LAUREA AMMATTIKORKEAKOULU Uuden edellä

Itsenäinen pelikehitys

Virtanen, Harri

2011 Leppävaara

Laurea-ammattikorkeakoulu
Leppävaara

Itsenäinen pelikehitys

Harri Virtanen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Helmikuu, 2012

Laurea-ammattikorkeakoulu **Tiivistelmä**
Leppävaara
Tietojenkäsittelyn koulutusohjelma

Harri Virtanen

Itsenäinen pelikehitys

Vuosi 2011 Sivumäärä 56

Tässä opinnäytetyössä analysoidaan itsenäistä pelikehitystä sekä käsitellään pelikehityksen eri vaiheita. Tavoitteena oli tutkia, mitä itsenäinen pelikehitys pitää sisällään sekä mitä työkaluja ja resursseja kehitykseen vaaditaan.

Itsenäinen pelikehitys on yleistynyt nykyaikana jatkuvasti kasvavalla nopeudella. Pelaajille tämä merkitsee tarjonnassa kasvua, kun kehittäjälle se taas merkitsee kilpailussa kasvua. Digitaalinen aikakausi mahdollistaa oman tuotoksen jakelun internetissä erittäin vähäisillä kustannuksilla verrattuna perinteiseen jakeluun. Tietoa ja kokemusta voi harjoittaa itsenäisesti. Resursseja, kuten musiikkia tai tekstuureita, on laillisesti saatavilla jopa kaupalliseen käyttöön ilmaiseksi.

Kuvailtu tilanne on itsenäiselle kehittäjälle ihanteellinen. Pelikehitys on monimutkainen prosessi, joka vaatii paljon sillä kaikki on tuotettava itse, vaikka tietyt resurssit olisivatkin helposti saatavilla. Itsenäinen pelikehitys sallii kehittäjälle vapauden tuotokseensa, mutta myös vastuu on kehittäjällä itsellään.

Työssä vertaillaan eri ohjelmointikieliä ja kehitysprosesseja. Tutkimus käy läpi kehityksen eri vaiheita, prosessien suunnittelua ja ottaa kantaa kehityksen aikana ilmeneviin mahdollisiin ongelmiin. Tutkimuksen aikana todettiin, että itsenäinen pelikehitys on monipuolinen prosessi, joka käsittää paljon muutakin kuin ohjelmointia. Grafiikka, äänet, pelattavuus ja käytettävyys täytyy kaikki saada toimimaan kokonaisuutena menestyvän pelin kehittämiseksi. Opinnäytetyö on katsaus pelikehitykseen maailmaan ja sen sisältämiin haasteisiin, joita kehittäjä tulee kohtaamaan.

Osana työtä ohjelmoitiin yksinkertainen peli, joka toteutettiin käyttämällä projektin aikana tutkittuja kehitysmenetelmiä. Lopullinen tuotos ei ole levitykseen tarkoitettu valmis peli, vaan työkalu joka sisältää erilaisia ohjelmointiharjoituksia. Liitteenä on arkistotiedosto, joka sisältää tutkimuksen aikana ohjelmoidun pelin sekä kaikki sen tarvitsemat resurssit Windows-, Linux ja Mac-käyttöjärjestelmille.

Asiasanat peli, ohjelmointi, kehitys, tutkimus

Laurea University of Applied Sciences
Leppävaara
Bachelor of Information Technology

Abstract

Harri Virtanen

Independent Game Development

Year 2011 Pages 56

This thesis analyzes independent game development and addresses the different stages in game development. The objective was to find out what independent game development comprises and to see what tools and resources are needed for it.

Independent game development has become more common over the years at an increasing pace. To gamers this means more diverse supply in games, whereas at the same time it means more competition to the (independent) game developer. The digital era makes it possible to distribute products over the internet with little to no cost, when comparing to the traditional retail process. Information and experience can be self-taught. Resources, such as music and textures, can be legally obtained from the Internet even for commercial use.

The situation could not be better for an independent developer. Game development is a complicated process that requires a great deal of effort even if some resources are readily available. Independent development allows the developer the freedom to maintain his/her own game, but it also places responsibility for everything on him/her.

There are comparisons of different programming languages and development processes. The study examines the various steps in the process and addresses problems that might be encountered during the development.

During the research process it was noted that independent game development is a diverse process that includes a great deal more than just programming. Graphics, sounds, gameplay and functionality all have to work together to make the game a success.

This thesis has an attachment that is an archive file containing a small game developed during the research work on this thesis. It works on the Windows-, Linux and Mac operating systems.

Keywords game, programming, development, research

Sisällys

1 Johdanto

Tämä opinnäytetyö käsittelee itsenäistä pelikehitystä ja siihen kuuluvia tuotantoprosesseja. Tutkimuksen tarkoituksena oli selvittää, mitä vaatimuksia ja haasteita itsenäinen pelikehittäjä kohtaa kehityksen aikana. Pelikehitys on nykyaikana pääasiassa isojen pelitalojen työtä ja näemme vuosittain julkaistavan pelejä, joilla on ollut mahdollisesti monien miljoonien eurojen budjetti. Tästä huolimatta indie-pelikehittäjät ovat yleistyneet ja markkinoille ilmestyy jatkuvasti uusia itsenäisesti kehitettyjä pelejä.

Tutkimusongelmana oli selvittää mitä pelikehitys käsittää sekä teoriassa että käytännössä. Pelin suunnittelu ja kehitys ovat monimutkaisia prosesseja. Koska vastuu on yhden tai muutaman kehittäjän harteilla, se vaatii omistautumista ja visiota kehittäjältä. Keskeisiä tutkimuskysymyksiä olivat:

- Mitä ”itsenäinen pelikehitys” sisältää käsitteenä?
- Miten paljon resursseja pelikehitykseen vaaditaan?
- Mitä tietoa ja taitoja pelikehitykseen vaaditaan?
- Mikä tekee pelistä menestyvän?

Käsittelyssä on pelikehitykseen liittyvää kehitystä sekä suunnittelua, mutta ei julkaisuprosessia tai markkinointia, sillä ne ovat opinnäytetyön aihepiirin ulkopuolella. Aihepiiriin kuuluu mm. alustavan suunnittelun vaiheet, käytettävät rajapinnat sekä ohjelmointiresurssit kuten äänet ja grafiikka.

Opinnäytetyön liitteenä on peli nimeltään *Spaceshooter,* joka ohjelmoitiin tutkimuksen aikana. Spaceshooter on harjoituksena toteutettu tekniikkademo eikä ole tarkoitettu kaupalliseen levitykseen.

2 Keskeiset käsitteet

ASCII-grafiikka	*American Standard Code for Information Interchange* – 7-bittinen merkistö jota käytetään roguelike-peleissä perinteisen grafiikan siajsta esittämään pelimaailmaa.
Alpha-versio	Kehityksen aikainen keskeneräinen versio joka ei sisällä lopullista toiminnallisuutta.
C / C++	C++ on C-ohjelmointikielestä vaikutteita saanut yleisesti käytetty oliopohjainen ohjelmointikieli
Debuggaus	Virheiden jäljitys ohjelmasta erilaisin apufunktioin tai ulkoisin työkaluin
DRM	Digital Rights Management. Käyttöoikeuksien hallinta ja suojausmenetelmä jolla pyritään ehkäisemään ohjelmiston tai sisällön laitonta käyttöä.
Framework	Runkorakenne, valmis ohjelmistopohja jonka päälle peli tai sovellus kehitetään.
Funktio	Ohjelmoinnissa käytetty rutiini, joka suorittaa halutut tehtävät yleensä itsenäisesti muusta koodista riippumatta
IDE	*Integrated Development Environment* - ohjelmistokehitysympäristö, joka sisältää tarvittavat sovellukset kehitykseen yleensä yhdessä paketissa.
Indie-kehittäjä	Independent developer – Itsenäinen kehittäjä, jota ei saa rahoitusta julkaisijalta
Kehitysympäristö	Ks. IDE
Lua	Vapaan lähdekoodin ohjelmointikieli. Suosittu scripting-kielenä mm. pelikehityksessä.
LÖVE 2D	Ilmainen ja vapaan lähdekoodin 2D-pelimoottori kehittyneellä fysiikanmallinnuksella.
MMO-peli	Massive Multiplayer Online-peli.
Open source-ohjelmistot	Vapaan lähdekoodin ohjelmistot.
Origin	*Electronic Arts* -yrityksen digitaalinen distribuutio ja DRM-järjestelmä videopeleille.
Pelimoottori	Valmis runkorakenne (framework), jonka päälle peli kehitetään. Sisältää tyypillisestä äänen, grafiikan ym. hallintaan liittyvät toiminnot.
Prototyypitys, protoilu	Termi jota käytetään ohjelmistojen alpha-versioiden nopeasta käytännön testauksesta ja kehityksestä.
Rajapinta	*Application Programming Interface* (API) – ohjelmointirajapinta,

	määritelmä jonka avulla eri ohjelmat voivat vaihtaa tietoa keskenään.
Roguelike	ASCII-grafiikalla toteutettu vuoropohjainen seikkailuroolipeli
Scripting, skriptauskieli	Ns. scriptauskielet ovat kieliä, joita voidaan käyttää ohjelmoimaan peliin toimintoja suorituksen aikana
SDK	*Software Development Kit* – kehityspaketti, jossa on valmiit työkalut esimerkiksi tiettyä alustaa varten.
Steam	*Valve Softwaren* kehittämä digitaalinen distribuutio ja DRM-järjestelmä videopeleille.
Top down -shooter	Ammuskelupeli, jossa kuvakulma on ylhäältä.
WASD-ohjaus	Näppäimistön ohjausskeema, jossa käytetään näppäimiä W, A, S ja D ohjaamiseen.
Virtuaaliympäristö	Tapa, jolla ohjelmat voidaan ajaa ns. erillisessä tilassa virtuaalisesti.

3 Tavoitteet

Opinnäytetyön tavoitteena oli tutkia itsenäistä ja harrastepohjaista pelikehitystä. Aktiivisena ohjelmoijana ja pelaajana päätin yhdistää nämä kaksi harrastetta. Molempia soveltaen tutustuin pelikehitykseen ja kehitin harjoituksena pelin, joka löytyy opinnäytetyön liitetiedostoista. Ulkoa katsottuna peliohjelmointi on ohjelmointia siinä missä minkä tahansa muu sovelluksen kehittäminen, mutta vaatii vision omaamista ja kykyä saada toimiva peli aikaiseksi. Osana työn tavoitetta oli lisätä tietotaitoa pelikehityksestä käyttämällä uutta ohjelmointikieltä pelin toteuttamiseen.

3.1 Ongelmien selvitys ja ratkaisujen soveltaminen

Jo suunnitteluvaiheessa todettiin, että tiettyjen asioiden toteutus vaatii ratkaisuja, joiden toteuttamisesta minulla ei ole kokemusta. Nämä olivat teknisiä haasteita, mutta myös mahdollisuuksia kehittää uusia tapoja lähestyä ongelmia ja ratkaista niitä. Tietyt ongelmat tulivat ilmi vasta kehitysvaiheessa tai määrittelyn muuttuessa, kun alkuperäistä suunnitelmaa jouduttiin muuttamaan, jotta saatiin aikaiseksi haluttu lopputulos.

Ongelmat ja niiden ratkaisut ovat aina tapauskohtaisia. Tämä johtuu siitä, että jokaisella projektilla on erilaiset vaatimukset, aikataulu ja resurssit. Joissakin tapauksissa ongelman ratkaiseminen voi tarkoittaa ominaisuuden poistamista tai muuttamista.

3.2 Kokemus ja oppiminen

Henkilökohtaisena tavoitteena käytin projektin etenemisprosessia tutustumiskeinona pelialaan ja peliteollisuuteen. Tavoite oli edistää omaa henkilökohtaista tietotaitoa pelikehityksestä sekä selvitä siinä vastaan tulevista haasteista, joita nykyajan peliohjelmointi tarjoaa pelikehitystä harrastaville.

Pelikehitys nykypäivänä on monipuolinen ala, joka käsittää grafiikkaa, ohjelmointia, musiikkia, ääniä ja taidetta. Ohjelmoijalle se antaa erittäin mielenkiintoisia haasteita toteutettavaksi, sillä vastassa on aina erilaisia määrittelyjä ja vaatimuksia. Itsenäinen pelikehitys mahdollistaa luovuuden käytön sekä vapauden päättää itse kehityksessä tapahtuvista muutoksista.

4 Pelikehittäjät nykyaikana

Pelejä kehittävät tahot voidaan luokitella karkeasti kahteen pääkategoriaan: isoihin pelitaloihin ja pienempiin Indie-kehittäjiin. Suurimmat ja tunnetuimmat pelit tulevat yleensä isoilta pelitaloilta, mutta digitaalisten jakelukanavien yleistyessä myös pienet pelikehittäjät ovat saaneet näkyvyyttä markkinoilla, kun perinteisen kauppalevityksen kustannukset eivät rajoita levitystä. Tässä luvussa esitellään lyhyesti näiden kahden tahon erot.

4.1 Pelitalot

Isot pelitalot saavat tyypillisesti rahoituksen julkaisijalta sen jälkeen kun julkaisijalle on esitelty pelin konsepti, alpha-versio tai muu näyttö tulevasta (tai valmiista) pelistä. Näiltä yrityksiltä julkaistaan tyypillisesti tunnetuimmat pelinimikkeet bisneksessä. Näistä esimerkkeinä ovat suositut FPS-pelit *Call of Duty* ja *Halo*-pelisarjat sekä RTS-pelejä edustava Starcraft-pelisarja.

Jotkut pelitalot ovat menestyneet niin hyvin yrityksenä, että niiden ei tarvitse hankkia erillistä julkaisijaa vaan ne voivat julkaista pelinsä itse. Näistä esimerkkinä tunnetun *Starcraft*-, *Diablo*- ja *World of Warcraft* –pelien kehittäjä *Blizzard Entertainment*, sekä *Half-Life:n* ja *Left4Dead:in* kehittäjä *Valve Software*. Vaikka ulkopuolista julkaisijaa ei vastaavassa tapauksessa tarvita, kehitys vaatii kuitenkin paljon resursseja.

Alla muutamia tunnettuja julkaisijoita jotka ovat kehittäneet menestyneitä pelejä:

- Blizzard Entertainment (Diablo- ja Starcraft-sarjat)
- Bungie (Halo-sarja)
- Atlus (Persona-sarja)
- Electronic Arts (Battlefield, The Sims)
- Konami (Metal Gear solid, Silent Hill)

4.2 Indie-kehittäjät

Termi *Indie Developer* tulee alun perin englanninkielen sanoista *independent developer*, joka tarkoittaa itsenäistä kehittäjää. Indie-kehittäjät eivät saa rahoitusta julkaisijalta ja tuotosten levitys tapahtuu yleensä internetin yli digitaalisesti. Nämä kehittäjät ovat yleensä pieni ryhmä ihmisiä tai joskus vain yksi henkilö joka vastaa kaikesta kehityksestä.

Tuotoksien julkaisu ja myynti hoidetaan tyypillisesti itse internetin välityksellä tai se myydään valmiina tuotteena julkaisijalle jälkikäteen, joka tapauksesta riippuen myös saa oikeudet sen käyttämiseen. Etuna tässä järjestelyssä on että taiteellinen ja tuotannollinen vapaus säilyy kehittäjillä, verrattuna julkaisijoista riippuviin kehittäjiin joiden tuotantoon julkaisijan päätökset vaikuttavat.

Joitakin tunnettuja indie-pelikehittäjiä ovat mm:

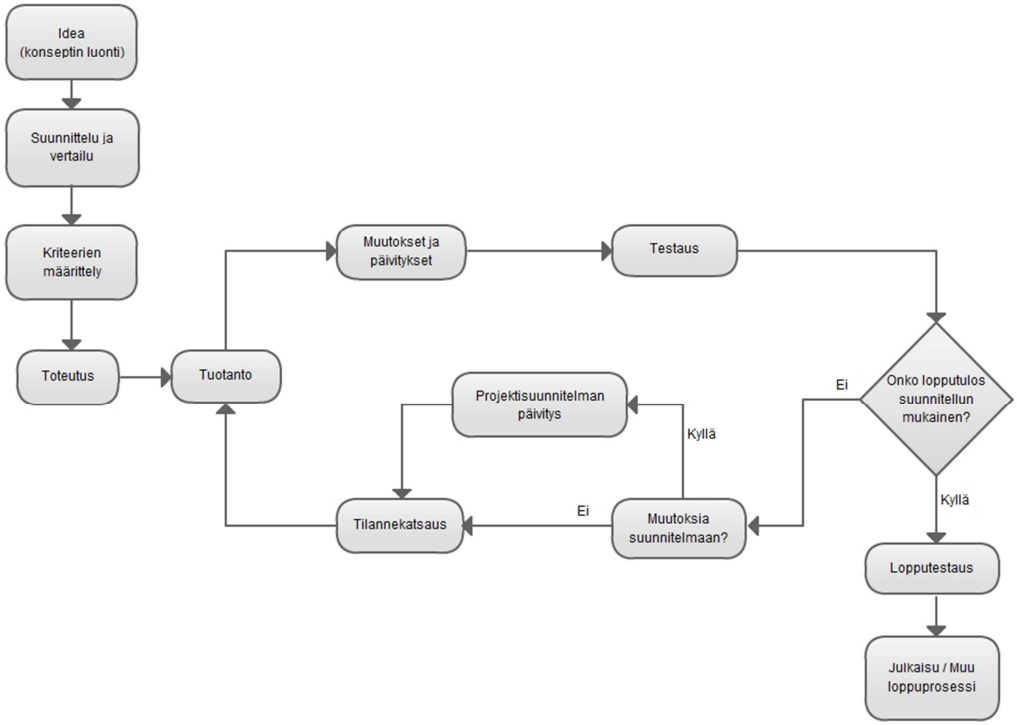
- Mojang Ab (Minecraft)
- 2D Boy (World of Goo)
- Frictional Games (Penumbra-sarja, Amnesia: The Dark Descent)
- Introversion Software (Uplink, DEFCON)
- Wolfire Games (Lugaru: The Rabbit's Foot)

Indie-pelit ovat yleensä pienempiä kokonaisuuksia verrattuna isojen pelitalojen peleihin ja suuri osa on toteutettu 2D-grafiikkaa käyttäen. Syy tähän on yleensä tuotannollinen, sillä kaksiulotteisen pelin ohjelmointi ja toteutus on yksinkertaisempaa kuin kolmiulotteisen. Tämä ei kuitenkaan ole estänyt niiden suosion nopeaa kasvua ja todistaa, että menestyvän pelin toteuttamiseen ei tarvita isoa rahoitusta tai kymmeniä työntekijöitä.

5 Kehitysprosessi

Tämä luku käsittelee kehitysprosessin kulkua ja sen eri vaiheita. Kuten kaikessa kehityksessä, myös pelikehityksessä on suotavaa olla etukäteen suunniteltu prosessi jonka mukaan projekti etenee.

Esimerkki:
Toteutusprosessikaavio liitteenä olevalle Spaceshooter-pelille oli seuraavanlainen:



Kuva 1: *Spaceshooter*-pelin prosessikaavio

Toteutus alkaa konseptilla, jonka pohjalta prosessia lähdetään suunnittelemaan. Idean pohjalta luodaan konsepti, jota jalostetaan yksityiskohdilla vision mukaiseksi. Suunnittelu- ja vertailuvaiheessa käydään läpi tarkemmin, mitä pelin tulee sisältää sekä vertaillaan tulevaa sisältöä kilpailevien pelien kanssa. Vertailun avulla on hyvä ottaa selvää erilaisten pelielementtien hyvistä ja huonoista puolista ja rakentaa niistä toimiva kokonaisuus. Tämän jälkeen käydään läpi kriteerit, mitä lopullisen tuotoksen tulee sisältää että sitä voidaan kutsua valmiiksi.

Alun suunnitteluvaiheen jälkeen prosessi jatkuu tuotantovaiheessa. Tuotantovaihe on käytännössä sykli jota toistetaan kunnes saavutetaan haluttu lopputulos. Tuotannon alussa mm. lisätään uusia ominaisuuksia, päivitetään resursseja ja hallitaan suunnitelman vaatimia muutok-

sia. Päivityksien jälkeinen askel on testaus, jonka aikana päivitetty sisältö testataan. Tämä ehkäisee ongelmia tulevassa kehityksessä kun tehdyt muutokset toimivat testatusti halutulla tavalla.

Testauksen jälkeen tarkastetaan onko lopputulos halutun mukainen. Toimiiko lisätty ominaisuus halutusti? Onko peli mahdollisesti valmis vai joudutaanko alkuperäistä suunnitelmaa muokkaamaan testausprosessista saadun tiedon perusteella? Mahdolliset muutokset päivitetään projektisuunnitelmaan ja vaatimuksiin. Tuotantosyklin lopussa on katsaus, arvioidaan projektin etenemistä, resurssien käyttöä, aikataulua sekä muita tekijöitä jotka vaikuttavat projektin etenemiseen. Tämän jälkeen tuotantosykli alkaa alusta ja tuotosta päivitetään suunnitelman mukaisesti.

Kun todetaan että lopputulos on suunnitellun mukainen ja täyttää vaatimukset, voidaan sille suorittaa lopputestaus. Lopputestauksen tarkoituksena on kitkeä pois mahdolliset ohjelmointivirheet, yhteensopivuusongelmat ja vastaavat ongelmat mitkä voivat vaikuttaa toimintaan. Testausprosessiin vaikuttaa luonnollisesti tuotoksen suunniteltu käyttöalusta, kohderyhmä ja käyttöympäristö.

Prosessin lopuksi peli voidaan julkaista. Julkaisu voi olla mikä tahansa loppuprosessi missä projekti julistetaan valmiiksi. Tämä ei välttämättä tarkoita kaupallista julkaisua, jos kyseessä on esimerkiksi ilmaispeli.

6 Pelin suunnittelu

Tässä kappaleessa käsitellään pelin suunnitteluun liittyviä vaiheita. Eri vaiheet on kuvailtu lyhyesti sekä niiden tarkoitusta selvennetty kokonaisuuden kannalta. Suunnittelu on tärkeä osa koko prosessia ja se antaa perustan kaikelle mitä kehityksen aikana tulee tapahtumaan. Se vaikuttaa työn tahditukseen, aikatauluihin, resurssien käyttöön ja testaukseen. On suositeltavaa että kaikki suunnitelmat ja muutokset dokumentoidaan talteen, sillä epäselvyydet kehityksen aikana voivat aiheuttaa ongelmia.

Suunnitteluvaiheessa oleellisinta on omata selkeä visio siitä mitä tulee tekemään. Uusia sovelluksia kehittäessä ilmenee aina ongelmia joita ei voida ennakoida, joten niistä selviäminen on oma haasteensa. Nämä ongelmat voivat viedä aikaa ja resursseja suunniteltua enemmän, tai niiden toteuttamisella voi olla epätoivottu vaikutus muuhun kehitykseen. Osa näistä ongelmista tulee ilmi vasta toteutusvaiheessa. Muutoksia suunnitelmaan joudutaan tekemään jälkikäteen, joka vuorostaan voi vaikuttaa lopullisen tuotoksen laatuun.

6.1 Tutkiminen

Alustava aiheen tutkiminen on hyvä toteuttaa etukäteen ennen varsinaisen työn aloittamista. Tutkimisen tarkoituksena on vastata tuotantoa kysymyksiin joilla tulee olemaan suuri vaikutus tuotantoprosessin aikana. Yleisiä kysymyksiä ovat mm.

- Mille alustalle peli toteutetaan?
- Mitä kieltä ohjelmoimiseen käytetään? Miksi?
- Mitä pelimoottoria käytetään?
- Tuleeko pelistä kaupallinen?
- Paljonko ja mitä resursseja on käytettävissä?
- Onko erityisrajoituksia mitä tulisi ottaa huomioon? (esimerkiksi älypuhelimien rajoitettu teho)
- Minkälainen kopiosuojaus tulisi olemaan?

Esimerkki:
Alustan valinta ratkaisee paljon tulevasta kehityksestä. Tuleeko peli pelkästään PC:lle, konsolille tai älypuhelimelle? Onko mahdollisuus että se ilmestyisi useammalle alustalle nyt tai myöhemmin? Entä mitä muutoksia tarvitaan jos alun perin konsolille toteutettu peli halutaankin toimimaan kotitietokoneella? PC on tyypillinen ja varma vaihtoehto, sillä se on pelikehitysalustana erittäin laajalle levinnyt ja hyvin dokumentoitu vaihtoehto. Pelin luonteesta riippuen (esim. yksinkertainen pulmanratkonta) voi sopia paremmin mobiililaitteille kun pelaaminen tapahtuu matkan aikana.

Käytetyn ohjelmointikielen valinta on myös tapauskohtaista, ja yleensä kannattaa valita sellainen kieli mitä hallitsee parhaiten. Ohjelmointikieliä käsitellään lisää kappaleessa 9.

Pelimoottori on runkosovellus jonka päälle koko peli rakennetaan, ja se käsittää tyypillisesti erilaisia hallittavia ominaisuuksia kuten pelimaailman painovoiman tai fysiikan mallinnuksen. Erilaisia pelimoottoreita on valmiita erilaisia käyttötarkoituksia varten, joten sen valinta on tapauskohtaista. Oman pelimoottorin ohjelmoiminen on luonnollisesti mahdollista, mutta monesti tarpeellisimmat ominaisuudet löytyvät jo olemassa olevista pelimoottoriratkaisuista.

Jos pelistä tulee kaupallinen, on kehittäjän syytä tutustua erilaisiin julkaisukanaviin ja levitysmahdollisuuksiin. Levitys voi tapahtua keskitetysti olemassa olevissa pelipalveluissa (*Steam*, *Desura*, *Origin*) jolloin peli saa tunnetun julkaisualustan ja potentiaalisen kookkaan yleisön. Toinen vaihtoehto on myydä omaa tuotostaan esim. vain sen omilla kotisivuilla, mutta ilman mainostusta sen tunnettavuus voi jäädä pieneksi.

Kehitykseen käytössä olevat resurssit ovat yleensä rajalliset. Indie-pelikehityksessä rahoitus voi olla pientä, muutamien tuhansien eurojen luokkaa tai sitä ei välttämättä saa lainkaan, vaan kaikki tarpeellinen on kustannettava itse. On olemassa kuitenkin ns. kickstarter-rahoituksia, joilla projektille voidaan saada rahoitusta ulkopuolelta. Eräs näistä on verkkosivu *http://www.kickstarter.com/*.

Joissakin tapauksissa pelikehityksessä joudutaan ottamaan huomioon erilaisia rajoittavia tekijöitä. Nämä tekijät voivat olla esim. suorituskykyyn liittyviä. Esimerkiksi pöytätietokoneet ovat monin verroin tehokkaampia laitteita kuin älypuhelimet, josta syystä niiden rajallinen suorituskyky on otettava huomioon kehityksessä. Vastaavat alustat eivät kykene käsittelemään yhtä näyttävää grafiikkaa tai monimutkaisia laskutoimituksia yhtä nopeasti, jonka takia älypuhelimella ajettava peli voi hidastua pelikelvottomaksi.

Kun peli julkaistaan kaupallisena, se yleensä edellyttää myös digitaalisen käyttöoikeuksien hallinnan (*DRM*) lisäämistä lopulliseen versioon. Tämä on toteutettavissa useilla eri menetelmillä. Muutamia tunnettuja tapoja nykypäivänä ovat mm. lisenssiavaimen syöttö ohjelmiston aktivoimiseksi tai vaatia käyttäjää luomaan tili sovelluksen käyttämiseksi, jolloin käyttöoikeudet ovat tiliin sidotut. Jälkimmäistä tapaa käytetään usein verkkopeleissä, joissa pelaaja joutuu olemaan jatkuvasti internetissä pelatakseen. Myös digitaalilevityspalvelut kuten *Steam* ja *Origin* ovat myös yksi tapa hallita käyttöoikeuksia.

Kehityksen yhteydessä tutkiminen käsittää muun muassa muiden virheistä oppimista. Moni menestynyt peli on voinut sisältää osia jotka ovat toteutettu väärällä tai epäoptimaalisella

tavalla. Syy tähän ei välttämättä aina ole tahallinen tai tietoinen ja tilanteeseen on voinut vaikuttaa aikataulu tai tekniset rajoitteet.

Analyysi:
Vuonna 2011 keväällä ilmestynyt *Duke Nukem Forever* sai negatiivista palautetta, kun pelaajalla sai olla pelissä kerrallaan vain kaksi asetta mukana. Tämä toteutus oli monelle pelisarjan edeltäviä osia pelanneelle suuri pelikokemusta haittaava tekijä, sillä edelliset osat antoivat pelaajalle mahdollisuuden kantaa aina kaikkia aseita mukanaan. Pelin kehittäjä julkaisi myöhemmin päivityksen, joka salli neljän aseen kannon samanaikaisesti.

Tämä suhteellisen yksinkertainen muutos kertoo tehokkaasti siitä, mitä pelaajat haluavat. Hyväksi todettuja kaavoja tai ratkaisuja ei ole aina suotavaa muuttaa jos ne toimivat. Duke Nukem Foreverin tapauksessa muutos oli teknisesti pieni mutta vaikutti pelikokemukseen epätoivotulla tavalla.

Vaikka ainutlaatuisia pelimekaniikkoja on mahdollista ja suotavaa kehittää, pelin perusrunko on yleensä toteutettu jossakin jo ilmestyneessä nimikkeessä. Pelejä on tuotettu valtava määrä vuosikymmenien aikana, mutta vain osa niistä on laajalle levinneitä ja yleisesti tunnettuja. Nämä menestyneet pelit ovat tunnettuja koska ne ovat onnistuneet toteuttamaan jotakin mikä saa pelaajan kiinnostumaa, pelaamaan ja sisältävät mielekästä tekemistä. Jos kehittäjä ei ole toteuttamassa omaperäistä pelikonseptia, joku muu on sitä jo mahdollisesti käyttänyt muodossa tai toisessa.

6.2 Vertailu

Vertailu on prosessi jonka tarkoituksena on vertailla omaa suunnitteluaan muiden suunnitteluun. Vertailussa olennaista on selvittää mitä muut ovat tehneet oikein ja mitä väärin. Tulosten avulla voidaan selvittää millainen ratkaisu tai toteutus halutaan omaan tuotokseen.

Vertailu eri vaihtoehtojen välillä ei aina ole yksinkertaista jos molemmissa on selviä hyötyjä ja haittoja. Tällaisia tapauksia varten suotavaa käyttää jonkinlaista benchmarking-vertailua, jossa otetaan huomioon kaikkien vaihtoehtojen hyvät ja huonot puolet. Tulosten perusteella voidaan päättää mikä soveltuu parhaiten omiin tarkoituksiin. Tätä vertailutapaa voidaan soveltaa esimerkiksi kun ollaan valitsemassa toteutustapaa pelille ja vaihtoehtona on 2D- tai 3D-grafiikat.

Kun halutaan vertailla pelien menestystä, on siihen monta erilaista tapaa. Yleensä tämä tarkoittaa tuttavapiiriltä kuultuja mielipiteitä ja omaa kokemusta. Universaalimpi tapa merkata pelien menestystä ovat kuitenkin käyttäjäarviot ja arvostelulehtien tai – sivustojen antamat arvosanat. Arvosanat ovat yleinen mittari pelin menestykselle, mutta niitä on syytä pitää vain suuntaa antavina. Korkean yleisarvonsanan saanut peli voi kuitenkin hävitä jossakin kategoriassa pelille, joka on saanut alhaisemman arvosanan.

Pelin suosio ja sen saamat käyttäjäarviot ovat suuntaa antava tekijä kun halutaan nähdä miten hyvin kyseinen peli on menestynyt. Se kertoo myös että pelissä on toteutettu onnistuneesti jotakin mistä pelaaja pitävät.

Metacritic.com on peliarvosteluille omistettu verkkosivusto jonka peleille antama arvosana on yleisellä tasolla mittapuu sille miten menestynyt kyseinen peli on. Metacritic antaa arvonsanansa keräämällä kymmenien eri peliarvostelusivustojen arvosanat yhteen ja laskemalla niistä verrattavan pistemäärän asteikolla 0-100. Sivusto sisältää myös yksittäisten käyttäjien arvioita tunnettujen arvostelusivustojen lisäksi.

Metacriticin kaikkien aikojen parhaimpien pelien listaan mahtuu paljon tunnettuja pelinimikkeitä kuten *Grand Theft Auto IV*, *World of Goo*, *Half-Life*, *Portal*, *Resident Evil 4* ja *Command & Conquer*. Lista on suuntaa antava ja ilmaisee käytännössä kaikkien mukana olevien sivustojen arvostelujen keskiarvon, mutta sivusto lukiessa voidaan nähdä mikä pelin on menestynyt ja miksi.

Sivustolla ensimmäisen sijan omaava *Grand Theft Auto IV* on brittiläisen *Rockstar Gamesin* julkaisema toimintaseikkailu, jossa pelaaja pääsee vapaasti tekemään tehtäviä, ajamaan ajoneuvoja ja tuhoamaan ominaisuutta. Peli on myös yksi kalleimmista ikinä tuotetuista peleistä, ja sen budjetti oli noin 100 miljoonaa dollaria. Pelin kehittäjä Rockstar North on brittiläinen yritys jossa työskentelee yli 100 työntekijää ja se on Take Two Interactiven tytäryhtiö. (Digitalbattle.com 6.11.2011)

Toisella sijalla listalla on *World of Goo*, joka on indie-kehittäjien *2D Boy* kehittämä. Pelissä on yksinkertainen idea: pelaajan on tarkoitus saada aikaiseksi kentän läpipääsyyn vaatimia rakenteita käyttäen pelissä esiintyvää limaa (*Goo*). 2D Boy -yrityksessä työskentelee vain kaksi ihmistä ja World of Goon budjetti oli noin 10.000 dollaria. (Venturebeat.com 6.11.2011)

Näillä kahdella pelillä on jaettu etusija Metacriticissä 98 pisteellä, mutta toinen on pienen budjetin ja muutaman ihmisen toteuttama indie-peli ja toinen on monikansallisen yrityksen tuottama peli miljoonabudjetilla. Tämä antaa vertailunäkökulman isojen pelitalojen sekä in-

die-pelikehittäjien menestyksestä ja viestii että pelin menestymiseen ei vaadita isoa rahoitusta. (Metacritic.com 6.11)

6.3 Peli-idea

Toteuttamiskelpoisen peli-idean kehittäminen on luonnollisesti olennainen osa pelisuunnittelua. Vaihtoehtoja, variaatioita ja toteutustapoja on rajattomasti. Menestyvän konseptin suunnittelu on haastavaa, mutta idean ei tarvitse myöskään olla täysin omaperäinen. Vanhasta ideasta voidaan saada aikaiseksi uusi pelikokemus tutkimisen ja vertailun avulla.

Jotkut pelit ovat olleet edellä aikaansa tai niiden toteutus ei ole ollut tasoa joka olisi kiinnostanut tai viihdyttänyt pelaajia vaikka idea on ollut varsin kehityskelpoinen. Esimerkkinä mainitusta on *Narbacular Drop*, joka ilmestyi vuonna 2005. Narbacular Dropissa pelaaja seikkailee luolastoissa käyttäen portaaleja, joita käyttäen pelaaja voi selvittää ongelmia ja edetä. Peli oli pelitekniseltä toteutukseltaan mielenkiintoinen mutta sen saama huomio jäi rajalliseksi toteutuksen vuoksi.

Muutamaa vuotta myöhemmin vuonna 2007 ilmestyi Valve Softwaren kehittämä Portal, joka käytti samaa ideaa ja modernisoi sen tavalla, jota pelaajat rakastivat. Portalin kehittämisen oli enemmän resursseja kuin Narbacular Dropin kanssa, mutta lopputuloksena oli laadukas peli josta pelaajat pitivät. Portal saikin erittäin positiivisen ja laajan vastaanoton.

Täysin omaperäisen idean kehittäminen ei ole pakollista, mutta huolellisesti toteutettu omaperäinen peli ainutlaatuisella toteutuksella menestyy todennäköisesti paremmin kuin vanhaa kaavaa käyttänyt peli joka ei tarjoa mitään uutta. Idean suunnitteluun ja jalostamiseen ei ole yksinkertaista ratkaisua ja kehityksen tulee tapahtua vertailun ja tutkinnan avulla.

6.4 Vaatimukset ja määrittely

Kehityksellä pitää olla vaatimukset ja kriteerejä joiden avulla tiedetään milloin on saavutettu haluttu lopputulos. Nämä kriteerit voivat olla mm. teknisiä, taiteellisia tai pelin tunnelmaan liittyviä. Vaikka toteutettava peli olisikin yksinkertainen, on sillä kuitenkin tietyt vaatimukset ja kriteerit mitä sen halutaan täyttävän. Määrittelyn tarkoituksena ei ole rajata kehitystä vaan ohjata sitä oikeaan suuntaan. Selkeät määritykset helpottavat kehitystä projektin aikana ja prosessi voidaan porrastaa eri askeleisiin joista jokainen sisältää oman saavutettavan lopputuloksensa.

Vaatimukset voivat olla jotakin yleistä toiminnallisuutta kuvaavia (esimerkiksi ”pelihahmon tulee kyetä lentämään”) tai tarkkaan määriteltyjä (esimerkiksi ”pelihahmon tulee kyetä lentämään vain tasolla 9 ja jos pelaaja on löytänyt lentämisen sallivan kyvyn”). Määrittely on alkuvaiheessa yleisellä tasolla toiminnallisuutta kuvaavaa ja tarkentuu kun ominaisuuden perustoiminnallisuus saadaan testattavaksi. Kehityksen edistyessä peli mukautuu määrittelyn vaatimuksiin mutta sama käy myös toisin päin. Määrittely voi muuttua kehityksen aikana useista syistä, kuten teknisistä rajoitteista tai resurssien puutteesta.

Määrittelyn voi dokumentoida käyttäen muistiinpanoja, dokumentointia, kuvaajia ja referenssimateriaalia. Määrittelyn tapa ja laatu on kehittäjäkohtaista ja joissakin projekteissa tarkempi määrittely tehdään vasta suunnittelun loppuvaiheessa kun tekninen toteutus on suunniteltu.

6.5 Ohjattavuus, käytettävyys ja interaktio

Pelin hallinta, käytettävyys ja koettu interaktio on tärkeässä asemassa kun pelikokemuksesta halutaan pelaajalle mieluista. Pelin antama välitön palaute on erittäin tärkeää, sillä palautteeton käyttöliittymä viestii pelaajalle että ohjaus ei toimi tai se toimii viiveellä. Pelaaja perustaa toimintansa opittuun toiminto-palaute – käyttäytymiseen, josta poikkeamisesta seuraa turhautuminen ja mielikuva laaduttomasta pelistä.

Muutamia esimerkkejä käytettävyydessä esiintyvistä ärsykkeistä joita tulisi välttää:

- Pelaaja tekee jotakin (esim. painaa näppäintä) mutta peli ei sitä rekisteröi tai pelaajan haluamaa asiaa ei tapahdu vaikka pitäisi
- Epälooginen käytettävyys (esim. valintajärjestyksen arvaamaton käyttäytyminen)
- Asioiden toteuttaminen monimutkaisesti kun siihen olisi yksinkertainen ratkaisu

Vaikka pelin tarinankerronta olisi huippuluokkaa ja grafiikka näyttävää, peliä ei ole miellyttävää pelata jos ohjattavuutta ei ole toteutettu hyvin. Ohjattavuus on kriittinen osa kaikessa missä liikutaan, koskien valikoita sekä pelihahmon liikuttamista. Liian herkät, hitaat tai huonosti tottelevat kontrollit turhauttavat pelaajaa kun liike on hallitsematonta ja ei käyttäydy odotusten mukaisesti. Tyypillisesti ohjattavuusongelmia näkee erilaisia ajoneuvoja ajaessa tai vastaavissa tilanteissa joissa ohjattavuus eroaa pelin normaalista ohjausskeemasta. Laadullinen ero voi johtua mm. siitä että vähemmän käytetyt ohjausskeemat pelissä eivät saa samanlaista laadullista huomiota kuin eniten käytetyt.

Viiveellinen palaute on myös asia jota tulee välttää. Se turhauttaa pelaajaa ja antaa kuvan että peli ei tottele komentoja koska palautetta ei tule, tai se annetaan väärään aikaan. Ellei kyseessä ole tarkoituksellinen ominaisuus ratkaista jokin pelitekninen seikka, palaute tulisi aina antaa pelaajalle välittömästi. Ääni- ja visuaaliset efektit ovat tätä varten hyvä ratkaisu, sekä tärinäominaisuutta tukevilla peliohjaimilla voidaan saada aikaiseksi fyysinen palaute.

Interaktio pelimaailmassa on syytä toteuttaa niin että se on miellyttävää ja loogista. Yksinkertaiset toiminnot, esimerkiksi kytkimen painaminen, ei saa vaatia erityistä tarkkuutta ilman että sille ole perusteltua syytä. Painamisen pitäisi onnistua kun pelaaja on tarpeeksi lähellä kytkintä. Esimerkki on kärjistetty, mutta samaa käyttölogiikkaa on hyvä soveltaa kaikkeen interaktioon mitä pelaaja tulee kokemaan.

Erilaisista peleistä kokemusta omaavan kehittäjän on helpompi tehdä päätöksiä kehityksen suhteen. Käytännössä tämä tarkoittaa että kehittäjä punnitsee eri vaihtoehdot ja valitsee sopivan tapauskohtaisesti. Eri peleissä esimerkiksi liikkuminen on toteutettu monella eri tapaa: jotkut pelit käyttävät liikkumiseen pelkkää näppäimistöä, toiset pelkästään hiirtä, kolmas ohjainta ja moni peli yhdistää hiiren ja näppäimistön.

6.6 Huomioita

Pelin alustava suunnittelu on pitkä prosessi joka koostuu useasta eri alueesta. Suunnittelun yksityiskohdat ovat aina projektikohtaisia, mutta tärkeimpinä ovat seuraavat:

- Tutki: selvitä mitä olet oikeasti tekemässä, ja rajaa alueet sen mukaan
- Vertaile kilpailua, hyviä ja huonoja puolia ja eri toteutusvaihtoehtoja
- Jalosta omaa ideaasi hyväksi todettujen ratkaisujen pohjalta ja pyri eliminoimaan negatiiviset puolet
- Määrittele saavutettavat tavoitteet ja ominaisuudet
- Huolehdi, että interaktio pelaajan ja pelin välillä on sulavaa

7 Visuaalinen suunnittelu

Tämä kappale käsittelee visuaalista ilmeen suunnittelua ja sitä varten huomioon otettavia asioita.

Pelin graafinen ilme määrittää luonnollisesti sen miten ihmiset pelin näkevät ja kokevat. Mielikuvituksellinen ja innovatiivinen peli voi kuitenkin jäädä pelaajalta kokeilematta jos sen graafinen ilme ei miellytä. Kaikille mieluista graafista ilmettä on lähes mahdotonta toteuttaa, mutta tärkeintä on että grafiikka sopii itse peliin ja siihen kuvaan mitä se itsestään antaa. Esim. avaruudessa tapahtuvalle pelille tyypillisiä grafiikoita ovat puhtaat värit ja futuristiset muodot.

Kuva 2: *Defense Grid*in käyttöliittymä ja tehtävävalikko. Visuaalinen tyyli on yhtenäinen ja valikot sopeutuvat pelin teemaan.

Grafiikan toteuttaminen peliin on vaativa prosessi, vaikka tavoitteena olisi minimalistinen ulkoasu. Esimerkiksi yksittäisen tekstuurin tekeminen hahmolle voi olla helppoa, mutta on haastavaa pitää kaikki grafiikka yhdenmukaisena ja yhteensopivana pelinuniversumin kanssa. Vaihtoehtoinen ratkaisu on käyttää vapaaseen käyttöön tarkoitettuja tekstuureja ja muita resursseja joita varten on sivustoja erityisesti pelikehitystä varten.

7.1 Sijaisgrafiikat

Kaiken grafiikan ei tarvitse olla valmista heti kun tuotanto aloitetaan. Yksi hyvä ja yleinen tapa on käyttää ns. *placeholder*-grafiikoita eli sijaisgrafiikoita kehityksen aikana. Sijaisgrafiikat ovat nimensä mukaisesti oikean grafiikan sijasta käytettäviä. Syy niiden käyttöön on että lopullisessa tuotoksessa käytettävä grafiikka ei ole vielä valmista.

Sijaisgrafiikat ovat yleensä yksinkertaisia muotoja selkeästi erottuvilla väreillä, joka helpottaa niiden hahmottamista taustasta kehityksen aikana mutta sijaisgrafiikkana voi luonnollisesti käyttää mitä vain grafiikkaa halutaan.

7.2 Graafinen ilme

Pelin graafinen ilme riippuu siitä miten kehittäjä haluaa pelin maailman esittää pelaajalle. Esimerkiksi sarjakuvamaisella ilmeellä ja värikkäillä grafiikoilla voidaan tehokkaasti viestittää että peli on kevytmielinen seikkailu. Tummia värejä, teemoja ja keskiaikaisia käyttöliittymäelementtejä käyttämällä voidaan luoda tunnelma vakavammasta ja mahdollisesti realistisemmasta pelistä.

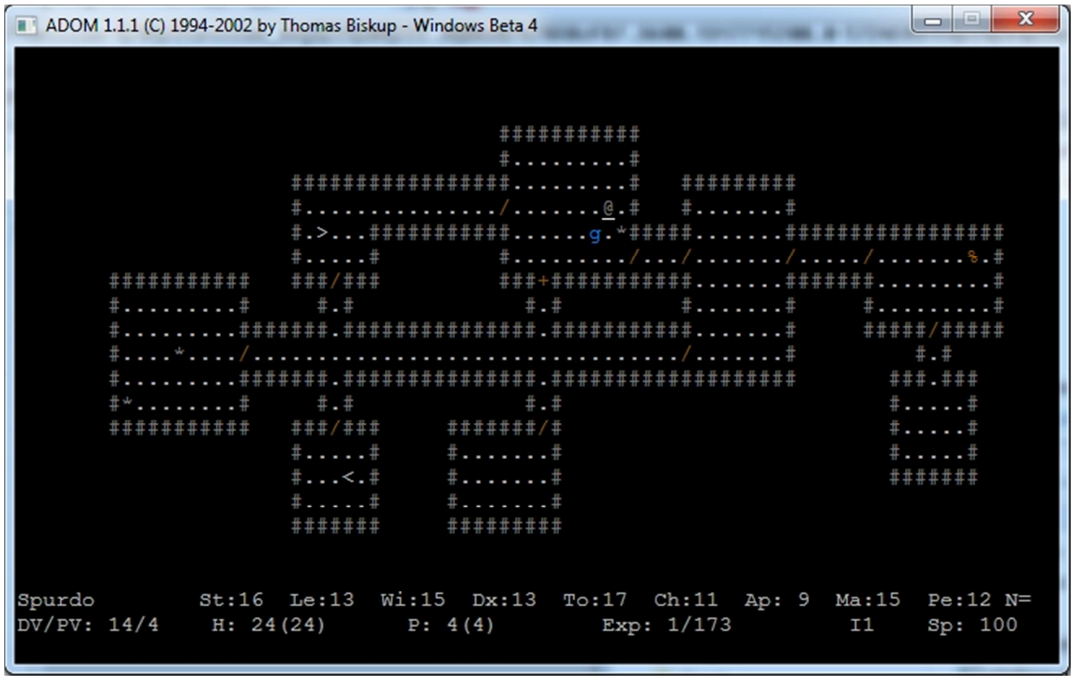
Kehittäjä haluaa luoda näyttävää katseltavaa kun kyse on visuaalisesta toteutuksesta. Peliin toteutettava grafiikka on kuitenkin vain osa kokonaisuutta johon kuuluu lisäksi mm. äänet, ohjaus ja tarina. Tämä tarkoittaa käytännössä että graafisen ilmeen ei tarvitse olla huipputasoa jos muut osat kokonaisuudesta ovat hyvin toteutettuja. Asialla on myös varjopuoli, sillä jos graafinen panostus on pois muilta pelin tekeviltä osa-alueilta voi lopputuloksena olla visuaalisesti näyttävä ja efektejä täynnä oleva peli joka ei omaa kiinnostavaa tarinaa, sisältää vähän sisältöä tai ei tarjoa pelaajalle miellyttävää tekemistä.

Realistiset grafiikat ovat nykyaikana nähtävillä monessa pelissä mutta mukaan mahtuu myös paljon innovatiivisia toteutuksia. Pelin ei aina kuitenkaan tarvitse olla graafisesti näyttävä jos pelin luonne ei sellaista edellytä. Realismia simuloivissa peleissä halutaan luonnollisesti käyttää mahdollisimman tarkkoja ja realistisia tekstuureita.

Positiivisina esimerkkeinä grafiikan minimaalisuudesta toimivat ns. roguelike-pelit kuten *Nethack* ja *Ancient Domains of Mystery* (ADOM). Roguelike-peleille yksi ydinominaisuuksista on että kaikki ”grafiikka” on olematonta ja pelimaailma esitetään täysin ASCII-merkeillä. Tästä huolimatta ne ovat erittäin monipuolisia pelejä joissa on paljon sisältöä ja ovat mielenkiintoisia pelata.

Roguelike-peleille tyypillisiä piirteitä ovat ASCII-grafiikan lisäksi luolastoseikkailut, pelihahmon tavaroiden hallinta ja että pelaajan löytämät tavarat ovat tuntemattomia ennen kuin ne tunnistetaan. Tällä tarkoitetaan että maasta löydetty tavallisen niminen tikari voi olla esimerkiksi kullattu tikari jossa on erikoisominaisuuksia.

ASCII-grafiikka on kärjistetty tapaus, mutta liian yksinkertaisia grafiikoita käyttävät pelit voivat myös karkottaa mahdollisia pelaajia jos nämä kokevat että haluavat pelin olevan graafisesti miellyttävämpi.



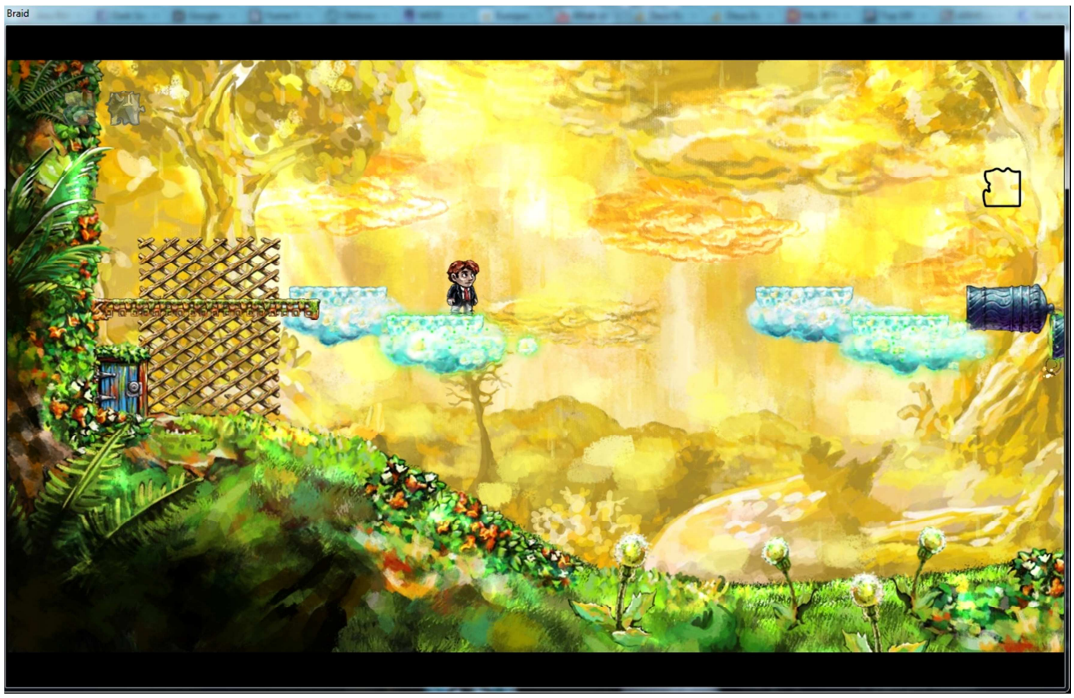
Kuva 3: *Ancient Domains Of Mystery* (ADOM), tyypillinen roguelike-seikkailu

Joissakin tapauksissa grafiikka on toissijainen tekijä ja pelattavuus menevät sen edelle. Tästä esimerkkinä on peli nimeltä *Minecraft* jonka pelimaailma koostuu täysin palikoista ja materiaalikuutioista ja pelaajan on vapaa luomaan mitä haluaa näistä rakennusmateriaaleista. Minecraft on graafisesti vaatimaton peli ja tekstuurit ovat toteutettu matalla resoluutiolla. Sen ei ole tarkoitus olla realistinen, vaan pelin ideana on asioiden luominen ja rakentaminen.

Vaatimattomat grafiikat voivat olla myös tekninen ratkaisu, sillä liian näyttävät efektit tai korkearesoluutioiset tekstuurit vievät myös käyttöalustan resursseja eikä peli enää pyöri miellyttävällä nopeudella.

Kuva 4: *Minecraft* ja palikoista koostuva maailma.

Mahdollisimman realistinen grafiikka ei ole aina kuitenkaan paras ratkaisu ja näyttävämmän lopputuloksen voi saada aikaiseksi omaperäisellä visuaalisella tyylillä. Pelin luonteen ja tunnelman vuoksi voi olla suotavampaa käyttää surrealistista grafiikkaa, maalaukselta näyttäviä taustoja tai koomiselta näyttäviä pelihahmoja.



Kuva 5: *Braid*, indie-peli joka tuli tunnetuksi omaperäisestä toteutuksestaan ja näyttävästä grafiikasta. Braid käytti ajan manipuloimista peliementtinä onnistuneesti.

Tärkeä huomioon otettava asia graafisessa suunnittelussa on sen erottuvuus ja selkeys. Yleensä ruudulla näkyy paljon pelaajalle tarkoitettua tietoa kuten energiamittari, panosten lukumäärä, pelaajan ase ja mahdollisesti myös muuta tietoa. Mm. MMO-pelien graafinen käyttöliittymä on täynnä paljon erilaista tietoa peliin liittyvistä asioista.

Ellei erityisesti haluta että visuaaliset elementit ovat huonosti erottuvia (esimerkiksi jonkun efektin takia), niiden tulisi aina erottua selkeästi muusta taustalla tapahtuvasta toiminnasta joka tilanteessa.

7.3 Yhtenäisyys

Graafisella yhtenäisyydellä tarkoitetaan sitä, että käytetyn grafiikan tyyli pysyy samanlaisena kautta pelin. Tämä tarkoittaa myös valikoita ja muita valintaruutuja joita peli sisältää. Keskiajalle sijoittuva peli sisältää todennäköisesti paljon teemoja ja elementtejä kyseiseltä aikakaudelta, joten futuristinen graafinen käyttöliittymä rikkoo yhtenäisyyden. Erilaisen grafiikan käyttö on kuitenkin suotavaa jos se palvelee pelin teknistä tai tarinallista tarkoitusta. Oikein toteutettuna erilaiset graafiset tyylit voivat siis olla pelikokemusta rikastuttavia eikä siihen negatiivisesti vaikuttavia. Graafista yhtenäisyyttä rikkovia grafiikoita voidaan käyttää myös halutun tunnelman aikaan saamiseksi, joten niiden käyttö on tapauskohtaista.

Graafisen ilmeen luomiseen ei ole muita rajoittavia sääntöjä, kuin että sen tulisi olla miellyttävä silmälle ja loogisen oloinen. Negatiivisella tavalla silmiin pistävä grafiikka ja efektien ylikäyttö voi turhauttaa pelaajaa ja tehdä pelistä epämiellyttävän pelata. Liian samantapaiset efektit pelin sisäisissä erilaisissa asioissa kuten pelaajan tavaroissa ovat huono idea. Jos pelaajalla on mukanaan esimerkiksi parannusjuoma sekä myrkkyjuoma joista molemmat ovat visuaalisesti hyvin samanlaisia, voi pelaajalle olla ikävä yllätys juodessaan vahingossa väärästä pullosta. Näiden kahden erottaminen visuaalisesti on silti helppoa, sillä pullojen muodon, koon ja tekstuurin voi vaihtaa että ne erottuvat toisistaan selkeästi.

7.4 Suunnittelussa huomioitavaa

Pelin visuaalinen suunnittelu ja toteutus voi olla haastavin osuus monelle kehittäjälle. Alustavasi suunnitellut hahmot ja maastot voivat kokea isoja muutoksia kehityksen aikana ja syitä voi olla useita. Joskus hahmoja joudutaan muuttamaan että ne sopeutuvat pelin tarinaan ja maailmaan paremmin kuin edelliset.

Suunniteltujen visuaalisten tyylien kehittyminen prosessin mukaan on normaalia ja jopa suotavaa, sillä jokaisen kehityksen osa-alueen on oltava valmis muuttumaan suunnitelmien muuttuessa. Visuaalisen ilmeen suunnittelussa oleellisia asioita ovat:

- Sijaisgraafikoiden käyttö: lopulliset grafiikat eivät yleensä ole saatavilla heti alkuun
- Graafinen ilme tulisi näyttää kyseiseen peliin kuuluvalta
- Visuaalisten elementtien yhtenäisyys on tärkeää pelin vakuuttavuuden kannalta

8 Ohjelmointikielen valinta

Oikean ohjelmointikielen valinta ei välttämättä ole itsestään selvää vaikka prosessi on yksinkertainen. Suurin rajoite ohjelmointikielen valitsemiselle voi olla rajapinta jolle ei löydy tukea kehittäjän haluamalle kielelle. Käytännössä kaikille yleisesti käytössä oleville rajapinnoille löytyy implementaatio monille eri ohjelmointikielille.

Ohjelmoinnin voi toteuttaa yleensä millä tahansa kielellä, mutta toiset kielet soveltuvat paremmin peliohjelmointiin kuin toiset. Peliä ohjelmoitaessa koodin lopullinen suoritusnopeus on tärkeää: laskutoimituksia voi tapahtua jatkuvasti satoja tai tuhansia kappaleita jokaisella ruudunpäivityksellä. Pelin tehovaatimuksista riippuen tämä voi aiheuttaa pätkimistä tai nykimistä jos suoritusalustalta loppuvat tehot kesken. Tästä syystä yksi suosituimmista peliohjelmointiin käytetyistä kielistä on C++. Se on ohjelmointikielenä erittäin nopea, suosittu ja tunnettu kieli. Sitä tuetaan lähes kaikilla mahdollisilla alustoilla ja sille löytyy suuri määrä erilaisia aputyökaluja.

Valinta on kehittäjällä ja lähes samaan lopputulokseen päästään käytännössä kaikilla kielillä. Kieli on vain työkalu kehittäjän ja tietokoneen välillä. Vaikka ohjelmointikielissä on paljon eroja syntaksissa, nopeudessa ja luettavuudessa, niiden perusrakenne on hyvin samanlainen.

Alaosioissa on lyhyt kuvaus ja vertailu muutamista suosituista peliohjelmointikielistä.

8.1 Assembly

Assembly on laiteläheinen ohjelmointikieli (*low-level programmin language*) ja on tämän ansiosta myös suorituskyvyltään erittäin nopea. Sillä ohjelmointi on korkeamman tason ohjelmointikieliin (kuten C++ tai Java) suhteellisen haastavaa, ja pelikehityksessä sitä käytetään paikoissa joissa vaaditaan ehdotonta suorituskykyä. Pienissä peleissä käytetyn abstraktimman ohjelmointikielen (kuten C++) suoritusnopeus on yleensä riittävä, joten Assemblyn käyttöä kannattaa harkita tapauskohtaisesti.

8.2 C++

C++ on olio-ohjelmointia tukeva kieli joka on pelikehityksessä erittäin suosittu nopeutensa ansiosta. Se on laajalti tunnettu kieli jolle on paljon erilaisia kehitystyökaluja, rajapintoja ja tukidokumentaatiota. C++:n etu ja haitta on että se sallii kehittäjän itse tehdä ratkaisut tiettyjen asioiden, kuten muistin käytön suhteen. Tämä mahdollistaa virheet kehittäjän puolesta, joten koodin huolellinen kirjoitus C++:lla on tärkeää. Tästä syystä muistivuodot ovat mahdol-

lisia, sillä C++ ei sisällä minkäänlaista automaattista muistinhallintaa vaan jättää sen käyttäjän vastuulle.

8.3 Python ja Lua

Python ja Lua ovat ns. skriptauskieliä joita käytetään yleisesti peliohjelmoinnissa lisäominaisuuksien tuottamiseen ja niillä harvemmin ohjelmoidaan koko peliä. Skriptikielten syntaksi on yleensä ohjelmakohtainen, sillä skriptaukselle avoinna olevat toiminnot ovat valmiiksi määritelty itse ohjelman koodissa. Aiheeseen liittyvänä esimerkkinä, mm. World of Warcraft käyttää lisäosissaan Luaa skriptauskielenä ja mahdollistaa käyttäjien tehdä itse lisäosia.

8.4 Java

Javan ehdoton etu verrattuna muihin ohjelmointikieliin on sen järjestelmäriippumaton kehitys. Ideaalisesti kaikki Javalla ohjelmoitu toimii kaikilla Javaa tukevilla järjestelmillä identtisesti ilman koodimuutoksia, ellei kehittäjä ole ohjelmoinut käyttöjärjestelmästä riippuvaa koodia tarkoituksella. Haittapuolina on hitaus verrattuna C++:n ja käytännössä kaikilta konsoleilta puuttuva tuki Javalle.

8.5 Ohjelmointkielen valinta

Ohjelmointikielet, kuten rajapinnatkin, ovat vain työkalu jota käytetään lopullisen tuloksen tuottamista varten. Kun on kyse suoritusnopeudesta, jaetaan ohjelmointikieliä sen perusteella miten laiteläheisiä (*low-level programming language*) ne ovat. Mitä laiteläheisempi ohjelmointikieli on, sitä nopeampi suorituskyky sillä on. Laiteläheiset ohjelmointikielet ovat yleensä syntaksiltaan vaikeita verrattuna korkean tason ohjelmointikieliin, jonka takia ohjelmointi tapahtuu yleensä korkean tason ohjelmointikielillä. Nykyaikaisella laitteistolla laiteläheisten tuoma nopeusetu voi olla minimaalinen, joten niiden käyttö jää tapauskohtaiseksi.

	Assembly	**C / C++**	**Python / Lua**	**Java**
Laiteläheisyys	Erittäin laiteläheinen	Laiteläheinen	Korkean tason kieli	Korkean tason kieli
Soveltuvuus	Suorituskykyä vaativat järjestelmät	Ohjelmistot, pelit, käyttöjärjestelmät	Pelit, automatisointi	Pelit, Ohjelmistot
Dokumentaatio, pelikehitys (1-10)	2	10	8	8

Taulukossa on eri ohjelmointikielet ominaisuuksineen vertailtuna. Tuloksena voidaan todeta, että C-kielet ovat yleisesti paras vaihtoehto kun halutaan ohjelmoida peli. Se on kielenä nopea ja laajasti käytetty peliteollisuudessa, sekä saatavilla oleva dokumentaatio pelikehitykseen liittyen on runsasta. C-kielet ovat kuitenkin laajoja kokonaisuuksia ja jättävät ohjelmoijan vastuulle paljon. Niillä on helppoa ohjelmoida hidasta tai huonoa koodia, sillä ne eivät sisällä kehittyneempiä ominaisuuksia korkeamman tason kielistä, kuten ns. roskien keräystä (*garbage collection*) joka ehkäisee mm. muistivuotoja.

On syytä huomioida, että kokenut Java-ohjelmoija voi saada Javalla aikaiseksi paremman lopputuloksen kuin keskitason C-osaaja. Ohjelmointikielen valinnassa on suotavaa käyttää hyödyksi aiempaa osaamista eikä vain siirtyä uuteen, tuntemattomaan kieleen. C-kielet ovat pelikehityksessä yleisiä, joten niiden tunteminen on silti ehdottomasti eduksi.

9 Rajapinnat

Tässä kappaleessa esitellään yleisimmät rajapinnat, niiden käyttötarkoitus ja soveltuvuus pelikehitykseen.

Rajapinnat (englanniksi *API*, *Application Programming Interface*) ovat käytännössä ennalta määriteltyjä sääntöjä ja määrityksiä, joiden avulla eri ohjelmat tai ohjelman osat voivat keskustella keskenään. Rajapintoja on monia ja yleensä ne omistautuvat vain yhdelle asialle, esimerkiksi grafiikalle (*OpenGL*, *Direct3D*, *SDL*) tai äänelle (*DirectSound*, *ALSA*, *OSS*).

Rajapintojen käyttämisen etuna on standardisoitu, testattu sekä yksinkertaistettu käyttö alemman tason toimintojen kanssa. Esimerkiksi jos ruudulle halutaan piirtää neliö: ilman rajapintaa saatetaan joutua piirtämään jokainen neliön reunan erikseen ruudulle eri komennoilla. Rajapinnan kanssa voidaan taas sanoa suoraan että halutaan piirtää neliö ja parametriksi annetaan vain neliön reunan pituus ja koordinaatit; rajapinta hoitaa tietojen perusteella neliön piirron ruudulle. On syytä tiedostaa, että jotkut rajapinnat käyttävät alemman tason rajapintoja suorittaakseen esimerkiksi grafiikan piirtoa. esimerkiksi SDL käyttää OpenGL:ää grafiikan piirtämiseen: SDL on oma rajapintansa, mutta grafiikan piirrosta taustalla vastaa alemman tason OpenGL.

9.1 DirectX ja Direct3D

DirectX on tarkemmin sanottuna kokoelma erilaisia rajapintoja eri tarkoituksiin jotka kulkevat tämän nimen alla. Muutamia esimerkkejä ovat *Direct3D*, *DirectSound* ja *DirectPlay*. Direct3D on 3D-rajapinta jolla on täysi tuki 3D- ja 2D-renderöinnille ja sitä käytetään luonnollisesti näitä ominaisuuksia vaativissa sovelluksissa, kuten peleissä. DirectSound on rajapinta äänikortin ja ohjelman välille, jolla voidaan hallita ohjelman tuottamia ääniä. DirectPlay on ohjelmakirjasto (*library*) verkkoyhteyksien hallintaan joka on tarkoitettu pääasiassa pelikehitystä varten, mutta luonteensa takia sitä voidaan käyttää missä vain verkkotoimintoja vaativissa ohjelmissa. DirectX on Microsoftin yksityisomistuksellinen rajapinta eikä sitä tästä syystä tueta muilla käyttöjärjestelmillä. DirectX on Microsoftin omistama ja tukema joten käyttöjärjestelmäintegraatio ei ole yleensä ongelma.

9.2 OpenGL

OpenGL (Open Graphics Library, vapaa grafiikkakirjasto) on standardi spesifikaatio ja järjestelmäriippumaton alusta 2D- ja 3D-grafiikan piirtoon. Sitä kehittää voittoa tavoittelematon

Khronos Group konsortio ja se on Direct3D:n kanssa toinen suosituimmista grafiikkarajapinnoista niin videopeleissä kuin muissakin graafisissa sovelluksissa.

Avoimen käyttölisenssinsä ansiosta OpenGL on käytössä laajasti Linux/UNIX- ja Mac-käyttöjärjestelmissä. OpenGL on vain grafiikan piirtämiseen tarkoitettu rajapinta, eikä se sisällä tukea esim. audiolle tai verkkotoiminnoille.

9.3 SDL

SDL (*Simple Directmedia Library*) on järjestelmäriippumaton rajapinta, joka tarjoaa pääsyn mm. ääni-, video-, hiiri- ja näppäimistötoimintoihin. SDL käyttää 3D-grafiikan piirtämiseen OpenGL-rajapintaa. Monipuolisten järjestelmärajapintojensa ansiosta se on pelikehitykseen erinomainen valinta, sillä se sisältää pelikehitykseen lähes kaiken tarvittavan yhdessä paketissa, lukuun ottamatta verkkotoimintoja.

9.4 Javan rajapinnat

Java on teknisesti ohjelmointikieli mutta omaa erilliset rajapinnat mm. grafiikalle ja äänille joten se on osaksi omassa luokassaan. Sen etuna on yhteensopivuus kaikkien Javaa tukevien järjestelmien kesken mutta haittapuolena on hitaus ja huono tuki konsoleilla. Java-ohjelmat ajetaan Javan omassa virtuaaliympäristössä joka on jokaisella käyttöjärjestelmällä identtinen, mikä takaa toimivuuden kaikilla alustoilla jos ohjelmoinnissa ei ole käytetty käyttöjärjestelmäkohtaisia riippuvuuksia. Raskaasti grafiikkapainotteiset ja nopeaa suorituskykyä vaativat ohjelmat kuten modernit pelit eivät virtuaaliympäristön emuloinnin hitauden vuoksi toimi yleensä vaadittavalla nopeudella. Pelien toteuttaminen Javalla on kuitenkin mahdollista, mutta sen rajoittunut suorituskyky on hyvä ottaa huomioon suunnitteluvaiheessa. Yksi tunnetuimmista Javalla toteutetuista peleistä on *Minecraft*.

9.5 Konsolirajapinnat

Pelikonsolit käyttävät yleensä omia rajapintojaan ja niille kehittäminen vaatii yleensä SDK-ohjelmiston jonka voi saada tiettyjä kriteerejä vastaan konsolin valmistajalta (tyypillisesti konsolin valmistajan myöntämä käyttölisenssi). Vertailun vuoksi todettakoon että Playstation 3 käyttää grafiikkarajapintanaan PSGL:ää joka on muunneltu versio OpenGL ES:stä. Xbox-pelikonsolit käyttävät muunneltuja versioita Microsoftin DirectX-rajapinnoista.

9.6 Rajapinnan valinta

Oikean rajapinnan valinta on tapauskohtaista sillä kaikkiin tapauksiin sopivaa yhtenäistä rajapintaa ei ole olemassa, vaikka samaan tarkoitukseen on standardoitu useita rajapintoja. Jokainen rajapinta soveltuu eri käyttötarkoitukseen, joten vaihtoehtojen vertailu on suositeltavaa. Rajapinnan valintaan vaikuttaa eniten käyttöjärjestelmä ja alusta mille peliä toteutetaan.

Muita rajapinnan valintaan vaikuttavia tekijöitä on mm:

- Ovatko kyseessä pöytäkoneet, kannettavat, älypuhelimet vai konsolit?
- Tarvitaanko 3D-grafiikkaa ollenkaan vai riittääkö 2D-grafiikka?
- Vaatiiko peli paljon suorituskykyä? Pitääkö rajapinnan olla nopein mahdollinen?

Näkyvimpiä käyttöeroja rajapintojen käytöstä peliteollisuudessa on Windows-käyttöjärjestelmille olevien pelien määrä verrattuna esim. Linux- tai Mac-käyttöjärjestelmiin. Suuri osa moderneista peleistä käyttää DirectX–rajapintaa joka rajoittaa niiden yhteensopivuuden Windows-järjestelmiin.

10 Ohjelmointi

Tässä kappaleessa käsitellään ohjelmointiosuutta ja siihen liittyviä huomioonotettavia asianhaaroja. Käsittelyssä on mm. yleiset ohjelmointikäytännöt, versionhallinta ja virheiden käsittely.

Peliohjelmointi on tekniseltä kannalta samanlaista kuin mikä tahansa ohjelmointi. Koodia kirjoittaessa kannattaa noudattaa yleisiä käytäntöjä vaikka lähdekoodia ei kukaan muu tulisi näkemäänkään. Seuraavat linjaukset eivät ole absoluuttisia, vaan niiden tarkoitus on olla ohjesääntönä hyvän koodin kirjoittamiseen.

1. Kommentoi – hyvin kommentoitu koodi kertoo muille kehittäjille ja itsellesi mitä kyseinen funktio tekee. Kommentointiin riittää nimi tai lyhyt kuvaus.
2. Nimeä järkevästi – looginen nimeämislogiikka toiminnoille auttaa virheiden jäljitystä ja helpottaa kokonaiskuvan hahmottamista.
3. Kirjoita selkeästi – kirjoita nimet, funktiot ja viittaukset selkeästi ihmistä varten, älä tietokonetta. Kirjoita yhden rivin mitään sanomattoman vaikealukuisen koodin sijasta useamman rivin selkeä rakenne mistä näkee mitä koodissa tapahtuu.
4. Käytä moduuleita – eristä kaikki toiminnot omiin kutsuttaviin funktioihinsa. Tämä helpottaa virheiden jäljitystä ja jälkikäteen tulevia koodimuutoksia.
5. Toiminnan kautta nopeuteen eikä päinvastoin – jos koodin pitää olla mahdollisimman optimoitua, keskity aluksi siihen että saat sen aluksi toimimaan. Kun koodin toiminta on varmistettu, sen jälkeen sitä voidaan nopeuttaa.
6. Käytä sulkeita – sulkeet helpottavat hahmottamista: tietokone laskee aina oikein, ihminen ei.

```
-- Oikein, helppolukuinen
if ((x == y) or (a ~= b)) then …
while (x >= 10) do …

-- Väärin, vaikea lukea ja jättää varaa virheille ja tulkinnalle, vaikka
lopputulos on identtinen
if x == y or a ~= b then …
while x >= 10 do …
```

7. Muistin vapauttaminen – jos ohjelmoidaan kielellä joka ei omaa automaattista muistinhallintaa (esim. C++), vapauta varattu muisti takaisin järjestelmän käyttöön.
8. Käsittele virheet – käsittele mahdolliset virheet koodissa aina jos niitä on mahdollista tapahtua kesken ohjelman suorituksen.
9. Älä käytä GOTO-komentoja – niiden käyttö sekavoittaa koodin tulkintaa.
10. Käytä muuttujia oikein – määritä muuttujat vasta kun niitä tarvitaan ja pidä ne erillään muusta koodista. Vältä globaaleja muuttujia.

11. Pidä funktiot yksinkertaisina – suuria funktioita on hankala tulkita. Siirrä toiminnot omiin alifunktioihinsa joista jokainen hoitaa oman tehtävänsä.

Ohjesäännöt ovat yleispäteviä ja tilanteesta riippuen voidaan niistä poiketa tarpeen vaatiessa. Jo valmiiksi tietyillä ohjesäännöillä kirjoitettua koodia ei ole viisasta lähteä muuttamaan, mutta kaikki jälkikäteen lisätty koodi olisi hyvä tehdä ohjesääntöjen mukaisesti, tai ainakin kunnioittaen vanhaa käytäntöä.

10.1 Versionhallinta

Versionhallintatyökalut ovat olennainen osa ohjelmistokehitystä. Versionhallinta on käytännössä pisteitä projektin eri vaiheista ja siihen tapahtuneista muutoksista. Kaikista versionhallintaan tallennetuista versioista voidaan saada tarvittaessa tiedoston tai projektin vanha versio. Tämä on kätevä ominaisuus jos koodiin tehdyt muutokset eivät toimi ja toimivaan tilaan palauttaminen käsin on liian monimutkaista: voidaan ottaa versionhallinnasta ennen muutoksia tallennettu versio jonka päälle kehitystä voidaan jatkaa.

Versionhallintatyökaluja on monia erilaisia joista tunnetumpia ovat *SVN*, *Git* ja *CVS*. Eroavuudet eri työkalujen välillä ovat pääasiassa edistyneissä ominaisuuksissa ja teknisessä toteutuksessa, kuten nopeudessa ja tietokannan koossa. Yleisesti ottaen ei ole väliä mitä ohjelmistoa käyttää versionhallintaan sillä kaikki tekevät samaa asiaa mutta hieman eri tavalla. Erikoisominaisuuksia tarvitsevalle on suositeltavaa vertailla eri työkalujen ominaisuuksia.

10.2 Valmiin koodin käyttäminen

Ohjelmoinnin aikana tulee vastaan myös ongelmia joihin ei aina itse tiedä vastausta. Peliohjelmoinnin ollessa kyseessä etuna on että ongelmaan on todennäköisesti jo ratkaisu olemassa. Tavallisesti vastausten etsimiseen riittää haku internetistä hakukoneella. Yksityiskohtaisempia kysymyksiä varten on hyvä liittyä mm. pelikehityskommuuneihin joilla on omat forumit sekä IRC-kanavat keskustelua varten. Kehitysalustakohtaisia kysymyksiä varten parempi vaihtoehto voi olla kyseistä alustaa varten oleva kommuuni, kuten iDevGames.com-sivusto joka erikoistuu Mac- ja Apple iOS –pelikehitykseen. Erilaisia kommuuneja on satoja joten niiden väliltä valitseminen on mieltymyskysymys. Yksi suurimmista pelikehityskommuuneista on *gamedev.net*-sivusto, joka tarjoaa käyttäjilleen yllämainitut palvelut.

Valmiiden ratkaisujen sisällyttäminen ja soveltaminen omiin tarpeisiin on suotavaa, sillä monesti asia on toteutettu paremmin etukäteen joten ole syytä alkaa itse kehittämään samaa

ratkaisua. Ulkopuolisen koodin kopioimisessa on kuitenkin omat ongelmansa. Tunnettujen ohjelmointikirjastojen (*software library*) käyttäminen on yleensä turvallista, mutta erikoistapauksiin tehdyt kirjastot voivat olla epävakaita tai jopa tietoturvariskejä ja voivat tästä syystä aiheuttaa ongelmia kehityksessä tai toiminnassa. Yhteensopivuussyistä on suositeltavaa käyttää tunnettuja ja testattuja kirjastoja aina kun mahdollista ja käyttää muiden koodia vain referenssinä.

10.3 Koodin ymmärtäminen

Koodia kopioitaessa muista lähteistä on erittäin tärkeää että itse ymmärtää mitä koodi tekee eikä käyttää sitä vain siksi että se näyttävästi toimii. Jos koodin toiminnallisuutta ei itse ymmärrä, ongelmien ilmestyessä voi olla hankalaa tai lähes mahdotonta selvittää johtuuko virhe omasta vai ulkopuolisesta koodista. Tämä ei tarkoita että kehittäjän pitää osata ulkoa jokaisen kirjaston toiminta, mutta että kaikki kehityksen aikana itse lisätyn koodin toiminta on ymmärrettävissä.

Aloittelevat kehittäjät saattavat nähdä helpommaksi tavaksi kopioida esimerkistä saamansa koodin ja käyttää sitä sellaisenaan. Tämä ei ole hyvä idea useasta syystä:

- alkuperäinen koodi voi olla tarkoitettu erilaiseen toimintaan vaikka lopputulos olisi ensi katsomalta identtinen
- alkuperäinen koodi voi tehdä turhia tai virheellisiä funktiokutsuja, jotka saattavat hidastaa ohjelman suoritusta
- alkuperäinen koodi voi olla tarkoitettu vain yksittäistä toimintoa varten: silmukassa ajettuna se voi olla hidas ilman optimointia
- alkuperäinen koodi voi olla riippuvainen muista kirjastoista tai resursseista, jotka ovat omaan tarkoitukseen turhia

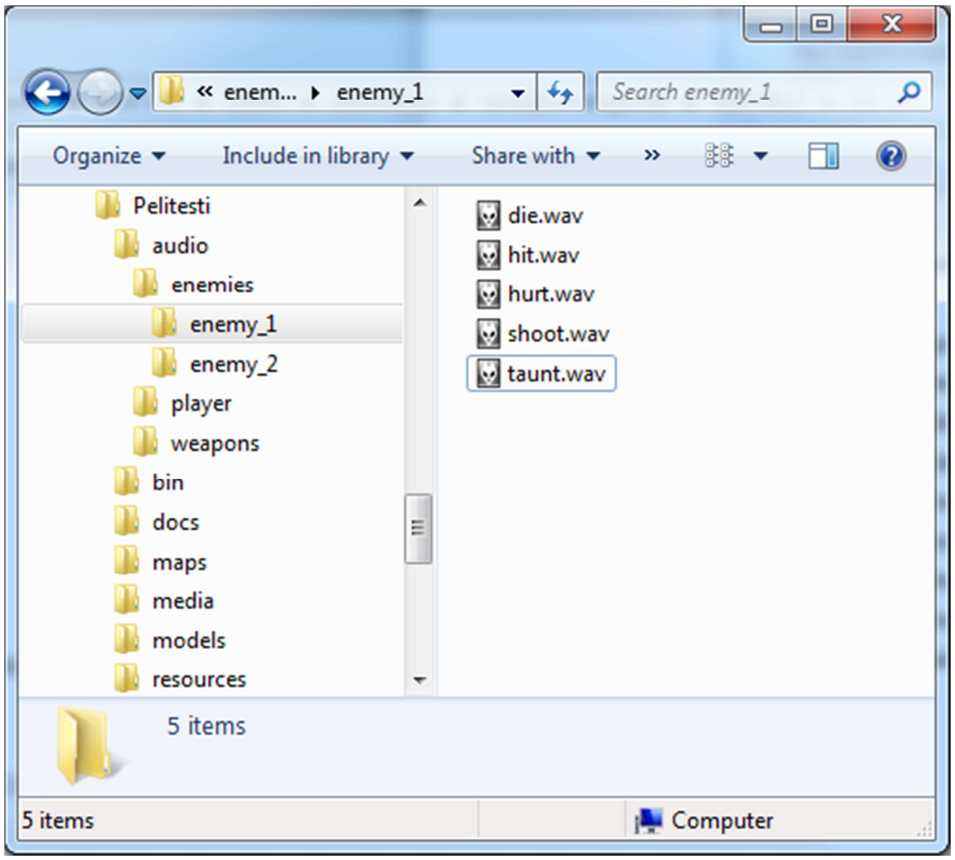
Yksinkertaisena esimerkkinä yllämainituista toimii toiminnallisuuden lisääminen peliin joka tuo mukanaan oman rajapintansa (esimerkiksi SDL:n grafiikkakirjasto). SDL vaatii toimiakseen alustustoimintoja ja vasta näiden jälkeen voidaan käyttää SDL:n muita toimintoja.

```
// SDL:n video-ominaisuuksien alustus (C++)
// Sisällytetään SDL.h-tiedosto ohjelmaan
#include <SDL.h>
// Muu alustuskoodi...
// Yritetään SDL:n alustusta
if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    fprintf(stderr, "Virhe alustaessa: %s\n", SDL_GetError());
    exit(1);
}
// Muu koodi...
```

Esimerkissä tarkistetaan palauttaako SDL_Init-funktio arvoksi nollan tai alemman luvun. SDL_Init:n tapauksessa *-1* tarkoittaa virhettä ja *0* onnistunutta alustusta. Virheen sattuessa tulostetaan virheulostuloon (*stderr*) kuvaus tapahtuneesta. Tapaus on yksinkertainen, mutta vaatii tietoa SDL:n käytöstä. Suoraan kopioitu tiettyä rajapintaa käyttävä koodi satunnaisesta koodiesimerkistä tuskin olisi toiminut, sillä ne yleensä olettavat että rajapinta on jo alustettu ja toiminnassa.

10.4 Kansiorakenteet

Projektin kansiorakenteen on syytä olla selkeä, sillä se on paikka minne tallennetaan kaikki pelin käyttämä staattiset resurssit. Yksittäisiä tiedostoja voi olla satoja tai tuhansia ja kansioita monta kymmentä projektin koosta riippuen. Kansiorakenteet ovat projektikohtaisia ja oleellisinta on säilyttää logiikka hierarkiassa.



Kuva 6: Esimerkki kansiorakenteesta

Kuvan esimerkissä on pelissä kahta erityyppistä vihollista ja ainakin toisella niistä on ääniefekti ampumista varten. Looginen paikka sijoittaa ääniresurssi on kansioon ”*/audio/enemies/enemy_2/shoot.wav*”. Vastaavasti looginen paikka pelaajan ampumisäänille olisi esimerkiksi ”*/audio/player/weapon_3/shoot.wav*”.

10.5 Virheiden käsittely

Virheiden käsittely ja niiden jäljittäminen on oleellista kun halutaan varmistaa pelin toiminta ja käyttäytyminen virhetilanteissa. Ihanteellisessa tilanteessa peli ei pääsisi kaatumaan virheeseen muista kuin ulkoisista syistä, kuten esimerkiksi käyttöalustan epävakaudesta. Kaikki muut virhetapaukset voi kehittäjä itse käsitellä koodissa. Toimintaan vaikuttavien virheiden tapahtuessa on suotavaa pyrkiä antamaan käyttäjälle ilmoitus siitä miten virheen voi välttää, kuten esimerkiksi asentamalla päivitetyt näytönohjaimen ajurit.

Peliä käynnistäessä on hyvä tarkistaa ovatko kaikki tarvittavat tiedostot ja muut sekä ilmoittaa käyttäjälle niiden mahdollisesta puutteesta tai muusta virheestä. PC-peleillä tarkastus katsoisi mm. keskusmuistin määrän, näytönohjaimen ominaisuudet ja mahdollisesti prosessorin nopeuden. Laitteistolukituilla alustoilla kuten pelikonsoleilla ja joillakin mobiililaitteilla nämä eivät välttämättä ole tarpeellisia, sillä komponentit ovat identtisiä tai jo tiedossa olevia.

Muutamia yleisiä käytäntöjä toiminnan varmistamiseen:

- varmista että kaikki resurssit löytyvät ennen niiden käyttöönottoa (esim. kaikki tarvittavat kirjastot asennettuna, tiedostoja ei puutu)
- varmista että laitteiston tehot riittävät ajamaan pelin
- jos toimintaa voidaan jatkaa virheestä huolimatta, siihen tulisi antaa mahdollisuus

Listan viimeinen merkintä on tapauskohtainen ja se tulee toteuttaa harkiten. Jos kyseessä on suhteellisen pieni virhe kuten asetustiedoston puuttuminen, ei pelin tarvitse antaa siitä virheilmoitusta vaan voidaan automaattisesti luoda oletusasetustiedosto ilman että suoritus keskeytyy.

10.6 Kehitysympäristön valinta

Kehitysympäristö (*IDE*) sisältää yleensä koodieditorin, virheenjäljitystyökalut ja koodin kääntäjän, jolla koodi saadaan käännettyä toimivaksi ohjelmaksi. Mukana tulee yleensä erilaisia aputyökaluja ja tuottavuutta lisääviä ominaisuuksia. Valmiin kehitysympäristön käyttö ei ole välttämätöntä, mutta ne ovat yleensä optimaalinen valinta kehittäjälle.

Kaikki kehitysympäristöt eivät tue kaikkia kieliä, joten kehittäjä saattaa joutua käyttämään useampaa kehityksen aikana jos projekti vaatii usean ohjelmointikielen käyttöä. Eri käyttöjärjestelmille on erilaisia kehitysympäristöjä ja ne eroavat pääasiassa lisäominaisuuksiltaan. Alla olevassa taulukossa on vertailtu kolme suosittua kehitysympäristöä ja listattu niiden ominaisuudet. Vertailuominaisuuksiin on valittu yleiset tekijät jotka vaikuttavat kehitysympäristön valintaan, mutta tarpeet voivat vaihdella projektikohtaisesti.

	MS Visual Studio (Express)	**Komodo Edit,** **Komodo IDE**	**Netbeans IDE**
Lisenssi	Maksullinen (Express-versiot ilmaisia)	Edit ilmainen, IDE maksullinen	Ilmainen
Tuetut alustat	Windows	Windows, Linux, Mac	Windows, Linux, Mac, Solaris
Tuetut kielet	C++, C#, .NET-kielet, HTML, SQL…	Perl, PHP, Python, Ruby, SQL, Lua…	Java, C/C++, PHP, Scala…
Tuki lisäosille (addon/plugin)	Kyllä	Kyllä	Kyllä
C++-kääntäjä	Kyllä	Ei (lisäosa)	Ei (lisäosa)

Microsoft Oy:n kehittämä **Visual Studio** – tuoteperhe on Windows-järjestelmille tarkoitettu kehitysympäristö. Express-versiot ovat ilmaisia myös kaupalliseen käyttöön, mutta ohjelman koko versio on niistä monipuolisin ja sisältää ominaisuuksia mitä Express-versiot eivät tarjoa, esimerkiksi resurssieditorin ja profilointityökalut. (MS Visual Studio, 2.12.2011)

Komodo Edit on ilmainen kehitysympäristö joka toimii usealla käyttöjärjestelmällä. Sillä on myös tuki laajennuksia varten joiden avulla käyttäjä voi itseä lisätä haluttuja lisäominaisuuksia. Komodo Edit on ominaisuuksiltaan rajoittuneempi versio Komodo IDE:stä, joka vaatii käyttölisenssin ja on kaupallinen. (Komodo.org, 2.12.2011)

Netbeans IDE on ilmainen kehitysympäristö joka toimii useilla käyttöjärjestelmillä. Se on pääasiassa suunnattu Java-kehitystä varten mutta modulaarisuutensa ansiosta se on laajennettavissa tukemaan muitakin ohjelmointikieliä. (Netbeans.org, 2.12.2011)

Vertailutaulukosta nähdään että käyttöjärjestelmä ja tuetut kielet rajoittavat valintaa eniten. Esimerkiksi .NET-kielien (C#, VB.NET ym.) ohjelmointi rajoittuu Visual Studioon, vaikka Linux-järjestelmillä on vaihtoehtona esim. *Monodevelop*-niminen ohjelma joka tukee C#:a. Java-kehitykseen soveltuvin vaihtoehto kolmesta on Netbeans IDE, sillä se on ainut joka tukee Javaa täysin ja se on saatavilla usealle eri käyttöjärjestelmälle. Komodo Edit ja IDE tarjoavat erilaisia ominaisuuksia versiosta riippuen ja tukevat ohjelmointikieliä joita kilpailijat eivät.

Kehitysympäristön valintaan vaikuttaa siis useampi tekijä. On syytä huomioida, että moneen kehitysympäristöön saa nykyään lisäosien avulla lisättyä tarvittuja ominaisuuksia tai tukia kielille. Kaupalliset vaihtoehdot tarjoavat hinnan vastineeksi yleensä ammattitason henkilökohtaista tuotetukea tai muita etuja. Ilmaisien vaihtoehtojen tuki saattaa rajoittua keskustelufoorumeihin ja muiden käyttäjien avuliaisuuteen. Valmis kehitysympäristö on tuottavuutta lisäävä työkalu ja sellaisen käyttö on suositeltavaa varsinkin laajoissa projekteissa joissa hallinta, testaaminen ja työn organisointi on tärkeää.

11 Äänimaailma

Äänimaailman suunnittelu on tärkeää pelin tunnelman kannalta. Musiikki ja taustaäänet vaikuttavat pelikokemukseen huomattavasti, joten niihin on syytä kiinnittää erityistä huomiota jos pelikokemus luottaa vahvasti äänien luomaan tunnelmaan.

Kuva 7: **Amnesia: The Dark Descent** luo vakuuttavan tunnelman äänien ja musiikin oikealla käytöllä.

Kauhupelit kuten *Amnesia: The Dark Descent* ja *Silent Hill*-pelisarja ovat hyviä esimerkkejä niistä, jotka hyödyntävät ääniefektejä paljon tunnelman luomiseksi. Äänien ja musiikin käyttö kauhupeleissä on verrattavissa esimerkiksi kauhuelokuviin: ilman ääniä katsottu elokuva ei todennäköisesti vaikuta katsojaan samalla tavalla kuin äänien kanssa katsottu.

Äänillä voidaan esittää pelaajalle muutakin kuin pelkkää tunnelmaa. Niillä voidaan antaa palautetta esimerkiksi pelihahmon saamasta iskusta, parannuksesta tai lähestyvästä vaarasta. Kuten muidenkin efektien kanssa, äänipalautteen tulisi tapahtua asioista mistä pelaaja odottaa sen tapahtuvan kuten kävelystä, ampumisesta tai ympäristössä tapahtuvista asioista.

11.1 Ääniefektien käyttö ja toistuvuus

Ääniefektien käyttö on tilannekohtaista ja niiden soveltuvuutta peliin tulee testata huolellisesti. Toistuvat äänet voivat olla pelikokemuksen kannalta negatiivisia jos ne toistuvat liian usein. Esimerkiksi ampumisesta, loitsuista tai valikoiden käyttämisestä johtuvat äänet on hyvä toteuttaa niin että niitä voi kuunnella toistuvasti ilman että ääni käy epämiellyttäväksi. Satunnaisesti käytetyissä äänissä saa käyttää erikoisia ääniä, mutta kaiken toistuvat ääniefektit olisi hyvä toteuttaa niin että niitä on miellyttävä kuunnella.

On hyvä huomioida, että ääniefektien tulee sopia yhteen käytössä olevan graafisen ilmeen kanssa. Tämä tarkoittaa, että käytössä olevat efektit sopeutuvat pelin luonteeseen ja kuulostavat luonnollista. Ääniefektien yhteensopivuusongelma tulee ilmi tilanteissa jotka helposti ylenkatsotaan.

Esimerkki: pelihahmo kävelee puu- tai metallilattian päällä, mutta askeleiden ääni on selvästi kuin astuttaisiin betonille. Tämä rikkoo pelimaailman luoman illuusion ja aiheuttaa hämmennystä joka vaikuttaa negatiivisesti pelikokemukseen.

Esimerkin tilanteessa asian korjaaminen on yksinkertaista, mutta sama logiikka pätee yleisellä tasolla kaikkien ääniefektien käyttöön.

12 Peliuniversumi, tarina ja juonenkuljetus

Peliuniversumilla tarkoitetaan kontekstia johon pelin tarina, juoni, ympäristö, hahmot tai tapahtumat liittyvät ja minkä sisällä ne tapahtuvat. Monet pelisarjat ja jopa eri pelit nykypäivänä käyttävät samaa peliuniversumia hyväkseen sitoakseen tarinan yhdeksi kokonaisuudeksi. Hyvänä esimerkkinä tästä on Valve Softwaren tuottamat Half-Life –sarjan pelit jotka ovat juonellisesti kytköksissä toisiinsa. Ne jakavat henkilöitä, paikkoja ja tapahtumia vaikka jokainen niistä on oma pelinsä. Saman pelitalon peleistä myös Portal-sarjan pelit ovat kytköksissä Half-Life –universumiin.

Peliuniversumin suunnittelu on luonnollisesti vapaata ja ilman rajoituksia, ellei kyseessä ole valmis universumi johon pelin on tarkoitus lisätä. On hyvä pitää mielessä tarinan kerronta, juonen kuljetus ja niiden tahdittaminen peliin sopivaksi. Tarinan suunnittelu ja sen osa-alueet ovat kuitenkin tämän opinnäytetyön ulkopuolella.

Vaikka jokaisella pelillä on teknisesti oma peliuniversuminsa, ei joillakin peleillä erillistä tarinaa välttämättä ole. Tällaisia pelejä voivat olla mm. ongelmanratkontapelit, joissa tyypilli-

sesti pääpaino on ongelmien haasteessa eikä tarinan kuljettamisella. Muutamana hyvänä esimerkkinä toimii hyvin tunnettu Tetris sekä vuonna 2008 kuulusaksi tullut indie-peli Audiosurf, jossa pelaajan oma musiikki luo kentät pelattavaksi.

Monet pelit ovat suhteellisen yksinkertaisia tekniseltä toteutukseltaan ja niiden pääpaino voi olla esimerkiksi immersiivisessä tarinassa tai juonen kuljetuksessa. Erinomaisia esimerkkejä tästä ovat mm. *Broken Sword* ja *The Longest Journey* –pelisarjojen pelit. Ne ovat perinteisiä *point-and-click*-seikkailupelejä jotka perustavat suosionsa suureksi osaksi erinomaiseen tarinalliseen toteutukseen.

Kuva 8: *The Longest Journey* on satumainen seikkailu jonka vahvuus on tarinan kerronnassa ja juonen kuljetuksessa.

Pelin tarinallinen toteutus on yksi haastavimmista osa-alueista pelikehityksessä. Pohjalla olevan tarinan monimuotoisuus vaikuttaa suoraan siihen mitä kaikkea pelin pitäisi sisältää. Tämä yleensä heijastuu siihen mitä erilaisia asioita kehityksessä tulee ottaa huomioon.

12.1 Tekniset vaikutteet

Pelin tarina voi viedä pelaajaa erilaisiin tilanteisiin kuten jäävuorelle, viidakkoon, veden alle, laavaluolaan, kaupunkiin tai avaruuteen sekä pelaaja voi joutua ajamaan eri tavalla käyttäytyviä ajoneuvoja. Mitä tämä tarkoittaa teknisen kehityksen kannalta? Sitä, että kehityksessä joudutaan ottamaan huomioon erilaiset ilmastot, maastot, materiaalit, pelissä esiintyvät rodut, ajoneuvot ja mahdollisesti myös fysiikanmallinnuksen. Lisäksi kaikelle tulisi ideaalisesti myös omat äänet, käyttäytymislogiikat ja grafiikat.

Jos pelin tarina oma omaperäinen eikä seuraa uskollisesti valmiiksi kirjoitettua tarinaa kuten kirjan juonta, on mahdollista että tarinaa joudutaan soveltamaan peliin sopivaksi mm. teknisten rajoitteiden takia. Tämä tapahtuu todennäköisimmin siksi että tarinan kirjaimellinen toteutus pelimaailmassa ei tulisi toimimaan halutusti tai sille on teknisiä esteitä.

Esimerkki: Pelissä on kohtaus joka vaatii fysiikanmallinnusta toimiakseen, mutta pelimoottori ei tue fysiikanmallinnusta. Tästä syystä voi olla suotavampaa että kohtaus päivitetään, varsinkin jos muut pelin kohtaukset eivät tarvitse fysiikanmallinnusta.

13 Käyttöliittymä

Tämä kappale käsittelee pelin käyttöliittymää ja suunnittelua. Käyttöliittymän tulisi olla yksinkertainen toteutukseltaan ja luontevaa käyttää.

Käyttöliittymä pelissä käsittää kaiken minkä kautta pelaaja antaa pelille komentoja. Tähän kuuluvat mm. valikot, ympäristön interaktiiviset kohdat ja ohjauskomennot pelihahmon liikuttamiseen. Käyttöliittymää luodessa on tärkeää pitää mielessä käyttölogiikka. Pelaajan antamiin komentoihin pelin hallinnassa sekä pelihahmon ohjattavuuteen on syytä kiinnittää erityistä huomiota, sillä pelikokemus on riippuvainen hallinnan toiminnallisuudesta.

13.1 Loogisuus

Pelaaja tulee käyttämään käyttöliittymää jatkuvasti pelin aikana riippumatta siitä onko kyseessä monimutkainen vai yksinkertainen hallinta. Looginen käytettävyys on tärkeää että pelaaja ei koe turhautumista ja ihmettele miksi asiat tapahtuvat täysin tavalla jota hän ei odottanut.

Esimerkki: Hiirellä ja näppäimistöllä ohjattava peli jossa pelihahmo ampuu hiiren vasemmasta napista ja hyppääminen tapahtuu välilyönnistä. Ajoneuvoon mennessä hiiren vasen nappi toimii kuitenkin kaasuna ja välilyönti ampumisena.

Yllä kuvatun esimerkin käyttäytyminen on epäloogista, sillä pelaaja odottaa ampumisen tapahtuvan samasta näppäimestä kuin ennenkin. Joissakin tapauksissa näppäimien toimintoja kuitenkin joudutaan vaihtamaan että pelaamisesta saadaan miellyttävämpää. Tällaisissa tapauksissa on hyvä ohjeistaa pelaajalle uudet näppäimet ruudulla.

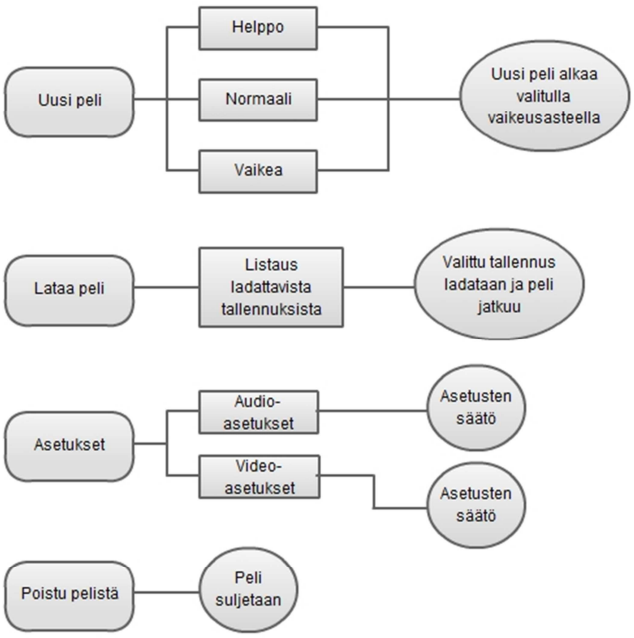
13.2 Valikot

Valikot ovat tavallinen näky lähes jokaisessa pelissä. Pelaajalle yleensä ensimmäinen vastaantuleva päävalikko esitetään pelin käynnistämisen jälkeen. Päävalikon sisältö riippuu pelistä, mutta se sisältää tyypillisesti uuden pelin aloituksen, vanhan pelin jatkamisen, pelin sulkemisen, asetukset ja mahdollisesti apulinkin tai linkin pelin tekijälistaan.

Valikoiden suunnittelussa on tärkeää ottaa huomioon mihin sitä tullaan käyttämään suurimman osan ajasta. Valikoiden suunnittelu loogisesti ja oikeita tarpeita vastaavaksi on oleellista. Listaelementtien asettelun tulisi antaa valikon käyttäjälle selkeä visuaalinen kuva valikon ra-

kenteesta. Alavalikkoja sisältävien listaelementtien tulisi erota selkeästi pääelementeistä esimerkiksi sisentämällä alavalikon elementtejä päävalikon elementin alla.

Monissa roolipeleissä pelaaja pääsee vaihtamaan hahmojen varustusta tai käyttämään tavaroita valikoiden kautta. Valikon listaus perinteisessä roolipelissä voi olla ”Aseet – Taiat – Tavarat” ja valikon avatessaan valittuna on oletuksena ”Aseet”-kohta. Jos huomataan että pelaamisen aikana vieraillaan pääasiassa tavaravalikossa, on järkevintä sijoittaa tavaravalikko niin että sen käyttö on vaivatonta, mutta esimerkissä tavaravalikkoon pääseminen vaatisi vielä kahden listauskohdan yli menemistä. Valikoiden käyttöä voidaan siis helposti parantaa vaihtamalla listakohtien järjestystä esimerkiksi muotoon ”Tavarat – Aseet – Taiat”. Näin yleisimmin vierailtuun alavalikkoon pääsee ilman ylimääräistä vaivaa.



Kuva 9: Esimerkki valikkologiikasta jossa alaosiot löytyvät johdonmukaisten valintojen alta.

13.3 Hallinta ja ohjattavuus

Ohjausskeema on pelikohtainen ja sen toteuttamiseen on useita vaihtoehtoja. Pelityypistä riippuen ohjaaminen voidaan toteuttaa useilla eri tavoilla ja ohjaimilla.

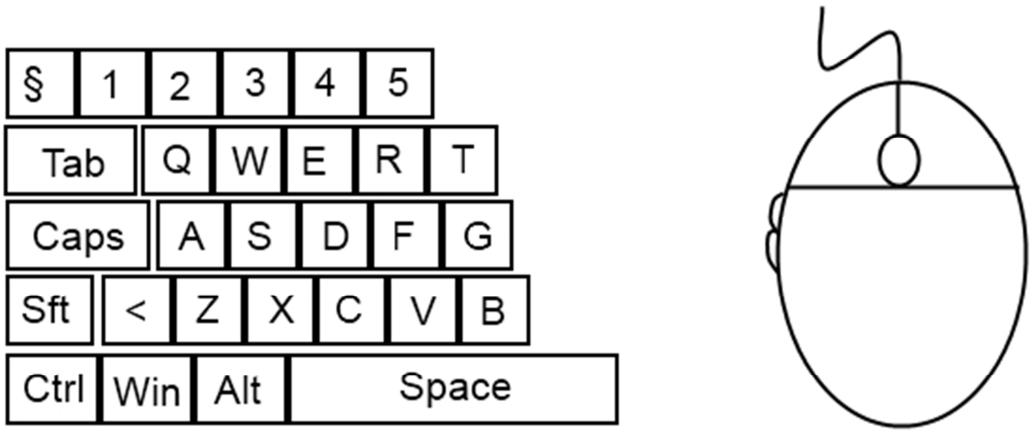
Ensimmäinen persoonan ammuskelupeleissä käytetään PC:llä pelatessa yleensä hiirtä ja näppäimistöä, kun taas pelikonsoleilla ohjaimena toimii peliohjain. Tyypillisesti PC-pelit toimivat kaikki näppäimistöllä ja/tai hiirellä sillä ne ovat valmiiksi saatavilla. Tarvittaessa voidaan

myös liittää erillinen peliohjain kuten ohjain-padi, ratti tai joystick. Lentosimulaattoripeleissä voidaan käyttää joystick-ohjainta ohjaamiseen, sillä se soveltuu paremmin lentokoneen liikkeiden hallintaan kuin hiiri tai näppäimistö. Ralli- ja ajopeleissä taas hyvänä ohjaimena toimivat luonnollisesti ratti ja polkimet.

Peliohjainta valittaessa on tärkeää pitää mielessä minkä tyyppisestä pelistä on kyse. FPS-pelit voidaan toteuttaa mm. hiirellä ja näppäimistöllä tai vaihtoehtoisesti peliohjaimella. Peliohjaimet sisältävät yleensä 1-2 ns. tattiohjain joilla liikkuminen ja tähtäys tapahtuvat. Tattien avulla tähtääminen ei kuitenkaan ole läheskään yhtä tarkkaa kuin hiirellä, mistä johtuen peliohjaimilla pelattaviin peleihin sisällytetään jonkinlainen tähtäyksen avustusjärjestelmä jonka on tarkoitus avittaa pelaajaa.

Esimerkki ohjattavuuden suunnittelusta kun käytössä on näppäimistö ja hiiri

Tavallisesti PC-peleissä interaktiota hallitaan käyttämällä hiirtä ja näppäimistössä niin sanottua WASD-klusteria. WASD-klusterilla tarkoitetaan näppäimistön aluetta jossa kyseiset kirjaimet sijaitsevat, ja nämä napit yleensä hallitsevat liikkumista ylös, vasemmalle, alas ja oikealle. Tarkempi tähtäys taas tapahtuu hiirellä. Tämä on perinteinen ohjausmenetelmä ja todettu toimivaksi monessa tapauksessa.



Kuva 10: Sormien ulottuvilla olevat näppäimet

On syytä huomioida että kuvassa 10 esitetyllä asettelulla hiirikädellä on käytössä vain 3-5 viisi nappia hiirestä riippuen: oikea, vasen, keskinappi (rulla) sekä yleensä pelihiiristä löytyvät kaksi sivunappia. Näppäimistöä käyttävällä kädellä taas on saatavilla lähes 30 näppäintä joita voidaan hyödyntää. On suositeltavaa että erityisen monimutkaisia tai vaikeasti hallittavia toimintoja ei sijoiteta hiiren nappuloihin. Näin hiiren pääasiallinen käyttötarkoitus säilyy kohdistamisessa ja monimutkaisempien toimintojen käyttö jää vasemmalle kädelle.

Moni nykyaikainenkaan peli ei tunnista hiiren sivunappuloita eivätkä kaikki pelaajat omista hiirtä josta lisänappulat löytyisivät. Tämä ongelma on helposti kierrettävissä, mutta se on asia joka on syytä ottaa huomioon jo suunnitteluvaiheessa.

13.4 Käyttöliittymän suunnittelu

Käyttöliittymän toteutuksessa tulee ottaa huomioon erilaisia asioita. Riippuen mille alustalle peli tehdään, on vastassa erilaisia haasteita. Pelikonsolit käyttävät yleensä jonkinlaista peliohjainta ohjaukseen, kun taas PC-pelit yleensä toimivat hiiren ja näppäimistön avulla. Käsikonsoleissa on erilaisia ohjaustoteutuksia riippuen käsikonsolista. Lisäksi on olemassa vielä erilaiset mobiililaitteet kuten älypuhelimet.

Mikä tahansa alusta onkaan kyseessä, tulee käyttöliittymä suunnitella alustan ehdoilla. Loogi-nen käyttöliittymä helpottaa käyttöä ja tekee pelikokemuksesta miellyttävän. Ohjattavuus pelin sisällä tulisi olla mutkatonta ja luonnollista. Ihanteellisessa tapauksessa pelin voisi antaa pelattavaksi kenelle tahansa joka ei sitä ennen ole pelannut ja tämä henkilö ei kokisi vaikeuksia pelin hallinnassa.

14 Testaus

Testaus on prosessi, joka takaa viimeistellyn pelin laadun ja toimivuuden. Se on myös osuus jonka itsenäinen kehittäjä saattaa jättää toteuttamatta ajan tai motivaation tai resurssien puutteessa.

Testaus on oleellinen osa koko kehitysprosessia ja sitä tapahtuu koko kehitysprosessin ajan: kun lisätään tai poistetaan ominaisuuksia, muutetaan resursseja (ääniefektejä ym.) tai lisätään uusia elementtejä. Sen tarkoitus on toimia laatutarkistuksena ja löytää etukäteen virheet tuotoksesta. Esimerkki: pelaaja tekee jotakin mitä kehittäjä ei osannut odottaa, kuten pääsee pelissä paikkaan mihin ei olisi tarkoitusta päästä. Vastaava tilanne voi päätyä pelin kaatumiseen tai johtaa tilanteeseen, josta pelaaja voi edetä mitenkään.

Testausprosessi

Testausprosessi ei koettele ainoastaan pelin teknistä toteutusta vaan kaikkea yhtenäisenä kokonaisuutena pelaajalle. Pelaamalla ja erilaisia asioita pelin sisällä kokeilemalla saadaan selville paljon muutakin. Muutamia esimerkkejä siitä mitä testaus voi osoittaa käyttäjäkokemuksesta:

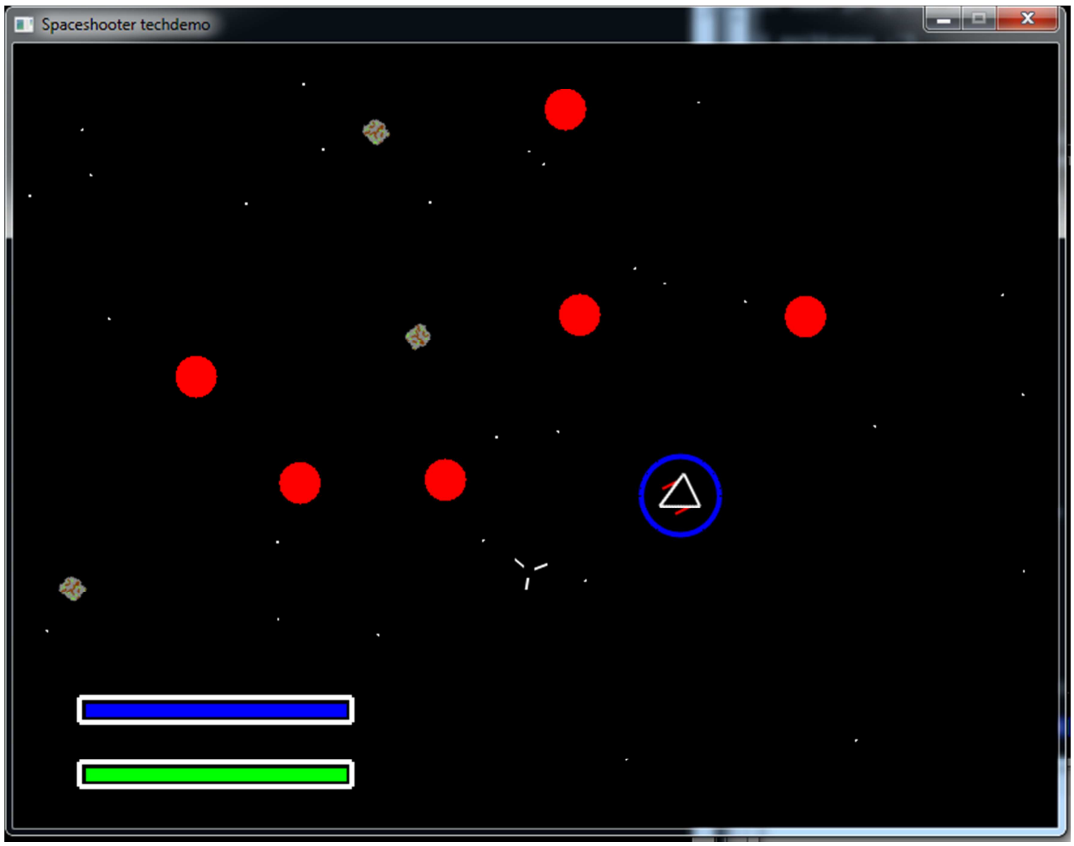
- Juoni – kuljetetaanko tarinaa hyvällä tahdilla vai onko tarvetta muuttaa tahtia, että pelaaja tuntisi edistyvänsä?
- Äänet ja musiikki - sopivatko musiikit ja äänet tunnelmaan ja luovatko ne halutun ilmapiirin? Toimivatko ne halutulla tavalla?
- Tasapaino – toimiiko pelin sisäinen interaktio (esim. tappelu) halutusti? Onko se sulavaa ja mielekästä?
- Pelihahmon kyvyt ja rajoitukset – annetaanko pelaajalle liikaa vapautta tehdä asioita joita ei pitäisi? Pitäisikö vapautta antaa lisää?

Vaikka testauksessa suuri vastuu on itse kehittäjällä, on hyvä peli antaa pelattavaksi testiryhmälle jos siihen on mahdollisuus. Mitä enemmän pelillä on testaajia, sitä laadukkaampi pelistä lopulta tulee. Testaajat ovat myös erinomainen lähde palautteelle: he kertovat mitä he kokivat, mitä pitäisi muuttaa tai päivittää sekä heidän testaajien avulla voidaan saada kiinni ohjelmointivirheitä jota omassa testauksessa olisivat menneet ohi. Oleellisinta testauksessa on toisto ja kaiken mahdollisen yrittäminen. Kehityksen edetessä peli altistuu yhä enemmän erilaisille ohjelmointivirheille ja muille haitoille joiden huomaaminen jälkikäteen voi olla vaikeaa testauksen aikanakin, sillä virhe voi tapahtua esim. vasta tiettyjen toistokertojen jälkeen tai sellaisessa tapauksessa jota pelaaja ei normaalisti tekisi ja johon ei kehityksen aikana osattu valmistua.

Pelissä saatavien tavaroiden kanssa tärkeää on tärkeää huomioida niiden tasapaino muun pelin kanssa. Esimerkiksi juuri lisätty ase voi olla pelin sisällä aivan liian ylivoimainen esine pelaajalle vaikka suunnitteluvaiheessa sen implementointi on vaikuttanut hyvältä idealta. Tämä ei ole erikoista, sillä kehityksen aikana muuttuu paljon toteutettavia asioita vasta sen jälkeen kun ne on nähty toiminnassa pelin sisällä.

15 Esimerkkipeli

Opinnäytetyön liitetiedostona on avaruusräiskintäpeli vähemmän omaperäisesti nimeltään ”Spaceshooter” joka on ohjelmoitu käyttämällä Lua:a ja Love2D-pelimoottoria. Ohjelmointiympäristössä kehitykseen on käytetty Komodo Editiä joka on ilmainen avoimen lähdekoodin koodieditori ja tukee Lua:n lisäksi monia muita kieliä. Versionhallintaan käytettiin Subversion-ohjelmistoa. Se on toteutettu Open Source – ohjelmistoja apuna käyttäen (versionhallintaa lukuun ottamatta) ja kaikki sen käyttämät resurssit (kuten grafiikka) on itse toteutettua.



Kuva 11: Kuva opinnäytetyön yhteydessä toteutetusta ammuskelupelistä, jonka omaperäinen nimi on *Spaceshooter*.

Pelin sisällä pelaajalla itsellään on käytössä avaruusalus, jonka vihollisalukset yrittävät tuhota. Pelaajalla on käytössään ase, jota käyttämällä hän voi itse tuhota lähestyvät viholliset. Vaarana avaruudessa vihollisten lisäksi on myös asteroideja joita pelaajan tulee vältellä selviytyäkseen. Pelaajalla on myös suojakilpi, joka voi ottaa osumia vastaan, mutta sen latautuminen kestää aikansa. Liitepeli on ohjelmointiharjoituksena toteutettu teknologiademo, jonka tarkoitus ei ole olla täysiverinen peli, vaan se on prototyyppi yksinkertaisesta avaruudessa tapahtuvasta ammuntapelistä.

Pelin runkorakenteena (*framework*) on käytetty LÖVE 2D-ohjelmaa (tyypillisesti kirjoitettu *Love*), joka on tarkoitettu erityisesti peliohjelmointiin ja nopeaan prototyypitykseen. Love sisältää myös fysiikanmallinnuksen jonka avulla kehittäjä voi saada aikaiseksi erittäin näyttäviä ja monipuolisia pelejä. Ohjelmointikielenä on käytetty *Lua*:a, joka on erittäin skaalauntuva ja nopea kieli jota käytetään myös Loven kehitykseen. Lua on suosittu videopelien kehityksessä nopeutensa ja luotettavuutensa ansiosta, mutta sitä käytetään yleensä ns. skriptaukseen eikä itse pelin ohjelmoimiseen, mutta soveltuu nopeutensa takia myös ohjelmointiin.

16 Yhteenveto

Johdannossa totesin valinneeni aiheekseni itsenäisen pelikehityksen, sillä se on mielenkiintoinen ja monipuolinen prosessi. Tutkimuksen aikana todettiin, että tämä väite pitää edelleen paikkansa, mutta monella eri tapaa kuin alustavasti oli oletettu. Itsenäinen pelikehitys vaatii kehittäjältä paljon: tietoa, taitoa, suunnittelukykyä, ohjelmointikokemusta ja tärkeimpänä visiota siitä, mitä haluaa tehdä. Osa kehitykseen liittyvistä vaiheista on haastavaa toteuttaa itsenäisesti. Tekstuurien kehitys, musiikin ja ääniefektien valinta, niiden mahdollinen tuotto ja sovitus ovat työläitä prosesseja. Vaikka vaihtoehtoina olisi vapaasti käytettäviä resursseja, ne eivät aina ole omaan tuotokseen sopivia.

Alkuperäisiä tutkimuskysymyksiä oli useita, mutta johdannossa tiivistin ne neljään oleelliseen ydinkysymykseen. Tutkimuksen aikana nämä kysymykset saivat myös vastaukset.

Itsenäinen pelikehitys käsitteenä sisältää useita eri asioita, kuten suunnittelua, vertailua sekä paljon tuotannollista työtä, sillä kaikki on tehtävä itse. Konsepti on itse kehitettävä ja jalostettava toteutuskelpoiseksi ideaksi. Suunnittelu on tehtävä itse. Grafiikka, äänet, ohjelmointi ja tarinan suunnittelu on toteutettava käsin, vaikka alustavia ratkaisuja olisikin saatavilla.

Pelikehitys vaatii kehittäjältä myös huomattavan paljon aikaa ja omistautumista projektille. Itsenäiselle kehittäjälle tämä voi tarkoittaa pelin kehittämistä iltaisin ja viikonloppuisin jos kehittäjä käy päivätöissä. Rahallinen tuki ei ole välttämätöntä, mutta se on avuksi jos kehitykseen tarvitaan esimerkiksi lisenssejä tai muita maksullisia resursseja.

Kehittäjän on myös omattava tietoa eri osa-alueista jotka tekevät pelin. Projektissa voi työskennellä useampi henkilö, joissa jokaisella on yleensä oma erikoisosaamisensa, kuten ohjelmointi tai grafiikan luominen. Jokaisella jäsenellä on hyvä olla tietämys muiden alueiden rajoituksista ja mahdollisuuksista vaikka ei itse samaa työtä tekisikään. Jokaisella on oltava kokonaiskuva projektista, joka edesauttaa yhteistä työskentelyä. Sama pätee myös vaikka projektissa työskentelisi vain yksi henkilö, ja tämän henkilön on tiedostettava kaikkien osa-alueiden ominaisuudet.

Mahdollisesti vaikein tutkimuskysymyksistä oli, että ”mikä tekee pelistä menestyvän?”. Hyvin menestyneitä pelejä yhdistää harva tekijä. Pelillä on voinut olla miljoonabudjettia ja satoja kehittäjiä tai vain muutama henkilö jotka eivät ole saaneet edes rahoitusta. Peli on voinut olla graafisesti realistinen 3D-simulaattori tai kaksiulotteinen tasohyppely räikeillä väreillä. Peli on voinut olla vahvasti tarinapainotteinen seikkailu tai juoneton ongelmanratkontapeli. Peli on voinut olla nopeatempoinen toimintapeli tai hidastempoinen klikkailuseikkailu. Pelin

toteutus on voinut olla jotakin uutta ja ennen näkemätöntä tai tuttua kaavaa noudattanut. Vertailun ääripäitä ja niiden välitiloja on rajattomasti.

Kaikki edellä mainittuja ominaisuuksia löytyy sekä hyvistä että huonoista peleistä. Hyvä peli saadaan aikaan yhdistämällä näitä ominaisuuksia, poistamalla toisia, keksimällä uusia ja muokkaamalla vanhoja. Oikein toteutettuna lopputuloksena on mielenkiintoinen, uutta tarjoava peli josta pelaajat innostuvat. Väärin toteutettuna tuloksena on vanhaa toistava peli joka ei tarjoa pelaajalle mielenkiintoista pelattavaa. Pelin luonti on aina tapauskohtaista ja sen menestykseen vaikuttavat luonnollisesti trendit sekä pelimarkkinoiden yleinen tilanne. Tärkeää on löytää pelaajien mieltymyksistä ns. markkinarako, jota hyväksikäyttämällä voidaan luoda menestyvä peli.

Pelin kehittäminen alusta loppuun on työläs prosessi, jonka aikana kaikki suunnitelmat voivat muuttua. Seurauksena projekti voi joutua kokemaan suuriakin muutoksia, joten alustava suunnittelu ja asioiden selvitys on tärkeässä asemassa. Myyntiin tulevassa tuotoksessa on myös otettava huomioon asiakkaat ja heidän tarpeensa, tuotetuki sekä mahdolliset lupaukset jatkokehityksestä. Se on työ, joka vaatii tekijältään päättäväisyyttä ja intohimoa pelien tekemiseen. Se on työ, jossa jokaisen kehittäjän on tunnettava perusteet jokaisesta kehitykseen kuuluvasta osa-alueesta.

Lähteet

ActiveState Komodo IDE. Viitattu 2.12.2011
http://www.activestate.com/komodo-ide

CodeCogs – Programming guidelines. Viitattu 6.11.2011.
http://www.codecogs.com/pages/standards/programming.htm

DirectX Developer Center. Viitattu 6.11.2011.
http://msdn.microsoft.com/en-gb/directx

Duke Nukem Forever update increases weapon slots on PC. Viitattu 6.11.2011
http://www.shacknews.com/article/69486/duke-nukem-forever-update-increases-weapon-slots-on-pc

Game programming languages. Viitattu 6.11.2011.
http://en.wikipedia.org/wiki/Game_programming#Programming_languages

List of indie game developers. Wikipedia. Viitattu 6.11.2011
http://en.wikipedia.org/wiki/List_of_indie_game_developers

List of indie game developers. Wikipedia. Viitattu 6.11.2011
http://tig.wikia.com/wiki/List_of_Indie_Game_Developers

LÖVE – Free 2D Game Engine
http://love2d.org/

Metacritic Reviews. Viitattu 6.11.2011
http://www.metacritic.com/

Metacritic (top games of all time). Metacritic.com. Viitattu 6.11.2011
http://www.metacritic.com/browse/games/score/metascore/all/all?view=condensed&sort=desc

Microsoft Visual Studio. Microsoft. Viitattu 2.12.2011
http://www.microsoft.com/visualstudio/

Mysore, Sahana. How the World of Goo became one of the indie video game hits of 2008. Venturebeat. Viitattu 6.11.2011
http://venturebeat.com/2009/01/02/the-world-of-goo-became-one-of-the-indie-hits-of-2008/

Netbeans IDE - Features. Netbeans. Viitattu 2.12.2011
http://netbeans.org/features/index.html

PSGL – Playstation Graphics Library. Wikipedia. Viitattu 6.11.2011.
http://en.wikipedia.org/wiki/PSGL

Roberto Ierusalimschy 2003. Programming in Lua, 2. painos. PUC-Rio: Lua.org

Top 10 most expensive video games budgets ever. Digitalbattle. Viitattu 6.11.2011
http://www.digitalbattle.com/2010/02/20/top-10-most-expensive-video-games-budgets-ever/

Ultimate Indie Game Developer Source List. Viitattu 6.11.2011.
http://www.mangatutorials.com/forum/showthread.php?742-The-Ultimate-Indie-Game-Developer-Resource-List

Liitteet

Liite 1

CD-levy, joka sisältää:

- Love 2D –asennusohjelman Windows-järjestelmille
- Spaceshooter-pelin vaatimat tiedostot ja resurssit
- Lähdekoodin Spaceshooter-pelille
- Englanninkieliset ohjeet asennukseen
- Ohjeita muille käyttöjärjestelmille asentamisesta