Tero Paavolainen

Käyttäjälähtöisen muokattavuuden parantaminen Unity-pelimoottorilla tehdyissä peleissä

Tietotekniikan pro gradu -tutkielma

16. kesäkuuta 2017

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Tero Paavolainen

Yhteystiedot: `tero.s.t.paavolainen@student.jyu.fi`

Ohjaaja: Ville Isomöttönen

Työn nimi: Käyttäjälähtöisen muokattavuuden parantaminen Unity-pelimoottorilla tehdyissä peleissä

Title in English: Improving the user driven modifiability of Unity made games

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Pelit ja pelillisyys

Sivumäärä: 87+13

Tiivistelmä: Tässä tutkimuksessa toteutettiin käyttäjälähtöistä muokattavuutta helpottava kirjasto Unity-pelimoottorille suunnittelutieteellisenä artefaktina. Artefaktia kehitettiin kolmen syklin ajan, joissa jokaisessa oli oma arviointiosuutensa kirjaston mielekkyyden tarkkailuun. Arviointiin käytettiin kahta esimerkkipeliä ja tapaustutkimusta kattavan arvioinnin takaamiseksi, joiden havaintojen avulla kirjastoa kehitettiin edelleen. Kirjasto on toimiva kokonaisuus ja siitä on apua käyttäjälähtöisen muokattavuuden sallivien pelien kehityksessä.

Kirjaston laadulliset vaatimukset *omaksuttavuus* ja *käytettävyys* todennettiin tapaustutkimuksella. *Suorituskykyä* testattiin tekemällä yksinkertainen testi ensimmäisellä esimerkkipelillä ja *turvallisuus* ja *yleistettävyys* otettiin huomioon kirjaston rakenteessa ja arkkitehtuurissa. Toisella esimerkkipelillä varmistettiin kirjaston uusien ominaisuuksien ja kokonaisuuden toimivuus.

Avainsanat: käyttäjälähtöinen muokattavuus, loppukäyttäjä, sovelluskehitys, Unity, modaus, pelit, pelinkehitys

Abstract: In this study a library was developed for allowing user driven modifiability on Unity game engine. The library was developed as a design science artifact. The artifact was evaluated during three cycles which each contained their own eva-

luation to validate the usefulness of the library. The evaluation was done using two example games and a case study to provide inclusive evaluation. The observations from these evaluations were used to develope the library further. The library is a working and validated software and it can be used for allowing user driven modifiability in game development.

The requirements of *adoptability* and *useability* were validated with case study. *Performance* of the library was tested with a simple test run on the first example game and *security* and *generalizeability* were taken into account when defining the structure and architecture of the library. The second example game was used to verify new features and the library as a whole.

Keywords: user driven modifiability, end-user, software development, Unity, modding, games, Game development

Kuviot

Taulukot

Sisältö

1 Johdanto

Pelit ovat nousseet kiinnostavaksi keskustelunaiheeksi tutkimuksessa ja muussa keskustelussa. Kuitenkin peleihin ja modaukseen liittyvää tutkimusta saatetaan pitää vielä vähempiarvoisena (Champion 2012; Nardi 2010, s.6). Esimerkiksi Champion (2012) kuvailee, kuinka hänen pelien modaukseen liittyvää artikkeliaan väheksyttiin ja artikkelin vertaisarvioija aikoi sivuuttaa koko artikkelin. Pelit ovat ohittaneet elokuvateollisuuden tuottavuudessa (Nardi 2010, s.8) ja pelejä tutkitaan positiivisista lähtökohdista (ks. pelimodien käytöstä opetuksessa Lowe 2009; El-Nasr ja Smith 2006; Yucel, Zupko ja El-Nasr 2006). Scacchin (2011b) mukaan pelit ovat toiseksi suosituin projektikategoria OpenSource-sovellusjakelualustoissa. Vuonna 2011 SourceForge-sovellusjakelualustassa *Pelit*-kategoria sisälsi 42 tuhatta aktiivista projektia, joissa kehitettiin joko pelejä, pelimoottoreita tai pelinkehitysympäristöjä. Peliharrastajien into muokata pelejä on ollutkin nousussa uusissa peleissä (Scacchi 2011b).

Monissa peligenreissä, esimerkiksi reaaliaikaisissa strategiapeleissä, roolipeleissä ja ensimmäisen persoonan ammuntapeleissä (*First person shooter* eli *FPS*), pelit suunnitellaan niin, että käyttäjälähtöinen muokattavuus eli *modaus* on helppoa toteuttaa (Postigo 2008). Christiansenin (2012) mukaan suurin osa suurista pelimoottoreista tukee nykyään modausta. Käyttäjälähtöisen muokattavuuden salliminen pelissä voi pidentää alkuperäisen pelin elinikää ja antaa inspiraatiota pelinkehittäjille lisäominaisuuksia tai jatko-osia varten (Champion 2012). Joitain kulttisuosion saavuttaineita pelejä, jotka eivät ole tarjonneet erityisiä työkaluja, on myös muokattu peliyhteisöjen toimesta binäärimuodossa tallennettuun koodiin tai dataan tietoja muokkaamalla.[1] Joitain pelejä on myös pidetty ajantasalla, jotta niitä voisi paremmin pelata nykyaikaisilla tietokoneilla.[2] Pelien modaus on myös onnistunut sovellusten laajennettavuuden muoto (Scacchi 2011a). Käyttäjälähtöisen muokattavuuden tutkimus

1. Esimerkiksi `http://www.blizzhackers.cc/viewtopic.php?t=459161` sivulla käsitellään joidenkin muokkausten tekemistä *Diablo2*-peliin.

2. Tästä esimerkkinä Ur-quan masters -peli, joka on kehitetty Star Control 2 pelistä. Tarkempaa tietoa osoitteesta `http://sc2.sourceforge.net/`

kasvattaa siis myös yleistä sovelluskehityksen tasoa.

Unity3D tai lyhyemmin Unity on nopeasti omaksuttava, monikäyttöinen pelimoottori, jolla voidaan julkaista useille eri alustoille pelejä tai sovelluksia. Sillä voi kehittää pelejä matalin kustannuksin pienissä projekteissa, usein ilmaiseksi (Xie 2012). Sillä on toteutettu useita kaupallisesti menestyneitä pelejä, kuten *Kerbal Space Program*, *HearthStone* (Dotan 2017) ja *Cities Skylines*.[3]

Tässä tutkimuksessa selvitin, *voidaanko Unitylle tehdä käyttäjälähtöisen muokattavuuden salliva kirjasto* ja *kuinka omaksuttava ja käytettävä se on*. Jälkimmäiseen kysymykseen keskityin vahvasti kehittämisnäkökulmasta: kuinka kirjaston omaksuttavuutta ja käytettävyyttä voidaan kehittää paremmaksi. Tein tutkimuksen suunnittelutieteellisenä tutkimuksena kuten Hevner ym. (2004) sen määrittelevät: tutkimuksessa toteutettiin suunnittelutieteellinen artefakti, jota kehitettiin ja arvioitiin iteratiivisesti. Artefaktina toimi käyttäjälähtöisen muokattavuuden salliva kirjasto Unity-pelimoottorille. Kirjaston tärkeimmät laadulliset vaatimukset olivat sen omaksuttavuus ja käytettävyys. Nämä kaksi laatuvaatimusta olivat laatuvaatimukset, joilla määriteltiin kirjaston *hyvyys*. Käytettävyydellä tarkoitan tässä työssä kirjaston käytettävyyttä ohjelmoijan näkökulmasta, en kirjaston käyttöliittymän käytettävyyttä. Kirjastolle asetettiin muitakin laadullisia vaatimuksia: yleistettävyys, turvallisuus ja suorituskyky.

Kehitin kirjastoa kolmessa syklissä, joissa sitä arvioitiin erilaisten arviointitapojen avulla. Ensimmäisessä syklissä arvioin kirjaston rakennetta ja suorituskykyä esimerkkipelillä, joka toimi skenariona ja prototyyppinä. Toisen syklin arvioinnissa keskityin kirjaston omaksuttavuuteen ja käyttöönotettavuuteen. Arviointimenetelmänä käytin tällöin tapaustutkimusta, jossa tutkin kirjaston toimivuutta Peliteknologia-kurssilla. Kolmannessa syklissä tein toisen esimerkkipelin, jossa testasin erityisesti edellisen syklin perusteella lisättyjen ominaisuuksien toimivuutta. Tämä esimerkkipeli toimi prototyyppinä. Nämä eri arviointityylit täydensivät toisiaan ja sallivat usean arviointityylin toteuttamisen tässä tutkimuksessa, mikä on Hevnerin ja muiden (2004) mukaan tärkeää suunnittelutieteessä.

3. ks. `http://www.citiesskylines.com/`

Halusin valita Unity-pelimoottorin tutkimuskohteeksi useasta syystä. Unity on minulle henkilökohtaisesti mielenkiintoinen pelimoottori, sillä olen käyttänyt sitä useissa projekteissa. Oman Unity-kokemukseni mukaan Unityllä on vaikeaa tehdä käyttäjälle muokattavissa olevia pelejä käyttäen vain Unityn tarjoamaa valmista arkkitehtuuria, joten käyttäjälähtöisen muokattavuuden sallivalle kirjastolle on olemassa tarve. Mahdollisuus ohjelmoida pelejä käyttäen C#-ohjelmointikieltä ja Unityn tuki monille eri alustoille julkaisuun tekevät siitä myös mielekkään tutkimuskohteen. Olisi mielenkiintoista tarkastella useita eri pelimoottoreita alustana käyttäjälähtöiselle muokattavuudelle, mutta tämän tutkimuksen puitteissa keskityin vain Unityyn. Hyvät muokkaustyökalut helpottavat myös kehittäjien työskentelyä, sillä sovellusten laajennettavuutta helpottavia teknologioita voidaan pitää hyvinä käytänteinä myös yleisessä sovelluskehityksessä.

Suunnittelutieteen käytännönläheinen lähestymistapa ongelmaan kiinnosti minua henkilökohtaisesti, sillä ajattelin artefaktin kehittämisen kehittävän minua ohjelmoijana ja itse kirjaston antavan konkreettista hyötyä jatkossa: kirjastoa voidaan käyttää sen käyttötarkoituksessa tulevissa projekteissa. Suunnittelutiede on hyvin teknologialäheistä tutkimusta ja sen tuotteita arvioidaan niiden hyödyllisyyden mukaan: toimiiko se (March ja Smith 1995). Suunnittelutieteen avulla pystytään myös tarkastelemaan, voidaanko tietty teknologia toteuttaa tai ottaa käyttöön.

Luvussa 2 käsittelen käyttäjälähtöistä muokattavuutta ja esittelen, kuinka se ilmentää sovellusten laajennettavuutta, mitä käyttäjälähtöinen muokattavuus yleisesti on sekä miten ja miksi se tulisi sallia peleissä. Luvussa 3 esittelen Unity3D-pelimoottoria ja tarkastelen, miten käyttäjälähtöinen muokattavuus on sallittu Unityssä ja vielä miten muissa peleissä on toteutettu käyttäjälähtöinen muokattavuus. Luvussa 4 käsittelen suunnittelutieteellistä tutkimusta, miten artefakteja voidaan arvioida siinä ja miten suunnittelutiede ilmenee ja miten se toteutetaan tässä tutkimuksessa. Luvussa 5 esittelen, kuinka tutkimus ja sen eri vaiheet onnistuivat. Luvussa 6 pohdin ja esitän johtopäätöksiä kirjastoon liittyen ja esittelen mahdollisia jatkotutkimusmahdollisuuksia.

2 Käyttäjälähtöinen muokattavuus

Käyttäjälähtöisellä muokattavuudella tarkoitetaan tässä tutkimuksessa käyttäjien tekemiä muutoksia johonkin valmiiseen peliin tai kokonaisuuteen, mitkä ovat usein tehty eri työkaluilla kuin itse alkuperäinen peli. Englanninkielisessä kirjallisuudessa käytetään termejä *mods* ja *modding*, kun käsitellään vastaavia asiakokonaisuuksia, mutta esimerkiksi Postigo (2007) käyttää nimitystä *add-on*. Sana *mod* on lyhenne sanasta *modification* (Poretski ja Arazy 2017). Modification-sanan voi suomentaa monin tavoin esimerkiksi muunnokseksi, muutokseksi tai muunnelmaksi, mutta mikään näistä ei mielestäni tuo esille tarpeeksi, että kyse on nimenomaan sovellukseen tehdystä muokkauksesta. Kuitenkaan sana *mod* tai suomenkielinen vastine *modi* ei näytä yleistyneen kirjakieleen, vaan on pikemminkin lainasana. Tästä huolimatta käytän yksittäisestä muokkauksesta sanaa modi. Kirjallisuudessa kutsutaan modeja tekeviä pelaajia modaajiksi (Scacchi 2010, esim.) tai esimerkiksi fani-ohjelmoijiksi (Postigo 2007).

On huomionarvioista, että käyttäjälähtöiseen muokattavuuteen liittyvässä tutkimuksessa on muutamia asiaan vahvasti perehtyneitä henkilöitä, joten suuri määrä kirjallisuudesta on samojen henkilöiden kirjoittamaa. Esimerkiksi Scacchi, Postigo ja Nardi ovat tutkineet kattavasti käyttäjälähtöistä muokattavuutta eri näkökulmista. Scacchi on tutkinut käyttäjälähtöistä muokattavuutta myös sovellusteknisestä näkökulmasta, joten käytän paljon hänen tekemää tutkimusta lähteenä. Poretskin ja Arazyn (2017) tutkimus on siitä harvinainen, että se on ensimmäisiä tutkimuksia, jossa pyritään määrällisellä tutkimuksella osoittamaan käyttäjälähtöisen muokattavuuden *arvo*.

Tässä luvussa esittelen aluksi, kuinka sovelluksia laajennetaan sovelluskehityksessä ja kuinka tämä heijastuu peleihin. Sitten kerron käyttäjälähtöisen muokattavuuden historiaa, esittelen modien erilaisia ilmentymiä ja kerron modien levityksestä. Modien levitykseen liittyen pohdin myös levityksestä aiheutuvia tietoturvaongelmia. Tietoturvaongelmien jälkeen käsittelen modeihin liittyviä epäselvyyksiä tekijänoikeusasioissa. Lopuksi käsittelen modeista syntyvää kaupallista hyötyä sekä perus-

telen syitä sallia modaus peleissä ja vielä pelaajien syitä modien tekoon.

2.1 Sovellusten laajennettavuus ja loppukäyttäjän sovelluskehitys

Loppukäyttäjän sovelluskehitys ja käyttäjälähtöinen muokattavuus ovat ilmentymiä samasta ilmiöstä (Scacchi 2011a; Ko ym. 2011): niissä tuotetta muokataan paremmin käyttäjälle sopivaksi tai mieluisaksi. Tarkemmin Scacchi (2011a) näkee modauksien olevan sovelluslaajennuksien muoto, kun Ko ym. (2011) esittelevät videopelien muokkauksen loppukäyttäjän sovelluskehityksen muotona. Scacchi (2011a) esittää pelien modauksen osoittavan sovelluslaajennosten käytännön arvon käyttäjäystävällisenä lähestymistapana ohjelmien kustomoinnissa. Hän jatkaa tälläisten sovelluksien pystyvän laajentamaan käyttäjälähtöisesti muokattavat pelit integroiduiksi sovelluskehityspaketeiksi ja monipuolisiksi tuotantolinjoiksi, jotka kukoistavat käyttäessään sovellusalueen ohjelmointikieliä. Tässä osiossa tarkastelen tarkemmin erilaisia tapoja parantaa sovelluksien laajennettavuutta.

Sovellusten laajennettavuus ja esimerkiksi loppukäyttäjän sovelluskehitys ovat samankaltaisia kokonaisuuksia, mutta eri näkökulmista. Kirjallisuudessa ensimmäisessä keskitytään enemmän sovelluksen kehittäjän näkökulmasta kykyyn muokata ja laajentaa sovellusta jälkikäteen, kun jälkimmäisessä mietitään, miten loppukäyttäjät sovellusta käsittelevät. Mielestäni sovelluksien laajennettavuus tiivistyy hyvin Batoryn, Johnsonin, MacDonaldin ja von Heederin (2002) ajatuksiin: "Extensibility is the property that simple changes to the design of a software artifact require a proportionally simple effort to modify its source code". Haluaisin vielä jatkaa tätä ajatusta lisäämällä, että aina ei tarvitse muokata ollenkaan lähdekoodia, mikäli käytetään esimerkiksi skriptikieliä apuna.

Sovelluksien laajennettavuutta on tutkittu jo suhteellisen pitkään: artikkeleita on jo ainakin vuodesta 1979. Batoryn ja muiden (2002) mukaan sovelluksien, joita on helppo kehittää paremmiksi, tekeminen on keskeinen ongelma nykypäivän ohjelmistotuotannossa. Ko ym. (2011) esittävät loppukäyttäjän sovelluskehityksen ole-

van kasvava ala, sillä sovelluksia kehittäviä ihmisiä on murto-osa verrattuna sovelluksia käyttäviin ihmisiin. He määrittelevät loppukäyttäjän sovelluskehityksen olevan ohjelmointia tavoitteena saada ohjelma henkilökohtaiseen käyttöön julkisen käytön sijaan. Tosin, varsinkin peleissä, käyttäjälähtöinen muokattavuus saattaa tähdätä kokonaisen yhteisön tavoitteiden täyttymiseen.

Batory ym. (2002) käsittelevät kolmea teknologiaa, joiden he uskovat helpottavan sovellusten laajennettavuutta: olio-ohjelmoinnilliset suunnittelumallit, sovellusalueen kielet ja tuotelinjastoarkkitehtuuri. Gamma ym. (1993) käsittelevät olio-ohjelmoinnillisia suunnittelumalleja. Heidän mukaan nämä mallit tarjoavat yhteisen sanaston, joilla kehittäjät voivat keskustella, dokumentoida ja tutkia erilaisia suunnitelmia. He jatkavat hyvän mallin nostavan tasoa, jolla henkilö ohjelmoi, ja toimivan rakennuspalikoina hankalampien kokonaisuuksien rakentamisessa. Malli myös tarjoaa aloittelijoille helpomman tavan oppia erilaisien kirjastojen käyttöä. Sovellusalueen kielien on tarkoitus tuoda lisää ilmaisuvoimaa yleiskäyttöisten ohjelmointikielten lisäksi mini-kielillä (Tratt 2008; Ko ym. 2011). Esimerkiksi Batory ym. (2002) tekivät Javasta oman versionsa, jota he käyttivät sovellusalueen kielenä tilakoneiden esittämiselle. Tuotelinjastoarkkitehtuurissa kehitetään tuoteperheille uudelleenkäytettäviä komponentteja (Batory ym. 2002). Scacchi (2011a) keskustelee Batoryn ja muiden (2002) artikkelista ja toteaa artikkelin kirjoittajien esittelemien kolmen tekniikan olevan usein käytössä peleissä, jotka sallivat käyttäjälähtöisen muokattavuuden.

Pelejä muokataan työkaluilla, joilla päästään käsiksi salaamattomaan esitykseen pelisovelluksesta tai alustasta (Scacchi 2011a). Scacchin (2011a) mukaan kyseistä esitystä yleensä käsitellään ja laajennetaan sovellusalueen skriptauskielellä. Ko ym. (2011) määrittelevät tarkemmin skriptikielten usein tarkoittavan korkeatasoisempia kieliä, jotka tulkataan kääntämisen sijaan. He jatkavat niillä pyrittävän yhdistelemään ja hallitsemaan allaolevien kokonaisuuksien yhteistyötä. Tratt (2008) määrittelee sovellusalueen kielen olevan kieli, joka on pienempi ja vähemmän yleiskäyttöinen kuin esimerkiksi Java, C++ tai Python. Hänen mukaansa perinteisesti sovellusalueen kielet muodostetaan tyhjästä ratkaisemaan käsillä olevaa ongelmaa itse-

näisinä järjestelminä, mikä lisää kehittäjien työtä ja saattaa tuottaa vaihtelevia tuloksia tuotetun kielen laadussa. Hän jatkaa sovellusalueen kielien myös yleensä laajenevan ajan myötä lainaten yleiskäyttöisemmpistä kielistä ominaisuuksia. Tällöin sovellusaluekieli saattaa alkaa muistuttamaan yleiskäyttöistä kieltä, mutta heikompilaatuisena, sillä siihen ei alunperin suunniteltu ajansaatossa lisättyjä ominaisuuksia. Tratt (2008) neuvookin välttämään sovellusalueen kielen tekemistä itse edellä luetelluista syistä. Ko ym. (2011) toteavat loppukäyttäjien voivan käyttää laajaa kirjoa erilaisia kieliä esimerkiksi makrojen nauhoittamisesesta sovellusalueenkieliin tai perinteisempiin yleiskäyttöisempiin kieliin, esimerkiksi C++:aan. He jatkavat tärkeintä olevan sen, kuinka hyvin kieli auttaa käyttäjää ratkaisemaan ongelmansa. Lieneekin perusteltua käyttää käyttäjälähtöistä muokattavuutta tukevissa peleissä valmiita, tulkattavia ja yleiskäyttöisiä kieliä ja tehdä skriptimoottorit näillä kielillä.

Parnas (1979) esittelee tarkemmin tuoteperheiden käyttöä tapana laajentaa sovelluksia. Hänen mukaansa sovelluksia ei pitäisi ajatella yksittäisinä sovelluksina, jotka ratkaisevat yhden ongelman, vaan pikemminkin sovellusperheinä tai tuoteperheinä. Parnasin (1979) mukaan sovellusperheen sisällä sovellukset voivat erota toisistaan viidellä tavalla:

1. ne voivat toimia erilaisella laitteistokokoonpanolla
2. ne voivat toimia samalla tavoin, mutta niihin voi tulla tai niistä voi lähteä dataa eri muodossa
3. niissä voi olla eroja käytetyissä tietorakenteissa tai algoritmeissa:
    - tarjolla olevien resurssien takia. Pieni välimuisti on mielestäni hyvä esimerkki tilanteesta, jossa tämä täytyy ottaa huomioon.
    - sisäänotetun datan tai tapahtumien tiheyden takia.
4. jotkut käyttäjät saattavat tarvita vain osaa toiminnallisuuksista, joita muut käyttäjät tarvitsevat. Nämä käyttäjät eivät välttämättä halua maksaa toiminnallisuuksista, joita he eivät tarvitse.

Sovellus pitäisikin suunnitella Parnasin (1979) mukaan niin, että se pystyy muuttumaan edellä määriteltyjen tapausten sisällä, ilman valtavaa koodin refaktorointia.

Parnas (1979) käsittelee neljä yleistä syytä, miksi sovellukset eivät yleensä ole laajennettavissa. Hänen mukaan sovelluksien laajennettavuus on hankalaa joko liiallisen datan jakamisen seurauksena, dataa muokkaavan ketjun seurauksena, useiden toimintojen toteuttavien komponenttien vuoksi tai kokonaisuuksien käytön monimutkaisuuden johdosta. Esittelen lyhyesti nämä ongelmakohdat seuraavissa kappaleissa.

Parnas (1979) tarkoittaa *liaallisen datan jakamisella*, että sovelluksen sisällä liian useassa paikassa on oletettu jonkun tietyn asian pitävän paikkansa. Hän antaa esimerkin käyttöjärjestelmästä, jossa suurimmassa osassa ohjelmakoodia oletettiin mahdollisten kielten määräksi tasan kolme. Järjestelmään olisi tarvinnut tehdä valtavia muutoksia, mikäli kieliä olisikin ollut neljä tai kaksi.

Dataa muokkaavan ketjun ongelma Parnasin (1979) mukaan on, että sovelluksessa on pieniä ohjelmakokonaisuuksia, joista ensimmäinen komponentti lähettää seuraavalle komponentille datan tietyssä muodossa ja tämä komponentti lähettää datan toisessa muodossa edelleen kolmannelle komponentille. Jos välistä poistaisi toisen komponentin, eivät komponentit enään pystyisi suoraan keskustelemaan keskenään. Haluaisin tosin huomauttaa, että tämä on nykyaikana käytetty arkkitehtuurityyli putkistoarkkitehtuuri (*pipeline architecture*, jolle löytynee omat käyttötarkoituksensa.

Komponentit, jotka toteuttavat useamman toiminnallisuuden, ovat Parnasin (1979) mukaan yleinen ongelma. Hänen mukaansa tälläisiä ongelmia syntyy, kun ohjelmoija on ajatellut ominaisuuden olevan liian pieni erotettavaksi toiseksi komponentiksi tai kun kaksi toiminnallisuutta on hyvin lähellä toisiaan. Hänen mukaansa tälläisissä tilanteissa tehokkaampi tai monikäyttöisempi komponentti olisi voitu toteuttaa yksinkertaisemmin, mikäli alunperin olisi tehty kaksi erillistä komponenttia.

Viimeinen esimerkki käsittelee tilannetta, jossa yksi *toiminnallisuus sovelluksesta perustuu toiseen kokonaisuuteen*. Tämä kokonaisuus taas saattaa nojata kolmanteen, jolloin koko sovellus toimii vasta, kun kaikki sovelluksen osat ovat valmiina. Parnas (1979) antaa esimerkkinä tähän käyttöjärjestelmän, jonka vuorottajan toiminnalli-

suus nojaa tiedostojärjestelmään. Kuitenkin esimerkiksi testauksen kannalta olisi hyvä, että vuorottajaa voisi käyttää ilman tiedostojärjestelmää. Omissakin projekteissa olen havainnut tälläisen rakenteen hyväksi, kun pelin logiikka oli erotettu vahvasti käyttöliittymästä. Tämä salli esimerkiksi tekoälyn testaamisen ilman käyttöliittymään liittyviä komentoja, mikä tehosti järjestelmän suoritusnopeutta merkittävissä määrin testauksessa.

Parnas (1979) määrittelee neljä erilaista tapaa parantaa sovelluksen laajennettavuutta: muutokset tulee ottaa huomioon jo suunnitteluvaiheessa, tieto pitää piilottaa komponenttien välillä, virtuaalikonemainen ajattelu ja suunnittele *käyttää*-relaatio sovelluksen osien välillä. Nämä ehdotukset ovat mielestäni toimivia käytänteitä myös yleisessä sovelluskehityksessä tänäkin päivänä.

Ensimmäisessä ohjeessa tulisi ottaa huomioon, että muodostetaan pienin osajoukko toiminnallisuua, jotka sovelluksen mahdollisesti pitäisi pystyä tekemään. Parnas (1979) lisää, että suurimman joustavuuden saa aikaiseksi tekemällä mahdollisimman pieniä lisäyksiä kerrallaan sovellukseen.

Toisella ohjeella, tiedon piilottamisella, Parnas (1979) tarkoittaa, että kaikkien muuttuvien komponenttien välille kannattaa tehdä rajapinta. Tämän rajapinnan pitäisi pysyä aina mielekkäänä ja olla yleisluontoinen, mutta rajapinnan sisällön toteuttavan komponentin ei. Hän jatkaa komponenttien erikoistumisen olevan tarpeellista ekonomisuuden ja tehokkuuden ylläpitämiseksi.

Kolmantena ohjeena Parnas (1979) esittelee virtuaalikonemaisen ajattelun. Sen sijaan, että tehtäisiin järjestelmiä, jotka ottavat sisältöä ja antavat ulos toisenlaista dataa, tulisi järjestelmän olla sellainen, että sille määriteltäisiin, miten sen tulisi toimia. Tässä korostuu esimerkiksi se, että ohjelmoija, joka määrittelee sovellusalueen kielellä ohjeita sovellukselle, ei pitäisi suoraan kommunikoida termein, jotka viittaavat alla olevaan järjestelmään tai laitteistoon. Muutosta alla olevan järjestelmän kutsuista virtuaalikoneen kutsuihin ei myöskään tarvitse tehdä kerralla. Ongelman voi pilkkoa pieniin osiin ja tehdä pieniä osakokonaisuuksia, jotka toteuttavat jonkun toiminnallisuuden. Tämän ajattelun voi rinnastaa esimerkiksi pelimoottorin päällä

pyörivään peliin, jossa pelimoottorin logiikka keskustelee itse pelimoottorin kanssa.

Neljäntenä ohjeena Parnas (1979) esittää *käyttää*-relaation suunnittelun sovelluksen osien välillä. Hänen mukaan *käyttää*-relaatiossa A:n toiminnallisuus vaatii B:n toimivan täydellisesti. Sen sijaan, että B:n täytyy olla täysin valmis, tulisi A toteuttaa niin, että A on valmis, kunhan se vain kutsuu B:tä onnistuneesti. Hän myös ehdottaa jakamaan sovelluksen hierarkian tasoihin, niin että tasolla 0 oleva komponentti ei tarvitse mitään muuta komponenttia toimiakseen ja tasolla i oleva komponentti tarvitsee vain i - 1 tason osia toimiakseen. Hänen mukaansa tälläisen hierarkian löytyminen sovelluksesta takaa sen, että jokainen taso on yksilöllinen osakokonaisuus sovelluksesta ja siten testattavissa erikseen.

Pelien modaus on onnistunut muoto loppukäyttäjien sovellusten muokkaamisesta (Scacchi 2011a). Ko ym. (2011) toteavat loppukäyttäjien sovelluskehityksessä vaatimuksien olevan lähtöisin käyttäjiltä itseltään, heidän perheiltään tai ystäviltään. Usein peleissä pelaaja itse on innokas tekemään muutoksia peliin ja haluaa itse päästä kokemaan pelissä jonkun ominaisuuden tai muuttamaan peliä hänelle mieleiseksi. Koska käyttäjällä on vahva ymmärrys, mitä hän haluaa pelissä, ovat vaatimukset suoraan hänen tiedossaan. Yksi ongelmakohta, jonka Ko ym. (2011) tuovat esille, on käyttäjän kannustaminen sovelluksen muokkaukseen: käyttäjä ei välttämättä halua tehdä sitä. Peleissä tälläistä ongelmaa ei vaikuttaisi olevan, sillä into muokata pelejä lähtee käyttäjästä itsestään. Pelien modausyhteisöt alkavat myös olemaan niin isoja, että yksittäinen modaaja saa niistä tukea omien ongelmiensa ratkaisemiseen. Peliyhtiöiden innokkuus käyttäjälähtöisen muokattavuuden sallimiseen helpottaa myös tässä, sillä peliyhtiön työntekijät voivat auttaa isommissa modaajayhteisöä koskevissa ongelmakohdissa. Kuten Scacchi (2011a) esittää, voitaisiin peleissä tapahtuvasta sovelluksien muokkauksesta ottaa oppia myös muussa sovelluskehityksessä. Tämä on perusteltua mielestäni edellä käsiteltyjen ajatuksien valossa.

2.2 Käyttäjälähtöinen muokattavuus peleissä

Pelien käyttäjälähtöinen muokattavuus on Scacchin (2011b) mukaan nousemassa johtavaksi tavaksi kehittää ja kustomoida pelisovelluksia. Useimmiten modeja kehittävät pelin innokkaimmat pelaajat (Scacchi 2011b). Scacchi (2010) kuvailee modaamista *meta-pelaamisen muotona*: pelaajat pelaavat peliä muokkaamalla sen järjestelmiä (ks. myös Kow ja Nardi 2010). Postigo (2007) esittää modien olevan mahdollinen testialusta uusille peli-ideoille. Hän tarkentaa, että jos pelinkehittäjät huomaavat yksittäisen modin olevan hyvin toimiva osa peliä, kehittäjät voivat palkata moditiimin kehittämään modista uuden pelikokonaisuuden. Käyttäjälähtöinen muokattavuus on myös pääsääntöisesti riippuvaista OpenSource-pohjaisista laajennoksista ja pitää sisällään ison yhteisön OpenSource-projekteja, jotka kehittävät tietokonepelisovelluksia ja työkaluja (Scacchi 2011b).

Champion (2012) käsittelee internetistä löytyviä julkaisuja, joissa on keskusteltu käyttäjälähtöisestä muokattavuudesta. Tässä listauksessa pelimodit ovat herättäneet keskustelua filosofisista kysymyksistä, aiheuttaneet turvallisuusselkkauksen ja nöyryytyksen USA:n turvallisuuspalveluissa tai käsitelleet Australian pakolaispolitiikkaa. Kirjallisuudessa on myös käytetty muokattavissa olevia pelejä erilaisiin opetustarkoituksiin (Lowe 2009; El-Nasr ja Smith 2006; Yucel, Zupko ja El-Nasr 2006).

2.2.1 Käyttäjälähtöisen muokattavuuden historiaa

El-Nasrin ja Smithin (2006) mukaan modit alkoivat yleistymään 90-luvun loppupuolella. Ensimmäinen pelimodi on tehty jo 1960-luvun alussa, kun MIT:n[1] opiskelijat kehittivät *Spacewar!*-pelin DEC PDP-1:lle (vuonna 1961 Nardi 2010, s.144; vrt. vuonna 1962 Christiansen 2012; Champion 2012). Christiansen (2012) toteaa Spacewar!-pelin olleen hakkereiden yhteistyön tulosta ja se oli tarkoitettu jaettavaksi muille pelaajille ilmaiseksi. Doomin esitetään olleen ensimmäinen muokattavaksi tarkoitettu peli ja se julkaistiin vuonna 1993 (Champion 2012; Sotamaa 2007). Peliin lisättiin esimerkiksi tähtiä taustalle ja painovoiman laskenta, ja peliä varten teh-

1. Massachusetts Institute of Technology

tiin yksinkertaisia ohjaimia hylätyistä osista. Hän myös , että ensimmäinen modeihin liittyvä oikeuskäsittely syntyi, kun kaksi MIT:n opiskelijaa valmistivat ja myivät *Missile Command* -pelihallipeliin modi-piiriä, jolla sai lisää pelattavaa. Oikeuskäsittely päättyi opiskelijoiden hyväksi Atarin sovittua käsittelyn maksaen opiskelijoille korvauksia, mikäli he eivät jatka piirien myyntiä (Christiansen 2012).

Christiansen (2012) tuo esiin Id Software -yhtiön olleen tyytyväinen modaajien intoon muokata Wolfenstein 3D -peliä ja päättäneen julkaista seuraavan pelinsä, *Doomin*, helpommin modaajille muokattavana. Kaikki pelin data laitettiin WAD (*Where's All the Data?*)-tiedostoihin, jotta modaajat pystyisivät helpommin muokkaamaan peliä. Doomista tehtiinkin esimerkiksi *Chex Quest* -peli, jota jaettiin muropaketeissa oheistuotteena (Christiansen 2012). Christiansen (2012) esittää Id Software -yhtiön jatkaneen avointa lähestymistään käyttäjälähtöiseen muokattavuuteen ja julkaisseen tulevatkin pelinsä käyttäjien muokattavina. Valve-yhtiö hyödynsi heidän avoimuuttaan ja lisensoi Id Softwaren *Quake II* -pelin moottorin ja teki ensimmäisen pelinsä *Half-Lifen* käyttäen sitä. Christiansen (2012) jatkaa Valven tehneen peleistään myös muokattavia, mikä synnytti innokkaan modaajayhteisön pelin ympärille. Half-Life-pelin modista taas syntyi *Counter-Strike*-peli (Christiansen 2012; Postigo 2007; Sotamaa 2007).

2.2.2 Mitä ovat modit?

Modi sana itsessäänkin on hyvin laaja käsite. Champion (2012) pohtiikin olevan hyvin mielivaltaista, mikä lasketaan modiksi. Scacchi (2010) ottaa hyvin laajan näkökulman siihen, mikä lasketaan modiksi. Scacchin (2010) mukaan mikä tahansa kustomointi, räätälöinti tai uudelleenmiksaus, oli se sitten sisällön, sovelluksen tai laitteiston muokkaus kelpuutetaan modiksi. Hänen näkökulmansa mukaan modaus voidaan jakaa viiteen erilaiseen muotoon: käyttöliittymän kustomointi, pelimuunnokset, machinima[2] ja taide modit, pelitietokoneen kustomointi[3] ja pelikonsolin

2. machinima tarkoittaa animoitujen filmien luomista käyttämällä videopelin grafiikkaa hyväksi.

3. Walt Scacchi ei enään myöhemmissä artikkeleissaan listaa pelitietokoneen kustomointia modauksen muodoksi (Scacchi 2011b, 2011a), mutta esimerkiksi Nardi (2010, s.144) näkee tietokoneen parantelun modauksen muotona. Champion (2012) huomioi, että tietokoneen muokkaukset ovat

hakkerointi. Usein kirjallisuudessa löytyy kolme (Kow ja Nardi 2010) tai neljä näistä muodoista (Scacchi 2011b), mutta Scacchin (2010) määritelmä modeista tuntuu sisällyttävän muut kirjallisuudessa esille tulleet määritelmät, joten käytän itsekkin tätä määritelmää.

Käyttöliittymän kustomointi on yleensä hyvin tarkoin kehittäjien määrittelemä modauksen muoto, jossa peliyhtiö tarjoaa tarkoin määritellyn rajapinnan, jonka avulla pelin käyttöliittymää voi muokata tai parantaa (Scacchi 2010). Scacchin (2010) mukaan pelinkehittäjät käyttävät tätä ominaisuutta taatakseen mahdollisimman suuren todennäköisyyden tuotteen onnistumisesta. Osa käyttäjistä taas saattaa pyrkiä hankkimaan itselleen etulyöntiaseman muihin pelaajiin nähden näyttämällä mahdollisimman paljon tietoa pelitilasta.

Scacchi (2010) jakaa *käyttöliittymän kustomoinnin* vielä tarkemmin kolmeen erilaiseen ilmentymään: pelihahmon vaatetuksen ja muokkaamisen salliminen, käyttöliittymän komponenttien värimaailman ja ulkoasun muokkaaminen sekä lisäkomponentit. Postigo (2007) kutsuu pelihahmon vaatetuksen tekijöitä nimityksellä *skinner*, tosin hänen määritelmäänsä skinneristä kuuluu, että pelaaja voi kehittää myös lisää työkaluja tai esineitä pelimaailman sisälle, mikä on ristiriidassa Scacchin (2010) määritelmän kanssa. Lisäkomponenteilla pelaaja voi kerätä ja näyttää esimerkiksi lisää informaatiota pelimaailmasta ja sen tilasta lisäämällä käyttöliittymään komponentteja. Scacchin (2010) mukaan käyttöliittymän kustomointi voi syventää pelaajan tunnetta pelin osallisena olemisesta eli pelaajan immersiota. Mielestäni kuitenkin käyttöliittymän muokkauksilla voi olla myös päinvastainen vaikutus: On huomionarvoista, että pelaaja saattaa modien avulla myös pyrkiä hankkimaan tietoa pelin ulkopuolisesta maailmasta. Pelaaja voi lisätä peliin esimerkiksi kellon, joka näyttää ajan pelin ulkopuolella. Tämä kello tai muu samankaltainen komponentti, saattaa rikkoa pelaajan tunnetta olla osana pelimaailmaa.

Pelimuunnokset ovat käyttäjälähtöisen muokattavuuden muoto, jossa käyttäjä muokkaa pelihahmoja, peliolioita, aseita, taikoja, kenttiä, maastoa, pelin sääntöjä tai pe-

oma muokkaustyylinsä, mutta toteaa muiden tutkijoiden pitäytyvän tästä näkökulmasta, jotka julkaisivat artikkelinsa samassa kirjassa.

limekaniikkoja (Scacchi 2010). Pelimuunnokset ovat Scacchin (2010) mukaan ehkä yleisin pelimuokkausten muoto. Hän huomauttaa suurimman osan pelimuunnoksista olevan vain osittaisia, eli ne muokkaavat vain muutamia osa-alueita pelistä. Postigo (2007) kutsuu pelimuunnosmodien tekijöitä nimenomaan modaajiksi ja esimerkiksi tarkemmin karttoihin keskittyviä kehittäjiä nimityksellä *mappers* eli karttojen tekijät. Jotkut muokkaajista menevät niinkin pitkälle, että tekevät täydellisen pelimuunnoksen, jossa uudesta pelistä on vaikeaa tietää, minkä pelin pohjalta se on tehty (Scacchi 2010; ks. esimerkki Chex Questistä Christiansen 2012) .Scacchi (2010) myös huomauttaa joidenkin tekevän peleistä parodioita, joissa sisältöä tai pelimekaaniikkaa muokataan humoristiseksi kopioksi alkuperäisestä tai muokattavaa peliä käyttämällä tehdään parodiaa jostain muusta.Christiansen (2012) antaa esimerkkinä Andrew Johnsonin ja Preston Nevinsin, jotka muuttivat eri pelejä sisältämään smurffi-hahmoja, sillä he vihasivat smurffeja.

Machinima ja *taidemodit* pyrkivät muokkaamaan itse pelikokemusta. Scacchi (2010) kuvailee pelejä käytettävän niissä johonkin muuhun tarkoitukseen, esimerkiksi elokuvamaiseen esitykseen tai interaktiivisen taiteen luontiin. Hän tarkentaa, että machinimassa voidaan pelata peliä uudelleen ja nauhoittaa tämä pelikokemus. Näitä pelivideoita voidaan muokata ja uudelleenmiksata toisten medioiden, esimerkiksi äänitteiden, kanssa, jotta voidaan saavuttaa parempi elokuvamainen tarinankerronta tai luova performanssidokumentaatio. Taidemodit hänen mukaansa käyttävät pelimoottoria tai peliä avuksi staattisen, dynaamisen tai performanssitaiteen esittämiseen.

Scacchi (2010) näkee myös *pelitietokoneen muokkauksen* yhtenä modauksen muotona. Hän selittää tässä muokkauksen muodossa pelaajien yrittävän joko virittää tietokoneensa mahdollisimman tehokkaaksi tai koristelevan tietokoneen kuoren ja muun ulkoasun mahdollisimman näyttäväksi. Hänen mukaansa tietokoneen koristelemisella pelaaja voi tuoda omaa omistautumistaan pelaamiselle esiin.

Viimeisenä erilaisena muokkauksen muotona Scacchi (2010) näkee *pelikonsolin hakkeroinnin*. Hän määrittelee tässä muokkauksessa käyttäjän pyrkivän laajentamaan kokemuksien kirjoa, jota pelikonsolilla voi kokea. Esimerkiksi käyttäjä voi muoka-

ta pelikonsolista henkilökohtaisen tietokoneen kaltaisen laitteen. Hän huomauttaa tästä saatavan kokemuksen ja ymmärryksen voivan olla hyödyksi tietoteknisten innovaatioiden tekemiseksi.

2.2.3 Modien levitys

Modit ladataan yleensä erillään itse pelistä johon ne tulevat, joten niillä pitää olla jonkinlainen jakelualusta. Modien jakelu tapahtuu omien huomioideni mukaan joko pelin sisäisesti, pelin tarjoavan palvelun kautta (esim. Steam Christiansen 2012; Postigo 2007), yleisen modien jakosivuston avulla (esim. Kow ja Nardi 2010; Scacchi 2010; Christiansen 2012; Poretski ja Arazy 2017) tai pelin omilla verkkosivuilla tai keskustelupalstoilla. Yksittäisiä modeja voi toki löytyä myös irrallaan modin kehittäjän omilta verkkosivuilta tai keskustelupalstalta, jossa modia kehitetään. On huomionarvoista, että modien mukana voitaisiin siirtää ajettavaa ohjelmakoodia, minkä takia modit voivat aiheuttaa tietoturvaongelmia.

Pelipalvelu *Steam* on Beckerin ja muiden (2012) mukaan moninpeli- ja kommunikointialusta, joka jakaa online-pelejä sekä pienemmiltä että isommilta kehittäjiltä. Haluan huomauttaa, että palvelussa voi ostaa myös pelejä, jotka eivät vaadi internet yhteyttä. Poretski ja Arazy (2017) käsittelevät uutisartikkelia,[4] jonka mukaan Steamissa myytiin 75% kaikista internetissä myydyistä peleistä vuonna 2013. Heidän mukaan Steamilla on myös yli 125 miljoonaa aktiivista käyttäjää. Steam-palvelussa voi myös jakaa modeja ja modaustyökaluja (Christiansen 2012; Postigo 2007). Tätä palvelua kutsutaan *Steam Workshopiksi* (Poretski ja Arazy 2017). Kehittäjien, jotka haluavat tehdä muokattavan pelin, kannattaakin tutustua kyseiseen palveluun. *Wikipedia - Steam Workshop games* (2016) listaa pelejä, jotka käyttävät apuna tätä palvelua.

Scacchi (2010) ja Christiansen (2012) toteavat `http://www.moddb.com`-sivun tarjoavan alustan modien jakamiselle ja keskustelulle. NexusMods-sivua[5] käytetään

4. `https://www.bloomberg.com/news/articles/2013-11-04/valve-lines-up-console-partners-in-challenge-to-microsoft-sony`
5. `http://www.nexusmods.com/games/?`

myös aktiivisesti modien jakamiseen (Poretski ja Arazy 2017). Poretskin ja Arazyn (2017) mukaan sivu on suurin modaajayhteisö sisältäen modeja yli 250 suurimmalle pelille. He jatkavat yhteisön koostuvan yli kymmenestä miljoonasta rekisteröidystä käyttäjästä, ja joka kuukausi sivulle tulee yli 1500 uutta modia.

Yksi vielä hyvin vähän huomiota kirjallisuudessa saanut seikka on modien turvallisuus. Modit voivat sisältää koodia, joka ajettaessa tekee erilaisia ikäviä asioita käyttäjän tietokoneelle. Esimerkiksi *Cities Skylines*[6]-pelin verkkosivuilla tuodaan esille, että modit saattavat aiheuttaa tietoturvariskin:

> The code in Mods for Cities: Skylines is not executed in a sandbox. While we trust the gaming community to know how to behave and not upload malicious mods that will intentionally cause damage to users, what is uploaded on the Workshop cannot be controlled. Like with any files acquired from the internet, caution is recommended when something looks very suspicious. (ks. Security Considerations *Modding API* 2017)

Myös *Minecraft*-peliin liittyen löytyy uutinen, jossa modeja oli käytetty viemään käyttäjä huijaussivustoille (Zorz 2017). Asia noussee kiinnostavaksi keskustelun aiheeksi tulevaisuudessa.

2.3 Modien tekijänoikeudet

Yksi paljon keskustelua herättävä aihealue modauksessa tuntuu olevan modien tekijänoikeudet. Kuuluvatko esimerkiksi pelimuunnoksien tekijänoikeudet itse tekijälle vai pelin alkuperäiselle tekijälle vai jotenkin yhteisesti kaikille? Scacchi (2010) toteaa peleissä usein olevan kahdet erilliset ehdot. Toiset ehdot suojelevat itse pelimoottoria, jotta sitä ei voitaisi jakaa eteenpäin käyttäjien toimesta ja toiset ehdot käsittelevät modien jakamista. Kowin ja Nardin (2010) mukaan peliyhtiöt käyttäytyvät usein siten, kuin modien oikeudet kuuluisivat heille. He korostavat selvästi,

6. Cities Skylines on suomalaisen julkaisijan Colossal Orderin Unityllä valmistama kaupungin rakennuspeli, jossa käyttäjälähtöistä muokattavuutta varten on tehty kattavat työkalut muokkauksen sallimiseksi. Verkkosivut osoitteessa `http://www.citiesskylines.com/`

että näin ei välttämättä ole, mutta tällä hetkellä kukaan ei tunnu kyseenalaistavan tilannetta. Poretski ja Arazy (2017) ajattelevat hyvin samankaltaisesti Kowin ja Nardin (2010) kanssa. He toteavat pelin käyttöehtojen usein pakottavan luovuttamaan oikeudet modeihin peliä kehittävälle yritykselle. Postigo (2008) esittelee kolme erilaista termiä, joita tutkimuksessa on käytetty modaajien tekemästä työstä: ilmainen työ, näkymätön työ ja leikkillinen työ (playbour). Hän haluaa tuoda artikkelissaan esille hyödyn epätasa-arvoa, joka modien kehittäjien ja peliyhtiön välillä vallitsee. Rahallinen hyöty modauksesta menee usein kokonaisuudessaan pelinkehittäjille.

Peliyhtiöt voivat asettaa sääntöjä modien kehitykselle kahdella tavoin: ohjelmoimalla ne suoraan pelisovellukseen tai lakiteitse (Kow ja Nardi 2010). Kow ja Nardi (2010) huomauttavat jälkimmäisen tavan heidän esimerkissään herättäneen valtavaa vastarintaa - modaajat eivät tykänneet, kun heitä uhkailtiin lakitoimilla. Kun taas toisessa heidän esittelemässä esimerkissä modia täytyi muokata, sillä se yksinkertaisti liikaa itse peliä. Pelin sisäistä modien sallintalogiikkaa muokkaamalla pystyttiin estämään tämänkaltainen modi, eikä modaajayhteisö ollut hermostunut asiasta.

Suurin osa modaajista tekee modausta harrastuksena, ja vain pieni osa modaajista tekee modausta pääsääntöisenä työnään[7] (Kow ja Nardi 2010). Modaajat saattavat lopettaa modien tekemisen milloin vain, sillä heitä ei sido minkäänlainen työsopimus tai pakote modien jatkamiseen (Kow ja Nardi 2010). Kow ja Nardi (2010) haastattelivat modaajayhteisön jäseniä, ja tutkimuksen mukaan vaikutti olevan yleisesti hyväksyttävää jatkaa toisen modin tekemistä, mikäli modi oli hylätty. He myös huomauttavat käyttäjien kokevan, että yksittäinen henkilö omistaa aina modin ja muut ovat enemmänkin auttamassa häntä. Modin omistajuuden voi siirtää eteenpäin. Vaikka näitä "sääntöjä" ei pystytä lainvoimaisesti ylläpitämään, Kow ja Nardi (2010) toteavat isompien jakelusivustojen poistavan näitä sääntöjä rikkovat modit, mikä vahvasti rajoittaa sääntörikkomuksia. He jatkavat modaajien taas vieroksuvan sivustoja, jotka eivät ole näitä yhteisön säännöksiä pitäneet yllä.

7. Itselleni jäi epäselväksi, viittasivatko Kow ja Nardi (2010) tällä World of Warcraftiin vai yleisesti pelialalla.

Scacchi (2010) esittelee kaksi esimerkkiä erilaisista käytännöistä ehtoihin liittyen. Scacchi (2010) kuvailee, kuinka *Aion*-verkkomoninpelissä mitään käyttäjien tekemiä modeja ei saa käyttää, ja mikäli pelaajan havaitaan käyttävän modeja, saatetaan hänen pelitilinsä poistaa. Toisena esimerkkinä hän esittää *World of Warcraft* -pelin, jossa käyttöliittymämuokkaukset ja lisäykset ovat sallittuja, mutta mikään muu pelimuunnos, takaisinmallinnus (*reverse engineering*) tai aktiviteetti, jolla pyritään ohittamaan pelin enkryptointimekanismit, eivät ole.

Kow ja Nardi (2010) käsittelevät *World of Warcraft* -peliin liittyvää tilannetta, jossa pelinkehittäjä, Blizzard-peliyhtiö asetti uusia ehtoja modien kehitykselle. Varsinkin kaksi näistä ehdoista sai valtavasti keskustelua aikaiseksi: modit pitää jakaa ilmaiseksi ja niissä ei saa pyytää lahjoituksia pelin sisäisesti. Käyttäjät eivät pitäneet ajatuksesta, että heillä ei olisi oikeutta myydä heidän tekemäänsä tuotetta eteenpäin. Jälkimmäinen ehto vaikeutti lahjoituksien keruuta, sillä useat käyttäjät eivät ladanneet modeja niiden omilta kotisivuilta, jossa lahjoituksia voisi antaa, vaan yleisiltä jakeluportaaleilta. World of Warcraftin modien kehittäjäyhteisö hermostui Blizzardin tekemistä muutoksista, sillä he olivat uskoneet, ettei Blizzard pyrkisi vaikuttamaan modien kehitykseen millään tavoin.

Kow ja Nardi (2010) jatkavat selittäen Blizzardin näkökulmaa tilanteeseen. Blizzard ei halunnut peliinsä lisämainoksia, sillä se olisi voinut heikentää itse pelikokemusta. Kow ja Nardi (2010) tuovat esille, että pelaajat olivat myös olleet huolissaan modien maksullisuudesta tai siitä, että heidän pelikokemuksensa kärsisi lahjoituspyyntöjen takia.

Modien jakelusivustot tulivat avuksi ja tarjosivat lahjoitusmahdollisuuden modien lataussivulle (Kow ja Nardi 2010). Tämä tasapainotti tilannetta ja aktiivisimmat modaajat saivat jonkun verran rahaa modeistaan, vaikkeivat niin paljoa kun aikaisemmin (Kow ja Nardi 2010). Kuitenkin kaikista eniten latauksia saaneet modintekijät jättivät pelin modauksen tai vähensivät modauksien tekemiseen käytettyä aikaa (Kow ja Nardi 2010). Postigo (2008) huomauttaa, että jos modaajat ja tekijänoikeuksien haltijat eivät ole yhteisymmärryksessä, voi yritysten tarpeet olla esteenä luovalle modaustyölle. Tällöin modaajat turhautuvat, sillä he eivät välttämättä voi työs-

kennellä sisällön kanssa, jota he rakastavat. Hän jatkaa modaajien tukijoiden ja pelin fanien olevan vihaisia, mikäli he eivät pääse käsiksi innovatiivisiin modeihin.

Postigo (2008) esittelee kaksi tilannetta, jossa modaajien työ oli mennyt hukkaan ulkopuolisten tekijänoikeuksien takia. Toisessa esimerkissä modaajatiimi oli tehnyt *Quake 3*:n pelimoottorille *Duke Nukem 3D* -pelin kaltaisen modin (Postigo 2008). Apogee, Duke Nukem 3D -pelin tehnyt yritys, vaati modin poistamista, sillä se rikkoi heidän tekijänoikeuksiaan: *Quake 3* -pelin omistajat pystyivät pelaamaan heidän peliään ilmaiseksi (Postigo 2008). Apogeen vaatimukset suututtivat suuren osan heidän fanikunnasta. Mielestäni Postigon (2008) lainauksista parhaiten fanien reaktiota toi esille lainaus:

> On the one hand, he [George Broussard] has a point about it being copyrighted material. On the other hand, you've got stuff like Star Trek fan fiction on the web everywhere, and the lawyers don't do anything about it. The reason is that these fans are the core of the community. The fact that they produce fan fiction (which is I think a pretty close analogy to this Duke → Q3 stuff) increases interest in the commercial product. I'd say, let it slide unless someone tries to make money off of it. Consider it flattery, and free advertising, and whetting of the appetite for Duke4 . . . (archvile, 2001)

"archvile" on henkilön käyttäjätunnus keskustelupalstoilla, josta Postigo oli kerännyt vastauksia.

Toisessa Postigon (2008) esimerkissä modaajat olivat tehneet *Battlefield 1942* -peliin modin, jossa oli lisätty peliin *GI Joe* -sarjan hahmoja ja tuotteita. Hasbro, GI Joe -sarjan kehittäjä, lähetti näille modaajille vaatimuksen lopettaa modin kehittämisen. Postigon (2008) mukaan fanit jakautuivat kahtia reaktioissaan Hasbron vaateisiin: osa faneista näki modin tappamisen puhtaasti yhtiön ahneutena kun taas osa faneista uskoi Hasbron menettäneen lupaavan mahdollisuuden hyötyä ilmaisesta julkisuudesta. Hasbro ei suostunut minkäänlaiseen kompromissiin ja modaajat joutuivat käytännössä lopettamaan GI Joe -modin tekemisen.

Darkest Dungeon-pelissä, jossa käyttäjälähtöisen muokattavuuden salliminen on to-

teutettu kattavasti, on keskustelupalstoilla pyydetty, että jokaiseen julkisessa levityksessä olevaan modiin lisätään teksti (SneakyPie 2015):

> <mod name> is not an official Red Hook Studios product or product modification, and Red Hook Studios Inc. is not responsible in any way for changes or damages that may result from using the mod. Furthermore, "Darkest Dungeon" and the Darkest Dungeon logo are trademarks of Red Hook Studios Inc. All content in the game is Copyright Red Hook Studios Inc. All rights reserved.

Samaisessa, virallisessa keskustelupalstan viestissä korostetaan, että modeja ei saa kaupallistaa millään tavoin, sillä se olisi vastoin pelin tekijän tekijänoikeuksia. Tässä esimerkissä modit nähdään hyvin läheisesti olevan pelintekijän hallinassa. Esimerkiksi mikäli modin kanssa tulee ongelmia ja kuluttaja voisi saada käsityksen siitä, että modi on Redhooksin tekemä tai sen vastuulla, poistaisi Redhooks kyseisen modin heidän keskustelupalstoiltaan.

Witcher-pelisarjassa on poikkeavat ehdot edellä esiteltyihin ehtoihin verrattuna (Poretski ja Arazy 2017). Poretski ja Arazy (2017) toteavat Witcherin End user license agreement (EULA) -sopimuksen sisältävän seuraavanlaisen lausekkeen:

> As far as we and you are concerned, you own any User Generated Content you created but we need you to give us certain rights over it so that we can actually transmit it through CD PROJEKT RED games and services.

Nämä ehdot ovat vapaammat, eivätkä vaadi minkäänlaista omistajuuden luovuttamista modin tekijältä. Poretski ja Arazy (2017) esittävät tämän olevan yksi tapa yrityksille ilmaista tukensa käyttäjälähtöistä muokattavuutta kohtaan. Scacchi (2010) toteaa, että peleissä kuten Unreal Tournament, Half-Life, NeverWinterNights, Civilization ja monissa muissa, EULA kannustaa käyttäjälähtöiseen muokkaamiseen ja näiden modien ilmaiseen jakeluun muille käyttäjille, joilla on lisensoitu pelikopio. Näissäkään pelin takaisinmallinnus tai pelimoottorin jakaminen modien käyttämiseksi ei ole sallittua.

Modaajat tuntuvat olevan tyytyväisiä, mikäli heidän modinsa sisältö otetaan jollain tavalla mukaan peliin, eivätkä he ole kokeneet tämän polkevan heidän oikeuksiaan (Nardi 2010; Kow ja Nardi 2010). Nardi (2010) toteaa, että yksi modaaja oli tyytyväinen modin sisällyttämisestä peliin, mutta samalla tunsi haikeutta, sillä hänen harrastuksensa oli viety pois: modaajalla ei ollut enään ylläpidettävää modia huolehdittavanaan.

2.4 Modien vaikutus tuotteen myyntiin

Kirjallisuudessa tuntuu olevan vahvasti sellainen ymmärrys, että modit lisäävät tuotteiden myyntiä, pidentävät pelin ikää ja tarjoavat lisää pelattavaa ja tehtävää pelaajille peliin liittyen. Scacchin (2010) mukaan aiemmin käsitelty kahden erilaisen sopimuksen lähestymistapa lisää pelien myyntiä, sillä jos joku haluaa pelata kiinnostavaa modia, pitää hänen myös omistaa alkuperäinen peli. Kow ja Nardi (2010) myös näkevät World of Warcraftiin tehtyjen modien tuovan lisäarvoa itse pelille. Champion (2012) toteaa modien voivan pidentää pelin elinikää tarjoamalla lisää pelattavaa sisältöä pelaajille. Christiansen (2012) huomauttaa myös tämän sisällön olevan ilmaista pelinkehittäjille.

Champion (2012) käsittelee haastattelua liittyen Civilization II -peliin, jonka mukaan modit tarjosivat viraalimarkkinointia. Modaajat markkinoivat omia modejaan tutuilleen, lisäten pelin markkinointia. Modit tarjoavat myös kehitysideoita pelinkehittäjille (Champion 2012; Sotamaa 2007). Yksittäisen modin sisältö, idea tai teema voidaan ottaa mukaan esimerkiksi pelin lisäosaan tai jatko-osaan. Yhtenä esimerkkinä onnistuneimmista pelimuunnoksista Scacchi (2011b) esittelee aiemmin mainitun *Counter-Strike*-modin *Half-life*-pelistä, joka on ensimmäisen persoonan toimintapeli. Hänen mukaansa Valve oli myynyt yli 10 miljoonaa kopiota Counter-Strike variaatioista vuodesta 2008 lähtien tehden siitä vuoteen 2011 mennessä onnistuneimman pelimuunnoksen.

Postigon (2007) mukaan modit lisäävät rikkautta pelin sisältöön, mikä taas auttaa pelin jatkuvaan menestykseen. Hän kuitenkin pohtii, että modit eivät itsessään ta-

kaa pelin onnistumista, sillä pelin onnistuminen on monen tekijän summa. Hän toteaakin, että parhaimmillaan modit tuovat lisää syvyyttä valmiiseen peliin. Christiansen (2012) kuvailee modauksen olevan hyvä tapa tarjota sisältöä peleihin myös marginaalisemmille ryhmille. Hänen mukaansa modaajat pystyvät keskittymään paremmin ei-kaupallisen tai kaupallisesti riskialttiin sisällön tekemiseen, kun taas isommat yhtiöt keskittyvät enemmän vain hittipelien luomiseen.

Poretski ja Arazy (2017) käsittelevat uutisraporttia, jonka mukaan pelaajien määrä 20-kertaistui Valven julkaistua *Portal 2* -peliin modaustyökalut. Postigo (2007) kuvailee, kuinka hänen tarkastelemiinsa suurimpiin FPS-genren peleihin oli tehty sisältöä arviolta 10,1-30,4 miljoonan dollarin edestä "isoissa" modeissa. Lisäksi hän arvioi kyseisissä peleissä olleen pieniä, alle kymmenen megatavun modeja 2,5 miljoonan dollarin edestä. Hänen mukaansa esimerkiksi "Battle Field 1942"-pelissä[8] fanien tekemän sisällön hinta oli noin puolet siitä, mitä koko pelin kehittämiseen oli budjetoitu. Näihin laskelmiin hän oletti yksittäisen ohjelmistokehittäjän saavan keskimäärin 45 000 dollaria vuodessa ja tekevän töitä 40 tuntia viikossa 52 viikkoa vuodessa.

Modit saattavat aiheuttaa myös ongelmia pelinkehittäjille. Sotamaa (2007) kertoo tilanteesta vuodelta 2006, kun *Elder Scrolls IV: Oblivion* -pelin ikärajaa nostettiin 13 vuodesta 17 vuoteen alastomuutta lisänneen modin vuoksi. Sotamaa (2007) pohtii tämän saattaneen aiheuttaa menetyksiä pelinkehittäjälle eli Bethesda-yhtiölle. Hän kuitenkin jatkaa pohtimalla, että modi toi esille vain valmiiksi koneelle tallennettua sisältöä: modi vain riisui hahmot. Tämänkaltaiset tilanteet saattavat olla syy esimerkiksi aiemmin esiteltyyn Redhooks-yhtiön käytäntöön, jossa modeissa tulee todeta, ettei modi ole yhtiön tekemä ja yhtiö ei ota vastuuta modin käytöstä.

Kuitenkin Champion (2012) esittelee, kuinka Bethesda julkaisi blogissaan tukeneensa modausyhteisöään jo pitkään hyvästä syystä.[9] Bethesda ilmaisi modaustyökalujen tekevän maailmasta paremman paikan: ne tekevät modaajista onnellisia, koska he voivat modata, ne tekevät kehittäjistä onnellisia, kun pelinkehittäjät huomaa-

8. Oletettavasti Battlefield 1942, mutta kirjoitettu välilyönnillä Postigon aiemmassa artikkelissa
9. Bethesda on esimerkiksi Elder Scrolls-pelisarjan valmistaja.

vat modaajien saavan kokemusta ja ne tekevät faneista onnellisia, koska heille on loputon virta sisältöä, jonka he voivat kokea.[10] Postigo (2007) argumentoi, että jo se, että peliyhtiöt käyttävät aikaa käyttäjälähtöisen muokkauksen sallimiseen todistaa yhtiöiden ymmärtävän faniyhteisön pitkittävän pelin elinikää. Christiansenin (2012) mukaan keskimääräinen ikä pelille on noin viisi vuotta, mutta vuonna 1993 julkaistulla Doomilla on edelleen aktiivinen modaajayhteisö, joka jakaa vuosittain *Cacowards*-palkinnon parhaille modeille.

Poretski ja Arazy (2017) ehdottavat peliyhtiöitä viestittämään, että peliyhtiö tukee käyttäjälähtöistä muokkausta. Esimerkkinä he käsittelevät Postigon artikkelia "Modding to the big leagues: Exploring the space between modders and the game industry", jossa kerrotaan Epic Gamesin julkaisseen ilmaiseksi kehittäjätyökalunsa pelimoottorilleen käyttäjien käytettäväksi. Mikäli modaajat haluaisivat ansaita rahaa tuotoksellaan, tulisi heidän maksaa 99$ dollaria maksava lisenssi. Poretski ja Arazy (2017) kehottavat yrityksiä tuomaan avoimuuttaan ja tukeaan käyttäjälähtöistä muokattavuutta kohtaan esimerkiksi tarjoamalla jonkunlaisen työkalun käyttäjille muokkauksen helpottamiseksi tai muokkaustutoriaaleja tai ylläpitämällä keskustelua modaajien ja pelinkehittäjien välillä. He toteavat avoimuuden ja tuen viestimisen lisäävän modaajien halukkuutta muokata pelejä.

Poretski ja Arazy (2017) tekivät tutkimuksen, jossa he tarkastelivat modien vaikutusta tuotteiden myyntiin ja samalla kehittäjien avoimuuden vaikutusta modauksen määrään. Heidän tutkimuksen mukaan yritys pystyi vaikuttamaan tuotteiden myyntiin kannustamalla *NexusMod*-palvelun modaajia modaamaan peliä, esimerkiksi tarjoamalla kehittämistyökalut yhteisölle. He toteavat ylipäätään, että mitä enemmän modeja käyttäjät tuottivat pelille, sitä enemmän peli myi heidän tutkimukensa aikana. Heidän lyhyessä yhteenvedossa keskiverto peli, joka myy 97 000 kappaletta, nousisi myynnissä 15 000 kappaletta, mikäli peli tukee modaajayhteisöä.

10. "Bethesda has a long history of supporting the modding community, and for good reason. It's a science fact that mod tools make the world a better place: they make modders happy because they can mod, they make developers happy to see modders gaining experience, and they make fans happy to see an endless stream of content they can mess around with." (Champion 2012). Kyseessä on verkkojulkaisu, joten lainauksella ei ole sivunumeroa.

Eli heidän mukaan käyttäjälähtöisen muokattavuuden tukeminen nostaisi myyntiä noin 15 %. He tosin toteavat, etteivät tulokset ole niin yksinkertaisia, sillä muutkin asiat vaikuttavat tuotteen myyntiin.

Poretski ja Arazy (2017) toteavat lisensoinnin saattaneen myös vaikuttaa tuotteen myyntiin. He olivat arvioineet yhtiöiden lisenssejä joko modien oikeudet itselleen ottaviksi tai ei, ja tämän arvon vaikutus tuotteen myyntiin lähestyi merkittävän rajaa ($p = 0.09$). Tämä tulos on ajatuksen tasolla mielekäs: lisenssiehdoilla tai varsinkin yhtiöiden asenteella modauksen sallimiseen voi olla vaikutusta tuotteen myyntiin. Tätä tulosta ei kuitenkaan tieteellisestä näkökulmasta voida pitää pohdintaa arvokkaampana. Heidän tuloksissaan parhaiten myynnin osalta menestyivät yleisörahoituksen saaneet pelit.

Yksi vielä hyvin vähän huomiota saanut käyttäjälähtöisen muokattavuuden muoto on peleissä, missä koko pelin toiminnallisuus perustuu käyttäjien tekemään sisältöön. Tästä esimerkkinä voisi antaa *Super Mario Maker*-pelin,[11] missä käyttäjät voivat valmiin editorin avulla luoda kenttiä, joita muut pelaajat pelaavat ladattuaan kentän itselleen. Tässä pelissä suurin osa kentistä on käyttäjien tekemiä ja kenttiä on tehty yli 7,2 miljoonaa kappaletta (*Super Mario Maker Wiki* 2017). Peli vaikuttaa olevan suosittu ja tuntuu kaupallistavan hyvin käyttäjälähtöisen muokattavuuden ominaisuuksia: käyttäjät tekevät siihen loputtomasti lisää sisältöä muille pelaajille ja pelaajat nauttivat sisällön tekemisestä.

2.5 Pelaajien motivaatio modien kehittämiseen

Postigo (2007) havaitsi kolme pääsyytä, miksi pelaajat kehittävät modeja.

- taitellisen sisällön luominen
- samaistuminen peliin, lisäten nautintoa pelissä
- kokemuksen hankkiminen

11. Tarkempaa tietoa saatavilla esimerkiksi virallisilta kotisivuilta osoitteessa `http://supermariomaker.nintendo.com/`

Ensimmäisessä syyssä, taiteellisen sisällön luomisessa, korostuu pelaajien mahdollisuus tehdä jotain kaunista (Postigo 2007). Postigo (2007) totesi osan modaajista tekevän taiteellista sisältöä, koska he halusivat olla osallisena yhteisöä. Tätä muotoa korostaa esimerkiksi Postigon (2007) lainaus yhdestä hänen haastatteluistaan:

> Building these maps is like a form of technical art. Just like a painter has his canvas and paint pallet, I have my monitor and building tool program. (VikingWiedel, author of Keep map for Call of Duty, e-mail interview, 2004)

Postigo (2007) havaitsi toisena syynä pelaajien modaukselle olevan pelaajien halun samaistua peliin. Tämä huomio vaikuttaa hyvin samankaltaiselta Scacchin (2010) huomion kanssa, missä hän määrittelee modauksen olevan meta-pelaamisen muoto. Postigo (2007) selittää pelaajien halunneen tuntea pelin heidän omanaan lisäämällä siihen esimerkiksi joitain osia heidän kulttuuristaan.

Kolmantena syynä muokkauksien tekemiselle Postigo (2007) havaitsi pelaajien halun hankkia kokemusta peliteollisuuteen pääsemiseksi. Champion (2012) käsittelee myös modausta mahdollisena sisäänpääsyväylänä peliteollisuuteen. Kuten Valve-peliyhtiö teki Half-Life-pelimodin kehittäjille, Postigo (2007) tuo esille mahdollisuutta, että peliyhtiö palkkaa modaajat tekemään heidän modistaan osan peliä tai uuden pelin. El-Nasr ja Smith (2006) argumentoivat pelaajan oppivan monia hyödyllisiä taitoja modauksesta, kuten 3D grafiikkaa, vektori geometriaa, tapahtuma- ja oliopohjaista ohjelmointia, tekoälyjen käyttöä ja tietoteknillisiä ja estetiikkaan vaikuttavia periaatteita, mitä kaikkia käytetään pelien kehityksessä. Varsinkin isoissa modausprojekteissa, modaajat oppivat työskentelemään tiimeissä muiden kehittäjien kanssa (Scacchi 2011a).

Poretski ja Arazy (2017) toteavat omien kokemuksiensa mukaan pelin "lineaarisuuden" tai "sulkeutuneisuuden" vähentäneen käyttäjien intoa muokata pelejä. Mikäli peli oli lineaarinen, missä tarina oli ennalta määrätty ja pelaaja käy tarinan lävitse vaikuttamatta tarinan kulkuun, oli modaajien into heidän mukaan vähäisempi kuin peleissä, jotka olivat avoimempia maailmaltaan tai tarinaltaan. He jatkavat, että jos peli oli hyvin pieni ja yksinkertainen kokonaisuus, saattoivat modaajat olla muok-

kaamatta sitä. He pohtivat tämän johtuvan siitä, että modaajien ei tarvitse muokata peliä, koska he saivat jo halutun kokemuksen itse pelistä. Parhaiten kiinnostusta modaukseen heidän mukaan löytyi peleistä, joissa oli avoin maailma ja epälineaarinen pelikokemus. Voin itsekkin yhtyä heidän ajatuksiinsa ja todeta käyttäväni modeja avoimissa peleissä, jossa itse pelin päämäärä saattaa olla hyvinkin epäselvä.

Pelaajat kehittävät modeja modien kehittämisen hauskuuden takia tai joskus saadakseen taloudellista hyötyä (Kow ja Nardi 2010; Nardi 2010, s.145). Eräässä Nardin (2010, s.145) haastattelussa modaaja tarkentaa rahallisen edun tulevan modauksesta lahjoituksien tai modien *premium*-versioiden kautta. Modaajan mukaan suurimmalle osalle modaus on kuitenkin harrastus (*labor of love*) työn sijaan. Hän mainitsi myös kuuluisuuden olleen itselleen yksi syy internetissä kaikille jaossa olevien modien tekemiselle. Myös *QuestHelper* modin tekijä, joka pystyi elättämään itsensä modin avulla, totesi Kowin ja Nardin (2010) haastattelussa vastaavaa:

> Researcher: To you, where does hobby stop and a job begin?
>
> ZorbaTHut: To me, they're the same thing. If it's not worth doing, it's not worth doing. If it is worth doing, it is worth doing. "Worth" is part money and part enjoyment, but it's all part of the same equation. I don't really describe it as "hobby" or "job" anymore. It's just "what I do.'

Edellä esitellyt esimerkit tuovat esille modaajien innokkuutta tehdä modeja nimenomaan omista syistään. Modaajat kehittävät modeja ilmaistakseen itseään, samaistuakseen peliin paremmin tai oppiakseen uusia tekniikoita. Voin itsekkin yhtyä näihin ajatuksiin, sillä olen muokannut pelejä uteliaisuudesta, korjatakseni jonkun ärsyttävän yksityiskohdan pelissä tai tehdäkseni itselleni mieleisen pelitavan paremmin toimivaksi pelissä.

3 Unity ja käyttäjälähtöisen muokattavuuden salliminen

Tässä luvussa käsittelen Unity3D-pelimoottoria (myöhemmin pelkkä Unity), käyttäjälähtöisen muokattavuuden sallimista siinä ja esimerkkejä erilaisista tavoista toteuttaa käyttäjälähtöinen muokattavuus peleissä. Unityyn liittyvää opetusmateriaalia on paljon tarjolla, esimerkiksi Norton (2013) opettaa yksinkertaisien toiminnallisuuksien luomista C#:n kanssa Unityllä. Kehittyneimmille ohjelmoijille taas löytyy Murrayn (2014) kirja Unity-kehityksestä. Murray (s.1-8) käsittelee hyviä käytäntöjä Unitylla ohjelmoidessa ja ylipäätänsä pelejä tehdessä.

3.1 Pelimoottori Unity3D

Unity on suosittu 3D-pelimoottori. Sillä on tehty 34 % tuhannesta "parhaasta" mobiilipelistä (*Unity Fast Facts* 2017) ja se on varsin käyttökelpoinen pienille tiimeille ja yksittäisille kehittäjille (Xie 2012). Sitä voi käyttää ilmaiseksi ennen kuin yrityksen liikevaihto kasvaa sataan tuhanteen euroon (ks. ilmaisen käyttäjän oikeudet *Unity Subscriptions* 2017). Unityllä tehtyjä pelejä pelaa 770 miljoonaa pelaajaa ympäri maailmaa (*Unity Fast Facts* 2017). Unityn yksi vahvuuksista on laaja kielivalikoima, jolla sitä voi ohjelmoida. Unityn skriptejä voi kirjoittaa C#-, UnityScript- tai BooScript-kielillä (Xie 2012). UnityScript muistuttaa hyvin vahvasti JavaScriptiä staattisilla tyyppimäärityksillä ja BooScript on hyvin samankaltaista kuin *Python*-ohjelmointikieli. Unity projektin pystyy kääntämään Windowsille, Macille, Xbox 360:lle, PlayStation 3:lle, Wiille, iPadille, iPhonelle ja Androididille (Xie 2012). Kuten Xie (2012) huomauttaa, pelejä voidaan ajaa myös selaimessa asentamalla selaimeen lisäosan, mutta nykyään myös suoraan WebGL:n avulla.[1] Unitylle on myös lisätty

1. WebGL on teknologia, jonka avulla pystytään JavaScriptillä piirtämään 3D- tai 2D-grafiikkaa HTML5-ympäristössä tavalla, joka muistuttaa hyvin läheisesti OpenGL-rajapinnan käyttöä. Lisää tietoa esim. osoitteessa `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`.

Nintendo Switch-tuki.[2]

Unity käyttää arkkitehtuurinaan komponenttimallia (Xie 2012) eli skriptit lisätään pelioliolle komponentteina ja niitä voidaan lisätä, poistaa, sammuttaa tai käynnistää uudelleen pelin ollessa käynnissä. Unityssä myös käytetään *Prefabeja* (Xie 2012). Nämä ovat jo aikaisemmin suunnitteltuja pelioliokokonaisuuksia, joita voidaan käyttää uudestaan. Prefabeille voidaan asettaa valmiiksi tietyt oletusarvot, mitkä ovat yleisimpiä peliolion tapauksessa, mutta yksittäistä oliota voidaan vielä muokata kenttään lisäämisen jälkeen, tehden yhdestä vihollisesta esimerkiksi nopeamman kun muista.

Unityssa on luotu valmiiksi joitain tapahtumia, joita kutsutaan jokaisesta skriptistä automaattisesti, esimerkiksi `Start()`- ja `Awake()`-aliohjelmat (Xie 2012). Esimerkiksi peliolion luomisen yhteydessä kutsutaan sen `Awake()`-metodia.

3.2 Käyttäjälähtöinen muokattavuus Unityssä

Unitylla tehtyjä projekteja on helppoa muokata ja käsitellä, mutta vain pelinkehittäjän näkökulmasta. Mikäli muokkaajalla ei ole projektia itsellään, ei hän pysty muokkaamaan Unityn projektia kovinkaan hyvin. Unity sallii ulkopuolisten *AssetBundle*-pakettien lataamisen (Unity Technologies 2017a), mutta tämäkin vaatisi muokkaajaa asentamaan Unityn ja tutustumaan AssetBundlen rakenteeseen siinä missä Unityynkin.

Suurin osa Unityn käyttämistä resursseista pakataan binääritiedostoihin ja tiedostojen sisältöä ei voi muokata purkamatta niitä. Unity sallii ajonaikaiseen lataamisen, mutta vain Resources-kansiosta (Unity Technologies 2017b). Unity myös sisältää eheystarkistuksen, joka luokittelee pelin rikkinäiseksi, jos Resources-kansion tiedostoja muokkaa. Näistä syistä Unity ei suoraan tue erityisen hyvin käyttäjälähtöistä muokattavuutta, ja nämä ongelmat ratkaistaan tässä tutkimuksessa tehdyllä kirjastolla.

2. ks. esimerkiksi artikkeli `https://blogs.unity3d.com/2017/03/03/unity-devs-shine-on-switch/`.

Muokattavuuden sallimisessa käyttäjälle on myös tärkeää pelkän ulkoasun lisäksi sallia käyttäjälle mahdollisuus muokata toiminnallisuutta. Tämä onnistuu yleensä käyttämällä jotain tulkattua kieltä pelin ohjelmakoodin lisäksi, missä määritellään pelin tapahtumat ja säännöt. Unityssä voi esimerkiksi ajaa C#-ohjelmakoodia ajonaikaisesti ja täten parantaa komentojen ja toiminnallisuuden muokattavuutta. [3] Näin esimerkiksi toimittiin Cities Skylines-pelissä. Myös Luaa voidaan käyttää Unityn kanssa esimerkiksiki MoonSharp-kirjastolla.[4]

3.3 Esimerkkejä käyttäjälähtöisen muokattavuuden sallimisesta peleissä

Erilaisten ratkaisujen kirjo peleissä käytetyissä muokattavuuden sallivissa ratkaisuissa on valtava. Suuri osa peleistä toteuttaa täysin oman rajapintansa muokattavuuden sallimiseksi ja saattaa käyttää omaa skriptauskieltään. Vaikka on ymmärrettävää, että projektin toteuttava ryhmä haluaa käyttää heille mieluisia työkaluja, olisi käyttäjien kannalta helpompaa, jos yhtäläisyyksiä tai yleisiä käytäntöjä löytyisi enemmän käyttäjälähtöisen muokattavuuden sallimiseksi. Yksi suosittu vaihtoehto tuntuu olevan Lua-ohjelmointikieli. Sitä on käytetty esimerkiksi Civilizations-sarjan uusimmissa peleissä ja World of Warcraft (Nardi 2010; Taylor 2009) ja Elder Scrolls Online -pelien lisätyökalujen ja käyttöliittymän muotoilussa (Scacchi 2011b).

Laukkanen (2005) on tehnyt graduaan varten kolme tapaustutkimusta kolmen eri pelin modauksesta ja niiden ympärillä olevista yhteisöistä. Varsinkin tekniseltä kannalta tehtyä tutkimusta käyttäjälähtöisen muokattavuuden sallimisesta on tehty suhteellisen vähän. Hän käsittelee tutkimuksessaan *Half-Lifea*, *The Simsiä* ja *Grand Theft Auto (GTA) III:sta ja Vice Cityä*. Esittelen seuraavaksi Laukkasen tutkimuksia ja niistä esille nousseita tapoja käyttäjälähtöisen muokattavuuden sallimiseksi peleissä.

Half-Life-peli käytti pohjanaan Quake-pelin moottoria, mikä tarkoitti, että Half-Li-

3. Tähän löytyy esimerkiksi kirjastot CS-Script for Unity: `https://www.assetstore.unity3d.com/en/#!/content/23510` ja UCompile: `https://www.assetstore.unity3d.com/en/#!/content/66267`

4. Saatavilla osoitteesta `http://www.moonsharp.org/`

fe-peliä pystyi muokkaamaan yhtä kattavasti kuin Quakea (Laukkanen 2005, s.61). Quaken modaajayhteisö oli jo oppinut muokkaamaan pelimoottoria, joten Half-Lifen muokkaaminen onnistui heiltä hyvin (Laukkanen 2005, s.61). Laukkanen (2005, s.50-61) esittää, että Half-Lifea voi muokata joko tekemällä skriptejä, joilla voidaan kutsua funktioita itse pelistä, tai muokkaamalla C++-lähdekoodia. Hän (s.62) pohtiikin tämän saattaneen olla Half-Lifen suurin vahvuus: sitä pystyi muokkaamaan "täydellä" ohjelmointikielellä kokonaisuudessaan. Hän jatkaa skriptejä pystyttävän muokkaamaan vielä jälkeenpäin käännetyissä modiprojekteissa, mutta C++ lähdekoodia ei, sillä se pakataan DLL-tiedostoihin.

Half-Life näytti tarjoavan siis koko pelin modattavaksi käyttäjille, eikä vain kattavaa rajapintaa muokkaamiselle. Näin kattava pelin lähdekoodien ja assettien jakaminen muokattavaksi tuntuu olevan vain harvojen yritysten tapa sallia muokattavuus. Half-Lifen pohjalta on tehty useita kattavia uusia pelejä eli ns. *täysiä pelimuunnoksia*, joista Laukkanen (2005, s.54-59) esittelee 15 erilaista. Näistä yksi suosituimmista lieneekin aiemminkin käsitelty Counter-Strike.

Laukkanen (2005, s.74-86) esittelee, kuinka The Sims -pelissä on varsin laajasti pyritty sallimaan käyttäjälähtöinen muokattavuus yksittäisissä kokonaisuuksissa, varsinkin hahmojen ulkonäön osalta. Esimerkiksi kasvojen muotoilulle ja kuvioittamiselle (*skinning*) on useita virallisia työkaluja tarjolla ja Laukkasen mukaan löytyy myös muutama epävirallinen. The Sims -pelissä muokkauksen sallivia työkaluja on helppoa käyttää, jotta kaikki voisivat kokeilla tehdä muokkauksia (Laukkanen 2005, s.99). Tämä on kuitenkin toteutettu modien monimuotoisuuden kustannuksella (Laukkanen 2005, s.99). The Sims ei salli kovinkaan syvällisiä muutoksia itse peliin, sen koodiin tai rakenteeseen. Laukkanen toteaa pelistä puuttuvan mahdollisuus luoda tai muokata esimerkiksi hahmojen malleja, animaatioita, hahmojen kanssakäymisiä, ääniefektejä tai jopa esineitä (*objects*). Hän (s.99) tuokin esille fanien pohdintaa, että onko Maxim- tai EA-yhtiöt jopa tarkoituksella rajoittaneet modausta, jotta yhtiöt voisivat myydä itse lisää sisältöä peliin.

Laukkanen (2005, s.109) pohtii GTA:n kehittäjien antavan vähemmän näkyvää tukea käyttäjälähtöiselle muokattavuudelle ohjeiden ja vastauksien muodossa kuin kah-

dessa muussa hänen esimerkissään. Hän (s.109) jatkaa GTA:n sallivan muokkauksen jotakuinkin kahden muun esimerkin välimaastossa: GTA:ssa useimmat modit ovat yksittäisiä muutoksia esimerkiksi skineihin, pelaajan malliin, ajoneuvoihin, aseisiin, tekstuureihin, karttoihin tai koodiin. Laukkanen (2005, s.116) esittelee, kuinka GTA:ssa voi ladata uudenlaisen skinin hahmolle, ja samalla toteaa sen olevan erittäin yksinkertaista: pelaajan täytyy värjätä uudelleen yksi, valmiiksi muodostettu BMP-tiedosto. GTA:ssa pelin skriptaus toteutetaan omalla skriptikielellä, eikä GTA:n lähdekoodeja pääse muokkaamaan (Laukkanen 2005, s.124). Hän toteaa täysien pelimuunnoksien olevan harvinaisempia tässä pelissä. Modaaja pystyy muuttamaan pelaajan ominaisuuksia tai esimerkiksi määrittelemään uudestaan tehtävien rakennetta (Laukkanen 2005, s.124). Laukkanen toteaa kuitenkin modaajien kehittäneen mielenkiintoisia modeja peliin, jotka esimerkiksi esittävät auton nopeuden tai sen vahingon määrän. Laukkanen argumentoikin tämän olevan mahdollista, sillä GTA tarjoaa jo valmiiksi erittäin kattavasti tietoa modirajapinnoilleen.

Vaikuttaa siltä, että monen pelaajan verkkoroolipeleissä tuntuu olevan yleistä tarjota käyttäjille mahdollisuus muokata käyttöliittymää. Taylor (2009) kuvailee modien toimintaa World of Warcraft -verkkomoninpelissä. Hän toteaa modien tekevän paljon muutakin kuin vain muokkaavan käyttöliittymää. Hän tarkemmin kuvailee modien sallivan toimintojen automatisointia, auttavan pelaamisen tarkkailussa, antavan tärkeää informaatiota ja ylipäätänsä helpottavan laajaa kirjoa yksinkertaisia ja monimutkaisia toimintoja. Taylor (2009) kuvailee vielä tarkemmin *CTRaidAssist*-modin toimintaa pelissä. Tämä modi esittää pelaajille automaattisesti tärkeää tilannetietoa pelistä liittyen vaikeisiin tilanteisiin. Tapahtuman huomanneen pejaajan pitäisi kirjoittaa käsin pelin keskusteluikkunaan sama tieto, mikäli modi ei olisi käytössä. Myös Elder Scrolls Online -verkkomoninpelissä voi muotoilla käyttöliittymää mieluisekseen Lua-skripteillä. Elder Scrolls Online -peli määrittää omat rajapintansa, minkä avulla pelin kanssa voi keskustella. Peli käyttää myös indeksitiedostoa, johon määritellään kaikki modin sisältämät tiedostot.

Seuraavaksi esittelen kokemuksiani käyttäjälähtöisen muokattavuuden sallimisesta peleissä. *Kerbal Space Program* -pelin modaukseen löytyy kattavat ohjeet modin luo-

misesta (*Wiki - Kerbal Space Program, Making an asset from start to finish* 2017). Huomionarvoista tässä ratkaisussa on se, että ohjeiden kohdassa 5 pyydetään asentamaan Unity. Tämä valivaihe voi olla iso kynnys käyttäjälle, joka ei ole aiemmin perehtynyt pelien kehitykseen. Tämä myös pakottaa modien tekijän oppimaan käyttämään Unitya, jotta hän pystyy tekemään uuden modin peliin, vaikka tekijä olisi muuten kokenut modien tekijä.

Darkest Dungeon -peli hoitaa muokattavuuden käyttäjälle lähestyttävästi. Puhtaalta pöydältä uuden modin tekeminen kyseiseen peliin olisi vaikeaa, mutta yhdenkin valmiin modin katsominen auttaa hyvin alkuun. Esimerkiksi uuden hahmotyypin luominen peliin vaatii neljä kansiota, joista kaksi voi hyvinkin aluksi pitää täysin samanlaisena jonkun valmiin hahmon kanssa - ne eivät poikkea valmiidenkaan hahmojen kesken merkittävästi. Uudelle hahmolle voi tehdä uudet kuvakkeet käytössä oleviin kykyihin, mutta tässäkin voi tyytyä valmiisiin kuvakkeisiin ja muokata niitä pikkuhiljaa. Oman modin tekemistä pääsee siten testaamaan nopeasti.

Hahmon ominaisuuksien ja kykyjen muokkaus onnistuu muuttamalla yhtä selkokielistä tekstitiedostoa. Darkest Dungeon-peli toteuttaa omanlaisensa tietojen tallennustekstiformaatin, jossa asioiden ominaisuuksia lisätään ja muokataan pisteen avulla, esimerkiksi:

```
'combat_skill: .id "battlefield_medicine" .level 0 .heal 1
1

   .launch 43 .target @~1234 .effect "Cure" "Cureself"
```

Kyseisellä rivillä määritellään Plague Doctor-pelihahmolle yksittäinen kyky. Rivillä tarkemmin käsitellään `battlefield_medicine`-kyvyn tietoja tasolla 0. Tämän lisäksi pelissä on valmiita efektejä, jota voi kyvyissä käyttää, mutta uusien kykyjen efektienkin luominen on yksinkertaista. Pelin lokalisaatio on tehty .xml-tiedostoon, jonne kirjoitetaan kaikilla kielillä yhteen hahmotyyppiin tarvittavat merkkijonot. Jotkut muut ominaisuudet, esimerkiksi kykyjen parantamiseen käytetyn kullan määrä, on määritelty JSON-tiedostoissa.

Grim Dawn -toimintaroolipelissä muokattavuus on toteutettu mielestäni käyttäjiä luotaan pois työntävästi.[5] Käyttäjän tarvitsee avata monen gigatavun kokoinen, pakattu datatiedosto, jonka jälkeen hän voi vasta aloittaa työskentelyn. Datan käsittelyä varten kannattaa myös katsoa jokunen esimerkkivideo pelin tarjoajilta. Vaikka peli vaikuttaa vaikeammalta lähestyä muokkauksen kannalta, on peliin tullut valtavat määrät modeja ja yhdistelmämodeja, jotka yhdistävät toisten tekemiä modeja yhdeksi paketiksi.

Onkin tärkeää ottaa huomioon, että pelaaja tuskin käyttää vain yhtä modia, vaan usein modeja saattaa olla käytössä jopa kymmeniä. Tästä johtuen muokattavuus tulee sallia tavalla, joka ei estä tai tee usean modin käyttämistä samaan aikaan hankalaksi. Darkest Dungeon-pelin muokattavuudessa onkin ongelmana tällä hetkellä, että jokaisen modin luomat uudet efektit pitää manuaalisesti käydä lisäämässä Effects-tiedostoon. Tähän tullee jonkunlainen helpompi ratkaisu, sillä peli on vielä Early Access -tilassa, eli ei täysin julkaistu. Modaajat ovatkin pyrkineet uusimmissa päivityksissä välttämään kokonaan uusien efektien luomista kykyihin ja pitäytyneet valmiiksi luotujen efektien käytössä.

Edellä on esitelty hyvin monenlaisia erilaisia tapoja lähestyä käyttäjälähtöistä muokattavuutta. Lähestymistavat vaihtelevat pelien välillä, kuinka paljon peliä saa tai pystyy muokata. Tämän lähestymistavan valinta on varsinkin peliä toteuttavan henkilön tai henkilöiden vastuulla päättää. Tärkeitä ominaisuuksia edellä käsitellyissä esimerkeissä ovat mielestäni helppo lähestyttävyys sekä työkalujen että vaadittavan kokemuksen näkökulmasta, samaan aikaan kuitenkin kattavan muokkausmahdollisuuden tarjoaminen ja usean modin salliminen kerralla samanaikaisesti. Quaken modaajayhteisön kyky muokata Half-Lifea saman tien tuo esille yhtenäisten käytänteiden tärkeyttä käyttäjälähtöisessä muokattavuudessa: käyttäjien on helpompi muokata tutuilla työkaluilla täysin uuden toteutuksen sijaan.

5. Pelin sivut osoitteessa `http://www.grimdawn.com/`

4 Tutkimus

Tässä luvussa käsittelen tarkemmin itse tutkimusta. Marchin ja Smithin (1995) mukaan informaatioteknologian tutkimusta voidaan lähestyä luonnontieteen tai suunnittelutieteen näkökulmasta. Tässä tutkimuksessa näkökulma on jälkimmäinen. Aluksi esittelen suunnittelutieteen teoriapohjaa ja artefaktien arviointia, minkä jälkeen jatkan siitä, kuinka nämä asiat ilmenevät omassa tutkimuksessani.

4.1 Suunnittelutiede

Suunnittelutieteellisessä tutkimuksessa, tai lyhyemmin suunnitelututkimuksessa, tuotetaan artefakti (Hevner ym. 2004). Tämä artefakti on konstruktio, malli, metodi ja/tai fyysinen ilmentymä (March ja Smith 1995). Artefaktia arvioidaan eri tavoin ja se pyrkii ratkaisemaan jonkun ongelman (Hevner ym. 2004; March ja Smith 1995). March ja Smith (1995) jaottelevat informaatioteknologian tutkimuksen kuvailevaan ja ohjailevaan tutkimukseen, joista jälkimmäinen vastaa heidän mukaan suunnitelututkimusta. Ohjaileva tutkimus pyrkii parantamaan informaatioteknologian tehokkuutta. Hevner ym. (2004) määrittelevät suunnittelun olevan sekä prosessi että tuote. Tarkemmin suunnitteluprosessi on kokoelma asiantuntijan toimintoja, jotka tuottavat innovatiivisen tuotteen, eli suunniteluartefaktin. Tämän artefaktin arvioinnista taas saadaan palautetta ja parempi ymmärys itse ongelmasta, mikä taas sallii sekä tuotteen että prosessin laadun parantamisen. Heidän näkemyksensä suunnittelutieteestä vastaa hyvin läheisesti Zelkowitzin ja Wallacen (1998) määrittelemää *tekniikan alan metodia* (*engineering method*), jossa insinöörit kehittävät ja testaavat ratkaisua hypoteesiin. Testin tuloksen perusteella insinöörit parantavat ratkaisua, kunnes ratkaisua ei tarvitse enää parantaa lisää.

Hevner ym. (2004) korostavat suunnitelututkimuksen olevan *etsintäprosessi*, jossa suunnittelua iteroidaan, kunnes ongelmaan löydetään toimiva ratkaisu. He määrittelevät keinojen, päämäärien ja lakien abstraktion ja esittämisen olevan tärkeää suunnitelututkimuksessa. He määrittelevät keinojen olevan käytettävissä ole-

vat toiminnot ja resurssit, joilla pystytään rakentamaan ratkaisu. Päämäärät esittävät ratkaisun tavoitteita ja rajoitteita. Lait puolestaan ovat hallitsemattomia voimia ympäristössä.

Hevner ym. (2004) käyvät läpi esimerkkiä, missä etsittäisiin "täydellinen" ratkaisu suunnittelututkimuksessa. Tämän ratkaisun ongelmana on, että informaatiojärjestelmän hallinnoijan pitäisi listata kaikki mahdolliset infrastruktuurit, käydä läpi niiden kaikkien käytännöllisyys ja rajoitteet ja määritellä kaikki maksu- ja hyötyvakiot. Kaikkien ratkaisujen läpikäymisestä syntyy valtava työmäärä. He toteavatkin näiden kaikkien ratkaisujen läpikäymisen olevan usein käytännössä mahdotonta, minkä takia etsintä tähtääkin vain tyydyttävään ratkaisuun. Heidän mukaan on tärkeää saada aluksi ratkaisu, jotta voidaan todeta, että ongelma voidaan ratkaista ja vasta myöhemmin keskittyä kysymykseen, miksi se toimii. Hevner ym. (2004) pohtivatkin, miten mitataan suunnitteluratkaisun *hyvyys*. He antavat kaksi esimerkkiä tähän: joko todistetaan tai näytetään, että ratkaisu on aina lähellä optimaalista raktaisua, tai vertaillaan omaa ratkaisua toisten tuottamiin ratkaisuihin samanlaisissa ongelmanratkaisutilanteissa.

Hevnerin ja muiden (2004) mukaan käyttäytymistieteet ja suunnittelutiede ovat toisiaan täydentäviä, erillisiä paradigmoja. Heidän näkökulmansa suunnittelutieteestä on hyvin käytännönläheinen: suunnittelutieteessä pyritään etsimään "mikä on tehokasta", kun taas käyttäytymistieteissä etsitään "mikä on totta". He korostavat tutkimuksen tulevan tähdätä liiketoiminnan kannalta järkeviin ongelmiin, mikä korostaa myös käytännönläheisyyttä. Luonnontieteissä, joihin Hevner ym. (2004) laskevat käyttäytymistieteet mukaan, etsitään esitettävissä olevaa muotoa todellisuuden olemuksesta, joka voidaan kerätä, selittää ja kirjoittaa ylös lauseiksi (Hevner ym. 2004; March ja Smith 1995). Suunnittelutiede pyrkii keksimään artefakteja saavuttaakseen tavoitteensa (March ja Smith 1995). Hevner ym. (2004) kehottavat tekemään tutkimuksen niin, että sen voi ymmärtää johdon työntekijät.

Marchin ja Smithin (1995) mukaan suunnittelutieteessä keskitytään artefaktien tuottamiseen tieteellisen teorian luomisen sijaan: siinä tuotetaan konstruktioita, malleja, metodeja ja/tai ilmentymiä, jotka kaikki voivat olla suunnittelutieteellisiä artefak-

teja (March ja Smith 1995). March ja Smith (1995) määrittelevät konstruktioiden olevan käsitteitä, joilla kuvaillaan ilmiöitä. Malleja käytetään kuvailemaan toimintoja, tilanteita tai artefakteja. Metodit ovat tapoja toteuttaa päämäärätavoitteisia aktiviteetteja. Edellä mainitut voidaan toteuttaa tuotteiksi, fyysisiksi ilmentymiksi, jotka on tarkoitettu tekemään jonkunlaisia toimintoja.

4.2 Artefaktien arviointi

Artefaktien arviointi on olennaista suunnittelututkimuksen onnistumisen kannalta (Hevner ym. 2004; March ja Smith 1995). Hevner ym. (2004) ehdottavat artefaktin arviointia esimerkiksi toiminnallisuuden, valmiuden, johdonmukaisuuden, tarkkuuden, toimintakyvyn, luotettavuuden, käytettävyyden, organisaatiollisen istuvuuden (*organizational fit*) tai muiden olennaisten laadullisten ominaisuuksien mukaan. Mikäli mitattava suure pystytään kuvaamaan matemaattisesti, on sen arviointi huomattavasti helpompaa (Zelkowitz ja Wallace 1998; Hevner ym. 2004). March ja Smith (1995) huomauttavat arvioinnin olevan vaikeaa, sillä artefaktin tehokkuus riippuu sen käyttätarkoituksesta ja käyttötarkoitus saattaa kattaa lavean alan erilaisia tehtäviä. Hevner ym. (2004) jaottelevat arviointitavat viiteen tyyliin ja niiden omiin alalajeihin:

- **havainnollinen**, joka jakautuu tarkemmin *tapaustutkimukseen* ja *kenttätutkimukseen*
- **analyyttinen**, joka jakautuu *staattiseeen*, *arkkitehtuurilliseen* ja *dynaamiseen analyysiin* ja myös *optimointiin*
- **kokeellinen**, joka jakautuu *kontrolloituun kokeeseen* ja *simulointiin*
- **testauksellinen**, joka jakautuu *funktionaaliseen* eli ns. mustan laatikon testaukseen ja *rakenteelliseen* eli ns. valkoisen laatikon testaukseen
- **kuvaileva**, joka jakautuu *argumenttiin* ja *skenarioon*

Valitun arviointitavan tulee sopia suunniteltuun artefaktiin ja esimerkiksi kuvailevaa arviointityyliä tulisi käyttää vain erityisen innovatiivisille artefakteille, joihin muut arviointityylit eivät sovellu (Hevner ym. 2004; Zelkowitz ja Wallace 1998). Zel-

kowitz ja Wallace (1998) korostavat, että liian usein tutkimuksessa tutkijat pyrkivät osoittamaan tuotteen toiminnallisuuden itse yksinkertaisilla testeillä, joista ei kerätä joko ollenkaan tai kerätään hyvin vähäisesti dataa. Tilanne saattaa olla muuttunut melkein kahdessakymmenessä vuodessa, mutta Zelkowitzin ja Wallacen (1998) huomio on mielestäni edelleen relevantti. Esimerkiksi Peffers ym. (2012) olivat löytäneet artikkelin, jossa ei heidän mukaan ollut minkäänlaista arviointia suunnittelutieteellisessä tutkimuksessa. Omille töilleen sokaistuu, jolloin niitä ei osaa arvioida samoin kuin ulkopuolinen arvioisi. Ulkopuolisen tekemä arvio on oletettavasti objektiivisempi.

Hevner ym. (2004) esittävät erilaisia, hyväksyttäviä arviointitapoja, mutta he eivät kuitenkaan selitä kovin tarkasti, miten erilaisia arviointitapoja tulisi käyttää. Peffers ym. (2012) käsittelevät tarkemmin erilaisia tapoja arviointia varten suunnittelutieteessä ja miten niitä on käytetty tutkimuksessa. Peffersin ja muiden (2012) tekemässä tutkimuksessa tekniset kokeet, tapaustutkimukset ja havainnolliset skenaariot olivat kaikista yleisimmin käytettyjä arviointitapoja.[1]

Peffers ym. (2012) määrittelevät esimerkiksi prototyyppien olevan toteutettuja artefakteja, jotka osoittavat artefaktien käytännöllisyyden. He määrittelevät havainnollistavissa skenarioissa artefaktia käytettävän synteettisessä tai oikean maailman tilanteessa sen hyödyllisyyden osoittamiseksi. He myös määrittelevät tapaustutkimuksien käyttävän artefaktia todellisen maailman tilanteessa, ei vain artefaktin hyödyllisyyden osoittamiseksi, mutta myös havainnollistaakseen sen vaikutusta ympäristöön. Hevner ym. (2004) kuvailevat tapaustutkimuksen tarkastelevan artefaktia syvällisesti sen käyttötarkoitusympäristössä. Zelkowitz ja Wallace (1998) kuvailevat tapaustutkimusta tarkemmin: tapaustutkimuksessa tutkijat keräävät dataa jostain projektista ajan kuluessa. Tätä dataa käytetään jonkun ominaisuuden mittaamiseen.

Tapaustutkimusta voidaan käyttää varsinkin *miten* ja *miksi* kysymyksiin vastaamiseen (Yin 2014, s.9). Yin (2014) esittelee useita erilaisia tapoja, joilla tapaustutkimuksessa voi kerätä materiaalia: asiaan liittyvä dokumentaatio, arkistolliset lähteet,

1. Tarkempia tuloksia varten katso Peffers ym. (2012) taulukko 4.

haastattelut, suorat havainnot, osallistuvat havainnot ja fyysiset artefaktit (Yin 2014, s. 106). Yin tuntuu jaottelevan työstä tehdyt raportit dokumentoinniksi, mutta esimerkiksi kyselylomakkeet haastatteluiden alle.

Tapaustutkimuksessa on tärkeää pitää erillään kerätty data ja tutkijan raportti sekä löydökset dataan liittyen (Yin 2014, s.123). Yin (2014) ehdottaa käyttämään esimerkiksi erillisiä tiedostoja itse tutkimusdatalle ja tutkimukselle.Kun data on arkistoitu mielekkäästi, voivat muut henkilöt perehtyä dataan ilman tutkimuksen läpikäymistä, mikä lisää tutkimuksen reliabiliteettia (Yin 2014, s.124). Tutkimuksen esityksen tulisi muodostaa "todistusketju", jonka avulla edetään tutkimuskysymyksistä vastauksiin asti (Yin 2014, s.126). Tätä ketjua pitäisi pystyä menemään alusta loppuun ja lopusta alkuun.

Yin (2014, s.133) toteaa tapaustutkimuksiin liittyen olevan vielä hyvin vähän valmiita paletteja, miten analysoida aineistoa. Hän kuitenkin esittää, miten hän aloittaa usein tapaustutkimuksien käsittelyn (s.134): Hän lähtee liikkeelle kysymyksistä datan sijaan. Hän tarkastelee, mikä data vastaa yhteen, yksinkertaiseen kysymykseen ja tekee perustellun päätelmän tämän perusteella. Tämän jälkeen hän jatkaa seuraavaan, isompaan kysymykseen. Tätä jatketaan, kunnes päätutkimuskysymyksiin on vastattu. Hän (s.135) ehdottaakin "leikkimään" datan kanssa: käymään dataa lävitse, katsomaan sitä eri tavoin ja tutkimaan sitä eri näkökulmista. Hän myös neuvoo kirjoittamaan ylös muistiinpanoja ja ajatuksia, mitä ajatuksia datasta on herännyt.

Yin (2014, s.136-142) esittelee neljä erilaista tapaa lähteä liikkeelle datan anyloisimiseen. Yksi näistä lähestymistavoista on käydä koko data lävitse "alhaalta ylöspäin" ja lähteä muodostamaan yksinkertaisia ajatuksia aiemman "leikin" perusteella (Yin 2014, s.136). Tästä leikistä on voinut huomata yksittäisen tai useamman ajatuksen, josta voi lähteä muodostamaan analyysia (s.137). Yin kuitenkin huomauttaa tämän saattavan olla hankalaa aloittelijoille, sillä he eivät välttämättä huomaa niin herkästi datasta löytyviä yhteyksiä. Huomionarvoista on, ettei Yin erikseen ole maininnut tapaustutkimusten käytöstä arvioinnissa esimerkiksi tietotekniikassa tai muuten sovellusten arvioinnissa, vaikka kirjan painovuonna 2014 on ollut jo tutkimuksia, joissa on käytetty tapaustutkimuksia arvoimista varten (esim. Peffers ym. (2012) löysi-

vät 7 tapaustutkimusta, joita käytettiin sovellusten arvioinnissa).

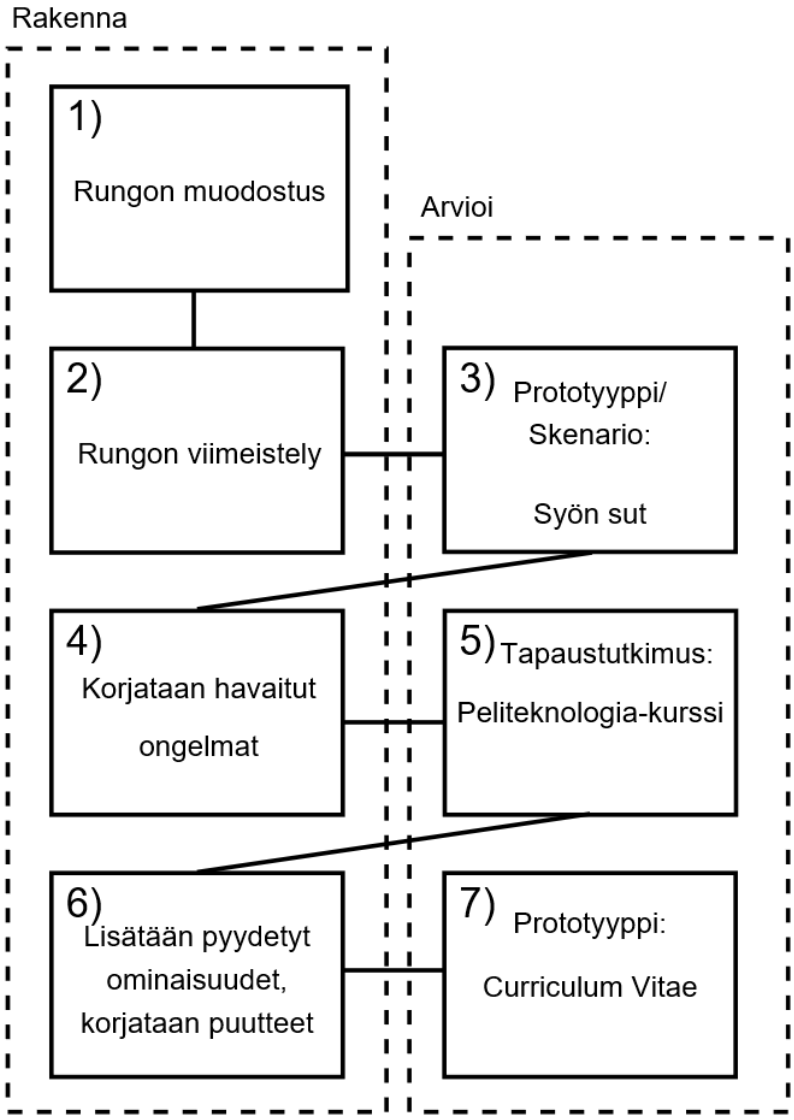
4.3 Suunnittelutiede tässä tutkimuksessa

Tässä tutkimuksessa toteutan suunnittelututkimusartefaktin. Artefakti on kirjasto, fyysinen toteutus, joka auttaa kehittäjiä helpommin sallimaan käyttäjälähtöisen muokattavuuden Unity-pelimoottorilla tehdyissä peleissä. Kuvio 1 esittelee kirjaston toteutukseen liittyvät vaiheet. Vasemmalla puolella olevat vaiheet kuvastavat itse kirjaston kehittämiseen liittyviä toimenpiteitä ja oikealla puolella olevat vaiheet kirjaston arviointiin liittyviä toimenpiteitä. Vaiheet 1, 2 ja 3 muodostavat ensimmäisen syklin, vaiheet 4 ja 5 toisen sekä vaiheet 6 ja 7 kolmannen ja myös viimeisen syklin. Sykli alkaa rakennusvaiheella tai vaiheilla ja päättyy sykliä koskevaan arviointiosuuteen. Seuraava sykli parantaa edellisen syklin ongelmakohtia ja siten koko kirjaston toiminnallisuutta.

Kirjastolle on asetettu viisi erillistä laatuvaatimusta: *omaksuttavuus*, *käytettävyys*, *suorituskyky*, *turvallisuus* ja *yleistettävyys*. Omaksuttavuudella tarkoitan tämän tutkimuksen puitteissa sitä, kuinka nopeaa kirjastoa on ulkopuolisen henkilön oppia käyttämään eli kuinka hyvin ulkopuolinen omaksuu kirjaston käytön. Käytettävyydellä taas tarkoitan kehittäjän kykyä käyttää kirjastoa sille tarkoitetussa käyttötarkoituksessa: onko kehittäjällä ongelmia saada haluttu toiminnallisuus toteutettua. Suorituskyky laatuvaatimuksena merkitsee, että pelin suoritusnopeus ei saa hidastua liikaa kirjaston käytön takia. Turvallisuudella tarkoitan sitä, kuinka turvallista pelaajan on käyttää modeja kirjastolla tehdyssä pelissä: aiheuttavatko modit tietoturvariskin. Yleistettävyydellä tarkoitan sitä, kuinka helppoa kehittäjän näkökulmasta kirjaston toiminnallisuutta on muokata, mikäli yksittäinen osakokonaisuus pitäisi vaihtaa toisenlaiseksi.

4.3.1 Kirjaston rakentamisvaiheet

Rakentamisvaiheet ovat kuvion 1 vaiheet 1, 2, 4 ja 6. Näissä vaiheissa lisään ja korjaan kirjastoon haluttuja ominaisuuksia, poistan ongelmakohtia ja lisään tarvittavia



Kuvio 1. Tutkimuksen eri vaiheet kuvattuna järjestyksessä

toimintoja. Vaiheissa *Rungon muodostus* (Vaihe 1) ja *Rungon viimeistely* (Vaihe 2) tapahtuu kirjaston pohjan rakentaminen ja suurin osa ohjelmakoodin tuottamisesta. Vaiheessa *Korjataan havaitut ongelmat* (Vaihe 4) korjaan vaiheen *Esimerkkipeli Syön sut* (vaihe 3) aikana löydetyt virheet kirjastosta ja valmistelen kirjaston tapaustutkimusta varten (Vaihe 5). Vaiheessa 6 korjataan tapaustutkimuksessa havaitut ongelmakohdat kirjastosta ja lisätään hyväksi havaitut ominaisuudet kirjastoon.

Rakentamisvaiheissa esittelen vaiheen aikana tehtyjä päätöksiä kirjaston toiminnal-

lisuuksien toteuttamiseen liittyen. Pyrin olemaan mahdollisimman *täsmällinen* ongelmakohtien ja ratkaisujen esittelyssä tässä vaiheessa, jotta tutkimus olisi toistettavissa muiden tutkijoiden toimesta. Esittelen myös pohdintaa ratkaisuihin liittyen. *Turvallisuuden* otan huomioon arkkitehtuurissa rajoittamalla skriptien pääsyä koneen tiedostoihin käsiksi. *Yleistettävyyden* otan huomioon kehittämällä arkkitehtuurista sellainen, että yksittäisiä ratkaisuja voidaan helposti vaihtaa jälkikäteen kirjastossa.

4.3.2 Kirjaston arviointi

Kirjaston arviointi toteutetaan skenariolla, prototyypeillä ja tapaustutkimuksella. Kuviossa 1 vaiheet 3, 5 ja 7 ovat arviointiin liittyvät vaiheet. Ensimmäinen peli toimii skenariona ja prototyyppinä, kun toinen esimerkkipeli on enemmän prototyyppi. Peffers ym. (2012) määrittelevät havainnollistavien skenarioiden olevan tilanteita, joissa käytetään artefaktia synteettisessä tai tosimaailman tilanteessa artefaktin soveltuvuuden tai käytännöllisyyden osoittamiseksi.[2] Mielestäni varsinkin ensimmäinen esimerkkipeli sopii tähän määritelmään hyvin. Ensimmäiseen esimerkkipeliin tehdään kattavat dokumentaatiot, jotta kirjaston käyttäjät näkisivät, kuinka kirjastoa käytetään. Sitä käytetään esimerkkinä tapaustutkimuksessa ja se esittelee kirjaston käyttöä. Hevner ym. (2004) luokittelevat skenariot kuvailevaksi arvioinniksi, jolloin tapaustutkimuksen käyttö niiden lisäksi tuo tutkimukselle lisää objektiivisuutta. Peffers ym. (2012) myös toteavat tapaustutkimuksen tarjoavan vahvempaa todistusaineistoa tarkasteltavan asian tehokkuudesta tai suorituskyvystä kuin argumentit tai havainnollistavat skenariot. Tapaustutkimusta taas ei pystytä toistamaan tämän tutkimuksen puitteissa useita kertoja, jolloin esimerkkipelit tarjoavat useamman arviointikierroksen kirjastolle, mikä on tärkeä osa suunnittelutieteellistä tutkimusta (Hevner ym. 2004). Tapaustutkimuksessa kirjasto ja esimerkkipeli annetaan Peliteknologia-kurssilaisille muokattavaksi ja testattavaksi.

Tutkimuksen tavoite on tehdä käyttäjälähtöisen muokattavuuden käyttöönotto Uni-

2. "Application of an artifact to a synthetic or real-world situation aimed at illustrating suitability or utility of the artifact." (Peffers ym. 2012, s.5)

ty-pelimoottorissa mahdolliseksi kirjastolla, jolloin yksi kuvaava suure on kehittäjien käyttämä aika pelin kehittämiseen. Kuitenkin käytetty aika riippuu merkittävästi siitä, millaisen pelin kirjaston avulla tekee ja kuinka tuttuja esimerkkiratkaisussa käytetyt teknologiat ovat. Projektien kehittämisajat ovat yleensä kuukausia, jopa vuosia, joten näin pitkäaikaiseen tarkkailuun ei tämän tutkimuksen puitteissa kyetä. Tutkimuksessa tehdyssä arvioinnissa keskitytäänkin ajan sijaan omaksuttavuuden ja käytettävyyden tarkkailuun. Näitä ominaisuuksia pyritään ylläpitämään mahdollisimman hyvin, jotta ohjelmoijan olisi mahdollisimman yksinkertaista ja nopeaa oppia käyttämään ja käyttää tuotettua kirjastoa. Muitakin tärkeitä laadullisia vaatimuksia kirjastolle on asetettu. Esimerkiksi suorituskyky otetaan kehittämisessä huomioon siinä määrin, että kirjasto ei saa olla niin raskas, että se itsessään hidastaisi näkyvästi Unity-pelimoottorin päivitys-sykliä. Tätä pystytään tarkkailemaan ensimmäisessä esimerkkipelissä.

Kirjaston toimivuutta testataan kehittämällä pelihallimainen esimerkkipeli vaiheessa *Esimerkkipeli Syön sut* (Vaihe 3). Pelissä on hyvin yksinkertainen pelilogiikka, mikä on kokonaisuudessaan skriptimoottorin varassa. Tällä tavoin voidaan varmistaa, ettei skriptimoottorin rakenne ole liian raskas Unity-pelimoottorin päällä pyöritettäväksi. Kirjaston kanssa pelin pitäisi pyöriä yli 60 kuvaa sekunnissa. Ensimmäinen esimerkkipeli näyttää, miten pelkästään kirjaston avulla voi tehdä pelejä, kunhan Unitylle tehty pohja on yhdistetty skriptimoottoriin tarpeeksi kattavasti. Esimerkkipeli dokumentoidaan hyvin ja siinä pyritään käyttämään kirjastoa kattavasti, jotta se voi toimia tapaustutkimuksessa esimerkkinä. Kuvaan tämän vaiheen toteutumisen Luvussa 5.4. Esimerkkipelin toteutuksessa saattaa tulla esille vielä joitakin ongelmakohtia kirjaston käytettävyydessä. Mikäli ongelmakohtia tulee esille, nämä korjataan.

Kirjasto annetaan Jyväskylän yliopiston *Peliteknologia*-kurssin osallistujien käytettäväksi (Vaihe 5). Peliteknologia-kurssi on kolmen tai viiden opintopisteen laajuinen kurssi Jyväskylän yliopistossa, joka järjestetään vuosittain. Kurssille osallistuu parikymmentä maisterivaiheen opiskelijaa, joista suurin osa lienee Jyväskylän yliopiston *Pelit ja pelillisyys* -maisteriohjelman opiskelijoita. Heistä voi olettaa heidän ole-

van henkilöitä, joilla on korkea ymmärrys peleistä ja pelien kehityksestä ja korkea harrastuneisuus kyseisellä aihealueella. Opiskelijat tekevät kurssilla viikkotehtäviä kurssilla ja tapaustutkimus käsittelee yhtä tälläistä viikkotehtävää. Tehtävässä he muokkaavat esimerkkipeliä tehdäkseen omanlaisen pelinsä esimerkkipelin rakennetta apuna käyttäen. Kurssin puolesta yhteen viikkotehtävään on laskettu käytettäväksi noin kahdeksan työtuntia. Tehtävän lopuksi opiskelijat kirjoittavat pohtivan tekstin oman tehtävänsä onnistumisesta ja antavat palautetta kirjaston toimivuudesta ja ongelmakohdista ja esittävät parannusehdotuksia.

Opiskelijoille esitetään seuraavat kysymykset: "Oliko kirjastoa helppoa käyttää?", "Havaitsitko ongelmia kirjaston toiminnassa tai puuttuiko kirjastosta jokin toiminto tai ominaisuus, jonka olisit halunnut siinä olevan?", "Nostiko tehtävä kiinnostusta modaukseen?" ja "Välittyikö kirjaston tavoite?". Näillä lisäkysymyksillä pyritään tarkentamaan opiskelijoiden vastauksia liittymään kirjaston *omaksuttavuuteen* ja *käytettävyyteen*. Kysymys "Nostiko tehtävä kiinnostusta modaukseen?" tuo esille opiskelijan ajatuksia tehtävän mielekkyydestä ja voi auttaa ensi vuoden tehtävän suunnittelussa. Opiskelijoilta kysytään myös arvio, kuinka kauan he käyttivät aikaa tehtävän tekemiseen. Ajankäyttö on mielekäs kysymys oman kehitykseni takia tehtävien suunnittelussa ja se on mielenkiintoinen yksityiskohta, sillä se voi esimerkiksi selittää joitain vastaukseen liittyneitä asioita. Esimerkiksi hyvin nopeasti tehdyssä vastauksessa ongelmat voivat johtua perehtymisen puutteesta dokumentaatioon kirjaston omaksuttavuuden sijaan. Saadut vastaukset kootaan ja tallennetaan ja niitä analysoidaan, kuten seuraavissa kappaleissa kuvataan.

Tässä tutkimuksessa tapaustutkimus on luonteeltaan kuvaileva (*descriptive*). Kuvaileva tapaustutkimus tutkii kohdetta oikean maailman tilanteessa (Yin 2014). Tapaustutkimuksella pystytään siis tässä tutkimuksessa selvittämään, miten kirjasto toimii oikeassa maailmassa. Kuvailevaan tapaustutkimukseen sopii lineaaris-analyyttinen esitystapa (Yin 2014, s.187).

Yin (ks. 2014, s.193-) käsittelee kolmea erilaista tapaustutkimusta, joissa tapaustutkimus on osana toista, isompaa tutkimusta. Näistä lähinnä omaa tutkimustani on kolmas esitelty tapaustutkimus, jossa suurempi tutkimus on käyttänyt tapaustutki-

musta apuna jonkun allaolevan prosessin valaisemiseen ja/tai selventämiseen. Tälläisessä tilanteessa kysymykset voivat olla samankaltaisia kuin päätutkimuksessa ja kyselyt voivat tapahtua rinnakkaisesti tai peräkkäin.

Tapaustutkimuksessa syntyneissä vastauksissa on sekä dokumentaation että haastatteluiden piirteitä. Opiskelijat vastaavat kyselyyn itsenäisesti ja kirjoittavat raportin tehdystä työstä, ja Yin mieltää raportit dokumentaatioksi. Kuitenkin heille on myös esitetty ennaltamääriteltyjä kysymyksiä, jotka tarkentavat heidän vastauksiaan aihealueeseen. Joka tapauksessa vastauksia on tarkoitus käsitellä pikemminkin laadullisena datana, kuin kyselyiden vastauksina määrällisesti.

Palautuksissa on myös mahdollisuus vuoropuheluun TIM-palvelussa[3], jossa vastaukset palautetaan. Tämä mahdollistaa esimerkiksi tarkentavien kysymysten esittämisen opiskelijoiden vastauksiin liittyen. Opiskelijat palauttavat myös tehdyn projektinsa, jota voidaan myös käyttää tutkimusaineistona. Yin (2014, s.120-121, s.220) kannustaakin käyttämään useita erillisiä tietolähteitä tapaustutkimuksen tekemiseen. Hän huomauttaa, että on olennaista tarkastella eri tietolähteistä hankittua tietoa yhdessä. Vastauksiin tekemäni kommentit ja vastausten arviointien perustelut vastaavat Yinin (2014) ideaan tehdä muistiinpanoja datasta: jouduin tehtävien arvostelun yhteydessä perehtymään opiskelijoiden koodiin ja pohtimaan heidän vastauksiaan.

Vaiheessa *Esimerkkipeli Curriculum Vitae* teen vielä toisen esimerkkipeli (Vaihe 7). Tässä esimerkkipelissä kiinnitetään huomiota erityisesti tapaustutkimuksen jälkeen lisättyihin ominaisuuksiin. Esimerkkipelistä löytyy tarkempi kuvaus luvusta 5.6. Tässä vaiheessa teen myös yksinkertaisen testin *suorituskyvyn* takaamiseksi. Ajan molemmat esimerkkipelit tietokoneellani ja tarkistan, että pelien päivitysnopeus ei putoa liian alhaiseksi.

3. TIM on Jyväskylän yliopiston kehittämä palvelu, johon henkilöt voivat itse kehittää dokumentteja. TIM muistuttaa toiminnallisuudessa vähän wiki-sivustoa, mutta sen sisällä voi esimerkiksi ajaa ohjelmakoodia. Löydettävissä osoitteessa `https://tim.jyu.fi/`.

4.3.3 Suunnittelutieteellisten ohjenuorien toteutuminen tässä tutkimuksessa

Hevner ym. (2004) määrittelevät seitsemän ohjenuoraa, joiden tarkoituksena on helpottaa tutkimuksen tekijöitä, arvostelijoita, toimittajia ja lukijoita ymmärtämään, mitä tehokas suunnittelututkimus vaatii. Kaikkien ohjenuorien seuraaminen täsmälleen ei ole pakollista, vaan kirjoittajien täytyy käyttää luovia taitojaan ja päätöksentekokykyä päättääkseen, milloin, missä ja miten ohjenuoria käytetään apuna. He muistuttavat tosin, että kaikkia ohjenuoria täytyy jollain tavoin käsitellä, jotta tutkimus voisi olla valmis. Seuraavaksi esittelen heidän määrittelemät seitsemän ohjenuoraa, jonka jälkeen vastaan jokaiseen.

Ensimmäinen ohjenuora Suunnittelututkimuksen tulee tuottaa konstruktio, malli, metodi tai ilmentymä.

Toinen ohjenuora Tutkimus kohdennetaan ongelmiin, jotka ovat ratkaisematta ja joilla on taloudellista arvoa.

Kolmas ohjenuora Artefaktia tulee arvioida jollain tavoin.

Neljäs ohjenuora Suunnittelututkimuksen tulee tarjota ainakin joko itse tutkimusartefaktin, laajentaa suunnittelututkimuksen tietopohjaa tai kasvattaa suunnittelututkimuksen metodologiaa.

Viides ohjenuora Suunnittelututkimuksen tulee olla täsmällinen. He tarkentavat, että suunnittelututkimuksessa artefaktin rakennus sekä arviointi tulee tehdä täsmällisesti, mutta samalla liialliseen täsmällisyyteen pyrkiminen vähentää tutkimuksen relevanssia.

Kuudes ohjenuora Suunnittelu on *etsintäprosessi*. Artefaktin tulee olla mahdollisimman lähellä optimaalista.

Seitsemäs ohjenuora Tutkimuksen pitäisi olla ymmärrettävissä sekä teknologiaan että johtotehtäviin suuntautuvalle henkilöstölle.

Ensimmäisen ohjenuoran vastauksena tässä tutkimuksessa tuotettiin ilmentymä kirjastosta, itse fyysinen tuote, jolla on arvoa sen käyttötarkoituksessa. Tutkimusta tehdessä vastaavaa kirjastoa ei ollut saatavilla, mikä vastaa Hevnerin ja muiden (2004) toisen ohjenuoran vaateeseen: ilmentymän tekemistä ennen ei ollut vielä tietoa, pystytäänkö kyseinen järjestelmä tekemään onnistuneesti, ja kuinka onnistuneesti se

toimisi. Toisen ohjenuoran vaatimus on kaksiosainen ja sen mukaan tutkimus tulee kohdentaa tarkemmin ongelmiin, joissa on taloudellista arvoa. Tähän osaan on vaikeampaa vastata yksiselitteisesti.

Unity on hyvin suosittu pelimoottori ja kuten aiemmin pohdittiin käyttäjälähtöistä muokattavuutta käsittelevässä luvussa, on käyttäjälähtöinen muokattavuus hyvä tapa pidentää tuotteiden elinikää ja tarjota lisää sisältöä ja arvoa pelille. Tutkimus siis pyrkii tuomaan lisäarvoa Unity-pelimoottorilla tuotettuihin peleihin, minkä uskon olevan mielekäs tavoite taloudellisesti. Kuitenkin tutkimus on tehty Pro Gradu -työnä, joten sitä ei ole toteuttu tietyn yrityksen tarpeisiin.

Tutkimus tuottaa tutkimusartefaktin, mikä vastaa neljänteen ohjenuoraan. Kolmas ohjenuora ehdottaa arvioimaan artefaktia. Arvioin sitä aiemmin esitetyin tavoin: skenariolla, tapaustutkimuksella ja prototyypillä.

Tutkimuksen täsmällisyys eli viides ohjenuora otettiin huomioon kuvailemalla kehitysvaiheet tarkasti ja esittämällä tapaustutkimuksessa saatu data kuten Yin on neuvonut olevan hyvä ja täsmällinen tapa tuottaa tapaustutkimus. Tutkimuksessa kuvaillaan tarkoin artefaktin rakennuksen ja arvioinnin eri vaiheita Luvussa 5, jotta samankaltaisen kirjaston tuottaminen olisi toistettavissa.

Kuudes ohjenuora tulee esille iteratiivisena etsintänä, joihin eri syklit auttavat. Kuten aiemmin esittelin Hevnerin ja muiden (2004) määrittelemää *etsintäprosessia*, tulee etsinnällä löytää toimiva ratkaisu. Artefaktin kehitystä on iteroitu usean syklin ajan, joissa sitä on arvioitu eri tavoin. Nämä syklit osoittavat artefaktin olevan toimiva.

Tässä tutkimuksessa pääpaino tullaan asettamaan oletukselle, että lukija on teknologiaan perehtynyt henkilö, mikä on osittain ristiriidassa seitsemännen ohjenuoran kanssa. Tämä ei kuitenkaan tarkoita sitä, että tekstistä pyritään tekemään vaikeaselkoista tai että käsitteitä ei selvennetä; Teksti kirjoitetaan kuin sen lukisi tietotekniikkaan perehtynyt henkilö kohtalaisella teknologian ymmärryksellä. Tutkimuksen päätavoite on toimia Pro Gradu -tutkielmanani ja edistää tiedettä käyttäjälähtöiseen muokattavuuteen liittyen, mutta samalla se käsittelee esimerkiksi kysymystä, miksi käyttäjälähtöinen muokattavuus kannattaa sallia (ks. 2.4.). Tutkimukseni

samalla siis perustelee johtotehtävien henkilöstölle, miksi tutkimus on tärkeä.

5 Unityn käyttäjälähtöistä muokattavuutta helpottava kirjasto

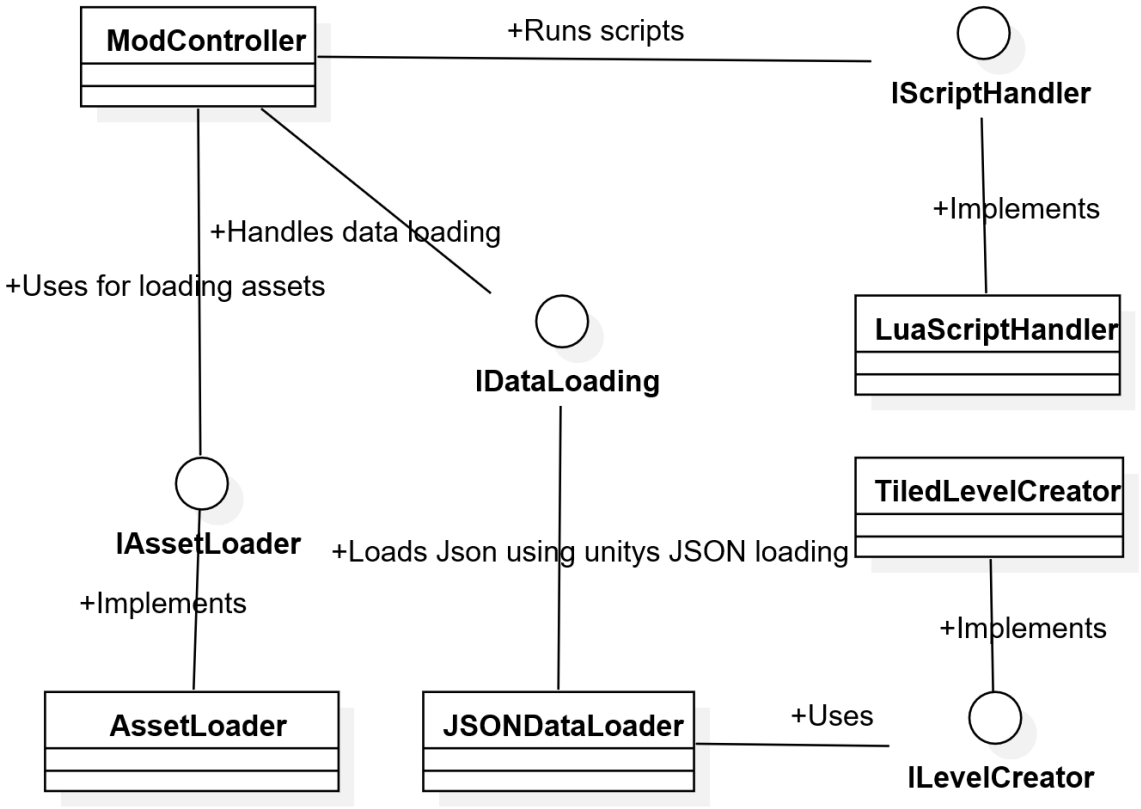
Tässä luvussa käyn läpi edellisessä luvussa esitetyn tutkimuksen toteutumisen. Aluksi kuvaan yleisesti toteutukseen liittyviä asioita, käsittelen kirjaston arkkitehtuuria ja rakennetta sekä sen käyttöä. Tämän jälkeen käyn läpi Kuviossa 1 esitetyt vaiheet. Vaiheet 3 ja 4 sekä 5 ja 6 ovat yhdistetty keskenään, jotta niiden takia syntyneet korjaukset voitaisiin käsitellä mielekkäinä kokonaisuuksina. Luvussa 5.5 käyn läpi tapaustutkimuksen toteutuksen.

5.1 Kirjaston rakenne ja sen käyttö

Kirjasto helpottaa kehittäjien työtä modien lisäämisessä Unity-pelimoottorilla tehtyihin peleihin. Kirjaston rakenne noudattaa komponenttimallia kuten Unitykin. Kirjasto toimii modauksen kontrollerina pelissä (ks. controllers Murray 2014, s.11): se keskustelee erilaisten kokonaisuuksien välillä salliakseen käyttäjälähtöisen muokattavuuden peleissä. Jotta yksittäisen ratkaisun vaihtaminen toiseen, rajapinnan toteuttavaan, ratkaisuun olisi mahdollisimman helppoa, käyttää kirjasto rajapintoja erilaisten kokonaisuuksien välillä keskustelemiseen. Tämä nosti kirjaston yleistettävyyttä ja noudattaa Parnasin (1979) toista ohjetta: tieto pitää piilottaa komponenttien välillä. Kuviossa 2 esitetään kirjaston rakenne. Esittelen seuraavaksi kirjaston arkkitehtuurin ja rakenteen, jonka jälkeen esittelen esimerkkitilanteen kirjaston käytöstä.

5.1.1 Arkkitehtuuri ja rakenne

`ModController`-luokka hallitsee osakokonaisuuksien yhteistyötä ja tarjoaa kattavasti erilaisia tapoja kutsua eri kokonaisuuksia. Pienemmät luokkakokonaisuudet eivät tiedät toisistaan, eivätkä `ModController`-luokan olemassaolosta, mikä noudattaa Parnasin (1979) ohjetta ohjelman tasomaiseen jäsentelyyn. Kaikki kehittäjän



Kuvio 2. Kirjaston arkkitehtuuri yksinkertaisesti kuvattuna

esittämät kutsut menevät `ModController`-luokan kautta alaluokille.

Tiedostonlatauskomponentin (Kuviossa 2 `AssetLoader`) avulla voi ladata tiedostoja mielivaltaisista kansioista tietokoneella. Tämä komponentin avulla voi ladata 3D-malleja, tekstuureja, ääniä ja tekstiä.

Dataa käsittelevä luokka (Kuviossa 2 `JSONDataLoader`) muuttaa annetun JSON-tiedoston tai tekstin olioiksi. Data tallennetaan mahdollisimman helppolukuisena, jotta sitä pystyisi tutkailemaan ilman erillisiä työkaluja. Tämän olen Luvussa 3.3 pohtinut olevan hyvä tapa sallia muokattavuus. Mielestäni JSON-tallennusformaatti on helppolukuista ja olen käyttänyt sitä aiemmin Unityn kanssa.[1]

Skriptimoottori (Kuviossa 2 `LuaScriptHandler`) suorittaa sille annetun tekstin Lua-ohjelmointikielenä. Jotta kirjastolla tehty peli olisi myös toiminnallisuudeltaan

1. Tarkempaa tietoa osoitteesta `http://www.json.org/`

käyttäjien muokattavissa, tuli kirjaston sallia pelilogiikan määrittely käyttäen käyttäjälle muokattavissa olevia skriptejä. Sovellusalueen skriptauskieliä käsittelin luvussa 2.1 ja esitin, miksei ole mielekästä luoda omaa sovellusalueen kieltä. Tarjolla oli useita, kattavia ohjelmointikieliä, joita voi helposti käyttää skriptikielenä Unity-ympäristössä. Yksi niistä oli aiemmin osiossa 3.3 esille tullut Lua-ohjelmointikieli, jota on käytetty monissa peleissä skriptauskielenä. Valitsin tästä syystä Luan skriptauskieleksi kirjastoon, jotta Luaa aiemmin käyttäneet käyttäjät voisivat helpommin muokata kirjastolla tehtyjä pelejä.

Yksittäinen kokonaisuus, esimerkiksi kirjaston skriptimoottori, on yksittäinen komponentti kiinnitettynä Unity-peliolioon. Tällä tavoin jokaisen skriptin voi hakea yhdistävässä ja hallinoivassa luokassa kutsumalla peliolion `GetComponent`-metodia:

```
modBase.GetComponent<IScriptHandler>();
```

Mikäli halutaan vaihtaa esimerkiksi skriptien käsittelijäkomponenttia, onnistuu se poistamalla vanha komponentti pelioliosta ja vaihtamalla uusi tilalle. Tämä onnistuu muutamalla muutoksella Unity-editorin kautta, mikä tekee kehittäjien skriptityökalun vaihtamisesta hyvinkin nopeaa. Uuden skriptimoottorin toiminnallisuuden tekeminen vie luonnollisesti jonkun verran aikaa kirjaston käyttäjiltä.

Unity-skriptit tallentavat viitteen toisiin tarvitsemiinsa olioihin, kuten Murray (2014, s.3) neuvoo tekemään. Hän ehdottaa myös käyttämään *Singleton*-patternia, missä olioihin viitataan staattisesti ja patterni varmistaa, että olioita voi olla olemassa vain yksi kappale. En halunnut pakottaa kuitenkaan kirjastosta singletonia, joten kirjastolle tehdään apuskripti `SingletonModsInUnity`, jonka avulla sitä voidaan kutsua staattisesti ja jonka avulla kirjastoa voidaa käyttää kuin se olisi singleton.

Tämän tutkimuksen puitteissa keskityin kehittäjien käyttäjälähtöisen muokattavuuden sallimiseen, enkä esimerkiksi valmistanut editoria itse tuotteen loppukäyttäjille muokkauksen helpottamiseksi. Kyseinen editori olisi sidonnainen valintoihin, joita pelinkehittäjät haluavat tehdä suhteessa pelissä käytettyyn muokkausen sallivaan teknologiaan. Esimerkiksi kehittäjät voisivat haluta käyttää kaikkia muita komponentteja valmiista kirjastosta, mutta kehittää täysin oman skriptauskielensä peliin.

Täten ei ollut kovin mielekästä pyrkiä varautumaan kaikkiin mahdollisiin tilanteisiin, eikä myöskään aika riittäisi toteuttamaan näin kattavaa editoria.

5.1.2 Kirjaston käyttöesimerkki

Kirjaston käyttö on täysin sidoksissa sillä toteuttavaan peliin ja siihen, missä määrin peliä halutaan sallia muokattavan tai kuinka suuri osa pelistä halutaan toteuttaa kirjaston avulla. Esittelen yksinkertaisen tilanteen, jossa kehittäjä käyttää kirjaston pääominaisuuksia kuvitteellisen roolipelin luomiseen, jossa on vuoropohjainen taistelumekaniikka. Esimerkin kehittäjä on hyvin Unityyn perehtynyt ohjelmoija, joka osaa käyttää kirjastoa kohtuullisesti. Vaiheet kuvataan yksinkertaistaen toteutukseen liittyvää työtä merkittävästi.

Kehittäjä aloittaa kirjaston käytön lisäämällä kirjaston projektiinsa. Kehittäjä lisää Unityn näyttämöön valmiiksi määritellyn pelioliopaketin (*Prefab*), jossa on määritelty kirjaston tarvitsemat komponentit ja viitteet. Tämä sallii kirjaston käytön pelioliosta, jotka sisältävät viitteen `ModController`-komponenttiin. Viitteen saa esimerkiksi käyttämällä olion singleton-toiminnallisuutta tai viemällä Unityn editorin avulla viitteen pelioliolta toiselle.

Kun kirjasto on käyttövalmis, voidaan sille antaa käskyjä erilaisten asiakokonaisuuksien suorittamiseksi. Kehittäjä haluaa käyttää apuna JSON-muotoisia kenttätiedostoja, jotka on määritelty Tiled-kenttäeditorin avulla. Kehittäjä määrittelee kentän kenttäeditorin avulla ja avaa kirjaston näyttämönluomisikkunan. Ikkunalle viedään viite `ModController`-luokan sisältävästä pelioliosta, kentän luontiin käytettävästä tekstitiedostosta ja kooditiedostosta, joka määrittelee yksittäisien kenttätiilien muuttamisen Unity-olioiksi. Kehittäjän antaman suorittamispyynnön jälkeen Unityssä aukeaa uusi näyttämö luoduilla peliolioilla. Unityssä on tapana käyttää tyhjiä peliolioita "kansioina", joten kehittäjä tekee tälläisen kansio-olion ja siirtää kentän rakenteeseen liittyvät pelioliot tämän olion alle. Näin Unityn näyttämön pelioliolistaus pysyy helppolukuisena.

Kehittäjä lisäsi kentän luonnissa kaksi pelihahmoa peliin. Hahmojen kuvat ladataan

kirjaston avulla *Mods*-kansion alla olevasta *Textures*-kansiosta nimillä "hahmo1" ja "hahmo2". Kehittäjä haluaa myös, että pelaaja voi keskustella hahmojen kanssa. Keskustelut hän toteuttaa tallentamalla ne JSON-datana omiin tiedostoihinsa *Data*-kansioon. Hän lataa tekstimuotoiset tiedostot käyttäen kirjaston tarjoamaa tiedostonlatausta. Pelin käynnistyessä hän muuttaa tekstin `ModController`-luokan avulla JSON-dataksi. `ModController`-luokka sisäisesti kutsuu `IDataLoading`-rajapintaa (ks. Kuvio 2) ja palauttaa kehittäjän määrittelemän luokkarakenteen mukaiset data-oliot. Yksittäinen pelihahmo muistaa nämä data-oliot keskusteluja varten. Keskustelun käymisen kehittäjä toteuttaa Unityssä C#-ohjelmakoodin avulla.

Kuvalla "hahmo2" luodun hahmon kanssa keskustelu johtaa taisteluun pelissä. Kehittäjä luo taistelua varten Unity-koodin puolella muutaman aliohjelman, joilla käyttöliittymän voi luoda. Tämän jälkeen hän määrittelee `CreateGUI`-Lua-skriptin, jossa käyttöliittymän rakenne määritellään. Lua-skriptistä voi kutsua taisteluun liittyviä tapahtumia, kuten hyökkää, peräänny tai käytä esinettä, jotta käyttöliittymäkomponenteilla voidaan määritellä toiminnallisuudet. Kehittäjä jakaa myös pelihahmojen tietoja Lua-skripteihin. C#-ympäristön muuttujia voidaan jakaa Luan puolelle globaaleina muuttujina kirjaston avulla. Näin Lua-skriptissä voidaan esittää esimerkiksi pelaajan ja vastustajan kestävyys taistelutilanteessa.

Kehittäjä on saanut yksinkertaisen pelin valmiiksi. Käyttäjälähtöinen muokattavuus tässä esimerkissä tulee esille siten, että pelin pelaaja pystyy muokkaamaan peliä monin tavoin. Pelaaja voi käydä vaihtamassa "hahmo2"-kuvan sellaiseksi kuvaksi, mitä vastaan hän haluaa taistella. Hän voi myös muokata keskusteluja muokkaamalla JSON-keskustelutiedostoja tai hän voi muokata käyttöliittymää muokkaamalla `CreateGUI`-skriptitiedostoa. Pelin kehittyessä pidemmälle peliin saatetaan lisätä esimerkiksi erilaisia hyökkäyksiä. Kun näiden hyökkäysten tiedot on tallennettu *Mods*-kansion alle ja luettu kirjaston avulla, voi pelaaja muokata tai lisätä hyökkäyksiä.

Yksinkertaiset ohjeet kirjaston käyttämiseen löytyvät osoitteesta `https://tim.jyu.fi/view/users/tesatapa/modsinunityohje`, mutta sivuille pääsy vaa-

tii TIM-tunnukset tai Jyväskylän yliopiston Korppi-tunnukset.[2] Osoitteesta löytyvät ohjeet olivat apuna tapaustutkimukseen osallistujilla.

5.2 Rungonmuodostusvaihe (Vaihe 1)

Kirjasto sisältää *ModController*-nimisen yleiskäyttöisen käsittelijän, joka hallinnoi ja koordinoi työkalujen välistä yhteistyötä. Tämä luokka hallinnoi eri osakokonaisuuksien yhteistyötä ja tallentaa esimerkiksi tiedostopolun haluttuihin tiedostoihin: kirjasto oletuksena hakee skriptitiedostoja pelin *Assets*-kansion tai pelin suoritustiedoston tasolta kansiosta nimeltä *Mods* ja sen sisältä kansiosta *Scripts*. Tämän kansion voi koodin avulla muuttaa mieleisekseen tai hakea suoraan käyttäen jotain toista tiedostopolkua. Kirjastossa on kolme suurempaa, erillistä kokonaisuutta: tiedostojen lataus, datan lataus ja skriptimoottori. Jokaisella on oma rajapintansa, jonka avulla niitä käytetään. Rajapinnat perivät myös yhden yhteisen rajapinnan toiminnallisuuksia varten, joita voi olettaa löytyvän kaikista kolmesta kokonaisuudesta. Tälläinen ominaisuus on Start-metodi. Unity automaattisesti kutsuu kaikkien `MonoBehaviour`-luokan perivien luokkien `Start`-metodia, mutta tästä saattaa syntyä ongelma, sillä suoritusjärjestykset saattavat mennä ristiin. Luokka A pyytää Startissa luokka B:tä tekemään toiminnallisuuden, joka vaatisi B-luokan Startin olevan jo suoritettu. ModController kutsuu kaikkien kirjaston alaluokkien `DoStart`-metodeja, jolloin täytyy vain tarkistaa, että `ModController`-luokan `Start` suoritetaan ennen muita pelin luokkia.

Unityssa on valmis lataaminen toteutettu *Resources*-kansion tai kansioiden sisältä. Tämä ei kuitenkaan toimi modien kanssa järkevästi, sillä tiedostot Resources-kansiossa ovat pakatussa muodossa, eikä niitä siten pääse mielekkäästi muokkaamaan ilman ylimääräisiä työkaluja. Unity sisältää myös eheystarkistuksen, joka antaa virheen, mikäli tiedostoja muokkaa käsin. Näistä syistä modit tallennetaan *Mods*-kansioon ajettavan tiedoston tasolle, missä ne ovat tekstimuotoisina tiedostoina ja mistä ne ladataan ajonaikaisesti. Teksti- ja raakatiedostojen latauksessa käytin aiemmin toteuttamaani tiedostonlataustyökalukokoelmaa nimeltä *Load*. Raakatiedos-

2. TIM-tunnukset voi tehdä sähköpostin omistava henkilö

tolla tarkoitan tässä tavumuodossa tallennettua tiedostoa, joka sisältää esimerkiksi tekstuurin tavumuotoisena. Load ei itsessään sisältänyt työkaluja esimerkiksi tekstuurien muodostamiseksi, joten tämä piti ratkaista erikseen.

Unityn puolella tarvitaan kaksi erillistä oliota kirjaston toiminnallisuuden saavuttamiseksi: kirjastoa käyttävä peliolio ja esimerkkiolio, josta kirjasto lataa komponentit. Nämä voisivat olla sama peliolio Unityn sisällä, mutta selvyyden vuoksi tein niistä erilliset oliot. Ohjelman käynnistyessä kirjasto lataa toiselta pelioliolta kaikki komponentit itselleen käyttäen apuna Unityn `GetComponent<TYPE>();`-metodia, jolla koodissa saa komponentissa viitteen toiseen komponenttiin.

5.3 Rungon viimeistely (Vaihe 2)

Latauksen toteutuksessa tuli ongelmia kaikkien erilaisten latauksien toteuttamisessa. Ongelma oli, että Unityssä on hyvin paljon valmiina erilaisia tapoja muuttaa raakadataa erilaisiksi kokonaisuuksi, esimerkiksi tekstuuriksi, mutta jokaiselle laukselle täytyi määritellä erikseen aliohjelma. Unityn Resources-kansiosta ladatessa voi tiedoston ladata yhdellä kutsulla määrittämällä ladattavan asian tyypin latausfunktiolle ja olisin halunnut samanlaisen toiminnallisuuden lataukseen. Unitysta löytyi WWW-luokka, mikä on suunniteltu tiedostojen lataamiseksi internetin avulla. Tätä luokkaa pystyin käyttämään myös tietokoneelta tiedostojen lataukseen varsinkin tapauksissa, joissa raakadatasta olisi pitänyt rakentaa Unityn komponentti. WWW-luokka tosin toimii asynkronisesti. Tämä on hyvä ominaisuus siltä osin, että tiedoston lataaminen ei pysäytä pelilogiikkaa ja tiedostoja pystyy lataamaan pelinkin aikana. Ongelma syntyy siitä, että Unity käyttää vanhempaa versiota C#-ohjelmointikielestä, jossa ei esimerkiksi *async*-ominaisuuksia ole toteutettu. Tämä tarkoittaa sitä, että lataus täytyy toteuttaa joko tekemällä säie, joka lataa tiedoston, tai käyttämällä Unityn tarjoamia rutiineja (*coroutine*). Kirjastossa käytetään jälkimmäistä vaihtoehtoa. Latauksen loputtua kirjasto kutsuu haluttua aliohjelmaa, jolle annetaan parametrina ladattu tiedosto.

Unityn AssetStoresta löytyi hyvin toimiva obj-formaattia käyttävä latauskirjasto,

jonka avulla kolmiulotteisten objektien lataus helpottui huomattavasti. Käyttäjän aaro4130 kirjoittama algoritmi tekee kaikki tarvittavat lataukset ajonaikaisesti, joten se sopii erinomaisesti kirjaston tarpeisiin. Algoritmi lataa myös malliin liittyvät tekstuurit, joten mallia voi käyttää suoraan Unityssä. Kaksiulotteisten kuvien lataaminen toteutettiin lataamalla tiedosto bittimuodossa käyttäen Load-paketin tarjoamia tiedostonlataustyökaluja ja sen jälkeen luomalla siitä uusi kaksiulotteinen tekstuuri.

Skriptimoottori toteutettiin käyttäen MoonSharp-kirjastoa,[3] minkä avulla pystytään ajamaan Lua-koodia suoraan C#-ohjelmointikielen päällä: MoonSharp-kirjasto suorittaa Lua-koodin C#-ympäristössä. Kirjasto on toteutettu erittäin hyvin toimivaksi Unityn kanssa. MoonSharp-kirjasto vaatii joitain tiettyjä ominaisuuksia, esimerkiksi tyyppien rekisteröimistä. Kirjastolle pitää määrittää kaikki C# tyypit, joita Lua-koodin puolella pystyy käsittelemään. Tämän pystyisi kiertämään sallimalla kaikkien tyyppien käytön, mutta tyyppien salliminen erikseen parantaa modien turvallisuutta. Modit eivät pääse käsiksi esimerkiksi tiedostojenkäsittelyyn liittyviin kirjastoihin, joita C#-kieli tarjoaa. Tämä on ainoa, selkeästi MoonSharpin mukaan määritelty ominaisuus tutkimuksessa toteutetussa kirjastossa.

Funktioiden tyypit täytyy viedä kirjastolle muutettuina vastaaviksi tyypeiksi. Tarkastelemme seuraavanlaista aliohjelmaa:

```
public int Laske(int a, int b)
{
return a*b;
}
```

Aliohjelma jaetaan kirjaston puolelle muuttamalla se C#:sta löytyvään `Func`-luokkaan ja kutsumalla funktiota `BindFunction`-metodia seuraavalla tavalla:

```
modController.BindFunction("laske",
   (Func<int, int, int) Laske);
```

3. ks. `http://www.moonsharp.org/`.

Tässä kutsussa merkkijono “laske” määrittää, millä nimellä aliohjelmaa voidaan kutsua Luan puolella. Func- muunnoksessa kaksi ensimmäistä tyyppiä ovat aliohjelman parametrien tyypit ja viimeinen on aliohjelman palautusarvon tyyppi. Tässä esimerkissä kaikki ovat kokonaislukuja. Samankaltaisesti olioita pystytään yhdistämään Luan puolelle kutsulla:

```
modController.BindObject("luaNimi", viiteOlioon);
```

Tämä sallii olion käsittelyn kaikissa skripteissä annetulla merkkijonolla. Luaa ja MoonSharppia käytettäessä funktiotkin voisi yhdistää Lua-koodiin käyttäen *BindObject*-metodia, mutta mielestäni oli mielekästä tehdä erikseen funktioita yhdistävä aliohjelma muita skriptauskieliä varten. Keskustelu rajapintojen välillä jää toteutettavaksi itse pelin koodiin. Kirjaston mukana tulee esimerkkiprojekti, jossa esitetään ja testataan kirjaston toimimista yksinkertaisessa Unity-pelissä, mitä esitellään seuraavassa alaluvussa.

5.4 Esimerkkipeli Syön sut ja korjaukset (Vaiheet 3 ja 4)

Tutkimuksessa toteutettiin esimerkkipeli, joka käyttää apuna tässä luvussa esiteltyä kirjastoa. Esimerkkipeli on hyvin yksinkertainen peli, jossa pelaaja pystyy liikuttamaan hahmoaan pienessä kentässä. Pelaajille on laitettu törmäyksenkäsittelijä, joka tarkkailee suuntaa, josta pelaaja törmää toiseen pelaajaan. Mikäli pelaaja on toisen pelaajan yläpuolella törmäyksen sattuessa, häviää alla ollut pelaaja pelin. Huomionarvoista on, että peli ei välttämättä näytä parasta tapaa ohjelmoida käyttäjälähtöisesti muokattavaa peliä tai isompaa pelikokonaisuutta ylipäätään, vaan pikemminkin korostaa kaikkia erilaisia ominaisuuksia, joita kirjasto tarjoaa. Peli osoittaa, miten pelin voi tehdä käyttäen lähes pelkästään kirjaston ominaisuuksia. Sitä voidaan siten pitää esimerkkipohjana yksinkertaisten pelien luomiselle ja kirjaston käyttämiseen.

Esimerkissä käytetään Json-muotoisia data-tiedostoja esimerkiksi maailman tietojen tallentamiseen ja kaikkien eri pelien tallentamiseen. Esimerkkipohjan päällä voisi siis olla useita erillisiä Lua-pelejä ja `ExampleGameController`-luokkaa ei tarvit-

sisi muokata millään tavoin, kunhan pelit määritellään *Games*-kansioon, josta esimerkkipohja etsii kaikkien pelien tietoja. Yksittäiseen peliin liittyen tallennetaan pelin nimi, Lua-kontrolleri, maailman luontiskripti ja minkä nimisiä kenttiä maailmanluonti skriptistä käytetään. Esimerkissä myös suurin osa asioista ladataan ajonaikaisesti kansiosta. Esimerkiksi taustamusiikki ja pelihahmojen kuvat ladataan kaikki Lua ja C# -koodin avulla ajonaikaisesti, mikä poikkeaa merkittävästi "normaalista" Unity käytöstä, jossa kenttä ladataan etukäteen määritetyistä kokonaisuuksista (*asset*).

Toiminnallisuus esimerkkipelissä on ohjelmoitu `ExampleGameController`-tiedostoon C#-ohjelmointikielellä ja Lua-tiedostoihin `Controller`, missä määritellään pelaajien liikuttaminen, luominen ja törmäysten käsittely, `WorldCreation`, jossa on listattu erilaisten kenttien luominen yhteen tiedostoon ja jota voitaisiin käyttää muissakin saman tyylisissä peleissä apuna, ja `GameRules`, mikä tarkastelee lähinnä pelin loppumista. Esimerkiksi `GameRules`-tiedoston käyttäminen `Controller`-luokassa tuo hyvin esille, kuinka toisesta Lua-skriptistä pystytään kutsumaan toista Lua-skriptiä, mikä sallii koodin jakamiseen useaan tiedostoon ja siten myös koodin uudelleenkäytön.

Esimerkkipeli dokumentoitiin tarkasti ja siihen liittyen tehtiin lyhyt ohjesivu.[4] Esimerkkipelin toteutuksessa ei havaittu suurempia puutteita kirjastossa ja omasta mielestäni kirjastoa oli helppo käyttää. Koko esimerkkipelin projekti paketoitiin .zip-pakettiin, jotta Peliteknologia-kurssilaiset voivat sitä käyttää.

5.5 Peliteknologia-kurssin tehtävä ja lisätyt ominaisuudet (Vaiheet 5 ja 6)

Suurin kirjastoon kohdistuva arviointi tapahtui tässä vaiheessa. Kävin pitämässä luennon Peliteknologia-kurssilla käyttäjälähtöiseen muokattavuuteen liittyen ja pohjustin opiskelijoille viikkotehtävän, jossa heidän tuli käyttää kirjastoa ja esimerkkipeliprojektia. Tehtävässä heidän tuli muokata esimerkkipeli uudenlaiseksi peliksi

4. ks. `https://tim.jyu.fi/view/users/tesatapa/modsinunityohje`

käyttäen esimerkkiohjelmakoodia ja -tiedostoja hyväksi. Itse kirjastoa ei ollut tarkoitus muokata, mutta mikäli tämä oli tarpeen tehtävän tekemiseksi, pyysin heitä kirjoittamaan tämän loppuraporttiin. Opiskelijat kirjoittivat tehtävästä pohtivan raportin, jossa heidän piti vastata kohdassa 4.3.2 määriteltyihin kysymyksiin. Vastaukset ovat luettavissa tekstimuodossa Luvussa A .

Heidän tehtävien vastauksensa olivat osana kurssin arvostelua ylimääräisen, täydentävän tehtävän muodossa: opiskelijat pystyivät parantamaan arvosanaansa tai korvaamaan puuttuvan tehtävän vastaamalla tähän tehtävään. Kurssin suoritti noin kaksikymmentä opiskelijaa tänä vuonna. Vastauksia tehtävään tuli neljä kappaletta, joka oli jotakuinkin odotettu määrä tehtävän ollessa vapaaehtoinen tehtävä.

Tehtävien vastauksista sai hyvää palautetta kirjaston toimivuudesta ja parannusehdotuksia kirjastoon ja esimerkkipeliin. Kurssilta pyydettiin vastausten anonyymiyttä, joten numeroin vastaukset ja käsittelin niitä numeroin. Vastaajilta kysyttiin myös arviota ajasta, kuinka kauan he käyttivät tehtävän vastaamiseen, mutta suurin osa ei tähän vastannut. Jos vastausta ei tullut erillisen viestinkään jälkeen, en pyytänyt enää uudelleen aika-arviota. Laitoin kaikki vastaukset taulukkolaskentaohjelman taulukkoon vastausten käsittelemistä varten.

5.5.1 Analyysin valmistelu ja teemojen esittely

Totesin, että kappaleita kokonaisuudessaan on hyvin vaikeaa lähteä analysoimaan, joten päätin jäsennellä virkkeet niiden asiasisällön mukaan, jotta tekstiä olisi pienemmissä asiakokonaisuuksissa käsittelyä varten. Halusin "leikkiä" datalla, kuten Yin kehottaa tekemään, ja tämä tuntui myös olevan itselleni mielekkäin tapa lähestyä asiaa.

Jäsentelin virkkeitä niiden sisällön mukaan viiteen erilaiseen asiasisältöön: Ongelma, ratkaisu, ehdotus, pohdinta tai omia ajatuksia. Osa virkkeistä saattoi liittyä useaan osa-alueeseen, mutta lajittelin tälläiset virkkeet mielestäni mielekkäimpään kategoriaan. Muutamissa tilanteissa jaottelin myös yksittäisiä lauseita ajatuksien mukaan, mikäli virkkeessä oli selvästi kaksi eri asiasisällön sisältävää lausetta. Kaikkia

virkkeitä ei erikseen luokiteltu, mikäli virke pyrki kuljettamaan tekstiä eteenpäin tai oli otsikko- tai kysymysrivi tai ei muuten sisältänyt asiasisältöä. Kun tekstiä oli jäsennelty enemmän, oli helpompaa lähteä tutkimaan yksittäisten virkkeiden ajatuksia ja pohtia, mitä tämä virke kertoo.

Ongelmat olivat virkkeitä, jotka toivat esille ongelman joko kirjaston käytössä tai tehtävää vastaamisessa. Eli jonkunlainen pysähtymiskohta tai virhe ohjelman suorituksessa tai ratkaisun tekemisessä, joka ei ollut odotettu. Ongelmien analysoinnissa oli olennaista katsoa, johtuiko ongelma kirjaston toiminnallisuudesta tai sen puutteesta vai opiskelijan tekemästä ratkaisusta.

Ratkaisu-virkkeiksi luokittelin virkkeet, joissa tekijä kuvaili vaiheen tehtävän ratkaisemiseksi tai miten oli esimerkiksi ratkaissut aiemman ongelman.

Pohdinta-virkkeiksi luokittelin virkkeet, joissa tekijä oli esimerkiksi pohtinut erilaisten ratkaisujen mielekkyyttä tehtävän ratkaisun kannalta. Pohdinta virkkeet saattoivat käsitellä myös esimerkiksi kirjaston toimintaa. Pohdinta virkkeet olivat usein hyvin lähellä *omia ajatuksia*-virkkeitä, sillä usein pohdinnassa tuotiin myös esille omia ajatuksia ratkaisuun tai asioihin liittyen. Tälläisissä tilanteissa virke merkattiin pohdinta-virkkeeksi pohdinnallisen sisällön takia.

Omia ajatuksia -virkkeiksi keräsin vastaajan omia ajatuksia, jotka käsittelivät esimerkiksi tuntemuksia tai ajatuksia. Esimerkiksi seuraavat virkkeet luokittelin vastaajan omaksi ajatukseksi:

> Vastaus 1: Lähdin kokeilemaan modausta lähinnä tavoitteella, että saisin kaksipistettä. Myös aihe kiinnosti jonkin verran, niin ihan mielelläni ajattelin tehdä.

Kun virkkeet olivat luokiteltu erilaisiin ryhmiin, oli helpompaa lähteä tutustumaan tekstiin keskittyen esimerkiksi yhteen ryhmään kerrallaan. Oli helppoa lähteä ongelmista liikkeelle ja selvittää, mitkä asiat herättivät ongelmia vastaajilla. Tästä heräsi ajatus esimerkiksi siitä, että *esimerkkipeliin liittyi ongelmia*. Keräsin kaikki tähän ajatukseen liittyvät virkkeet kasaan ja lähdin tarkemmin tutkimaan, mitä erilaisia

teemoja näistä ajatuksista löytyi. Usein ajatukseen liittyen löytyi muutamia tarkempia teemoja, joihin pystyin keskittymään tarkemmin ja pohtimaan mahdollista ratkaisua tarvittaessa.

Pyrin varmentamaan raporttien esille tuomaa kerrontaa tarkastelemalla vastaajien palauttamia projekteja. Suuressa osassa tilanteita tämä tarkastelu ei tuonut mielekästä lisätietoa raportin lisäksi: jos vastaaja raportoi tehneensä jonkunlaisen ominaisuuden, ominaisuus myös löytyi itse koodista. Mainittakoon poikkeuksena vastaus 4. Vastauksen 4 raportissa vastaaja toi vahvasti esille epävarmuuttaan osaamisestaan, esimerkiksi lisälauseilla kuten "missä nyt ei varmasti ollut mitään järkeä.". Tämä epävarmuus ei kuitenkaan näkynyt itse koodin puolella: suurin osa ratkaisuista, joita tekijä epäili, olivat oikein toimivia.

Taulukossa 1 käsitellään löydetyt teemat, pohditaan syitä teemoihin ja listataan tarvittaessa, miten ongelmakohta ratkaistaan. Seuraavaksi käsittelen taulukossa esitetyt teemat järjestyksessä.

5.5.2 Esimerkkipelin ongelmat

Virkkeiden lajittelun jälkeen jäi selvästi mieleen yksi asiakokonaisuus: esimerkkipeliin liittyvät ongelmakohdat. Näistä oli mielestäni helpointa lähteä liikkelle. Nämä tulivat eniten ongelma-virkkeissä esille ja joissain pohdinta-virkkeissä. Ratkaisuosiossa taas osa vastaajista oli saanut kierrettyä esimerkissä löytyneen ongelman.

Esimerkkipeliin liittyen oli selvästi kaksi ongelmaa: *kontrollerin päivitys* ja *pelin lisääminen rinnakkaisiksi*. Ensimmäinen ongelma, kontrollerin päivitys on selvästi omaa huolimattomuuttani jäänyt vain laittamatta niin, että oikeaa kontrolleria kutsuttaisiin päivityskutsussa esimerkkipelin koodissa. Esimerkiksi vastauksen 2 raportissa oli kerrottu, miten ongelma oli saatu ratkaistua ja tämä näkyi myös toimivan itse projektin puolella. Toinen ongelma taas liittyi enemmän tehtävän ohjeistukseen: Ohjeistuksesta sai käsityksen, että yhteen tiedostoon tulee lisätä pelit, kun itse kirjaston puolella toiminnallisuus oli muutettu niin, että jokainen peli luetaan omasta tiedostostaan tietyn kansion sisältä.

Teema	Pohdinta	Ratkaisu
Esimerkkipelin ongelmat		
Kontrollerin päivitys	Kontrollerin päivitys unohtui	Kutsutaan oikeaa kontrolleria koodissa
Pelien lisääminen rinnakkaisiksi	Dokumentaatio oli epäselvää	Esimerkkipelin dokumentaatio selkeämmäksi
Tehtävän mielekkyys		
Kirjaston ja esimerkin rajan epäselvyys	Mikä on kirjastoa, mikä esimerkkipeliä?	Useampi esimerkkipeli tapaustutkimukseen
Modauksen autenttisuus	Valmiimpi pelikokemus olisi ollut aidompi	Kurssilla kannattaa käyttää valmista peliä, joka löytyy markkinoilta
Tehtävän mielenkiintoisuus yleisesti	Osa vastauksista tyytyväisiä tehtävään kokonaisuutena	Ei toimenpiteitä
Virheviestien mielekkyys		
Luan virheviestit sekavia	MoonSharp ei kerro tarpeeksi virheestä	Parannetaan kirjaston virheiden käsittelyä
Kirjastoon liittyvä pohdinta		
Ongelma äänien latauksessa	Äänien latauksessa tiedostopolku väärin	Korjataan äänien latauksen polku
Kenttäeditori	Helpottaisi kenttien luomista peleihin	Lisätään tuki kirjastoon
Omaksuttavuus ja helppokäyttöisyys	Esimerkkikoodit hyviä ja yleisesti helposti lähestyttävä	Ei toimenpiteitä
Jäljelle jääneet ehdotukset		
Esimerkkipeleihin liittyviä	Ei mielekkäitä ratkaista erikseen	Korjaantuvat muiden korjausten mukana

Taulukko 1. Löydetyt teemat, ehdotukset ja niihin liittyvät ratkaisut

5.5.3 Tehtävän mielekkyys

Toinen selkeästi esille noussut ongelma oli *tehtävän mielekkyys*. Se nousi useissa vastauksissa esille ja siihen liittyvät virkkeet olivat helppoa kerätä kasaan aineistosta. Tarkemmin tähän asiakokonaisuuteen liittyvien vastauksien analysoinnin jälkeen nousi kaksi ongelmakohtaa esille: Vastaajille oli vaikeuksia erottaa itse kirjaston ja esimerkin rajaa ja tehtävä ei korostanut modausta tarpeeksi aidossa ympäristössä. Kuitenkin osa vastaajista oli tyytyväisiä tehtävään kokonaisuutena.

Kaikki vastaajat olivat sitä mieltä, että tehtävä ei ollut paras mahdollinen tehtävä tuomaan esille *modausta peleissä autenttisella tavalla*. Tehtävässä oli tarkoitus tuoda esille modauksen sallimista pelinkehittäjän ja käyttäjän välisenä kommunikaationa, mutta vastaajat olisivat ilmeisimminkin kaivannut tehtävää, jossa muokataan valmista kokonaisuutta. Tehtävän keinotekoisuus tuntui myös haitanneen vastaajia esimerkiksi vastauksissa:

> Vastaus 3: Tehtävää olisi ehkä voinut muokata niin että Unity projektissa olisi selkeästi tehty valmiimpi peli kokonaisuus jota muokata. Nyt kaikki oli käytännössä tehty Lualla, joten todellisen modauksen tuntua ei tullut. Enemmän tehtävä tuntui valmiiden asettien käyttämiseltä Luassa.

ja

> Vastaus 1: Ehkä itselläni enemmänkin tosiaankin enemmän pelin tekemistä pelimoottorin avulla kuin modaamista tai ainakin tällainen tunne itselleni jäi.

Ratkaisu tähän ongelmaan liittyy enemmänkin kurssin tehtävään. Tapaustutkimuksessa käytetty tehtävä ei ollut niin mielekäs modausta esittelevänä tehtävänä kuin se olisi voinut olla, joten sen voisi korvata "oikean" pelin muokkaukseen liittyvällä tehtävällä.

En ole täysin varma, kuinka *esimerkkipelin rajaa* pystyisi paremmin tuomaan esille. Samalla myös vaikutti, että osa vastaajista oli ymmärtänyt kirjaston ja esimerkin eron ja osalle se oli jäänyt enemmän epäselväksi. Esimerkiksi vastauksessa 3 oli mainittu, että "Kirjaston käyttö oli pohjimmiltaan helppoa.", mikä antaisi kuvan, että te-

kijä on ymmärtänyt kirjaston ja esimerkin eron, kun vastauksessa 1 tuotiin selvästi esille ongelma kirjaston ja esimerkin eron ymmärtämiseksi: "En tiedä tarkkaan, mikä osa oli varsinaisesti kirjastoa ja mikä esimerkkikoodia.". Jonkunlainen rakenne kaavio voisi tuoda tarkemmin tätä eroavaisuutta esille myös, mutta uskon useamman esimerkkipelin tarjoavan vertailukohdan siitä, mikä on itse kirjasto. Tämä on varsinkin samankaltaisia tutkimuksia varten mielekäs idea tai mikäli kirjastoa halutaan tutkia uudestaan jatkossa. On kuitenkin harmillista, että tapaustutkimuksen aikana oli epäselvyyttä siitä, mikä on itse kirjasto, sillä 1:n vastauksissa oli hyviä ehdotuksia, mutta suurin osa näistä koski esimerkkipeliä ja sen rakennetta.

Tehtävän mielekkyyteen liittyen oli vielä ylipäätään keskustelua *tehtävän mielenkiintoisuudesta kokonaisuutena*. Näissäkin mielipiteet olivat jakautuneet kahtia: vastauksissa 1 ja 3 jäi positiivinen vaikutelma tehtävästä kokonaisuutena esimerkiksi vastauksessa yksi todettiin lopuksi, että tehtävää oli kaikenkaikkiaan ihan mukavaa tehdä, kun vastauksissa 2 ja 4 tehtävä ei niinkään nostanut innostusta modaukseen liittyen:

> Vastaus 2: Tehtävä ei esimerkkipelin ja sen pohjalta luomani oman versioni simppelihköyden vuoksi ehkä niinkään nostanut erityistä lisäkiinnostusta modaukseen, mahdollisesti jonkun hiukan 'hienomman' ja/tai monipuolisemman pelin grafiikoiden ym. muutteleminen saattaisi ajaa asian tässä suhteessa paremmin..

5.5.4 Virheviestien mielekkyys

Parissa vastauksista esitettiin, että *MoonSharpin tai Luan virheviestejä oli hyvin vaikeaa ymmärtää*. Tämä tuli esille vastauksissa 3 ja 4 esimerkiksi seuraavalla tavalla "Yritin tehdä globaaleja muuttujia näistä, mutta MoonSharp herjasi jatkuvasti, usein varsin hyödyttömillä virheilmoituksilla." (Vastaus 4). Virheviestit saattavat olla hyvin hankalia ymmärtää varsinkin, jos on tottunut esimerkiksi Visual Studio-kehitysympäristön tarkkoihin ilmoituksiin tietyllä rivillä.

5.5.5 Kirjastoon liittyvä pohdinta

Vastaajat olivat pohtineet myös kirjaston toimintaa vastauksissaan. Varsinkin vastauksessa kaksi perehdyttiin selvästi kirjastossa löytyneeseen ongelmaan: *äänien lataus ei toiminut niin kuin sen kuuluisi*. Äänitiedostojen lataus projektin ulkopuolelta toimi sattumanvaraisesti ja useimmiten se ei toiminut ollenkaan. Vastauksessa kaksi ehdotettiin myös mahdollisuutta *suunnitella kenttiä kenttäeditorilla*. Nämä molemmat olivat tärkeitä huomioita kirjaston kannalta.

Suurimmassa osassa vastauksia kirjaston käyttö vaikutti onnistuneen. Myös vastauksessa 1 oli korostettu yksittäisiä kirjaston ominaisuuksia toimivina, vaikka muuten vastauksessa 1 saattoi olla epäselvyyttä kirjaston ja esimerkin rajasta. Varsinkin koodiesimerkeistä tuli hyvää palautetta. Ne helpottivat kirjaston käyttöönottoa:

> Vastaus 1: Myös ohjeistus oli mielestäni oikein hyvä ja selitti todellakin asioita, niin ei tarvinnut arvailla. Myös esimerkit olivat todellakin auttavia.

Pohdintaa oli myös esimerkiksi mallin mielekkyydestä ja siitä, että ilman mitään dokumentaatiota, kirjastoa saattaisi olla vaikea käyttää - mikä ei mielestäni tässä ole tavoitteenakaan.

> Vastaus 2: Kirjaston toiminnallisuuden hahmottaminen erityisesti LUA- ja C#-puolen välisen kommunikaation lainalaisuuksien osalta vaati hiukan pähkäilemistä, mutta oli muuten melko selkeää. Toisaalta helppohan tuota oli muokata aiemman pohjalta mallia katsoen - sen verran monimutkaiselta kirjaston toiminnallisuus kuitenkin vaikutti että uuden, vastaavan pelin luominen tyhjästä olisi takuulla paljon haasteellisempaa.

5.5.6 Jäljelle jääneet ehdotukset

Suurin osa ehdotuksista tuli esille jo jonkun muun teeman alla, mutta koin mielekkääksi vielä kerätä jäljelle jääneet ehdotus-virkkeet ja tarkistaa, ettei niistä löydy jotain uusia ehdotuksia. Vastauksen 1 ehdotukset liittyivät suurimmalta osalta esimerkkipelin rakenteeseen, eikä niinkään kirjastoon. Nämä ehdotukset tulevat täy-

tetyiksi *esimerkkipeliin liittyvien ongelmakohtien korjausten yhteydessä*. Vastauksessa 4 ehdotettiin myös seuraavaa "Kaiketi hyvä dokumentaatio tai jokin kaavio eri tiedostojen välisestä suhteesta voisi selkiyttää toimintaa.". Jonkunlainen kaavio voisi selkeyttää esimerkkipelin toimintaa, mutta mielestäni olisi mielekkäämpää pyrkiä kirjaston toiminnallisuuden ja rakenteen esille tuomiseen, eikä esimerkkipelien rakenteen esille tuomiseen. Ylipäätänsä molemmat ehdotukset koskivat enimmäkseen esimerkkipeliä eikä niinkään itse kirjastoa. Tämä johtunee siitä, että myös vastauksissa 1 ja 4 oli eniten vaikeuksia erottaa esimerkkipeli ja kirjasto toisistaan.

5.5.7 Korjaukset

Korjasin esimerkkipeliin liittyneet kontrolleri ongelmat: tein yhden muuttujan, jotta kaikkialla luokassa pystytään kutsumaan oikeaa kontrolleria. Kurssin tehtävänanto ja kurssilla käytetty tehtävä eivät kosketa tätä tutkimusta, mutta nämä voidaan ottaa huomioon kurssilla ja muussa opetuksessa jatkossa.

MoonSharp tarjoaa kattavat työkalut debuggaukseen, mutta ne pitäisi ottaa erikseen käyttöön koodin puolelta, joten ei ole mielekästä pitää niitä oletuksena päällä. Kuitenkin itse virheiden käsittely skriptien suhteen kirjastossa oli olematonta. Skriptitulkkiin lisättiin kaikkiin virheisiin erillinen käsittely ja tulostettiin tarkempi viesti virheestä Unityn *Debug*-logiin, jotta virhe olisi vielä selkeämpi kuin aiemmin. Testatuissa tilanteissa itse virheen viesti oli jo yhtä tarkka kuin lisätieto, mutta ehkä jonkun virheen kohdalla tarkempi viesti voi tuoda lisätietoa virheen aiheuttaneesta koodin pätkästä.

Äänien latauksessa oli jäänyt määrittelemättä tiedostopolku kokonaisena. Ääniä yritettiin ladata suhteellisesta polusta, mikä ei toiminut ollenkaan tavan kanssa, jolla Unity oli laitettu äänet lataamaan. Mielenkiintoisinta oli, että ratkaisu oli kuitenkin testatessa toiminut jollain tavalla, mikä Unityn dokumentaation mukaan ei olisi pitänyt olla alkuunkaan mahdollista. Esimerkkiprojektin ei olisi koskaan pitänyt ääntä soittaakkaan, mutta testatessa se kuitenkin on taustamusiikkia soittanut. Onkin mielenkiintoista, miksi ratkaisu on alunperin toiminyt ja nyt ei toiminut. It-

se ongelma oli hyvinkin helppo korjata, kun se oli löytynyt. Polun korjaus ratkaisi äänien lataukseen liittyvän ongelman. Myös toisessa asynkronisessa latauksessa oli samankaltainen ongelma koodinpuolella, jonka korjasin samalla.

Yksi ehdotetuista ominaisuuksista vastauksissa oli, että kirjastolla pystyisi luomaan myös kenttiä ilman, että sitä täytyy tehdä Lua-koodin avulla. Tämä toiminnallisuus lisättiin kirjastoon. Kirjasto osaa tehdä kentän JSON-tiedostosta, jonka muoto on samankaltainen kuin Tiled-kenttäeditorin[5] JSON-muotoisena tallennettu kenttätiedosto. En ole tietoinen yleisemmästä formaatista, missä kenttiä yleisesti tallennettaisiin, ja Tiled-editorin kenttätiedosto on hyvinkin yleisluontoinen, mikäli ei toteuta jokaista ominaisuutta kenttään liittyen. Kenttäeditoria on myös mahdollista vaihtaa samalla tavoin kuin muita ominaisuuksia kirjastossa käyttämällä Unityn komponentti-rakennetta. Tähän liittyen toteutin vielä toiminnallisuuden Unityn Scenen luomiseksi tiedoston perusteella editorin puolella. Tämä sallii kentän tallentamisen ja muokkauksen pohjatiedoston kääntämisen jälkeen. Kentän voi luoda pohjan avulla ja viimeistellä Unityn editorin puolella.

Esimerkkipeli Syön Sut päivitettiin ajan tasalle ja testattiin, että peli toimii edelleen muutoksien jälkeen. Varmistin myös, että peliin liittyvät ongelmakohdat olivat kunnossa ja esimerkiksi äänien lataus toimi nyt ongelmitta.

5.5.8 Tapaustutkimukseen liittyvää pohdintaa

Seuraavaksi esittelen pohdintaa tapaustutkimukseen liittyen. Tuon esille Yinin pohdintaa tapaustutkimuksen mielekkyydestä kirjallisuudessa yleisesti ja vertaan sitä edellä esitettyyn tapaustutkimukseen. Myös Zelkowitz ja Wallace (1998) pohtivat mahdollisuutta toteuttaa tapaustutkimuksen sijaan kontrolloitu tutkimus.

Yin (2014, s.19-22) käsittelee yleisesti erilaisia huolia, joita voi liittyä tapaustutkimukseen ja sen ymmärrykseen. Esimerkiksi yleisesti epäillään tapaustutkimuksen

5. Tiled-kenttäeditori on kenttäeditori kaksiuloitteisia pelejä varten, jossa pystytään suoraan muodostamaan kenttä käyttämällä kuvia. Editorista tarkemmin löytyy osoitteesta `http://www.mapeditor.org/`

täsmällisyyttä, yleistettävyyttä, työn määrää ja vertailtavuutta.[6]

Toin kaikki tapaustutkimukseen liittyvät yksityiskohdat mahdollisimman tarkasti esille, jotta tutkimus olisi täsmällistä ja myös muiden toistettavissa. Yin (2014, s.20-21) argumentoi, että tapaustutkimuksessa ei pyritä tilastolliseen yleistettävyyteen vaan siinä laajennettaan ja yleistetään teoriaa. Tämä poikkeaa suunnittelutieteen näkökulmasta, sillä siinä artefaktien toimivuuden osoittaminen on etusijalla. Samasta syystä tapaustutkimuksen yleistettävyys ei ollut tämän tutkimuksen valossa suuri huolenaihe. Mielestäni aina voidaan löytää parannettavaa kirjastosta, mutta tapaustutkimuksella sain esille yksittäisiä ongelmia, jotka oli hyvä korjata. Tapaustutkimuksen uusiminen toistaminen parantaisi kirjaston laatua entisestään, joten sen tehtävä ei niinkään ollut tarjota yleistettävää teoriaa vaan tukea suunnittelutieteellistä tutkimusta.

Yinin (2014) mukaan huolta on ollut tapaustutkimuksen laajuuteen liittyen. Hän selventää, ettei tapaustutkimuksen tarvitse olla pitkä, vaan tämä käsitys liittyy yleisemmin tapaustutkimuksiin, joissa käytetään ethnograafista tutkimusta tai osallistuvaa havainnointia. Hän jatkaa, että tapaustutkimuksen pystyy tekemään poistumatta puhelimen tai internetin äärestä, riippuen tutkittavasta aiheesta. Hän (s.21) korostaa, että data on hyvä esittää niin, että siihen on helppoa tutustua.

Yinin (2014, s.21-22) mukaan kirjallisuudessa on kyseenalaistettu, onko tapaustutkimus paras tapa tutkia asioita verrattuna muihin tutkimustyyleihin (tapaustutkimuksen *comparative advantage*). Tässä tutkimuksessa tullaan käyttämään muitakin arviointitapoja täydentämään kirjaston arviointia, joten en itse ota kantaa tapaustutkimuksen asemaan tutkimuksen teossa. Tarkennettakoon kuitenkin, että Yin argumentoi vahvasti, miksi tapaustutkimukset ovat hyödyllisiä (ks. Yin 2014, s. 21-22).

Yinin (2014) mukaan anonyymiyttä pitäisi pitää vähemmän toivottuna vaihtoehtona tapaustutkimuksessa, mutta kurssin puitteissa anonyymiys luvattiin kurssilaisille.

6. Yin (2014, s.20) Käsittelee myös sitä, että joskus tapaustutkimustiedettä sekoitetaan opetustapauksiin (*teaching case*), jossa dataa muokataan esimerkkiin sopivaksi. En ole itse törmännyt tälläiseen ongelmaan, eikä se ole merkityksellinen tämän tutkimuksen kannalta.

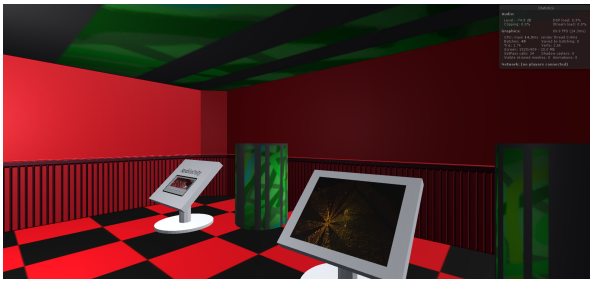
Tässä tutkimuksessa kurssi oli kerrottu nimeltä, mutta vastaajien nimiä ei. Yin varoittaa anonyymiyden toteuttamisessa olevan paljon työtä. Materiaalia oli kuitenkin niin vähän, että anonymiteetin toteuttaminen ei teettänyt erityisen paljoa työtä.

Tapaustutkimuksen asetelmassa oli myös joitakin kontrolloidun tutkimuksen piirteitä. Esimerkiksi Zelkowitz ja Wallace (1998) määrittelevät yhdeksi kontrolloidun tutkimuksen tavaksi synteettisen ympäristön kokeen, jossa tutkitaan jotain yksittäistä asiakokonaisuutta järjestelmän rakenteessa tai käytössä. Heidän koeasetelmassa noin kahdestakymmenestä kolmeenkymmeneen henkilöä ratkoo ongelmaa ennalta määritellyn ajan. Hevner ym. (2004) määrittelevät kontrolloidun kokeen etsivän laadullisia ominaisuuksia kontrolloidussa ympäristössä. Peliteknologia-kurssilla työskentely on hyvin itsenäistä ja yhteen tehtävään varattu aika on noin kahdeksan tuntia. Kahdekan tuntia olisi pitkä aika yhtäjaksoiselle kokeelle, varsinkin kun vastaajien täytyisi perehtyä osaan tehtävässä käytetyistä teknologioista etukäteen.

Zelkowitz ja Wallace (1998) luonnehtivat yleisesti kontrolloidun tutkimuksen vaativan useita toistoja ja heidän esimerkeissään kaikissa kontrolloiduissa tutkimuksissa oli kontrolliryhmä. Jos osa vastaajista tekisivät tehtävän käyttämättä työkalua kontrolliryhmänä, olisi se erittäin ongelmallista kurssin kannalta: kaikilla kurssilaisilla pitäisi olla samanlainen arvosteltava tehtävä. Ongelma oli myös, että opiskelijat saivat valita tehtävät, jotka he tekevät kurssilla. Oli siis mahdollista, että vastauksia tehtävään tulisi vähän, kuten kävikin. Zelkowitz ja Wallace (1998) ehdottavat, että toistetussa kokeessa tulisi olla kahdestakymmenestä neljäänkymmeneen osallistujaa, jotta tulokset olisivat tilastollisesti merkittäviä. Tällöin vain neljän henkilön vastaaminen tehtävään olisi antanut kyseenalaisen tilastollisen merkittävyyden. Ja vielä lopuksi, kuten tutkijat itsekkin pohtivat, tutkimuksiin voidaan tehdä muutoksia, mutta lopulta muutosten seurauksena kontrolloitu tutkimus muuttuu jonkunlaiseksi variaatioksi tapaustutkimuksesta.

5.6 Esimerkkipeli Curriculum Vitae (Vaihe 7)

Kirjastoon tehtyjen muokkauksien jälkeen rakennettiin 3D-seikkailupeli, jossa on museomainen teema. Halusin toteuttaa yksinkertaisen portfolio-pelin, jossa pelaaja voi lukea omasta osaamisestani pelikehitykseen liittyen. Esimerkkipelissä keskityttiin testaamaan varsinkin uusia ominaisuuksia, joita lisättiin tapaustutkimuksen johdosta. Esimerkiksi pelin kentät luotiin käyttäen JSON-tiedoston pohjalta määriteltyä kenttää. Kuviossa 3 näkyy yhdestä huoneesta kaksi esittelyikkunaa.



Kuvio 3. Näkymä pelistä, jossa on useamman projektin esittelevät näyteikkunat

Halusin tehdä pieniä muutoksia vielä itse kentän rakenteeseen sen luomisen jälkeen, joten kentän ajonaikaisen luomisen sijaan käytin kirjastoon lisättyä toiminnallisuutta Unity näyttämön (*scene*) luomiseksi. Tämä ominaisuus salli tehdä koko pohjakentän hyvinkin nopeasti Tiled-editorilla valmiiksi, mutta tehdä hienosäätöä vielä lopuksi osalle peliolioista. Esimerkiksi tiedostot, joista keskustelut luetaan erilaisia näyteikkunoita varten, pystyi määrittämään näyttämöä varten valmiiksi jälkikäteen. Tämä olisi ollut työläämpi toteuttaa Tiled-editorin avulla, sillä silloin jokaiselle esittelyikkunalle olisi pitänyt määrittää kaikki tiedot jo luomisvaiheessa. Jälkikäteen muokkaus salli myös ikkunoiden kuvituksen vaihtamisen oikeanlaiseksi.

Kuviossa 4 näkyy esimerkki keskusteluikkunasta, jossa on auki keskustelu. Keskustelun tekstit ja niihin liittyvät vastaukset ovat tallennettuna JSON-tiedostoihin, jotka ladataan kirjaston avulla. Yksinkertaisen keskustelumaisen tekstin esittämiseen JSON toimi oikein hyvin.

Tässä esimerkkipelissä käytettiin myös SingletonModsInUnity-komponenttia kirjastoon viittaamisessa apuna, jotta nähtiin, toimiiko se oikein. Yksittäisestä telineen

Kuvio 4. Keskustelu käynnissä pelissä

kontrollerista täytyi päästä käsiksi kirjaston toiminnallisuuteen, jotta keskustelut saatiin ladattua. Ominaisuus toimi ongelmitta.

Esimerkkipeliä tehdessä huomasin yhden tärkeän ominaisuuden puuttuvan kentän luomisesta. Jotta vältyin luomasta aliohjelmia idealla "TeeOikeaSeina", halusin tietää annetun datan numeron aliohjelmassa, joka luo seiniä. Numeron avulla pystyin kääntämään esimerkiksi oikeita seiniä 90-astetta, ilman tarvetta aliohjelmille "LuoOikeaSeina" ja "LuoSuoraSeina".

Ominaisuuksien toteuttamisen jälkeen testasin vielä projektin suorituskykyä ajamalla pelejä editorilla ja tarkkailemalla pelin kuvanopeutta. Kirjaston suorituskyvyssä ei havaittu ongelmia ensimmäisen esimerkkipelin suorituksessa, vaikka kirjasto saattaakin hidastaa ohjelman suoritusta jossain määrin. Unityn päivityssyklissä skriptien käsittely onnistui ilman kuvien piirtonopeuden putoamista alle 60:n kuvan sekuntivauhtia. Peli pyöri noin 1000 kuvaa sekunnissa unityn *profilerin*[7] tilastojen mukaan läppärilläni, jossa on Intel Core i5-6300HQ CPU 2.30GHz:n taajuudella, 8 gigatavua muistia ja Nvidia GeForce GTX 960M näyttösovittin. Mainittakoon, että päivitysnopeus putoaa hetkellisesti alle 60 kuvaa sekunnissa, kun pelin käyn-

7. Profiler on Unityn tarjoama työkalu pelin suorituskyvyn tarkkailuun. Sillä pystyy esimerkiksi näkemään, mitkä skriptit syövät eniten aikaa päivityssyklissä.

nistää uudelleen, jolloin Unity lataa kaiken kenttään liittyvän uudestaan ja skriptit lataavat pelissä käytettävät kuvat ja musiikin ja skriptit. Toisessa esimerkkipelissä, Curriculum Vitaessa, oli myös yli 60:n kuvan päivitysnopeus, mutta kyseisessä esimerkkipelissä skriptejä ei käytetty samalla tavoin päivityssyklissä kuin ensimmäisessä pelissä käytettiin. Peli oli myös 3D-peli, joten näiden pelien suoritusnopeuden keskenään vertailu ei ole mielekästä.

6 Pohdinta ja jatkotutkimus

Tässä tutkimuksessa kehitin suunnittelutieteellisen artefaktin: kirjaston, joka helpottaa käyttäjälähtöistä muokattavuutta. Kehitin kirjastoa usean syklin ajan, mikä vastaa suunnittelutieteen vaatimuksia (Hevner ym. 2004). Tutkimuksessa oli kolme erillistä *rakenna ja arvioi* -sykliä. Nämä syklit olivat arvioinnin mukaan jaoteltuina prototyyppinä ja skenaariona toiminut esimerkkipeli Syö mut, tapaustutkimus Peliteknologia-kurssilla ja prototyyppinä toiminut esimerkkipeli Curriculum Vitae. Jokainen sykli toi kirjastoa lähemmäksi haluttuja toiminnallisuuksia ja tapaustutkimuksen puitteissa löytyi parannettavia ja lisättäviä ominaisuuksia kirjastoon. Toisen esimerkkipelin kehittäminen oli tehokasta kirjaston avulla, sillä kenttäeditori ja valmis JSON-tulkki nopeuttivat työtä.

Kirjastoa voidaan käyttää apuna myös projekteissa, joissa käyttäjälähtöinen muokattavuus ei ole ensisijaisena tavoitteena. Tällöin projektissa voidaan sallia käyttäjälähtöinen muokattavuus myöhemmin, kunhan projektissa on käytetty kirjastoa muokattavaksi sallittavien skriptien ja datan käsittelyyn. Kuten aiemmin käsittelin luvussa 2.4, käyttäjälähtöisen muokattavuuden salliminen on kannattavaa peliprojekteille.

Olen ottanut kirjaston laadulliset vaatimukset kehityksessä huomioon. *Omaksuttavuutta* ja *käytettävyyttä* on vaikeaa mitata numeerisesti, mutta mielestäni tapaustutkimus osoitti, että ulkopuoliset pystyvät käyttämään kirjastoa esimerkkipohjaa muokkaamalla. Osa vastaajista oli tuonut esille kirjaston käytön olleen helppoa, ja kirjaston yksittäisiä toiminnallisuuksia oli myös kehuttu toimiviksi. *Yleistettävyyden* otin huomioon kirjaston suunnitteluvaiheessa käyttämällä komponenttimallia. *Suorituskykyä* testasin kirjaston viimeisen vaiheen lopussa eikä siitä löytynyt ongelmia. *Turvallisuuden* otin huomioon pakottamalla kirjaston avulla kehittävän kehittäjän määrittelemään tyypit, joita Luan puolelta voidaan kutsua.

Yleistettävyys kirjastossa toimi mielekkäästi, eikä kirjaston toiminnallisuuden muuttaminen vaadi muutoksia koodin osalta. Komponenttimalli sallii eri työkalujen käyt-

töönoton toteuttamalla rajapintaa vastaavan luokkakokonaisuuden ja vaihtamalla sen Unity-editorissa. Myös pienemmät kokonaisuudet ovat toteutettu samalla ideella. Esimerkiksi kenttäeditori on toteutettu vaihdettavaksi komponenttia vaihtamalla.

Kuten Luvussa 5.6 totesin, ei esimerkkipelien *suorituskyvyssä* ollut merkittäviä ongelmia. Testissä ensimmäisen esimerkkipelin suoritusnopeus oli noin tuhat kuvaa sekunnissa, kun esimerkiksi tyhjällä näyttämöllä se oli 2000-2700 kuvaa sekunnissa. Tälläisten päivitysnopeuksien suora vertailu ei ole erityisen mielekästä ilman tarkkaan suunniteltua testiympäristöä. Esimerkiksi koodin syväprofiloinnin ottaminen mukaan tyhjässä skenessä pudotti päivitysnopeuden alle kahteen tuhanteen kuvaan sekunnissa, mikä johtui vain profiloinnin raskauden kasvamisesta. Testillä voitiin kuitenkin todeta hyvin, ettei ongelmia suoritusnopeuden suhteen ollut.

Peli ei merkittävästi hidastunut esimerkiksi niin, että päivityssykli olisi pudonnut alle kuudenkymmenen kuvan sekuntivauhdin. Käyttämäni tietokone oli suhteellisen tehokas testaukseen, mutta hitaat ratkaisut pystyvät hidastamaan päivitysnopeuden käyttämälläni tietokoneella liian alhaiseksi. Esimerkiksi jatkuva peliolioiden alustus ja tuhoaminen Unityssä on hyvin raskas toimenpide, joka tähän pystyy.

Kirjastossa on otettu huomioon siinä käytettävien modien *turvallisuus* skriptien yhteydessä. MoonSharp-kirjastolla, joka sallii Luan ajamisen C#-ympäristössä, pitää erikseen sallia, mitä tyyppejä Lua-koodissa voi käyttää. Mikäli kehittäjä jakaa vain pelin kannalta olennaiset tyypit eikä päästä modaajia Lua-koodin kautta käsittelemään tiedostojen latausta, ei skriptien pitäisi aiheuttaa tietoturvaongelmia. Tässä vastuu kuitenkin jää kirjastoa käyttävälle pelinkehittäjälle, sillä kehittäjä voi edelleen jakaa `IO`-metodeja Luan puolelle, joiden avulla olisi mahdollista tehdä haittaa koneen haltijalle.

Tapaustutkimukseen tuli suhteellisen pieni määrä vastauksia. En näe tämän olevan ongelma, sillä palautetuista vastauksista sai hyviä kehitysideoita ja löytyi muutamia korjattavissa olevia ongelmakohtia. Kuitenkin isommalla otannalla olisi voinut tulla enemmän erilaisia parannusehdotuksia kirjastolle tai löytynyt joitain ongelmia,

joita ei tällä kertaa löydetty. Kirjaston kehittyessä olisi mielekästä myös testata sitä uudestaan isommalla otannalla ja/tai pidempiaikaisesti. Kannattaisi myös pyrkiä tuomaan paremmin esille kirjaston ja sen päälle tehdyn esimerkkipelin raja, minkä uskon tapahtuvan kahden esimerkin avulla. Kahdesta esimerkistä pystyisi jo paremmin havaitsemaan, mitkä osat suorituksesta ovat kirjaston vastuulla ja mitkä ovat osa esimerkkejä.

Kirjasto pystyy luomaan kenttiä Tiled-kenttäeditorin avulla. Jatkossa voisi vielä tutkia, miten kenttäeditorin avulla pystyisi määrittämään parametreja kirjaston aliohjelmille. Toki kenttäeditorin luomaa JSON-tiedostoa muokkaamalla mikä vaan haluttu toiminnallisuus on mahdollista toteuttaa, mutta editorin kanssa toiminen olisi hyvä pitää yksinkertaisena: edellä mainittu lisäisi yhden välivaiheen lisää kenttien luomiselle.

Toinen kehitystä vaativa toiminnallisuus on näyttämön luomiseen käytetty ikkuna kenttäeditorissa. Ikkuna toimii kohtalaisesti, mutta on ulkonäöltään vaikea käyttöinen ja se unohtaa sille annetut tiedot kentän luomisen jälkeen tai näyttämöitä vaihtaessa. Ikkunaa voitaisiin kehittää tallentamalla tiedot esimerkiksi Unityn editorin välimuistiin.

Unity tarjoaa oman tulkkinsa JSON-tiedostojen käsittelylle ja tätä tulkkia käytetään kirjastossa. Unityn tulkki on siitä hankala, että sen toiminnallisuutta varten dataluokat tarvitsee määritellä tietyllä tyylillä, jotta Unity osaa luoda olioita käyttäen JSON-dataa. Seuraavaksi on esimerkki yhdestä yksinkertaisesta dataluokasta:

```
 [Serializable]
public class CharacterPackage
{
  public Character[] characters;
}
```

Nykyisessä toteutuksessa dataluokat täytyy määritellä omiksi luokikseen etukäteen ja dataluokkien attribuuttien pitää olla julkisia. Itse en mielelläni käytä julkisia taulukoita dataluokissa vaan käsittelen niitä C#-ohjelmointikielen *ominaisuuksien* avul-

la. Toki taulukkoa voi käsitellä luomalla sille ominaisuuden, mutta luokassa on silti julkinen viite taulukkoon. Myös datan sisältäessä paljon sisäkkäisiä olioita tarvitsee tehdä valtavia määriä luokkia. Ratkaisu tähän voisi olla geneerinen luokka, jonka avulla pystyisi ilmaisemaan paremmin luokkarakenteita. Toteutuksessa voisi käyttää myös toisenlaista JSON-tulkkia, mutta ulkopuolisen tulkin toiminnasta Unityn kanssa ei ole takeita.

Nykyinen ratkaisu toimii kuitenkin ongelmitta ja vaikuttaa olevan yleinen tapa, miten JSON-kääntäjät toimivat C#-ohjelmointikielessä Unityn kanssa (*Unity Forums - Howto use JSON in 5.3 [Guide]* 2015). Kuten Hevner ym. (2004) toteavat, täydellisen ratkaisun etsiminen teettäisi valtavasti työtä, joten toimivan ratkaisun löytäminen riittää. Käyttämäni ratkaisu on linjassa muiden samankaltaisten ratkaisujen kanssa, mikä vastaa Hevnerin ja muiden (2004) määrittelemään tarpeeseen verrata omaa ratkaisua jo valmiina oleviin ratkaisuihin. Nykyisen ratkaisun muuttaminen säästäisi kehittäjän aikaa luokkien määrittelyssä ja sallisi enemmän vapautta luokan rakenteeseen. Kuitenkin palvelut, kuten *json2csharp*[1], tekevät paremman ratkaisun toteuttamisesta tarpeetonta. C#-ohjelmointikieli sallii käyttää *dynamic*-avainsanaa ongelman ratkaisemiseen, mutta tämä ominaisuus on vasta versiossa 4.0,[2] kun Unity käyttää versiota 3.5.[3] Mikäli Unity tukisi tätä toiminnallisuutta, ei väliluokkia tarvitsisi määritellä, kun kääntäjä määrittäisi luokat ajonaikaisesti.

Tämän tutkimuksen puitteissa en pystynyt pureutumaan yhteen kirjaston kannalta mielenkiintoiseen mahdollisuuteen: kuinka Steam Workshopia voidaan käyttää kirjaston kanssa. Yhteensopivuuden toteuttamiseksi pitäisi saada tarkemmin tietoa Steam Workshopin rakenteesta ja siitä, asettaako se joitain oletuksia pelille tai sen modien lataukselle. Suora yhteensopivuus kauppapaikan kanssa olisi mielekäs omi-

1. Palvelu tekee valmiiksi C#-luokat annetun JSON-datan perusteella. Palvelu löytyy osoitteesta `http://json2csharp.com/`.

2. ks. esim. `https://www.codeproject.com/Tips/460614/Difference-between-var-and-dynamic-in-Csharp`

3. ks. esim. keskustelut `https://forum.unity3d.com/threads/unity-c-s-version.355757/` ja `http://answers.unity3d.com/questions/1305565/as-of-12017-what-is-the-highest-version-of-net-tha.html`.

naisuus kirjastolle.

Olisi myös mielenkiintoista testata esimerkkipelin kaltaisen pelin suorituskykyä, kun se on toteutettu ilman kirjastoa, käyttäen Unityn “normaalia” ohjelmointitapaa. Tätä voitaisiin verrata kirjastolla toteutettuun esimerkkipeliin. Näin pystyttäisiin tarkemmin mittaamaan kirjaston ja esimerkiksi skriptimoottorin aiheuttamaa suorituskyvyn menetystä. Tämän tutkimuksen puitteissa oli mielekkäämpää kehittää kirjaston ominaisuuksia, jotka tekivät kirjastosta toimivan ja käyttökelpoisen.

Lähteet

Batory, Don, Clay Johnson, Bob MacDonald ja Dale von Heeder. 2002. "Achieving Extensibility Through Product-lines and Domain-specific Languages: A Case Study". *ACM Trans. Softw. Eng. Methodol.* (New York, NY, USA) 11, numero 2 (): 191–214. ISSN: 1049-331X. doi:10.1145/505145.505147.

Becker, Roi, Yifat Chernihov, Yuval Shavitt ja Noa Zilberman. 2012. "An analysis of the Steam community network evolution". Teoksessa *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, 1–5. doi:10.1109/EEEI.2012.6377133.

Champion, Erik. 2012. "Introduction: Mod Mod Glorious Mod". *ETC Press.* http://press.etc.cmu.edu/content/introduction-mod-mod-glorious-mod.

Christiansen, Peter. 2012. "Between a mod and a hard place". Teoksessa *Game Mods*, 27–50. ETC Press. http://dl.acm.org/citation.cfm?id=2554086.

Dotan, Gur. 2017. *Top 10 Unity Games Ever Made.* http://blog.soom.la/2015/01/top-10-unity-games-ever-made.html.

Gamma, Erich, Richard Helm, Ralph Johnson ja John Vlissides. 1993. "Design Patterns: Abstraction and Reuse of Object-Oriented Design". Teoksessa *ECOOP' 93 — Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings*, toimittanut Oscar M. Nierstrasz, 406–431. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-47910-9. doi:10.1007/3-540-47910-4_21.

Hevner, Alan R., Salvatore T. March, Jinsoo Park ja Sudha Ram. 2004. "Design Science in Information Systems Research". *MIS Quarterly* 28 (1): 75–105. ISSN: 02767783. http://www.jstor.org/stable/25148625.

Ko, Andrew J., Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi ym. 2011. "The State of the Art in End-user Software Engineering". *ACM Comput. Surv.* (New York, NY, USA) 43, numero 3 (): 21:1–21:44. ISSN: 0360-0300. doi:10.1145/1922649.1922658.

Kow, Yong, ja Bonnie Nardi. 2010. "Who owns the mods?" *First Monday* 15 (5). ISSN: 13960466. http://firstmonday.org/article/view/2971/2529.

Laukkanen, Tero. 2005. "Modding Scenes". http://urn.fi/urn:isbn:951-44-6448-6.

Lowe, Russell. 2009. "Computer game modding for architecture". https://www.researchgate.net/publication/30872448_Computer_Game_Modding_for_Architecture.

March, Salvatore T., ja Gerald F. Smith. 1995. "Design and natural science research on information technology". *Decision Support Systems* 15 (4): 251–266. ISSN: 0167-9236. doi:10.1016/0167-9236(94)00041-2. http://www.sciencedirect.com/science/article/pii/0167923694000412.

Modding API. 2017. http://www.skylineswiki.com/Modding_API.

Murray, Jeff W. 2014. *C# game programming cookbook for Unity 3D.* CRC Press.

Nardi, Bonnie. 2010. *My life as a night elf priest: An anthropological account of World of Warcraft.* University of Michigan Press. eprint: https://www.digitalculture.org/books/my-life-as-a-night-elf-priest/.

El-Nasr, Magy Seif, ja Brian K. Smith. 2006. "Learning through game modding". *Computers in Entertainment* 4:7. doi:10.1145/1111293.1111301.

Norton, Terry. 2013. *Learning C# by Developing Games with Unity 3D.* Packt Publishing Ltd.

Parnas, David L. 1979. "Designing Software for Ease of Extension and Contraction". *IEEE Transactions on Software Engineering* SE-5, numero 2 (): 128–138. ISSN: 0098-5589. doi:10.1109/TSE.1979.234169.

Peffers, Ken, Marcus Rothenberger, Tuure Tuunanen ja Reza Vaezi. 2012. "Design Science Research Evaluation". Teoksessa *Design Science Research in Information Systems. Advances in Theory and Practice: 7th International Conference, DESRIST 2012, Las Vegas, NV, USA, May 14-15, 2012. Proceedings*, toimittanut Ken Peffers, Marcus Rothenberger ja Bill Kuechler, 398–410. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-29863-9. doi:10.1007/978-3-642-29863-9_29.

Poretski, Lev, ja Ofer Arazy. 2017. "Placing Value on Community Co-creations: A Study of a Video Game 'Modding' Community". Teoksessa *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 480–491. CSCW '17. Portland, Oregon, USA: ACM. ISBN: 978-1-4503-4335-0. doi:10.1145/2998181.2998301.

Postigo, Hector. 2007. "Of Mods and Modders". *Games and Culture* 2 (4): 300–313. doi:10.1177/1555412007307955.

———. 2008. "Video Game Appropriation through Modifications". *Convergence* 14 (1): 59–74. doi:10.1177/1354856507084419.

Scacchi, Walt. 2010. "Computer game mods, modders, modding, and the mod scene". *First Monday* 15 (5). ISSN: 13960466. http://journals.uic.edu/ojs/index.php/fm/article/view/2965.

———. 2011a. "Modding As a Basis for Developing Game Systems". Teoksessa *Proceedings of the 1st International Workshop on Games and Software Engineering*, 5–8. GAS '11. Waikiki, Honolulu, HI, USA: ACM. ISBN: 978-1-4503-0578-5. doi:10.1145/1984674.1984677.

———. 2011b. "Modding as an Open Source Approach to Extending Computer Game Systems". Teoksessa *Open Source Systems: Grounding Research: 7th IFIP WG 2.13 International Conference, OSS 2011, Salvador, Brazil, October 6-7, 2011. Proceedings*, toimittanut Scott A. Hissam, Barbara Russo, Manoel G. de Mendonça Neto ja Fabio Kon, 62–74. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-24418-6. doi:10.1007/978-3-642-24418-6_5.

SneakyPie. 2015. *DarkestDungeon-pelin keskustelupalsta.* Esittelee virallisia käytäntöjä, jotka Red Hook Studio on määritellyt modauksesta pelilleen. http://www.darkestdungeon.com/topic/red-hooks-official-statement-on-darkest-dungeon-modding/.

Sotamaa, Olli. 2007. "On modder labour, commodification of play, and mod competitions". *First Monday* 12 (9). ISSN: 13960466. doi:10.5210/fm.v12i9.2006. eprint: http://firstmonday.org/ojs/index.php/fm/article/view/2006.

Super Mario Maker Wiki. 2017. https://en.wikipedia.org/wiki/Super_Mario_Maker.

Taylor, T.L. 2009. "The Assemblage of Play". *Games and Culture* 4 (4): 331–339. doi:10.1177/1555412009343576.

Tratt, Laurence. 2008. "Domain Specific Language Implementation via Compile-time Meta-programming". *ACM Trans. Program. Lang. Syst.* (New York, NY, USA) 30, numero 6 (): 31:1–31:40. ISSN: 0164-0925. doi:10.1145/1391956.1391958.

Unity Fast Facts. 2017. https://unity3d.com/public-relations.

Unity Forums - Howto use JSON in 5.3 [Guide]. 2015. https://forum.unity3d.com/threads/howto-use-json-in-5-3-guide.373455/.

Unity Subscriptions. 2017. https://store.unity.com/?_ga=2.99012382.1604045617.1495439074-1532967001.1474017091.

Unity Technologies. 2017a. *Unity dokumentaatio - Asset Bundle.* Unityn oma dokumentaatio. https://unity3d.com/learn/tutorials/topics/scripting/assetbundles-and-assetbundle-manager.

———. 2017b. *Unity dokumentaatio - Loading resources at runtime.* Unityn oma dokumentaatio. https://docs.unity3d.com/Manual/LoadingResourcesatRuntime.html.

Wiki - Kerbal Space Program, Making an asset from start to finish. 2017. Wikissä neuvotaan yksinkertaiset vaiheet modin mukaan ottamiseen Kerbal Space Program-peliin. http://wiki.kerbalspaceprogram.com/wiki/Tutorial:Making%5C_an%5C_asset%5C_from%5C_start%5C_to%5C_finish.

Wikipedia - Steam Workshop games. 2016. Listausta steam workshoppia käyttävistä peleistä. https://en.wikipedia.org/wiki/Category:Steam%5C_Workshop%5C_games.

Xie, Jingming. 2012. "Research on key technologies base Unity3D game engine". Teoksessa *2012 7th International Conference on Computer Science Education (ICCSE)*, 695–699. doi:10.1109/ICCSE.2012.6295169.

Yin, Robert K. 2014. *Case study research : design and methods.* 5th ed. Los Angeles: SAGE.

Yucel, Ibrahim, Joseph A. Zupko ja Magy Seif El-Nasr. 2006. "IT education, girls and game modding". *Interact. Techn. Smart Edu.* 3:143–156.

Zelkowitz, M. V., ja D. R. Wallace. 1998. "Experimental models for validating technology". *Computer* 31, numero 5 (): 23–31. ISSN: 0018-9162. doi:10.1109/2.675630.

Zorz, Zeljka. 2017. *Minecraft players get scams instead of mods.* https://www.helpnetsecurity.com/2017/03/23/fake-minecraft-mods/.

Liitteet

A Tapaustutkimuksen vastaukset

Tapaustutkimuksen vastaukset ovat tekstimuotoisina ja anonymisoituina seuraavissa osioissa. Vastaukset sisältävät raportin, omat kommenttini heidän vastauksiinsa ja heidän mahdolliset vastaukset kommentteihin. Joitain rivivaihtoja lisätty tekstiin pitkien ilmaisujen eteen.

A.1 Vastaus 1

Tein ekstratehtävän modauksesta.

Lähdin kokeilemaan modausta lähinnä tavoitteella, että saisin kaksipistettä. Myös aihe kiinnosti jonkin verran, niin ihan mielelläni ajattelin tehdä.

Päätin ensin ruveta toteuttamaan hyppäämistä. Kuitenkin ensimmäiseksi ongelmaksi muodostui, ettei peli lähtenyt käyntiin, minkä jonkin ajan kuluttua huomasin johtuvan siitä, että ohjelmassa oli mysteeri virhe, joka pysäytti ohjelman automaattisesti. Näin ollen alussa piti painaa pause pois päältä. Tämän jälkeen hommat alkoivat sujumaan paremmin. Koodiin perehdyttyäni ja lopulta löydettyäni lua-koodit rupesin toteuttamaan hyppäämistä. Kuitenkaan itse hyppyvoiman luominen oli pienoinen ongelma sillä tälle ei suoraan ollut tukea. Kuitenkin arveltuani, että olion rigidbody liittyy olion fysiikkaan ja googlattuani vielä asian, niin onnistuin ratkaisemaan asian bindaamalla rigidbodyn ja vector2 lua koodille sopivaksi. Tämän jälkeen melko suoraviivaisesti sain hyppäämisen toimimaan kun käytin addforce funktiota. Tämä kaikki onnistui luan puolella kunhan tein sopivat bindaukset c# puolelle. Esimerkit auttoivat siinä, että edes tiesin kuinka bindata ja mitä tarvitsee bindata. Myös käyttäminen sujui todella luontevasti. Lähinnä piti vain googlata kuinka oliosta saadaan rigidbody esiin.

Tämän jäkeen piti kuitenkin vielä keksiä kuinka tunnistetaan onko olio maassa. Mietin ensin, että käyttäisin jonkinlaista törmäyksentarkistusta tai sitten seuraisin joten-

kin nopeuksia ja vertaisin edelliseen tilaan. Kuitenkin ensimmäinen ehdotus olisi ollut kohtuu työläs sillä pitäisi tilaa koko ajan seurata ja myös pitäisi osata kaikki tasot eritellä, jotta seinät eivät estäisi hyppäämmistä. Tämän takia päädyin simppeliin ratkaisuun eli onko olion velocity.y nolla, missä ainoa ongelma on lähinnä, että hypyn huippu kohdassa se on myös nolla. Tarkemmalla pohdinnalla tämä on varmaankin melko turha huoli sillä kiihtyvyys on luultavasti decimaaliluku ja kiihtyvyyttä päivitetään pätkissä eli kiihtyvyys voi varmaankin mennä 0.00054:stä suoraan -0.00012 ilman, että 0 edes koskaan tulee esiintymään.

Tässä vaiheessa sain hyppäämisen toimimaan ja aloin tekemään kolmen pisteen suoritusta. Ensimmäisenä kaivoin esiin data kansion ja sieltä json-tiedostot. Kuitenkin pelien lisääminen rinnakkaisiksi mielestäni ei onnistunut jsonin puolelta, koska json-tiedosto itsessään oli vain yksi peli eikä lista peleistä, jossa vain sattuu olemaan vain yksi peli. Jotta pelit voisivat olla rinnakkain, niin pitäisi jsonin rakennetta muokata, jolloin parsija menisi varmaan rikki. Itse asiassa vilkaisin parsijaa, mutta se oli melko automatisoitu enkä halunnut koskea siihen. Tämän takia tyydyin korvaamaan vanhan pelin tiedostossa.

Kentät taas onnistuivat lisäämään rinnakkaisiksi ihan hyvin. Itseäni tosin hieman ihmetytti, että miksi sekä peli-jsonissa ja kenttä-jsonissa molemmissa mainitaan kentän luomis-skripti. Mielestäni vain toisessa olisi tarpeellista käyttää. Luultavammin vain pelissä koska silloin voidaan tehdä pelikohtaisia muutoksia. Toisaalta tällöin pelien olisi vaikeampi käyttää samoja kenttiä. Parannusideana se, että kentät voisivat olla eri tiedostoissa, mitä nykyinen järjestelmä ei tue, sillä kaikkien pitää olla samassa (ainakin omasta mielestäni näin näyttäisi olevan).

Joka tapauksessa sain datatiedostot laitettua kohtuu hyvin kuntoon omaa peliäni varten, johon tein oman kontrollerin ja kentän luonnin. Pienenä yllätyksenä tuli, että kaatuu, jos pelin nimen kirjoittaa väärin unityn puolella eikä arvota peliä kuten käsittääkseni kentillä tehdään.

Kun sain yleiset asiat valmiiksi niin aloin toteuttamaan peliä, jossa pelihahmo juoksentelee kentällä edes takaisin väistellen taivaalta satavia mörököllejä. Mielestäni tä-

mä oli kohtuu simmpeli peli, mikä ei juurikaan vaadi sisällöin luomista kuten esim tasohyppelypelit.

Ensiksi kopioin esimerkkikoodin ja poistin kaiken turhan kuten toisen pelihahmon ja erilaiset tasot. Myös kentän kokoa säädin.

Kuitenkin itselleni muodostui ongelmia, kun yritin tappaa ominaisuutta, että ylöspäin ja alaspäin nuolista liikutaan. Vaikka poistin nämä toiminnasta kokonaan, niin yhä itsepintaisesti ne toimivat yhä. Kuitenkin hämmentävästi tiedoston muut ominaisuudet kyllä päivittyivät, mutta tämä ei vain suostunut toimimaan. Ongelma selvisi, kun huomasin, että update-toiminto oli kova koodattu c# puolelle käyttämään control-tiedostoa, vaikka ideana varmasti olisi, että käytettäisiin data-tiedoston jsoneissa käytettävät tiedostot. En jaksanut perehtyä jsonin parsimiseen itse vaan muutin suosiolla kovakoodauksen käyttämään omaa tiedostoani. Tämä oli kuitenkin yksi asia, joka tulisi varmaan korjata. Myöhemmin myös huomasin törmäyksenkäsittelyssä vastaavan, mutta tämän sain huomattua nopeasti kiitos aiemman kokemuksen. Sain myös kivoja virheitä kuten "function is not a function", mikä oli hieman huvittava virhe.

Tässä vaiheessa muodostin funktion, joka loi pikkuisen örkin korkealla satunnaiseen paikkaan ja laittoi sen putoamaan painovoiman vaikutuksesta. Ajastaminen onnistui kohtuu helposti sillä deltatime tuli luan puolelle parametrinä, joten todella helppoa.

Toinen vaihe oli huomata, jos putoava örkkeli törmää johonkin. Itse asiassa törmäyksen käsittely oli todella helposti tehtyä ja esimerkkiä seuraamalla onnistui helposti. Tyydyin vain katsomaan, että oliko toinen törmääjistä pelihahmo, jolloin hävittiin peli. Joka tapauksessa joka törmäyksessä örkkeli tuhotaan, jotta ei niitä kerry loputtomasti. Tässä oli ideana, että maahan osuttaessa örkkeli tuhoutuisi, mutta myös jäi ominaisuus (ei siis bugi), että jos kaksi örkkeliä törmäsi toisiinsa, niin molemmat tuhoutuivat. Tämä on kuitenkin kohtuu harvinaista eikä häiritse peliä liikaa. Toisaalta asian korjaaminen olisi ollut kohtuu työlästä, kun olisi joko pitänyt tunnistaa lattia tai kaikki örkit.

Joka tapauksessa sain pelin ihan pelattavaan kuntoon ja en muokannut käytännössä yhtään c# koodia, muuta kuin bindaukset ja kovakoodatut kohdat. Kaikki käytännössä siis onnistui luan puolella ja jsoneissa, mikä on sinänsä kivaa modauksen kannalta. Tällöin jos c# lukitaan kehittäjien toimesta, niin voidaan bindauksilla rajoittaa, että mitä kaikkea voidaan muokata. Antaa siis kehittäjille kivasti liikkumavaraa.

-Oliko kirjastoa helppoa käyttää?

En tiedä tarkkaan, mikä osa oli varsinaisesti kirjastoa ja mikä esimerkkikoodia. Itse kuitenkin kirjoitin lähinnä vain luaa. Jos kuitenkin kirjaston vastuulla oli linkittää tiedostoja niin se onnistui mielestäni ihan hyvin sillä tosiaankin lähes kaikki onnistui jsonilla ja lualla.

-Havaitsitko ongelmia kirjaston toiminnassa tai puuttuiko kirjastosta jokin toiminto tai ominaisuus, jonka olisit halunnut siinä olevan?

Kovakoodatut tiedostojen nimet, mikä varmaan menee sen piikkiin, että oli esimerkki. WorldCreationScript oli mielestäni turhaan molemmissa jsoneissa. En tiedä kumpaa sitten oikeasti käytettiin tiedoston luonnissa.

-Nostiko tehtävä kiinnostusta modaukseen?

Nosti. Todella helppokäyttöistä sinänsä. Tosin itsellä koodaustaustaa, niin oli lähinnä tunne, että koodasin peliä lualla. Ei tuonut niin paljon siis esiin ajatusta, että muokattaisiin tiedostoista jotakin arvoja yms. Kuitenkin ihan kivaa.

-Välittyikö kirjaston tavoite?

Mikä tavoite? Kai pääosin tosiaan kaikki onnistui luan avulla, mutta en ole varma oliko se tavoite. Jsoneija taas hyödynnettiin kohtuu vähän. Esim jotain pelihahmoja voitaisiin ladata jsoneista eri arvoilla, joita sitten voitaisiin muokata jsonissa.

Vastauksiani saa käyttää tutkimuksessa.

Vastaukseni löytyy täältä.

Minuun saa sähköpostilla myös yhteyttä xxx@student.jyu.fi

Kaiken kaikkiaan tehtävää oli ihan mukava tehdä. Ehkä itselläni enemmänkin tosiaankin enemmän pelin tekemistä pelimoottorin avulla kuin modaamista tai ainakin tällainen tunne itselleni jäi. Mukavaa oli tosiaan myös se, että bindaukset onnistuivat todella vaivattomasti ja lua ja c# keskustelivat oikein sujuvasti keskenään.

Arvostelisin oman suoritukseni 3 pisteen arvoiseksi.

Myös ohjeistus oli mielestäni oikein hyvä ja selitti todellakin asioita, niin ei tarvinnut arvailla. Myös esimerkit olivat todellakin auttavia.

Kommenttini: Arvosana:3 -

Oli mukava lukea vastauksesi aihealueesta. Vastauksessasi oli havaittavissa selkeästi erilainen peli, johon olit toteuttanut käyttäen eri rakenteita uuden toiminnallisuuden. Json-tiedostojakin oli käytetty ja muokattu. Mielestäni onnistuit hyvin vastaamaan tehtävään 3:n arvosanan mukaan, kuten itsekkin arvioit. -

Unohtui laittaa palautusohjeisiin vielä, että paljonko arvioit menneen aikaa tehtävän vastaukseen? -

Kirjasto on enimmäkseen se, mitä ExampleGameControllerissa kutsutaan koko ajan, eli se modController. Tuon rungon on siis tarkoitus olla hyvin yksinkertainen peli, miten kirjaston eri ominaisuuksia voi käyttää ja ajattelin että tuollaisesta hyvin avoimesta pelistä olisi helpointa lähteä muokkaamaan. Aion itse toteuttaa vielä gradun puitteissa toisen esimerkkipelin, jossa keskityttäisiin enemmän juuri Json-tiedonkin tallentamiseen, esimerkiksi dataa varten. -

Olin unohtanut vaihtaneeni tuon AvailableGames toiminnallisuuden katsomaan läpi kaikki tiedostot, eikä sitä yksittäistä tiedostoa. Oli vaikea arvioida sopivan kokoinen tehtävä, tehtävässä kuitenkin saattoi olla useita uusia teknologioita kokeiltavana. Ja todellakin, tuo WorldCreationScript oli jäänyt kahteen paikkaan. Nopeasti sitä tulee sokeaksi omalle koodilleen.

Aikaa meni ehkä jotain 8-9 tuntia. Saattaa parilla tunnilla heittää, kun kävin kaupassa ja tein muuta välillä

A.2 Vastaus 2

Kuvaus pelistä

Kaksinpelattavassa Hippa-pelissä on tarkoitus jahdata toista pelaajaa Labyrinttikentässä ja saada tämä kiinni ylhäältä käsin. Kentän keskiössä on eräänlainen free fight-zone jossa voi olla kontaktihippasilla ja paeta sitten labyrintin käytäville huilimaan jos meno yltyy turhan hektiseksi. Pelaaja 1 liikkuu näppäimistä WSAD, pelaaja 2 nuolinäppäimistä. Peli alkaa alusta pelaajan saadessa toisen kiinni. Yksityiskohtainen kuvaus työstä

Työvälineet: Unity 5.52f1 Personal sekä Notepad++

Huomasin että ExampleGameController.cs lataa kaikki .json-päätteiset tiedostot kansiosta "Mods", joten lisäsin sinne määrittelyn uudelle pelilleni "Hippa.json-tiedostoon, ja muutin "AvailableGames.json-tiedoston nimeksi "EatMe.json"muuttaen samalla pelien määrittelyn periaatteen muotoon 'yksi .json-tiedosto per peli'.

Otin "Controller.lua":sta kopion "HippaController.lua"ja asetin pelimäärittelytiedostoni "Mods.json"controllerName:n osoittamaan siihen, tarkoituksenani jättää alkuperäinen esimerkkipeli rauhaan ja luoda sen rinnalle siihen perustuva uusi versio - mutta tämä controllerName-määritys ei toiminut suoraan, jolloin huomasin että ExampleGameController.cs:ssä oli määritetty asioita tehtäväksi suoraan "Controller.lua":sta (funktioissa AddCollisionHandler ja Update) joten muutin kyseiset funktiokutsut muotoon
' modController.DoFunctionFromScript(gameName+"Controller"... ' lisäten gameName-stringin "Controller":in eteen jolloin kutsutaankin aina käynnissä olevan pelin mukaan gameName (esim. "EatMe") + "Controller-nimistä .lua-skriptiä. Tämän seurauksena minun oli muutettava myös 'Controller.lua'-tiedoston nimeksi 'EatMeController.lua', ja 'EatMe.json'-pelimäärittelytiedoston controllerNameksi "EatMeController".

Otin "Hippa.json-pelimääritystiedostossa käyttöön oman worldCreationScriptini "HippaWorldCreation". Lisäsin worldDetails.json:iin ja HippaWorldCreation.lua:an oman

levelini "labyrinth"sekä poistin HippaWorldCreation:ista vanhat levelit "desperados"& "walled".

Kasvatin HippaControllerissa pelihahmojen moveSpeediä ja poistin zoomauksen tarpeettomana (hävitin camera- ja zoomSpeed-muuttujat sekä zoomauksen toiminnallisuuden update()-funktiosta optimoidakseni sitä, poistin update():sta myös "Jump-toiminnon).

Vaihdoin taustamusiikiksi itse tehdyn "Mods.ogg"ja lisäsin uuden AudioSourcen "soundPlayer"ladaten siihen ääniefektin "fx.ogg"joka ei kuitenkaan jostain syystä toistunut this.soundPlayer.Play()-funktiokutsulla, kokeilin tehdä ja castata myös oman lua-puolelta kutsuttavan funktionsa tuon ääniefektin toistamiselle vailla tulosta (kokeilin myös valita Unityn Inspectorissa Sound Playeriksi ExampleGameControllerin samaan tapaan kuin Music Playerinkin kohdalla mutta tällöin musiikki lakkasi soimasta ja törmäyksen sattuessa loopattiinkin tuota ääniefektiä - efektin lataamisesta modControllerilla omaan AudioClipiinsä ja toistamalla se sitten musicPlayer.PlayOneShot()-funktiolla taasen seurasi kummallista äänipuuroa).

Korvasin möröt HippaControllerissa uusilla spriteilla 'player.png' ja 'player2.png'. HippaWorldCreationissa latasin uuden taustakuvan 'background.png', siirsin kiviseinät pelialueen reunamille ja koostin venytetyistä 'wall.png'-spriteistä labyrinth-kentän johon synnytin pelaajat satunnaisiin koordinaatteihin.

Laitoin pelin alkamaan alusta kun pelaaja saa toisen kiinni lisäämällä Restart()-funktiokutsun EndGame-funktioon ExampleGameControllerissa. Vastaukset kysymyksiin

Oliko kirjastoa helppoa käyttää?

Kirjaston toiminnallisuuden hahmottaminen erityisesti LUA- ja c#-puolen välisen kommunikaation lainalaisuuksien osalta vaati hiukan pähkäilemistä, mutta oli muuten melko selkeää. Toisaalta helppohan tuota oli muokata aiemman pohjalta mallia katsoen - sen verran monimutkaiselta kirjaston toiminnallisuus kuitenkin vaikutti että uuden, vastaavan pelin luominen tyhjästä olisi takuulla paljon haasteellisem-

paa.

Havaitsitko ongelmia kirjaston toiminnassa tai puuttuiko kirjastosta jokin toiminto tai ominaisuus, jonka olisit halunnut siinä olevan?

Työnkuvauksessani mainitsemat yhtäaikaisten äänilähteiden käyttöön liittyvät ongelmat vaikuttivat suhteellisen ylitsepääsemättömiltä. Kenttien suunnitteleminen käsin koordinaatit syöttäen ja koordinaattimuunnokset huomioon ottaen oli melkoisen työlästä, olisikohan koordinaattisysteemin selkiyttämiselle & jonkin sortin kenttäeditorille mahdollisesti tarvetta?

Nostiko tehtävä kiinnostusta modaukseen?

Tehtävä ei esimerkkipelin ja sen pohjalta luomani oman versioni simppelihköyden vuoksi ehkä niinkään nostanut erityistä lisäkiinnostusta modaukseen, mahdollisesti jonkun hiukan 'hienomman' ja/tai monipuolisemman pelin grafiikoiden ym. muutteleminen saattaisi ajaa asian tässä suhteessa paremmin..

Välittyikö kirjaston tavoite?

Modauksen perusperiaatteet (kenttien ja toiminnallisuuden skriptaus, grafiikan ja äänten muuttaminen jne.) se selvensi mielestäni hyvin.

Vastaustani saa käyttää anonyymisti tutkimuksessanne.

Kommenttini:

Arvosana: 5 -

Viiteen vaatittavat asiat ja muutokset löytyivät projektista. Peli on itsessään yksinkertainen, mutta muutoksia on kuitenkin tehty verrattuna alkuperäiseen.

Kauanko suunnilleen meni vastauksen toteuttamiseen? Tämä kiinnostaa jatkossa jos joutuu samankaltaisia tehtäviä miettimään ja se on mielenkiintoista myös kirjaston kannalta. -

Jostain syystä uusimmassa Unityn versiossa tuo äänien lataus heittää poikkeuksen

tyhjällä virheviestillä. Pitää selvittää, onko vika omassa koodissa vai Unityn puolella. Mielenkiintoinen ongelma. Tarkemman selvityksen jälkeen selvisi, että tuo WWW halusi koko polun ja sille meni tuossa esimerkissä vain suhteellinen polku, josta se ei tiedostoa löytänyt. Hyvä kysymys on, miten esimerkki on ylipäätänsä toiminut jossain välissä samalla koodilla ja nyt ei toimi. Tuo oli kuitenkin hyvä bugi löytää. -

Kun korjasin äänien latauksen, oma koodisi toimi oikein hyvin ääniin liittyen. -

Nuo esimerkit on sitä varten juuri, että näkee miten tuota olisi tarkoitus käyttää. Tuohon kieltämättä olisi vaikea perehtyä ilman minkäänlaista esimerkkiä.

Tuo kenttien suunnitteleminen käsin on hyvä pointti. Aloitin itseasiassa tekemään tällä viikolla Json-pohjaisen scenen muokkauksen, joka jossain määrin ymmärtää Tiled-editorin kenttäformaattia.

Ihan hyvä pointti myös, että jonkun valmiin pelin modaus voisi paremmin tuoda esille "oikeaa"modausta. Tehtävä oli tämänkaltainen, jotta saisin materiaalia gradua varten.

A.3 Vastaus 3

Palautettu unity-projekti

Palautetussa tehtävässä peliä on muokattu niin että kukkia ja kiviä sisältävässä kentässä kiviä voi poimia ja vihollinen pitää päihittää heittämällä kerättyjä kiviä. Huijasin ratkaisussa vähän lisäämällä unityprojektiin uusia tageja.

Kirjaston käyttö oli pohjimmiltaan helppoa. Enimmäkseen käytin toiminnoista funktioiden mappaamista. Jonkinlainen debuggausmahdollisuus, tai paremmat virheviestit Lua-skriptien puolelta olisivat olleet tervetullut lisä.

Tehtävää olisi ehkä voinut muokata niin että Unity projektissa olisi selkeästi tehty valmiimpi peli kokonaisuus jota muokata. Nyt kaikki oli käytännössä tehty Lualla, joten todellisen modauksen tuntua ei tullut. Enemmän tehtävä tuntui valmiiden

asettien käyttämiseltä Luassa.

En ole itse harrastanut modausta, mutta voisin kuvitella tekeväni sitä joskus. Tämä harjoitus toimi hyvänä päänavauksena sille toiminnalle.

Vastauksiani saa käyttää tutkimuksessa.

Kommenttini: Arvosana: 3 -

Kauanko vastaukseen meni aikaa suunnilleen? -

Pelissä saa tällä hetkellä loputtomat ammukset yhdellä kivellä, mutta ehkä tämä oli tarkoitettu ominaisuus. Debuggauksen parantaminen on ehdottomasti tärkeä kohde, mutta se on osittain myös Unityn ja MoonSharpin puolella ja pitäisi perehtyä tarkemmin vielä, miten noita saa paremmin debugattua. -

Tuosta on tullut muiltakin palautetta, että ei ollut paras mahdollinen tehtävä modaukseen syventymiseen. Tehtävän rakenne oli tälläinen, jotta saisin graduun materiaalia ja testattua kirjaston toiminnallisutta ja palautetta kirjastosta. Varmaan jatkossa kannattaa olla ihan valmis peli, jota muokataan.

A.4 Vastaus 4

Unity-projekti zippinä. Sisältää joitakin turhia tiedostoja, kun en uskaltanut enää koskea mihinkään, ettei vain hajoa.

Aloitin extrasyklin tekemisen tutustumalla luaan sen dokumentaation ja muutaman YouTube-tutoriaalin kautta. Tämän jälkeen siirryin tarkastelemaan annettua projektia yksityiskohtaisemmin. Muokkasin erityisesti lua-scriptejä ja katsoin, mitä vaikutusta muutoksilla oli pelissä, ja pyrin tätä kautta selkiyttämään itselleni systeemin toimintaperiaatetta. Toiminta tuntui selkeältä ja ajattelin luoda yksinkertaisen tasohyppelypelin.

Lähdin alkuun muokkaamaan .json-tiedostoja peliäni varten, ja tässä vaiheessa kohtasinkin ensimmäiset ongelmat, pitkälti luultavasti siksi, että en ole JSONia aikaisemmin juurikaan käyttänyt. Uuden kentän tiedot sain helposti laitettua worldDe-

tails.jsoniin, mutta en ihan ymmärtänyt, miten uuden pelin tiedot piti lisätä AvailableGames-tiedostoon. Yritin luoda taulukkoa erilaisille peleille, ja lisätä omani vanhojen pelien rinnalle tässä taulukossa, mutta se ei toiminut. Painit tämän kanssa vielä jokusen tovin, kunnes päädyin lopulta vain luomaan täysin uuden JSON-tiedoston omaa peliäni varten, missä nyt ei varmasti ollut mitään järkeä.

Saatuani pelini tiedot JSON-tiedostoihin, ryhdyin muokkaamaan Controller.lua-tiedostoa. Halusin saada sen toimimaan niin, omassa pelissäni painovoima vaikuttaisi hahmoihin, mutta valmiiksi annetut pelit pysyisivät ennallaan. Tämä osoittautuikin yllättävän hankalaksi, kun en alkuun keksinyt, missä tämän painovoiman asetuksen edes pitäisi tapahtua, tai että miten pääsen käsiksi hahmojen rigidbody elementteihin niiden luomisen jälkeen. Yritin tehdä globaaleja muuttujia näistä, mutta MoonSharp herjasi jatkuvasti, usein varsin hyödyttömillä virheilmoituksilla. MoonSharp tuntui valittavan aina, kun yritin tehdä jotain, mutta en enää yksityiskohtaisesti muista aivan kaikkea yrittämääni. Lopulta luovutin alkuperäisen Controllerin kanssa, ja päädyin luomaan uuden .lua-tiedoston kontrolleriksi omaa peliäni varten, ja viittasin tähän tiedostoon pelini .json-tiedostossa. Ratkaisu oli sinällään järjetön, sillä valtaosa koodista oli vain suoraan kopioitu alkuperäisestä kontrollerista.

Uuden kontrollerin luominen ei kuitenkaan ratkaissut ongelmiani, sillä jostain syystä kontrollerin updateen ei koskaan menty. Painin tämänkin kanssa jonkin aikaa, mutta en onnistunut ratkaisemaan ongelmaa. Tajusinkin / muistinkin vasta paljon myöhemmin, että ExampleGameController.cs kutsuu vain alkuperäisen kontrollerin updatea. En kuitenkaan enää keksinyt, että miten voisin järkevästi kutsua oman kontrollerini updatea oman pelini tapauksessa niin, että alkuperäisten pelien kohdalla edelleen käytössä olisi alkuperäisen kontrollerin update. Tässä vaiheessa luovuinkin alkuperäisestä tasohyppelypeliajatuksestani, ja lähdin luomaan tehtävänannossa ehdotettua hyppy-ominaisuutta, ja pelissäni olikin jo painovoima valmiina.

En kuitenkaan osannut lisätä .lua-tiedostossa hyppyä varten voimaa hahmon rigidbodyyn. Muistaakseni MoonSharp valitti ratkaisuistani toistuvasti. Päädyinkin sitten toteuttamaan kaikkein yksinkertaisimman mahdollisen ominaisuuden, eli hah-

moa voi kiihdyttää painamalla "Jump-painiketta pohjaan. Tätä varten jouduinkin lisäämään kytkennän Unityn KeyDown-funktioon ExampleGameController'ssa, mutta tämä ei ollut suuri vaiva. Halusin vielä saada kameran seuraamaan pelaajaa (poistin alkuvaiheessa toiselta hahmolta ohjattavuuden tasohyppelyä varten). Tämän sain jotenkin toimimaan suhteellisen pienellä vaivalla.

Halusin vielä lopuksi "poistaa"painovoiman, kun sille ei enää ollut tarvetta. Tässä kohtasinkin viimeisen ongelmani, kun Controller.lua-tiedostossa hahmojen rigidbodyjen gravityScale-arvon muuttamisella ei tuntunut jostain syystä olevan mitään vaikutusta. Tajusinkin jonkin ajan päästä, että tekemäni oma kontrolleri edelleen vaikuttaa näiltä osin peliin, ja painovoima-arvo määräytyikin siellä. Vasta tässä vaiheessa tajusin, että kontrollerini toimi muuten, mutta updatea olisi pitänyt kutsua .cs-tiedostossa.

Kaiken kaikkiaan sanoisin, että aikaa kului hieman päälle 8 tuntia, mihin sisältyy noin 1,5 tuntia luaan ja JSONiin tutustumista. Tämän raportin kirjoittamiseen kulunut aika ei kuitenkaan ole sisällytetty tuohon arvioon.

Eräänlaisena yhteenvetona vastaan vielä tehtävänannosa annettuihin kysymyksiin:

"Oliko kirjastoa helppo käyttää?": Alussa kesti aika pitkään, ennen kuin ymmärsin, miten kukin pieni osa ja tiedosto vaikuttaa lopputulokseen, ja missä kaikkialla muihin skripteihin ja toimintoihin pitää viitata. Kuvittelin tämän kuitenkin setviytyneen, kun testailin alkuperäisen projektin toimintaa muokkaamalla koodeja ja tarkastelemalla lopputulosta. Lopulta en kuitenkaan saanut mitään järkevää aikaiseksi, mutta tähän vaikuttanee se, että en ole luaa ennen käyttänyt.

"Havaitsitko ongelmia kirjaston toiminnassa tai puuttuiko kirjastosta jokin toiminto tai ominaisuus, jonka olisit halunnut siinä olevan?": Todella vaikea sanoa, johtuivatko kohtaamani ongelmat kirjaston toiminnasta vai siitä, että käytin sitä väärin. Kaiketi hyvä dokumentaatio tai jokin kaavio eri tiedostojen välisestä suhteesta voisi selkiyttää toimintaa.

"Nostiko tehtävä kiinnostusta modaukseen?": Ei juurikaan, kun tuntui, että mikään

ei oikein onnistunut.

"Välittyikö kirjaston tavoite?": Tälläisen yksinkertaisen pelin kohdalla kirjaston kautta modaus tuntui vähän hassulta varsinkin, kun välillä joutui silti muokkaamaan itse pelin c#-koodia. Mieluummin olisin tehnyt kokonaan Unityllä ja c#:lla.

Vastaustani saa minun puolestani käyttää tutkimuksessa.

Kommenttini:

Arvosana:3

Lisätoiminnallisuus oli vähän vähäistä, mutta olit kuitenkin saanut uuden pelin lisättyä hyvin ja kenttää muokattua ja raportti oli erittäin tarkasti dokumentoitu ja kuvattu eri vaiheet. Harmillista, että tehtävän tekemisessä tuli monia ongelmakohtia vastaan. -

Pelin lisäysratkaisu oli hyvä noin. Olit myös hyvin saanut maailmat lisättyä muiden rinnalle. #-

Testasin muuttaa gravity muuttujan arvoa controllerissa ja olioille tuli painovoima. RigidBodyihin pääsisi käsiksi samankaltaisella koodilla kuin c#:ssa:

local body = morkul.GetComponent("Rigidbody2D");

Tuohon voisi myös tehdä oman aliohjelman c#:n puolelle, joka palauttaa bodyn. Lisäyksessä myös palautetaan body, niin sieltä pääsee bodyyn käsiksi. #- Tuo update kutsu oli itsellä unohtunut päivittää. Tuosta valitusta pelistä voisi tehdä attribuutin ja kutsua attribuutin controlleria aina updatessa, niin toimisi aina. -

Tuo kaavio voisi olla ihan hyvä lisäys tuota esimerkkipeliä varten. -

Harmi että tehtävä ei innostanut modaukseen. Tehtävän rakenne oli tälläinen, jotta saisin testautettua kirjastoa ja nyt tehtävä enemmän näyttää, mitenkä kehittäjä itse voi sallia modauksen ja samalla itse käyttää modauksen rajapintaa pelin luomiseen, mutta ei ehkä paras tehtävä puhtaasti modaukseen liittyen siis.