Kasimir Ilmonen

Pääsilmukoiden rakenne peleissä

Tietotekniikan kandidaatintutkielma

15. toukokuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Kasimir Ilmonen

Yhteystiedot: kajysail@student.jyu.fi

Työn nimi: Pääsilmukoiden rakenne peleissä

Title in English: Structure of game loops

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Tutkielmassa käsitellään peleissä käytettävien pääsilmukoiden rakennetta ja toimintaa. Tutkielmassa esitellään lyhyesti pääsilmukkaan liittyviä käsitteitä ja jaetaan pääsilmukan perusrakenne kolmeen vaiheeseen eli syötteeseen, päivitykseen ja palautteeseen. Perusmallin lisäksi tutkielmassa esitellään erilaisia monimutkaisempia pääsilmukkamalleja. Nämä kehittyneemmät mallit mahdollistavat muun muassa paremman suorituskyvyn ja säännöllisemmän toiminnan kuin vain perusmallilla voidaan saavuttaa.

Avainsanat: pelit, pääsilmukka, ohjelmointi

Abstract: This reseach tells about different structures of game loops. The research shortly presents terms connected to game loops and it divides game loops to three parts, which are input, update and output. Along with the simple model, the research presents more complicated models, which offer better performance and more deterministic behaviour than with the simple model.

Keywords: games, game loop, programming

Kuviot

Sisältö

1 Johdanto

Tässä tutkielmassa käsitellään peleissä käytettävien pääsilmukoiden rakennetta ja niiden toimintaa. Tutkielman alussa käsitellään perinteinen pääsilmukkamalli ja selitetään sen eri osien tehtävät. Tällä pyritään luomaan peruskäsitteistö, jota tullaan käyttämään myöhemmissä vaiheissa, kun käsitellään monimutkaisempia pääsilmukkamalleja.

Tutkielma painottuu vahvasti esittelemään erilaisia monisäikeisiä pääsilmukkamalleja, joilla pyritään saavuttamaan suorituskyvyllisiä ja laadullisia parannuksia suhteessa yksinkertaiseen pääsilmukkamalliin. Esiteltyjen mallien rakenne pyritään tuomaan esille ja lisäksi sen vaikutus pääsilmukan toimintaan. Toiminnallisista muutoksista pyritään erityisesti erittelemään kyseisen mallin vahvuudet ja myös siihen sisältyvät heikkoudet.

Tutkielman tarkoituksena on esitellä erilaisia pääsilmukkamalleja ja siten tarjota erilaisia mahdollisia ratkaisuja toteuttaa pääsilmukka esiteltyjen vahvuuksien ja heikkouksien perusteella. Työssä pyritään muodostamaan käsitys erilaisten pääsilmukoiden rakenteista ja eri lähestymistapojen hyödyistä.

Käytettävän pääsilmukkamallin valinta on sikäli tärkeää, että sillä voidaan vaikuttaa huomattavasti pelin suorituskykyyn ja myös pelin toimintalogiikkaan. Esimerkiksi tämän päivän tietokoneiden prosessoreista valtaosa on moniytimisiä ja siten monisäikeisyyttä tukevan pääsilmukkamallin valitseminen tuo huomattavia parannuksia pelin suorityskykyyn, mikä mahdollistaa entistä laajempien ja elämyksellisesti rikkaampien pelien toteuttamisen.

Tutkielman toisessa luvussa käsitellään perinteisen pääsilmukkamallin rakennetta ja pääsilmukoihin liittyviä käsitteitä. Kolmannessa luvussa esitellään erilaisia monisäikeisiä pääsilmukkamalleja ja tuodaan esille niiden keskinäiset eroavaisuudet. Neljännessä luvussa esitellään pääsilmukkamalli, joka rakenteellaan hyödyntää näytönohjaimen käyttöä pääsilmukkaan kuuluvien tehtävien suorittamisessa. Viidennessä luvussa esitellään eri mallien väliset eroavaisuudet ja kunkin mallin vahvuudet. Kuudes luku sisältää tutkielman yhteenvedon.

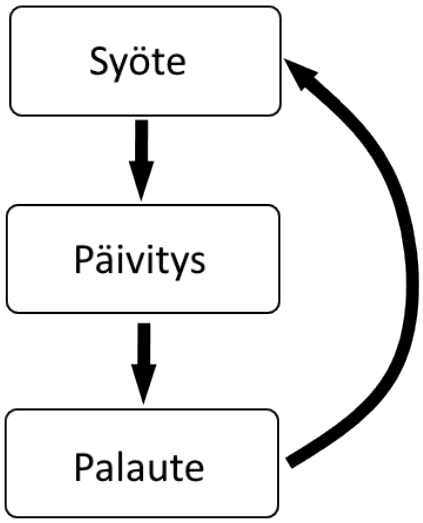
2 Pääsilmukan perusmalli

Useimpien pelien perusominaisuus on, että niiden toiminnallisuus ei ole riippuvaista käyttäjän syötteistä, vaan peli pystyy toimimaan ilman, että käyttäjä tarjoaa minkäänlaista syötettä. Pelin pitää pystyä päivittämään pelin tilaa ja muodostamaan pelaajalle palaute uudesta pelin tilasta. Tämän takia peli ei voi jäädä odottamaan käyttäjän syötettä vaan peli jatkaa toimintaansa pelilogiikan suoritukseen ja syötteen muodostamiseen. Tätä varten pelien toiminta sijoitetaan yhteen tai useampaan pääsilmukkaan, joiden vastuulla on suorittaa pelin toimintoja. Tässä luvussa käsitellään yksinkertaisen pääsilmukan rakennetta ja toimintaa. (McShaffry 2012)

2.1 Pääsilmukan osat

Pääsilmukka rakentuu yleensä syötteen vastaanottamisesta, pelin tilan päivittämisestä ja palautteen antamisesta (Valente, Conci ja Feijó 2005). Kuva 1 havainnollistaa tätä rakennetta. Syötteen vastaanottamisvaiheessa peli kerää tietoja, joiden pohjalta pelilogiikka päivittää pelin tilaa. Käytännössä tarkistetaan, onko käyttäjältä tullut syötettä, kuten hiiren liikuttamista tai näppäinten painamista, ja laitetaan mahdolliset syötteet muistiin seuraavien vaiheiden käytettäväksi. Lisäksi myös verkkoyhteyden kautta tulevat viestit voidaan nähdä osana syötteiden vastaanottamista, sillä se voi myös sisältää pelin päivittämisen kannalta oleellista tietoa (Tulip, Bekkema ja Nesbitt 2006).

Pelin päivitysvaiheessa muodostetaan saadun syötteen ja aiemman pelitilan pohjalta pelille uusi pelitila. Vaikka päivityksessä käytetäänkin käyttäjältä saatuja syötteitä, siihen kuuluu myös osa-alueita, jotka päivittyvät vaikka käyttäjältä ei saataisikaan syötettä, tai jotka eivät edes ole käyttäjän syötteestä riippuvaisia. Pelin päivitykseen kuuluu muun muassa tekoälyn toimet, fysiikan mallinnus ja pelin sisäiseen logiikkaan kuuluvat toimet (Valente, Conci ja Feijó 2005). Päivitysvaihe voidaan jakaa kahteen osaan, jossa ensimmäisessä osassa saadaan alustavasti uusi pelitila, ja toisessa osassa ratkotaan ensimmäisessä osassa ilmenneitä ristiriitoja, joita muodostuu kokonaisuuksien välisistä vuorovaikutuksista. (Tulip, Bekkema ja Nesbitt 2006). Tällaisia on esimerkiksi fysiikan mallinuksessa ilmenevät törmäystapahtu-



Kuvio 1. Pääsilmukan perusmalli

mat.

Pääsilmukan viimeisessä vaiheessa eli palautevaiheessa muodostetaan päivityksestä saadusta uudesta pelitilasta käyttäjälle palaute, joka useimmiten on pelikuvaa ja -ääntä (Zamith ym. 2009). Käyttäjälle voidaan kuitenkin antaa palautetta myös muilla tavoilla kuten esimerkiksi peliohjaimen värinänä. Palautevaihe voidaan kuitenkin nähdä pelin toiminnan kannalta tarpeettomana, sillä se ei vaikuta pelitilaan millään lailla. Tämän vuoksi palautevaihetta ei tarvitse suorittaa jokaisella pääsilmukan kierroksella ja palaute voidaankin suorittaa vain niin usein kuin on tarpeellista sujuvan pelikokemuksen kannalta.

2.2 Pääsilmukan päivitysnopeus

Jos pääsilmukan annetaan edetä niin nopeasti kuin mahdollista, ilmenee ongelmia eri laitekokonaisuuksien välillä. Esimerkiksi nopeammalla tietokoneella, pelihahmo voi liikkua huomattavasti nopeammin kuin hitaammalla tietokoneella, jos liikkumisnopeus on suoraan sidoksissa pääsilmukan kierrosnopeutteen. Näin käy, jos esimerkiksi käytetään sellaisia päivityksiä, joissa siirretään kohdetta joka kierroksella vakiomäärä yksiköitä. (Gregory 2009, ss. 313)

Jotta vältetään pelilaitteiden suorituskyvyn aiheuttamat erot, voidaan päivitykset sitoa kier-

rosten sijasta aikaan. Tämä onnistuu käyttämällä arvioitua kierrokseen kuluvaa aikaa ja laskemalla sen pohjalta suoritettava muutos (Gregory 2009). Jatketaan kohteen siirtämisesimerkkiä. Siinä siirretään kohdetta vakiomäärä yksiköitä jokaista kulunutta mikrosekuntia kohden. Tällöin vaikka peliä pelataan eri pelilaitteilla, muutokset ovat samassa ajassa yhtä suuria, ja erona on vain, kuinka pienissä erissä muutosta päivitetään (Valente, Conci ja Feijó 2005).

Edellä mainittu arvioitu kierrosaika voidaan saada useammalla eri tavalla, joista yksinkertaisinta on ottaa kunkin kierroksen alussa ja lopussa aika talteen ja niistä erotus, jolloin kullakin kierroksella voidaan käyttää edelliseen kierrokseen kulunutta aikaa. Tässä ratkaisussa ei kuitenkaan huomioida sitä, että kierrosten väliset nopeuspoikkeamat voivat olla huomattaviakin, mikä voi aiheuttaa erinäisiä piikkejä päivityksen tuotoksissa. Jotta saadaan tasattua yksittäisten poikkeamien aiheuttamaa vaikutusta, voidaan käyttää yksittäisen edellisen kierroksen sijaan useiden edellisten kierrosten keskiarvoa, mikä tasaa ilmiötä. (Gregory 2009, ss. 314)

Toisaalta, koska näyttölaitteen ruudunpäivitysnopeus on rajallinen tai sen nostaminen ei tuo suurtakaan hyötyä, voidaan rajoittaa palautteen antaminen, eli pelikuvan muodostamisen, toteutumaan vain tietyin väliajoin, mikä useimmiten on 25, 30, 50 tai 60 kertaa sekunnissa. Näin säästytään turhalta laskennalta ja siten säästyy resursseja ilman, että pelin laatu kärsii. (Gregory 2009)

Lisäksi myös monet päivitystoiminnot voidaan sitoa johonkin tiettyyn tahtiin. Tämä onnistuu asettamalla pääsäie nukkumaan, jos on jo onnistuttu suorittamaan kaikki toiminnot nopeammin kuin tavoiteajassa. Tällöin ei tarvitse tarkkailla samalla tavalla aiempien kierrosten suoritusnopeuksia, sillä voidaan olettaa kierrosnopeuden olevan vähintään haluttu nopeus, mikä tekee silmukan toiminnasta ennakoitavampaa (Gregory 2009). Valente, Conci ja Feijó 2005 mukaan, koska tiedetään kunkin kierroksen nopeus, voidaan toistaa pelitilan muutokset täsmällisesti pelkästään syötteen ja alkutilan perusteella. Hyötynä mainitaan, että se tekee virheiden etsimisestä helpompaa ja yksinkertaistaa pelin tapahtumien toistamiseen liittyvien toimintojen toteuttamisen. Myös verkkopelin toteuttaminen voi olla helpompaa, sillä pelaajien välinen synkronisointi pohjautuu pelkästään syötteiden käsittelyyn kullakin kierroksella (Valente, Conci ja Feijó 2005). Toisaalta pitää vielä jotenkin käsitellä tilanteet, joissa suo-

ritusaika on tavoiteaikaa hitaampi, jolloin ollaan jääty jälkeen halutusta ajasta.

3 Monisäikeiset pääsilmukat

Edellisessä luvussa käsiteltiin yksinkertaista pääsilmukkamallia, joka on täysin toimiva ja ihan hyvin käytettävissä oleva. Kuitenkin nykypäivän tietokoneissa ja pelilaitteissa käytetään yhä enenemässä määrin useampi ytimisiä prosessoreita, joten yksisäikeiset pääsilmukkarakenteet eivät ole ihanteellisia, kun pyritään tehokkuuteen. Tämän takia on suotavaa pyrkiä pääsilmukkaa suunniteltaessa suoraan monisäikeisyyden mahdollistamiseen ja pyrkiä erottamaan toisistaan riippumattomia kokonaisuuksia (Rauber 2013). Tässä luvussa käydään läpi vaihtoehtoisia pääsilmukan malleja, joiden vahvana etuna on niiden monisäikeisyys.

3.1 Haarautettu säännöllinen -malli

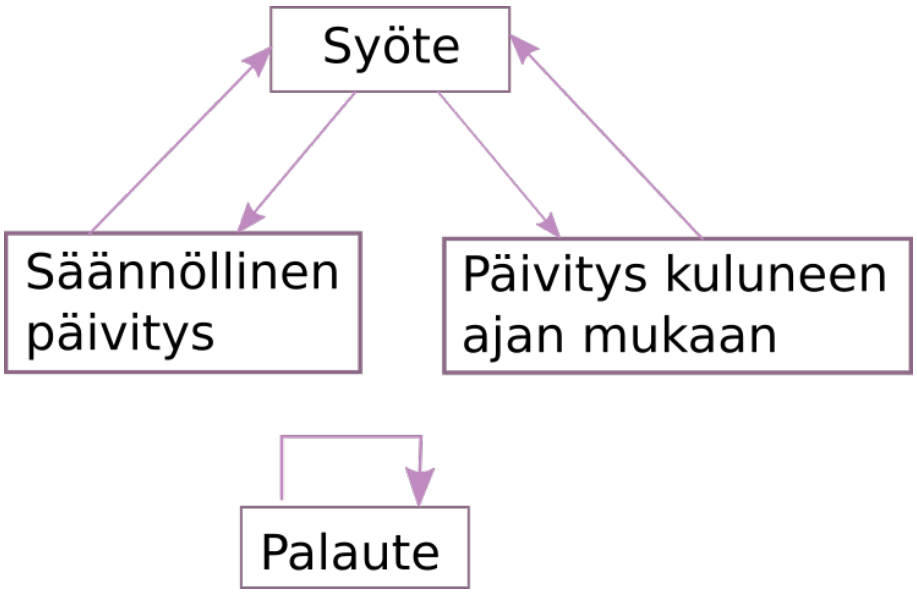
Luvussa 2 käsiteltyä pääsilmukan perusmallia voidaan kehittää hyödyntämällä monisäikeisyyttä ja perusmallin rakennetta muuttamalla. Valente, Conci ja Feijó 2005 esittelevät perinteisestä pääsilmukka -rakenteesta johdetun mallin, joka pyrkii tarjoamaan paremman pelikokemuksen.

Esitellyssä mallissa on palautteen antaminen irrotettu omaan säikeeseensä, jolloin peli voi suorittaa ruudunpäivityksen pelilogiikkaan kuluvasta ajasta riippumattomasti, jolloin pelaajalle tarjotaan paras mahdollinen ruudunpäivitysnopeus. Tällöin palautteen antaminen pelaajalle muodostetaan aina viimeisimpien pelitilanteesta olevien tietojen perusteella. Pelitilannetta päivitetään toisesta pelilogiikan ja syötteen keräämisen sisältävästä säikeestä. (Valente, Conci ja Feijó 2005)

Varsinainen pelilogiikan päivittäminen on jaettu kahteen haaraan, joista ensimmäinen suoritetaan säännöllisin väliajoin ja toinen haara taas niin usein kuin mahdollista (katso kuva 2). Ensimmäisessä haarassa siis pyritään hyödyntämään säännöllisyydestä saatavaa luvussa 2.2 mainittua ennakoitavuutta, joka helpottaa huomattavasti tietynlaisten toimintojen toteuttamista. (Valente, Conci ja Feijó 2005)

Toisessa haarassa taas toteutetaan toiminnot ennalta määrätyn ajan sijaan edellisestä suorituskerrasta kulunutta aikaa käyttäen (katso luku 2.2). Jotta pelin toiminta pysyisi säännöl-

lisenä, sijoitetaan tähän haaraan sellaiset toiminnot, jotka eivät vaikuta itse pelilogiikkaan vaan pikemminkin ovat siitä irrallisia. Esimerkiksi animaation päivittäminen ei vaikuta pelilogiikkaan, mutta mitä useammin animaatioita päivitetään, niin sitä parempi on pelikokemus. (Valente, Conci ja Feijó 2005)



Kuvio 2. Haarautettu säännöllinen -malli

Toimintojen jakamisella kahteen haaraan onnistutaan säilyttämään pelikokemuksen samanlaisena, vaikka peliä pelattaisiin eri tehoisilla laitteistoilla, mutta kuitenkin pelikokemus voi parantua esimerkiksi sujuvammin päivittyvien animaatioiden ansiosta (Valente, Conci ja Feijó 2005). Esitellyssä mallissa ei kuitenkaan hyödynnetty monisäikeisyydestä saatavia suorituskyvyllisiä etuja, mikä johtunee siitä, että malli on esitelty aikana jolloin moniytimiset prosessorit olivat huomattavasti harvinaisempia ja siten tarkastelun ulkopuolella.

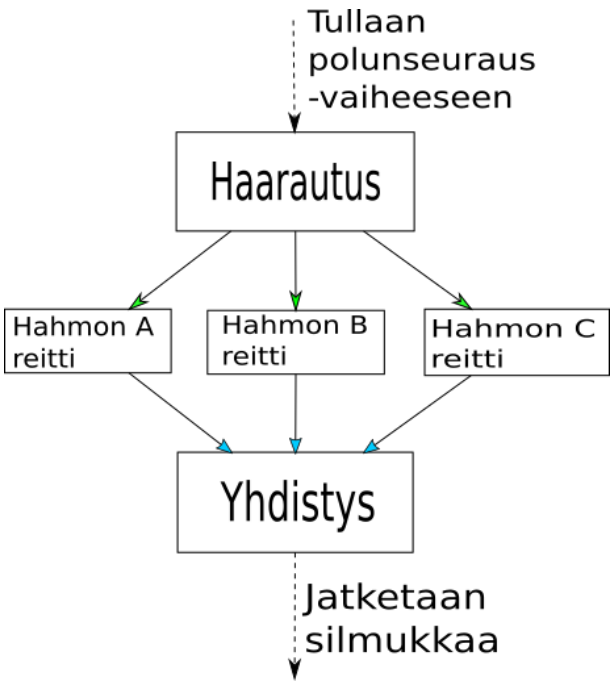
3.2 Haarauta ja yhdistä -malli

Yksinkertaisimpia tapoja hyödyntää monisäikeisyyttä pääsilmukassa on jakaa monisäikeisyyteen soveltuvat tehtävät pienemmiksi rinnakkaisiksi osatehtäviksi. Tässä Gregory 2009 kuvaamassa mallissa noudatetaan perinteisen yksisäikeisen pääsilmukkamallia, mutta yksittäisten tehtävien kohdalla voidaan aina haarauttaa useita säikeitä suorittamaan samaa tehtävää (katso kuva 3). Esimerkiksi jos pelissä on useita pelihahmoja, voidaan kaikille näille

suorittaa polunseurausalgoritmi samanaikaisesti.

Esitellyn mallin luonteeseen kuuluu, että säikeet ovat tilapäisiä, sillä tehtävän suorittamisen jälkeen tulos yhdistetään ja ylimääräiset säikeet poistetaan jättäen jäljelle vain pääsäikeen (Gregory 2009). Tällainen säikeiden luominen ja poistaminen on kuitenkin itsessään työlästä ja säikeiden hallinta itsessään voi liian pienien tehtävien kohdalla kuluttaa kaiken saadun hyödyn. Tässä auttaa kuitenkin, jos jaamme tehtävät säikeille ryhmittäin (Gregory 2009). Esimerkiksi annamme kullekin säikeelle kymmenen pelihahmon polunseurausalgoritmin suorittamisen, jolloin ei tarvita niin montaa säiettä, jolloin säikeidenhallinnan osuus vähenee.

Tarkastelemalla tätä mallia huomataan sen vahvuutena samankaltaisuus perinteisen yksisäikeisen pääsilmukkamallin kanssa. Tällöin voidaan yksisäikeisellä mallilla toteutettu peli muuttaa hyödyntämään tässä esiteltyä mallia, kuten voidaan tehdä luvussa 3.1 esitellylle mallille, jolloin päästään nopeasti hyötymään monisäikeisyydestä. Myös mallin yksinkertaisuus on vahva etu, sillä pelistä on helppo löytää toimintoja, jotka suoritetaan useaan kertaan, mutta vain eri kohteelle.



Kuvio 3. Haarauta ja yhdistä -malli

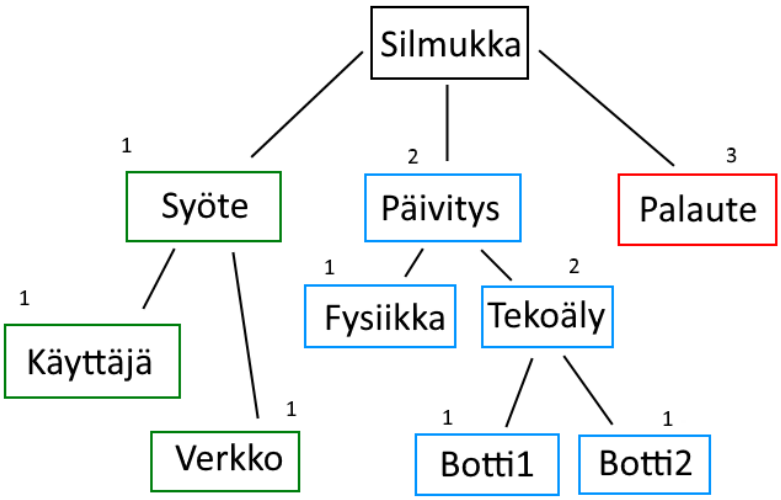
Voidaan myös havaita, että mallissa ei hyödynnetä monisäikeisyyttä tilanteissa, joissa tehtävät ovat toisistaan poikkeavia, mutta kuitenkin toisistaan riippumattomia. Tällöin pysähdytään aina suorittamaan yhtä tehtävätyyppiä, kunnes viimeinenkin säie on saanut oman osansa tehtyä ja vasta sitten jatketaan eteenpäin. Tämän takia malli ei juurikaan skaalaudu tasaisesti tehtävien lisääntyessä vaan auttaa vain yksittäisiä tehtäväkokonaisuuksia suoriutumaan nopeammin.

3.3 Tehtäväallas-malli

Eräänä monisäikeisen pääsilmukan mallina voidaan käyttää tutkijoiden Tulip, Bekkema ja Nesbitt 2006 artikkelissa käytettyä mallia. Kyseinen malli pohjautuu siihen, että pääsilmukan sisällä olevat tehtävät jaetaan kohtuullisen pieniin osatehtäviin, ja näitä tehtäviä suorittamaan luodaan prosessorin ydinten määrään perustuen säikeitä. Säikeet suorittavat tehtäviä vapaasti ja aina edellisen tehtävän suoritettuaan, ne ottavat uuden tehtävän suoritettavakseen. Näin tehtäviä voidaan suorittaa useita samanaikaisesti ja työ jaetaan automaattisesti tasaisesti kaikille prosessorin ytimille. (Tulip, Bekkema ja Nesbitt 2006)

Jotta tehtävät suoritetaan järjestyksessä, josta ei aiheudu ristiriitoja, annetaan jokaiselle tehtävälle sen suoritusjärjestystä kuvaava luku. Tämän luvun perusteella voidaan jakaa tehtävät säikeille halutussa järjestyksessä. Lukuja jaetaan suurempien osatehtävien perusteella eli kaikille samaan kokonaisuuteen kuuluvilla osatehtävillä on aina sama luvun alkuosa. Saman tasoisilla toisistaan riippumattomilla tehtävillä on taas sama luku, koska niiden keskinäisellä järjestyksellä ei ole merkitystä. Näin muodostuu tehtäväpuu, jonka haarat kuvastavat suurempaa kokonaisuutta ja lehdet taas kuvastavat yksittäisiä suoritettavia tehtäviä (katso kuva 4). (Tulip, Bekkema ja Nesbitt 2006)

Tämän ratkaisun etuna on, että se jakaa tasaisesti kuormituksen prosessorin ytimien välillä ja toimii hyvin, jos pystytään löytämään pienempiä rinnakkaisia osatehtäviä (Tulip, Bekkema ja Nesbitt 2006). Jotta tätä mallia voidaan käyttää, tulee tehtävien kuitenkin olla sellaisia, että ne ovat toisistaan vahvasti riippumattomia ja eivät käytä samoja resursseja yhtäaikaisesti, mikä voi aiheuttaa erinäisiä häiriöitä, kun pyritään käsittelemään samoja tietoja yhtäaikaisesti.



Kuvio 4. Esimerkki tehtäväpuusta

3.4 Prosessipohjainen malli

Vaihtoehtoisesti pääsilmukassa olevia tehtäviä, kuten polunseurausta tai animaatioita, voidaan käsitellä prosesseina, joilla kullakin on oma tilansa. Tila kertoo onko prosessi aktiivinen, pysäytetty vai onko se saavuttanut jonkin lopputiloista. Prosessien toimintaa valvoo tehtävien hallinta (task manager), joka pitää yllä listaa kaikista prosesseista ja ajaa jokaisella pääsilmukan suorituskierroksella kaikkien aktiivisten prosessien päivitystoiminnon. Esimerkiksi polunseurauksen tapauksessa tämä olisi pelihahmon liikuttamista. (McShaffry 2012)

Jokaisella prosessilla voi lisäksi olla lapsiprosessi, joka otetaan osaksi suoritettavien prosessien listaa, jos vanhempiprosessi saa lopputilakseen onnistumisen. Epäonnistumisen kohdalla taas lapsiprosessia ei oteta osaksi suoritettavia prosesseja. Lapsiprosesseja voidaan kuitenkin lisätä vapaasti prosesseille käyttämällä tehtävien hallintaa jolloin voidaan luoda tehtäväketjuja. Esimerkiksi polunseurauksella voi olla lapsena juomisanimaatio, jolla taas on lapsena elämäpisteiden (hit points) parantuminen. Tästä seuraisi siis, että hahmo kävelisi juomapaikalle, suorittaisi juomisanimaation ja sitten pelihahmon elämäpisteet nousisivat. (McS-

haffry 2012)

Tällainen toteutusratkaisu muodostaa helpon tavan ketjuttaa tehtäviä, mikä helpottaa jopa monimutkaisten tehtäväkokonaisuuksien luomista. Tehtäväketjut voidaan myös lukea tiedostosta esimerkiksi XML eli Extensible Markup Language -formaattia käyttäen, mikä mahdollistaa erilaisten editorien tekemisen. (McShaffry 2012)

Edellä esitelty malli ei kuitenkaan suoraan tue rinnakkaisuutta, sillä ei tiedetä, mitkä prosessit ovat toisistaan riippumattomia. Tätä varten tulee luokitella prosessit prioriteetin mukaan toisistaan riippumattomiin ryhmiin luvussa 3.3 esitellyllä tavalla. Myös muut ratkaisut ovat mahdollisia.

Tämän ratkaisun etuna on helppo muokattavuus ja, jo edellä mainittu, pitkienkin tapahtumaketjujen luominen vaivattomasti. Myös tätä tekniikkaa käyttävän editorin käyttö keventää huomattavasti ohjelmoijan työtaakkaa ja pelisuunnittelijat saavat vapaat kädet toteuttaa visioitansa. Toisaalta taas heikkoutena on mallin monimutkaisuus, mikä näkyy erityisesti monisäikeisyyteen pyrittäessä.

4 Näytönohjaimen hyödyntäminen pääsilmukassa

Koska näytönohjaimissa on huomattavasti enemmän ytimiä kuin prosessoreissa, ne soveltuvat paremmin tilanteisiin, joissa pitää suorittaa suuria määriä toisistaan riippumattomia yksinkertaisia tehtäviä (Rauber 2013). Tämän takia peleissä, joissa suoritusnopeus on olennaisessa osassa, voidaan pyrkiä parempaan tehokkuuteen käyttämällä näytönohjainta prosessorin sijasta joidenkin tehtävien suorittamiseen. Näytönohjaimen käyttö voidaan ottaa huomioon yksittäisten tehtävien sijasta koko pääsilmukan rakenteessa, jolloin saamme pääsilmukasta paremmin skaalautuvan pääsilmukan tehtävien määrän kasvaessa.

Eräänä keinona hyödyntää näytönohjainta pääsilmukkaa laatiessa on käyttää tutkijoiden Joselli ym. 2010 esittelemää mallia. Kyseisessä mallissa pyritään automaattisesti jakamaan tehtävät mahdollisimman tehokkaasti prosessorin ytimien ja näytönohjaimen välillä, minkä pohjana on luvussa 3.3 esiteltyyn tapaan jakaa pääsilmukan tehtävät kohtuullisen pieniin osatehtäviin. Kyseisen mallin erona kuitenkin on, että jokaiselle tehtävälle määritellään, voiko sen toteuttaa prosessorilla, näytönohjaimella tai kummalla tahansa. Esimerkiksi tiedostoihin kirjoittaminen on vain prosessorin tehtävissä, kun taas kuvan renderöinti on käytännössä vain näytönohjaimelle mahdollista. Monet yksinkertaiset laskennalliset tehtävät taas voivat olla molemmilla suoritettavissa.

Koska Joselli ym. 2010 -artikkelissa esitelty ja Zamith ym. 2009 -artikkelissa laajennettu malli on tehtävien jaon suhteen monimutkaisempi kuin luvun 3.3 malli, on siinä otettu käyttöön luvussa 3.4 esitellyn mallin tapaan tehtävien hallinta (task manager). Tehtävien hallinta huolehtii tehtävien jakamisesta säikeille ja huolehtii tehtävien suoritusjärjestyksestä riippuvuuslistan (dependency list) avulla. Ei kuitenkaan ole varsinaisia esteitä havaittavissa, ettei tätä voitaisi tehdä luvussa 3.3 käytetyn tehtäväpuun avulla.

Tehtävien hallinnalla ei kuitenkaan pyritä pelkästään tehtävien suorittamiseen oikeassa järjestyksessä, vaan lisäksi sen ominaisuuksiin kuuluu määrittää, suoritetaanko tehtävä käyttäen näytönohjainta vai prosessoria. Tämä määrittely tehdään luonnollisesti sen mukaan, kumpi toteutus tehtävälle on ohjelmoitu. Jos molemmat vaihtoehdot ovat käytettävissä, silloin hyödynnetään erinäisiä heuristiikoita, joiden pohjalta osataan antaa tehtävä suoritettavaksi joko

prosessorille tai näytönohjaimelle. (Joselli ym. 2010)

Erinäisinä tekijöinä tätä valittaessa voidaan käyttää aiempien suorituskertojen tuloksia eli kuinka nopeasti prosessori tai näytönohjain on aikaisemmin suorittanut kyseisen tehtävän. Myös se, että onko prosessorin ytimiä tai näytönohjain vapaana käytöstä on yksi ratkaisuperusteista. Näiden lisäksi heuristiikoita voidaan muokata tilanteeseen sopivammiksi ja tehokkaammin toimivimmiksi. (Joselli ym. 2010)

Esitellyn mallin etuna voidaan nähdä näytönohjaimen käytöstä saatava suorituskyvyn kasvu ja se että ohjelma pystyy itse jakamaan tehtävät tasaisesti prosessorin ja näytönohjaimen välillä. Myös mallin hyvin pitkä samankaltaisuus luvussa 3.3 esitellyn mallin kanssa saattaa olla hyödyllistä, koska voidaan laajentaa tehtäväallas-malli käyttämään näytönohjainta joissakin tehtävissä hyödyksi. Toisaalta kuitenkin näytönohjaimen käyttäminen saattaa olla haastavaa ja myös esiteltyjen heuristiikkojen ohjelmoiminen voi olla työlästä, erityisesti kun otetaan huomioon, että joudutaan tekemään kaksi erilaista toteutusta samalle tehtävälle, joka on sekä prosessorin ja näytönohjaimen suoritettavissa.

5 Esiteltyjen mallien vertailu

Esitellyillä malleilla on kullakin omat vahvuutensa ja heikkoutensa. Tässä luvussa tarkastellaan esiteltyjen mallien eroavaisuuksia ja yhtäläisyyksiä.

Luvussa 3.1 esitelty haarautettu säännöllinen malli on pohjaltaan laajennettu versio pääsilmukan perusmallista, jossa on eroteltu palautteen antaminen omaan säikeeseen. Haarautettu säännöllinen malli ei ole toteutuksena erityisen haastava, mutta se tekee pelin toiminnasta säännöllisempää ilman, että pelin laatu kärsii siitä. Kyseinen malli ei kuitenkaan sellaisenaan tuo suorituskyvyllisiä parannuksia, sillä se hyödyntää vähän monisäikeisyyttä.

Haarauta ja yhdistä -malli (katso luku 3) on puolestaan yleisesti käytettävä malli, joka on helppo yhdistää moniin muihin lineaarisesti eteneviin malleihin, kuten haarautettu säännöllinen -malliin. Oikein käytettynä tämä malli parantaa suorityskykyä, mutta jatkuva säikeiden luominen osaltaan heikentää saatuja hyötyjä. Malli on kuitenkin arvioidusti helppo toteuttaa ja siten siitä saadut hyödyt ovat vähällä työllä saavutettavissa.

Tehtäväallas-malli (katso luku 3.3) on perinteisestä pääsilmukkamallista vahvasti poikkeava ja siksi se voi olla kohtuullisen hankalaa toteuttaa. Kuitenkin malli hyödyntää vahvasti monisäikeisyyttä ja siten tuo merkittäviä parannuksia suorituskykyyn, kuitenkin ilman jatkuvaa säikeiden luontia ja tuhoamista kuten haarauta ja yhdistä -mallissa. Malli on myös skaalautuu hyvin, mikä auttaa sitä säilyttämään suorityskyvyssä saavutetun tehokkuuden tehtävämäärän kasvaessa.

Prosessipohjainen malli (katso luku 3.4) tekee pelin kehittämisen helpommaksi, sillä se mahdollistaa tapahtumien ketjuttamisen yksinkertaisesti toteutettavalla tavalla. Kyseinen malli myös vahvasti tukee tapahtumaketjujen luomisessa käytettävien editorien hyödyntämistä ja on siten kehittäjälle hyödyllinen malli. Kuitenkin malli ei itsessään tuo suorituskykyyn parannuksia, mutta yhdistämällä johonkin toiseen malliin se on mahdollista saavuttaa.

Näytönohjainta hyödyntävässä mallissa (katso luku 4) hyödynnetään pitkälti samoja periaatteita kuin tehtäväallas-mallissa ja on suurelta osin laajennus siihen. Tämän ansiosta kyseinen malli saavuttaa samat suorituskyvylliset edut kuin tehtäväallas-malli. Lisäksi se hyödyntää

tehokkaasti näytönohjainta pääsilmukan tehtävien suorittamisessa ja siten mahdollistaa paremman suorituskyvyn kuin mitä pelkällä tehtäväallas-mallilla voitaisiin saavuttaa.

6 Yhteenveto

Tutkimuksessa esiteltiin perinteisen pääsilmukan rakennetta ja selitettiin sen osien eli syötteen ottamisen, päivityksen ja palautteen antamisen tarkoitukset. Myös pääsilmukoissa käytettävän kierrosnopeuden hallintaa käsiteltiin ja siihen liittyen säännöllisestä kierrosnopeudesta saatavia hyötyjä.

Tutkimuksessa onnistuttiin myös esittelemään erilaisia monisäikeisiä pääsilmukkamalleja kuten haarautettu säännöllinen -malli, haarauta ja yhdistä -malli, tehtäväallas-malli ja prosessipohjainen malli. Lisäksi esiteltiin näytönohjainta hyödyntävä malli. Näiden mallien eroavaisuuksia tuotiin esiin ja eri malleja vertailtiin keskenään.

Lyhyesti sanottuna perusmallia laajentava haarautettu säännöllinen -malli tuo mukanaan säännöllisen toiminnan, mikä helpottaa useiden erilasten toimintojen toteutusta. Haarauta ja yhdistä -malli on yleisesti käytettävä ja helposti yhdistettävä malli, joka tuo suorituskykyyn parannuksia. Tehtäväallas-malli on taas suorituskykyä tehokkaasti parantava malli, joka myös skaalautuu hyvin tehtävämäärän kasvaessa. Prosessipohjainen malli puolestaan mahdollistaa tapahtumien ketjuttamisella vaivattoman pelin sisällön luomisen. Tehtäväallas-mallia laajentava näytönohjainta hyödyntävä malli parantaa saatavaa suorituskykyä entisestään.

Lähteet

Gregory, Jason. 2009. *Game engine architecture.* A K Peters.

Joselli, Mark, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, Regina Leal-Toledo, Aura Conci, Paulo Pagliosa, Luis Valente ja Bruno Feijó. 2010. "An Adaptative Game Loop Architecture with Automatic Distribution of Tasks Between CPU and GPU". *Comput. Entertain.* (New York, NY, USA) 7, numero 4 (tammikuu): 50:1–50:15. ISSN: 1544-3574. http://doi.acm.org/10.1145/1658866.1658869.

McShaffry, Mike. 2012. *Game coding complete.* Cengage Learning.

Rauber, Thomas. 2013. *Parallel Programming : for Multicore and Cluster Systems.* 2nd ed. 2013. Toimittanut SpringerLink (Online service). 516. Berlin, Heidelberg: Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-37801-0.

Tulip, James, James Bekkema ja Keith Nesbitt. 2006. "Multi-threaded Game Engine Design". Teoksessa *Proceedings of the 3rd Australasian Conference on Interactive Entertainment,* 9–14. IE '06. Perth, Australia: Murdoch University. ISBN: 86905-902-5. http://dl.acm.org/citation.cfm?id=1231894.1231896.

Valente, Luis, Aura Conci ja Bruno Feijó. 2005. "Real time game loop models for single-player computer games". Teoksessa *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment,* 89:99.

Zamith, Marcelo, Mark Joselli, Luis Valente, Esteban Clua, Anselmo Montenegro, Regina Celia P Leal-Toledo ja Bruno Feijo. 2009. "A game loop architecture with automatic distribution of tasks and load balancing between processors". *Proceedings of SBGames:* 5–8.