Harri Keränen

Oliosuuntautuneisuus pelikehityksessä

Tietotekniikan kandidaatintutkielma

29. huhtikuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Harri Keränen

Yhteystiedot: keranhos@student.jyu.fi

Ohjaaja: Jonne Itkonen

Työn nimi: Oliosuuntautuneisuus pelikehityksessä

Title in English: Object-orientation in game development

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 26+0

Tiivistelmä: Pelien suunnittelun ja kehityksen yleisimpä keinoja on oliosuuntautuneisuuden hyödyntäminen. Olioajattelu perustuu käsitteeseen, jossa ohjelmiston, tai jonkin muun kokonaisuuden nähdään koostuvan monesta pienestä entiteetistä, jotka tekevät yhteistyötä jonkin asian saavuttamiseksi. Peleissä esiintyvät hahmot, esineet, ympäristöt ja niiden taustalla toimivat mekaniikat voivat hyödyntää oliosuuntautuneisuuden periaatteita.

Avainsanat: kandidaatintutkielmat,pelit,olio-ohjelmointi,oliosuuntautuneisuus,kehitys

Abstract: One of the most common ways of designing and developing a game is utilizing object-orientation. Thinking objects is based on a concept, where a software, or something else is seen as a compilation of many small entity, which work together to achieve something. In games, the characters, items, environments and the mechanics running in the background can utilize the principles of object-orientation.

Keywords: Bachelor's Theses,games,object-oriented programming,object-orientation,development

Esipuhe

Olen pelannut pelejä koko ikäni, ihan lapsesta lähtien. Pelien periaatteet ovat siis olleet minulle tuttuja isomman osan elämästäni. Säännöt, jotka peleihin liittyvät ovat koodareiden tekemiä rivejä pelien taustaohjelmistossa. Sääntöjen omaksuminen pelkästään pelin kautta liittyy puhtaaseen kokemukseen.

Tiesin alusta alkaen, että haluasin tehdä jollain tavalla peleihin liittyvän tutkielman, mutta harkitsin vielä jotain tietotekniikkaan yhdistävää tekijää niin, ettei aiheeni vaeltaisi liikaa humanististen tieteiden puolelle. Tehdessäni oliosuuntautuneisuuden kurssin tenttiä kiinnostuin olioajattelusta. Nimenomaan ajattelusta ja teoriasta. Mietin, että oliosuuntautuneisuuden periaatteita on helppo oppia, vertaamalla niitä oikean maailman asioihin ja sääntöihin. Ja jos pelien tarkoitus on luoda jonkinlainen omanlaisensa todellisuus omine sääntöinensä, voidaan olioajattelua soveltaa myös pelimaailmojen puolelle kehityksen, pelaamisen tai suunnittelun näkökulmasta.

Asioiden selvittäminen lähtikin siis olioajattelun teorian opiskelusta, ja sen asioiden miettimistä pelien näkökulmasta. Varsinaisesta aiheesta oli aluksi hankala löytää hyvää lähdettä kirjallisuuden ja tutkimuksen puutteen, ja pelien lähdekoodien tekijänoikeuksien ja salassapidon vuoksi. Myöhemmin huomasin kuitenkin yhteyden: ohjelmoinnin opetus. Jyväskylän yliopiston Ohjelmointi 1-kurssin harjoitustyönä tehdään peli C#-kielellä hyödyntäen Jypelikirjastoja. Aloin etsimään tietoa nimenomaan olio-ohjelmoinnin opetusta pelien kautta, ja löysin useita esimerkkejä oliosuuntautuneisuuden periaatteiden hyödyntämisistä peleissä. Tietenkin tutkielmani perustuu paljolti myös omaan kokemukseeni verraten aiemmin hyödynnettyjä tapoja.

Kiitokset kuuluvat kanssaopiskelijoille, opettajille, kaikille maailman pelaajille, lähteiden ja pelien tekijöille, ja ennen kaikkea kannustavalle puolisolleni.

Jyväskylässä 29. huhtikuuta 2022

Harri Keränen

Kuviot

Sisällys

1 Johdanto

Oliosuuntautuneisuuden periaatteet ovat osana pelien suunnittelua ja kehitystä, ja nämä periaatteet ovat havaittavissa pelimaailmassa. Pelioliot toimivat niille ohjelmoitujen sääntöjen mukaisesti, ja tämä voi olla helpostikin havaittavissa, kun peliä pelaa. Olio-ohjelmointi tulee olemaan relevantti pelien ohjelmointitapa myös tulevaisuudessa. [1] [2]

Iso osa nykyään käytetyistä yleisistä pelimoottoreista käyttää jotain oliopohjaista kieltä, kuten C#, C++, Python, Java tai JavaScript. Näiden kielten pohjalta ovat isot pelien kehittäjät, kuten Blizzard Entertainment-pelistudio soveltaneet omia työkalujaan, joilla he pelinsä tekevät (“What Coding Engine Does Blizzard Use?” 2022). Oliosuuntautuneisuus peleissä on siis asia, jota kannattaa tutkia ja kehittää, ja se on relevantti pelien kehityksessä sen kaikissa vaiheissa. [3] [4]

Tämä tutkielma on kirjallisuuskatsaus, jossa esitetään aluksi toisessa luvussa pelien esimerkkien perusteella valitut oliosuuntautuneisuuden periaatteet. Kolmannen luvun pelien kehityksen vaiheiden esittely auttaa hahmottamaan pelien kehitystä muunkin, kuin koodauksen kannalta. Pelien kehitys on muutakin kuin koodausta, ja oliosuuntautuneisuus kattaa myös oliomaisen ajattelun, ei vain ohjelmoinnin. Olioajattelu on jätetty kolmannen luvun varsinaisesta tekstistä pois, mutta sitä voi silti käsitellä sen näkökulmasta. Neljännessä luvussa esiintyy varsinainen kontribuutio. Luku käsittelee lähteistä saatujen esimerkkien lisäksi myös tekijän omia kokemuksia peleissä verraten niitä valikoituihin oliosuuntautuneisuuden periaatteisiin. Luvun selvästi nimetyt peliesimerkit ovat tekijän omia havaintoja kyseisen pelin toiminnasta, ja niiden todenperäisyyttä on tarkennettu lähteiden avulla. Koska pelien koodit ovat hyvin harvoin saatavilla isolle yleisölle, luvun lähteinä ei ole käytetty pelien koodia, vaikka se olisi kaikkein konkreettisin lähde tietojen todenperäisyydelle.

1. Oliosuuntautuneisuus on ajattelu- ja ohjelmointitapa, jossa monet entiteetit tekevät yhteistyötä saavuttaakseen jotain.

2. Pelioliolla tarkoitetaan pelissä toimivaa oliota, jonka piirteet ovat näkyviä pelin koodissa, ja itse pelimaailmassa.

3. Pelimoottori on ohjelmistokehys, jonka päälle rakennetaan peli. Aiheesta kerrotaan lisää luvussa 4.

4. Ohjelmointikieliä on monenlaisia, kuten olio-ohjelmointikielet ja funktio-ohjelmointikielet, jotka käsittelevät dataa puhtaasti funktioiden, eivät olioiden kautta, tehden siitä matemaattisemman vaihtoehdon.

2 Oliosuuntautuneisuus

Tarkastellaan oliosuuntautuneisuutta, ja siitä relevantteja asioita liittyen ohjelmointiin. Käsiteltävät asiat ovat valittu tämän tekstin pääaiheen tarpeen, ja yksinkertaisuuden perusteella (poislukien Historiaa-osio).

2.1 Olio

Olioita on kenties paras verrata ihmisen soluihin. Ne eivät tiedä toisistaan mitä sisältävät tai missä tilassa olevat (kapselointi), mutta ne kommunikoivat ja työskentelevät yhdessä suorittaakseen toimintoja. Olioita voidaan verrata myös pieniin tietokoneisiin, jotka viestittelevät toisilleen. Ne ajattelevat itsenäisesti, ja suorittavat ohjelman toimintoja, eli toimivat sen rakennuspalasina. Oliot tietävät jotain, suorittavat toimintoja, tekevät oikeita päätöksiä ja pitävät yllä yhteyksiä muiden olioiden välillä. (Wirfs-Brock ja McKean 2003).
Näin toimii olioilla tehty ohjelma. Itse ohjelmaakin voidaan pitää oliona. Olioajattelussa kaikkea voidaan pitää oliona. Jopa fyysisiä asioita, kuten ihmisiä tai esineitä.

2.2 Luokka

Luokka määrää sen, millaisia oliot ovat. Olioita, jotka ovat tehty jonkin luokan avulla, kutsutaan tämän luokan instansseiksi. Luokasta saadaan tietoa sen instansseista. Luokan kaikki instanssit toimivat samalla tavalla. (Wirfs-Brock ja McKean 2003). Tiivistettynä: luokkaa voidaan pitää oliona, jonka tehtävä on tehdä toisia olioita.

2.3 Rajapinta

Rajapinta on lista toiminnoista, mitä olio voi tehdä. Esimerkiksi, jos on olemassa Auto-luokka, Skootteri-luokka ja Rekka-luokka. Kaikilla näillä luokilla tulisi olla Käynnistä-ominaisuus. Tapauksissa, joissa luokilla on pakko olla jokin sama ominaisuus, on rajapintojen tehtävä varmistaa tämä. Rajapinnat eivät muodosta attribuutteja tai määrittele datan käsittelyä suo-

raan. (“Interfaces” 2008).

2.4 Perintä

Perintä on toinen tapa laajentaa olion osaamista. Toinen luokka kykenee omaksumaan toisen luokan ominaisuuksia. Perivää luokkaa kutsutaan aliluokaksi, ja perittävää yliluokaksi. Useamman luokan periminen on mahdollista. (Wirfs-Brock ja McKean 2003).
Esimerkiksi, jos on aliluokat Kissa, Koira ja Lehmä, voisi näiden yliluokka olla Eläin-luokka (tai Nisäkäs), koska se on niille yhteinen piirre. Kaikki perivät eläimen ominaisuudet. Eläin voisi puolestaan periä Elollinen-luokan ominaisuudet.
Tässä tekstissä yliluokka ja superluokka tarkoittavat molemmat luokkaa, jonka jokin toinen luokka perii. Superluokka-sanalla tarkoitetaan tässä tekstissä isossa perintähierarkiassaan todella ylhäällä, usein jopa ylimpänä olevaa luokkaa, jonka usein perii useampi kuin yksi luokka. Yliluokka-sanaa käytetään tässä tekstissä kuvailemaan enemmänkin kahden luokan välistä perintäsuhdetta, ei isoja hierarkioita.

2.5 Polymorfismi

Olio pystyy lähettämään saman viestin monien eri luokkien instansseille. Näissä eri luokissa kyseisen viestin vastaanottaminen voi synnyttää erilaisen lopputuloksen. (Wirfs-Brock ja McKean 2003). Esimerkiksi, olio lähettää viestin kahdelle eri luokkien instanssille: sanoJotain. Luokan A instanssi vastaanottaa viestin ja sanoo jotain: "Heipä hei". Luokan B instanssi vastaanottaa viestin ja sanoo: "Tervemenoa!". Sama viesti, mutta eri lopputulokset, koska näissä kahdessa luokassa on sanoJotain-viestin vastaanottaminen määritetty eri tavalla.

2.6 Historiaa

Ensimmäisenä olio-ohjelmointikielenä pidetään Simulaa, josta myöhemmin kehittyi Smalltalk 1970-luvulla. Simulaan sisältyi ideat luokista, olioista ja olioiden viittauksista. Smalltalk-72 tarkensi ja laajensi olioiden ideaa pieninä tietokoneina. Oliot sisälsivät tietoa ja operaatioita liittyen kyseiseen tietoon, kuten tietokone yhdistää tietoa sisältävän muistin ja tietoa

käsittelevän loogisen yksikön. 1980-luvun loppuun mennessä olioajattelu saavutti teollista kiinnostusta. Monia muita olio-ohjelmointia tukevia kieliä oli kehitetty, kuten C++. (Black 2013).

3 Pelien kehitys

Käydään läpi pelien kehityksen keskeistä tietoa. Esitellään pelikehityksen hyödyllinen työkalu: Game Design Document, ja käydään läpi seitsemän kehityksen vaihetta. Pelien ajattelu oliosuuntautuneesti edellyttää pelien kehityksen ymmärtämistä, koska on syytä tietää, että millaisista lähtökohdista pelien oliot voisivat syntyä. Tässä luvussa ei kuitenkaan käsitellä varsinaista olioajattelua yhdistäen sitä pelien kehitykseen, koska tekstin pääosaisena aiheena on asian tutkiminen lähinnä pelimaailman kautta, ei kehitysaskelien. Tämän luvun tietoa voi silti yhdistää aiemmin läpikäytyihin, ja myöhempien peliesimerkkien perusteella valikoituihin oliosuuntautuneisuuden periaatteisiin.

3.1 Yleistä

Pelejä tehdään mm. seuraavista syistä: unelmien jakaminen, opetus, harrastus tai raha (Bethke 2003). Pelejä kuitenkin on nykyään tapana pitää yhtenä taiteen lajina. Taiteen lajit voivat sisältää toisiaan. Kirja ei itse voi sisältää musiikkia, mutta musiikki voi sisältää taustatarinan. Pelit ja elokuvat yhdistävät useaa taiteenlajia yhdeksi kokonaisuudeksi. Elokuvissa on kirjoitettu tarina, musiikkia, visuaalista taidetta tai teatterimaisuutta. Pelit voivat sisältää kaikkea näistä, ja pienet elokuvat (välinäytökset) peleissä ovatkin yleisiä. Pelit eroavat elokuvista lisäämällä mukaan interaktiivisuuden.

Pelit hallitsevat ainutlaatuista asemaa viihteen alalla suuren interaktiivisuutensa takia. Tämä on hyvin erityinen piirre, koska pelaaja on pelin ja sen tarinan tärkein asia. Vain pelissä voi yrittää erilaisia asioita, kokea lopputuloksia ja tutkia pelimaailmaa- ja mekaniikkoja mielensä mukaan. (Bethke 2003). Interaktiiviset elokuvat ja sarjat ja niiden kiistanalainen asema peleinä ovat omia aiheitansa, ja niihin ei paneuduta tässä tekstissä tämän enempää.

Interaktiivisuus on toteutettava jotenkin. Pelaajalle on annettava jotain, jonka kanssa olla vuorovaikutuksessa. Muuten peliä ei voi kutsua varsinaiseksi peliksi. Pelin genre vaikuttaa pelin kokoon, ja pelin koko määrää interaktiivisuuden monimutkaisuuden. Roolipeleissä pelaaja luo oman hahmon, ja pelaaminen on pääosin vaikuttamista tämän käytökseen sille tehdyssä maailmassa. Maailma on iso sana, varsinkin, kun kyseessä on pelit. Kokonaisen maailman luominen on iso urakka ja erittäin monimuotoinen projekti. Peliä varten tarvitaan

moniosaavainen tiimi. Mitä suurempi peli, sitä moniosaavaisempi sen on oltava. Kuka tahansa voi suunnitella peliä ja vaikka kuinka kauan, mutta peli ei ole mitään muuta kuin se, minkä taiteilijat ja ohjelmoijat luovat (Bethke 2003).

3.2 Game Design Document

Game Design Document (GDD), eli pelisuunnitteludokumentti auttaa ymmärtämään pelin visiota. Se sisältää mm. idean tai konseptin, genren, tarinan, hahmot, ydinmekaniikat, pelattavuuden, taso- ja maailmasuunnittelun, taidetta ja monetisaatiostrategiaa. GDD:tä jatkuvasti päivitetään ja uudistetaan tuotannon yhteydessä, koska visio ei aina vastaa teknisiä tai taloudellisia mahdollisuuksia. Useimmat isoimmat kehitysstudiot käyttävät GDD:tä, ja pienemmät voivat jättää sen pois, käyttäen sen sijaan joustavampia käytäntöjä. GDD pitää asiat organisoituina. GDD auttaa myös pitchauksessa ja mainonnassa. (“How video games are made” 2019).

3.3 Pelikehityksen vaiheet

Pelikehityksen vaiheet voidaan jakaa joko kolmeen tai seitsemään vaiheeseen. Kolme perusvaihetta ovat esituotanto, tuotanto ja jälkituotanto (“How video games are made” 2019). Tässä tekstissä tarkastelemme seitsemää vaihetta, jotka kattavat nämä kolme. Peli on ohjelma, eli koodia, joten kaikkien vaiheiden perusasioiden tuntemus auttaa itse pelin parissa työskentelystä, ja sen koodaamisen ymmärtämisestä.

3.3.1 Suunnittelu

Suunnitteluvaiheessa pelin perusasiat päätetään.
Ydinkysymyksinä toimivat:

- Mitä tehdään?
- Millainen budjetti on?
- Kenelle suunnattu?
- Mille alustalle (konsoli, tietokone, puhelin ym.)?

(“The 7 stages of Game Development” 2019).
Tässä vaiheessa voidaan aloittaa kirjoittamaan GDD:tä, jos suunnitelma näyttää toteutettavalta.

3.3.2 Esituotanto

Esituotannossa mietitään mahdollista tarinaa ja tarinankerrontaa. Hahmojen ja tarinan ideaa on voitu miettiä suunnitteluvaiheessa, mutta esituotannossa päätetään jo tietyistä asioista. Teknisiä asioita mietitään rajojen näkökulmasta, eli tuleeko pelin tekemiseen haasteita, tai onko peli mahdollista tehdä ollenkaan. Ensimmäisiä prototyyppejä aletaan tekemään, ja aikataulutetaan tuotantoa. (“The 7 stages of Game Development” 2019). Esituotannon jälkeen GDD:n olisi hyvä olla valmis, vaikka se tulee mahdollisesti päivittymään tuotantovaiheessa.

3.3.3 Tuotanto

Varsinainen työstäminen alkaa tuotantovaiheessa. Hahmoja suunnitellaan ja mallinnetaan pelimaailmaan. Lisätään tehosteita, kuten äänet, ja visuaaliset tehosteet. Ääninäyttelijät tekevät työnsä ääniohjaajan kanssa. Tasojen kehittäjät rakentavat pelialueet. Fysiikat ja muut mekaniikat ohjelmoidaan. Projektin päälliköt varmistavat, että jokainen osasto ja niiden tiimit hoitavat työnsä. (“The 7 stages of Game Development” 2019). Tämän tekstin myöhemmät peliohjelmointiin liittyvät osiot tapahtuvat siis tässä vaiheessa.

3.3.4 Testaus

Tuotantovaiheessa kehitettyjä asioita testaillaan koko ajan, ja tarvittaessa ne siirretään takaisin tuotantovaiheeseen. Testauksen ja tuotannon voisi siis kuvitella tapahtuvan samaan aikaan, tai ikään kuin syklinä. Testaukseen kuuluu tyypillinen bugien löytäminen ja eri ominaisuuksien hyväksikäyttö. Samalla käydään läpi tasapainoon liittyviä asioita, eli onko peli liian vaikea tai helppo. Näitä asioita tarkastelemalla saadaan kuva siitä, että onko peliä ylipäätään hauska pelata. (“The 7 stages of Game Development” 2019).

3.3.5 Esijulkaisu

Peliä markkinoidaan, mainostetaan ja esitellään tapahtumissa. Pelistä voidaan julkaista alpha- ja betaversiot jopa yleisön käyttöön. (“The 7 stages of Game Development” 2019). Alpha-versiot on tapana jättää esittelyitä ja demoamista varten. Beta-versio voidaan julkaista rajatulle määrälle pelaajista testaamiseen. Tätä voidaankin kutsua nimellä beta-testing, tai suomeksi beta-testaus. Se on yleistä varsinkin moninpelien kanssa.

3.3.6 Julkaisu

Viimeiset bugit korjataan, ja tehdään loput viimeistelyt julkaisua varten. Julkaisussa peli annetaan laajan yleisön käyttöön pelattavaksi. (“The 7 stages of Game Development” 2019).

3.3.7 Jälkituotanto

Julkaisun jälkeisenä aikana tunnistetaan yhä olemassa olevia bugeja. Peliin voidaan kehittää lisäsisältöä, maksullista tai ilmaista. Tyypillisiä päivityksiä ovat tasapainopäivitykset (balance patch), ja bugeja korjaavat päivitykset (bug-fix). (“The 7 stages of Game Development” 2019). Pelit voivat olla todella isoja tuotantoja, ja pelaajien ajattelutapa voi olla hyvinkin erilainen verrattuna pelin kehittäjiin. Siksi bugeja löytyy jopa julkaisun jälkeen. Suuremmalta osalta yleisöä saadaan palautetta pelistä, joten sitä voidaan parannella yhä enemmän.

4 Olioiden käyttö peleissä

Tässä luvussa yhdistetään aiemmat tiedot olioista ja peleistä. Oliot ovat entiteettejä, jotka tekevät yhteistyötä jonkin asian saavuttamiseksi. Olio-ohjelmoinnissa tämä saavutettava asia on toimiva ohjelma. Peli on myös ohjelmoijien koodaama ohjelma, joka sellaisen lailla ratkaisee ongelman, joka on tässä tapauksessa tylsyys. Tässä luvussa tarkastellaan, mitkä asiat peleissä ovat olioita, ja miten ne toimivat yhdessä tehdäkseen pelistä toimivan ohjelman.

4.1 Ohjelmoitavia osa-alueita

Kuten aiemmin todettiin, peli on se, minkä taiteilijat ja ohjelmoijat siitä tekevät. Ohjelmoijien työhön kuuluu koodin tekeminen: 3D-pelimoottori, verkko, taiteen siirtäminen peliin yms. ja pelin vision ymmärtäminen (Bethke 2003). Tässä GDD tulee esiin. GDD:n avulla saadaan kehittäjät ymmärtämään, millaisesta pelistä on kyse.
Tässä osassa tarkastellaan seuraavia ohjelmoitavia osa-alueita, ja sekoitamme alustavasti oliosuuntautuneisuutta mukaan. Lisää oliosuuntautuneisuuden vaikutuksia peleihin tulee ilmi tämän luvun Pelioliot-osassa.

4.1.1 Pelimoottori

Pelimoottori on videopelin ohjelmistokehys. Sen päälle rakennetaan pelejä ajan ja vaivan säästämiseksi, tosin jotkut isommat peliyhtiöt luovat aina oman moottorin uudelle pelille. Tyypillinen pelimoottori sisältää mm. 3D-grafiikkaa, fysiikkaa, törmäyksien havaitsemista, ääniä, animointia, tekoälyä ja verkko-ominaisuuksia (“A Platform-based for Game Development to Improve The Object-oriented Programming Skills” 2016). Pelimoottoreissa voi olla useita kielimahdollisuuksia, tai käyttävät jotain omaa kieltään. Omat kielet perustuvat johonkin olemassa olevaan ohjelmointikieleen, tosin niiden tuki on alkamassa poistua käytöstä joissain pelimoottoreissa.
Unreal-pelimoottori käytti omaa UnrealScript-komentokieltään, kunnes muuttui täysin C++ -pohjaiseksi (“Unreal Engine 4 - First Look” 2012). UnrealScript oli C++:n ja Javan innoittama, ja sisälsi samanlaisen käsityksen olioista, luokista ja perinnästä kuin useimmat olio-

pohjaiset ohjelmointikielet. Luokat siis toimivat kuin pohjapiirrustuksina pelissä toimiville *actoreille* (pelissä toimiva olio, kuten esineet ja hahmot). (“UnrealScript Classes” 2012). Unity-pelimottorissa pelien ohjelmointiin pystyi käyttämään kolmea eri ohjelmointikieltä: UnityScript, Boo ja C#. UnityScript muistutti erittäin paljon JavaScriptiä. UnityScriptin ja Boon virallinen tuki on päättynyt, mutta UnityScriptiä pystyy silti käyttämään ulkoisin keinoin. Jatkossa Unityn ainoana ohjelmointikielenä on C#, oliopohjainen kieli. (“UnityScript’s long ride off into the sunset” 2017).

4.1.2 Pelimekaniikat

Pelimekaniikkoihin sisältyy fysiikat, ja miten objektit ja oliot, kuten aseet, tavarat ja hahmot toimivat (Bethke 2003). Kuten aiemmin todettiin, nämä ovat usein valmiina pelimoottorin kirjastoissa, mutta niitä pystyy muuntamaan mielensä mukaan. Pelimekaniikat määräävät siis pelin taustatoiminnan päästä varpaisiin. Grafiikoiden, käyttöliittymän ja äänien voidaan kuvitella toimivan ikään kuin peliohjelmnan front-endinä, eli asiana, jonka pelaaja välittömästi huomaa. Pelimekaniikat sen sijaan toimivat back-endinä, eli asiana, joka toimii taustalla määräten asioita, mutta silti pelaajan huomaamattomissa, ainakin välittömällä tasolla. Toisin sanoen, pelimekaniikat tulevat pelaajalle tietenkin tutuksi kun peliä pelaa, tai tutoriaalin kautta, mutta miten asiat toimivat koodissa, eivät pelaajalle näy suoraan.

4.1.3 3D-grafiikoiden renderöinti

3D-grafiikoiden ohjelmointi on hyvin matemaattista työtä, jossa pitää ymmärtää laskentaa, vektoreita, matriiseja, trigonometriaa ja algebraa (Bethke 2003). 3D-malli voi toimia luokkana, johon kaikki samankaltaiset 3D-mallit perustuvat. Monet pelit sisältävät erilaisia ulkoasuja (skin) hahmoilleen, jonka pelaaja voi valita. Nämä asut eroavat pelihahmosta ainoastaan ulkomuodoltaan, animaatiot ovat (yleensä) samat. Hahmoilla on oma perusasu (base skin), joka toimii vakiona. Perusasu voi siis toimia animaation apuna, ja siitä voidaan periä, tai sen päälle rakentaa vaihtoehtoinen asu, käyttäen sitä kehyksenä.

4.1.4 Tekoäly

Pelitekoäly määrää ei-pelattavien pelihahmojen käyttäytymisen. Se sisältää metodeja ja tiloja, jotka ovat tyypillisiä olioille. Tekoälyn ominaisuudet riippuvat erittäin paljon pelistä (Bethke 2003). Yleinen tapa hallita käyttäytymistä on olion tilan vaihteleminen. Tästä hyvänä esimerkkinä toimivat hiiviskelypelit. Vihollisilla on perus partiointitila, etsintätila, ja kriisitila. Partiointitilassa vartijat menevät omia reittejään pitkin tai pysyvät paikoillaan, ja pelaajan huomatessaan menevät kriisitilaan, jossa he alkavat ampua, tai napata pelaajan. Kadottaessaan pelaajan vartijat menevät etsintätilaan (tai takaisin partiointitilaan), jossa he etsivät pelaajaa aktiivisesti, joskus välittämättä partiointireiteistään tai vartiointipisteistään. Jälleen pelaajan huomatessaan siirtyy tekoäly kriisitilaan. Olioiden tiedon piilotuksen takia muut pelin oliot eivät kykene ymmärtämään kyseisiä tiloja, mutta pelaaja kykenee, usein oman kokemuksensa kautta. Tässäkin tapauksessa front-end ja back-end käsitteet tulevat esille. Ellei peli ilmaise hahmojensa tiloja suoraan, pelaaja ei voi välittömästi ymmärtää tekoälyn omaksumaa tilaa, mutta pystyy kokemuksensa kautta tekemään havaintoja, ja oppia pelin hahmojen tekoälystä tätä kautta.
Joskus tekoälyä ei tarvita lainkaan: kuten peleissä, joita pelataan vain yksin ja vailla minkäänlaista vastusta (esim. useimmat puzzle-pelit).

4.1.5 Käyttöliittymä

Pelit tarvitsevat myös käyttöliittymänsä. Erilaiset valikot, paneelit ja HUD (heads-up display, "heijastusnäyttö") kuuluvat pelien käyttöliittymiin (Bethke 2003). HUD sisältää kaiken informaation, mikä on pelaajalle tärkeää, ja on mahdollista nähdä välittömästi peliruudulta kaiken muun tapahtuman kanssa. HUDissa yleensä näytetään elämänpisteet, aika, aseet, panokset, eteneminen, kartta, tähtäysristikko tai kompassi (“Heads-up display” 2012).

4.1.6 Ääni

Yleensä äänet otetaan valmiista kirjastosta (Bethke 2003). Nämä kirjastot ovat pelimoottorin sisällä tai hankittu ostamalla. Muuten käytetään kehittäjän omistamaa arkistoa, tai itse äänitettyjä uusia ääniä. Äänitiedosto toimii koodissa oliona, joka laitetaan soimaan jonkin

tapahtuman yhteydessä. Se on siis riippuvainen muiden olioiden metodeista, koska äänet muutenkin sitoutuvat tapahtumiin. Tietyissä tapauksissa tapahtumattomuus laukaisee myös ääniä: joskus pelihuumoriin kuuluu pelihahmojen tylsyyden toteamus, jos pelaaja ei teekään mitään.

Tiettyä hetkeä odottavaa oliota kutsutaan triggeriksi, eli liipaisimeksi. Trigger laukaistaan, kun jotain tapahtuu, esim. pelaaja astuu tietylle alueelle, jokin aikaraja ylittyy, pelaaja kerää tavaran tai vihollinen ampuu aseellaan. Joskus trigger ei ikinä saa tarvitsemaansa tilannetta, ja ääntä ei kuulu tai jotain muuta ei ikinä tapahdu. ("Triggers" 2020).

4.1.7 Työkalut

Pelikehityksessä voidaan säästää paljon aikaa ja vaivaa kehittämällä oma työkalu pelin kehittämiselle tai tekemiselle, ja käyttämällä sitä monen tulevan pelin kehityksessä. Tämä on yleistä kehittäjillä, jotka tekevät tai ovat tehneet samankaltaisia pelejä useita kappaleita, kuten BioWare. Vaikka kehittäjästudio ei olisikaan keskittynyt moiseen, saattaa työkalun kehittyminen ja myöhempi käyttöönotto tulla luonnostaan. (Bethke 2003).

Pienempien työkalujen, kuin pelimoottorien kehitys keskittyy pienempiin tehtäviin. Obsidian Entertainment-pelistudion dialogi-instanssin luova työkalu säästää aikaa, koska keskusteluinstansseja esiintyy paljon heidän roolipeleissään. Työkalu sisältää paljon koodia, jonka yleisin tehtävä on varmistaa jotain pelitilanteesta. ("Technical Tools for Authoring Branching Dialogue" 2019).

4.1.8 Tasot/Tasotyökalu

Tasojen kehitystä voidaan pitää osana työkalukehitystä. Useimpien pelien, joiden kentät on tehty juuri siihen tehdyllä kenttäeditorilla, usein julkaistaan pelin julkaisun yhteydessä (Bethke 2003). Warcraft 3 ja Starcraft 2 -pelien editorit julkaistiin peliensä yhteydessä, ja molempien pelien kentät on tehty vastaavilla editoreillaan. Warcraft 3:n editori oli niin monipuolinen, että se synnytti jopa oman peligenren, MOBAn (Multiplayer Online Battle Arena).

4.1.9 Verkko

Verkko-ominaisuuksien ohjelmointi peliin noudattaa monia samoja protokollia, kuin muissakin verkon käyttötarkoituksissa. Modernit pelit toimivat lähes täysin TCP/IP ja UDP verkkoprotokollilla (Bethke 2003). Tässä tekstissä emme paneudu tämän enempää pelien verkkoohjelmointiin, koska se on täysin oma aiheensa, ja hyvin samankaltainen muun verkkoohjelmoinnin kanssa. Lisäksi, verkko-ominaisuuksilla ei ole suoraa oliosuuntautuvaa vaikutusta pelin toimintaan, minkä tarkastelu on tämän tekstin tarkoitus.

4.2 Pelioliot

Olio-orientoitunut suunnittelu on ihmisille hyvin luonnollista. Oikeassakin elämässä mietimme asioita olioimaisesti: asioilla on tietyt piirteet ja käyttäytyminen. Sama pätee myös peliin, eli voisi sanoa, että peleissäkin kaikki ovat olioita: hirviöt, seinät, kolikot, bonukset, aseet ja panokset. Pelien kehittämisen ajattelu tarkoittaa olioiden ajattelua ja miten ne reagoivat toisiinsa ja pelaajan vaikutukseen. Pelin kehittäjän voisi siis sanoa ajattelevan luonnollisesti oliosuuntautuneella tavalla. (“Game Design In Education” 2004).
Tässä osassa pohditaan mahdollisia tai jo olemassa olevia konkreettisia tapoja yhdistää olioajattelua pelimaailman tapahtumiin ja oliosuuntautuneisuuden piirteitä pelin käyttäytymiseen yhdistämällä myös aiemmin todettua tietoa. Sana peliolio tarkoittaa tässä tapauksessa pelissä työskentelevää entiteettiä kuten esine, hahmo tai maailma, joka toimisi pelin koodissa omana olionaan. Peliolioon yhdistetään siis suoraan olion perusominaisuudet ja sen ominaisuudet itse pelissä.

4.2.1 Hahmot

NPC, eli non-playable character on pelihahmo, jolla pelaaja ei pelaa. Useimmissa peleissä NPC:t käyttäytyvät tekoälynsä mukaan. Jos pelihahmoa ajatellaan oliona, se sisältää tietoa, ja omaa metodeja. NPC:t eivät voi tietää, tai niillä ei ainakaan olisi syytä tietää pelaajahahmo-olion tiedoista, kuten liikkeistä mitään olioiden tiedon kapseloinnin takia (Huang, Zhang ja Xu 2018). Ajatellaan seuraavaksi pelihahmoja luokkien näkökulmasta, koska niiden avulla on helpompi kuvailla peliolioiden ominaisuuksia.

Yleensä NPC:llä on kaksi luokitusta: vihollinen ja liittolainen. Joskus näiden välimaastossa olevia kuvauksia (esim. neutraali) voidaan antaa hahmoille. Peleissä esiintyvät viholliset noudattavat aina vähintään yhtä seuraavista säännöistä: ne yrittävät aiheuttaa pelaajalle haittaa, tai toisin päin. Vihollinen-luokka tai rajapinta voivat toimia perustana vihollisen toiminnalle. Rajapinnassa olisi siis lista toiminnoista, mikä on vihollishahmolle olennaista: pelaajan havaitseminen, hyökkääminen, kuoleminen, ääntely jne.

Eri vihollistyyppien luokat taas määräävät eri vihollistyyppien ominaisuudet. Koska rajapinnassa on määritelty vihollisen perusmetodit, on luokassa määriteltävä erityispiirteet. Minecraftin zombie-vihollinen on erittäin yksinkertainen. Se kävelee satunnaisiin suuntiin tai pysyy paikoillaan, kun pelaaja, tai jokin muu zombien vihollinen ei ole näkyvissä. Vihollisensa huomatessaan se kävelee päin ja hyökkää kimppuun lähietäisyydeltä. Kun zombien kimppuun hyökätään, se kutsuu kaikki lähellä olevat zombiet apuun. Tekemistä vailla (idle) olevan zombien käytöksessä on siis pientä satunnaisuutta, ja toimii sen ensimmäisenä metodina. Zombielle on myös ohjelmoitu triggeri, joka tarkistaa itsensä tiettyjen aikojen välillä. Nämä ajat perustuvat Minecraftin *tickeihin*: tiettyyn aikarajaan (tarkalleen 0.05 sekuntia), jolloin pelin entiteetit päivittävät itseään (“Tick” 2011). Tämä triggeri määrää sen, että voiko zombie huomata pelaajan vai ei. Kun kyseinen triggeri on hyväksyttävästi laukaistu, zombie huomaa pelaajan, ja hyökkää tämän kimppuun. Zombiella on myös variantti: pikkuzombie. Pikkuzombie on pienempi, nopeampi, ja ääntelee korkeammalla äänentaajuudella. (“Zombie” 2009).

Pikkuzombie-luokka on todennäköisesti peritty alkuperäisestä zombie-luokasta, ja siihen on muokattu sen erityisominaisuudet. Perintää voidaan hyödyntää myös, kun luodaan ei-pelaajahahmoja, jotka eivät ole vihollisia. Nämä hahmot voivat olla nimettyjä, kutein usein roolipeleissä tavataan tehdä. Roolipeliä esimerkkinä käyttäen näiden hahmojen kanssa tapahtuva tyypillinen interaktio on puhuminen tai kauppojen teko. Jos ideana on, että jokaiselle hahmolle voi puhua, voi tämän määritellä rajapinnassa, tosin jos on kyse pelistä, missä ei ole yhtään geneeristä hahmoa, on kenties parempi käyttää luokkia. Luokassa voidaan määritellä tarkasti tekoälyä ja keskustelun optioita. Hahmojen perimisessä voidaan käyttää myös superluokkaa, johon perustuu kaikki pelin hahmot, jossa jokainen aliluokka perii ominaisuudet, kuten sijainnin, elämänpisteet ja metodit, kuten sijainnin päivitys ja törmäyksen tarkistukset (Huang, Zhang ja Xu 2018).

Polymorfismi sallii operaatiolle erilaisia muotoja, perustuen oliotyyppeihin (Huang, Zhang ja Xu 2018). Overwatch-pelissä on pelaajan ohjaama tarkka-ampujahahmo nimeltä Ana. Anan perustarkoitus on parantaa liittolaisiaan ampumalla heihin parantavia lääkepiikkejä. Ana kykenee myös kuitenkin vahingoittamaan vihollisia samalla aseella. Pelissä ei ole mitään erillistä ampumapainiketta parantaville, ja vahingoittaville ammuksille. Ana siis ampuu aina täysin samalla tavalla samoin ammuksin, mutta ammuksen vaikutus riippuu siitä, kuka sen vastaanottaa. ("Heroes: Ana" 2016). Tämä viittaa polymorfismin periaatteeseen, jossa olion lähettämän viestin aiheuttama lopputulos riippuu vastaanottavasta oliosta.

4.2.2 Esineet

Pelissä esiintyvät esineet, kuten aseet, bonukset, päivitykset, panssarit, panokset ym. voidaan implementoida eri tavoin riippuen esineestä. Yksinkertaisempia ovat pienet päivitykset pelaajahahmon ominaisuuksiin ja tilaan. Otetaan esimerkkiesineeksi elämänpisteitä parantava laatikko. Tämä esineolio toimisi vain yksinkertaisena 3D-mallina, jonka ainoat metodit voivat vähimmillään olla ovat tuhoutuminen, ja törmäyksen tarkistus. Jos esine on tuhoutumaton muuten, kuin pelaajan kerätessä sen, voitaisiin sen kerääminen ja sen vaikutukset ohjelmoida ilman varsinaista metodia, ainakin teoriassa. Mahdollisuus ohjelmoida näin riippuu ohjelmointialustasta ja pelimoottorista. Kun esine kerätään, eli se tuhoutuu pelimaailmasta, voidaan pelaajahahmon elämänpisteitä nostaa jollain määrällä. Eli käytännössä: pelaaja kerää parannuslaatikon, laatikko huomaa tämän, tuhoutuu, pelin triggeri huomaa tuhoutumisen, ja pelaajahahmo saa elämänpisteitä. Vaihtoehtoisesti, ja kenties luotettavammalla tavalla voitaisiin tehdä metodi pelaajan parantamiselle, samalla tavalla kuin pelissä voisi olla hahmoja, jotka parantavat pelaajaa. Tämä metodi aktivoituu, kun laatikko huomaa tulleensa kerätyksi, ja tuhoutuu vasta sen jälkeen pelimaailmasta.

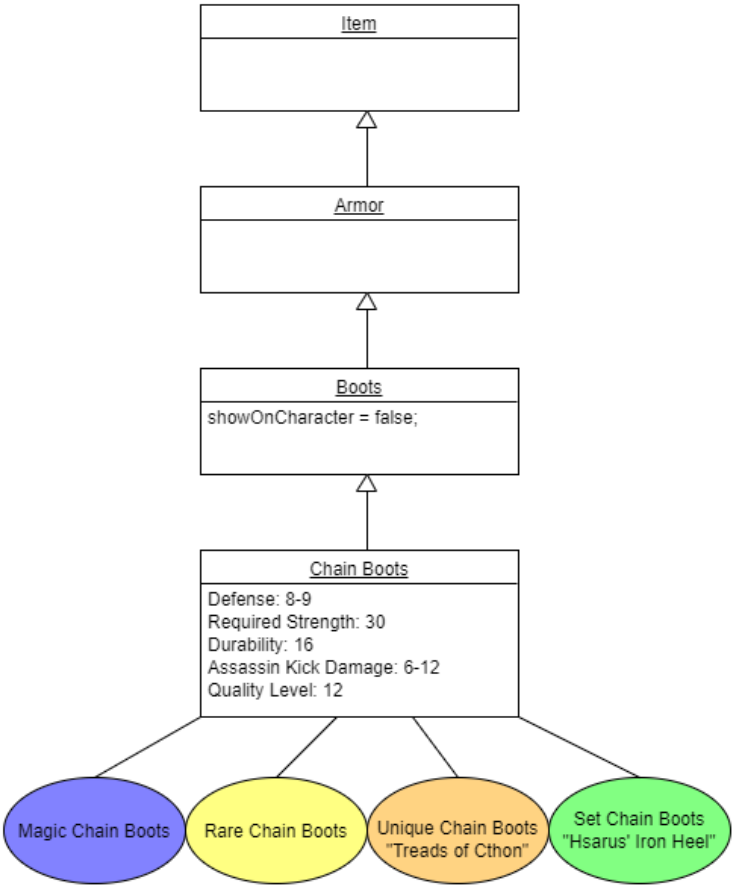
Monimutkaisempana esimerkkinä toimisi Diablo 2:n esineet, joista tärkeimmät ovat erilaisia aseita ja varusteita. Useimmilla Diablo 2:n esineillä on jonkinlainen pohja, joista pystyy ilmentymään parempia versioita kyseisestä tavarasta ("Basic Item Information" 2019). Useimmissa tapauksissa nämä versiot ovat magic (taika), rare (harvinainen), unique (ainutlaatuinen) ja set (setti). Otetaan esimerkiksi saappaat: Chain Boots-esine. Saappaat voivat siis ilmentyä joillain edellä mainituilla versioina. Ainutlaatuiset- ja settiesineet pysyvät ai-

na samanlaisina, pientä satunnaisuutta lukuunottamatta. Taika- ja harvinaiset esineet sen sijaan ovat aina täysin satunnaisilla ominaisuuksilla varustettuja. Nämä paremmat versiot pitävät pohjansa (Chain Boots) kaikki ominaisuudet, ja sisältävät myös lisäominaisuuksia. Teoriassa, kyseessä on siis moneen suuntaan haarautuva perintähierarkia, jonka superluokka on item-luokka, eli esineen määrittelevä luokka (kuvio 1).

4.2.3 Maailma

Pelimaailman entiteetit ovat yleensä sidoksissa kenttäeditoriin tai pelimoottoriin. Varsinkin koristeina toimivat entiteetit ovat asetettu peliin kenttäeditorilla tai pelimoottorilla, eivätkä yleensä noudata mitään muita metodeja. Taivaalla leijuvat koristepilvet eivät ole vuorovaikutuksessa pelaajan, tai minkään muun entiteetin, kuin itse pelikentän kanssa. Koriste-olioiden ainoa tarkoitus on siis vain olla olemassa, ja pelaajan havaittavissa. Ympäristön pelioliot voisi luokitella kahdella tavalla: vuorovaikutteinen, ja ei-vuorovaikutteinen, kuten koristeet.
Pelimaailman ympäristö ja sen kanssa vuorovaikutuksessa olo on monen pelin perusta. Tärkein tarkoitus pelimaailman olioilla on muovata pelialueesta tarkoituksenmukainen. Seinät, lattiat, katot, ikkunat, maa, kaikki, joilla on jotain tekemistä pelattavuuden kanssa ovat paras esimerkki. Joskus ympäristöä pystyy muokkaamaan jollain tavalla, kuten seinien tai ikkunoiden rikkominen. Tällöin ympäristön pelioliolla on oltava erilaiset, muiden peliolioiden kanssa vuorovaikutuksessa olevat attribuutit ja metodit. Vuorovaikutteisen ympäristön pelioliolla voisi siis tarkoituksesta riippuen olla esimerkiksi kuntopisteet. Tuhottava seinä ei tuhoutuisi heti yhdestä iskusta, vaan vaatisi lisää vaivaa. Törmäyksen havaitseminen löytäisi iskut, jotka vaikuttavat tuhottavaan ympäristöön. Polymorfismia voisi hyödyntää myös: samat iskut eivät välttämättä tee yhtä paljon vahinkoa seinään, kuin vaikkapa pelihahmoihin.
Ympäristön luominen pelissä voisi hyödyntää perintää. Jos pelihahmolla on kyky luoda seinä, voisi se periä ominaisuuksia pelimaailman muista seinistä. Tämä olisikin kenties suotavaa, sillä samat tutu säännöt pätisivät luotuihin seiniin. Törmäyksien havaitseminen ja mahdollinen tuhoutumattomuus olisivat olennaisia piirteitä.
Heroes of the Storm on ylhäältä päin kuvattu Starcraft 2-strategiapelin moottorilla tehty MOBA-peli. Pelissä on pelaajan ohjaama hahmo nimeltä Hogger. Hoggerin perustarkoitus on taistella kaksin vihollista vastaan ja hallita alueita. Hoggerilla on kyky pyöriä ja kimpoil-

la seinistä tehden vahinkoa vihollishahmoihin. Hoggerilla on myös kyky luoda ympäristöä, tarkalleen ottaen heittää maahan pieni neliön muotoinen kasa romua, joka toimii samalla tavalla, kuin mikä tahansa pelin tuhoutumaton seinä. Seinän läpi ei voi kävellä normaaleissa olosuhteissa, ja seinä reagoi niitä hyödyntäviin taitoihin, eli Hogger pystyy käyttämään romukasojaan kimmokkeina pyörimistaidossaan. Avustuksena Hoggerilla (tai jollain muulla ympäristöä hyödyntävällä hahmolla) pelatessaan pelaaja kykenee näkemään kaikkien seinien ääriviivat sisältäen Hoggerin itse luomat romukasat. (“There’s a Gnoll in the Nexus!” 2020). Hoggerin romukasat ovat todennäköisesti perineet pelin seinien ominaisuuksia. Samat ominaisuudet, kuten ääriviivojen näkyvyys ja vaikutukset muiden hahmojen taidoista viittaavat perintään tai rajapinnan toteutukseen.



Kuvio 1. Esimerkki Diablo 2-pelin tavarahierarkiasta. Chain Boots voi ilmentyä joko omana valkoisena versionaan, tai sitten sinisenä, keltaisena, kultaisena tai vihreänä versiona. Kuviosta puuttuu mm. harmaa (socketed)-esineversio, ja pelin muut tavaratyypit Boots-luokan lisäksi. Huom. kuvio ei ole täysin todenperäinen pelin koodin kanssa, eli esim. attribuuttia showOnCharacter tuskin löytyy pelin koodista, ainakaan kyseisellä nimellä.

5 Yhteenveto

Peleissä on ilmiselviä piirteitä oliosuuntautuneisuudesta. Jos ihminen ajattelee luonnostaan oliomaisella tavalla, vaikuttaa se myös pelikehitykseen, ja tätä kautta itse pelin läheisyyteen oliosuuntautuneisuuden kanssa. Pelin kehittäjä tietenkin tietää parhaiten, että onko peli tehty niitä oliosuuntautuneisuuden periaatteita noudattaen, mitä pelaajat saattavat olettaa oman kokemuksensa perusteella. Toisaalta peliä voi ajatella oliomaisesti, vaikka sitä ei olisikaan tehty niin kuin on oletettu, tai edes oliopohjaisella kielellä. Pelimoottorien viimeaikaista omistautumista oliopohjaisille kielille voitaisiin pitää hyvänä varmistuksena sille, että olio-ohjelmointi tulee olemaan yleisin pelien ohjelmointitapa myös tulevaisuudessa.

Asian tutkiminen voi olla kuitenkin hankalaa. On yleistä tietoa, että millaisia kieliä ja kirjastoja eri pelimoottorit käyttävät, mutta pelistudioiden tekemien työkalujen ohjelmistotausta tai pelien lähdekoodit eivät ole ison yleisön tiedossa. Lisäksi, jos tiedot perustuvat lähinnä olettamuksille, voi syntyä virheitä tiedon luotettavuudessa.

Oliosuuntautuneisuus on erittäin laaja käsite, jossa on paljon pelien kannalta pohdittavia asioita. Siksi tämänkin tutkielman aiheet oliosuuntautuneisuudesta rajattiin tarkoin. Tämän tutkielman aiheesta saisi helposti kirjoitettua vaikka kokonaisen kirjan. Olisi kenties helpompaa tutkia pelejä oliosuuntautuneisuuden kanssa yksi aihe kerrallaan. Eli vaikkapa esim. "Luokat peleissä", tai "Pelien olioiden polymorfismi", ja tarkentaa näin. Oman pelin prototyypin kehitys peliolioiden tutkimista varten olisi loistava tapa havainnollistaa oliosuuntautuneisuuden periaatteita peleissä.

Lähteet

"A Platform-based for Game Development to Improve The Object-oriented Programming Skills". 2016. https://www.researchgate.net/profile/Banyapon-Poolsawas/publication/291083368_A_Platform-based_for_Game_Development_to_Improve_The_Object-oriented_Programming_Skills/links/569df81d08ae950bd7a797dc/A-Platform-based-for-Game-Development-to-Improve-The-Object-oriented-Programming-Skills.pdf.

"Basic Item Information". 2019. http://classic.battle.net/diablo2exp/items/basics.shtml.

Bethke, Erik. 2003. *Game Development and Production.* 1. painos. Reading, MA: Wordware Publishing.

Black, Andrew P. 2013. "Object-oriented programming: Some history, and challenges for the next fifty years". *Information and Computation* 231 (1–2): 3–20. https://doi.org/10.1016/j.ic.2013.08.002.

"Game Design In Education". 2004. http://www.cs.uu.nl/research/techreps/repo/CS-2004/2004-056.pdf.

"Heads-up display". 2012. https://minecraft.fandom.com/wiki/Heads-up_display.

"Heroes: Ana". 2016. https://playoverwatch.com/en-us/heroes/ana/.

"How video games are made: the game development process". 2019. https://www.cgspectrum.com/blog/game-development-process.

Huang, Yanzhi, Ting Zhang ja Ling Xu. 2018. "The Development of a Game with Applications of Objectoriented Programming Concepts". *American Journal of Advanced Research* 2 (1): 7–13. https://doi.org/10.5281/zenodo.1410734.

"Interfaces". 2008. Viitattu 21. helmikuuta 2021. https://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html.

"Technical Tools for Authoring Branching Dialogue". 2019. https://www.youtube.com/watch?v=oRHl2PLKwfY.

“The 7 stages of Game Development”. 2019. https://www.g2.com/articles/stages-of-game-development.

“There’s a Gnoll in the Nexus!” 2020. https://news.blizzard.com/en-us/heroes-of-the-storm/23574265/there-s-a-gnoll-in-the-nexus-hogger-heads-to-heroes-of-the-storm.

“Tick”. 2011. https://minecraft.fandom.com/wiki/Tick.

“Triggers”. 2020. https://journey.fandom.com/wiki/Triggers.

“UnityScript’s long ride off into the sunset”. 2017. https://blog.unity.com/community/unityscripts-long-ride-off-into-the-sunset.

“Unreal Engine 4 - First Look”. 2012. https://web.archive.org/web/20120524062935/http://gameindustry.about.com/od/trends/a/Unreal-Engine-4-First-Look.htm.

“UnrealScript Classes”. 2012. https://docs.unrealengine.com/udk/Three/UnrealScriptClasses.html.

“What Coding Engine Does Blizzard Use?” 2022. https://www.mcnallyinstitute.com/what-coding-engine-does-blizzard-use/.

Wirfs-Brock, Rebecca, ja Alan McKean. 2003. *Object Design: Roles, Responsibilities, and Collaborations.* 1. painos. Reading, MA: Addison-Wesley Professional.

“Zombie”. 2009. https://minecraft.fandom.com/wiki/Zombie.