehitykseen tarkoitettu kehitysympäristö ja työkalukokoelma. Sen tulevaisuus on käymistilassa, sillä Microsoft on nyttemmin (2013) luopunut sen jatkokehittämisestä, mikä tarkoittaa, ettei tukea uusille alustoille ja käyttöjärjestelmille ole luvassa. Toisaalta XNA:n suosio pelinkehittäjien parissa on ollut sen verran hyvää, että siitä on tehty vapaan lähdekoodin sovitus nimeltään MonoGame, joka esikuvastaan poiketen mahdollistaa julkaisun myös muille kuin Microsoftin alustoille.

2 Pelinkehitys prosessina

2.1 Tietokonepelin määrittely

Ennen tietokonepelien ja niiden kehityksen historiaan siirtymistä selvennän muutamia käsitteitä, tai ainakin tuon esiin niiden monimerkityksellisyyden tai epätäsmällisyyden. Ensimmäisenä on vuorossa termi *peli*, joka sattui olemaan myös juuri se sana, jota filosofi Ludwig Wittgenstein käytti havainnollistaessaan teoksessaan *Filosofisia tutkintoja* eksaktien määritelmien muodostamisen vaikeutta. Wittgenstein listasi useita uskottavia ja järkeenkäyviä ehdotuksia pelin määritelmäksi, mutta kumosi niistä jokaisen yksi toisensa jälkeen osoittamalla jonkin tunnetun pelin, joka ei mahtunut juuri senhetkisen määritelmän sisään. Tämä epämääräisyys ei kuitenkaan ollut hänen mielestään minkäänlainen ongelma, sillä hänen mukaansa ihmiset luonnostaan kommunikoivat käyttäen väljiä ja joustavia kategorioita tiukasti rajattujen ja määriteltyjen käsitteiden sijaan. [1.]

Jokaisella lienee selkeähkö käsitys siitä, mikä peli on, huolimatta sen täsmällisen määrittelyn vaikeudesta [2]. Vaikka yleisesti arvostettu onkin, Wittgensteinin näkemys ei kuitenkaan ole saanut hänen jälkeensä tulleita pidättäytymään yrityksistä jälleen uusien määritelmien luomiseksi. Käyn läpi yhden määritelmän, eli suomalaisen pelinkehittäjän Mikko Monosen [3] näkemyksen käsitteestä peli. Mononen listaa ominaisuuksia, joita peleillä on, mutta muistaa korostaa, että ehdottomien sääntöjen sijaan ne ovat enemmänkin summittaisia suuntaviivoja.

Ensimmäisinä Monosen listalla ovat säännöt ja peliympäristö. Jokaisessa pelissä on rajattu ympäristö ja säännöt, joiden mukaan pelissä toimitaan. Esimerkkinä on shakki, jossa peliympäristönä on shakkilauta, jolla nappuloita liikutellaan sääntöjen mukaan. Pelissä on myös päämäärä, tavoite, jota kohti pyritään. Jos päämäärä puuttuu, kyseessä on pelkkä lelu. [3.]

Toisena on muutos, jonkinlainen satunnaistekijä, jonka vuoksi pelitilanne ei koskaan ole täysin ennustettavissa tai samanlainen kuin aiemmin. Yksinkertaisimmillaan tämä voi olla vastustaja, jonka liikkeitä ei voi etukäteen tietää. Pelissä myös kilpaillaan. Vastustajina voivat olla toiset pelaajat, keinotekoiset vastustajat tai peliympäristön luomat rajoitukset. Sen lisäksi kullakin osallistuvalla pelaajalla tulee olla jossain määrin tasa-

vertainen mahdollisuus voittaa. Tämä ei kuitenkaan välttämättä tarkoita täyttä symmetriaa.

Kolmantena vuorossa olevat ominaisuudet ovat tärkeät aktiivisuus ja vuorovaikutteisuus. Peliä ei voi vain passiivisesti vastaanottaa, vaan se vaatii osallistumista. Se vaatii, että pelaaja ”ajattelee ja yhdistelee, suunnittelee, tekee päätöksiä, keskittyy, harjoittaa mieltä, vastaanottaa tietoa, oppii ymmärtämään määritellyn järjestelmän vaikutusta, hyväksyy sääntöjä, oppii toimimaan toisten kanssa ja voittamaan ja häviämään, oppii itsestään ja muista, käyttää mielikuvitusta ja harjoittelee taitavuutta ja reaktioita”. Tämä kaikki tiivistyy siihen, että pelatessa väistämättä oppii, jos ei muuta, niin ainakin pelin säännöt.

Viimeisinä Monosen listalla, joka perustuu pitkälti Wolfgang Kramerin ja Chris Crawfordin näkemyksiin, ovat vapaus ja turvallisuus. Mikäli vapaaehtoisuus puuttuu, kyseessä ei ole peli, vaan velvoite. Turvallisuus tarkoittaa tietoisuutta siitä, että pelitilanne on rajattu ja että se, mikä tapahtuu pelin maailmassa, jää pelin maailmaan eikä sen lopputulos juurikaan vaikuta todelliseen maailmaan. Yhteenvetona voi sanoa, että peli on jotakin, joka täyttää suuren osan näistä luetelluista ehdoista ainakin jossain määrin. [3.]

Tietokonepeli ja *videopeli* ovat termejä, joita usein käytetään huolettomasti sekaisin. Kyseessä on siis peli, jota pelataan tietokoneella. Periaatteessa videopeli on yläkäsite, jonka alle mahtuvat myös pelihallien kolikkopelit ja pelikonsolien pelit, mutta rajat ovat melko häilyviä. Joidenkin näkemysten [4 s. 5] mukaan tietokonepeli ja videopeli jopa ovat toisensa täysin pois sulkevia käsitteitä, yhdistävänä yläkäsitteenään elektroniset pelit. Aion kunnioittaa alan perinteitä ja käyttää näitä käsitteitä käytännössä synonyymeinä.

Pelikonsolit ovat käytännössä tietokoneita, joista on riisuttu kaikki ylimääräinen, niin, että ainoa asia, mitä niillä voi tehdä, on pelata pelejä. Viimeisimmät konsolisukupolvet, kuten Sonyn Playstationit ja Microsoftin Xboxit, ovat kuitenkin hieman häivyttäneet tätäkin rajaa tarjoamalla myös muita ominaisuuksia, kuten DVD-elokuvien, musiikki-CD:iden ja muun median toistoa sekä Internetin selailua. Lisälaitteina on nähty jopa, yleensä tietokoneiden erääksi tuntomerkiksi vahvasti miellettyjä, näppäimistöjä.

2.2 Pelinkehityksen lyhyt historia

Videopelien historia ulottuu tutkimuslaitosten ja yliopistojen puolella kauemmaksikin taaksepäin, mutta ensimmäiset kaupalliset viritelmät tulivat markkinoille 1970-luvulla. Vuonna 1972 julkaistu maailman ensimmäinen pelikonsoli Magnavox Odysseus jäi melko tuntemattomaksi, mutta sitä melko pian seurannut Atarin pöytätennispeli Pong saavutti suuren suosion, ja se oli ensimmäinen kaupallisesti menestynyt videopeli. Tämän videopelien ensimmäisen aikakauden voidaan katsoa päättyneen vuoden 1983 romahdukseen, jonka symboliksi ovat nousseet miljoonat New Mexicon autiomaan alle tarpeettomina haudatut Atarin pelikasetit. Muitakin maininnan arvoisia koneita ja pelejä kuin tässä luetellut olisi, mutta siirryn eteenpäin, sillä tämän aikakauden yksityiskohtaisempi käsittely ei ole tämän työn näkökulmasta oleellista. [5.]

1970- ja 1980-lukujen vaihteessa etualalle tulivat niin sanotut kotitietokoneet, jotka ovatkin huomattavasti kiinnostavampia tämän työn näkökulman kannalta, sillä ne antoivat mahdollisuuden myös loppukäyttäjälle kehittää ohjelmia ja sovelluksia. Esimerkkeinä aikakauden koneista voisivat olla Apple II (1977), Commodore 64 (1982) ja kuvassa 1 nähtävä Sinclair ZX Spectrum (1982). Monet pelialan nykyisistä suurista nimistä, kuten Sid Meier, Will Wright ja Richard Garriott, aloittivat uransa näiden tietokoneiden parissa. Kehitysryhmät ja itse projektitkin olivat tässä vaiheessa vielä yleensä hyvin pieniä. Sierra On-Linen perustajat Roberta ja Ken Williams kehittivät pelinsä Mystery House kahdestaan keittiön pöydän ääressä. Steve Wozniak kehitti *arcade*-klassikko Breakoutin ensimmäisen version yksinään neljässä päivässä. [6.]

Kuva 1. Sinclair ZX Spectrum, kotitietokone vuosikymmenten takaa [7].

1980-luvun lopun ja 1990-luvun alun aikana varsinaiset kotitietokoneet hävisivät markkinoilta ja myös käytöstä. Avoimen arkkitehtuurin niin sanottujen IBM PC -yhteensopivien tietokoneiden noustua valta-asemaan ei enää ollut merkittävää teknistä eroa kotien ja yritysmaailman käyttämien koneiden välillä. Tämä kausi jatkuu vielä nykyäänkin.

Yleisen tietoteknisen kehityksen ja varsinkin tallennuskapasiteetin kasvun seurauksena peliprojektien vaatima työmäärä on jatkuvasti kasvanut. Tässä on merkittävänä osallisena myös alalla säännöllisesti toistuva pyrkimys elokuvalliseen ilmaisuun. Budjetit nousevat säännöllisesti kymmeniin miljooniin euroihin tai korkeammallekin, ja jopa puolen miljoonan euron projektia pidetään pienenä [8]. Biowaren kehittämää ja jo vuonna 2002 julkaistua Neverwinter Nights -peliä oli parhaimmillaan tekemässä 72 täysipäiväistä kehittäjää, ja sitä tehtiin viisi vuotta [9]. Eikä tämä edes ole ääriesimerkki alalta.

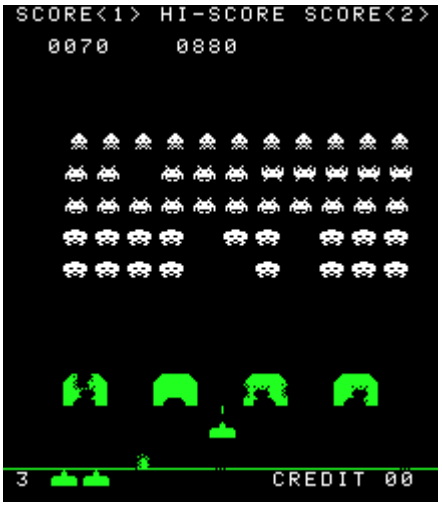
Mammuttitaudin vastavoimaksi on ollut havaittavissa myös suuntausta pienempään. Niin sanottu väliohjelmisto eli middleware on yksi keino vähentää tarvetta keksiä pyörää aina uudestaan jokaista uutta projektia varten. Toisten, täysin ulkopuolisten tahojen tekemän koodin käyttämistä on pidetty helppona ja hieman halveksittavana ratkaisuna, mutta nyttemmin siitä on tullut paljon hyväksyttävämpää. Markkina-analyytikko Wanda Meloni arvioi vuonna 2002, että noin kahdeksan prosenttia peleistä käytti middlewarea. Vuoden 2009 vastaava luku oli 55 prosenttia [10]. Varsinaisen middlewaren lisäksi tar-

jolle on tullut myös erilaisia kehitysympäristöjä, jotka mahdollistavat sen, ettei tätä pelien vertauskuvallista talonvalmistusta tarvitse aloittaa atomitasolta keräämällä savea tiiliä varten, vaan rakennuspalikat ovat paljon pidemmälle jalostettuja. Yksi esimerkki niistä on XNA, jota tarkastelen lähemmin myöhemmin ja jota käytin tämän työn esimerkkiprojektin tuottamiseen.

2.3 Eri videopeligenreistä

Tietokonepeleihin on vuosien saatossa muodostunut monia erilaisia genrejä, tai lajityyppejä. Jaottelu tehdään käytetyn pelimekaniikan mukaan. Sen sijaan vaikka se, minkälaisessa ympäristössä – esimerkiksi avaruus, villi länsi tai lukion välitunti – toimitaan tai minkälaisia kerronnallisia elementtejä mahdollisen tarinan kertomisessa käytetään, on yleensä melko irrelevanttia lajityyppimäärittelyn kannalta. Peli voi olla tiukasti yhden genren edustaja, mutta yhtä hyvin se voi olla yhdistelmä kahta tai useampaa, eräänlainen hybridi. Se voi myös olla mihinkään valmiiseen määrittelyyn sopimaton ja muodostaa aivan uuden genren. Kuten aina, tarkkojen rajojen veto lajityyppien välille on melko väkinäistä ja tulkinnanvaraista. Tulkinta ja jaottelu, johon pohjaan tämän eri videopeligenrejen läpikäynnin, on Robert T. Bakien. [11.]

Toimintapelit on laaja ylägenre, jonka edustajat sisältävät nimensä mukaisesti nopeatempoista toimintaa. Jotain todennäköisesti räjäytetään, tuhotaan tai tapetaan, kun kyseessä on toimintapeli. Puhtaan reaktiotestin asemasta ne vaativat pelaajaltaan myös tilanhahmotuskykyä ja edes jonkinlaista alkeellista taktiikan tajua. Kuuluisia varhaisia esimerkkejä ja lajityypin uranuurtajia ovat Spacewar (1962), Pong (1972) ja kuvan 2 Space Invaders (1978).



Kuva 2. Space Invaders. Taito. 1978.

Ensimmäisen persoonan ammuntapelit (tai FPS-pelit, englannin sanoista first person shooter) muodostuivat omaksi lajityypikseen 1990-luvun alussa id Softwaren pelien Wolfenstein 3D (1992) ja Doom (1993) (katso kuva 3) myötä, vaikka genreen sinänsä istuvia pelejä oli ilmestynyt aiemminkin. Lajityypin nimen mukaisesti näissä peleissä pelimaailmaa katsotaan ohjattavan pelihahmon silmien kautta. Näissä peleissä mies juoksee sokkelossa ja ampuu kaiken, tai mahdollisesti useampi mies, jotka ampuvat kaiken lisäksi toisensa, sillä moninpeli tuli mukaan jo Doomissa, ja on ollut tärkeä, mutta ei toki välttämätön, osa genren pelejä siitä lähtien.

Kuva 3. Doom. id Software. 1993.

Ammuntaa on myös yleensä maustettu jotenkin: 1990-luvulla etsittiin avainkortteja sokkelon kätköistä ja ratkottiin yksinkertaisia pulmia, jotta päästiin jatkamaan matkaa, 2000-luvulla taas tuijoteltiin elokuvamaisia välinäytöksiä, jotta pelin juoneksi kirjoitettua niin sanottua tusinascifiä tai Tom Clancy -pastissia saatiin vietyä taas hieman eteen-

päin. Sokkelo, jossa mies juoksi, myös laajeni avomaastoksi johon eksyä (Operation Flashpoint, 2001) tai vaihtoehtoisesti kapeni niin sanotusti ”putkeksi” (Call of Duty, 2003), mikä antoi pelinkehittäjille paremman mahdollisuuden tarkemmin määritellä pelikokemus.

FPS-pelit ovat nykypäivänä yksi suosituimmista ja kaupallisesti tärkeimmistä genreistä alalla. Esimerkiksi Treyarchin kehittämää ja Activisionin marraskuussa 2010 julkaisemaa Call of Duty: Black Opsia ehdittiin vajaassa vuodessa myymään jopa 25 miljoonaa kappaletta [12], eivätkä sarjan aiemmat osatkaan kovin paljoa pienempiin lukemiin ole jääneet. Kuvasta 4 näkee, että Black Opsin grafiikka on jo melko realistisen oloista.

Kuva 4. Call of Duty: Black Ops. Treyarch, Activision. 2010.

Tasohyppelyt jatkavat kuvaavien genrenimien listaa. Niissä hypitään tasoilta (englanniksi platform, mistä genren nimi platformer) toisille väistellen vaaroja ja vihollisia. Jälkimmäisiä voi toisinaan myös litistää. Lajityypin kuuluisia ja sitä määritteleviä pelejä ovat esimerkiksi Super Mario Bros. (1985), Pitfall! (1982) ja Sonic the Hedgehog (1991). Alun perin nämä pelit olivat kaikki niin sanotusti sivuttain vieriviä ja maailma, jossa pelin sankari loikki, oli kuvattu sivulta päin kaksiulotteisella grafiikalla (kuva 5), mutta pelityyppi taipui myös kolmiulotteiseksi kohtuullisen kivuttomasti, kun se tuli teknisesti mahdolliseksi (Super Mario 64, 1996).

Kuva 5. Sonic the Hedgehog. Sega. 1991.

Moni nykyinen tasohyppelygenren peli käyttää 3D-grafiikkaa, mutta siitä huolimatta pitää pelimekaniikan tiukasti vain kahdessa ulottuvuudessa, kuten suomalaisen Frozenbyten vuonna 2009 julkaisema Trine, joka näkyy kuvassa 6. Näin peli ei vajoa taisteluksi monimutkaisten kontrollien ja hankalien kamerakulmien kanssa, mikä usein seuraa kolmannen ulottuvuuden mukaan tullessa, mutta se saadaan kuitenkin näyttämään hyvältä.

Kuva 6. Trine. Frozenbyte. 2009.

Seikkailupelit keskittyvät yleensä pelimaailman tutkimiseen, ongelmanratkaisuun ja interaktiiviseen tarinankerrontaan. Ne ovatkin ehkä eri genreistä lähimpänä perinteisiä

kertomataiteen muotoja, kuten kirjallisuutta ja elokuvia. Seikkailupelit voivat olla tekstipohjaisia, kuten Zork (1980), jolloin peli kertoo vain tekstin muodossa pelaajalle, mitä tämän ympärillä tapahtuu ja pelaaja yrittää vastavuoroisesti komentaa peliä tekstikomennoin (”avaa ovi”, ”ota kynä”, ”hyppää kaivoon”) pelin tekstiparserin yrittäessä tulkita näitä komentoja parhaansa mukaan.

Graafiset seikkailupelit taas kuvaavat peliympäristön graafisesti, ja niiden käyttöliittymät ovat useimmiten täysin hiirivetoisia. Yksi esimerkki näistä on kuvan 7 Lucasfilm Gamesin Maniac Mansion (1986). Lajityypin kultakausi osui 1980-luvun loppupuolelle ja 1990-luvun alkuun, jolloin moni tuleva klassikko julkaistiin ja jolloin sitä jopa pidettiin eräänlaisena tietokonepelien kruununjalokivenä.

Kuva 7. Maniac Mansion. Lucasfilm Games. 1986.

Sittemmin seikkailupelien genre on vaipunut marginaaliin, osittain siksi, että jotkin sen aiemmin yksityisoikeutenaan pitämät elementit, kuten tarinankerronta, on otettu käyttöön muidenkin genrejen peleissä. Toisena syynä, jonka Larry-seikkailupeleistä tunnettu Al Lowe esittää [13], voi nähdä sen, että nykyiset tietokoneet eivät enää vaadi käyttäjiltään aktiivista ongelmanratkaisukykyä siinä määrin kuin vielä DOS-aikoina, jolloin konetta komennettiin kirjoittamalla mystisiä lyhenteitä ja komentokehotteen kursori välkkyi uhkaavasti eikä Googlelta voinut kysyä mitään. Niinpä ongelmanratkaisuun

keskittyminen ei enää kiinnosta pelienkään puolella. Tuoreita julkaisuja kuitenkin ilmestyy edelleen, kuten vaikka interaktiiviseksi elokuvaksi kallistuva Heavy Rain (2010) tai perinteisempää linjaa edustava Gray Matter (2010).

Kuva 8. Zelda; A Link to the Past. Nintendo. 1991.

Toimintaseikkailu (action-adventure) on kahden aiemmin mainitun lajityypin yhdistelmä. Se on seikkailupeli, joka sisältää liikaa toimintaa ollakseen puhdas seikkailupeli, tai toimintapeli, jossa varsinaista toimintaa ei ole tarpeeksi, vaan mukana on kaikenlaista haahuilua, puuhastelua ja jopa ongelmanratkaisua. Tämä on erittäin laaja ja hankalasti rajattava lajityyppi, jonka alle mahtuvat niin uranuurtaja Legend of Zelda (1985) jälkeläisineen keijukaismetsissään kuin Grand Theft Auto -sarjakin gangsteripullisteluineen. Se on myös yksi nykypäivän suosituimmista genreistä. Näiden molempien sarjojen myöhemmistä osista on näyte kuvissa 8 ja 9.

Kuva 9. Grand Theft Auto 4. Rockstar Games. 2008.

Tappelupeleissä pelaajien hahmot ottavat mittaa toisistaan tai tietokoneen ohjaamista hahmoista areenamaisissa yksi vastaan yksi -tilanteissa. Peliteknisesti tämä tarkoittaa lukuisien potku-, lyönti-, torjunta-, heitto-, väistö- ja vastaiskukomentojen hallitsemista ja oikea-aikaista käyttöä, jotta vastustajan pelihahmo on se, joka erän jälkeen löytyy nurmen pinnasta.

Yhtenä genren perustan luojista voidaan pitää Konamin vuonna 1985 julkaisemaa ja kuvassa 10 näkyvää Yie Ar Kung-Fua. Genren suuruuden ajan sarjoja 1990-luvun alkupuoliskon tienoilla olivat Street Fighter, Mortal Kombat, Virtua Fighter ja Tekken. Uudempaa tuotantoa taas edustaa esimerkiksi Soulcalibur IV (2008), josta on näyte kuvassa 11.

Kuva 10. Yie Ar Kung-Fu. Konami. 1985.

Näiden reilujen yksi vastaan yksi -tilanteiden versus-pelien lisäksi löytyy myös niin sanottuja beat 'em up -pelejä (vapaasti suomentaen ”turpiin kaikille”), kuten Double Dragon (1987), joissa pelaajan tai pelaajien hahmot etenevät kadulla tai missä etenevätkään ja antavat ”turpiin kaikille”, jotka vastaan tulevat, ja vastaantulijoita piisaa. Beat 'em upien kontrollit ovat yleensä versus-pelejä huomattavasti yksinkertaisemmat. Tappelupelien nykysuosio on melko kapea, mutta toisaalta innokkaiden harrastajien myötä myös vakaa.

Kuva 11. Soulcalibur IV. Namco Bandai. 2008.

Vuoropohjaisten strategiapelien juuret ovat suoraan perinteisissä lautapeleissä, kuten esimerkiksi shakissa. Niissä kaksi tai useampi taho, joko ihmis- tai tietokoneohjauksessa, yrittää saavuttaa asetetut voittotavoitteet suunnitelmia laatien ja niitä toteuttaen, kukin vuorollaan hahmojaan ja yksiköitään liikuttaen.

Periaatteessa kyseessä on siis lautapeli, joka on vain toteutettu tietokoneella, mikä mahdollistaa esimerkiksi hieman monimutkaisemmat säännöt, kun kaikki massiiviset laskutehtävät voi huoletta jättää koneen laskettaviksi. Pelitapahtumia voi myös jättää tarvittaessa pimentoon pelaajilta – esimerkiksi niin, että pelaaja ei näe, mitä armeijoita vihollinen vuorten takana rakentaa – mikä olisi aika hankalaa toteuttaa lautapeleissä. Toteutuksesta saa myös tehtyä näyttävämmän kuin vain pahvilevy ja kasa nappeja. Civilization ja Master of Orion ovat takavuosien klassikoita, ja ensin mainitun muutaman vuoden välein julkaistavat jatko-osat ovat järjestään myynti- ja arvostelumenestyksiä.

Reaaliaikaisessa strategiassa vuoroja ei odoteta, vaan pelin tapahtumat vierivät eteenpäin jatkuvasti, mikä tekee lajityypin peleistä huomattavasti kiihkeätahtisempia kuin rauhallisesti etenevistä vuoropohjaisista strategioista. Komentoja jaetaan kun ehditään. Genren teki tunnetuksi Dune 2 vuonna 1992, ja Command and Conquer ja Warcraft sekä molempien jatko-osat ja jäljittelijät täyttivät markkinat sitä seuraavina vuosina.

Monet nykyiset strategiapelit yhdistelevät näitä kahta lajityyppiä. Creative Assemblyn Total War -sarjassa on sekä vuoropohjaisia että reaaliaikaisia osioita. Paradoxin Europa Universalis -pelit taas etenevät reaaliajassa, mutta pelaaja voi halutessaan hidastaa tai jopa pysäyttää pelin kulun, jos tuntee tarvitsevansa lisää aikaa komentojen antamiseen.

Olen pyrkinyt käymään läpi ainakin ne genret, jotka ovat lähellä sitä, mihin esimerkkiprojektissani yritän päätyä. Tavoitteena, kuten luvussa 5 tarkemmin kerrotaan, on jonkinlainen toimintaseikkailu maustettuna tasohyppelyllä ja tappelupelillä.

Muita videopeligenrejä ovat muun muassa roolipelit, niiden online-versiot eli niin sanotut MMO-pelit, monenlaiset simulaatiot sekä urheilu-, ralli-, rytmi- ja puzzle-pelit. Useimmat genret ovat olleet olemassa jo vuosikymmenien ajan, ja vaikka jonkin genren varhaisten ja nykyisten edustajien ulkomuodot poikkeavatkin toisistaan, niillä on silti hyvin paljon yhteistä. Tätä yritin tuoda hieman ilmi antamalla esimerkkejä usealta vuosikymmeneltä.

2.4 Katsaus nykytilanteeseen

Peliala on murrostilassa monestakin syystä. Ensinnäkin on tapahtumassa konsolisukupolven vaihdos. Sonyn PlayStation 4 ja Microsoftin Xbox One julkaistaan loppuvuodesta 2013. Porrastuksista johtuen julkaisut tapahtuvat joillain markkina-alueilla vasta vuodenvaihteen jälkeen. Nintendon Wii U on jo tullut markkinoille, mutta ei ainakaan vielä ole saavuttanut erityisen merkittävää markkina-asemaa. Nyt väistymässä oleva edellinen sukupolvi on kestänyt poikkeuksellisen pitkään, lähes kahdeksan vuotta. Asiantuntijat yrittävät ennakoida, kuka tämän seuraavan erän voittaa ja jääkö uusin konsolisukupolvi mahdollisesti viimeiseksi, jos pelit siirtyvät kokonaan pilvipalveluihin.

Digijakelun läpimurto on viimein tapahtunut, ja lähes kaikki suuremmatkin pelit saa nykyään ostettua suoraan digitaalisesti. PC-puolella digijakelun mahtitekijä on Valve verkkokauppa Steaminsa kanssa. Sama yritys sekoittaa pelialan tilannetta lisää aikomalla tuoda markkinoille ensisijaisesti pelikäyttöön tarkoitetun Linux-pohjaisen SteamOS-käyttöjärjestelmän ja sitä varten tarkkaan spesifioidun Steambox-koneen. Steamboxin valmistuksen Valve aikoo jättää kolmansille osapuolille. Joka tapauksessa laite kilpailee melkein suoraan perinteisten konsolien kanssa.

Digijakelun kehittyminen on mahdollistanut entistä pienemmille kehitystiimeille pelien tuomisen yleisön saataville. Tätä mahdollisuutta on käytetty ahkerasti hyväksi, kuten näkyy, kun vilkaisee vaikka Applen tai Androidin kauppapaikkaa.

Älypuhelimien kehittyminen on ajanut jo aiemmin perinteiset kädessä pidettävät pelikoneet ahtaalle: miksi ostaa erillistä kannettavaa laitetta pelaamiseen, kun taskussa on jo vastaava, joka toimii satunnaisesti myös puhelimena. Nyt tablettitietokoneet kaventavat välimatkaa puhelinpelien ja perinteisten tietokone- ja konsolipelien välillä.

Applen App Storessa pelien hinnat laskettiin ensin muutamaan euroon, joten nyt seuraavana loogisena askeleena on menty alas nollaan euroon saakka. Rahat otetaan sen sijaan myymällä pelaajille palveluita ja virtuaalitavaroita pelin sisällä. Kyseessä on niin sanottu mikromaksu- tai free2play-systeemi, ja sen tunnetuin edustaja ainakin Suomessa on Supercell pelillään Clash of Clans.

Se, millä tavalla peliala lopulta asettuu aloilleen, on epävarmaa, mutta myös mielenkiintoista seurattavaa. [14.]

3 Pelinkehitysprosessin vaiheita

3.1 Insinöörien ja taiteilijoiden yhteensopivuudesta

Tässä luvussa käydään läpi pelituotannon eri vaiheita ja osa-alueita suunnittelusta aina julkaisuun ja sen jälkeiseen aikaan asti. Pelituotanto on erittäin harvoin täysin lineaarinen prosessi, jossa edetään järjestelmällisesti vaiheesta toiseen. Eri vaiheet menevät päällekkäin ja limittäin, ja niihin joudutaan usein palaamaan myöhemmin tekemään muutoksia jonkin myöhemmässä vaiheessa tapahtuneen yllättävän käänteen vuoksi. Kuten hieman eri alalla toiminut Helmuth von Moltke osuvasti totesi: ”Hyväkään suunnitelma ei kestä ensimmäistä kosketusta vihollisen kanssa” [15]. Osittain tällainen kaoottisuus on ominaista kaikille ohjelmistotuotantoprojekteille, mutta pelialalla se korostuu, sillä tavoiteltu tuotos on aina jonkinlainen tekniikan ja taiteen enemmän tai vähemmän vaikea yhdistelmä. Tuotantovaiheessa tämä voi toisinaan ilmetä eräänlaisena ’insinöörit vastaan taiteilijat’ -vastakkainasetteluna.

Osan kuvatuista vaiheista moni pelituotantoprojekti ohittaa täysin, vaikka toisille ne voivat olla erittäin tärkeitä kokonaisuuden kannalta. Jokainen projekti tekee vain sen, mikä palvelee sen omia tarkoitusperiä. Tässä luvussa kuvattavat eri vaiheet edustavat siis vain eräänlaista abstraktia yleistä pelituotantoprojektia, eivät yhtä tiettyä konkreettista projektia. Tuottajan näkökulmaa kuvaan pääosin Tom Sloperin [16] näkemyksiin pohjautuen.

3.2 Pelituotanto tuottajan näkökulmasta

Pelintuotantoprojekti käynnistyy ja etenee vaiheittain ja moni tuotannossa lopulta mukana oleva liittyy siihen vasta siinä vaiheessa, kun hänen taitojaan tarvitaan, ja vastaavasti siirtyy sivuun, kun niitä ei enää tarvita.

Ensimmäisessä, eli konseptivaiheessa, mukana kuitenkin aina on – mahdollisesti projektin ensimmäisenä jäsenenä – tuottaja, joka muun muassa päättää projektin suurista linjoista ja pitää huolta yhteydenpidosta eri sidosryhmien suuntaan. Tuottajan tehtävänimikkeenä voi yrityksestä riippuen olla jokin muukin, esimerkiksi projektipäällikkö, mutta tehtävänkuva on kuitenkin samansuuntainen.

Konseptivaiheen tarkoitus on saada aikaiseksi konseptidokumentti luotavasta pelistä. Konseptidokumentilla on muitakin tarkoituksia, mutta tässä vaiheessa sen tärkein tehtävä on esittää visio lopullisesta pelistä, jotta kaikki projektiin mukaan lopulta lupautuvat tahot ymmärtävät, mitä oikein ollaan tekemässä. Sen on toki oltava myös tarpeeksi vetoava, jotta kaikki tarvittavat tahot, kuten vaikka rahoittajat, ylipäätään suostuvat mukaan. Aloite konseptin luomiseen voi tulla julkaisijalta, joka haluaa esimerkiksi uuden jatko-osan hyvin myyneeseen pelisarjaansa tai uuteen hittielokuvaan perustuvan pelin, jonka tekemiseen se on juuri ostanut lisenssin elokuvastudiolta. Se voi olla myös jokin alkuperäinen idea pelistudion sisältä tai joltakin täysin ulkopuoliselta.

Konseptidokumentin kirjoittaa tuottaja tai pelisuunnittelija, mutta sen sisältöä varten on syytä konsultoida kaikkia tuotantoon osallistuvia. Lisenssipelin kohdalla tämä tarkoittaa erityisesti lisenssinhaltijaa ja konsolipelien kohdalla alustanhaltijaa, eli esimerkiksi Xboxin tapauksessa siis Microsoftia. Määriteltäviin asioihin kuuluvat muun muassa peliprojektin kohdealusta, maantieteellinen kohdealue, alustavat aikataulut ja budjetit.

Kun dokumenttiin on kerätty kaikkien hyvät ideat ja toiveet, se tiivistetään mahdollisimman lyhyeksi, jotta se ei pitkästyttäisi kiireisiä johtoportaan edustajia, joille se esitellään tapaamisessa, jossa on tarkoitus päättää, kannattaako koko projektin edes edetä varsinaisiin tuotantovaiheisiin saakka. Jos ja kun tuottaja onnistuu tässä kokouksessa vakuuttamaan rahoittajat siitä, että hänen projektillaan on tarpeeksi hyvät onnistumisen mahdollisuudet ja että siihen kannattaa sijoittaa julkaisijan resursseja, projektille niin sanotusti näytetään vihreää valoa ja se voi siirtyä konseptivaiheesta eteenpäin varsinaisiin tuotantovaiheisiin. [16.]

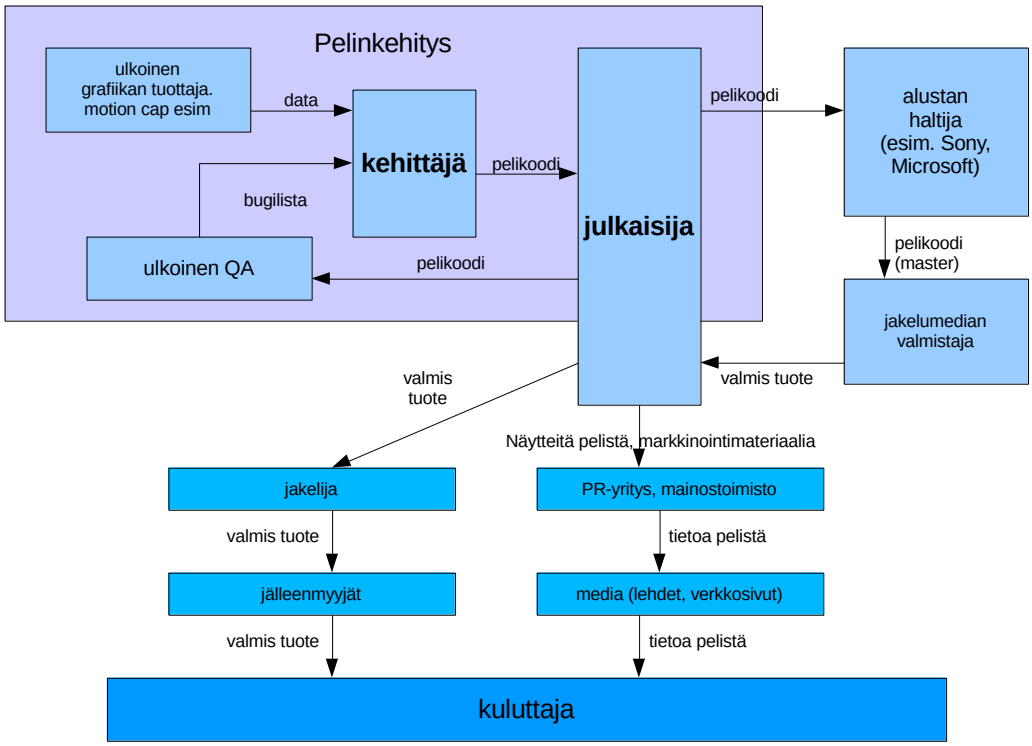
Tuotantovaiheen voi jakaa kolmeen eri vaiheeseen, jotka ovat esituotanto, tuotanto ja jälkituotanto. Esituotanto sisältää konseptia huomattavasti yksityiskohtaisempaa suunnittelua ja projektitiimin kokoamista, tuotannon aikana peli varsinaisesti luodaan, ja jälkituotanto sisältää ohjelmointivirheiden etsimistä ja korjaamista ja mahdollisten pelipaketin, ohjekirjojen ja muun oheistavaran tuottamista.

Esituotantovaiheessa tuotettavien dokumenttien olisi periaatteessa tarkoitus olla niin tarkasti kirjoitettuja, että kaksi eri kehitystiimiä päätyisi niitä käyttäen samoihin lopputuloksiin. Tärkeimmät dokumentit ovat suunnitteludokumentti (game design document, GDD), joka kertoo ja kuvailee yksityiskohtaisesti, mitä on tuotettava, ja tekninen dokumentti (technical design document, TDD), joka kertoo, mitä tekniikoita käytetään. Lyhy-

esti ja yksinkertaistaen: suunnitteludokumentti määrittelee ongelman ja tekninen dokumentti sen ratkaisun. Yksityiskohtaisen budjetin ja aikataulun pitäisi olla myös tässä vaiheessa selvillä.

Tuotantovaiheessa varsinainen työ viimein alkaa. Siitä enemmän alaluvussa 3.3, mutta tärkeää tässä vaiheessa on pitää hallinnassa se valtava määrä koodia ja pelin osasia (assets), joka nyt alkaa tuotantotiimin toimesta syntyä.

Mukaan tulevat ajallaan myös testaajat, laadunvalvonta (quality assurance, QA) ja markkinointi. Laadunvalvonta voi olla kehitysstudion oma sisäinen, mutta se voi olla myös ulkopuolisen tahon suorittamaa. Kuvassa 12 esitellään, mitä kaikkia tahoja voi olla mukana pelin saattamisessa yleisön saataville.



Kuva 12. Konsolipelin matka kehittäjältä kuluttajalle [16].

Jälkituotannossa ollaan, kun pelistä on versio, jossa on kaikki koodikin mukana ja siitä enää korkeintaan korjataan löytyvät ohjelmointivirheet. Tätä versiota on tapana sanoa beta-versioksi, vaikka samaa nimikettä saatetaan käyttää aivan toisissa vaiheissa ole-

vistakin peliprojekteista. Jälkituotantoon kuuluvat myös pelin mahdollinen lokalisointi, pelilevyjen painatus ja jakelu sekä kaikki mahdollinen markkinointiin liittyvä.

Projekti ei yleensä kehittäjienkään osalta lopu siihen, että ohjelma saadaan valmiiksi ja siirrettyä julkaisijan huoleksi. Karkeasti voidaan sanoa, että ohjelma, tai tässä tapauksessa peli, voidaan mieltää joko tuotteeksi tai palveluksi, ja näiden kahden ääripään välillä on monia erilaisia yhdistelmiä. Ohjelmatuote ei vaadi kuin minimaalisen panostuksen julkaisun jälkeen, esimerkiksi selkeimpien virheiden korjaamisen. Palvelutyyppinen peli, esimerkiksi kuukausimaksullinen online-peli, saattaa vaatia kymmenittäin täysipäiväisiä työntekijöitä pitämään yllä infrastruktuuria, tarjoamaan käyttäjätukea ja jopa varsinaisia kehittäjiä tuottamaan uutta sisältöä peliin.

3.3 Kehityksen osa-alueet ja kehittäjien roolit

Peliä varten pitää rakentaa kaikenlaista, kuten esimerkiksi grafiikka, äänet, hahmomallit, 'pelimoottori', tekoäly, logiikka, fysiikkamallinnus, syötteen vastaanotto ja palaute (input/output) ja myös työkalut niiden luomiseen. Kaiken tämän tekevät pelin kehittäjät, jotka voidaan karkeasti jakaa ohjelmoijiin ja sisällöntuottajiin.

Kehitystoiminnan tärkein rooli on ohjelmoija, ja varhaisten projektien yhden hengen tiimit muodostuivat – jos yhden hengen joukosta tuollaista sanaa voi käyttää – juuri ohjelmoijasta, joka hoiti ohessa muut tehtävät ja roolit miten parhaiten taisi.

Tuotantoryhmien kasvettua roolit ja tehtävät jakautuivat omille erikoisosaajilleen ja ohjelmoijan toimenkuva muuttui siten, ettei ohjelmoija enää ollut kokonaisuutta hallitseva visionääri ja auteur. Ohjelmoijasta tuli enemmänkin tukirooli, joka antoi työkalut ja rakensi kivijalan, joiden avulla erilaiset erilliset sisällöntuottajat tuottivat kaiken sen, minkä loppukäyttäjä näkee peliä pelatessaan.

Ohjelmointityön voi jakaa kolmeen toisistaan selkeästi erottuvaan osa-alueeseen: pelikoodiin, pelimoottoriin ja työkaluihin. Tällainen tuotetun koodin perusteella tehty jako on tietenkin vain yksi mahdollisista jakotavoista. [17.]

Pelikoodi on se, mitä ihmiset yleensä ajattelevat, kun puhutaan peliohjelmoinnista. Se sisältää kaiken suoraan itse peliin liittyvän, kuten esimerkiksi sen, millä tavalla kamera

käyttäytyy, kun hiirtä liikutetaan, tai sen, miten tekoälyoliot reagoivat tiettyihin tapahtumiin pelimaailman sisällä.

Tämän alueen ohjelmointi sisältää usein runsaasti skriptikielisen koodin tuottamista. Varsinainen pelikoodi voidaan siis usein kirjoittaa esimerkiksi vaikka Luan kaltaisella skriptikielellä, kun muu osuus kirjoitetaan jollain C++:n tapaisella. Skriptikielen käytön tarkoituksena on mahdollistaa nopeampi iterointi esimerkiksi pelimekaniikan toteuttamisessa ja antaa teknisesti suuntaantuneemmille sisällöntuottajille keinot näiden testattavien muutosten tekemiseen omin päin ilman tarvetta kierrättää jokaista iteraatiota varsinaisten ohjelmoijien kautta.

Skriptikielinen koodi on yleensä sen verran selkokielistä, että se on jokaisen vähänkin teknisesti orientoituneen työryhmän jäsenen luettavissa. Suunnittelija voi esimerkiksi helposti löytää koodista vaikka kohdan, joka säätelee pelaajan ohjaaman hahmon liikkumisnopeutta, ja itsekseen kokeilla siihen eri arvoja, kunnes löytää hakemansa.

Pelimoottori tässä kontekstissa tarkoittaa kaikkea sitä koodia, joka päätyy lopulliseen sovellukseen, mutta ei ole spesifistä juuri tätä pelisovellusta varten kirjoitettua koodia. Tähän kuuluu kaikki se koodi, joka piirtää näyttävän grafiikan ruudulle, saa muhkeat ääniefektit jytisemään, hoitaa tarvittavan verkkoliikenteen, laskee tekoälyn tarvitsemat reitinhakualgoritmit, simuloi fysiikkamallinnuksen ja niin edelleen.

Pelimoottoria voi ajatella eräänlaisena tukirakenteena, jonka varaan varsinainen pelikoodi on rakennettu. Sen yhtenä tehtävänä on eristää pelikoodi alustasta (eli käyttöjärjestelmästä ja fyysisestä laitteistosta), jolla peliä on lopulta tarkoitus ajaa. Pelikoodin olisi tarkoitus olla täysin tietämätön alustasta. Pelimoottorin voi siis nähdä abstraktiokerroksena pelikoodin ja laitteiston välissä. (Tässä yhteydessä kaikki pelimoottorin ja laitteiston väliset ohjelmalliset abstraktiokerrokset mielletään kuuluviksi laitteistoon.) Erilaiset middleware-tyyppiset ohjelmistot saattavat vähentää, tai jopa kokonaan poistaa, tähän osioon kuuluvaa ohjelmointityötä. Niistä enemmän luvussa 4.

Kolmantena osa-alueena ovat *työkalut*, kuten kenttä- ja äänieditorit. Näillä työkaluilla sisällöntuottajat tekevät suurimman osan työstään. Ne saattavat olla täysin projektin ohjelmoijien tekemiä tai sitten suoraan kaupan hyllyltä haettuja 3ds Maxeja ja Photoshopeja. Jälkimmäisen vaihtoehdon kohdalla ohjelmoitavaa ei paljon tietenkään jää.

Ehkä yleisin toimintatapa kuitenkin on käyttää valmisohjelmia, mutta kirjoittaa niihin itse laajennuksia (plugin) oman projektin tarpeiden mukaan. Sisällöntuottajat pystyvät näin käyttämään vakaata ja itselleen ennalta tuttua ohjelmaa (esimerkiksi Photoshop) talon sisällä hätäisesti kyhätyn todennäköisesti dokumentoimattoman viritelmän sijaan, eikä ohjelmoijien tarvitse tuottaa muuta kuin laajennuksen vaatima lisätoiminnallisuus, kun käyttöliittymä ja perustoiminnot jo löytyvät alkuperäisestä ohjelmasta. Aina tietenkään valmisratkaisut eivät ole edes laajennuksien kanssa riittäviä käsillä olevaan tarpeeseen. Työkaluihin kuuluvat myös yksinkertaisia tehtäviä automatisoivat skriptit, tiedostoformaattikonvertterit ja koodintestausohjelmat. [17, s. 168.]

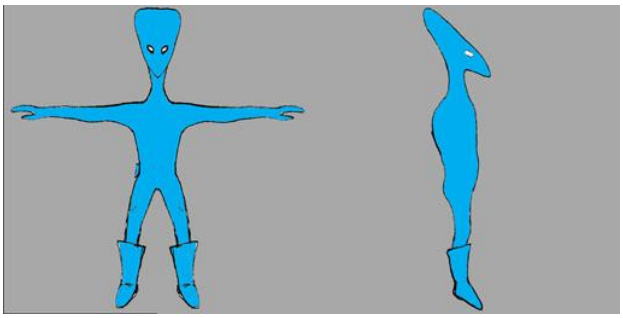
Peliohjelmoijilla on tilaa erikoistua, sillä esimerkiksi vaikka grafiikan, verkkotekniikan, äänien ja tekoälyn ohjelmoiminen saattavat kukin vaatia hyvinkin erilaisia taitoja. Niitä kaikkia tarvitaan peliprojektissa, ja ne pitäisi saada myös toimimaan sujuvasti yhdessä. Kaikesta vähän tietävälle yleisosaajalle on siis myös käyttöä monien erikoistuneempien ohjelmoijien tuotosten yhteen saattajana.

Sisällöntuottajat luovat kaiken sen, minkä ohjelmoijien koodi saa ruudulla lopulta elämään. Heihin kuuluvat muun muassa graafikot ja artistit, joihin taas kuuluvat 3D-mallintajat, tekstuuriartistit ja animoijat. Käsittelen esimerkkinä tehtäviä, joita liittyy 3D-mallintamiseen.

Hyvin suuressa osassa nykyisiä pelejä pelimaailma esitetään lähes yksinomaan 3D-grafiikan avulla. Poikkeuksen muodostavat lähinnä yksinkertaisemmat mobiili- ja selainpelit, ja, mitä suuremman budjetin projekteihin mennään, sitä varmempaa on, että pelin grafiikka on pelkkää 3D:tä. Kolmiulotteisen grafiikan tekee 3D-mallintaja. Tämä on taas yksi rooli, joka vaatii sekä taiteellista silmää että teknistä osaamista. 3D-mallintaja on kuvanveistäjä, teknikko, taiteilija ja insinööri, joka pyrkii siihen, että hänen luomansa mallit ovat ilmaisuvoimaisia, haetun tyylin mukaisia ja tehokkaita. Niissä ei ole käytetty liikaa polygoneja, jos niitä ei tarvita, ja ne ovat topologialtaan järkeviä ja polygonimääriltään määriteltyjen rajojen sisällä. Polygonimallinnus on ylivoimaisesti suosituin nykyisin käytetyistä mallinnustavoista, mutta muitakin metodeja toki on, kuten esimerkiksi vokselimallinnus. [18, s. 657.]

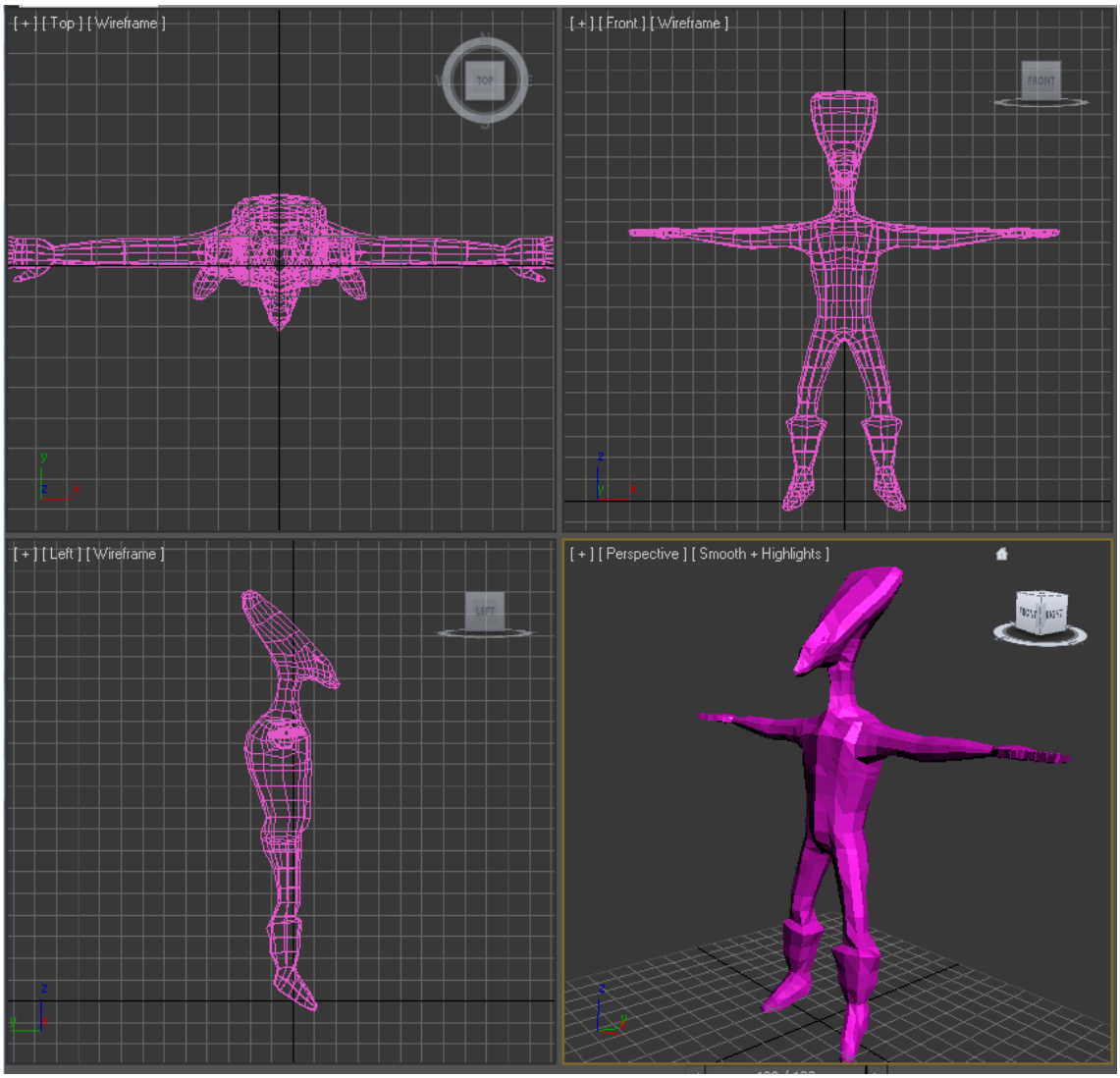
Esimerkkinä mallinnuksesta käydään läpi pintapuolisesti tämän työn esimerkkipelin Nopean Kuhan erään pelihahmon luominen. Tämä hahmo on se, jota pelaaja ohjaa, ja se on työnimeltään Exon. Varsinaisessa pelissä nimeksi vaihtuu Kuha.

Olettaen, että mallintajalla on jo jonkinlainen käsitys siitä, minkälaista hahmoa hän on lähdössä luomaan, on ensimmäisenä työvaiheena hahmon luonnosteleminen. Luonnolliseen asentoon piirretty luonnos antaa paremman kuvan siitä, minkälainen hahmo on, mutta mallintamisen referenssikuvana suoraan edestä ja sivulta kuvattu jäykässä krusifiksiasennossa seisova hahmo (kuten kuvassa 13) on huomattavasti hyödyllisempi.



Kuva 13. Exon (eli Kuha) luonnosteltuna kynällä, skannattuna ja ääriviivojen sisäinen alue väritettynä.

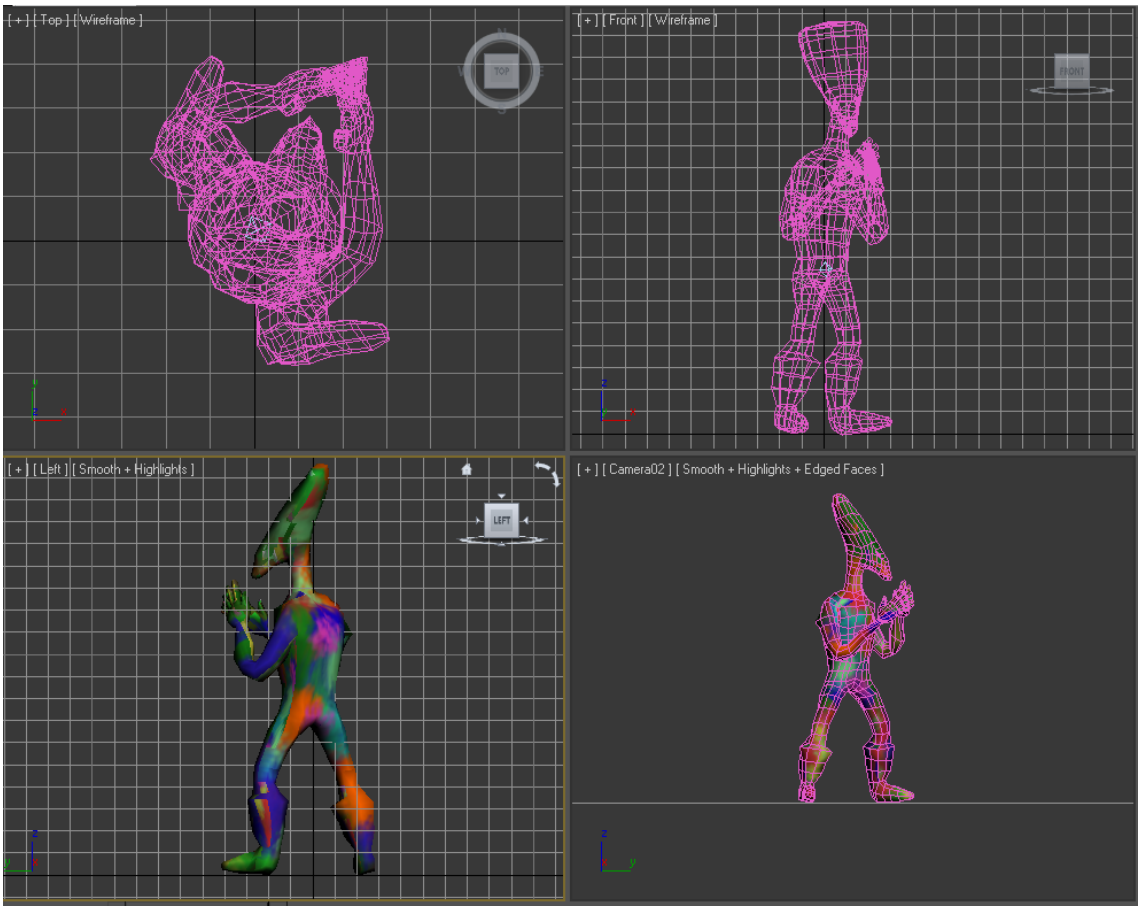
Niin sanottu laatikkomallintaminen eli box modeling on yksi suosittu mallinnustapa. Mallinnusohjelman näyttämölle (scene) ladataan edellisen vaiheen referenssikuvat ja luodaan yksi polygonilaatikko. Tätä laatikkoa aletaan sitten manipuloida moninaisin tavoin: venyttää, leikata (niin että yhden pinnan muodostavasta polygonista tehdään kaksi polygonia), 'työntää ulos' objektin (laatikosta on huono puhua jo lyhyen vääntelyn jälkeen) yksittäisiä tahkoja, säätää yksittäisten verteksien (polygonien kärkipisteiden) sijaintia ja niin edelleen. Tämän työlään työvaiheen päätteeksi alun kahdentoista polygonin laatikosta pitäisi olla tullut jotakin referenssikuvien mukaista, kuten kuvassa 14 on onnellisesti käynyt.



Kuva 14. Rautalankamalli eli wireframe kuvattuna neljästä suunnasta mallinnusohjelmassa.

Seuraavatkin vaiheet saattavat olla sangen työläitä. Aikaansaatu malli on vain tyhjä kuori, joka pitää kiinnittää eli ’rigata’ (rigging) kiinni yksinkertaisempaan ja helpommin animoitavaan luurankorakenteeseen. Se on myös täysin väritön (tai yksivärinen), joten se vaatii teksturointia pintaansa. Exonin luurankona on 3ds Maxin vakio kaksijalkainen luuranko. Pintana taas on tarkoitus olla pelkkä värien sekamelska, jossa ei ole niin tarkkaa, onko sininen läiskä otsassa vai polvessa, joten tällä kertaa joitakin mutkia saadaan vedettyä suoremmiksi. Animointi tehdään vain kuoren sisällä olevalle luurangolle. Kuori, joka on siis ainoa osa, jonka käyttäjä lopulta näkee, taipuu ja venyy luurangon mukana enemmän tai vähemmän täydellisesti. Tulos on sitä parempi, mitä paremmin aikaisempi riggaus on tehty. Exonin kohdalla onnistuttiin melko hyvin, kuten kuvasta 15 voi nähdä, ja ainoat isommat virheet ovat aina vaikeilla olkapäiden seuduilta. Nämä vaiheet ovat useimmissa vähänkään isommissa projekteissa kaikki eri ihmis-

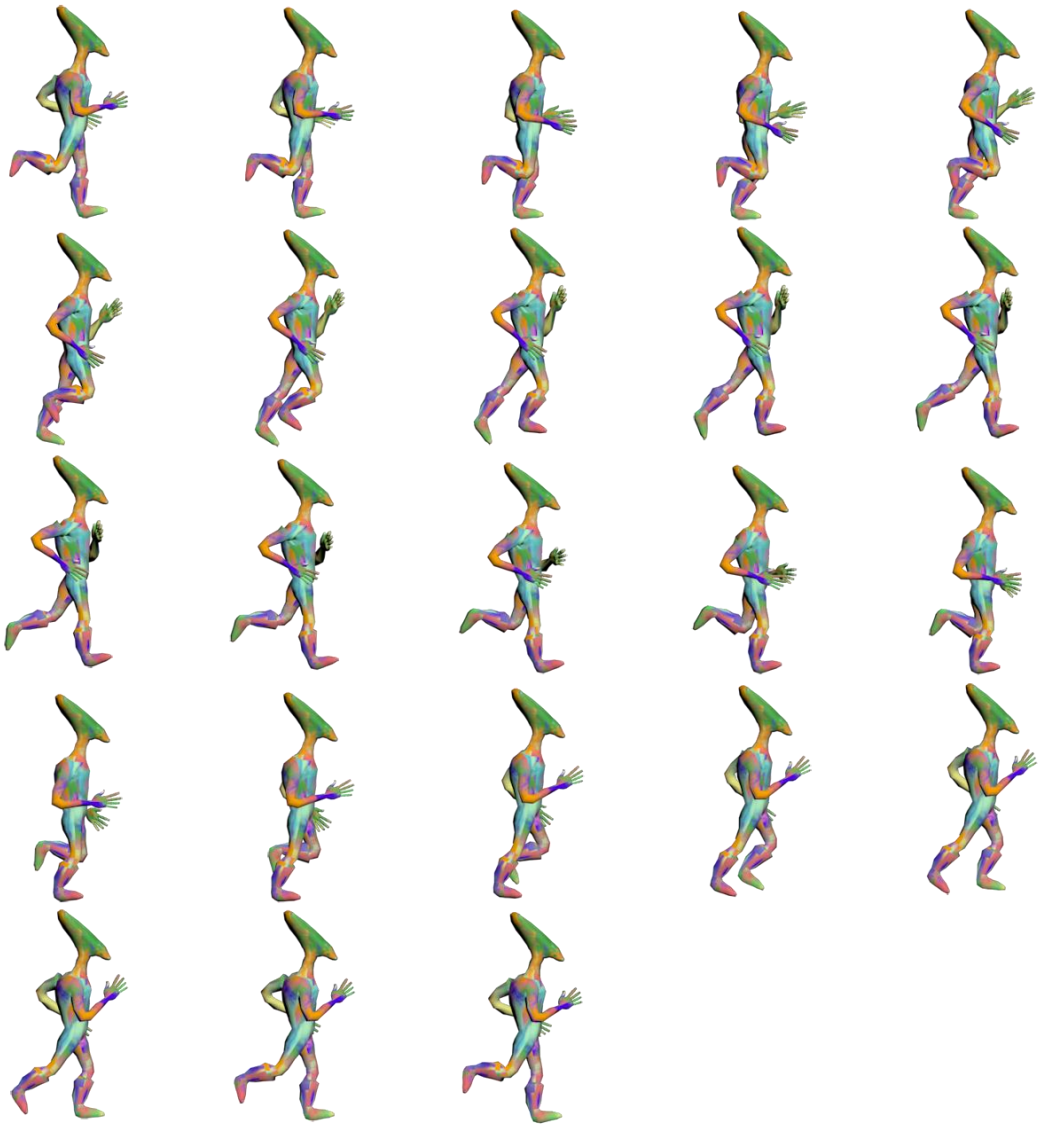
ten vastuualueilla. Mallintaja mallintaa, tekstuuriartisti teksturoi ja animoija animoi (ja tuntee myös nimen animaattori). Grafiikan tuottaminen on työvoimaintensiivistä, ja työvoiman olisi siis toivottavaa olla sekä taiteellisesti että teknisesti lahjakasta, joten ymmärrettävästi se ei ole kovin halpaakaan. Se onkin yksi alue, joka helposti ulkoistetaan varsinaisen tuotantotiimin ulkopuolelle, jos ei kokonaan, niin ainakin osittain.



Kuva 15. Rigattu, teksturoitu, ja animoitu Exon poseeraa kameroiden keskellä.

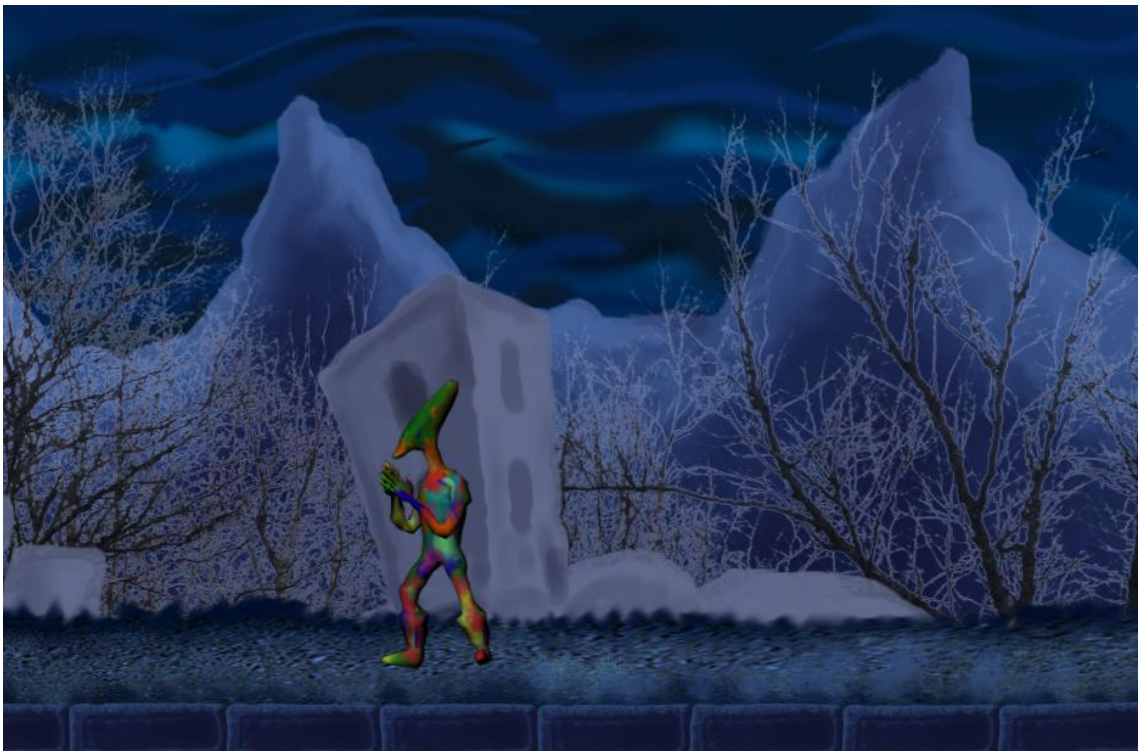
Tässä vaiheessa, jos Nopea Kuha olisi 3D-peli, malli tekstuureineen ja muine oheistiedostoineen tuotaisiin ulos mallinnusohjelmasta. Näin ei kuitenkaan ole, vaan Nopea Kuha tarvitsee kaksiulotteisia bittikarttakuvia hahmosta. Sitä varten hahmoon ripustetaan mallinnusohjelmassa kamera, jolla otetaan kuvia sopivissa kohdissa eri animaatioita. Nämä kuvat, tai spritet, kootaan muutamalle sprite sheetille, jotka voidaan sitten ladata käyttöön peliohjelmassa. Yksi tällainen spritesheet, joka sisältää Exonin juoksuanimaation tarvitsemat kuvat, on kuvassa 16. Tämä useiden kuvien kokoaminen yhteen kuvatiedostoon on ollut tapana tehdä suorituskykysyistä. (Se vaatii vähemmän

I/O-operaatioita, näytönohjaimen muistiin mahtuu vain tietty määrä tekstuureja, joten se saattaa joutua lataamaan niitä muistiinsa koko ajan uudestaan ja niin edelleen.)



Kuva 16. Esimerkki Kuhan juoksuanimaatiosta.

Kun kaikki spritet on ladattu sovellukseen, niitä pääsee käyttämään vapaasti: juoksuanimaation kuvia näytetään, kun pelihahmoa liikutetaan, lyöntianimaation, kun hahmoa käsketään lyömään. Se, miten tämä tarkemmin tapahtuu, nähdään luvussa 5, missä tarkastellaan hieman Nopean Kuhan koodia.



Kuva 17. Nyt on Kuha pelissä.

Seuraavaksi tarkasteluun otetaan keinot, joilla pelintekijöiden, niin ohjelmoijien kuin sisällöntuottajienkin, työmäärää voidaan keventää.

4 Kehitysympäristöt ja väliohjelmistot

4.1 Väliohjelmistot

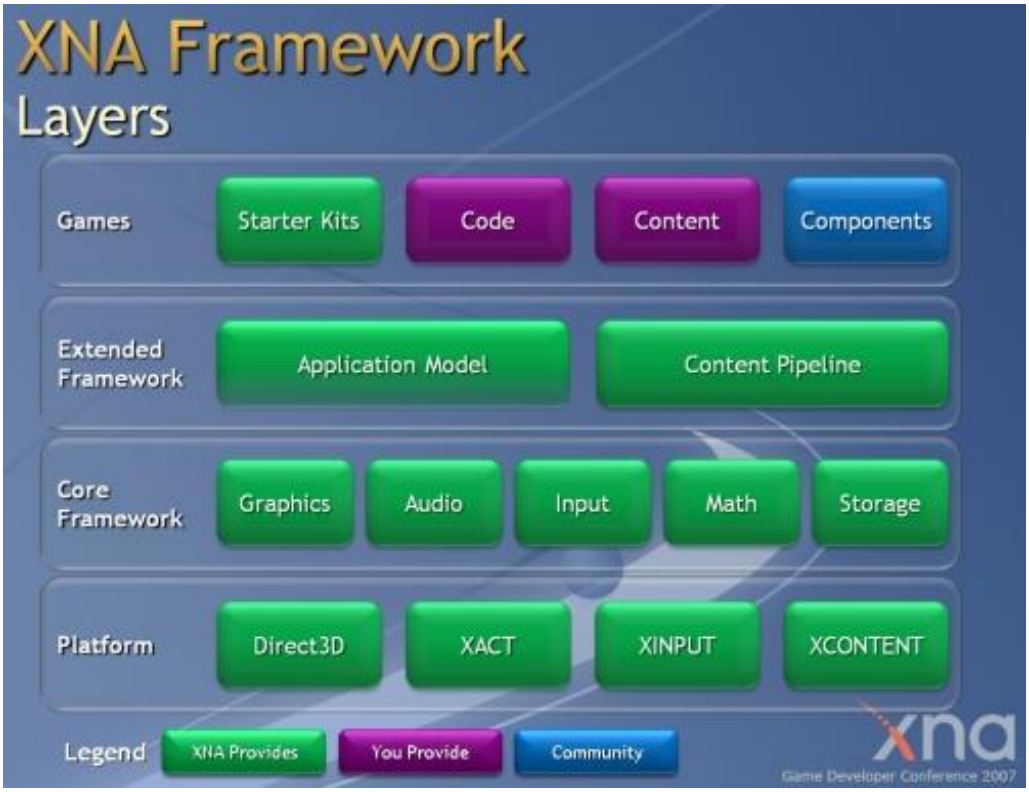
Väliohjelmisto eli middleware on laaja ja vähemmän täsmällinen termi. Sanakirjamääritelmä sille on muun muassa ”ohjelma, joka toimii käyttöjärjestelmän ja sovellusten välissä ja jonka tehtävänä on mahdollistaa useiden eri lähteistä peräisin olevien ohjelmien toimiminen yhdessä”. Pelinkehityksen yhteydessä sillä kuitenkin yleensä tarkoitetaan pelimoottoria tai joitakin sen yksittäisiä osia, jotka valmiissa peliohjelmassa pitävät huolta jonkin yksittäisen osa-alueen toteutuksesta, kuten esimerkiksi äänen tai kuvan renderöinnistä, fysiikkamallinnuksesta, animaatiosta, tekoälystä, streamauksesta tai verkkoliikenteestä. Pelinkehittäjän näkökulmasta kyseessä ovat palikat, jotka mahdollistavat pelin saamisen valmiiksi ja markkinoille edes jotenkin järkevässä aikataulussa. Joillain väliohjelmistoilla hallittu alue on hyvinkin erikoistunut, mutta ne hoitavat sen niin hyvin, että ovat kuitenkin erittäin suosittuja (esimerkiksi SpeedTree). Toiset taas ovat todellisia yleistyökaluja, jotka hoitavat lähes kaiken (Unreal-enginet). Hyvä middleware on modulaarista ja uppoaa olemassa olevaan koodiin ilman suurempaa päänvaivaa.

Moni pelialan toimija ei tee itse pelejä lainkaan, vaan keskittyy ainoastaan näihin erilaisiin väliohjelmistoihin. Yksi tällä hetkellä merkittävimmistä näistä niin sanotuista lapiokauppiaista pelialalla on Unity ja sen kauppaamat pelimoottori ja kehitysympäristö. Jopa itse Microsoft hylkäsi oman XNA:nsa ja siirtyy tukemaan Unitya [19; 20]. Seuraavaksi otan tarkasteluun XNA:n, sillä vain kuolleet kuhat uivat myötävirtaan.

4.2 Microsoft XNA – esimerkki kehitysympäristöstä

XNA on Microsoftin tuottama kehitysympäristö pelinkehittäjien käyttöön. Nimenä XNA on vain humoristinen itseensä viittaava lyhenne ”XNA's Not Acronymed”, joka ei sinänsä tarkoita mitään. Julkaisua edeltänyttä nimeä Xbox New Architecture ei otettu koskaan käyttöön, sillä nimike Xbox haluttiin jättää vain pelikonsolin käyttöön. Tämä nimeämistyyli, johon nimenvaihdon yhteydessä päädyttiin, on Unix-maailmasta tuttua, joten XNA:n nimen voi nähdä Microsoftin yrityksenä rakentaa uutta imagoa. Itsenäisten ohjelmistokehittäjien ystävä olisi mieluisampi mielikuva kuin ahne suuryritys.

XNA-kehitysympäristö muodostuu useasta osasta. Tärkein osa, eli XNA-ohjelmistokehys, on rakennettu Microsoftin oman .NET-ohjelmistokehyksen päälle. .NET-kehyksen laajat luokkakirjastot helpottavat tuotetun koodin uudelleenkäyttöä, eikä pyörää niin sanotusti tarvitse keksiä uudestaan kovin montaa kertaa. Ajoympäristö (Common Language Runtime, CLR) ja virtuaalikone kääntävät käytetyn koodin lopulta laitteistokohtaiseksi natiivikoodiksi, mikä mahdollistaa saman koodin käytön kehitettäessä ohjelmistoa usealle erilaiselle ympäristölle. XNA-kehys lisää tähän erityisesti pelinkehityksessä käytettyjä luokkia ja komponentteja. Yksinkertainen graafinen esitys ohjelmistokehyksen rakenteesta on kuvassa 18.



Kuva 18. XNA-ohjelmistokehyksen eri kerrokset [21].

.NET ymmärtäisi useampiakin ohjelmointikieliä, mutta XNA:n kanssa käytössä on ainoastaan C# (C sharp). Microsoftin kehittämä C# on yleiskäyttöinen oliopohjainen ohjelmointikieli. Sen syntaksi on melko lähellä C++:aa, joka on ollut alalla standardina jo useiden vuosien ajan, mikä tekee ohjelmoijien siirtymisen C#:n käyttäjiksi melko kivuttomaksi.

Toinen osa on XNA Game Studio, joka on kokoelma pelien kehittämiselle oleellisia työkaluja. Ne asennetaan Visual Studion, Microsoftin ohjelmankehitysympäristön, pääl-

le, jolloin ne integroituvat osaksi sitä. Työkaluihin lukeutuu muun muassa niin sanottu sisältöputki (content pipeline), joka helpottaa huomattavasti peliprojektin resurssien eli esimerkiksi graafisten komponenttien ja äänitiedostojen organisointia ja hallintaa. Saattaa aluksi kuulostaa melko mitättömältä, mutta kun ajattelee, millaisia tietomääriä peliprojekteihin tarvitaan, tämänkaltaisia apuvälineitä osaa arvostaa. Esimerkiksi MechWarrior-pelin julkaisuversion resursseista arviolta noin 40 prosenttia oli sellaisia, joita peli ei käyttänyt laisinkaan, mutta joita ei voinut poistaa, koska pelin tekijätkään eivät olleet enää perillä siitä, mitä resursseja käytettiin ja mitä ei. [22.]

XACT (Cross Platform Audio Creation Tool) on yksi XNA:n työkaluista. Sitä käytetään ääniresurssien hallintaan sekä musiikin ja äänitehosteiden suunnitteluun. Päätarkoitus on kitkan vähentäminen ohjelmoijien ja äänisuunnittelijoiden välisestä yhteistyöstä. [23.]

4.3 MonoGame – vapaan lähdekoodin vaihtoehto

XNA:n tulevaisuus on käymistilassa, sillä Microsoft on nyttemmin (2013) luopunut sen jatkokehittämisestä, mikä tarkoittaa, ettei tukea uusille alustoille ja käyttöjärjestelmille ole luvassa. Kuitenkin siis vain käymistilassa, ei vielä tiensä päässä. Syynä tähän on MonoGame.

XNA:n suosio pelinkehittäjien parissa on ollut sen verran hyvää, että siitä on tehty vapaan lähdekoodin sovitus nimeltään MonoGame, joka esikuvastaan poiketen mahdollistaa julkaisun myös muille kuin Microsoftin alustoille. MonoGamen lisäämiin kohdealustoihin kuuluvat Linux, OSX ja mielenkiintoisesti myös Windows 8. Android ja iOS luonnistuvat myös, mutta ne käyttävät Xamarinin sovitusta .NET-ohjelmistokehyksestä, mikä vaatii kehittäjiltä yhden lisälisenssin hankkimista. [24.]

4.4 XNA pienten kehitystiimien käytössä

Seuraavaksi tarkastelen, miten XNA voisi käytännössä toimia ja osaltaan kääntää suurtuotantoja suosivaa trendiä toiseen suuntaan. Kattavan tutkimusaineiston puutteessa käännyn anekdootinomaisen todistusaineiston puoleen ja annan puheenvuoron kaupallisenkin tason pelejä kehittäneelle James Silvalle [22].

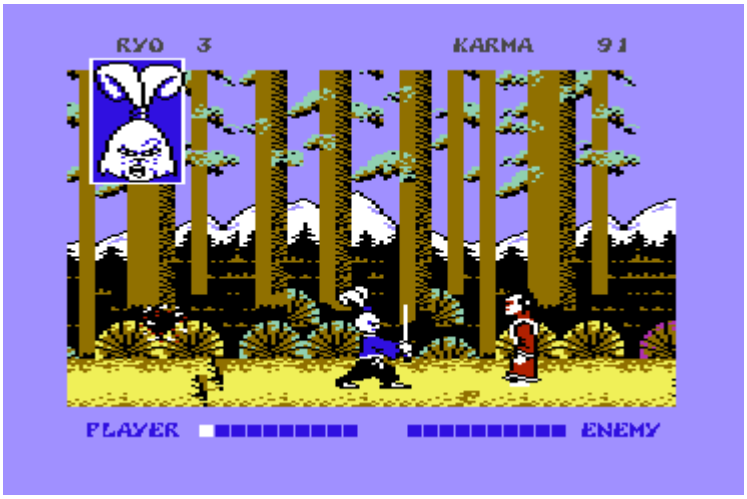
Itsenäisten pelintekijöiden suhtautuminen XNA:han on suurelta osin ollut myönteistä. Esimerkkinä on juuri The Dishwasher: Dead Samurai -pelin XNA:lla tehnyt James Silva, joka kertoo vaikuttavimman puolen hänestä olleen sen, kuinka lähes nollabudjetilla toimivakin pelinkehittäjä pystyy tuottamaan ja julkaisemaan pelin viimeisimmän sukupolven konsolille, eli Xbox 360:lle. Aikaisempien sukupolvien aikana tällainen oli mahdotonta [22]. Mikä merkittävintä, Silva siis toteutti koko projektin yksin, aivan kuten monet alan pioneerit vuosikymmeniä aiemmin.

5 Nopea Kuha – erään pelin tarina

5.1 Kuhan lähtökohdat

Esimerkkiprojektina tehty peli on nimeltään Nopea Kuha. Se on melko yksinkertainen 2D-grafiikkaa käyttävä sivuttaissuunnassa etenevä tasohyppely/”beat 'em up”. Pelisuunnittelu on luovaa työtä, ja luovuuden salaisuus on Albert Einsteinin sanojen ja Marimekon entisen suunnittelijan Kristina Isolan esimerkin mukaan lähteiden huolellinen piilottaminen [25; 26]. Eräs tapa kätkeä kopiointilähteensä, mutta kuitenkin samalla välttää plagiointisyytökset lähteiden mahdollisesti myöhemmin paljastuessa, on niin sanotusti maalata kopionsa sen verran lavealla pensselillä, että alkuperäisen ja kopion yhtäläisyyttä on vaikea nähdä, vaikka ne asettaisi rinnakkain vertailtaviksi.

Suunnittelussa lähdettiin siis vanhasta hyvästä periaatteesta ”tehdään siitä kuin peli X, mutta parempi”. Vaikka tulos ei lähimainkaan muistuta alkuperäistä tavoitetta, eikä se edellä mainitusta syystä saa niin tehdäkään, ainakin tällainen lähestymistapa mahdollistaa projektin käynnistymisen ja tarjoaa tarvittaessa selkeän referenssipisteen muillekin projektin jäsenille designerin lisäksi. Ammattilaishenkisemmät ryhmät toki pyrkivät yleensä toimimaan hieman toisin, vaikka vain siitä syystä, että avoimesti suoran kopion tekemiseen lähteminen voi käydä ammattiylpeydelle. Noloakin se voi olla.



Kuva 19. Usagi Yojimbo, koko nimeltään Samurai Warrior: The Battles of Usagi Yojimbo. Beam Software/Firebird. 1988.

Peli X:nä toimivat Nopean Kuhan tapauksessa kaksi Commodore 64:n pienempää klassikkoa, Usagi Yojimbo (Beam Software/Firebird, 1988) ja Barbarian (Palace Software, 1987), joista näytteet kuvissa 19 ja 20. Tälläkin kertaa lopullinen tuotos oli hyvin kaukana siitä, mitä alun perin lähdettiin hakemaan. Esikuvia tuskin voi edes tunnistaa, ellei niitä käy lukemassa alkuperäisestä määrittelydokumentista. Barbariania ei oikeastaan voi tunnistaa sittenkään, sillä niin vähiin sen ominaisuuksien lainailu jäi. Nopea Kuha on paljon lähempänä perinteistä tasoloikkaa kuin kumpikaan esikuvistaan, jotka ovatkin enemmän ”beat ’em up” -genreä, Barbarian jopa aivan puhtaasti.

Kuva 20. Barbarian. Palace Software. 1987.

Nopeassa Kuhassa pelaajan tehtävänä on siis ohjata pelin päähenkilöä Kuhaa vasemmalta oikealle halki kenttien vaaroja, esteitä ja vihollisia väistellen. Kaikki Kuhan kyvyt eivät ole heti pelin alussa käytettävissä, vaan esimerkiksi korkeamman hypyn ja viholliset tuhoavan hyökkäyksen pelaaja saa käyttöönsä vasta myöhemmissä kentissä. Nyt tehdyssä versiossa on yhteensä yhdeksän kenttää, mutta niitä pystyisi tarvittaessa tekemään helposti lisää.

Huolimatta siitä, että lopullisessa pelissä grafiikka on kaksiulotteista, tuotantovaiheessa käytettiin myös 3D-tekniikoita. Pelihahmot esimerkiksi mallinnettiin, rigattiin ja animoitiin 3D Studiossa, minkä jälkeen näistä animaatioista renderöitiin tarpeeksi yksittäisiä ruu-

tuja bittikartoiksi, joita sitten saattoi käyttää 2D-animaatioina pelin sisällä. Luvussa 3 esittelin tätä vaihetta kuvien kera.

Halusin Nopealle Kuhalle ainakin jonkin verran omaleimaisen ulkoasun. Sitä lähdettiin hakemaan sekoittamalla yhteen useita erilaisia grafiikantuottamismenetelmiä. Pelissä on niin esirenderöityä 3D-grafiikkaa ja Photoshopin perussiveltimillä maalattua grafiikkaa kuin valokuvia ja puuväreillä piirrettyjä kuvia. Ne kaikki yhdistyivät yllättävän hyvin yhdeksi kokonaisuudeksi. Eniten toivomisen varaa jätti tiilien ulkoasu, joka ei oikein istunut useimpiin ulkokenttiin. Tiilistä olisi pitänyt tehdä useampia variaatioita ja versioita, jotka olisivat ulkoasun puolesta sopineet paremmin ulkoilmaan. Lopuksi viimeistelin kokonaisuuden ruudunvenytys- ja vilkkuvaloefekteillä. Jälkimmäisiä käytin kuitenkin hyvin säästeliäästi.

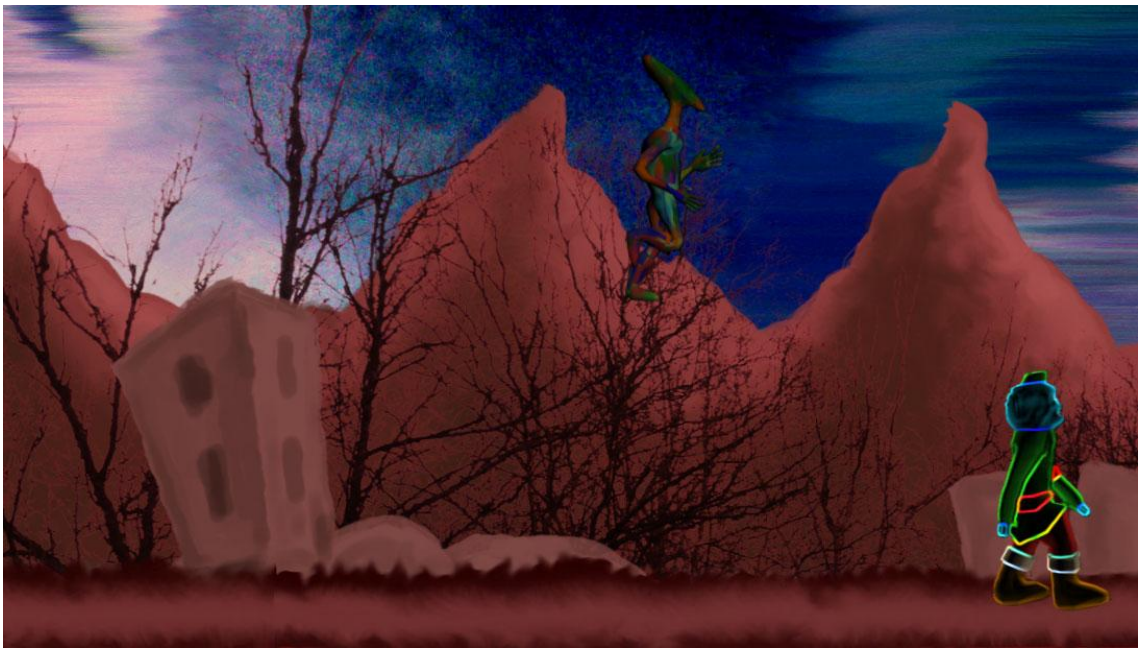
Äänissä keskityin ääniefektien sijasta musiikkiin. Nopea Kuha sisältää nykyisellään vain kaksi ääniefektiä, joiden lisääminen ja monipuolistaminen olisikin yksi ilmeinen jatkokehityssuunta. Sen sijaan musiikkiraitoja löytyy peräti kymmenen: yksi jokaista kenttää kohden ja vielä yksi päävalikkoon. Kaikki musiikki on omista arkistoistani löytynyttä julkaisematonta materiaalia, jota ei siis tarvinnut erikseen Nopeaa Kuhaa varten säveltää. Musiikin tyylilajeissa kuljetaan samaa sillisalaattilinjaa kuin visuaalisellakin puolella.

Itse ohjelmoinnissa työkaluina käytettiin aluksi Visual Studio 2008:aa ja XNA Game Studio 3.1:tä, joista kuitenkin siirryttiin projektin venymisen ja siirtymisen helppouden vuoksi versioihin 2010 ja 4.0, jotka olisivat tarjonneet myös mahdollisuuden Windows Phone -käyttöjärjestelmän tukemiseen. Ohjelmointikielenä käytin C#-kieltä. Hieman liioitellen voisi sanoa, että varsinainen ohjelmointityö noudatti eräänlaista Frankensteinin hirviö -käytäntöä, jossa kaikenlaisia tutoriaaleista, esimerkkiohjelmista ja Internetin syövereistä kaivettuja koodinpätkiä yhdisteltiin leikkaa ja liimaa -tyyliin. Tämä tosin on osittain vain näkökulmakysymys, mutta tarkempi silmäys koodin puolelle alaluvussa 5.3. Tärkeimpinä malleina olivat Microsoftin omat koodiesimerkit GameStateManagement ja Platformer [27; 28]. Ensimmäinen käsittelee valikoissa ja eri pelitilojen välillä siirtymistä, jälkimmäinen taas joitakin tasohyppelypelin perustoimintoja.

Yksi hyvä tapa saada peliin uusia mielenkiintoisia ominaisuuksia on tehdä niitä vahingossa. Tämä ei vaadi ankaraa ajatustyötä suunnitteluvaiheessa, vaan valpasta tarkkaavaisuutta toteutusvaiheessa. Ehkäpä juuri havaittu kömpelö ohjelmointivirhe, joka

saa pelin hahmot toimimaan odottamattomalla tavalla, ei olekaan jotain, josta on päästävä välittömästi eroon. Virheellinen toiminta voi hyvin olla oikeastaan parempaa kuin se, jota alun perin suunniteltiin. Näin kävi Nopean Kuhan tapauksessa.

Kytkin epähuomiossa pelaajan hahmoon yhtä aikaa sekä ylhäältä päin kuvatun pelin kontrollit neljään ilmansuuntaan liikkumisineen että sivulta päin kuvatun pelin vasemmalle ja oikealle liikkumiseen tarkoitetut kontrollit. Tämän seurauksena sivulta päin kuvatun pelin sankari Kuha sai alun perin tasakorkuisiin hyppyihinsä huomattavasti säätövaraa. ’Pohjoiseen’ kulkeminen hypyn aikana sai aikaan tavallista korkeamman hypyn ja ’etelään’ kulkeminen matalamman. Kuvassa 21 näkyy tavallista korkeammalle loikkaava Kuha. Oikean korkuisen hypyn käyttämisestä oikeaan aikaan taas tuli yksi Nopean Kuhan pelillisesti tärkeimmistä elementeistä. Tämän kaltainen virheiden ja vahingossa vastaan tulevien ratkaisujen hyödyntäminen vaatii käytännössä matalaa organisaatiota, jossa suunnittelevat, toteuttavat ja päättävät tahot ovat lähellä toisiaan.



Kuva 21. Kuha loikkaa pelintekijän suunnitelmia korkeammalle. Tylsistyneenä poispäin lampsivaa vihollista tämä ei kuitenkaan säväytä.

Nopean Kuhan tavoitteisiin kuului mahdollisimman kevyt graafinen käyttöliittymä, jossa erilaiset mittarit, laskurit ja symbolit olisivat näkymättömissä ja kaikki tarpeellinen informaatio olisi pelaajan ulottuvilla vain itse pelimaailmaa tuijottamalla. Pistemittareista ja ammuslaskureista ja vastaavista päästiin eroon helposti jättämällä vain niihin kuuluvat ominaisuudet kokonaan pois Nopeasta Kuhasta, vaikka ne ovatkin ominaisuuksia, jot-

ka yleensä tulevat ensimmäisten joukossa mieleen, kun pohtii, mitä peliin vielä lisäisi. Pelaajan hahmon terveysmittarista oli hankalampi päästä eroon, sillä pelaajalla pitäisi olla koko ajan tiedossa, kuinka lähellä haudan reunaa hänen kovia kokenut hahmonsa kulloinkin hoippuisi.

Ratkaisu löytyi Kuhan riemunkirjavasta ulkomuodosta: täysissä voimissaan olleessaan se hehkuisi täydessä väriloistossaan, terveyden heiketessä värit tummenisivat portaattomasti, ja viimeisenkin värin kaikottua Kuha kuukahtaisi kuolleena maahan. Ohjelmointiteknisesti tämä hoitui yllättävän helposti vain lisäämällä muutama rivi Player-luokan Update-metodiin, kuten esimerkkikoodista 1 näkyy.

```
playerColorInt = health * 255 / 100;
playerColor = new Color(playerColorInt, playerColorInt, playerColorInt);
```

Esimerkkikoodi 1. Pelihahmon väri määräytyy sen terveyden mukaan.

Kuhan terveyden määrittävän health-muuttujan vaihteluväli on nollasta sataan. Muuttujaa playerColor taas käytin Draw-metodin parametrina hahmon värittämiseen.

Nykypeleistä otin Nopeaan Kuhaan sen, että se pyrkii itse opettamaan pelaajan pelaamaan ilman, että tämän tarvitsee ohjekirjasta vilkaista edes kontrolleja. Nämä peliin sisään rakennetut tutoriaalit tein lyhyiksi tekstiohjeiksi, jotka ilmestyvät näkyviin oikeaan aikaan. Ruudulle pomppivat tekstit eivät oikein sovi yhteen graafisen käyttöliittymän suurimpaan mahdolliseen keveyteen pyrkimisen kanssa, mutta päädyin niihin, koska vaihtoehto, eli ruudulle pomppivat näppäinsymbolit, ei olisi ollut kovin suuri parannus.

Pidin tutoriaaliohjeet myös lyhytsanaisina ja helposti ohitettavina, jotta ne eivät liikaa häiritsisi niitä pelaajia, jotka eivät niitä tarvitse tai halua. Pakkosyötetyt ja liian pitkään kädestä pitelevät alun opetteluosuudet ovat olleet eräitä pelitalojen keinoja niiden laajentaessa yleisöään aina vain laajemmiksi. Harrastuneemman yleisön parissa ne kuitenkin helposti aiheuttavat hylkimisreaktioita. Nopea Kuha on tarkoitettu pikemminkin kokeneelle pelaajalle kuin vain kenelle tahansa, joten annostelin ohjeita hyvin varovasti.

5.2 Testaus eli Kuhan koitos

Nopean Kuhan ulkopuolinen testaus suoritettiin pienellä testaajajoukolla. Testaajat kuuluivat kohderyhmään, sillä heillä oli runsaasti kokemusta niin tietokonepeleistä yleensä kuin myös tasoloikista erityisesti. Nopea Kuha ei tosin aivan puhdasverinen tasohyppely ole, mutta se kuitenkin taipuu siihen suuntaan vahvasti. Testaajilla siis oli hyvät valmiudet antaa relevanttia palautetta.

Testitilanteessa testaajalla oli Nopea Kuha asennettuna koneellaan ja pikakuvake työpöydällä. Itse pelinkehittäjänä istuin sivummalla tarkkailemassa ja tekemässä muistiinpanoja, samalla kun varoin antamasta minkäänlaisia vinkkejä. Alkuvalikosta itse peliin selviytyminen ja peruskontrollien, eli vasemmalle ja oikealle juoksemisen ja tavallisen hypyn, omaksuminen onnistui testaajilta ilman ongelmia.

Kuva 22. Siniset pirulaiset uhkaavat marssia Kuhan kumoon.

Sen sijaan laatikot, joista pystyi sekä kävelemään läpi että hyppäämään päälle, aiheuttivat hieman vaivaa. Pelin toisessa kentässä näitä laatikoita pitkin hyppimällä pitää välttää alhaalla liikkuvat viholliset. Hämäävästi kuitenkin näiden vihollisten läpi tai ohi pääsee melkein juoksemaan, mutta kuitenkin vain melkein. Uusi pelimekaniikka eli laatikoita pitkin kiipeäminen esitellään tilanteessa, jossa oppimisen häiriötekijöinä ovat ympärillä pyörivien vihollisten aiheuttama lisäpaine ja valheellinen vihjaus toisesta kulkureitistä ja ratkaisutavasta. Nopean Kuhan aikaisemmassa versiossa laatikon päälle kii-

peäminen esiteltiin jo ensimmäisessä kentässä, mutta tämä siirrettiin myöhempään vaiheeseen peliä. Uudesta ensimmäisestä kentästä tämä laatikon päälle kiipeämisen esittely jäi pois. Pelimekaniikan esittely tarkoittaa tässä yhteydessä siis sitä, että pelaaja laitetaan tilanteeseen, josta ainoana ohipääsykeinona on tämän uuden kyvyn, toimintatavan tai pelimekaniikan käyttäminen.

Nopean Kuhan läpäisemisestä testaajat suoriutuivat ensimmäisellä kerralla noin viidessätoista minuutissa. Seuraavalla kierroksella, kun kontrollit olivat tutummat ja pelin pulmat oli ratkaistu, läpipeluuaika väheni alle kolmannekseen. Vertailun vuoksi kerrottakoon, että oma läpipeluuaikani pelinkehittäjänä on parhaimmillaan aavistuksen yli kaksi minuuttia. Ajat kuulostavat lyhyiltä, mutta itse olen niihin tyytyväinen. Ne ovat myös oikeastaan melko hyvin linjassa monien 1980-luvun pelien läpipeluuaikojen kanssa ja 1980-luvultahan Nopean Kuhan esikuvat ovat. Läpipeluuaika saattoi olla lyhimmillään vain muutamia minuutteja ja peliaika koostua lukuisista epäonnisista uudelleenyrityksistä korkean vaikeustason vuoksi.

Audiovisuaalisesta toteutuksesta testaajilta tuli kiitoksia. Erikoismaininnan sai pelin puolenvälin luola ja sen avautuva punainen taustagrafiikka. Risuja sen sijaan peli sai siitä, että hahmojen niin sanotut osumalaatikot (hitbox) tuntuivat jotenkin sattumanvaraisilta. Hitboxien avulla tehdään törmäystarkistus, joka siis on se mekanismi, joka määrittelee, että jos Kuha seisoo tässä ja vihollinen tuossa, osuvatko he silloin toisiinsa vai eivät. Kuhan ja vihollisten toisiinsa törmäilystä oli vaikeaa ennakoida, mitä siinä tulee tapahtumaan. Osuminen piikkiansoihin tuntui myös tapahtuvan aivan liian helposti verrattuna siihen, miltä piikkiansat näyttivät.

Hyppymekaniikkaa kehuttiin mielenkiintoiseksi, mutta toisaalta myös moitittiin hieman huteraksi. Erityisesti ongelmaksi koettiin se, että jos hypyn aloittaa nuoli ylös -näppäimellä, varsinainen hyppynappi eli välilyönti ei tee silloin mitään ja hyppy jää todennäköisesti paljon tarkoitettua lyhyemmäksi ja matalammaksi.

5.3 Silmäys koodin puolelle

Nopean Kuhan koodia ei käydä yksityiskohtaisesti läpi, mutta näytän miten luvussa 3 aikaan saatu kokoelma juoksuanimaation ruutuja saadaan piirrettyä ruudulle. Liitteessä 1 on kuitenkin näytetty muutamien pelimekaniikan kannalta tärkeimpien luokkien ra-

kennetta. Joukossa on myös Player-luokka, josta kaikki tämän luvun koodiesimerkit ovat peräisin.

XNA:n Game-luokka hoitaa ohjelmoijan puolesta paljon perustyötä, ja jonkin kuvan saaminen ruudulle näkymään ja reagoimaan käyttäjän komentoihin onnistuu hyvin vähällä vaivalla. Ohjelmoijan periaatteessa tarvitsee vain kertoa, mikä tekstuuri kuuluu minnekin, eikä muistiosoitteista tai sellaisista tarvitse välittää. Game-luokka sisältää tärkeät metodit (funktiot) Update ja Draw. Itse peliohjelman pääluokka periytetään tästä ohjelmistokehyksen luokasta.

Updatea, joka huolehtii pelin ja pelilogiikan tilan päivittämisestä, kutsutaan 60 kertaa sekunnissa. Draw, jonka tehtävänä on piirtää grafiikka ruudulle, pyrkii samaan, mutta suostuu kernaasti alhaisempaan taajuuteen, mikäli koneesta alkaa teho loppua.

Esimerkkikoodit 2–8 esittävät, miten ja milloin kuvassa 16 näkyvää sprite-kokoelmaa käytetään pelissä. Siihen on poimittu Player-luokasta ne kohdat, jotka tämän tekevät.

Esimerkkikoodissa 2 uusi animaatio runningAnimation määritellään, ja sen alku- ja loppuruuduiksi annetaan 1 ja 23. Seuraavaksi tekstuuriin exonRunSprite ladataan kuvassa 16 nähty tiedosto, jonka nimi siis on exona_run_full_00_22. Nämä tehdään silloin, kun Player-luokasta tehdään instanssi pelimaailman puolelle.

```
    class Player
    {
            …
       Texture2D exonRunSprite;
            …
       public Animation runningAnimation = new Animation("running", 1, 23, 100);
            …
       public void LoadContent()
       {
            …
       exonRunSprite =
GameplayScreen.Content.Load<Texture2D>("exona_run_full_00_22");
            …
       }
```

Esimerkkikoodi 2. Juoksuanimaation lataaminen muistiin.

Update-silmukassa ajastin timer tarkastaa joka kierroksella, joko nyt olisi kulunut tarpeeksi aikaa, että animaatiota voisi siirtää yhden ruudun eteenpäin. Mikäli näin on,

ajastin nollaa itsensä ja ryhtyy odottamaan seuraavaa animaatioruutua (esimerkkikoodi 3).

```
        public void Update(GameTime gameTime)
        {
            timer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
            if (timer > interval)
            {
                currentPos++; // player animation
                timer = 0f;
            }
```

Esimerkkikoodi 3. Siirtyminen animaation seuraavaan ruutuun.

Mikäli pelihahmo juoksee ja animaatio on edennyt juoksuanimaation viimeiseen ruutuun, animaatio siirtyy takaisin alkuun (esimerkkikoodi 4).

```
if (currentPos >= runningAnimation.EndingFrame && state == playerState.running)
            {
                currentPos = runningAnimation.StartingFrame - 1;
            }
```

Esimerkkikoodi 4. Laskurin siirtyminen animaation ensimmäiseen ruutuun.

Animaation tämänhetkisen kohdan currentPos perusteella lasketaan suorakulmio exonSourceRectille sijainti ja koko, kuten esimerkkikoodista 5 nähdään. Tämä suorakulmio on ikään kuin tähtäin, joka valitsee ja noukkii spritesheetiltä (kuva 16) näytölle piirrettäväksi oikean yksittäisen ruudun.

```
exonSourceRect = new Rectangle((currentPos % 5) * spriteWidth, (currentPos / 5) *
spriteHeight, spriteWidth, spriteHeight);
```

Esimerkkikoodi 5. Oikean kohdan valitseminen spritesheetiltä.

Mikäli pelihahmon tila ei ole mikään neljästä esimerkkikoodissa 6 mainituista ja pelihahmon sijainti x-akselilla on eri kuin Update-silmukan edellisellä kierroksella, pelihahmon tilaksi annetaan juokseva ja käytettäväksi spritesheetiksi juoksuanimaation sisältävä spritesheet. Katso esimerkkikoodi 6.

```
            if (state != playerState.dying && state != playerState.dead
                && state != playerState.punching && state != playerState.hurt)
            {

                if (lastPlayerPosition.X != playerPosition.X)
                {
                    currentSprite = exonRunSprite;
                    state = playerState.running;
                }
```

Esimerkkikoodi 6. Osa koodista, joka määrittelee mikä pelihahmon tila kulloinkin on.

Sen perusteella, onko hahmon edellinen sijainti ollut vasemmalla vai oikealla puolella, hahmolle annetaan katsomissuunta sekä myös peilausefekti, mikäli katsomissuunta on eri kuin alkuperäisissä animaatioruuduissa. Mikäli liikkuminen on ollut vain vähäistä, katsomissuuntaa ei muuteta. Katso esimerkkikoodi 7.

```
            if (lastPlayerPosition.X < playerPosition.X &&
Math.Abs(lastPlayerPosition.X - playerPosition.X) > 5)
            {
                playerEffects = SpriteEffects.None;
                facing = playerFacing.right;
            }

            if (lastPlayerPosition.X > playerPosition.X &&
Math.Abs(lastPlayerPosition.X - playerPosition.X) > 5)
            {
                playerEffects = SpriteEffects.FlipHorizontally;
                facing = playerFacing.left;
            }
```

Esimerkkikoodi 7. Hahmon katsomissuunnan selvittäminen.

Lopuksi Draw-metodi piirtää Kuhan ruudulle oikeaan paikkaan oikean näköisenä käyttämällä parametreinään ylemmissä esimerkeissä annettuja muuttujien arvoja, kuten esimerkkikoodissa 8 näytetään.

```
        public void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(currentSprite, playerPosition, exonSourceRect,
playerColor, 0f, exonOrigin, 1f, playerEffects, 0);
        }
```

Esimerkkikoodi 8. Metodi, joka piirtää pelihahmon ruudulle.

Tätä Player-luokan Draw-metodia kutsutaan pelimaailman piirtävän GameplayScreen-luokan oman Draw-metodin sisältä.

6 Yhteenveto

Insinöörityön tavoitteena ollut pelin luominen onnistui, eli XNA vaikuttaa varsin toimivalta teknologialta, vaikka Microsoft ei sitä enää varsinaisesti tue. Vaikeuksia tuli vastaan, mutta niistä päästiin tavalla tai toisella lopulta yli tai eroon.

Pelinteko prosessina sen sijaan on merkittävältä osaltaan kysymys siitä, kuinka hyvin tuotannon eri toimijat, tiimit ja tiimien jäsenet saadaan toimimaan yhdessä, joten tältä osalta teorian sovittaminen käytäntöön jäi vaillinaiseksi yhden hengen tuotantotiimin vuoksi. Jotkin periaatteet, kuten mahdollisimman nopea prototyypin toimintaan saattaminen sekä jatkuva iterointi ja testaaminen, tulivat käyttöön aika luonnostaan, joten niille oli mukava saada nimi.

Nopea Kuha on nykytilassaan yksinkertainen, mutta sitä voi kuitenkin hyvin jatkokehittää muutamaankin suuntaan. Yksi mahdollisuus on sen kääntäminen eri alustoille, joita on MonoGamen myötä jo aika monta alun perin XNA:lla Windowsille tehdyn projektin ulottuvilla. Toinen on pelin laajentaminen uusia kenttiä ja uudenlaisia vastuksia lisäämällä. Nämä onnistuvat helposti, vaikka Nopea Kuha on vain osittain datalähtöisesti toteutettu peli. Ulkoasussa on viimeistelyn varaa, eikä äänimaailmaakaan olisi pahitteeksi hieman rikastuttaa. Nopea Kuhaa voisi myös yrittää hivuttaa lähemmäksi esikuvaansa Usagi Yojimboa ja lisätä siihen seikkailullisia elementtejä, kuten esimerkiksi hahmoja, joiden kanssa keskustella, ja erilaisia esineitä käytettäväksi. Joitakin yllättäviä yhtäläisyyksiä näissä kahdessa jo onkin, kuten kuvasta 23 voi nähdä.

Kuva 23. Usagi Yojimbo ja Nopea Kuha: Kuin kaksi marjaa.

Suurten budjettien valtavat peliprojektit eivät ole katoamassa mihinkään ainakaan vielä, mutta niiden rinnalle, aikaisempaa näkyvämmälle paikalle, palaavat pienemmät, niin sanotut indie-pelit. Tämä tapahtuu osin, ja varsinkin XNA:n tapauksessa, suuryhtiöiden myötävaikutuksella, mutta kyse ei ole hyväntekeväisyydestä. Pelien digitaaliset jakelukanavat mahdollistavat sen, että jopa halvalla myytävä pieni peli voi tehdä voittoa, mikä oli mahdoton ajatus aikana, jolloin jokainen peli piti kierrättää CD-tehtaan ja lukuisien tukkumyyjien kautta ennen sen päätymistä jälleenmyyjän hyllyyn. Tämän jälkeen piti vielä toivoa, että asiakas sen sieltä poimisi mukaansa melko pian arvokasta hyllytilaa viemästä.

Joka tapauksessa, kuten Wanda Meloni [10] asian ilmaisee, pelinkehitys elää tällä hetkellä eräänlaista renessanssia, eikä XNA ole yhtään huono työkalupakki niille, jotka haluavat siihen osallistua, kuten Nopea Kuha tulee toivottavasti jossain määrin osoittamaan. Tämä kehityskulku on vain vahvistunut niinä neljänä vuotena, jotka Melonin lausunnosta on ehtinyt kulua.

Lähteet

1 Ludwig Wittgenstein. 2002. Verkkodokumentti. Stanford Encyclopedia of Philosophy.<http://plato.stanford.edu/entries/wittgenstein/>. Luettu 5.10.2013.

2 Barry, Isaac. 2010. Game Design. Teoksessa Rabin, Steve (toim.). Introduction to Game Development. Second Edition. USA: Course Technology.

3 Mononen, Mikko. 2003. Pelisuunnittelu tietokonepelisuunnittelijan näkökulmasta. Verkkodokumentti. <http://www.pingstate.nu/omnilayer/yksi/media/552/pelisuunnittelu1.html>. Luettu 5.10.2013.

4 Novak, Jeannie. 2008. Game Development Essentials. Second Edition. Canada: Delmar Cengage Learning.

5 Ahonen, Marko. Videopelaamisen esihistoria Odysseuksesta harharetkiin. Pelit 5/2003.

6 Grace, Lindsey. 2009. Truly Independent Game Development: A Case For Making Games By Yourself. Verkkodokumentti. <http://www.gamecareerguide.com/features/776/truly_independent_game_.php>. 20.8.2009. Luettu 6.10.2013.

7 Bertram, Bill. 2005. Sinclair 48K ZX Spectrum computer (1982). Verkkodokumentti. Wikimedia Foundation. <http://commons.wikimedia.org/wiki/File:ZXSpectrum48k.jpg>. Luettu 24.11.2013.

8 Kuutio, Aleksi ja Sinerma, Olli. 2009. Suomen peliteollisuus tienhaarassa. Pelit 7/2009, s. 26–29.

9 Greig, Scott. 2002. Postmortem: Bioware's *Neverwinter Nights. V*erkkodokumentti. <http://www.gamasutra.com/view/feature/2918/postmortem_biowares_neverwinter_.php>. 4.12.2002. Luettu 6.10.2013.

10 Meloni, Wanda. 2009. The Brief - The Gaming Renaissance Movement. Verkkodokumentti. <http://www.gamasutra.com/blogs/WandaMeloni/20090508/1344/The_Brief__The_Gaming_Renaissance_Movement.php>. 8.5.2009. Luettu 6.10.2013.

11 Bakie, Robert T. 2010. A Brief History of Video Games. Teoksessa Rabin, Steve (toim.). Introduction to Game Development. Second Edition. USA: Course Technology.

12 Activision reveals sales figures for Black Ops and Modern Warfare 2. 2011. Verkkodokumentti. Shacknews. <http://www.shacknews.com/article/69577/activision-reveals-sales-figures-black-ops-and-modern-warfare-2>. 3.8.2011. Luettu 10.11.2013.

13 Al Lowe Something Awful Q&A. 2011. Verkkodokumentti. Mouser/Webdog. <http://tindeck.com/listen/gyxs>. 6.4.2011. Kuunneltu 6.10.2013.

14 Game Over. 2013. Game Developer Magazine 6/2013, s. 27–32.

15 Helmuth von Moltke the Elder. Verkkodokumentti. Wikimedia Foundation. <http://en.wikiquote.org/wiki/Helmuth_von_Moltke_the_Elder>. Luettu 8.11.2013.

16 Sloper, Tom. 2010. Game Production and Project Management. Teoksessa Rabin, Steve (toim.). Introduction to Game Development. Second Edition. USA: Course Technology.

17 Llopis, Noel. 2010. Teams and Processes. Teoksessa Rabin, Steve (toim.). Introduction to Game Development. Second Edition. USA: Course Technology.

18 Johnson, David. 2010. 3D Modeling. Teoksessa Rabin, Steve (toim.). Introduction to Game Development. Second Edition. USA: Course Technology.

19 Crossley, Rob. 2013. Microsoft email confirms plan to cease XNA support. Verkkodokumentti. ComputerAndVideoGames.com. <http://www.computerandvideogames.com/389018/microsoft-email-confirms-plan-to-cease-xna-support/>. 1.2.2013. Luettu 24.11.2013.

20 Xbox at BUILD and Microsoft's Collaboration with Unity Technologies. 2013. Verkkodokumentti. Microsoft. <http://news.xbox.com/2013/06/xbox-one-unity-announcement>. 27.6.2013. Luettu 24.11.2013.

21 Multerer, Boyd. 2007. Consumers to Creators: The Road to XNA Game Studio Express. Konferenssiesitelmä. Microsoft. Game Developer Conference 2007.

22 Stuart, Keith. 2008. XNA and the future of bedroom coding. Verkkodokumentti <http://www.guardian.co.uk/technology/gamesblog/2008/feb/28/XNAandthefutureofbedroom>. 28.2.2008. Luettu 26.10.2011.

23 XNA Game Studio 3.1:n dokumentaatio. Verkkosivusto. Microsoft. <http://msdn.microsoft.com/en-us/library/bb200104.aspx>. Luettu 26.10.2011.

24 About MonoGame. 2013. Verkkodokumentti. MonoGame Team. <http://monogame.net/about>. Luettu 24.11.2013.

25 Albert Einstein Quotes. Verkkodokumentti. Goodreads Inc. <http://www.goodreads.com/quotes/5793-creativity-is-knowing-how-to-hide-your-sources>. Luettu 9.11.2013.

26 Kristina Isola myöntää kopion. 2013. Verkkodokumentti. Yle. <http://yle.fi/uutiset/kristina_isola_myontaa_kopion/6663592>. 29.5.2013. Luettu 9.11.2013.

27 Xbox LIVE Indie Games Game State Management code sample. 2011. Verkkodokumentti. Microsoft. <http://xbox.create.msdn.com/en-US/education/catalog/sample/game_state_management>. 24.5.2011. Luettu 5.11.2013.

28 Xbox LIVE Indie Games Platformer code sample. 2010. Verkkodokumentti. Microsoft. <http://xbox.create.msdn.com/en-US/education/catalog/sample/platformer>. 4.10.2010. Luettu 5.11.2013.

Kaavio Nopean Kuhan tärkeimmistä luokista