// Esitellään gradun nimi

Ohjelmointikoodin kommenttien tekstilaji

Elina Perkiö
Tampereen yliopisto
Kieli-, käännös- ja kirjallisuustieteiden yksikkö
Suomen kielen tutkinto-ohjelma
Pro gradu -tutkielma
Joulukuu 2015

Tampereen yliopisto
Kieli-, käännös- ja kirjallisuustieteiden yksikkö
Suomen kieli

PERKIÖ, ELINA: // Esitellään gradun nimi. Ohjelmointikoodin kommenttien tekstilaji

Pro gradu -tutkielma, 95 sivua
joulukuu 2015

Pro gradu -työn tavoitteena on selvittää, millaisia ovat ohjelmointikoodin yhteydessä esiintyvät kommentit tekstintutkimuksen näkökulmasta. Huomion kohteena on erityisesti se, mitkä piirteet määrittävät kommenttien tekstilajia, sekä se, miten kommentit toimivat yhdessä koodin kanssa. Ohjelmointikoodin kommentit ovat luonnollisella kielellä kirjoitettuja lyhyitä tekstejä, jotka sijoittuvat ohjelmointikoodin sekaan. Ohjelmoinnin oppikirjamääritelmän mukaan kommenttien tehtävänä on selventää ohjelman toimintaa ihmisille, jotka lukevat koodia.

Työn tutkimuskysymykset ovat seuraavat: Millaisen tekstilajin ohjelmointikoodin kommentit muodostavat? Millaisessa suhteessa kommentit ovat niiden kommentoimaan koodiin? Millaiset tekijyyteen, maailman kuvaamiseen ja koheesiokeinoihin liittyvät piirteet määrittävät kommentteja?

Tutkimusaineisto koostuu yliopiston ohjelmointikurssin Java-kielisistä harjoitustöistä. Näistä varsinaisen analysoinnin kohteena on 60 koodin sisällä sijaitsevaa laajaa kokonaisuutta, jota kutsutaan Java-ohjelmoinnissa metodiksi.

Tutkimuksen teoreettisena perustana toimii systeemis-funktionaalinen näkemys teksteistä ja tekstilajeista. Analyysin jaottelun perusteena on systeemis-funktionaalisen teorian piiriin kuuluvat M. A. K. Hallidayn luomat metafunktiot, joita ovat ideationaalinen, interpersoonainen ja tekstuaalinen. Poimin aineistosta esimerkkejä maailman kuvaamiseen, tekijyyteen ja koheesiokeinoihin liittyvistä piirteistä ja analysoin niiden vaikutusta tekstilajin muodostumiseen.

Tutkimuksen tuloksena voidaan osoittaa, että ainakin tämän aineiston osalta ohjelmointikoodin kommentit muodostavat melko yhtenäisen tekstilajin, sillä niillä on yhdistäviä ja vain tälle tekstilajille tyypillisiä piirteitä. Kommenttien tekstilajille ominaista on se, että se on riippuvainen kommentoimastaan koodista, mikä on määritelmä kommenttien ja koodin suhteelle. Koodi määrää kommenttien esiintymisjärjestyksen ja koodin sisältö vaikuttaa siihen, millainen sitä kommentoiva koodi on.

Interpersoonaisen metafunktion osalta kommenteista nousee esiin passiivi yleisimpänä predikaattiverbin persoonamuotona, ja passiivista on mahdollista osoittaa erilaisia implisiittisiä tekijöitä. Ideationaalisen metafunktion osalta kommenteista nousee esiin tapoja rakentaa maailmaa joko pelin tai koodin näkökulmasta, sekä se, että kommenttien diskurssiyhteisöä rakennetaan erityissanaston avulla. Tekstuaalisen metafunktion osalta keskeisiksi nousevat ne melko vähäiset koheesiokeinot, joiden avulla eri kommenttien välille rakennetaan sidosteisuutta.

Avainsanat: tekstintutkimus, tekstilaji, kommentti, ohjelmointi, systeemis-funktionaalinen teoria

Sisällysluettelo

1. Johdanto

Tämän työn tutkimuskohteena ovat Java-kielisen ohjelmointikoodin kommentit ja se, millaisen tekstilajin ne luovat yhdessä varsinaisen koodin kanssa. Tutkimus yhdistää lingvistisiä tekstintutkimuksen käytänteitä tietoteknisessä yhteydessä esiintyviin luonnollisella kielellä kirjoitettuihin teksteihin.

Tietokonekielen ja luonnollisen kielen yhdistelmä ei ole saanut huomiota suomen kielen tutkimuksessa, vaikka 2010-luvulla tekstit ja tietotekniikka kulkevat käsikkäin lähes jokaisen ihmisen elämässä. Suuret lukijajoukot lukevat sähköisessä muodossa tekstejä sanomalehdistä ja artikkeleista aina romaaneihin ja resepteihin saakka. Kaiken tietotekniikan taustalla toimii aina joku tietokoneelle tarkoitettu koodi, jota kuka tahansa ei osaa lukea tai voi ymmärtää, vaikka koodin lopputulosta päivittäisessä elämässään hyödyntäisikin. Tätä koodia tietokoneet eivät tuota itsenäisesti, vaan niidenkin taustalla on ihminen, joka osaa ”puhua tietokonetta”, siis kirjoittaa koodia, jota ymmärtävät sekä koodaaja että tietokone. Tällaiseen ohjelmointikoodiin liittyy lähes aina myös luonnollista kieltä, joka on tarkoitettu ainoastaan koodia lukevan ihmisen silmille, ja tätä luonnollisen kielen ujuttamista koodin sekaan kutsutaan ohjelmoinnissa kommentoinniksi.

Ohjelmointikielellä tarkoitetaan sovittuja komentoja, joilla määrätään tietokoneen ohjelman toimintaa, ja koodiksi kutsutaan sitä tietyllä ohjelmointikielellä kirjoitettua tekstiä, joka aiheuttaa tietokoneessa erilaisia toimintoja (Wikla 2003: 3–4). Ohjelmointikoodissa kommenttien funktiona on kertoa, mitä itse koodissa tapahtuu. Ne ovat luonnollisella kielellä kirjoitettuja, itse koodista erotettuja tekstinpätkiä, joiden ytimekkäänä tehtävänä on selventää ohjelman toimintaa. Kommentit näyttävät aineistossa käytännössä seuraavanlaiselta:

```
//Tarkistetaan suunta, johon kuljetaan
if (suunta == 'u'){
```

Kiinnostuin teknisen ohjelmointikoodin ja siihen liittyvien luonnolliskielisten kommenttien yhdistelmästä Java-ohjelmoinnin peruskurssilla, jolla hämmästyin sitä, miten tärkeää oikeanlainen kommentointi on onnistuneen ohjelman saavuttamiseksi. Pelkkä taito kirjoittaa toimivaa teknistä koodia ei yksinään riitä, vaan ohjelman kirjoittajan pitää osata myös kommentoida oleellisia asioita niin, että ohjelmointikoodin lukijat ymmärtävät, mistä on kyse. Ohjelmointikoodin kommenttien kirjoittamiseen on annettu ohjeita erilaisissa ohjelmointioppaissa, mutta varsinaista määritelmää

ohjelmointikoodin kommenttien tekstimuodolle en ole onnistunut löytämään. Sen löytäminen on yksi tutkimukseni tavoitteista.

Kommentit ovat kiinteässä yhteydessä itse koodiin, ja suhde näiden kahden välillä on myös varmasti omanlaisensa tapaus kaikkien kommenttien ja kommentoitavien tekstien kentällä. Luonnollisen kielen ja ohjelmointikoodin kietoutuminen yhteen onnistuneeksi ohjelmointikoodiksi on ainutlaatuinen tapahtuma, sillä siinä yhdistyy tietokoneen käännettäväksi tarkoitettu, tarkkaan jokaista puolipistettä myöten määritelty koodikieli ja kommentit, joita ohjelmoija kirjoittaa itseään ja muita koodia tarkastelevia ihmisiä varten. Toinen tavoitteeni onkin osoittaa, millainen on suhde ohjelmointikoodin ja siihen liittyvien kommenttien välillä.

Tutkimuksen lähtökohtana on olettamus siitä, että ohjelmointikoodin kommentit muodostavat tekstilajin. Perusteena tekstilajisuudelle pidän sitä, että kommenteilla on selkeä ennalta määrätty tehtävä (Wikla 2003: 15–16) sekä sitä, että ne esiintyvät aina tietyssä kontekstissa tietyn muotoisina. Kommenttien tutkiminen tekstilajina on luontevaa ja perusteltua sikälikin, että tekstilaji eli genre on nykyisessä tietoyhteiskunnassa ihmisen ajattelua ohjaava ja määrittelevä tekijä, ja genretietoisuus eli lajien tunnistaminen on edellytys tässä yhteiskunnassa elämiselle (Heikkinen & Voutilainen 2012: 17–18). Erityisenä kannustimena juuri ohjelmointikoodin kommenttien tutkimiselle pidän Heikkisen ja Voutilaisen (2012: 19–20) toteamusta siitä, että tekstilajitutkimuksen olisi perusteltua ylittää tieteenalojen rajat. Genreä on tutkittu ylivoimaisesti eniten kirjallisuudentutkimuksen ja kielitieteen saralla (Heikkinen & Voutilainen 2012: 19–20), ja vaikka tutkimukseni onkin kielitieteellinen, rikkoo se aiheeltaan perinteiset tieteenalarajat, kun kommenttien olemassaolosta vastaa ohjelmointikoodi.

Kommenttia tekstilajina ei ole määritelty tarkkaan, sillä eri yhteydessä kommenttinimityksellä voidaan viitata täysin erilaisiin teksteihin. Kommentiksi kutsutaan kaikkea uutisen kylkeen liitetystä toimittajan kirjoittamasta lyhyehköstä tekstistä niihin vihaisiin yksisanaisiin asiattomuuksiin, joita joidenkin verkkolehtien artikkeleiden perästä voi löytää. Tahdon nostaa tutkimuksen kohteeksi yhdenlaisen kommentin, jotta kommentti tekstilajina saisi lisää valoa ja sen yhdistäviä tekijöitä olisi mahdollista avata lisää. Ei varmasti ole mahdollista nimetä yhtä isoa kommentin tekstilajia, niin paljon nämä kommenteiksi kutsutut tekstit toisistaan poikkeavat, mutta voi olla, että erilaisilla kommenteilla on joitakin yhteisiä piirteitä, joiden takia niitä juuri kommenteiksi kutsutaan.

Teoreettisena lähtökohtana tutkimukselle toimii systeemis-funktionaalinen tekstiteoria, ja erittelen kommenttien piirteitä teoriaan tiiviisti liittyvien metafunktioiden avulla. Analysoin, miten koodin tekijyys heijastuu kommentteihin ja miten koodiin suhtaudutaan, mikä kuuluu interpersoonaisen metafunktion piiriin. Hahmottelen, millaista kuvaa kommentit rakentavat ohjelmointi- ja

pelimaailmasta ja miten kommenttien diskurssiyhteisö osoitetaan, mikä kytkeytyy ideationaaliseen metafunktioon. Näiden lisäksi pohdin koheesiokeinojen kautta, millä tavalla kommentit ovat, mikä liittyy tiiviisti tekstuaaliseen metafunktioon. (Ks. Halliday & Hasan 1985: 15–23.)

Tutkimuskysymykseni ovat seuraavat:

- Millaisen tekstilajin ohjelmointikoodin kommentit muodostavat?
- Millaisessa suhteessa kommentit ovat niiden kommentoimaan koodiin?
- Millaiset tekijyyteen, maailman kuvaamiseen ja koheesiokeinoihin liittyvät piirteet määrittävät kommentteja?

Ensimmäisenä teen lyhyen teoreettisen katsauksen ohjelmointiin ja tutkimuksen tietoteknisiin ulottuvuuksiin (luku 2.1), minkä jälkeen kuvaan työni taustalla vallitsevaa tekstintutkimuksen teoriaa (luku 2.2). Luvussa 3 kuvailen aineistoani ja sen käsittelyn ja analysoinnin menetelmiä. Luvun 4 tehtävänä on johdattaa ohjelmointikoodin kommenttien maailmaan ja rakentaa kuva siitä, millaisia kommentit voivat olla ja mikä niiden tehtävä yleisesti on. Luvussa 5 analysoin kommentteja ja jäsennän analyysin Hallidayn metafunktioiden avulla.

2 Teoriaa tekstintutkimuksen ja ohjelmoinnin taustalla

Tutkimukseni yhdistää toisiinsa tekstintutkimusta ja tietojenkäsittelytiedettä. Varsinaisesta teoreettisesta taustasta vastaa tekstintutkimus, mutta otan huomioon myös oleelliset osat ohjelmoinnin tekniikasta.

2.1 Pieni katsaus ohjelmointiin

Ohjelmoinnin perusajatusten ja Java-ohjelmointikielen lyhyt esittely on tarpeen, jotta voin analyysiosiossa nimetä ja perustella ohjelmointikieleen kiinteästi liittyviä käsitteitä ja piirteitä. Tavoitteenani ei ole luoda kattavaa ja aukotonta kokonaiskuvaa Javasta vaan esitellä se niin, että lingvistisen tutkimukseni käsittelemään aihepiiriin pääsee sisälle.

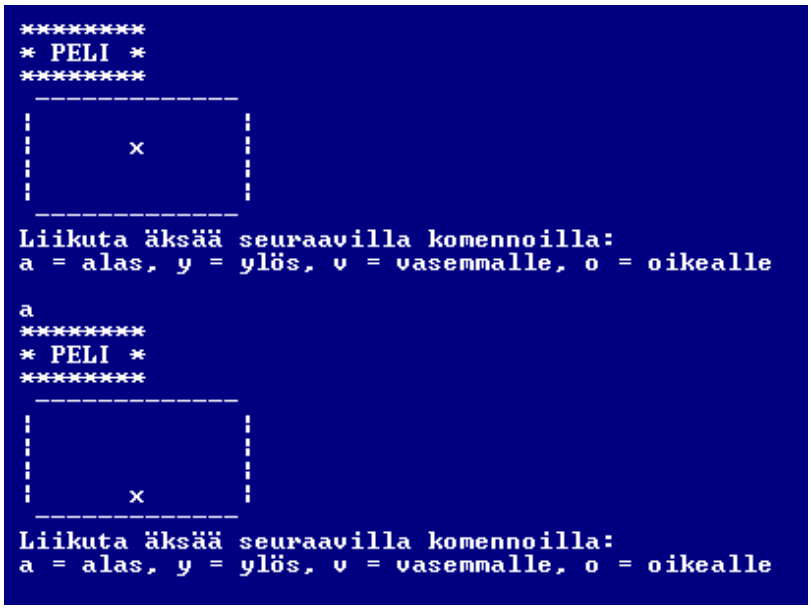
2.1.1 Java-kieli lyhyesti

Yksinkertaistetusti ohjelmoinnin perusajatus on se, että ihmisen pitää kirjoittaa koodia, jota tietokone ymmärtää ja voi ymmärrettyään tehdä ihmisen määräämiä toimintoja. Tietokone ymmärtää konekieltä ja ohjelmoija osaa kirjoittaa lausekieltä, ja jotta ohjelmoijan kirjoittamasta lausekielestä saadaan konekieltä, jota tietokone osaa käyttää, on välissä kääntäjä, joka tulkkaa lausekielen konekieleksi. Suurin osa ohjelmointikielistä on nykyään lausekieliä, ja tässä työssä keskityn ainoastaan Java-kieleen, joka on Sun Microsystems Inc. -yhtiön vuonna 1996 julkaisema yleiskäyttöinen olio-ohjelmointikieli. (Wikla 2003: 3–4, 9.)

Ohjelmointi on abstraktia ajattelua, jonka apuvälineenä käytetään algoritmeja eli vaiheittaisia kuvauksia jonkin tehtävän suorittamisesta. Javassa käytetään muuttujia ja operaatioita rakentamaan toivottua toimintaa algoritmeihin. (Laurikkala 2014b: 2–8.) Java-kielellä tehdään tavallisia ohjelmia, minkä lisäksi myös www-sivuille sijoitettavia osia on mahdollista rakentaa (Wikla 2003: 10). Ohjelmointikoodit, joita tämä tutkimus käsittelee, ovat perinteisiä ohjelmia. Kielenä Java on suhteellisen pieni, mikä tarkoittaa, että valmiita osia on melko vähän, mutta se on kuitenkin hyvin ilmaisuvoimainen (Wikla 2003: 9).

Aineiston ohjelmat, joihin liittyviä kommentteja analysoin, ovat ainoastaan komentoikkunassa tapahtuvia alkeellisia merkkikuvioita. Kuvassa 1 on kuvitteellinen esimerkki siitä, miltä komentoikkunassa käytettävä peli voisi näyttää. Tämä ei siis ole aineiston koodien tuottama peli, mutta sen visuaalinen olemus on samankaltainen. Komennot annetaan komentoikkunalle, joka tulostaa pelinäkymän joka kerta uudelleen, kun käyttäjän antama komento on suoritettu. Kuvassa 1 komentoikkunaan tulostuu ensin pelin nimi (PELI), viivoin ympäröity pelikenttä, jonka keskellä x sijaitsee

sekä ohjeet pelaamiseen. Pelaaja antaa pelille komennon ”a”, minkä jälkeen pelin nimi, kenttä ja ohjeet tulostuvat uudelleen, mutta x on siirtynyt alas. Aineiston pelit ovat siis todella alkeistasoista Java-koodia.



Kuva 1

2.1.2 Kommentit ohjelmointikoodissa

Yksinkertaisuudessaan kommentit ovat luonnollisella kielellä kirjoitettuja ohjelman toimintaa selventäviä lauseita ohjelmointikoodin seassa (Laurikkala 2014: 12). Niiden tehtävänä on selittää ohjelman rakennetta ja toimintaa ja auttaa ohjelmakoodin lukijaa ymmärtämään koodia, eli ne eivät vaikuta millään tavalla ohjelman toimintaan. Ainoastaan ihmiset, jotka lukevat ohjelmatekstiä, tarvitsevat kommentteja ja lukevat niitä, joten esimerkiksi pelin pelaaja ei näe eikä tarvitse kommentteja laisinkaan. Kommentoinnin keskeisyys johtuu esimerkiksi siitä, että suuria ohjelmia ylläpidetään, mitä helpottaa hyvä kommentointi – usein se on jopa edellytys. (Wikla 2003: 15–16, 110.) Tällöin myös sellainen henkilö, joka ei ole itse kirjoittanut koodia, hahmottaa toiminnan helpommin ja nopeammin kuin ilman luonnolliskielisiä selityksiä. Syyt kommentoinnille ovat lähtöisin inhimillisestä tavasta käyttää kieltä konstruoimaan kokemusmaailmaamme (Shore 2012a: 146, ks. myös Halliday 1978: 112), sillä ohjelmointikieltä käyttävien ihmisten on helpompaa ja nopeampaa ymmärtää koodia, kun se on kuvattu luonnollisella kielellä koodatun tekstin lisäksi (Laurikkala 2014a: 12).

Martinin (2009: 55) näkemys kommentoinnin tarpeellisuudesta on toisenlainen, sillä hänen mukaansa kommentit peittelevät huonoa koodia. Pidän Wiklan näkemystä kommentoinnista perus-

tellumpana, sillä ohjelmointikielen ja luonnollisen kielen lukeminen eivät nähdäkseni ole verrattavissa toisiinsa siinä mielessä, että ne olisivat kokeneellekaan koodaajalle yhtä vaivattomia lukea. Luonnollisen kielen lukemaan ja kirjoittamaan oppiminen tapahtuu kuitenkin aina ensin[1].

Varautuneesta suhtautumisestaan huolimatta Martin (2009: 55–59) esittelee erilaisia kommentoinnin syitä, jotka toimivat erinomaisena pohjatietona tälle tutkimukselle. Kommentti voi olla informatiivinen, jolloin se sisältää perustietoa esimerkiksi jonkin metodin paluuarvosta, tai kommentti voi olla selitys jollekin koodissa tehdylle ratkaisulle, joka ei ole perustapaus. Kommentin tehtävänä voi olla selventää jonkin koodinpätkän merkitystä tai lopputulosta, esimerkiksi paluuarvojen arvot voi kirjoittaa auki, jolloin koko koodia ei tarvitse seurata niiden tulkitsemiseksi. Kommentti voi myös alleviivata jonkin osan tärkeyttä. (Martin 2009: 55–59.)

Konkreettisella asteella kommentin sisältö liittyy yleensä koodissa seuraavaan riviin tai seuraaviin riveihin, mikä käy analyysiluvusta 4.2 ilmi. Kuten ohjelmointikoodissa yleensä, koodin sisennyksillä eli rivin alkuun sijoitettavilla välilyönneillä on oma tehtävänsä kertoa, mihin kommentti loogisesti liittyy: yhteenkuuluva kommentti ja koodi alkavat vasemman reunan tasaukselta samasta kohdasta (Laurikkala 2014a: 12–14). Kommenttien esittäminen tapahtuu Javassa kolmella tavalla. Tavallisin tapa erottaa kommentti koodista on kaksi vinoviivaa (//) rivin alussa, jolloin koodin kääntäjä tulkitsee loppurivin kommentiksi eli luonnollisella kielellä kirjoitetuksi tekstiksi, joka ei ohjaa ohjelman toimintaa millään tavalla. Esimerkki 1 näyttää, miltä tällainen kommentti voisi koodissa näyttää.

```
(1)  // Tavallisesti kommentti erotetaan koodista kahdella vinoviivalla rivin alussa.
```

On myös mahdollista erottaa riviä pidempi pätkä kommentiksi, ja esimerkiksi ohjelman tai erilaisten laajempien kokonaisuuksien alussa tällaisen kommentin käyttö on järkevää. Tällöin kommentti aloitetaan yhdellä vinoviivalla ja tähdellä (/*), ja kommentin päättyminen merkitään tähdellä ja vinoviivalla (*/). (Laurikkala 2014a: 12–15; Laurikkala 2014c: 5.) Esimerkissä 2 esitellään tällaisen kommentin teknistä käyttöä.

```
(2)  /*
     * Tällaisessa pidemmässä kommentissa voi käyttää vinoviivaa ja tähteä aloittamaan
     * kommentin ja päättämään tähteä ja vinoviivaa. Vasemmassa reunassa sijaitsevat tähdet
     * ovat teknisessä mielessä turhia, mutta ohjelmointikurssilla ja ohjelmointioppaissa
     * ne ovat visuaalinen käytänne.
     */
```

[1] Ks. lapsen lukemaan oppimisesta esim. Kiiveri 2006: 57–75.

Myös dokumenttikommentti, joka alkaa ilmauksella ”/**” ja loppuu ”*/”, on mahdollinen (Wikla 2003: 110), mutta tällaisia tapauksia ei aineistossani esiinny.

Kommentit on pakko erottaa koodista esittelemilläni tavoilla, joten kommenttien tunnistaminen ja niiden poimiminen aineistosta ei vaadi tutkijan omaa tulkintaa. Tällöin ongelmaksi ei muodostu kehämäisyys, josta systeemis-funktionaalista genreanalyysia on ajoittain kritisoitu. Kehämäisyyden ongelman aiheuttaa tutkijan intuition varassa rajattu aineisto, joka jo itsessään voi rajata osan lajiin todellisuudessa kuuluvista teksteistä pois, ja tutkijan käyttämä malli taas elää sitä mukaa, kun hän löytää aineistosta poikkeavia piirteitä. (Heikkinen & Voutilainen 2012: 26.) Kommenttien poimiminen aineistosta ei vaadi tutkijan intuitiota, enkä siis voi tahattomastikaan jättää jotakin kommenttien piiriin kuuluvaa pois, sillä kommenttien esittäminen ohjelmointikoodissa ei ole millään muulla tavalla mahdollista.

2.1.3 Kielitiede ja tietojenkäsittelytiede yhdessä

Kieliteknologiaksi kutsutaan keinoja, joilla tietokone saadaan ymmärtämään, käsittelemään ja tuottamaan ihmisten käyttämää luonnollista kieltä. Suomen kielen asema kieliteknologian kentällä on melko heikko verrattuna niin isoihin maailmankieliin kuin naapurimaidemme kieliin, sillä kieliteknologian tutkimukseen ja kehittämiseen on keskitytty varsin vähän. Tietoteknisten sovellusten ensisijainen kieli on usein englanti, eikä suomea huomioida välttämättä ollenkaan, mikä on nykymaailmassa uhka suomen kielen säilymiselle. (Hakulinen, Kalliokoski, Kankaanpää, Kanner, Koskenniemi, Laitinen, Maamies & Nuolijärvi 2009: 139, 144.) Tämän tutkimuksen alaa ei varsinaisesti ole kieliteknologia, sillä kommentit ovat ihmisten ihmisille kirjoittamia, eikä tietokoneen ole tarkoitus niitä ymmärtää. Tietotekniikan ja suomen kielen tutkiminen yhdessä toisiaan tukevina osasina on kuitenkin merkityksellistä suomen kielen kannalta, sillä se osoittaa, että tietotekniikan saralla on vielä osa-alueita, joissa ei käytetä ainoastaan englantia.

Myös luonnollista kieltä ja ohjelmointia on tutkittu jonkin verran. Tietojenkäsittelytieteen alalla luonnollisen kielen ja ohjelmoinnin piirteitä on hyödyntänyt Donald E. Knuth. Hän on kehittänyt ohjelmointikielen, jota kutsuu ”lukutaitoiseksi ohjelmoinniksi” (englanniksi ”literate programming”). Ideana Knuthin WEB-nimisessä ohjelmointikielessä on yhdistää dokumentaatio, jonka alalle kommentointikin kuuluu, ja ohjelmointi samaan kieleen niin, että ensisijaisessa asemassa ohjelmointikielen lukijoina ovatkin ihmiset tietokoneen sijaan. (Knuth 1984: 97.) Kieli on esitelty 1980-luvulla, eikä se ainakaan toistaiseksi ole ottanut niin isoa roolia ohjelmointimaailmassa, että sitä voisi esimerkiksi Tampereen yliopistossa opiskella (Tampereen yliopiston opinto-opas 2012–2015). On kuitenkin kiinnostava ajatus, että ohjelmat itsessään koostuisivat ihmisille helpommin luettavista kielistä, jolloin kommentoinninkin merkitys mahdollisesti vähenisi. Ainakaan WEB-

kielessä kommentoinnista ei ole kuitenkaan luovuttu (Knuth 1984: 99), joten kommentointi lienee elinvoimainen ilmiö ohjelmoinnissa vielä pitkään.

WEB-kielen jälkeen on kehitetty myös muita ohjelmointikieliä, joissa luonnollista kieltä hyödynnetään osana ohjelmointia. Esimerkiksi Attempto Controlled English ja NaturalJava ovat kieliä, joissa ohjelmoidaan luonnollista kieltä käyttäen, mutta niissä sanasto on rajoitettua. Ongelma luonnollisen kielen ja ohjelmoinnin vapaassa yhdistämisessä, jossa esimerkiksi kokonaiset luonnollisella kielellä kirjoitetut lauseet käännettäisiin koodiksi, on luonnollisen kielen monimutkaisuus. (Lincoln & Veres 2013: 212.) System-English eli sEnglish on myös yksi ohjelmointisysteemi, joka on ihmiselle luettava ja koneelle toteutuskelpoinen (Lincoln & Veres 2013: 213).

2.2 Tekstintutkimus

Tässä luvussa esittelen tähän tutkimukseen kiinteästi liittyvät käsitteet ja tutkimuksen taustalla vaikuttavan teoriapohjan. Aloitan määrittelemällä moniin erilaisiin asioihin viittaavan tekstin käsitteen, kuten se tutkimuksessani esiintyy (ks. Heikkinen 2012: 59–60), minkä jälkeen esittelen tekstilajin käsitteen ja sen tutkimusta kielitieteen piirissä. Systeemis-funktionaalinen kielioppi on tärkein teoria tutkimuksen taustalla, joten esittelen sen historian ja keskeiset käsitteet eli tilannekontekstin ja metafunktiot (Halliday & Hasan 1985: 15–17). Esittelen lyhyesti myös rakenteen näkökulman tekstilajiin sekä diskurssiyhteisön käsitteen, sillä molemmat nousevat analyysiluvussa kommenttien kannalta aiheellisiksi.

2.2.1 Teksti

Koska tarkastelen tässä tutkimuksessa kommenttien tekstilajia, nousee varsin keskeiseen rooliin tekstin käsite. Käsitteen pohtiminen on aiheellista siltä kannalta, että ohjelmointikoodin kommenttien muodostamat tekstit eivät ole niin selkeitä kuin esimerkiksi uutistekstit, jotka tyypillisesti alkavat otsikosta ja joissa asiat esitetään yleensä järjestyksessä tärkeimmästä vähiten tärkeään (Mäntynen 2006: 59–60). Uutistekstin rajaaminen ja esittäminen yhtenä kappaleena uutistekstejä on selkeää, kun taas raja kommenttitekstien välillä on tulkittavissa kahdella tavalla: joko koko ohjelmointikoodin kommentit voi tulkita yhdeksi tekstiksi, tai sitten jokaisen kommentin voi tulkita omaksi tekstikseen. Tekstin käsitettä avaamalla pyrin osoittamaan sen, miten tekstin erilaisia rajauksia voidaan perustella, ja millä tavalla kommentteja tässä tutkimuksessa lähestytään.

Siihen nähden, kuinka tavallinen ja jokapäiväinen käsite teksti on, on sen määrittely hankalaa. Yleisimmällä tasolla tekstiksi voidaan kutsua mitä tahansa kulttuurituotetta kuten rakennusta tai musiikkikappaletta, yleiskielessä teksteiksi nimitetään esimerkiksi kaikkea kirjoitettua materiaalia,

ja lingvistiikassa ja fonetiikassa teksti voidaan määritellä käsitteeksi, jolla viitataan kielen katkelmaan (Heikkinen 2012: 59–60). Teksti-käsitteellä voi viitata rajattuun kokonaisuuteen sekä rajaamattomaan tekstin määrään (Shore ja Mäntynen 2006: 9). Nykyisillä tutkimusaloilla, kuten tekstilingvistiikassa ja tekstintutkimuksessa, huomio kohdistetaankin kielenkäytön kokonaisuuksiin lause- tai sanatason sijaan (Heikkinen 2012: 63). Teksti-sanan etymologia ohjaa näkemään tekstit tiiviinä ja sidosteisina kokonaisuuksina, sillä latinan texere-verbi on tarkoittanut konkreettisesti kudottua (Heikkinen 2012: 59), joten jotta jotakin kokonaisuutta voidaan kutsua tekstiksi, on sen oltava tarpeeksi tiivis ja sidosteinen. Tämä sidosteisuuden vaade rajoittaa kommenttien kokonaisuuden tekstiksi nimittämistä, sillä kommenttien järjestys ja alisteisuus toisiinsa nähden on paljon riippuvaista ainoastaan koodin jäsentelystä ja suhteista. Kommentit eivät myöskään ole sidosteisia ainakaan niin paljon, kuin tekstit yleensä totutusti ovat.

Kielitieteen piirissä tekstin määrittely on ollut alkujaan keskeinen ongelma (Kalliokoski 1996: 27–28). Tekstilingvistiikassa vallinneet vaiheet voi kiteyttää kolmeen: tekstikielioppien vaiheeseen (1960-luvun loppu ja 1970-luvun alku), tekstuaalisuuden rakenteisuuden sekä lingvistisen ja sosiaalisen aspektin vaiheeseen (1970-luvun loppu ja 1980-luvun alku) ja tekstin tuottamisen ja vastaanottamisen sosiaalisten ja kognitiivisten prosessien vaiheeseen, jossa on keskitytty yleisen teksti- ja diskurssitieteen kehittämiseen (Heikkinen 2012: 62).

Systeemis-funktionaalinen teoria on yksi tämän hetken tekstintutkimuksen hallitsevimmista teorioista, ja siinä teksti nähdään sosiaalisen kanssakäymisen lingvistisenä muotona. Tekstillä tarkoitetaan kaikkia kielen ilmentymiä, joita joku kielenkäyttäjä ymmärtää, ja ne ovat prosesseja, joissa merkitykset kehittyvät. (Halliday 1978: 122, Halliday 1985: 3, Halliday & Hasan 1985: 10, Heikkinen 2012: 65.) Halliday siis määrittelee tekstin yksinkertaisesti funktionaaliseksi kieleksi, mikä tarkoittaa, että kielellä on joku tehtävä jossain kontekstissa. Tällaisen määritelmän mukaan kaikki tilanteessa jonkin tarkoituksen vuoksi esiintyvä kirjoitettu tai puhuttu kieli on tekstiä. (Halliday & Hasan 1985: 10.)

Systeemis-funktionaalisen teorian tekstinmäärittelyssä keskeistä onkin se, että tekstit ja niiden merkitykset esiintyvät kontekstissaan, josta niitä ei milloinkaan saisi irrottaa: teksti ja konteksti ovat saman prosessin eri ulottuvuuksia (Halliday & Hasan 1985: 5). Tekstit itsessään ovat ennen kaikkea semanttisia yksiköitä, sillä sanojen ja rakenteiden avulla ilmaistaan ensisijaisesti merkityksiä. Tekstit ovat yhtä aikaa tuotteita ja prosesseja, sillä ne ovat konkreettisia rakenteita, joita voi esimerkiksi nauhoittaa ja opiskella, mutta yhtälailla ne liikkuvat jatkuvassa semanttisten valintojen prosessissa. (Halliday & Hasan 1985: 10.)

Hallidayn tekstille antama määritelmä (ks. Halliday & Hasan 1985: 5–12) ohjaa epäilemättä näkemään, että kommentit ovat tekstejä. Kommenteilla on selkeä funktio koodin luonnolliskielisinä

selittäjinä, ne esiintyvät aina samassa kontekstissa koodin yhteydessä ja niiden avulla välitetään koodiin kätkettyjä merkityksiä. Siihen, ovatko kommentit yksittäisinä kappaleina vai kokonaisuuksina tekstejä, Hallidayn määritelmä ei anna suoraa vastausta, ja pohdin vastausta siihen tekstuaalisen metafunktion yhteydessä. Joka tapauksessa systeemis-funktionaalinen teoria ohjaa näkemään ohjelmointikoodin kommentit teksteinä.

Toisten näkemysten mukaan tekstuaalisuuden kriteereistä merkittävimpinä pidetään koheesiota ja koherenssia (Heikkinen 2012: 64). Ollakseen teksti tekstin pitää olla sidosteista, ja tätä sidosteisuutta kutsutaan koheesioksi. Koherenssilla viitataan yleisesti siihen, miten lausetta laajemmat tekstijaksot kytkeytyvät toisiinsa (Heikkinen 2012: 64). Koherenssia on tekstin funktionaalinen yhtenäisyys, millä tarkoitetaan kielenkäyttäjien yhteistä tietoa, yhteisiä päätelmiä ja oletuksia (Heikkinen 2012: 64). Sidosteisuus voi syntyä kielellisten valintojen ja saman semanttisen kentän sanavalintojen avulla, toisin sanoen koheesiokeinot voi jakaa kieliopillisiin ja leksikaalis-semanttisiin (Heikkinen 2012: 63). Hyödynnän tutkimuksessani molempia koheesiokeinoja, sillä molemmat ovat keskeisessä asemassa tekstejä tulkittaessa; en näe, että kumpaakaan voisi jättää pois. Koheesiokeinoja arvioin luvussa 5.3.

Ohjelmointikoodin kommenttien ongelmallisuus teksteinä nousee esiin juuri koheesion ja koherenssin käsitteiden avulla. Kommentit osoittavat koheesiota semanttisella kentällä sanaston puolesta, mutta kieliopillista sidosteisuutta esimerkiksi pronominien ja konjunktioiden avulla esiintyy harvemmin kuin perinteisissä teksteissä, jotka on tarkoitettu luettaviksi sellaisinaan (ks. luku 5.3). Kommentteja ei ole tarkoitus lukea omana kokonaisuutenaan, vaan ne esiintyvät aina koodin yhteydessä, eikä pelkkiä kommentteja lukemalla ole välttämättä mahdollista saada kokonaiskuvaa, jonka kommentit yhdessä koodin kanssa muodostavat.

Pidän tutkimuksessa mukana sekä kokonaiset koodit että yksittäiset kommentit, sillä koherenssiin on mahdollista päästä käsiksi ainoastaan kokonaisuuksien avulla. Toisaalta jokainen kommenttikin muodostaa oman pienemmän tekstinsä, sillä funktionaalisesti kommentit ovat hyvin homogeeninen joukko, koska niiden tarkoitukset on määritelty ennalta selvästi. Myös systeemis-funktionaalisen teorian jako konkreettisiin teoksiin ja abstrakteihin teksteihin (Heikkinen 2012: 65) tukee kokonaisuuksien huomioimista yksittäisten kommenttien lisäksi, sillä teokseksi voisi ehkä nimetä kokonaisen ohjelmointikoodin, muttei yksittäistä kommenttia. Teosta voi myös tutkia tekijän intentiona (Heikkinen 2012: 65), joskaan tekijyys ei nouse kommenttien kaltaisessa tekstilajissa keskiöön.

2.2.2 Tekstilaji

Tekstilajit ovat läsnä jokaisen ihmisen elämässä arkisista tilanteista lähtien, sillä millaisessa tahansa tilanteessa tekstistä puhutaan, se usein luokitellaan johonkin lajiin kuuluvaksi (Shore & Mäntynen 2006: 9). Vaikka tekstilajeja tunnistetaan arjessa, voi tieteellinen luokittelu poiketa tällaisesta arkiluokittelusta (Mäntynen & Shore 2008: 25). Tekstilajien tieteellinenkin tutkiminen on perusteltua ja tärkeää, sillä genret ovat keskeisiä kaikessa inhimillisessä toiminnassa auttaessaan ihmisiä hahmottamaan maailmaa ja luodessaan sosiaalisen käyttäytymisen toimintamalleja (Heikkinen & Voutilainen 2012: 17). Erityiseen asemaan tekstilajit ovat nousseet nyt 2000-luvulla, sillä yhteiskuntamme on entistä tieto- ja tekstipainotteisempi. Ihmiset viettävät yhä enemmän aikaa erilaisten tekstien äärellä niin työympäristöissään kuin vapaa-ajallaankin. (Heikkinen & Voutilainen 2012: 17–18.) Tekstilajien tunnistaminen on esimerkiksi nostettu keskeiseksi osaksi peruskoulun ja lukion äidinkielen opetusta (Shore & Mäntynen 2006: 12), mikä osoittaa, että tekstilajien tunnistaminen ja nimeäminen on keskeisessä osassa myös muualla kuin ainoastaan kielen- ja kirjallisuudentutkimuksen piirissä.

Tekstilajin tutkimus kielitieteessä on verrattain nuorta sen tarjoamiin mahdollisuuksiin nähden, sillä sitä on aktiivisesti tutkittu vasta parin vuosikymmenen ajan (Nieminen 2010: 27–28). Tekstilajia on sen tutkimuksen aikana pyritty määrittelemään todella monella tavalla (Saukkonen 2001: 142), eikä tekstilajin tutkimusalan käsitteistöstä ole vakiintunutta käytäntöä (Nieminen 2010: 27–28). Jo ainoastaan kielitieteellisissä tutkimuksissa tekstilajille on annettu suuri joukko toisistaan poikkeavia määritelmiä, joiden yhdistävänä piirteenä voidaan pitää sitä, että genret ohjaavat kaikenlaisten tekstien ja kulttuuristen tuotteiden luomista, ja toisaalta tekstit ja tuotteet muuttavat jatkuvasti genrejä (Heikkinen & Voutilainen 2012: 22–23).

Pioneerina modernille genretutkimukselle pidetään Mihail Bahtinia, jonka keskeinen näkemys on, että genret ohjaavat kaikkea kielellistä toimintaa niin kirjallisuuden kuin sosiaalisen kanssakäymisenkin saralla. Genret ovat vakaita eri asteisesti ja uudistuvat jatkuvasti. Elinvoimainen on myös Bahtinin näkemys siitä, että tekstit ovat aina kytköksissä toisiin teksteihin. (Heikkinen & Voutilainen 2012: 21.) Bahtinin lisäksi genren alkutaipaleen keskeisiä tutkijoita ovat olleet esimerkiksi J. R. Firth ja M. A. K. Halliday (Shore & Mäntynen 2006: 20), ja heidän työtään genren saralla käsittelen seuraavassa luvussa, jonka aiheena on syvemmin systeemis-funktionaalinen teoria.

Kun tutkitaan tekstilajeja, on tekstien luokittelun perusteita pohdittava jo aineistonrajausvaiheessa. Jos tutkimuskohteena on tietty tekstilaji, tehdään aineistoa rajattaessa oletus siitä, mitkä tekstit lajin piiriin lasketaan. (Shore & Mäntynen 2006: 11.) Kuten olen aiemmin todennut, ohjelmointikoodin kommenttien osalta aineiston rajaamisessa ei esiinny haparointia, sillä lasken tekstila-

jiin kuuluviksi kaikki kommentteina koodiin kirjoitetut tekstit, jotka voi erottaa //- tai /*-merkkiyhdistelmien avulla. Tekstilajin tunnistamisessa ja määrittelyssä on pohjimmiltaan kyse siitä, että tekstien eri esiintymistä voi osoittaa tarpeeksi samankaltaisuuksia (Mäntynen & Shore 2008: 25). Toisaalta tekstilajien kannalta huomioon otetaan tekstin tilanteinen, sosiaalinen, historiallinen ja kulttuurinen konteksti (Shore & Mäntynen 2006: 41). Jotta tekstin voi sanoa kuuluvan genreensä, on aina otettava huomioon sen konteksti (Halliday & Hasan 1985: 68). Genret ovatkin esimerkiksi niiden lähelle sijoittuvaan diskurssin käsitteeseen verrattuna tilannesidonnaisempia (Pietikäinen & Mäntynen 2009: 81).

Genrejen normit ovat useimmiten kirjoittamattomia, jolloin sosiokulttuurinen ja tilanteinen konteksti nousevat keskeisiksi (Pietikäinen & Mäntynen 2009: 83). Tällaisia normeja on myös ohjelmointikoodin kommenttien kirjoittajilla tiedossaan, sillä ohjeisiin kirjoittamattomia yhteneväisyyksiä löytyy. Toisaalta genreillä on myös esimerkiksi rakenteeseen tai kieleen liittyviä normeja, jotka voivat olla tiukkojakin (Pietikäinen & Mäntynen 2009: 83). Yksi kiistämätön normi, joka kommenttien tekstilajiin liittyy, on niiden muoto //- tai /*-alkuisena.

Genrejen luokittelua on joiltain tahoilta kyseenalaistettu esimerkiksi pohtimalla, voiko yksittäinen teksti kuulua vaihteleviin genreihin, joiden rajat ovat häilyväisiä (Heikkinen & Voutilainen 2012: 30–33). Ei myöskään ole ennalta määrättyä, minkä verran yhteisiä piirteitä teksteillä pitää olla, jotta niiden voi sanoa kuuluvan samaan tekstilajiin (Mäntynen & Shore 2008: 30). On tunnustettu, että genret ovat avoimia ja sumearajaisia, mutta se on perusteltavissa kielenkäytön inhimillisyydellä: luokat ovat dynaamisia ja muuttuvia (Heikkinen & Voutilainen 2012: 30–33). Lajit ovat aina suhteessa toisiinsa, ja suhteet voivat muuttua ja lajit sekoittua toisiinsa (Mäntynen & Shore 2008: 33). Lajien luokkiin tai tyyppeihin jaottelu on kuitenkin luontevaa, kun ihmisillä on käytettävissään erilaisiin kielenkäyttötilanteisiin erilaisia toiminnan ja kielenkäytön tapoja (Heikkinen & Voutilainen 2012: 30–33).

Tutkiessani yhtä tekstilajiksi olettamaani luokkaa niin, että tavoitteenani on selvittää, millainen tekstilaji on kyseessä, on minun myönnettävä, että lajien empiirinen analyysi on väistämättä tyypillisyyksien etsimistä ja lajiin kuuluvuuden ehtojen pohtimista (Heikkinen & Voutilainen 2012: 33). Tekstilajeja luokittelemalla tai tyypittelemällä on mahdollista saada tietoa ja ymmärrystä erilaisten kielenkäyttötilanteiden käytänteistä ja havaintoja siitä, kuinka maailmaa jonkin tekstilajin näkökulmasta tarkkaillaan. Jo nämä ovat itsessään perusteita sille, että jonkin mahdollisen tekstilajin – ohjelmointikoodin kommenttien – piirteitä ja lainalaisuuksia on mielekästä tutkia.

Vaikka genreä on tutkittu lähinnä kirjallisuudentutkimuksen ja kielitieteen saralla, ovat kysymykset genrestä aiheellisia monilla muillakin tieteenaloilla (Heikkinen & Voutilainen 2012:19). Tästä syystä tutkimukseni ei ainoastaan kuvaa kommenttien tekstilajia vaan yhdistää sen ohjelmoin-

tiin. Tuon genretutkimuksen askeleen pidemmälle tietojenkäsittelytieteisiin yhdistettynä ja pyrin osoittamaan, millainen rooli luonnollisen kielen ulkopuolelle jäävillä elementeillä on tekstilajin määrittelyssä.

Kuvaan tekstilajin määrittelyä ja tutkimusta tarkemmin seuraavassa luvussa, jossa esittelen tutkimuksen taustateoriana käyttämäni systeemis-funktionaalisen teorian.

2.2.3 Systeemis-funktionaalinen teoria

Systeemis-funktionaalisen teorian esittämä tekstilajiteoria on yksi vaikutusvaltaisimmista lingvistisen tutkimuksen piirissä (Heikkinen 2012: 65). Kyseessä on 1980-luvulla syntynyt M. A. K. Hallidayn genre- ja rekisteriteoria, jonka keskiössä on alusta asti ollut kielenkäytön laji (Ventola 2006: 96, Shore 2012a: 131). Perusajatuksena systeemis-funktionaalisen teorian taustalla on se, että kielen muotojen ja rakenteiden käyttöä motivoivat sosiaalinen ympäristö ja puhetilanne (Ventola 2006: 97). Teoriaa taustoittaa siis ajatus siitä, että kieli on yksi sosiaalisen toiminnan muoto, joka määritellään ensisijaisesti toimintana ja tekemisenä. Kieli ja sosiaalinen todellisuus ovat aina saumattomassa yhteydessä toisiinsa. Teoria keskittyy kuvaamaan sitä, miten kielellä ilmaistaan merkityksiä ja miten kielen funktionaalinen perusluonne on rakentunut kielen systeemiin sisään. (Luukka 2002: 89–90, 98.)

Systeemis-funktionaalisen teorian vaikutukset näkyvät siinä, kuinka sen käsitykset lajeista ovat vaikuttaneet moniaalle ja etenkin tekstintutkimukseen (Shore 2012a: 131). Teorian lajikeskeisyys sopii hyvin tämän tutkimuksen tarkoituksiin, joten käytän aineistoa tulkitessani systeemis-funktionaalisen teorian lähtökohtia ja käsitteistöä. Tämän teorian piirissä ei vallitse täyttä yhtenäisyyttä käsitteiden ja näkemysten osalta (Shore 2012a: 131), ja erilaisia suuntauksia, sovelluksia ja kehittäjiä on valtavasti (Luukka 2002: 90), joten esittelen teorian taustalla olevan historian lyhyesti ja sen jälkeen keskityn systeemis-funktionaaliseen teoriaan sellaisena kuin teorian varsinainen luoja Halliday sen esittää (Luukka 2002: 90–91).

Systeemis-funktionaalisen teorian edelläkävijänä pidetään J.R. Firthiä, joka ei koskaan muotoillut ajatuksiaan varsinaiseksi teoriaksi, mutta jolla oli vaikutuksensa Hallidayn systeemis-funktionaaliseen kielitieteeseen (Luukka 2002: 95–96). Firth kehitti neljä nykyteorian synnyn pohjalla toiminutta käsitettä: tilannekontekstin, osakielen, merkityksen hajottamisen eri tasoihin ja merkitysten analyysin systeemeinä ja rakenteina. Tilannekontekstilla Firth tarkoittaa, että kieli on aina sosiaalista toimintaa, joka esiintyy jossakin kontekstissaan. Myös kulttuuri ja kieli riippuvat toisistaan. (Halliday & Hasan 1985: 9–10, Shore 2012a: 132.) Näen ohjelmointikoodin kommentitkin selkeänä sosiaalisena toimintana, vaikka ne usein osoitetaan vain tulevaisuuden itselle, joka

lukee koodia myöhemmin. Tilannekonteksti on selvä ja todella tarkkaan rajattu näissä kommenteissa, sillä koodin kommentti ei ole koodin kommentti, mikäli se ei esiinny koodin seassa.

Osakieli on tietty osa kielestä, jota analysoidaan – on siis valittava tutkimuksen kohteeksi joku osa, sillä koko kieltä on mahdotonta analysoida (Shore 2012a: 133). Tämän tutkimuksen osakieleksi nimeäisin ohjelmointikoodin kommentit aineistoni ohjelmissa. Merkityksen hajottamisella Firth tarkoittaa yksinkertaistetusti sitä, että kaikki kielen tasot aina kontekstista sanastoon ja fonologiaan rakentavat kielen merkityksiä. Systeemi muodostuu Firthin mukaan niistä paradigmaattisista vaihtoehdoista, jotka voi valita tiettyyn rakenteeseen, ja rakenne tarkoittaa sitä kokonaisuutta, jonka systeemit muodostavat. (Shore 2012a: 133–134.)

Firthin oppilasta Hallidayta pidetään systeemis-funktionaalisen teorian kiistattomana isänä. (Luukka 2002: 91). Halliday kehitteli teoriaansa 1960- ja 1970-luvuilla kantavana ajatuksenaan se, että kieliopin ja kielentutkimuksen motivaation tulee aina olla lähtöisin tästä maailmasta (Luukka 2002: 92). Halliday määrittelee kielen kulttuurin rakentajaksi ja sosiaalisten tilanteiden toiminnaksi (Halliday 1978: 1–3). Kieltä opitaan ja käytetään vuorovaikutuksessa yhteisön muiden jäsenten kanssa, ja kieli on olemassa vain ihmisyhteisöä varten (Luukka 2002: 98). Hallidayn metafunktiot ovat esimerkiksi Shoren mukaan käyttökelpoinen tapa tehdä tekstintutkimusta, ja samoin ovat ajatelleet myös monet muut suomenkielisten tekstien tutkijat, jotka ovat käyttäneet metafunktioina analysoinnin välineinä (Shore 2012a: 156).

Systeemis-funktionaalisen teorian nimen systeemi-osuus viittaa siihen, että kyseessä on systeeminen teoria, jossa merkitykset nähdään valintoina ja kieli on näiden merkitysvalintojen verkko (Halliday & Hasan 1985: 14). Kielisysteemillä tarkoitetaan kielellisten valinnanmahdollisuuksien järjestäytynyttä joukkoa eli merkityspotentiaalia, jota kielenkäyttäjä voi käyttää (Luukka 2002: 104). Funktionaalisella lähestymistavalla taas tarkoitetaan sitä, että kieltä tarkastellaan tekoina ja merkityksiä rakentavina toimintoina. Kieltä pohditaan siltä kannalta, miten ihmiset käyttävät sitä eri tilanteissa erilaisiin tehtäviin. (Luukka 2002: 101.) Hallidayn teoriassa funktionaalisuus on kielen systeemin ominaisuus, ei pelkän kielen käytön ominaisuus (Halliday & Hasan 1985: 46).

Keskeisessä asemassa Hallidayn teoriassa on siis kielen funktionaalisuus, ja hänen pyrkimyksenään on saada yleiskuva kielen käyttötarkoituksista (Halliday & Hasan 1985: 15–17). Tätä varten Halliday esittelee kolme metafunktiota, jotka muodostavat koko kielisysteemin perustan (Luukka 2002: 103). Metafunktiot ovat tapoja luoda merkityksiä, jotka ovat läsnä kaikissa sosiaalisissa konteksteissa (Halliday 1978: 112). Metafunktioita ovat ideationaalinen, interpersoonainen ja tekstuaalinen, ja niiden muodostamat kielen osa-alueet ovat toisistaan riippuvaisia, sillä jokainen kielellinen ilmaus pitää sisällään yhtä aikaa ideationaalisia, interpersoonaisia ja tekstuaalisia merkityksiä (Halliday & Hasan 1985: 15–17, Luukka 2002: 103).

Ideationaalinen metafunktio tarkoittaa sitä, että käytämme kieltä maailman konstruoimiseen eli todellisuuden hahmottamiseen ja siitä kuvan rakentamiseen (Halliday & Hasan 1985: 45, Luukka 2002: 102). Tämä metafunktio jakautuu vielä kahtia eksperientaaliseen ja loogiseen, joten metafunktioita voi ajatella olevan myös neljä (Halliday & Hasan 1985: 45). Eksperientaalisella metafunktiolla viitataan siihen, miten käytämme kieltä maailman tapahtumien, toimintojen ja tilojen konstruoimiseen, ja eksperientaalinen merkitys tarkoittaa niitä lauseen tiettyjä ominaisuuksia, jotka heijastavat todellista maailmaa kuten sen koemme. Halliday osoittaa eksperientaalisen metafunktion merkityksen analysoimalla lausetta, joka sisältää erilaisia metaforisia tasoja. (Halliday & Hasan 1985: 18–19, 45.) Loogisella metafunktiolla puolestaan viitataan siihen, miten kieltä käytetään konstruoimaan olioiden ja tapahtumien yhdistelmiä, siis siihen, miten lauseet ja lausekkeet kytketään kielellä toisiinsa. Jokaisessa luonnollisessa kielessä on keinoja ilmaista loogisia suhteita esimerkiksi rinnastuksen ja alistuksen keinoin. Hallidayn esimerkkilauseessa loogista merkitystä luo ja-konjunktio kahden lauseen välissä, sillä se rakentaa lauseiden välistä merkityssuhdetta. (Halliday & Hasan 1985: 20–21, 45.)

Interpersoonaisen metafunktion avulla ylläpidetään vuorovaikutussuhteita ja ilmaistaan mielipiteitä, tunteita, asenteita ja arviointeja (Luukka 2002: 102). Sillä tarkoitetaan kielellä suoritettavaa sosiaalista kanssakäymistä ja sitä, mitä kielellä halutaan saada aikaiseksi, sillä kielellä rakennetaan puhujan ja kuulijan välistä vuorovaikutusta. Interpersoonaisella metafunktiolla tarkoitetaan osallistujarooleja ja modaalisuutta eli esimerkiksi modaalisia lausetyyppejä, kommenttiadverbiaaleja, persoonaa ja intonaatiota. (Halliday & Hasan 1985: 20, 45.)

Tekstuaalisen metafunktion alalle kuuluvat tekstin koheesiokeinot ja erilaiset teeman- ja informaationkulut. Tekstuaalisella merkityksellä tarkoitetaan sitä, miten kielen avulla rakennetaan tekstiin erilaisia tasoja; tekstuaalinen merkitys rakentaa merkityksen tekstiin. (Halliday & Hasan 1985: 23, 45, ks. myös Shore 2012a: 145–149.)

Halliday on kehittänyt eteenpäin Firthin esittelemää tilannekontekstia, joka on Hallidayn teorian tärkein kielen ja sosiaalisen kontekstin yhteyttä kuvaava käsite. Käsitteen avulla luodaan ajatus siitä, että kieli on olemassa vain, kun se toimii jossakin ympäristössä, ja myös siitä, että tietyt kielelliset valinnat ovat mahdollisia ja todennäköisiä joissakin tilanteissa. (Luukka 2002: 99.) Tilannekonteksti muodostuu alasta, osallistujarooleista ja kielen ilmenemismuodosta (Halliday & Hasan 1985: 12).

Ala on se sosiaalinen toiminta, jonka osana teksti esiintyy ja johon kielen käyttäjät ovat sitoutuneet. Tekstin ala tarkoittaa yksinkertaisesti sitä, mitä kielenkäyttötilanteessa tapahtuu, ja se ilmenee ideationaalisten merkitysten kautta. (Halliday & Hasan 1985: 12, 25, Halliday 1978: 143.) Osallistujarooleilla viitataan siihen, ketkä osallistuvat kielenkäyttötilanteeseen ja mitkä heidän statuk-

sensa ja roolinsa ovat (Halliday & Hasan 1985: 12, Halliday 1978: 144). Osallistujaroolit jakautuvat sosiaalisiin rooleihin, jotka kuvaavat kielenkäyttötilanteen osallistujien suhdetta tilanteeseen, ja kielellisiin rooleihin, jotka ovat tekstin sisällä ja ilmaistaan kielellisin keinoin (Shore 2012a: 134). Osallistujaroolit ilmenevät tekstin interpersoonaisista metafunktioista (Halliday & Hasan 1985: 25). Ilmenemismuodolla tarkoitetaan kielen roolia kielenkäyttötilanteessa. Sillä tarkoitetaan konkreettista vuorovaikutuksen kanavaa, eli esimerkiksi sitä, onko teksti puhuttua vai kirjoitettua. Ilmenemismuodolla viitataan myös tilannesidonnaisuuden asteeseen ja kielenkäyttötilanteen pyrkimyksiin, eli siihen, mitä tilanteen osallistujat odottavat kielen aiheuttavan. Ilmenemismuoto havaitaan tekstin tekstuaalisten merkitysten kautta. (Halliday & Hasan 1985: 12, Halliday 1978: 144–145.)

Esittelen esimerkinomaisesti nämä käsitteet tämän tutkimuksen kannalta. Ala on karkeasti ajateltuna Java-ohjelmointi, sillä ohjelmointi on toiminta, jonka osana kommentit (eli teksti) esiintyvät. Sosiaaliset osallistujaroolit kommenttien kirjoittamisessa voisivat olla esimerkiksi ohjelmoija, joka koodia kirjoittaa, ja toinen ohjelmoija, joka sitä lukee. Kielelliset roolit taas voisivat olla kirjoittaja ja lukija. Ohjelmointikoodin kommenttien ilmenemismuotona voisi pitää ainakin sitä, että ne on kirjoitettu luettaviksi ja niiden pyrkimys on oppikirjaselityksen mukaan selittää koodia. (Ks. Shore 2012a: 134–135.)

Hallidayn oppilas James Martin on myös luonut omanlaistaan versiota systeemis-funktionaalisesta teoriasta. Hänen versiossaan kiinnostuksen kohteena on diskurssisemantiikka eli lausetta laajemmissa yksiköissä reaalistuvat merkitykset. Martinin teoriassa osa käsitteistä on ristiriidassa Hallidayn esittelemien käsitteiden kanssa, ja muutoinkin diskurssisemantiikan analyysit ovat melko erillään leksikkokieliopeista. (Shore 2012a: 151–156.) Tämän takia en perehdy enempää diskurssisemantiikan alaan, vaan jätän sen mainitsemisen tasolle, vaikka se onkin osa systeemis-funktionaalista teoriaa.

2.2.4 Rakenteen näkökulma tekstilajiin

Hallidayn läheinen kollega Ruqaiya Hasan on esittänyt tekstin rakenteen olevan kiinteässä suhteessa genreen ja hän on muutoinkin korostanut rakenteen merkitystä esimerkiksi onnistuneen tekstin kirjoittamisessa ja kielten opiskelussa (Halliday & Hasan 1985: 68–69). Hasanin mukaan juuri tekstin kokonaisrakenne määrittää sen genren, eli toisiaan rakenteellisesti muistuttavat tekstit edustavat samaa genreä (Ventola 2006: 97). Yksi Hasanin keskeisimmistä käsitteistä onkin tekstin yleinen rakennepotentiaali, joka sisältää tekstin rakenteen pakolliset ja vapaaehtoiset osat (Halliday & Hasan 1985: 64).

Pakolliset osat määräävät sisältönsä ja järjestyksensä puolesta tekstin genren, ja valinnaiset osat ovat tekstilajin kannalta mahdollisia, mutta eivät pakollisia (Halliday & Hasan 1985: 62). Jos

siis pakolliset osat ovat samanlaiset kahdessa eri tekstissä, edustavat ne samaa tekstilajia, vaikka valinnaiset osat olisivatkin erilaisia. Pelkät pakolliset osat riittävät tekemään tekstistä lajinsa edustajan. (Halliday & Hasan 1985: 60–62.) Valinnaisetkaan osat eivät ole mielivaltaisesti päätettyjä, vaan niissä pätevät tekstilajin määräämät säännöt, jotka voidaan nimetä ja osoittaa (Halliday & Hasan 1985: 62).

Rakennepotentiaalissa voidaan huomioida myös osien toistuminen ja järjestys (Halliday & Hasan 1985: 64). Toistumisella tarkoitetaan sitä, että joku osa esiintyy tekstissä useammin kuin kerran, ja se on aina valinnainen ilmiö (Halliday & Hasan 1985: 63). Tämän lisäksi Hasan on luonut Hallidayn tilannekontekstin jatkoksi käsitteen ”contextual configuration”, joka tarkoittaa tilannekontekstin tiettyä tilaa, eli se on yhdistelmä tietyn alan, osallistujaroolin ja ilmenemismuodon arvoista (Halliday & Hasan 1985: 56). Contextual configuration on siis yksi ilmentymä tilannekontekstista, kun taas rakennepotentiaali sisältää tekstin rakenteen osat (Halliday & Hasan 1985: 64).

2.2.5 Kommenttien tekstilaji

Kommentti-nimitystä käytetään arkikielessä monista toisistaan poikkeavista teksteistä, ja yhtä yhtenäistä tekstilajia tästä kirjavasta joukosta on tuskin mahdollista löytää. Kommenteiksi kutsutaan esimerkiksi sanomalehtien painettuja kommenttijuttuja, verkkoartikkeleiden lopusta löytyviä kenen tahansa kirjoittamia mielipiteitä ja kannanottoja ja käännöksen yhteyteen liitettyjä selityksiä siitä, mitä käännöksessä on tehty ja miksi (Tampereen yliopisto: Käännöskommentit). Kielitoimiston sanakirja (Kielitoimiston sanakirja: kommentti) määrittelee kommentin selitykseksi ja reunahuomautukseksi.

Nevalainen (2013) on tutkinut pro gradu -tutkielmassaan tarkemmin sanomalehtien kommenttijuttujen tekstilajisuutta, ja hänen tuloksensa esittävät, että kommenttijutun tekstilaji on varsin ongelmallinen, sillä sen kirjoittamiseen ei ole annettu juuri minkäänlaisia ohjeita ja se muistuttaa paljon muita tekstilajeja. Kommenttijuttujen tekstilajisuutta puolestaan puoltaa se, että niiltä löytyvät selkeät tavoitteet. (Nevalainen 2013: 74.) Tilanne on samankaltainen myös ohjelmointikoodin kommenttien kohdalla, sillä tarkkoja kielellisiä vaatimuksia ei ole missään annettu, mutta niiden tavoitteet ovat kuitenkin kaikille tekstilajin käyttäjille selkeät. Voi siis olla, että kommenteiksi luettavilla tekstilajeilla on jotakin yhteistä.

Seitsemännen luokan äidinkielen ja kirjallisuuden oppikirjassa kommentti määritellään näin: ”Kommentti on lyhyt lisäselitys tai -näkökulma esimerkiksi uutiseen.” (Keinänen, Paalanen, Tiainen 2007: 104). Lukion äidinkielen ja kirjallisuuden oppikirjassa kommentin kerrotaan esittävän lisätietoja tai uuden näkökulman johonkin tekstiin, ja kommentin yleiseksi paikaksi nimetään kommentoitavan lehtijutun vierus (Grünn, Günthal & Uusi-Hallila 2004: 240). Lukiolaisille suunnattu

Verkko-Piste (2008) määrittää kommentin tekstilajin artikkelin tai uutisen yhteydessä julkaistuksi erilliseksi tekstiksi, jossa kirjoittaja tuo esille mielipiteensä käsiteltävästä asiasta.

Tekstilajeista puhuttaessa kommenteiksi mielletään siis lähinnä uutisen tai artikkelin yhteydessä esiintyvä kommenttijuttu. Yhteistä eri lähteiden määrittelyille on se, että kommentti tuo lisäarvoa jollekin toiselle tekstille.

2.2.6 Diskurssiyhteisö

Diskurssiyhteisöksi voidaan kutsua ihmisjoukkoa, jolla on yhteiset tekstiä ja tekstin tehtäviä koskevat normit (Kalliokoski 2002: 149). Diskurssiyhteisön käsitteen genren käsitteen yhteyteen on nostanut John Swales (Shore & Mäntynen 2006: 29, Swales 1990: 21–27). Hänen mukaansa tekstilaji on diskurssiyhteisön vakiintunut tekstiluokka, ja lähtökohdat diskurssiyhteisön määrittelyssä ovat yhteisissä kielellisissä tavoitteissa, joihin yhteisön jäseniksi päässeet pyrkivät (Swales 1990: 24–27, 48–58).

Swales määrittelee kuusi kriteeriä, joiden perusteella ryhmää voidaan kutsua diskurssiyhteisöksi, ja ne ovat seuraavanlaiset: (1.) Diskurssiyhteisöllä on yleisesti hyväksytyt yhteiset tavoitteet. (2.) Diskurssiyhteisöllä on vakiintuneet kanavat jäsenten väliseen kommunikointiin. (3.) Diskurssiyhteisön jäsenet osallistuvat yhteisön kommunikointiin hankkimalla tietoa ja antamalla palautetta. (4.) Diskurssiyhteisö hyödyntää ja käyttää jäsentensä tunnistamia tekstilajeja, jotta yhteisön tavoitteet saadaan täytettyä. (5.) Diskurssiyhteisö käyttää omaa erityissanastoaan. (6.) Diskurssiyhteisön jäsenet hallitsevat yhteisön aiheet ja diskurssit. (Swales 1990: 24–25, 28.)

Swalesin määritelmää diskurssiyhteisöstä pidetään melko tiukkana, sillä hänen tekemänsä rajaukset diskurssiyhteisöstä ja siihen kuulumisesta ovat tarkkaan määriteltyjä, ja harvat yhteisöt täyttävät kaikki kriteerit. Jos joku edellä luettelemistani diskurssiyhteisön kriteereistä puuttuu, kyseessä ei Swalesin mukaan ole diskurssiyhteisö. (Swales 1990: 21–27, Shore & Mäntynen 2006: 29.) Kriteerit toimivatkin parhaiten tieteellisen kirjoittamisen ja akateemisen yhteisön ympäristössä, jossa Swales on niitä tarkastellut (Shore & Mäntynen 2006: 29). Swalesin diskurssiyhteisölle luomat kriteerit ovat kuitenkin toimiva lähtökohta monenlaiselle tutkimukselle (ks. esim. Vuorijärvi 2013: 35). Diskurssiyhteisö voi myös muodostua yhteisen harrastuksen tai kiinnostuksen kohteen ympärille (Vuorijärvi 2013: 33).

Swalesin mukaan tekstilajin nimi on yksi kriteeri tekstin tekstilajisuudelle, ja tekstilajin nimeää sitä käyttävä diskurssiyhteisö (Swales 1990: 54–55). Tämä on kielitieteen kannalta uusi ajatus (Shore & Mäntynen 2006: 31). Ohjelmointikoodin kommentteja kutsutaan yksinkertaisesti kommenteiksi (ks. esim. Wikla 2003: 15), ja vaikka kommentilla voidaan viitata moniin muihinkin tekstilajeihin (ks. luku 2.2.5), on teksteille annettu oma nimi, jota diskurssiyhteisön jäsenet käyttävät.

Java-kielen yhteydessä kommentti ei tarkoita yhteisön jäsenelle mitään muuta kuin koodiin kirjoitettua kommenttiriviä, jollaisia tässä työssä tutkin.

Jotta tekstilajia voidaan tutkia yhteisössään, on sen yhteisö ensin määriteltävä (Vuorijärvi 2013: 32). Ohjelmointikoodin kommenttien kirjoittajien diskurssiyhteisön voisi ajatella olevan yhtä ohjelmointikoodin kirjoittajien diskurssiyhteisön kanssa, sillä kommentteja kirjoitetaan ainoastaan koodin yhteydessä ja niitä luetaan koodin yhteydestä. Toisaalta kommentit ovat koodin luonnolliskielinen osa, ja tutkimukseni kohteena on juuri kommenttien tekstilaji, joten käytän vastedes diskurssiyhteisön käsitettä kommenttien kirjoittajien yhteydessä. Kommenttien diskurssiyhteisö täyttää ainakin osan Swalesin kriteereistä (Swales 1990: 24–25), sillä kommenttien kirjoittajilla on selkeä päämäärä kuvata koodissa tapahtuvia asioita, he käyttävät vakiintunutta kanavaa eli kommentoivat aina koodin yhteyteen, ja kommenttien kirjoittajat hallitsevat koodauksen alan.

Olen nostanut diskurssiyhteisön käsitteen osaksi analyysilukua 5.2.2, jossa käsittelen kommenteissa esiintyvää erityissanastoa.

3 Tutkimusaineisto

Aineistona tutkimuksessa on 60 ohjelmointikoodia, joissa on Java-kielellä kirjoitettua koodia ja koodiin liittyviä kommentteja ja joista jokaisen on kirjoittanut eri ihminen. Aineisto koostuu lausekielinen ohjelmointi -kurssin viimeisestä ja pakollisesta harjoitustyöstä, ja aineiston minulle on luovuttanut tietojenkäsittelyopin yliopistonlehtori Jorma Laurikkala.

Olen itse opiskellut lausekielisen ohjelmoinnin kurssin sekä aiheisiin syvemmälle paneutuvan olio-ohjelmoinnin perusteiden kurssin. Tästä syystä en joudu analysoimaan kommenttien tekstilajia ulkopuolisena tarkkailijana, vaan minulla on ohjelmointiin ja sen kommentointiin liittyvää tietoa jo valmiiksi. Osaan lukea aineiston koodeja kokonaisuuksina ja pystyn hahmottamaan, mitä kommenttien tekniset sisällöt tarkoittavat. Tällaiset lähtökohdat antavat minulle välineet ymmärtää tekstilajia diskurssiyhteisön jäsenenä (ks. luku 2.2.6) ja myös ottaa huomioon kommenttien ja koodin kytköksiä toisiinsa, kun hahmotan, mitä kommenttien kommentoimat koodirivit tarkoittavat. Lähteenä käyttämäni Jorma Laurikkalan luentomateriaalit olenkin saanut haltuuni käytyäni kyseisen kurssin.

3.1 Aineiston valinta ja piirteet

Aineiston harjoitustyö on laajahko Java-kielellä kirjoitettu ohjelma, jonka toimintaan on annettu tarkkaan määritellyt ohjeet mutta jonka toteutuksesta kukin opiskelija on vastannut itsenäisesti. Jokainen aineistoon kuuluva ohjelma toimii siis käytännössä tismalleen samalla tavalla, mutta ratkaisuvaihtoehtoja ohjelman toteutukselle voi oikeastaan sanoa olevan yhtä monta kuin opiskelijoitakin. Persoonalliset toteutukset ja yhteinen tehtävänanto tekevät koodeista ja niiden kommenteista tutkimuksen kannalta vertailukelpoisia.

Lausekielisen ohjelmoinnin kurssi on osa tietojenkäsittelytieteiden perusopintoja, eli se on pakollinen kaikille tietojenkäsittelytieteitä pääaineenaan opiskeleville. Kurssin osaamistavoitteisiin kuuluu muiden muassa taito ohjelmoida itsenäisesti pieniä ohjelmia Java-kielellä, taito kirjoittaa helposti ymmärrettäviä ohjelmia ja taito hallita monimutkaisuutta aliohjelmien avulla (Tampereen yliopiston opinto-opas 2012–2015). Kurssille tulevilla opiskelijoilla ei tarvitse olla minkäänlaisia lähtötietoja Java-ohjelmoinnista, mutta viimeiseen harjoitustyöhön mennessä he ovat suorittaneet puolen vuoden aikana 10 opintopisteen laajuisen kurssin ja tehneet yhden aiemmankin laajan harjoitustyön.

En paljasta harjoitustyön tarkkaa aihetta tai kurssin ajankohtaa, jotta opiskelijoiden yksityisyydensuoja on taattu, ja yksittäisten kommenttien esittäminen esimerkkeinä mahdollistuu. Samasta syystä en myöskään liitä tutkimuksen oheen harjoitustyön tarkkaa tehtävänantoa, mutta kuvaan sii-

nä esiintyvät oleelliset piirteet tässä luvussa. Olen allekirjoittanut Jorma Laurikkalan kanssa aineistonkäyttösopimuksen, jossa lupaan olla näyttämättä kokonaisia koodeja kenellekään seminaarin ohjaajaa lukuun ottamatta. Yksittäiset havainnollistavat esimerkit eivät kuitenkaan nähdäkseni mahdollista kenenkään yksittäisen opiskelijan tunnistamista, ja lopulta kyse on myös erittäin yleistasoisesta työstä. Kommentit eivät sisällä henkilökohtaisia tietoja, ja jos jotakin koodin kirjoittajan henkilöllisyyteen viittaavaa löytyykin kommenteista, en tietenkään ota sitä julkiseksi esimerkiksi tutkimukseen. Vastaanotin koodit ilman opiskelijoiden nimiä, opiskelijanumeroita ja sähköpostiosoitteita, jotka saattoivat jossakin vaiheessa koodia ilmetä, joten myöskään itselläni ei ole tietoa siitä, keiden kirjoittamia ohjelmia tutkin.

Opiskelijoiden töiden tutkiminen antaa mahdollisuuden perehtyä laajaan aineistoon ja tutkia samankaltaisia toimintoja jokaisesta koodista, kun jokainen koodi aiheuttaa samanlaisen lopputuloksen ohjelman muodossa. Toisaalta kontekstina tällainen ohjelmoinnin perusteiden kurssi on antanut aloitteleville ohjelmoijille rajoitteita koodaamiseen ja kommentoimiseen. Harjoitustöiden tehtävänannossa on kommentointiin ohjeistettu seuraavanlaisesti: ”-- kommentoi riittävästi ja oikeissa paikoissa, liitä jokaiseen metodiin yleisluontoinen kommentti --", mikä tarkoittaa, että ilman kommentteja ei ole jäänyt yksikään läpimennyt harjoitustyö. En kuitenkaan näe tätä ongelmana, sillä tarkoituksenani on tutkia kommentteja ja niiden suhdetta koodiin, joten on tietenkin tärkeää, että kommentteja esiintyy. Kunkin opiskelijan on myös itse tehtävä ratkaisu siitä, mikä on riittävää kommentointia ja missä sijaitsevat oikeat paikat kommenteille, sillä yllä kuvaamani ohjeistus on ainoa tehtävänannossa esiintyvä viittaus kommentointiin. Vaihtelua ilmeneekin yhteisestä tehtävänannosta huolimatta, sillä vähäisimmillään kommenttien suhde koko koodin pituuteen on 7 % ja enimmillään 36 %.

Koodeja aineistossa on yhteensä 119 kappaletta ja tutkin näistä valittua 60 koodin satunnaisotosta. Valitsin jokaisesta koodista samankaltaisen kiinteän kokonaisuuden eli metodin tutkittavakseni. Java-ohjelmoinnin näkökulmasta metodilla tarkoitetaan nimettyä algoritmia, jonka voi ajatella isossa ohjelmakokonaisuudessa eräänlaisena aliohjelmana (Wikla 2003: 39–40). Metodi suorittaa jonkun selkeästi rajatun toiminnon. Esimerkiksi todella yksinkertainen metodi voisi laskea kaksi lukua yhteen ja palauttaa lukujen yhteenlasketun arvon.

Valitsin tutkittavakseni juuri metodin laajuisen osan jokaista ohjelmaa, sillä metodit ovat helposti käsiteltävissä ja vertailtavissa olevia kokonaisuuksia niin kokonsa kuin sisältönsäkin puolesta. Kokonaisten ohjelmien kommenttien tutkiminen rajaisi koodien määrän todella pieneksi, jolloin aineistossa ei esiintyisi eri kommentoijien tuomaa variaatiota. En tehnyt metodin valintaa mielivaltaisesti, sillä jokaisesta koodista on mahdollista löytää eräänlainen liikutusmetodi, joka ei tietenkään jokaisella opiskelijalla aiheuta tismalleen samaa toimintoa, mutta joka toimintojensa, aihepiirinsä ja

laajuutensa puolesta on vertailukelpoinen toisten opiskelijoiden liikutusmetodien kanssa. Sen lisäksi, että liki jokaisesta koodista on mahdollista löytää liikutusmetodi, se on myös sopivaa peruskoodia, joka tuskin tarvitsee enempää tai vähempää kommentointia kuin muukaan koodi.

Karsin aineistosta heti ennen satunnaisotosta pois sellaiset koodit, joiden kommentointi on tehty englanniksi. Tällaisia oli 119:n joukossa ainoastaan kaksi kappaletta. Aineistoon kuuluu myös yksi tyhjä harjoitustyö, jossa on vain viesti opettajalle eikä lainkaan koodia tai kommentteja, ja myös sen jätin alkuvaiheessa pois.

Ohjelmointikoodin pituutta mitataan riveissä, ja lyhyin metodi on 15 riviä pitkä ja pisin metodi 268 riviä pitkä. Riveiksi lasketaan myös tyhjät rivit. Keskimäärin rivejä metodissa on 81 kappaletta. Kommenttienkin määrä vaihtelee radikaalisti eri opiskelijoilla: vähimmillään kommentteja on koko metodissa vain yksi ja enimmillään 56. Yhteensä tutkimuksen kohteena olevia kommenttirivejä on 873 kappaletta, ja näistä omiksi kommenteikseen lasken 805 kappaletta. Olen määritellyt usean rivin muodostavan yhden kommentin, mikäli ne esiintyvät peräkkäin ja käsittelevät samaa aihetta. Esimerkiksi esimerkissä 3 on määrittelytapani mukaan vain yksi kommentti, vaikka kommenttirivejä onkin neljä ja ensimmäinen rivi on kokonaan oma virkkeensä. Kommentin kokonaisuus on niin selkeä, että kyseessä on ainoastaan yksi kommentti.

(3) **//Liikkumiskomento siirtää matoa haluttuun suuntaan paitsi jos madon vartalo on liikkumisen esteenä.**
//Tällöin palautetaan alkuperäinen mato. Tarkastellaan myös erikoistilanne, jossa madon pää
//on ennen liikkumista kentän reunalla ja seuraava komento saa madon liikkumaan kentän
//vastakkaiselle reunalle. (A43)
if (v == 'u') {

Olen saanut jokaisesta anonyymista koodin tekijästä taustatietoina sukupuolen, tieteenalayksikön, tutkinto-ohjelman ja aiemman ohjelmointikokemuksen. Näistä tiedoista tutkimuksen kannalta relevantiksi nousee ainoastaan ohjelmointikokemus, sillä tieto ohjelmoijien tasosta ohjaa suhtautumaan heidän kirjoittamiinsa koodeihin ja kommentteihin oikealla tavalla. Ohjelmointikokemuksen asteikkona on 1–5, joista 1 tarkoittaa ”ei ohjelmointikokemusta ollenkaan” ja 5 ”vahvaa ohjelmointikokemusta”. Opiskelijoista 60 % arvioi ohjelmointitaitonsa arvosanalla 1 ja 28 % arvosanalla 3. Vain kolme opiskelijaa arvioi ohjelmointitaitonsa arvosanalla 5. Toisin sanoen suurin osa aineiston opiskelijoista on alkeistason ohjelmoinnin opiskelija, ja tietämys ohjelmoinnin ja kommentoinnin käytänteistä on syntynyt ja kehittynyt kurssin aikana.

3.2 Aineiston käsittelyn ja analysoinnin menetelmät

Aineiston käsittelymenetelmät ovat melko yksinkertaisia, sillä vastaanotin aineiston valmiiksi sähköisessä muodossa. Aineisto koostuu käytännössä 119:ä Java-muotoisesta tiedostosta, joista jokaisella on eri tekijä. Ennen aineiston minulle luovuttamista Jorma Laurikkala oli numeroinut koodien tekijät niin, ettei kenenkään henkilöllisyys paljastu, mutta jokaisella koodilla on kuitenkin oma tunnistetietonsa (eli anonyymi 1, anonyymi 2 ja niin edelleen). Tämän lisäksi sain Excel-tiedoston, josta löytyy taustatiedot jokaisesta anonyymista numeroituna niin, että koodin kirjoittaja ja taustatiedot on mahdollista yhdistää toisiinsa.

Aloitin aineiston käsittelyn etsimällä koodeista sellaista metodia, joka löytyisi samankaltaisena jokaisesta koodista. Muutamasta vaihtoehdosta valitsin liikuttelumetodin, sillä liikkuminen oli joko osana metodin nimeä tai siihen liitettyä kommenttia liki jokaisessa koodissa. Luvussa 2.1.2 esittelemässäni kuvitteellisessa pelissä liikutusmetodi voisi tarkoittaa sitä ohjelman osasta, joka liikuttaa pelin x:n käyttäjän valitsemaan suuntaan. Kun käyttäjä antaa esimerkiksi komennon ’a’ peli kutsuu liikutusmetodia, jossa x:n siirto suorakaiteen alareunaan toteutetaan, ja lopputuloksena pelikentän x on liikkunut käyttäjän valitsemaan suuntaan. Tämän jälkeen kopioin manuaalisesti jokaisesta aineiston koodista tuon liikuttelumetodin sellaisenaan omaan tekstitiedostoonsa, ja siitä edelleen vielä Excel-tiedostoon, jossa kommenttien piirteiden tehokas käsittely on mahdollista.

Olen poiminut esimerkkejä sattumanvaraisesti kaikista aineiston metodeista, ja ilmoitan aina esimerkin yhteydessä opiskelijan tunnistetietona toimivan numeron 1–60 (esim. A11). Esimerkkien esityksessä olen päätynyt sellaiseen ratkaisuun, että kommenttien fontti on kokonaan tummennettu, jotta niiden erottaminen koodista onnistuu myös ohjelmointikoodin lukemiseen tottumattomalta. Erityiset kielenpiirteet olen lisäksi alleviivannut. Aineiston ohjelmointikoodien kirjoittajien yksityisyyttä suojellakseni olen muuttanut esimerkkien kommenteista ja koodeista pelin keskushahmon nimen yksinkertaisesti *hahmoksi,* jotta pelin aihe pysyisi salassa. Tämä ei varsinaisesti vaikuta analyysiin, sillä pelihahmon ainoa oleellinen piirre on sen kuvitteellinen elollisuus, ja myös alkuperäiseen pelihahmoon viitataan aineistossa aina yleisnimellä, ei erisnimellä. Näin vuosikurssi, jonka työtä tutkin, pysyy anonyymina.

Aineiston analyysin olen jäsentänyt Hallidayn metafunktioiden avulla kolmeen osaan, sillä niiden avulla aineiston keskeiset ja tekstilajia määrittävät piirteet on mahdollista osoittaa. Interpersoonainen metafunktio toimii kehyksenä kommenttien tyypillisten persoonamuotojen ja modaalisuuden ilmaisemiselle, ideationaalinen niille tavoille, joilla maailmoja rakennetaan kommenteissa ja tekstuaalinen kommenttien löyhille koheesiokeinoille. (Ks. metafunktioista esim. Halliday & Hasan 1985: 44–45.) Analyysin jäsentäminen osiin näiden käsitteiden avulla auttaa hahmottamaan, miten

ohjelmointikoodin kommenttien eri piirteet kytkeytyvät toisiinsa, ja millaisia merkityksiä niihin sisältyy.

4 Ohjelmointikoodin kommenttien yleisluonne

Tavoitteenani on rakentaa kuva siitä, millainen on ohjelmointikoodin kommenttien muodostama tekstilaji. Suomenkielisistä kommenteista ei ole aiemmin tehty lingvististä tutkimusta, ja kommentit ovat muutenkin vierasta aluetta muille kuin ohjelmoinnin ja kommentoinnin diskurssiyhteisön jäsenille (ks. Swales 1990: 24–27). Siksi tämän luvun tarkoitus on rakentaa kuva siitä, millaisia ohjelmointikoodiin kirjoitetut kommentit yleisesti aineistossa ovat. En huomioi vielä tässä yhteydessä Hallidayn metafunktioita, joiden avulla tarkka kuva tekstilajin erilaisista merkityksistä rakentuu (ks. Halliday & Hasan 1985: 44–45), vaan keskityn kommenttien kokonaisvaltaisiin piirteisiin. Näiden piirteiden kuvaus on pohjana seuraavalle luvulle, jossa analysoin kommentteja tarkemmin metafunktioihin jäsenneltyinä.

Aloitan kuvaamalla koodin vaikutusta kommentteihin ja nimeämällä yleisimmät tyypit, jotka kommenteista voidaan sen kommentoiman koodin perusteella nimetä. Tarkoituksena on osoittaa, miten erilaiset koodirakenteet ohjaavat kirjoittamaan erilaisia kommentteja. Tämän jälkeen siirryn syvemmälle koodin ja kommentin yhteyteen ja esittelen kommenttien vaikutuspiirin koon vaihtelevuutta, sillä erilaiset kommentit viittaavat koodiin erilaisin tavoin. Näiden koodin luomiin lähtökohtiin perustuvien piirteiden osoittaminen on tärkeää, jotta kokonaiskuva kommenttien tekstilajista saadaan rakennettua. Kaikki kielenkäyttö esiintyy jossakin kontekstissa (Halliday & Hasan 1985: 45), joten kommenttien kontekstin huomiotta jättäminen jättäisi myös tekstilajin kuvauksen vajavaiseksi.

4.1 Koodisisällön ja kommentin muodon yhteys

Aineiston ohjelmointikoodissa käytetään tiettyjä Java-kielen rakenteita runsaasti, ja on mahdollista osoittaa, että erilaisia ohjelmointirakenteita kommentoidaan eri tavalla – voisi oikeastaan sanoa, että tietyillä koodirakenteilla on omia kommenttityyppejään. Nämä kommenttityypit liittyvät toisiinsa sisällöllisten ja rakenteellisten samankaltaisuuksien kautta. Olen jakanut ohjelmointirakenteet karkeasti muutamaan ryhmään, joista esittelen niihin liittyvien kommenttien tyypilliset piirteet. Rakennetyyppejä ovat metodi, if-ehtorakenne, for-toistorakenne, metodikutsu sekä muuttujien esittely. Näiden tyyppien ulkopuolelle jää osa koodista ja näin ollen kommenteistakin, mutta nostan tässä esille kaikkein keskeisimmät rakennetyyppien muodostamat ryhmät, sillä tutkimuksen tavoitteena ei ole luoda mahdollisimman tyhjentävää kommenttien kirjon kuvausta, vaan yleiskuva kommenttien tekstilajin luonteesta.

Aineisto koostuu ohjelmoinnin peruskurssin harjoitustöistä, joten aineiston koodissa käytetyt rakenteet ovat melko yksinkertaisia ja niiden määrä on rajallinen. Javassa olisi mahdollista käyttää myös tämän harjoitustyön laajuuden ja vaatimustason ulkopuolelle jääviä rakenteita, mutta keskiössä on osoittaa, millainen yhteys koodilla ja kommentilla on, mikä onnistuu myös suppeammalla määrällä erilaisia ohjelmointirakenteita. Näin myös tämän työn tarkoitus lingvistisenä tutkimuksena ei jää ohjelmoinnin monimutkaisuuden varjoon.

4.1.1 Metodinesittelykommentti

Kutsun metodiin liittyvää kommenttia metodinesittelykommentiksi. Sellaisessa usein kerrotaan, mitä toimintoja metodissa on ja mahdollisesti myös siihen liittyvistä parametreista ja paluuarvoista[2]. Laurikkala antaa luentomateriaaleissaan ohjeen kirjoittaa aina ennen metodin otsikkoa ”yleisluonteinen kommentti metodin tarkoituksesta sekä mahdollisista parametreista ja paluuarvoista” (Laurikkala 2014e: 16), mikä on todennäköisesti yksi vaikutin siihen, että aineistossa melkein jokaista metodia ennen löytyy metodinesittelykommentti. Aineistossa ainoastaan kymmenessä koodissa metodinesittelykommenttia ei ole. Esimerkkien 1 ja 2 metodinesittelykommentit ovat melko tyypillisiä ryhmässään, minkä tunnistaa useista kielellisistä piirteistä. Olen lihavoinut esimerkeistä koko kommentin, jotta se erottuu esimerkkiin sisältyvästä koodista selkeästi. Jatkan tällä menettelyllä esimerkkien suhteen koko työn ajan, sillä monissa kohdissa koodi- ja kommenttirivit vuorottelevat, ja haluan varmistaa kommenttien helppolukuisuuden myös sellaisille lukijoille, jotka eivät ole tottuneita lukemaan vuorottelevia rivejä koodia ja kommentteja.

(4) **// Metodi hahmon liikuttamiseen. Parametreja ovat seuraavat: pelikenttä, hahmon**
// paikkataulukko sekä käyttäjän antama syöte. Paluuarvo on true, jos hahmo
// ei törmää itseensä. Jos se puolestaan törmää, paluuarvo on false, ja peli
// saadaan näin päätettyä. (A41)
public static boolean liikutaHahmoa(char[][] kentta, int[][] hahmonPaikka,
char annettu) {

Esimerkin 4 kommentin pituus on neljä riviä, kun taas suurin osa muista, ei metodia esittelevistä kommenteista on vain yhden rivin mittaisia. Ensimmäinen metodinesittelykommenteille tyypillinen piirre onkin, että ne ovat usein muita kommentteja pidempiä. Esimerkin 4 kommentti ei ole ainoastaan pitkä, vaan se muodostaa neljän virkkeen mittaisen tekstikokonaisuuden, minkä voi tunnistaa jo siitä, että teksti on kirjoitettu ilman virkkeiden jälkeen tulevia rivivälejä. Lauseet jatkuvat suoraan toistensa perään ja rivivälit ovat ainoastaan jakamassa kokonaisuutta sopivan mittaisiin riveihin.

[2] Parametrien avulla voidaan antaa metodille lisätietoja, ja ne sijaitsevat metodin nimen jälkeen suluissa: *metodin nimi (parametrit)*. Paluuarvoksi kutsutaan sitä arvoa, jonka metodi voi palauttaa sitä kutsuneeseen paikkaan koodipätkän lopussa. (Wikla 2003: 42–43)

Kommentin viimeinen virke liittyy edeltäjäänsä jatkaen samasta aiheesta (*paluuarvosta*) ja sidosteisuutta rakentaa myös *puolestaan*-adverbi. Rakenteellisesti useat metodinesittelykommentit ovat paitsi pitkiä myös usean lauseen mittaisia kokonaisuuksia.

Seuraava piirre, joka on tyypillinen lähes kaikille metodinesittelykommenteille, on koko kommentin alku sanalla *metodi*. Tämä yksinkertainen piirre on mainitsemisen arvoinen sikäli, että muiden rakenteiden tyypillisenä piirteenä ei ole kommentin alussa sijaitseva rakenteen nimi, kuten seuraavista luvuista käy ilmi. Metodi-sanan korostamisen syynä voi olla esimerkiksi se, että halutaan painottaa metodin laajuutta muihin kommentoitaviin piirteisiin nähden. Kommentin vaikutusalakin on laajempi kuin muissa kommenteissa (ks. luku 4.2), joten ehkä metodin nimellä aloittamalla on tarkoitus osoittaa se, ettei kommentti kosketa ainoastaan seuraavaa riviä, jolla metodi esitellään.

Sanan *metodi* lisäksi tietyt sanat esiintyvät useissa metodinesittelykommenteissa, ja jo Laurikkalan ohjeistuksesta metodia ennen esiintyviin kommentteihin selviää, että nämä sanat ovat *parametri* ja *paluuarvo* (Laurikkala 2014e: 16). Juuri nämä sanat ovat esimerkin 4 virkkeiden ensimmäisiä sanoja: ”*Parametreja ovat seuraavat—*", ”*Paluuarvo on true--*" ja ”*—paluuarvo on false--*". Finiittiverbin edessä sijaitseva sana on usein suomen kielessä lauseen teema, joka osoittaa puheenaiheen (Alho & Kauppinen 2009: 180, VISK § 871), eli nämä sanat ovat virkkeissä keskeisillä paikoilla osoittamassa puheenaiheen. Niiden merkitys on tärkeä metodinesittelykommentin piirteiden joukossa.

Sisällöllisesti tällaisessa kommentissa on välttämätöntä esitellä, mitä metodi tekee tai saa aikaan (*Metodi hahmon liikuttamiseen*). Lisäksi kommentti usein sisältää tiedon siitä, mihin se johtaa eli mitä metodi palauttaa (*Paluuarvo on true, jos hahmo ei törmää itseensä*; *Jos se puolestaan törmää, paluuarvo on false*). Tämän lisäksi parametrit voidaan antaa tavalla tai toisella, ja esimerkissä 4 ne on annettu kaksoispisteen jälkeen alkavassa listassa (*Parametreja ovat seuraavat:*).

Esimerkki 5 sisältää samoja metodinesittelykommentille tyypillisiä piirteitä kuin esimerkki 4: Se on tavallista kommenttia pidempi, sillä tekstiosuus on kolme riviä ja koko kommentti viisi riviä pitkä, ja kommentti muodostaa kahden virkkeen ja kolmen lauseen mittaisen tekstin, jonka voi lukea yhtenä kokonaisuutena.

(5)
```
*/
* Metodi saa parametrina hahmon paikkataulukon, kenttätaulukon ja käyttäjän syötteen/
* ja siirtää hahmon paikkaa käyttäjän syötteeseen perustuen./
* Metodi palauttaa tilanteesta riippuen uuden tai alkuperäisen paikkataulukon./
*/ (A32)
public static int[][] liikutaHahmoa(int[][] paikkaTaulukko, char syote, char[][] kentta) {
```

Esimerkin 5 kommentti alkaa sanalla *metodi*, ja myös toisen virkkeen teemapaikalla on tämä rakenteen nimi. Myös sana *parametri* löytyy, mutta *paluuarvon* sijaan kommentissa on käytössä verbi *palauttaa*, jolloin sisällölliset piirteet ovat samat. Kommentista käy ilmi, mitä metodi tekee (*siirtää hahmon paikkaa käyttäjän syötteeseen perustuen*), mitkä ovat sen parametrit (-- *saa parametreina paikkataulukon, kenttätaulukon ja käyttäjän syötteen*--) ja mitkä ovat sen paluuarvot (-- *palauttaa tilanteesta riippuen uuden tai alkuperäisen paikktaulukon*).

Esimerkit 4 ja 5 ovat tyypillisiä esimerkkejä pitkästä metodinesittelykommentista. Aineistosta löytyy näiden pitkien lisäksi myös lyhyempiä metodinesittelykommentteja. Esimerkit 6–8 ovat lyhyempiä metodinesittelykommentteja, joista puuttuu osia äsken esittelemästäni piirrejoukosta.

(6) **// Metodi, joka muuttaa hahmon koordinaatteja halutun käskyn mukaisesti** (A31)

```
public static int[][] hahmonLiikutus(char[][] merkit, int[][] paikat, char kasky){
```

(7) **// Metodi hahmon liikuttamiseen** (A38)

```
public static boolean liikutaHahmoa(char suunta, int[][] hahmo, int[] hahmonPaa, int rivit, int sarakkeet) {
```

Esimetkit 6 ja 7 ovat yhtä kaikki tunnistettavia metodinesittelykommentteja. Pituudeltaan molemmat ovat vain yhden rivin mittaisia, eikä virkkeitä ole enempää kuin yksi, joten varsinaista tekstikokonaisuutta ei muodostu. Molemmat esimerkit kuitenkin alkavat *metodi*-sanalla, mikä sitoo ne kiinteäksi osaksi luvassa olevaa metodia. Sen sijaan sanoja *parametri* tai *paluuarvo* ei esiinny kummassakaan kommentissa. Sisällöllisistä piirteistä yksi löytyy kummastakin esimerkistä, sillä niiden kommenteissa esitellään, mitä metodi tekee (esimerkissä 6: *muuttaa hahmon koordinaatteja halutun käskyn mukaisesti* ja esimerkissä 7: *Metodi hahmon liikuttamiseen*). Esimerkissä 6 otetaan esiin metodin parametreja, sillä yksi metodin parametreista on *char kasky*, ja kommentissa viitataan siihen, että toimitaan *halutun käskyn mukaisesti*.

Esimerkki 8 on myös lyhyt metodinesittelykommentti, mutta toisin kuin esimerkeissä 6 ja 7 kommentti ei ala sanalla *metodi* eikä sisällä kyseistä sanaa missään muussakaan kohdassa.

(8) **/* Siirretaan hahmoa kayttajan antamaan suuntaan.**
***/** (A3)

```
public static int siirraHahmo(char[][] kentta, int[][] hahmo, int hahmonPituus, int suunta, int reikia) {
```

Esimerkin 8 ainoat metodinesittelykommentille tyypilliset piirteet ovat sisällöllisiä, sillä myös tämän esimerkin kommentissa kerrotaan, mitä metodilla tehdään (*Siirretaan hahmoa kayttajan antamaan suuntaan*). Samoin kuin esimerkissä 7 tässäkin kommentissa viitataan parametriin ilman sa-

naa *parametri*, sillä yhden parametriluettelon jäsenen nimi on *int suunta*, ja kommentissa mainitaan ”*käyttäjän antama suunta*”.

Näiden aineistoa kuvaavien esimerkkien valossa voi todeta, että metodinesittelykommentille tyypillisiä ovat usean rivin mittainen pituus, *metodi*-sanalla kommentin aloittaminen, *parametreihin* ja *paluuarvoihin* suoraan tai välillisesti viittaaminen sekä metodin tarkoituksen kertominen ja parametrien ja paluuarvojen esitteleminen. Näistä ainoa pakollinen piirre on metodin tarkoituksen kertominen, sillä ilman kaikkia muita piirteitä metodinesittelykommentti on yhä metodinesittelykommentti. Muut piirteet ovat valinnaisia. (Pakollisista ja valinnaisista piirteistä ks. Halliday & Hasan 1985: 62.)

4.1.2 Ehtorakenteen kommentti

Seuraavaksi otan käsittelyyn if-rakennetyypin[3], josta käytän tästedes nimitystä ehtorakenne, sillä esimerkiksi Wikla (2003: 26) kutsuu if-lausetta ehdolliseksi lauseeksi. Myös kommenteista käy ilmi se, että rakenteessa keskiössä on juuri ehto, joten nimitys ehtorakenne on havainnollisempi kuin esimerkiksi jos-rakenne. Tämän rakenteen kommenteissa on aineistossa kaksi selkeää pääryhmää. Ensimmäisessä ryhmässä kerrotaan ehto, jolla ehtorakenteen sisältämät toiminnot suoritetaan, ja toisessa kerrotaan, mitä rakenteella pyritään tarkastamaan.

Esimerkit 9–13 edustavat kommentteja, joissa kerrotaan ehtorakenteen ehto.

(9) **//Jos hahmo törmää seinään, siirretään hahmo kentän vastakkaiselle laidalle.** (A32)
```
if (paikkaTaulukko[0][1] < 0)
    paikkaTaulukko[0][1] = kentta[0].length - 1;
```

(10) **// Jos käyttäjä päättää lopettaa peli päättyy heti.** (A42)
```
if (suunta == LOPETA) {
    System.out.println(HYVASTIT);
}
```

Esimerkkien 9 ja 10 ehtolauseeseen liittyvissä kommenteissa kerrotaan ehto ehtolauseen suorittamiselle sekä se, mitä ehdon täyttyessä tapahtuu. Tällaisessa tapauksessa sivulauseen ja päälauseen suhdetta kommentissa voi kutsua konditionaaliseksi, sillä toinen lauseista esittää ehdon, jonka täyttyessä toisen lauseen kuvaama tapahtuma tapahtuu, mikä on konditionaalisen suhteen määritelmä

[3] If-lauseella eli ehdollisella lauseella määrätään, että tietty osa koodista suoritetaan vain, mikäli tietty ehto täyttyy (Wikla 2003: 26). Esimerkiksi:

```
if (ehto x)
    koodia, joka tapahtuu, jos ehto x täyttyy
else
    koodia, joka tapahtuu, jos ehto x ei täyty
```

(VISK § 1134). Nämä esimerkit ovat aineistossa esiintyvistä ehtorakenteen kommenteista kaikkein tyypillisimpiä. Molemmat alkavat *jos*-konjunktioisella sivulauseella, jossa kerrotaan se, mikä ehtolauseen ehto on. Tyypillisesti konditionaalista suhdetta merkitseekin juuri jos-adverbiaalikonjunktio (VISK § 1134). Toisaalta osasyynä *jos*-alkuisuudelle voi olla myös se, että Java-kielen *if*-lauseen suomennos olisi *jos*-lause.

Esimerkin 9 sivulause *Jos hahmo törmää seinään* liittyy koodilauseeseen, jossa taulukon kokoa verrataan nollaan, ja esimerkissä 10 sivulause *Jos käyttäjä päättää lopettaa* liittyy koodilauseeseen, jossa *suunta*-muuttujan arvo on *LOPETA*. Nämä ovat siis ehdot sekä luonnollisen kielen konditionaalisessa rakenteessa että koodissa, minkä havaitsee siitä, että ehdoille löytyy vastineensa koodista. Tyypilliseen ehtorakenteeseen liittyvään kommenttiin kuuluu myös, että sivulauseen jälkeen seuraavassa päälauseessa kuvataan, mitä tapahtuu, jos sivulauseen kuvaama ehto toteutuu. Esimerkissä 9 kerrotaan, mihin hahmo siirretään, ja esimerkissä 10, että peli päättyy heti.

Nämä kommentit paitsi sisältävät luonnolliskielisen kuvauksen siitä, mitä koodin ehtorakenteessa tapahtuu, myös tekevät sen koodin kanssa samassa järjestyksessä. Koodissa ensin annetaan ehto (esimerkki 10: *if(suunta == LOPETA)*), ja vasta tämän jälkeen siirrytään tapahtumiin ehtorakenteen sisällä eli niihin tapahtumiin, jotka toteutuvat vain ehdon täytyttyä. Kommentissakin ehdon sisältävä sivulause on ensimmäisenä, ja sitä seuraa tieto ehdon täytyttyä tapahtuvista asioista.

Esimerkissä 11 on tapaus kommentista, jollaisia löytyy aineistosta jonkin verran ehtorakenteeseen liittyneenä.

(11)
```
// Jos ei olla peruuttamassa. (A39)
    if (liikutaan == true) {
        // For-silmukka syötetaulukon häntää edeltävien koordinaattien korvaamiseen.
        for (int i = pituus - 1; i > 0; i--) {
            …
```

Tämä esimerkin 11 ensimmäinen kommentti sisältää ainoastaan irrallisen *jos*-alkuisen sivulauseen, eikä se siis muodosta konditionaalista suhdetta. Kommentilla kuvataan ainoastaan ehtorakenteen sisältämä ehto, eikä oteta kantaa siihen, mitä ehdon täytyttyä tapahtuu. Kommentitta ei kuitenkaan jää ehtorakenteen sisälle jäävä koodi, sillä esimerkin 11 tapauksessa heti ehtolauseen jälkeen on kirjoitettu uusi kommentti, jossa kerrotaan tulevista tapahtumista. Muodoltaan kommentti on erilainen kuin se olisi, mikäli lauseista olisi yhdistetty yhden virkkeen mittainen konditionaalinen kokonaisuus, mutta merkityksen tasolla on mahdollista ajatella, että konditionaalinen suhde on pilkottu ja piilotettu kahteen kommenttiin. Yhdeksi kommentiksi muokattuna esimerkin 11 kommentit voisivat näyttää tältä: *// Jos ei olla peruttamassa, korvataan for-silmukassa syötetaulukon häntää edeltävät koordinaatit.*

Ehtorakenteen ehdon sisältävistä kommenteista löytyy aineistossa myös sellaisia tapauksia, joissa *jos*-konjunktiota ei ole lainkaan. Tällaisten kommenttien alkuun voi kuitenkin ajatella sisältyvän implisiittisen *jos*-konjunktion, sillä ne esittävät samalla tavalla ehtorakenteen ehdon kuin *jos*-konjunktion sisältävät kommentit (ks. esimerkit 9–11). Esimerkkien 12 ja 13 tapauksissa kommentti alkaa kuvaamalla ehtorakenteen ehdon (*Komentona siirto ylös* ja *Virheellinen parametriarvo*) ilman *jos*-konjunktiota. Esimerkin 13 kommentti antaa myös tiedon ehdon täytyttyä tapahtuvasta koodista, sillä *Palataan heti virhearvon kanssa* viittaa koodissa ehtolausetta seuraavaan riviin (*return null;*). Vaikka kommentista puuttuu konditionaaliselle lauseelle tyypillinen *jos*, on kommentin kahden lauseen kokonaisuus silti merkityssisällöltään konditionaalinen.

(12)
```
// Komentona siirto ylös. (A33)
    if(komento == YLOS) {
        if(paaRivInd == 0)
            paaRivInd = kentta.length -1;
            …
```

(13)
```
// Virheellinen parametriarvo. Palataan heti virhearvon kanssa. (A60)
    if (jono == null || jono.length() == 0 || i < 0 || i > jono.length() - 1)
        return null;
```

Kommenttien lauseista puuttuu alistuskonjunktion lisäksi predikaattiverbi. Ajatusta implisiittisestä *jos*-konjunktiosta tukee se, että merkityksen muuttumatta kommenttien lauseisiin voi lisätä konjunktion ja predikaattiverbin: *// Jos komentona on siirto ylös* ja *// Jos parametriarvo on virheellinen*. Mahdollinen selitys tällaiselle kommentointiratkaisulle on siinä, että kommentin pituus ja monimutkaisuus halutaan pitää minimissä. Esimerkkien 12 ja 13 kommentit ovat supistuneisuudestaan huolimatta ymmärrettäviä ehtorakenteen kommentteja, ja niiden sisältö koostuu ainoastaan kaikkein tärkeimmistä ja informaatiorikkaimmista elementeistä.

Toisen selkeän ryhmän ehtorakenteeseen liittyvien kommenttien joukossa muodostavat tarkistuskommentit. Nämä kommentit alkavat useimmiten tarkastusta kuvaavalla verbillä ja sisältävät tiedon siitä, mitä ehtorakenteella tarkastetaan, eli toisin sanoen kuvaavat ehtorakenteen ehdon. Esimerkki 14 on tyypillisimmästä tilanteesta, jossa ehtorakentetta kommentoidaan *jos*-rakenteen sijaan tarkistusrakenteella, ja myös esimerkki 15 sisältää tarkastuskommentin.

(14)
```
// Tarkistetaan, että taulukoille on varattu muistia. (A41)
    if(kentta != null && hahmonPaikka != null) {
```

(15)
```
//Tarkistetaan mentiinkö reunan yli (A55)
    if (apuhahmotaulu2[0][1] == -1)
        apuhahmotaulu2[0][1] = pelikentta[0].length-1;
```

Esimerkissä 14 ehtorakenteella testataan, onko *kentta*- ja *hahmonPaikka*-taulukoille varattu muistia, kuten kommenttikin kertoo. Ehtorakenteen ehto sijaitsee kommentissa *että*-konjunktioalkuisessa sivulauseessa, joka tulee *tarkistaa*-verbin jälkeen. Kommentti tarkoittaisi liki samaa asiaa, jos se alkaisi passiivimuotoisen *Tarkistetaan*-verbin sijaan *jos*-konjunktiolla (*// Jos taulukoille on varattu muistia*). Samalla tavalla myös esimerkissä 15 virkkeen alisteinen kysymyslause kertoo, mikä ehtorakenteen ehto on, ja rakenteen voisi korvata sisällön paljoa muuttumatta jos-rakenteella (*// Jos mentiin reunan yli*).

Merkitys on siis lähes sama kuin aiemmin esittelemilläni *jos*-alkuisilla kommenteilla. *Tarkistetaan*-rakenne on kuitenkin hieman *jos*-rakennetta raskaampi, sillä siinä päälauseita on kaksi yhden sijaan, kun *tarkistetaan* muodostaa oman päälauseensa, jonka jälkeen ehto vasta esitellään. Mahdollisen syyn sille, että koodaaja päätyy kommentoimaan tällä rakenteella, tarjoaa ehtorakenteen sisältämä koodikokonaisuus. *Tarkistetaan*-muotoa kommentissa käytettäessä koodin ehtorakenteen jälkeen jatketaan suoraan seuraaviin koodirakenteisiin, eli koodissa ei tapahdu mitään sellaista, mitä voisi kommentissa erityisesti kuvailla ehdon toteuduttua tapahtuneeksi. *Tarkastaa*-verbillä viitataan siihen, että ehdon täytyttyä jatketaan normaalisti eteenpäin, kun tarkastuksen tulos osoittautuu positiiviseksi, tai ollaan jatkamatta, mikäli se osoittautuu negatiiviseksi.

If-rakenteen kaikille kommenteille yhteistä on se, että ne sisältävät vähintään koodin ehtorakenteen suluissa esitetyn ehdon luonnolliselle kielelle käännettynä. Kommentit voivat myös sisältää *jos*-konjunktion ja kuvauksen ehdon toteuduttua tapahtuvasta toiminnosta. Mahdollinen on myös rakenne, jossa ehtoon viitataan *tarkistetaan*-päälauseen jälkeen sijaitsevassa sivulauseessa.

4.1.3 Toistorakenteen kommentti

Kolmas kommenttityyppi liittyy for-silmukoihin[4], joita kutsun tästedes toistorakenteiksi ja -lauseiksi, sillä esimerkiksi Wikla (2003: 29) käyttää kyseisestä rakenteesta nimitystä toistolause. Tämän tarkennuksen tekeminen on oleellista siksi, että toistolauseille on myös muita mahdollisia rakenteita kuin for-rakenne (esimerkiksi while-rakenne), mutta koska esittelen tässä tutkimuksessa ainoastaan for-toistorakenteeseen liittyvät kommentit, voin nähdäkseni kutsua for-rakennetta yleisesti toistorakenteeksi ilman sekaannusten syntymistä. Yleisesti käytetty nimitys toistorakenteille on myös silmukka. Toistorakenteiden yhteydessä esiintyvillä kommenteilla sisältöön ja rakenteeseen liittyvät säännönmukaisuudet eivät ole yhtä tarkkoja ja selvärajaisia kuin aiemmin kuvaamillani metodilla ja ehtorakenteella. Tästä huolimatta on mahdollista osoittaa selkeitä toistuvia ominai-

[4] Silmukaksi eli toistolauseeksi kutsutaan lausetta, jolla voi helposti käydä läpi joitakin arvoalueita. Silmukan sisällä olevia lauseita suoritetaan niin kauan, että for-lauseessa annettu jatkamisehto ei pidä paikkaansa. For-lauseen muoto on *for (alkuasetus; jatkamisehto; eteneminen) { toistettavat lauseet }*. (Ks. Wikla 2003: 27–29.)

suuksia ennen toistorakennetta sijaitsevissa kommenteissa. Esimerkkien 16 ja 17 kommentit kommentoivat toistorakenteita ja toimivat tehtävässään tyypillisellä tavalla.

(16) **//Päivitetään hahmoa eteenpäin** (A55)

```
for (int rivi=hahmonpituus-1; rivi>0; rivi--) {
    for (int sarake = 0; sarake < 2; sarake++) {
        apuhahmotaulu2[rivi][sarake] = hahmotaulu[rivi-1][sarake];
    }
```

(17) **// Siirretään hahmon ruho lähemmäs päätä.** (A5)

```
for (int i = 1; i < hahmo.length; i++) {
    hahmo[hahmo.length - i][0] = hahmo[hahmo.length - (i + 1)][0];
    hahmo[hahmo.length - i][1] = hahmo[hahmo.length - (i + 1)][1];
}
```

Esimerkkien 16 ja 17 rakenteet ovat keskenään samanlaiset passiivialkuisen predikaattiverbin osalta (*Päivitetään* ja *Siirretään*), ja myös lauserakenteet muistuttavat toisiaan, sillä lauseenjäsenet ovat järjestyksessä predikaatti, objekti ja adverbiaali. Kyseessä on tyypillinen tiivis kommentin lauserakenne, sillä mitään ylimääräistä informaatiota ei anneta, ja predikaatti on passiivimuotoinen (ks. luku 5.1.1). Erityisesti toistorakenteelle ominaiseksi kommentin tekee sen sisältö eli se, mitä kommentoidaan. Kumpikaan kommentti ei viittaa silmukointiin tai alkioiden läpikäymiseen, mitä toistorakenteen avulla voidaan tehdä, vaan molempien kommenttien sisältö liittyy vasta toistolauseen sisällä tapahtuviin asioihin. Esimerkissä 16 sisältö koskettaa hahmon päivittämistä eteenpäin, ja vasta kahden toistorakenteen jälkeen koodissa toteutuu tämä kuvattu tapahtuma (esimerkin kolmas rivi koodia). Esimerkissä 17 on aivan samanlainen tilanne, sillä vasta silmukan sisällä olevissa lauseissa tapahtuu kommentissa kuvattu ”siirtäminen”.

Tämä tilanne muistuttaa ehtorakenteen yhteydessä esiintyvään kommenttiin mahdollisesti liittyvää osaa, jossa kuvataan ehtorakenteen toteuduttua tapahtuvia tapahtumia. Erona toistorakenteen yhteydessä on se, että kommentissa jätetään koko silmukka huomiotta: se on ikään kuin näkymätön kommentille, ja kommentin sisältö koskettaa vasta myöhempänä sijaitsevia koodirivejä. Kommentilla voi tällaisessa tapauksessa paitsi selostaa koodin tapahtumia, myös osoittaa, mikä osa koodista on sen tärkeintä ydintä, sillä silmukka on usein vain väline jonkin keskeisemmän asian tekemiselle. Toisaalta voi esittää kysymyksen siitä, miksi kommentoitujen toistorakenteiden tapauksissa kommentoidaan ennen silmukkaa eikä vasta juuri ennen sitä tapahtumarikasta osaa, jota halutaan kommentoida. Syynä tälle voi olla ainakin se, että toistorakenne vaikuttaa usein koodiin niin, että ei ole sama asia suorittaa lausetta sen sisällä ja ulkopuolella, joten kommentin ajatellaan kattavan koko sen luoman kokonaisuuden, kun se sijaitsee ylemmällä tasolla. Suurin osa toistorakenteisiin liittyvistä kommenteista on siis predikaattialkuisia ja silmukan sisältöön kantaa ottavia tapauksia, joissa itse silmukka jätetään mainitsematta.

Aineistossa huomattavasti pienempi mutta kuitenkin erottuva osa toistorakenteisiin liittyvistä kommenteista viittaa itse silmukkaan. Yleisimmäksi osoittautunut keino tehdä näin löytyy esimerkin 18 kommentista. Siinä käytetään ilmausta *käydä läpi* kiinnittämään huomio toistorakenteen tekniseen toimintaan. Kommentin yhteys toistolauseeseen rakentuu esimerkissä myös siten, että kommentissa viitataan silmukassa esiintyvään *rivi*-muuttujaan sekä määrään (*yksi*), jonka silmukka lisää laskuriinsa jokaisen kierroksen jälkeen (*rivit yksi kerrallaan*). Tällainen kommentti liittyy tiiviisti ainoastaan sitä seuraaviin silmukoihin, eikä ota kantaa siihen, mitä niiden sisällä sijaitsevassa koodissa tapahtuu.

(18)
```
// Käydään läpi rivit yksi kerrallaan. (A51)
    for (int rivi = 0; rivi < kohde.length; rivi++)
        for (int sarake = 0; sarake < kohde[0].length; sarake++)
```

Toisenlainen ja aineistossa harvinaisempi tapa viitata suoraan toistolauseeseen esiintyy esimerkin 19 kommentissa. Siinä toistorakenne sidotaan kommenttiin yksinkertaisesti käyttämällä sanaa *silmukka*. Tämän jälkeen kuvataan toiminta, joka silmukan sisällä tapahtuu (*hahmon vartalo liikutetaan*). Jos kommentista jättäisi pois alun (*Silmukka, jossa*) se menisi samaan kategoriaan tyypillisimpien toistorakenteeseen liittyvien kommenttien kanssa (ks. esimerkit 16–17).

(19)
```
// Silmukka jossa hahmon vartalo liikutetaan. (A25)
    for(int i = 1; i < hahmo.length; i++) {
        hahmo[hahmo.length - i][0] = hahmo[hahmo.length - (i+1)][0];
        hahmo[hahmo.length - i][1] = hahmo[hahmo.length - (i+1)][1];
    }
```

Jos ehtorakenteen kommenteista voi puuttua alusta *jos* siten, että sen voi silti nähdä kuuluvan impliseettisesti lauseeseen, voihan myös olla, että silmukkaan liittyvistä kommenteista suurin osa on pudottanut *silmukka, jossa* -alun tarpeettomana pois, jolloin koko silmukkalause jää kommentoimatta ja kommentin vaikutusala siirtyy koskettamaan silmukan sisällä sijaitsevaa koodia. Tällainen rakenne on kommenttien diskurssiyhteisön jäsenille täysin ymmärrettävä, joten tiiviyden nimissä ratkaisu on yleinen.

4.1.4 Muuttujanesittelykommentti

Muuttujan esittelyyn[5] liittyvät kommentit ovat aineistossa usein selkeästi samanmuotoisia ja samansisältöisiä. Tästäkin kommenttiryhmästä löytyy kahta erilaista tapaa kommentoida esiteltäviä muuttujia, mutta ensin esittelemäni tapa on huomattavasti yleisempi. Esimerkit 20 ja 21 edustavat aineiston tyypillisintä muuttujanesittelykommenttimuotoa.

(20) **// muuttujat joitten mukaan hahmoa liikutetaan** (A40)
```
int x = 0;
int y = 0;
boolean lopetus = false;
```

(21) **//lippumuuttuja, joka ilmaisee onko yritetty liikkumissuunta ok** (A30)
```
boolean dirOk = true;
```

Esimerkkien 20 ja 21 kommentit koostuvat kahdesta osasta. Ensimmäinen osa on kommentin alussa sijaitseva sana *muuttuja* taikka jokin määriteyhdyssana[6], jonka edusosa on sana *muuttuja*, kuten esimerkissä 21 (*lippumuuttuja*). Toinen osa on *joka*-alkuinen relatiivilause, joka toimii *muuttuja*-sanan määritteenä (-- *joitten mukaan hahmoa liikutetaan* ja -- *joka ilmaisee onko yritetty liikkumissuunta ok*). Ensimmäisen osan tehtävänä on kiinnittää kommentti sitä seuraavaan muuttujaan tai muuttujiin, siis osoittaa, että juuri muuttujaa kommentoidaan. Esimerkin 21 tapaisissa kommenteissa ensimmäinen osa myös nimeää muuttujalle osoitetun tarkoituksen yhdyssanan määriteosalla, sillä *lippumuuttuja* on yleisesti tunnettu nimitys tietynlaiselle muuttujalle.

Kommentin toinen osa antaa tietoa muuttujan tarkoituksesta, mikä etenkin esimerkin 20 tapaisissa kommenteissa on välttämätöntä, sillä pelkkä sana *muuttuja* ei anna vielä mitään lisäarvoa tai tulkinta-apua koodille. Relatiivilauseen kuvaamat toiminnot evät yleensä toteudu vielä kommenttia seuraavassa koodissa eli muuttujanesittelyrivillä, sillä muuttujan todellinen käyttöympäristö voi olla vasta myöhemmin koodissa, ja käyttötarkoitus tulee luonnollisesti ilmi vasta käyttöympäristössä. Esimerkiksi esimerkin 20 jälkeen on kahdeksan riviä esiteltyihin muuttujiin täysin liittymätöntä koodia ennen kuin muuttujaa käytetään siinä tarkoituksessa, joka muuttujanesittelykommentissa on kerrottu, eli avustamaan hahmon liikuttamisessa. Väli esiintymisympäristöön vaihtelee, sillä esimerkissä 21 rivejä on välissä vain yksi.

Samaa tyyppiä esimerkkien 20 ja 21 kanssa on periaatteessa myös esimerkki 22, vaikkei se noudatakaan samanlaista lauserakennetta.

---

[5] Muuttujan esittelyllä tarkoitetaan ensimmäistä kertaa, kun muuttuja esiintyy koodissa. Esittelyssä annetaan muuttujalle tietotyyppi ja uniikki nimi. Esimerkiksi muuttujanesittelylauseessa ”*int x = 0;”* tietotyyppi on *int*, muuttujan uniikki nimi on *x* ja muuttujalle annettu arvo on *0*.

[6] Määriteyhdyssana koostuu määriteosasta ja edusosasta (VISK § 403). Esimerkiksi yhdyssanan *kumisaapas* määriteosa on *kumi* ja edusosa on *saapas*.

(22) // **Ruuan muuttujat.** (A42)
```
int ruuanSar = 0;
int ruuanRiv = 0;
```

Esimerkin 22 sisällössä on samoja piirteitä, sillä myös tässä esiin nostetaan sana *muuttuja* ennen muuttujien esittelyä, ja muuttujille annetaan löyhä tarkoitus, sillä ne ovat juuri *Ruuan muuttujia*, mikä liittää ne pelissä tiettyyn käyttökontekstiin. Haluan nostaa tämän esimerkin esiin osoittaakseni, että aineistosta löytyy myös tapauksia, joissa muuttujien esittelyä kommentoidaan ainoastaan yhdellä substantiivilausekkeella, eikä niiden tarkoitukseen paneuduta yhden sivulauseenkaan vertaa.

Kommentoinnin kohteena on tällaisissa tapauksissa muuttujan olemus muuttujana sekä se käyttöympäristö, jossa muuttuja esiintyy. Arvoa, joka kussakin esimerkissä muuttujalle annetaan (esimerkissä 20 *0*, *0* ja *false*, esimerkissä 21 *true* ja esimerkissä 22 *0* ja *0*) ei kommentoida millään tavalla, vaikka se on keskeinen ominaisuus muuttujalla. Kaikissa näissä esimerkeissä arvo annetaan koodissa jo kommenttia seuraavalla rivillä, joten sen toistaminen tällaisissa tapauksissa olisi sisällöllisesti melko hyödytöntä.

Aineistosta löytyvä toinen tapa kommentoida muuttujia ottaa toiminnallisemman lähtökohdan muuttujien esittelyyn. Tästä tavasta ovat esimerkit 23 ja 24, joiden koodeissa esitellään uudet muuttujat aivan kuten aiemmissakin esimerkeissä.

(23) // **Selvitetään rivien ja sarakkeiden lukumäärät.** (A18)
```
int rivlkm = mp.length;
int sarlkm = t[0].length;
```

(24) // **Haetaan tutkittavan merkin arvo.** (A19)
```
char tutkittava = hahmonpäänmerkki(hahmonsijaintitaulukko, syote, pelikenttä);
```

Esimerkkien 23 ja 24 kommenteista ei löydy sanaa *muuttuja*, mutta muuttujien nimiin viitataan kommenteissa (esimerkin 23 kommentissa sanotaan: -- *rivien ja sarakkeiden lukumäärät* ja koodissa on muuttujat nimeltään *rivilkm*, *sarlkm;* esimerkin 24 kommentissa sanotaan: *tutkittavan merkin arvo* ja koodissa on muuttuja nimeltä *tutkittava*). Muuttujien käyttötarkoituksen sijaan kommentissa kiinnitetään huomio siihen toimintaan, jonka lopputuloksena muuttujille saadaan arvot. Molemmat esimerkit alkavat passiivimuotoisella verbillä (*Selvitetään* ja *Haetaan*), ja näitä seuraa arvo, jota muuttujalle muuttujaa esittelevällä koodirivillä *selvitetään* tai *haetaan*.

Erona aiempien esimerkkien muuttujienesittelykommentteihin on siis lähestymistapa siihen, mikä kommentin mukaan on muuttujanesittelyrivillä tärkeää ja kommentoinnin arvoista. Myös koodi, jossa muuttujat esitellään, on hieman erityyppistä eri lähtökohdin kirjoitettujen kommenttien alla. Esimerkeissä 20–22, joissa kommentoidaan muuttujaa sen olemuksen ja käyttötarkoituksen

näkökulmasta, ovat muuttujille esittelylauseissa annetut arvot hyvin yksinkertaisia (*0*, *true* ja *false*). Tällaisten arvojen kommentti lienee melko tarpeetonta. Esimerkeissä 23 ja 24 taas arvo, joka muuttujalle annetaan, tapahtuu erillisen koodilauseen avulla (esimerkissä 23 selvitetään merkkijonon pituus *length*-operaatiolla ja esimerkissä 24 kutsutaan *hahmonpaamerkki*-metodia). Tämä ero koodissa voi selittää sitä, miksi muuttujiin ja niiden esittelyyn suhtaudutaan erilaisin tavoin kommentoidessa.

4.2 Kommentin vaikutuspiiri

Ohjelmointikoodin kommentoinnissa yleisesti tunnettu käytäntö on se, että kommentti liittyy koodissa seuraavaan riviin tai seuraaviin riveihin, mikä tarkoittaa sitä, että tekstissä ensin esiintyy kommentti ja sen jälkeen sen kommentoima koodi (esim. Laurikkala 2014a: 14). Tämä kommentointitapa ei ole Java-kielen tekniikan kannalta välttämätöntä, joten käytännössä kommentti voi sijaita koodiin nähden missä tahansa, kunhan se on erotettu kommenttimerkinnällä eli esimerkiksi //-merkillä varsinaisesta koodista. Kommentit ovat kuitenkin olemassa vain niitä lukevia ihmisiä varten, joten epäloogisessa ja vieraassa paikassa sijaitsevan kommentin on mahdollista aiheuttaa väärinymmärryksiä tai jäädä kokonaan irralliseksi ympäristöstään.

Aineistossa lähes kaikki kommentit ovat konkreettisen sijaintinsa ja merkityssisältönsä puolesta kytköksissä niiden jälkeen seuraavaan koodiin, ja tällaisesta tilanteesta ovat esimerkit 25–27. Kussakin näistä esimerkeistä kommentissa kuvataan suoraan juuri se toiminta, joka heti niiden jälkeen suoritetaan.

(25) **//Liikutetaan hahmoa.** (A8)
```
public static void liiku(int[][] hahmo, char[][] kentta, char syote) {
```

(26) **// Käydään läpi rivit yksi kerrallaan.** (A51)
```
for (int rivi = 0; rivi < kohde.length; rivi++)
  for (int sarake = 0; sarake < kohde[0].length; sarake++)
…
```

(27) **// Jos käyttäjä antaa l,r,u,d käskyn vaihdetaan hahmonsijaintitaulun arvoja sen mukaan miten hahmon pää käyttäytyy.** (A19)
```
if ((syote == 'l') | (syote == 'r') | (syote == 'u') | (syote == 'd')) {
…
```

Esimerkin 25 kommentissa kerrotaan, että liikutetaan hahmoa, ja sitä seuraavalla rivillä esitellään metodi, jonka nimi on *liiku*, ja jonka muodostamassa kokonaisuudessa hahmo liikkuu. Esimerkin 26 kommentissa kuvataan sitä heti seuraavan silmukan toimintaa, sillä for-silmukoiden avulla toteutetaan kommentin kuvaama rivien yksitellen läpikäyminen. Esimerkissä 27 on yhtä lailla selvää, että

kommentti käsittelee seuraavaa koodia, sillä kommentissa kuvataan tarkalleen koodin toteuttama toiminta.

Aineistosta ainoastaan yksi prosentti edustaa tavallisesta mallista poikkeavaa kommentointijärjestystä sillä tavalla, että kommentti ei ole omalla rivillään vaan jatkaa sitä koodiriviä, jota se kommentoi. Esimerkki 28 on siis lähinnä kuvaus tällaisesta poikkeuksesta, joka kuitenkin on teknisesti mahdollinen ja koodin lukijan ymmärrettävissä. Tässä esimerkissä muuttujan nimi *paax* osoittaa, että kommentti liittyy juuri tähän koodin osaan, sillä kommentissa mainitaan *paan x-koordinaatti*, josta muuttujan nimi on selvä lyhenne, ja seuraavalla rivillä vuorossa on *y*.

(28)
```
int paax = hahmo[0][0]; // paan x-koordinaatti (A3)
int paay = hahmo[0][1]; // paan y-koordinaatti
```

Kommentin ja koodin esiintymisjärjestyksen voi todeta olevan aineistossa pientä määrää poikkeuksia lukuun ottamatta vakituinen. Tämä on keskeinen osa kommentin ja koodin suhdetta, sillä tekstissä osien järjestyksellä on oma painoarvonsa.

4.2.1 Kommentin kattama koodimäärä

Vakiintuneen esiintymisjärjestyksen lisäksi kommentin vaikutusaluetta voi tarkentaa esittämällä, kuinka suureen osaan koodia yksi kommentti voi vaikuttaa, sillä kommentti voi kattaa erimittaisia osia koodista. Aineistossa on kolme erilaista tyyppiä kommentin vaikutusalan koolle: (1.) kommentti voi kattaa kokonaisen metodin mittaisen alan, (2.) kommentti voi kattaa yhden tiiviin rivikokonaisuuden tai (3.) kommentti voi kattaa ainoastaan yhden rivin.

Suurimmassa osassa (83 %) aineiston metodeja on kommentti, joka esittelee metodin toiminnan ja näin ollen kattaa kokonaisen metodin verran koodia. Tällaisia metodinesittelykommentteja ovat esimerkit 29 ja 30. Metodinesittelykommentti sijaitsee aina juuri ennen metodinesittelylausetta, jolloin se jo sijaintinsa puolesta liittyy metodiin. Kommenttien semanttinen sisältö osoittaa myös sen, että kommentissa ei ainoastaan kerrota, mitä seuraavalla rivillä tapahtuu, vaan sisältö liittyy kaikkeen koodiin, jota metodi sisällään pitää.

(29)
```
// metodi liikkumiselle, palauttaa totuusarvon joka kertoo halutaanko
// lopettaa (A40)
public static boolean liikkuminen(int[][] hahmo, char suunta, int leveys,
int korkeus) {
```

(30)
```
/* Annetaan hahmolle liikkumisohje valintasyötteen mukaan. Hahmon paikkatiedot
 * päivitetään valintaa vastaavaksi. Hahmon peruuttaminen ei ole sallittua./
 * Lopuksi palautetaan päivitetyt paikkatiedot main-metodiin./
```

```
 */ (A15)
public static int[][] liikutaHahmoa(int[][] paikat, char valinta, char[][] kentta) {
```

Esimerkissä 29 osoitetaan jo mainitsemalla sana *metodi* kommentin alussa, että kommentti koskee metodia. Tässä kommentissa kerrotaan, mitä metodi tekee (*metodi liikkumiselle*) ja sen, millaisen lopputuloksen koko siihen sisältyvä koodi saa aikaan (*palauttaa totuusarvon*). Esimerkin 30 kommentti kuvaa hieman edellistä esimerkkiä tarkemmin metodin toimintaa, ja siinä rakennetaan metodin sisältämän koodin järjestyksestä ja kulusta pienimuotoinen tarina sidoskeinojen, kuten toiston (*valintasyöte, valinta*; *hahmolle, hahmon, hahmon*) ja *lopuksi*-sanan avulla. Yhtä kaikki molemmissa kommenteissa kommentoidaan koko metodia, jolloin kommentin vaikutusala on metodin verran koodia.

Selkeimmät tapaukset, joissa kommentti kattaa usean toisiinsa liittyvän rivin verran koodia, muttei kuitenkaan kokonaisen metodin verran, ovat sellaiset, joissa koodi sisältyy johonkin ehtorakenteen tai toistorakenteen muodostamaan kokonaisuuteen[7]. Esimerkin 31 kommentti kattaa neljän rivin mittaisen for-toistorakenteen: ensin sidotaan kommentti liittymään juuri for-silmukkaan (*Silmukka jossa*), minkä jälkeen kuvataan toimintoa, joka silmukan sisällä tapahtuu. Tämä yksinkertainen ja lyhyt esimerkki osoittaa sen, minkätyyppisiin kooditapauksiin viittaan selkeillä ehto- tai toistorakenteen muodostamilla kokonaisuuksilla, sillä kokonaisuudet ovat usein monimutkaisempia. Tällöinkin kommentti liittyy usein koko rakenteen koodiin, kuten esimerkissä 32. Tämän esimerkin kommentti ei sisällä kuvausta koodin toiminnasta, mutta siinä kerrotaan, että ehtorakenteen sisältämä koodi tapahtuu, mikäli *komentona on siirto ylös*. Kommentti siis pätee jokaisen ehtorakenteen sisällä olevaan riviin, sillä jokaisen kohdalle voisi ajatella oman kommenttinsa *Komentona siirto ylös*.

```
(31) // Silmukka jossa hahmon vartalo liikutetaan. (A25)
     for(int i = 1; i < hahmo.length; i++) {
         hahmo[hahmo.length - i][0] = hahmo[hahmo.length - (i+1)][0];
         hahmo[hahmo.length - i][1] = hahmo[hahmo.length - (i+1)][1];
     }
```

[7] Valintarakenteet ja toistorakenteet ovat koodissa sellaisia osia, jotka voivat pitää sisällään toisia valinta- tai toistorakenteita. Ne ovat ikään kuin otsikoita, jotka kietovat sisäänsä muuta koodia:

```
toistorakenne {
                koodia
                valintarakenne {
                                koodia
                }
                koodia
}
```

```
(32) // Komentona siirto ylös. (A33)
     if(komento == YLOS) {
        if(paaRivInd == 0)
          paaRivInd = kentta.length -1;
        else
          paaRivInd = paaRivInd - 1;
     }
```

Koodikokonaisuus, jonka kommentti kattaa, ei välttämättä ole yksinkertaisesti jonkun rakenteen muodostama sisäkkäinen osa, vaan kommentti voi kattaa muilla keinoilla pääteltävissä olevan osan koodia. Yksi selkeä keino rajata kommentin kattama kokonaisuus on tyhjien ja koodin kannalta tyhjänpäiväisten rivivälien lisääminen ennen ja jälkeen rajatun kokonaisuuden. Esimerkissä 33 on juuri niin, että ennen esimerkissä esiintyvää kommentin ja koodin kokonaisuutta on tyhjä rivi, kuten myös sen jälkeen. Esimerkissä 34 tyhjä riviväli on ainoastaan kokonaisuuden jälkeen, mutta koska kommentilla on tapana liittyä sitä seuraavaan koodiin, voi ennen kokonaisuutta sijaitsevaa riviväliä pitää lähinnä esteettisenä seikkana.

```
(33) // Tarkastellaan siirtyikö hahmo kentän reunojen ylitse
     // ja annetaan tarvittaessa koordinaatti kentän toiselta puolelta (A45)
     if(hahmoy >= pelikentta.length
       hahmoy = 0;
     else if(hahmox >= pelikentta[0].length)
       hahmox = 0;
     else if(hahmoy < 0)
       hahmoy = pelikentta.length - 1;
     else if(hahmox < 0)
       hahmox = pelikentta[0].length - 1;
```

```
(34) // Pyyhitään hahmon viimeinen merkki pois, ettei se jää kentälle. (A20)
     int x, y;
     x = hahmo[hahmo.length - 1][0];
     y = hahmo[hahmo.length - 1][1];
     kentta[x][y] = ' ';
```

Ainoastaan rivivälien perusteella ei tietenkään voi päätellä, millaisen alan kommentti kattaa, joten painoarvoa on annettava myös kommentin sisällölle. Esimerkin 33 kommentin alussa kuvattu toiminta (*Tarkastellaan siirtyikö hahmo kentän reunojen yli*) tehdään sitä seuraavilla koodiriveillä neljä kertaa jokaisen ehtorakenteen avulla. Samaten kommentin loppuosa (*annetaan tarvittaessa koordinaatti kentän toiselta puolelta*) tehdään jokaisen neljän ehtorakenteen sisällä. Kommentin voi siis ajatella toistuvan jokaisen rakenteen kohdalla uudelleen, mutta koska se olisi sisällöltään samanlainen kussakin kohdassa, on kommentoija kirjoittanut sen ainoastaan kerran ja rajannut välilyönnein tämän kohdan yhdeksi kokonaisuudeksi.

Esimerkissä 34 taas kommentin sisältöosat liittyvät jollakin tavalla johonkin osaan sitä seuraavaa koodia. *Pyyhitään* liittyy kokonaisuuden viimeiseen kommenttiriviin *kentta[x][y] = ' ';*, jossa pyyhitään yksi osa eli sijoitetaan siihen välilyönti. *hahmon viimeinen merkki* taas viittaa kahteen aiempaan riviin (*x = hahmo[hahmo.length - 1][0]; y = hahmo[hahmo.length - 1][1];*), joissa etsitään hahmon viimeinen merkki. Ensimmäinen rivi on ainoa, jota kommentissa ei varsinaisesti mainita, mutta koska sen jälkeen tapahtuvat asiat kuvataan kommentissa, on senkin katsottava kuuluvan kommentin kuvaamaan kokonaisuuteen.

Oman selvärajaisen ryhmänsä muodostavat sellaiset kommentit, jotka kattavat ainoastaan yhden rivin verran koodia. Yksi kommentti yhtä riviä kommentoimassa toteutuu useimmiten sellaisissa tapauksissa, joissa esitellään muuttuja. Tällaisissa kommenteissa kuvataan usein, mikä on esiteltävän muuttujan rooli. Esimerkissä 35 esitellään muuttuja *n*, jonka kerrotaan lyhyessä kommentissa olevan laskuri. Esimerkissä 36 taas esitellään vakiomuotoinen muuttuja *HAHMONPAARIVI*, ja kommentissa kuvataan sen tehtävä koodin kannalta (*Avustava vakio*) ja arvo, joka sille annetaan (*Hahmon pään rivi taulussa*). Kummassakaan tapauksessa ei ole epäselvää, jatkuuko kommentin kattama alue pidemmälle, sillä seuraavilla riveillä siirrytään uuteen toimintoon. Esimerkki 37 on samankaltainen tapaus, sillä myös siinä esitellään muuttujia ja kommentti liittyy ainoastaan esittelyihin, mutta esittelyrivejä on yhden sijasta kaksi. Tällaisiakin tapauksia löytyy aineistosta niin useita, että tahdon nostaa myös sen esiin yhdenkaltaisena variaationa yhden rivin kattavasta kommentista.

(35) **//n on laskuri** (A59)
```
int n = 0;
```

(36) **//Avustava vakio. Hahmon pään rivi taulussa** (A10)
```
final int HAHMONPAARIVI = hahmotaulu.length - 1;
```

(37) **// Alustetaan liikkumisille muuttujat.** (A25)
```
int riviMuutos = 0;
int sarakeMuutos = 0;
```

Yhden rivin kattavat kommentit eivät liity ainoastaan muuttujien esittelyn yhteyteen, sillä aineistosta löytyy useista muistakin yhteyksistä yhteen ainoaan koodiriviin liittyviä kommentteja. Muuttujien esittelyn kaltainen on esimerkin 38 kommentti, sillä siinä kerrotaan taulukosta, joka sitä seuraavalla koodirivillä luodaan. Tilanne muistuttaa paljon esimerkin 36 kommenttia, sillä myös tässä taulukon luontia kuvaavassa kommentissa kerrotaan ensin taulukon tehtävä koodin kannalta (*Avustava taulukko*) ja sen jälkeen arvot, joita taulukkoon tallennetaan tulevassa koodissa (*-- tallennetaan hahmonPaikat-taulukon //sen hetkiset arvot*). Taulukko on rakenteena pelkkää muuttujaa

monimutkaisempi, joten yksin siihen liittyvä kommentti voi useammin olla tarpeellinen kuin pelkkään muuttujaan liittyvä.

(38) **//Avustava taulukko, johon tallennetaan hahmonPaikat-taulukon //sen hetkiset arvot**. (A58)
```
int[][]apuTaulukko = new int [rivilkm][sarakelkm];
```

Esimerkissä 39 esiintyy aineistossa tyypilliseksi osoittautunut yhden rivin kattava kommentin ja koodin yhdistelmä. Siinä kommentoidaan riviä, jolla kutsutaan jotakin metodia eli siirrytään koodissa hetkeksi toiseen paikkaan. Tällaiset metodikutsua edeltävät kommentit muistuttavat jonkin verran kommentteja, jotka edeltävät kokonaista metodia (ks. esimerkit 29 ja 30), sillä näissäkin kuvataan koko kutsuttavan metodin toimintaa. Esimerkissä 39 tehdään ensin selväksi, että kyseessä on juuri metodin kutsuminen, ja sen jälkeen kuvataan lyhyellä sivulauseella, mitä kutsuttava metodi tekee.

(39) // **Kutsutaan metodia, joka "kääntää hahmon".** (A32)
```
paikkaTaulukko = kaannaHahmo(paikkaTaulukko, paikkaTaulukkoKopio);
```

Koska koodin kommentoinnin tulisi olla ytimekästä ja oleellista, voisi yhtenä mahdollisena syynä näille yhden rivin kattaville kommenteille pitää sitä, että sen yhden rivin koodi on kokonaisuuden kannalta tärkeää. Kommentin tehtävänä on usein selventää koodista sellaista osaa, jonka ymmärtäminen voisi olla hankalaa ilman luonnollisella kielellä kirjoitettua selvennystä, ja muuttujat ovat juuri se osa koodista, jonka koodin kirjoittaja voi nimetä kuten haluaa.

Näiden kolmen erilaisen ryhmän perusteella voi todeta, että aineistosta ei ole mahdollista löytää selkeää numeerista arvoa sille, kuinka monta riviä koodia kommentti tyypillisesti kattaa. Kommentti voi liittyä jokaiseen riviin koko koodissa tai metodissa, tai se voi liittyä vain yhteen ainoaan riviin. Kaikissa tapauksissa kommentilla on aina se selkeä koodi, jota se käsittelee, oli se sitten yhden tai tuhannen rivin mittainen.

4.2.2 Kommenttien hierarkkisuus

Javassa ohjausrakenteita eli ehto- ja toistorakenteita voi olla toistensa sisällä niin, että rakenteesta tulee looginen ja hierarkkinen rakennelma. Sisäkkäisyyden luomaa monimutkaisuutta helpottaa se, että peräkkäisesti ja loogisesti yhteenkuuluvat kokonaisuudet sisennetään (Laurikkala 2014a: 15). Kuten kommentitkin, sisentäminen on ainoastaan koodia lukevaa ihmistä varten tehty visuaalinen

apukeino. Esimerkin 40 avulla on mahdollista havaita, miltä koodin ohjausrakenteiden sisäkkäisyys käytännössä näyttää: sisennetty osa on aina alisteinen sille osalle, joka on yhden asteen verran vähemmän sisennetty, ja näin koodista kommentteineen tulee porrastetun näköistä.

(40)
```
// Sijoitetaan uudet hahmon paikat taulukkoon. (A19)
    for (int rivi = hahmonpituus; rivi > 0; rivi--) {
        for (int sarake = 0; sarake < hahmonsijaintitaulukko[0].length; sarake++) {
            if (rivi > 0) {
                hahmonsijaintitaulukko[rivi][sarake] = hahmonsijaintitaulukko[rivi - 1][sarake];

                // Tutkitaan syötiinkö ruokaa.
                if (tutkittava == '+') {
                    // Lisätään hahmoon palanen, jos ruokaa syötiin.
                    hahmonsijaintitaulukko[hahmonpituus][0] = hahmonsijaintitaulukko[hahmonpituus][0];
                    hahmonsijaintitaulukko[hahmonpituus][1] = hahmonsijaintitaulukko[hahmonpituus][1];
                }
                …
```

Koodin ollessa sisäkkäistä ja tietyllä tavalla hierarkkista näyttävät myös siihen liittyvät kommentit muodostavan aineistossa samanlaisen rakenteen. Ylemmän tason kommentti koskettaa kaikkia alemman tason kommentteja, sillä ylemmän tason kommentti liittyy jollakin tavalla koko sen alapuolella sijaitsevaan koodiin. Näin on esimerkissä 40, jossa on yhteensä neljä edeltäjälleen alisteista ohjausrakennetta (*for*-, *for*-, *if*- ja *if*-alkuiset rivit). Kommentit sijaitsevat kolmella eri tasolla, minkä havaitsee helpoimmin sisennysten luoman rakenteen avulla. Ensimmäinen kommentti kuvaa koko ensimmäisen ohjausrakenteen toiminnan tarkoitusta ja lopputulosta, ja koodipätkän lopussa tehdään ylimmän tason kommentin kuvaama toiminto (*Sijoitetaan uudet hahmon paikat taulukkoon*). Tämä esimerkki on vain pätkä yhden koodin ohjausrakenteen monipolvista kokonaisuutta, joten lopullinen sijoitus ei ole tässä esimerkissä mukana.

Seuraava kommentti liittyy viimeisen tason ehtorakenteeseen. *Tutkitaan syötiinkö ruokaa* liittyy ylemmän tason kommenttiin siten, että hahmon uuden paikan määrittelyssä täytyy tietää, kasvoiko hahmon pituus. Seuraava kommentti on kyseisen ohjausrakenteen sisällä ja liittyy tiiviisti edeltävään ja yhtä tasoa ylempänä sijaitsevaan kommenttiin. Teemankulun näkökulmasta lauseita pitää aina tarkastella tekstiyhteydessään (Shore 2008: 40), mikä kommenttien näkökulmassa tarkoittaa sitä, että korkeamman hierarkiatason kommentit on huomioitava kommenttia luettaessa. *Lisätään hahmoon palanen, jos ruokaa syötiin* on selvästi tietoinen sitä yhtä tasoa ylempänä sijaitsevasta kommentista, sillä sivulauseen *jos ruokaa syötiin* teemapaikalla on *ruokaa*. Teeman tehtävänä on sitoa tekstiä aiemmin sanottuun (Shore 2008: 40), minkä teemapaikan *ruokaa* tässä yhteydessä tekee.

Esimekki 40 on suoraviivainen ja kommentteja on ainoastaan kolme kappaletta, mutta aineistosta löytyy myös useita esimerkki 41:n kaltaisia reilusti monimutkaisempia rakenteita. Tässä esimerkissä jokaista ohjausrakennetta on kommentoitu, joten kommenttien sisäkkäisyys tulee ilmi selkeästi.

```
(41) //Loopataan kopioitavan taulun jokainen rivi läpi toisesta rivistä lähtien  (A10)
     for (int rivi = 1; rivi < hahmotaulu.length; rivi++) {
         //Loopataan rivin kaikki sarakkeet läpi
         for (int sarake = 0; sarake < hahmotaulu[0].length; sarake++){

             //Hahmon pään rivin rivikoordinaatti. Suunta alas ("d")
             if (rivi == HAHMONPAARIVI && sarake == 0 && suunta == 'd'){
                 //Pään rivikoordinaatti alimmaisella rivillä
                 if (PAANRIVI == KENTANRIVIT - 1){
                     //Päänrivikoordinaatti siirtyy ensimmäiselle riville
                     aputaulu[rivi][sarake] = 0;
                 }
                 //Pään rivi ei reunassa
                 else {
                     //Pään rivikoordinaatti siirtyy yhden rivin alaspäin
                     aputaulu[rivi][sarake] = PAANRIVI + 1;
                 }
                 //Kaulan sarake ja rivi sama kuin vanha pään koordinaatti
                 aputaulu[rivi - 1][sarake] = hahmotaulu[rivi][sarake];
             }

             …
```

Aineistossa kommenttien vaikutuspiiri koodin sisällä on monipolvista, ja kommentin vaikutuspiirissä voi olla toisia kommentteja, joihin ylemmällä tasolla sijaitseva kommentti myös koskettaa. Jokaista sisäkkäistä ohjausrakennetta ei toki välttämättä kommentoida kuten esimerkissä 41, mutta jos koodista löytyy sisäkkäisyyttä ja limittäisyyttä, löytyy sitä myös kommenteista. Se voi olla yksinkertaista ja vain muutamassa kohdassa ilmenevää, kuten esimerkissä 40 tai toistuvaa ja koodin tavoin tiheää kuten esimerkissä 41.

Se, että kommentit mukailevat koodin rakennetta aina merkitysten sisäkkäisyyden ja typografisten sisennysten osalta, vahvistaa näkemystä siitä, että kommentit ovat riippuvaisia koodista. Kommentit eivät sijaitse irrallaan ja erillään ylimmällä tasolla, vaan osoittavat tarkasti tiettyyn osaan koodia mukaillen koodin sijaintia ja jäsentelyä.

4.3 Metodin kokonaisrakenne

Hasanin mukaan tekstin kokonaisrakenne määrittää sen genren, eli toisiaan rakenteellisesti muistuttavat tekstit edustavat samaa genreä (Ventola 2006: 97). Tämän takia nostan tässä luvussa lyhyen

pohdinnan alle sen, millainen on ohjelmointikoodin metodin kommenttien luoma kokonaisuus, ja voiko niiden kokonaisrakenteeseen perehtyä tarkemmin.

Metodin kokonaisrakenne on melko ongelmallinen tutkittava kielellisistä lähtökohdista, sillä kommenttien kielellisellä sisällöllä ei ole juuri mitään vaikutusta siihen, missä järjestyksessä kommentit metodissa esiintyvät. Käsittelin luvussa 2.1.1 tekstiä ja sitä, miten kommentit on tulkittavissa teksteiksi – kommenttien kokonaisuuksina vai yksittäisinä kommentteina. Metodin kokonaisrakenne kertoo yleisemmällä tasolla koko koodin läpi esiintyvien kommenttien jatkuvuudesta tai jatkumattomuudesta, mutta syyt jatkuvuudelle tai sen puutteet eivät johdu kommenttien kielellisestä olemuksesta.

Kommentit ovat sikäli alisteisia koodille, että niiden paikat ja sisällöt määräytyvät koodin mukaan: jos esimerkiksi jonkin koodinpätkän paikka vaihtuu tekstitiedostossa, siihen liittyvän kommentin on siirryttävä mukana. Tämä tarkoittaa sitä, että kommenttien järjestys koodin sisällä on määrätty koodin mukaan eikä kommenttien luoman kokonaisuuden mukaan. Hasanin rakennepotentiaalissa voidaan huomioida osien järjestys (Halliday & Hasan 1985: 64), mikä ei siis kommenttien osalta ole kytköksissä kommenttien omaan tekstilajiin.

Kun esimerkiksi uutisen rakenne on melko vakiintunut niin, että alussa on tärkein tieto ja loppua kohti siirrytään yksityiskohtiin (ks. esim. Valtonen 2012: 65), ei kommenttien muodostamassa kokonaisuudessa ole samanlaisia rakenteen lainalaisuuksia nimettävissä ainakaan itse kommenttien takia. Rakenteellisia johdonmukaisuuksia löytyy kyllä, mutta niiden syynä ei ole se, että kommenttien olisi esiinnyttävä tietyssä järjestyksessä, vaan se, että koodirivien järjestys on merkityksellinen. Kommenttien järjestys metodissa suhteessa toisiinsa on mahdollista osoittaa yleispiirteittäin: Aineistossa metodinesittelykommentti sijaitsee jokaisessa esiintymässään ennen koodiriviä, jolla esitellään metodi, ja lähes kaikki muuttujienesittelykommentit sijaitsevat heti metodinesittelyrivin jälkeen. Metodin viimeisillä riveillä sijaitsee usein kommentti, jossa kerrotaan paluuarvoista. Kommenttien sijaan koodi on kokonaisrakenteen tapauksessa se, joka määrää kokonaisrakenteen osien järjestyksen. Kommentit eivät sijaitse missään omista lähtökohdistaan johtuen, vaan syynä rakenteen johdonmukaisuuksille on ainoastaan koodi.

Kommentit muodostavat rakenteellisen suhteen toisiinsa ainoastaan niiden kommentoiman koodin avulla, ja ne ovat täysin riippuvaisia koodin määrätystä järjestyksestä. Tämän kommentin ja koodin rakenteellisen riippuvuuden voi ajatella yhdeksi tärkeäksi kommentin ja koodin suhteen rakentajaksi. Ne sijaitsevat kiinteässä yhteydessä toisiinsa, mikä on elinehto niiden yhteydelle ja ylipäätään suhteen olemassaololle. Jätän tutkimuksessani metodin kokonaisrakenteen yksityiskohtaisen erittelyn huomiotta, sillä kokonaisrakenteen kielellisillä piirteillä ja rakenteiden järjestyksellä

ei ole yhteyttä kielellisiin piirteisiin. Metodin kokonaisrakenteen kieliopillisia ja leksikaalisia koheesiokeinoja esittelen tarkemmin luvussa 5.3.

5 Kommenttien piirteet metafunktioittain

Tässä luvussa analysoin kommentteja ja niitä koodiin sitovia piirteitä Hallidayn metafunktioiden avulla (ks. Halliday & Hasan 1985: 44–45). Metafunktioilla tarkoitetaan kielen funktionaalisia perustehtäviä, jotka muodostavat perustan koko kielisysteemille (Luukka 2002: 102; ks. tarkemmin luku 2.2.3, jonka aiheena on systeemis-funktionaalinen teoria). Olen päätynyt metafunktioiden varaan rakentuvaan tekstianalyysiin, koska Halliday itse on perustanut tutkimuksensa lähinnä metafunktioiden varaan ja koska suomalaisessa systeemis-funktionaaliseen teoriaan pohjautuvassa tekstintutkimuksessa tekstiaineistoa on yleensä analysoitu metafunktioittain. Hallidayn metafunktiot ovat myös melko vakiintuneita, ja systeemis-funktionaalinen teoria tarjoaa monenlaisia työkaluja tekstin analyysiin. (Shore 2012a: 156–157). Tekstilajia määriteltäessä tärkeässä roolissa ovat kielelliset keinot, joilla tekstimaailmaa ja tekstin sisäisiä suhteita rakennetaan, ja näitä keinoja voi löytää analysoimalla kieltä (ks. Shore 2012b: 162). Otan tutkimuksessa huomioon kaikki kolme metafunktiota, sillä metafunktiot ovat kytköksissä toisiinsa ja kaikista teksteistä löytyy ideationaalisia, interpersoonaisia ja tekstuaalisia merkityksiä (Halliday 1978: 112).

Aloitan analyysin tutkimalla kommenttien sosiaalisia rooleja ja niihin rakennettuja suhtautumistapoja, jotka kuuluvat Hallidayn metafunktioiden osalta interpersoonaisen metafunktion piiriin (Halliday & Hasan 1985: 20, 45). Kiinnitän huomion siihen, miten koodin tekijyys heijastuu kommentteihin ja millaisia rooleja kommentteihin liittyy, sillä yksi kommenttien erityinen piirre on se, että osallistujia voivat olla tietokone, koodaaja, pelihahmo tai pelaaja. Tämän osalta ohjelmointikoodin kommenteissa suureen rooliin nousevat passiivi ja passiivin luoma implisiittinen tekijyys. Passiivin käyttö on erittäin keskeinen piirre kommenttien tekstilajia määriteltäessä, joten nostan sen ensimmäisenä tarkemman analyysin kohteeksi. Pienemmässä roolissa esiin nousevat myös kommenttien kielen modaaliset valinnat ja sanavalintojen ilmentävät suhtautumistavat.

Seuraavan luvun lähtökohtana on se, että ohjelmointikoodin kommentit ovat osaltaan rakentamassa koodi- ja ohjelmointimaailmaa. Tämän voi luokitella kuuluvaksi ideationaalisen metafunktion piiriin, sillä ideationaalisella metafunktiolla tarkoitetaan kielen keinoja konstruoida ympäröivää maailmaa (Halliday & Hasan 1985: 45). Tekstilajin kannalta esiin nousee se, millaisen kuvan kommentit antavat kommentoimastaan koodista, sillä kommentit kykenevät asettamaan joko peli- tai koodimaailman lukijan silmissä ensisijaiseksi. Kiinnitän huomion myös leksikaalisten valintojen kautta syntyvään diskurssiyhteisöön, sillä diskurssiyhteisö hyödyntää ja käyttää jäsentensä tunnistamia tekstilajeja (Swales 1990: 24–25, 28).

Tämän luvun viimeisessä alaluvussa otan huomioon ne keinot, joilla kommenttien luomaa tekstikokonaisuutta pidetään koossa. Kiinnitän huomion erilaisiin koheesiokeinoihin, joiden avulla voidaan rakentaa kuvaa siitä, millä tavalla kommentit ovat teksti. Määritän luvun tekstuaalisen metafunktion piiriin kuuluvaksi, sillä tekstuaalisella metafunktiolla tarkoitetaan niitä keinoja, joilla tekstiä pidetään koossa, mikä käytännössä tarkoittaa tekstin koheesiokeinoja ja erilaisia teeman- ja informaationkulkuja (Halliday & Hasan 1987: 45).

Shore (2012b: 158) on analysoinut artikkelissaan erästä tekstiä ja sen tekstimerkityksiä juuri metafunktioiden avulla. Käytän tätä artikkelia löyhänä ohjenuoranani siihen, miten eri metafunktioita voi tekstiä analysoitaessa hyödyntää, sillä Shore vie rinnakkain metafunktioiden käytäntöä ja teoriaa, jolloin analyysin soveltaminen tämän tutkimuksen tarpeisiin on mahdollista.

5.1 Interpersoonainen metafunktio

Interpersoonaisen metafunktion kautta puhuja asettaa itsensä tilannekontekstiin ilmaisemalla omia asenteitaan ja arvioitaan ja pyrkimällä vaikuttamaan kuulijoihinsa (Halliday 1978: 112). Kielen avulla rakennetaan vuorovaikutuksellisia rooleja ja ilmennetään asenteita, tunteita ja arviointeja, ja interpersoonaisessa metafunktiossa kiinnitetään yleensä huomiota suhtautumista heijastaviin sanavalintoihin, persoonaan ja modaaliaineksiin (Luukka 2002: 103, Shore 2012b: 177). Interpersoonainen metafunktio heijastaa tekstin osallistujarooleja, joilla tarkoitetaan yleisesti sitä, ketkä ovat osallisina kielenkäyttötilanteessa: millaiset heidän roolinsa ovat tilanteessa ja millaisia asenteita he ilmaisevat (Halliday & Hasan 1985: 25, Halliday 1985: 33).

Nostan tässä luvussa esille ne merkittävimmät interpersoonaiset merkitykset, joita ohjelmointikoodien kommenteissa rakennetaan. Tekijyyden ja erilaisten roolien kannalta merkittävin piirre kommenteissa on passiivi, joka ilmentää kommenteissa erilaisia implisiittisiä tekijöitä. Modaalisten valintojen avulla osoitan, minkälaisiin asioihin kommenteissa voidaan liittää modaalisuutta, ja suhtautumista ilmentävien sanavalintojen avulla luon kuvan vastakkaisista suhtautumistavoista, joita kommentteihin sanavalintojen avulla rakennetaan.

5.1.1 Passiivi ja tekijyys

Yksi interpersoonaisen metafunktion piiriin kuuluva kieliopillinen piirre on tekstin persoonamuoto (Halliday 1985: 87). Shore (2012b: 178) kiinnittääkin omassa systeemis-funktionaalisessa analyysissaan huomion siihen, mikä on esimerkkitekstin yleisin persoonamuoto. Läpi kommenteista koostuvan aineiston selkeästi yleisin predikaattiverbin persoonamuoto on passiivi, joten nostan täs-

sä luvussa esiin ohjelmointikoodin kommenttien passiivimuotoisuuden välittämät interpersoonaiset merkitykset.

Yksinkertaisesti ajateltuna passiivimuotoisesta lauseesta puuttuu subjekti ja sen avulla on mahdollista kuvata asioita ottamatta kantaa tekijän identiteettiin, jolloin tekijä jätetään tarkoituksella taka-alalle (VISK § 1313, 1325). Yksi suomen kielen passiivin keskeinen ominaisuus onkin tekijän piilottaminen. Monissa kielissä passiivimuotoisessa lauseessa voi olla tunnettu ja lauseessa mainittu tekijä, mutta suomen kielessä passiivin tekijä jää implisiitiksi. (Kuiri 2000.)

Suomen kielessä on useita passiiviluontoisia rakenteita, joista yksipersoonainen passiivilause on yleisin ja monikäyttöisin. Siinä ilmisubjektia ei ole ja finiittiverbi on passiivimuotoinen, eli se ei vaihtele eri persoonissa. (VISK § 1313.) Tällaisessa passiivilauseessa teemapaikalle sijoittuu subjektin sijaan joku toinen jäsen (VISK § 1315). Yksipersoonainen passiivilause on esimerkiksi seuraava: *"Täällä huolehditaan työntekijöiden hyvinvoinnista."* Muut passiivirakenteet ovat monipersoonaisia, jolloin finiittiverbi taipuu persoonissa (VISK § 1332). Monipersoonainen passiivilause on esimerkiksi seuraava: *"Virtanen tuli erotetuksi."* Puhekielessä passiivimuotoa käytetään monikon ensimmäisessä persoonassa, jolloin tekijät ovat tunnetut, mutta syntaktisesti nämä käytöt eivät kuulu passiiviin aktiivitaivutuksellisuutensa ja me-ilmisubjektinsa vuoksi (Kuiri 2000, VISK § 1315).

Passivimuodon takana olevat tekijät voi joissakin tilanteissa tunnistaa paremmin kuin toisissa, mutta tekijöitä ei koskaan nimetä suoraan (Kuiri 2000). Monesti on kuitenkin pääteltävissä, keneksi tai keiksi passiivin tekijät ajatellaan, ja kuuluuko kirjoittaja itse näiden tekijöiden joukkoon (Alho & Kauppinen 2009: 130). Esimerkiksi lauseesta *”Työpaikallani pidetään kahvitauko kolmelta”* voi päätellä, että myös puhuja pitää kahvitauon kolmelta, kun taas lauseesta *”Työpaikallani käydään YT-neuvotteluita”* voi ymmärtää, ettei puhuja ole itse neuvottelemassa. Passiivilla puhuja voi siis viitata myös itseensä (Kuiri 2000). Passiivilauseen implisiittinen subjekti on ensisijaisesti monikollinen ja ihmistarkoitteinen (VISK § 1315).

Käsittelen tässä luvussa yksipersoonaisia passiiveja, joita aineiston persoonamuodot pääasiassa ovat (VISK § 1313). Osoitan ensin yleisesti, miten passiivimuotoa käytetään aineistossa, minkä jälkeen esittelen ne erilaiset implisiittiset tekijät, joita kommenttien passiiveihin sisältyy.

Passiivin käyttö on aineistossa sikäli odotustenmukaista, että kurssin opettaja Laurikkala (2014c: 5–8) käyttää järjestelmällisesti passiivia opettaessaan kommentoimaan luentomateriaaleissaan ja myös kommentoidessaan esimerkkikoodia. Kyseessä ei kuitenkaan voi ajatella olevan ainoastaan Laurikkalan kommentointitapa, jonka kaikki aineiston opiskelijat olisivat omaksuneet, sillä myös Wikla käyttää passiivia kommentoidessaan esimerkkikoodia (ks. esim. Wikla 2003: 94, 176, 241) esimerkiksi seuraavalla tavalla:

```
// luetaan arvot:
[koodia]
// kerrotaan alkiot 7:llä:
[koodia]
// tulostetaan matriisi:
[koodia]
```

Selkein syy passiivin käytölle ohjelmointikoodin kommenteissa lienee se, ettei huomiota haluta kiinnittää tekijään vaan toimintoon, sillä kommenttien kuvaamat tilanteet ovat usein toimintaan, ei toimijaan, liittyviä. Passiivimuodolla pyritäänkin usein hämärtämään lauseen tekijää (Kuiri 2000). Tekstin tekijä eli koodin kirjoittaja ei nosta tekijyyttään esille, vaan keskeiseen asemaan asettuu se koodin aiheuttama toiminta, jonka hän on kirjoittamalla saanut aikaiseksi.

Neljän eri opiskelijan koodeista poimitut esimerkit 42–45 ovat tyypillisiä tapauksia tällaisesta passiivimuotoisesta kommentista, jossa korostetaan toimintaa. Passiivimuotoinen predikaattiverbi on jokaisessa esimerkissä teemapaikalla. Teemapaikalle sijoittuu lauseen ensisijainen aihe (VISK § 1370). Passiivin yhteydessä oleellista on se, että kussakin esimerkissä teemapaikalla on passiivimuotoinen predikaatti eli lauseen ensisijaisena sisältönä on toiminta, jota kuvataan koodissa tapahtuvaksi. Objektiteemaisella passiivilauseella asia esitetään objektitarkoitteen näkökulmasta, eli esimerkiksi lauseessa "Työntekijä erotettiin tehtävistään" asia esitetään työntekijän näkökulmasta (VISK § 1325). Kun kommenteissa teemapaikalla on usein verbi, voidaan ajatella, että lauseen sisältö halutaan ilmaista toiminnan näkökulmasta.

(42) **//talletetaan aiempi sijainti** (A44)
```
aiempiX = hahmoSijainti[0][0];
aiempiY = hahmoSijainti[1][0];
```

(43) **//päivitetään hahmon sijainti** (A58)
```
hahmonPaikat[0][1] = hahmonPaikat[0][1]+1;
```

(44) **// Liikutellaan hahmoa komentojen mukaan.** (A61)
```
switch (komento) {
    case OIKEA:
---
```

(45) **// Kopioidaan kaikkien muiden hahmon palojen paitsi pään sijainti** (A6)
```
for (int laskenta = hantaKoordinaattirivi; laskenta > 0; laskenta--) {
---
```

Esimerkin 42 kommentin teemapakailla on passiivimuotoinen verbi *talletetaan*. Kommentin keskeinen sisältö on se, että kyseessä on talletus ja että talletuksella on jokin kohde, eikä se, kuka talletuksen käytännössä tekee. Samoin esimerkin 43 kommentissa oleellista on päivittäminen, ei se, kuka päivittää. Esimerkeissä nostetaan keskeisimmäksi sisällöksi predikaattiverbin kuvaama toiminta. Toiminnan korosteisuuden voi havaita myös siitä, että kyseiset esimerkit olisivat hankalasti

tulkittavissa siten, että lauseen kirjoittaja ajattelisi implisiittiseksi subjektiksi niihin itsensä, sillä hän ei itse suorita varsinaista verbin taustalla olevaa toimintoa. Jos esimerkin 43 lause olisikin *//päivitän hahmon sijainnin*, antaisi se vaikutelman sellaisesta tilanteesta, jossa koodin kirjoittaja aktiivisesti päivittäisi tiedon ohjelmalle joka kerta, kun sitä käytetään. Tällaista päivittämiseen ei tällä kommentilla viitata, sillä kommenttia seuraavalla koodirivillä tietokone laitetaan päivittämään paikkataulukon parametreja yhteenlaskuoperaatiolla.

Näissä tavallisimmissa tapauksissa syynä passiivin käytölle voidaan siis pitää toiminnan asettumista keskiöön ja kirjoittajan oman roolin vähäisyyttä toiminnan lopullisessa tapahtumisessa. Tämän lisäksi myös kommentin sisällön yleisluontoisuudella voi olla vaikutusta passiivimuodon valintaan. Yksikään esimerkkien 42–45 kommenteista ei pidä sisällään mitään henkilökohtaista tai tekijään viittaavaa. Verbit *tallettaa*, *päivittää*, *liikutella* ja *kopioida* ovat sävyltään neutraaleja, eikä niillä ole tarkoitus ottaa kantaa toiminnan hyvyyteen tai huonouteen. Mikään esimerkkien predikaattiverbeistä ei myöskään kuvaa ainakaan ensisijaisesti inhimillistä toimintaa. Kaikki toiminnot toki voisivat olla myös ihmisen suorittamia, mutta yhtälailla niillä voi viitata tietokoneen suorittamiin toimintoihin, kuten näissä tapauksissa tehdään, sillä esimerkin 43 lailla jokaista muutakin kommenttia seuraa koodirivejä, joilla tietokone suorittaa kommentissa kuvatun toiminnon.

Toiminnan korostamisen lisäksi syynä esimerkkien ja niiden kaltaisten kommenttien passiivimuotoisuudelle voikin olla kommenttien kohteen yleisluontoisuus. Kommenttien kohteena on tarkkaan tehtävänannon mukaan tehty ohjelmointikoodi, joka ei ole yhtä henkilökohtainen työ kuin esimerkiksi tarkkojen ohjeiden mukaan kirjoitettu oppimispäiväkirja olisi. Koodissa mahdollisten ratkaisujen määrän voi ajatella olevan rajattu, kun taas oppimispäiväkirjaa kukaan opiskelija tuskin kirjoittaisi sanasta sanaan tismalleen samalla tavalla, mikä on yksi selkeä ero keinotekoisesti luodun ohjelmointikielen ja ihmisten kommunikointiin käyttämän luonnollisen kielen välillä.

Sama yleisluontoisuus voi päteä koodissa ja sen kommenteissa myös tämän aineiston kurssityön ulkopuolella, sillä Javassa on lista tiettyjä ”varattuja sanoja”, joilla toimintoja on mahdollista aiheuttaa (Wikla 2003: 110). Jokainen Java-koodin kirjoittaja operoi näiden sanojen avulla, joten käytännössä ohjelman toteuttavia toimintoja ja niiden nimiä ei ole taitavankaan ohjelmoijan mahdollista päättää itse. Muuttujien ja operaatioiden tunnukset kukin ohjelmoija nimeää itse haluamallaan tavalla, mutta se on vain osa koodia, joka rakentuu varattujen sanojen listan varaan. Siis vaikka koodia voi pitää luomisen lopputuloksena, ja jokainen koodi on tavallaan uniikki, eivät yksittäiset kommentoitavat koodirivit useinkaan ole kovin henkilökohtaisia tuotoksia, jolloin kommentitkaan eivät ole yleisilmeeltään henkilökohtaisia. Passiivimuotoista predikaattia käyttämällä jätetään koodin henkilökohtainen kädenjälki toissijaiseen arvoon ja annetaan tilaa muille koodissa tärkeille osille.

Vaikka passiivimuotoinen lause jättää aktiivisen tekijyyden taka-alalle, on sille ajateltu tekijä mahdollista päätellä (Alho & Kauppinen 2009: 130). Näin ollen ohjelmointikoodin passiivimuotoisista kommenteistakin löytyy tyyppejä, joissa tekijäksi on selkeästi tulkittavissa joku koodin osapuolista. Passiivin implisiittisen tekijän tunnistaminen on interpersoonaisen metafunktion piirissä oleellista, sillä interpersoonaisella metafunktiolla tarkoitetaan sitä, miten kieltä käytetään rakentamaan sosiaalisia suhteita, ja yksi keino tutkia tätä on tekstin persoonan kautta (Shore 2012b: 161, 177). Kun passiivi on läpi aineiston kommenttien yleisin persoonamuoto, voi sitä ja sen ajateltuja tekijöitä tutkimalla saada selville, miten kommenttien kirjoittaja suhtautuu koodiin, sen lukijaan ja itseensä kirjoittajana.

Karkeasti jaoteltuna koodiin ja sen luomaan ohjelmaan osallistuu aineistossa passiivin näkökulmasta neljä tekijää: (1.) tietokone, joka suorittaa toimintoja koodissa, (2.) koodaaja, joka antaa tietokoneelle välineet toteuttaa koodia, (3.) käyttäjä, joka pelaa koodin luomaa peliä, ja (4.) hahmo, jota pelissä ohjaillaan. Esittelen seuraavissa luvuissa aineiston passiivimuotoisista kommenteista esimerkkejä kaikista neljästä tekijätyypistä äsken luettelemassani järjestyksessä, joka myös vastaa passiivin tekijyyksien yleisyyttä aineistossa.

5.1.1.1 Implisiittisenä tekijänä tietokone

Kaikkein eniten aineistosta löytyy kommentteja, joissa tekijäksi on perustelluinta tulkita tietokone. Käytän tietokonetta tässä yhteydessä tarkoittamaan sitä osaa tai ohjelmaa, joka koodia tulkitsee, en konkreettista keskusyksikköä, näyttöä ja näppäimistöä. Näin ympäripyöreän käsitteen käyttäminen yksinkertaistaa ohjelmoinnin tekniikkaa jonkin verran, mutta ajatus taustalla on yhtä kaikki se, että joku tietokoneen sisäinen osa on tekijänä, ja se riittää tarkkuudeksi kielitieteellisen tutkimuksen puitteissa. Esimerkit 46–48 ovat tyypillisiä passiivikommentteja, joissa tekijäksi voi ajatella tietokoneen. Oleellista on se, että jokaisen kommentin jälkeen ohjataan koodilla tietokone suorittamaan kommentissa kuvattu toiminto, sillä kommentteja seuraava koodi on merkittävänä apuna kommentin implisiittistä tekijää pääteltäessä.

(46) // **Vähennetään alaosan laskuria** (A6)
rivMaksimi = rivMaksimi - 1;

(47) // **Palautetaan taulukko.** (A39)
return paikat;

(48) // **Tulostetaan tervehdysteksti.** (A42)
System.out.println(WORM);
System.out.println(WORM1);
System.out.println(WORM2);

Esimerkki 46 on puhdas malliesimerkki tilanteesta, jossa kommentin implisiittinen tekijä on tietokone. *Vähentämisestä* kommentoimisen jälkeen seuraavalla rivillä tietokone suorittaa vähennykseksi kutsuttavan toimenpiteen, jossa muuttujasta (*rivMaksimi*) vähennetään yksi (*rivMaksimi - 1*) ja tämän jälkeen sijoitetaan uudeksi arvoksi *rivMaksimi*-muuttujaan. Kommentissa kuvatun toiminnon suorittaa todistetusti tietokone, jolloin sen rooli implisiittisenä tekijänä kommentissa on selvä. Jos tässä tapauksessa implisiittiseksi tekijäksi yrittäisi pelkän kommentin perusteella tulkita koodin kirjoittajan, tulisi seuraavalla koodirivillä olla valmiiksi suoritettu laskutoimitus, jonka ihminen on ennen tiedostoon kirjoittamista suorittanut.

Esimerkki 47 on yhtä puhdas tapaus. Ilmaisulla *palautetaan* tarkoitetaan sitä, että seuraavalla rivillä koodissa tehdään palautustoimenpide, joka käytännössä tapahtuu ohjelman käyttämisen aikana varatun sanan (*return*) määräämällä komennolla. Kun palauttamisen tekee koodissa itsenäisesti tietokone niin, että koodaaja on vain antanut sille käskyn, on kommentissa yhtä lailla kuin koodissakin implisiittisenä tekijänä tietokone. Yhä samoin periaattein toimii esimerkki 48, jossa kommentissa kerrotaan, että *tulostetaan*, ja seuraavalla kolmella koodirivillä aiheutetaan tulostuminen (*System.out.println()*), jonka tietokone hoitaa itsenäisesti. Koodi on kaikissa näissä tapauksissa määrittelevänä jäsenenä päättelyketjussa, jossa tekijä osoitetaan.

Koodin suuren roolin lisäksi yhteinen piirre esimerkkien 46–48 kommenteille on se, että predikaattiverbien merkitys tukee tietokoneen tulkintaa implisiittiseksi tekijäksi. Verbit *vähentää*, *palauttaa* ja *tulostaa* kuvaavat juuri tietokoneelle tyypillisiä toimintoja, vaikka kuten myös esimerkeissä 42–45, voisivat ne tarkoittaa yhtä lailla inhimillistä toimintaa. Java-koodin yhteydessä tietokonemaiseksi toiminnaksi verbit merkkaa jo se, että kaikille on vastineensa Java-kielessä. *Vähentämisen* laskutoimituksen merkki on Javassa ”-”, *palauttamisen* varattu sana on ”return” ja *tulostamisen* valmiiksi määritelty tulostusoperaatio on ”System.out.println” (ks. Wikla 2003: 17, 20, 110).

Sanasto perustelemassa tietokonetekijää käy ilmi myös siitä, miten paljon samankaltaisilla tai tismalleen samoilla verbeillä varustettuja passiivimuotoisia kommentteja aineistosta löytyy. Sanasto on vakiintunutta, kun sillä osoitetaan tietokoneen suorittamia toimia, joita on ohjelmointikielissä rajallinen määrä. Esimerkeissä 49–51 on kommentteja, joissa kuvataan samat toiminnot kuin esimerkeissä 46–48: myös näissä esimerkeissä tehdään laskutoimituksia *lisätään*, *palautetaan* ja *tulostetaan*. Tietokoneen päättely implisiittiseksi tekijäksi käy samoin perustein kuin edellisissäkin esimerkissä, sillä kommentteja seuraavat koodirivitkin aiheuttavat samantyyppisen toimenpiteen.

(49) //**Laskuriin <u>lisätään</u> yksi.** (A22)
liikelaskuri++;

(50) // **Jos hahmolle ei ole varattu muistia, palautetaan tyhjäarvo.** (A20)
```
else
  return null;
```

(51) //**tulostetaan uusi ruokamerkki kentalle.**
```
boolean ruokaKentalla = Automaatti.tarjoile(kentta);
```

Esimerkin 52 kommentin passiivimuotoinen predikaatti on sekin selkeästi tulkittavissa tietokonetekijäiseksi. Tässäkin esimerkissä suoritetaan passiivipredikaatin (*loopataan*) lupaama toimenpide for-silmukan avulla, joten jo se sitoo tietokoneen kommentin lauseen implisiittiseksi tekijäksi, kun kommentissa kuvatun toimenpiteen tekee tietokone. Erityisen huomion ansaitsee verbi *loopata*, joka ei ole yleiskieleen kuuluva sana, vaan joka kuuluu ohjelmoinnin erityissanastoon (lisää luvussa 5.2.2). Sana juontuu englannin kielen sanasta *loop*, jonka merkitys on tässä yhteydessä ”tehdä silmukka” tai ”silmukoida”. Ei ole epäilystäkään siitä, että kommentin kirjoittaja olisi tarkoittanut, että hän esimerkiksi itse ”tekisi silmukoita”, sillä sanan merkitys olisi tällöin hyvin kaukana ohjelmointikoodin ja sen kommenttien kontekstista. On ilmeistä, että tällaisen alaan liittyvän verbin käyttäminen ohjaa tulkinnan tietokoneeseen tekijänä.

(52) //**Loopataan rivin kaikki sarakkeet läpi** (A10)
```
for (int sarake = 0; sarake < hahmotaulu[0].length; sarake++){
```

Esittelemieni esimerkkien valossa voi todeta, että monessa kommentissa passiivilla viitataan yksinomaan tietokoneen suorittamaan toimintaan. Tietokone valtaa suurimman osan passiivimuotoisten kommenttien implisiittisen tekijän paikoista, mutta myös sellaisia kommentteja löytyy, joissa tekijänä voi pitää lauseen kirjoittajaa, eli harjoitustyön tekijää ja kommentin ja koodin tuottajaa. Nämä ovat huomattavasti harvinaisempia kuin edellä esittelemäni tietokonetekijäiset, ja monet sisältävät toisena tekijänä myös tietokoneen.

5.1.1.2 Implisiittisenä tekijänä koodin kirjoittaja

Esimerkkien 53–54 kommenttien passiivimuotojen implisiittiseksi tekijäksi voi ajatella koodin kirjoittajan.

(53) ///**Luodaan metodi liikkumista varten** (A48)
```
public static int[][] liikutaHahmoa(char ohjaus, int[][] paikat, char[][] kentta, int sarakkeet, int rivit,
boolean syominenOK, char TAUSTA, char YLOS, char ALAS, char OIKEA, char VASEN, char
KAANTO){
…
```

Esimerkin 53 kommentissa viitataan siihen prosessiin, jossa koko koodi on tehty, sillä siinä käytetään verbiä *luoda* ennen metodin esittelyä. Kommenttia seuraavan koodin avulla voidaan perustella,

että implisiittinen tekijä luomistyössä on ihminen koodin takana. Kommentti ei suoraan viittaa koodissa tapahtuviin asioihin, sillä sitä seuraavassa metodissa ohjelma ei ”luo metodia”, vaan kommentilla viitataan toimenpiteeseen, jonka koodin kirjoittaja tekee. Voi olla, että hän on kirjoittanut kommentin muistutukseksi itselleen siitä, että kohtaan tulee luoda metodi liikkumista varten, tai että hän on jäsentänyt koodinkirjoitusprosessiaan kommentoimalla koodin lisäksi myös omaa toimintaansa. Tällainen kommentti on kirjoittajan metatekstiä omasta koodin- ja kommenttienkirjoituksestaan, sillä sen avulla hän viittaa omaan tulevaan toimintaansa, ei tulevaan koodin toimintaan. *Hahmon liikuttaminen* sen sijaan on tietokoneen tehtävä, mutta kommentin ainoa passiivimuotoinen predikaatti on juuri *luoda*-verbi, jolloin kirjoittaja nousee implisiittisen tekijän asemaan.

Esimerkin 54 kommentissa on yksi virke, jonka passiivin tekijäksi voi ajatella kirjoittajan itse.

(54) /* Tarkoituksena on kiepsauttaa koko hahmo ympäri. **Ajatellaan, että**
*** jaetaan hahmoTaulukko kahtia, jolloin siinä on yläosa ja alaosa.** Hahmon pää on /
* yläosassa ja häntä alaosassa. Sen jälkeen aletaan vaihtaa vastakkaisten /
* koordinaattien paikkaa ylä- ja alaosasta./
*/ (A6)

Myös tässä esimerkissä 54 kirjoittaja viittaa omaan kirjoitusprosessiinsa ja käyttää mentaalista verbiä *ajatella*. Mentaalisilla verbejä ovat sellaiset, joiden tehtävänä on kuvata mielen sisäisiä tapahtumia, tiloja, tekoja ja toimintoja (VISK § 445). Ajatteleminen on toiminto, jota tietokone ei voi tehdä, joten on selvää, että sillä viitataan juuri ihmisen harjoittamaan toimintaan. Mentaaliset verbit edellyttävätkin, että osallistuja on elollinen ja tajunnallinen, jollainen ihminen luonnollisesti on, mutta jollaiseksi tietokonetta ei voi luokitella (VISK § 445). Jo tämä verbien semanttinen määrittely on syy sille, että tässä kommentissa implisiittinen tekijä on inhimillinen.

Kommentista löytyy myös tietokoneelle osoitettu toimenpide eli *taulukon kahtia jakaminen*, mutta se esitetään vasta passiivimuotoiselle *ajatellaan*-lauseelle alisteisessa sivulauseessa, jolloin päälauseen passiivin implisiittisenä tekijänä ei ole tietokone vaan kirjoittaja. Näin myös sitä seuraava lause on alisteinen kirjoittajan tekijyydelle, sillä toimintoa ei varsinaisesti suoriteta, vaan sitä ainoastaan *ajatellaan* eli prosessoidaan ihmisen mielen sisällä. Ilman *ajatella*-verbiä (*//Jaetaan hahmoTaulukko kahtia*--) kommentin tekijän henkilökohtainen näkemys ei tulisi esiin. Myös merkitys muuttuisi ilman mentaalisen verbin (ks. VISK § 445) läsnäoloa, sillä kommentissa korostetaan sitä, että käytännössä taulukko ei jakaudu kahtia, mutta inhimillistä hahmottamista helpottaakseen kirjoittaja kuvaa toiminnon näin, sillä tietokone pystyisi hahmottamaan taulukon yhtenäkin kappaleena.

Esimerkit 55–58 eivät ole esimerkkien 53–54 tavoin yhtä puhtaasti tulkittavissa ainoastaan kirjoittajatekijäisiksi, mutta niistä löytyy piirteitä, joiden perusteella pelkän tietokoneen nimeäminen passiivimuodon implisiittiseksi tekijäksi ei tule kyseeseen.

(55) / **Tarkastellaan siirtyikö hahmo kentän reunojen ylitse**
// ja annetaan tarvittaessa koordinaatti kentän toiselta puolelta (A45)
```
if(hahmoy >= pelikentta.length)
  hahmoy = 0;
else if(hahmox >= pelikentta[0].length)
  hahmox = 0;
…
```

(56) // **Päätellään ollaanko peruuttamassa.** Peruuttaessa pään ja päätä
// seuraavan merkin paikat ovat samat. (A4)
```
boolean peruutetaan = paanRivInd == paikat[1][0] && paanSarInd == paikat[1][1];
```

(57) // **Luetaan syötteet käyttäjältä ja niiden perusteella päätetään**
// hahmon päämerkin liikkumissuunta koordinaaatistossa. (A7)
```
if (syote == VASEMMALLE){
paasarake --;
}
...
```

(58) // **Käyttäjän antaessa välilyönnin ei muuteta mitään, koska se tulkitaan**
// vuoron ohittamiseksi.
```
if (suunta == ' ')
   return true;
```

Kaikkien näiden esimerkkien 55–58 passiivimuotoinen predikaattiverbi kuvaa sellaista ihmisen päänsisäistä toimintaa, jota ei voi pitää yksinomaan tietokoneen suorittamana. Myös tässä tapauksessa syynä on verbien mentaalisuus (ks. VISK § 445). *Tarkastella*, *päätellä*, *päättää* ja *tulkita* ovat huomattavasti inhimillisempiä verbejä kuin esimerkeissä 46–52 esittelemäni tietokonetekijäiset verbit, sillä ne kaikki kuvaavat ajatteluprosessia jollakin tavalla. Näistä verbeistä *päätellä*, *päättää* ja *tulkita* kuvaavat selvästi mielensisäistä toimintaa, sillä tapahtumat eivät ole konkreettisesti havaittavissa (VISK § 445, 472). Verbi *tarkastella* voisi kuvata myös konkreettista tilannetta, jossa katse kiinnitetään tiettyyn kohteeseen, mutta tässä yhteydessä sillä viitataan mielen sisällä tapahtuvaan toimintaan. Kutakin kommenttia seuraava koodi tuo myös tietokoneen läsnäolevaksi passiivin tekijään, sillä esimerkiksi esimerkin 56 konkreettisen *päättelyn* suorittaa kuitenkin lopulta tietokone. Koodin kirjoittaja on antanut logiikan koodin toiminnalle, ja saattaa siksi kommentissaan kuvata omaa ajatteluprosessiaan, mutta tietokone tekee koodissa käytännön työn.

Passiivin implisiittisenä tekijänä voi olla samassa kommentissa sekä koodin kirjoittaja että tietokone. Esimerkin 57 kommentissa predikaatti *päätetään* viittaa koodin kirjoittajan henkiseen toimintoon, mutta *luetaan syötteet* sisältää implisiittisenä tekijänä tietokoneen. Tällaisissa tilanteissa,

joissa passiivimuotoiset predikaatit tuovat läsnä oleviksi sekä kirjoittajan että tietokoneen näkökulmat, voi passiivin merkitys lähennellä me-passiivia. Me-passiivilla tarkoitan ilmiötä, jossa puhutussa kielessä käytetään passiivia monikon 1. persoonan tilalla (Kuiri 2000). Ohjelman tuottaminen on tietokoneen ja koodin kirjoittajan yhteispeliä, jossa molemmat tarvitsevat toisiaan. Vaikka monikon 1. persoonan käyttö olisikin liian raskaan tyylistä koodin kommenteissa (esim. *// Luemme syötteet käyttäjältä ja niiden perusteella päätämme* --), sillä näin häivytetyksi haluttu tekijyys korostuisi, voi passiivi joissain tilanteissa olla tapa viitata yhteiseen toimintaan.

Kommenttien ja koodin suhteesta löytyy monitulkintaisuutta, ja aina ei ole mahdollista aukottomasti päätellä, viittaako kommentin ja koodin kirjoittaja passiivimuodolla itseensä vaiko tietokoneeseen.

5.1.1.3 Implisiittisenä tekijänä pelihahmo tai pelaaja

Tietokoneen ja käyttäjän lisäksi aineistosta on mahdollista löytää sellaisia passiivimuotoja, joiden implisiittinen tekijä on pelin keskushahmo, ja sellaisia, joiden implisiittinen tekijä on pelin pelaaja. Verrattuna tilanteisiin, joissa implisiittinen tekijä on tietokone tai kirjoittaja, on tällaisia tapauksia aineistossa huomattavasti vähemmän.

Esimerkkien 59–61 kommenttien implisiittiseksi tekijäksi voidaan ajatella pelin pelaajan. Koodin ja kommenttien kirjoittaja ei periaatteessa voi tietää, kuka peliä todella pelaa, sillä pelaaminen tapahtuu vasta koodin kirjoittamisen jälkeen eli pelin valmistuttua, joten pelaaja voi olla kuka tahansa aina koodin kirjoittajasta itsestään lähtien. Pelaajia ei kuitenkaan tämän aineiston pelissä voi samalla pelikerralla olla enempää kuin yksi, joten on mahdollista yleistää tämän implisiittisen tekijän nimeksi epämääräinen ja yleinen ”pelaaja”.

(59) **// Pelataan, jos on varattu muistia. (A3)**
```
if (kentta != null && hahmo != null) {
```

(60) **// metodi joka tarkastaa mikä merkki (eli komento) on annettu, ja muuttaa**
// taulukon, "merkkienPaikat", numeroita sen mukaan (A13)
```
public static int[][] komennot (int[][] merkkienPaikat, char komento,char[][] kentta) {
```

(61) **// Jos halutaan kääntää hahmo toisinpäin (käyttäjä painanut s:ää).** (A54)
```
if (siirto == 's'){
  toisinpain(hahmonPaikat, apuPaikat);
  return apuPaikat;  }
```

Esimerkin 59 kommentin virkkeessä on kaksi lausetta, joista ensimmäisen tekijäksi voi tulkita pelaajan ja toisen tietokoneen. On selvää, että verbillä *pelata* viitataan siihen henkilöön, joka peliä pelaa, sillä *pelaaja* on verbin *pelata* johdos. Pelaamisella tarkoitetaan yleisesti sitä tilannetta, jossa

ohjelmaa todellisuudessa käytetään, sillä koodissa ei ole esimerkiksi *pelaa*-nimistä osasta. Esimerkin 60 osa *mikä merkki on annettu* viittaa vääjäämättä pelaajaan, sillä aiemmassa osassa peliä käyttäjältä on pyydetty syötteenä joku merkki. Koodi siis kytkee pelaajan tämän kommentin osan tekijäksi. Merkkiä ei anneta millään tavalla koodin sisällä, vaan se tulee joka kerta koodin ulkopuolelta eli ohjelman käyttäjältä, joka tämän ohjelman tapauksessa on pelaaja. Esimerkin 61 verbi *haluta* riittäisi yksinään sitomaan tekijäksi juuri pelaajan, sillä pelin näkökulmasta hän on ainoa, joka voi tehdä valintoja peliä koskien. Tämän lisäksi kommentin kirjoittaja vielä painottaa käyttäjän asemaa tekijänä sulkuihin lisätyllä lauseella, jonka perussubjektina on *käyttäjä*.

Esimerkeissä 62–64 passiivimuodon implisiittisenä tekijänä on perusteltua pitää pelissä ohjailtavaa hahmoa. Kaikissa kommenteissa predikaattiverbit kuvaavat konkreettista toimintaa (*olla reunalla*, *kulkea*, *ollaan liikkumassa*), ja peliin osallistujista ainoa konkreettiseksi luokiteltavaan toimintaan kykenevä on pelissä liikkuva hahmo.

(62) //**Jos ollaan reunalla.** (A2)
```
if (hahmonpaa[1] == kentanleveys){
…
```

(63) // **Jos kuljetaan vasemmalle.** (A4)
```
else if (suunta == 'l'){
  paanSarInd = paanSarInd -1;
```

(64) // **Jos ollaan liikkumassa reunan yli, hypäytetään pää kentän toiselle puolelle.** (A61)
```
if ((reunalla == true) && (koordinaatit[0][1] == 0)) {
  if (hahmovanhax == (t[0].length - 1)) {
```

Kommenttien merkitys on vahvasti riippuvainen sitä seuraavasta koodista. Yksi tärkeimmistä perusteista implisiittisen tekijän nimeämisessä on se, mitä kommentin kommentoimassa koodissa tapahtuu. Kommentti on siinä mielessä riippuvainen koodista, että ilman sitä on huomattavasti hankalampaa päätellä, kenet implisiittiseksi tekijäksi on ajateltu. Joissakin tilanteissa päättely onnistuu esimerkiksi pelkän kommentin predikaattiverbin semantiikan perusteella, ja joissakin tilanteissa kommentti tarvitsee rinnalleen koodin, jotta tulkinnan voi eri vaihtoehtojen välillä tehdä.

Passiivin implisiittinen tekijä on suurimmassa osassa tapauksia tietokone. Myös koodin ja kommentin kirjoittaja paljastuu joissakin tapauksissa implisiittiseksi tekijäksi, vaikka usein näihin yhdistyy myös implisiittinen tietokonetekijä. Implisiittisen tekijän voi siis tunnistaa useista tapauksista. Suomen kielen passiivin yksi ominaisuus on piilottaa tai häivyttää lauseen tekijä (Kuiri 2000), joten vaikka implisiittinen tekijä kertoo tekstilajista paljon, on todennäköistä, että passiivin käytöllä halutaan pienentää aktiivisen tekijän roolia ja korostaa kommenttien kuvaamien tapahtumien toiminnallisuutta.

5.1.2 Modaaliset valinnat

Interpersoonaiselta kannalta tekstiä tarkasteltaessa huomiota kiinnitetään modaaliaineksiin tai niiden puuttumiseen (Halliday 1985: 87, Shore 2012b: 177). Modaalisuudella tarkoitetaan arvioita asiaintilan todenmukaisuudesta ja toteutumismahdollisuuksista, eli modaalisilla kielenaineksilla puhuja voi ilmaista esimerkiksi sen, kuinka todennäköisenä, mahdollisena, pakollisena tai toivottavana hän asiaintilaa pitää. Kielenainekset näiden ilmaisuun ovat kunkin kohdalla omanlaisensa. (VISK § 1551.) Interpersoonaiseen alaan liittyviksi modaalisiksi valinnoiksi voidaan kutsua esimerkiksi modusta, modaalisia partikkeleita ja kommenttiadverbiaaleja (Shore 2012b: 147). Esittelen nyt niitä interpersoonaisia modaalisia valintoja, jotka nousevat keskeisiksi ja hallitseviksi ohjelmointikoodin kommenteissa.

Aineiston modaalisten valintojen osalta suurin huomio kiinnittyy siihen, että kommentit sisältävät melko vähän modaalisia elementtejä, sillä läheskään jokaisesta kommentista modaalisuutta ei voi havaita. Asiat esitetään passiivimuotoisen predikaattiverbin avulla totena, kuten hyvin tyypillisestä esimerkistä 65 voi huomata.

(65) **// Tulostetaan tervehdysteksti. (A42)**
System.out.println(WORM);

Esimerkissä 65 verbi on indikatiivimuotoinen eli tunnukseton, eikä sen tarkoituksena ole tässä yhteydessä esimerkiksi käskeä tai ohjailla, jolloin indikatiivimuotoisen verbin voisi sanoa ilmaisevan modaalisuutta (VISK § 1590). Tyypillisesti kommentti kuvaa koodissa tapahtuvan asian puhtaasti sellaisena kuin se koodissa tapahtuu, eikä siinä anneta näkemystä koodin tapahtumien hyvyydestä tai huonoudesta. Myöskään todennäköisyyteen tai pakollisuuteen ei kommenteissa pääasiassa tarvitse ottaa kantaa, sillä kommenttien kuvaamat tapahtumat tapahtuvat koodissa, tai muuten kommenttia ei olisi olemassa. Kommentin olemassaolo on riippuvainen sen kommentoimasta koodista, joten niiden koodia kuvaileva sisältö on lähes aina paikkansapitävää.

Esimerkiksi seuraavanlaista kommenttia ei valmiiseen koodiin olisi mielekästä kirjoittaa: ”*Tulostetaan todennäköisesti tervehdysteksti*”. Koska koodin ja kommenttien kirjoittaja on sama henkilö, on hänen tietoisuutensa koodin aiheuttamista toiminnoista välttämättä niin korkealla tasolla, ettei valmiin koodin kommenteissa varaa jää epäilyille toimintojen todennäköisyydestä tai mahdollisuudesta. Koodin ja kommentin kirjoittajalla ei todennäköisesti ole tarvetta pohtia kommentoimaansa asiaa ja sijoittaa tekstiin modaalisia elementtejä, sillä mahdolliset pohdinnat on tehty siinä vaiheessa, kun erilaisten koodien toimivuutta on testattu. Lopulta kommentoijan mielipiteellä tai näkemyk-

sellä asiaintilasta ei ole juurikaan koodin tai sitä lukevan toisen koodaajan kannalta väliä, joten modaalisia aineksia saatetaan jopa tietoisesti vältellä.

Välttämättömyyden ja mahdollisuuden ilmaisukeinoista (VISK § 1551) ainoastaan muutamia yksittäiskappaleita löytyy aineistosta. Sisällölliset tilanteet, joissa näitä modaalisuuden ilmaisukeinoja käytetään, ovat melko samankaltaisia eri kommenteissa. Niissä modaaliset kielenpiirteet liittyvät tilanteisiin, joissa tapahtumat ovat pelin pelaajan käsissä ja modaalisuus ei kosketa varsinaisesti itse koodia vaan sen rakentamaa peliä. Tällaisesta tilanteesta ovat esimerkit 66–67, joista olen alleviivannut modaaliset kielenainekset. Näiden esimerkkien kommenteissa kirjoittaja ottaa kantaa tilanteen toteutumismahdollisuuksiin, mutta modaalisuus ei kosketa itse koodia.

(66) //**Muuttuja joka muuttuu falseksi, kun kayttaja tekee siirron joka ei ole sallittu.**
//Jos siirto ei ole laillinen, hahmon liikuttelua ei tapahdu. (A37)
boolean laillinen = true;

Esimerkissä 66 modaalisuutta ilmentää sanavalinta *sallittu*, joka luokitellaan mahdollisuutta kuvaavaksi adjektiiviksi (VISK § 1551). Esimerkissä kantaa otetaan käyttäjän tekemään siirtoon ja sen sallittavuuteen (-- *kun kayttaja tekee siirron joka ei ole sallittu* --), ja kerrotaan mahdollisista tapahtumista, kun siirto ei ole sallittu (*Muuttuja joka muuttuu falseksi* ja *-- hahmon liikuttelua ei tapahdu)*. Kommentin ensimmäisen virkkeen ensimmäinen lause ei toteudu, jos käyttäjän siirto on sallittu, joten koko virke kuvaa mahdollisia tapahtumia, toisin kuin suurin osa kommenteista, joissa kerrotaan koodissa tapahtuvat asiat, kuten ne yksiselitteisesti ovat (ks. esimerkki 65). Mahdollisuus ei kuitenkaan liity koodiin, jota sen kirjoittaja kommentoi, vaan peliin ja käyttäjän tekemiin valintoihin siellä. Kommentti ei kosketa koodin sallittavuutta tai mahdollisuutta, vaan käyttäjän valintojen seurauksia, joiden ilmaisun apuna käytetään modaalista adjektiivia *sallittu*. Adjektiivi tekee koko kommentista modaalisen.

Esimerkin 67 kommentin modaalisuus välittyy verbistä *tarvittaessa*, joka kuvaa välttämättömyyttä (VISK § 1551).

(67) // **Tarkastellaan siirtyikö hahmo kentän reunojen ylitse**
// ja annetaan tarvittaessa koordinaatti kentän toiselta puolelta (A45)
if(hahmoy >= pelikentta.length)
hahmoy = 0;

Esimerkissä 67 tarvittavuuden lähtökohdat määritellään kommentin ainoan virkkeen ensimmäisessä lauseessa, sillä tarve *siirtää koordinaatti kentän toiselta puolelta* (2. lause) syntyy ainoastaan, jos *hahmo siirtyy kentän reunojen ylitse* (1. lause). Myös tässä esimerkissä modaalisuutta esiintyy

kommentissa, jossa esitellään peliin sijoittuvia tapahtumia. Hahmon siirtyminen kentän reunojen ylitse on kiinni pelaajan ohjausvalinnoista, joten itse koodissa ei ole tilannetta, jossa tilanteen välttämättömyyteen tarvitsisi ottaa kantaa. Ilmauksella *tarvittaessa* ei tässä tapauksessa siis esimerkiksi tarkoiteta tilannetta, jossa *tarvittaessa* pelastettaisiin epäonnistunut koodi. Modaalisilla valinnoilla kommenteissa tavallaan keskustellaan pelin pelaajan kanssa, sillä modaalisuutta ei tarvitsisi pelkän koodin näkökulmasta juurikaan olla, kuten esimerkin 65 avulla esitin. Näillä keinoilla osoitetaan, että tekstin – niin koodin kuin kommenttienkin – kirjoittamisen ainoa itseisarvoinen syy ei ole jokaisessa tilanteessa oikeaoppinen ohjelmointikoodi, vaan sitä kirjoitetaan, jotta ohjelman käyttäjä voi pelata koodilla ohjelmoitua peliä. Koodiin ja kommentteihin suhtaudutaan keinoina rakentaa tämä peli, ja sen pelaaja otetaan yhdeksi osallistujaksi kommenttien kielen avulla.

Pääasiassa modaalisuutta ilmenee siis kommenteissa, joissa ohjelman käyttäjän tekemät ratkaisut määräävät lopputuloksen. Modaalisia valintoja ei ole tarpeellista tehdä tilanteissa, joissa kuvaillaan ainoastaan kommentissa esiintyviä tapahtumia, sillä niiden kohdalla totuutta ja mahdollisuutta ei tarvitse pohtia. Aineisto koostuu kuitenkin valmiista koodeista, jotka ovat menneet onnistuneesti koneellisen kääntäjän läpi eli toisin sanoen niiden toteuttamat pelit toimivat ainakin jollakin tavalla, ja myös ihminen eli kurssin opettaja on tarkastanut työt. Tällöin koodeihin ja näin ollen kommentteihinkaan ei ole voinut jäädä epävarmoja ja toimimattomia elementtejä, joita keskeneräisessä työssä olisi saattanut olla, ja jotka olisivat saattaneet aiheuttaa modaalisia elementtejä myös koodin kommentointiin.

Kommenttien modaalisista piirteistä nousee vahvasti esiin normatiivisuus eli se, mitä pelissä saa tai ei saa tehdä. Tällainen on esimerkki 66, ja ennen kaikkea esimerkeissä 68 ja 69 modaalisuudella osoitetaan kuvattujen asioiden normatiivisuutta.

(68) **//hahmoa ei päivitetä, jos siirto ei ole laillinen eli**
//jos se menee itseään päin tai koittaa peruuttaa (A29)
laillinen(kentta, ptaulu, paaRivi, paaSarake, hantaRivi, hantaSarake);

(69) **/* Annetaan hahmolle liikkumisohje valintasyötteen mukaan. Hahmon paikkatiedot**
*** päivitetään valintaa vastaavaksi. Hahmon peruuttaminen ei ole sallittua./**
*** Lopuksi palautetaan päivitetyt paikkatiedot main-metodiin./**
***/** (A15)
public static int[][] liikutaHahmoa(int[][] paikat, char valinta, char[][] kentta) {

Esimerkeissä 68 ja 69 osoitetaan sanavalinnoilla *laillinen* ja *sallittu*, että koodin rakentamassa pelissä on sääntöjä, joita pelaajan pitää noudattaa ja joiden noudattamista koodin tulee valvoa. Se, onko *siirto laillinen* (esimerkki 68) tai *peruuttaminen sallittua* (esimerkki 69) ei ole kiinni koodista, sillä Java-kielellä pelihahmon peruuttaminen tai tietynlainen siirto olisi kyllä mahdollista toteuttaa. Olen

jo todennut, että modaalisuus liittyy kaikissa esimerkeissä peliin ja käyttäjän tekemiin valintoihin, ja niin myös näissä.

Normatiivisten elementtien takaa nousee kuitenkin esiin kysymys siitä, miten jonkin asian laillisuus tai sallittavuus on määritelty. Kyseessä on tarkkaan ohjeistettu ohjelmointikurssin harjoitustyö, jonka kehykset ovat niin tiukat, että peruuttamiseen ja laillisiin siirtoihin on annettu ohjeet. Tämän takia modaalisuuden takana voi piillä opettajan kirjoittama tehtävänanto, jolla on oma roolinsa koodin ja kommenttien syntyprosessissa. Tehtävänannon mukanaoloa ei ole kirjoitettu kommenttiin auki, mutta modaalisten valintojen avulla voi olettaa, että se on syynä joidenkin asioiden mahdollisuudelle tai mahdottomuudelle. Toisaalta tehtävänannon ohjeistusten syynä ovat pelille kirjoitetut säännöt, joten pohjimmiltaan normatiivisuuden ohjelmointikoodin kommenteissa aiheuttaa sen toteuttaman pelin säännöt. Peli ei olisi peli, jos sitä voisi käyttää kuten tekee mieli, vaan jokaisella pelillä täytyy olla jonkinlaiset säännöt, mikä näkyy peliin liittyvissä kommenteissa modaalisuutena ja normatiivisuutena.

Modaalisuus aineiston ohjelmointikoodin kommenteissa on kaikin puolin vähäistä, sillä kommentoinnin tarkoitus on kuvailla koodia ja siinä tehtyjä valintoja objektiivisesti. Myös aineiston koodien valmius ja toimivuus ovat yksi syy sille, miksi modaalisuutta ei ilmaista koodia kohtaan. Modaalisia piirteitä löytyy aineistossa sellaisista kommenteista, jotka käsittelevät pelaajan tekemiä valintoja, sillä koodin ulkopuolinen pelaaja voi tehdä ohjelmassa juuri sellaisia ratkaisuja kuin tahtoo.

5.1.3 Sanavalintojen rakentamat suhtautumistavat

Asenteet ja arviot ovat osa tekstin interpersoonaista merkitystä (Halliday & Hasan 1985: 45), ja niitä voivat heijastaa tekstin sanavalinnat. Yhden interpersoonaisen metafunktion kannalta tarkasteltavan osa-alueen Shoren analyysissa muodostavatkin suhtautumista ilmentävät sanavalinnat (Shore 2012b: 177).

Kiinnitän tässä luvussa huomion yleisesti sellaisiin suhtautumistapoihin, joita kommenteista löytyvät sanavalinnat usein osoittavat. Kuten modaalisten valintojen kohdalla, totean aluksi, että suhtautumista ilmentäviä sanavalintoja ei todellakaan löydy jokaisesta kommentista, sillä useimmissa kommenteissa koodia kuvaillaan neutraalisti juuri sellaisena kuin se on. Esimerkki 70 havainnollistaa tätä sanavalintojen neutraaliutta, joka on ominaista aineiston kommenteille. Esimerkin kommentissa kuvaillaan sitä seuraavilla koodiriveillä tapahtuva taulukon kopiointi ja annetaan perustelut kopioimiselle ilman sanavalintoja, joiden voisi tulkita ilmentävän jotakin suhtautumistapaa. Kommentin kaikki sanavalinnat pyrkivät vain osoittamaan koodiin kirjoitetun tapahtuman totuudenmukaisesti.

(70) // **Kopioidaan taulukko paikan siirtämistä varten** (A18)
```
int[][] apuTaulukko = new int[mp.length][mp[0].length];
apuTaulukko = kopioiTaulukko(mp,apuTaulukko);
```

Kommenttien sanavalinnat eivät neutraaliudessaan ole aina täysin vapaita suhtautumisen ilmentämisestä. Sanavalinnoilla voidaan ilmaista sitä, kuinka ammattimaisesti ja asiantuntevasti koodiin ja kommentteihin suhtaudutaan. Esittelen esimerkeissä 71–74 pareittain kommentteja, jotka sisältävät koodin ammattimaisuuteen suhtautumista erilailla ilmentäviä sanavalintoja ja jotka liittyvät samankaltaiseen koodiin. Näin pareina esittämällä suhtautuminen tulee selkeästi ilmi, kun voi huomata, millaisia sanavalintoja kommenteissa käytetään tai ollaan käyttämättä, ja mitä sävyeroja sanavalintojen erot luovat. Myös esimerkki 75 havainnollistaa koodin ammattimaisuuteen suhtautuvia sanavalintoja.

Esimerkissä 71 puhutaan *hahmon ruhosta* ja esimerkissä 72 puhutaan *hahmon vartalosta*. Sanat viittaavat samaan tarkoitteeseen mutta ovat sävyiltään erilaiset.

(71) // **Siirretään <u>hahmon ruho</u> lähemmäs päätä.** (A5)
```
for (int i = 1; i < hahmo.length; i++) {
    hahmo[hahmo.length - i][0] = hahmo[hahmo.length - (i + 1)][0];
    hahmo[hahmo.length - i][1] = hahmo[hahmo.length - (i + 1)][1];
}
```

(72) // **Silmukka jossa <u>hahmon vartalo</u> liikutetaan.** (A25)
```
for(int i = 1; i < hahmo.length; i++) {
    hahmo[hahmo.length - i][0] = hahmo[hahmo.length - (i+1)][0];
    hahmo[hahmo.length - i][1] = hahmo[hahmo.length - (i+1)][1];
}
```

Molempien esimerkkien 71 ja 72 kommentoima koodi on sisällöltään tismalleen samanlainen, joten kommentit ovat sanojen synonyymisuuden lisäksi yhteneväiset kommentoitavan koodin osalta. Sanavalintana *vartalo* on neutraalimpi ja asiallisempi kuin *ruho*, sillä *ruho* luo mielikuvan suuresta ja kömpelöstä vartalosta, jota pikkiriikkisellä hahmolla tuskin pelissä ajatellaan olevan. Sanana *ruho* voi synnyttää ajatuksen kuolleesta vartalosta, jollainen pelissä aktiivisesti liikkuva hahmo ei myöskään ole. Tekemällä tällaisen sanavalinnan kommentin kirjoittaja osoittaa, että suhtautuu koodiin ja kommentointiin kepeästi, sillä sanavalinnoilla hassuttelu ei tulisi kyseeseen, jos kirjoittajan ensisijainen tavoite olisi kommenteillaan vakuuttaa lukijalle olevansa ammattimainen koodaaja.

Esimerkin 71 avulla voi huomata, että kepeää ja humoristista sanaa voi käyttää kommentissa ilman, että se aiheuttaa koodia ja kommenttia ymmärtävässä lukijassa eli diskurssiyhteisön jäsenessä hämmennystä tai epäilyä tekstin kuulumisesta kommenttien tekstilajiin. Tämä heijastuu kommentin tekstilajin vapaamuotoisuudesta ja epävirallisuudesta, sillä asiapitoisuuden ja tiiviyden ra-

joissa kommentissa voi usein käyttää leikittelevääkin kieltä. Sen ei tarvitse ainakaan opiskeluun liittyvän harjoitustyön tapauksessa olla virallista, asiallista ja ammattimaista, sillä koodin oikea toimiminen riittää osoittamaan kirjoittajan pätevyyden. Sanavalinnat ilmentävät kirjoittajan suhtautumista koodin vakavuusasteeseen, joka voi vaihdella neutraalista ja asiallisesta leikittelevään ja vitsikkääseen. Esimerkki 71 edustaa kepeää suhtautumista koodin ja kommentoinnin ammattimaisuuteen, ja esimerkki 72 edustaa neutraalia ja asiallista suhtautumista.

Myös esimerkkien 73 ja 74 kommenteissa sanavalinnoilla ilmaistaan suhtautumista koodin ammattimaisuuteen.

(73) // **hahmoa liikuttava metodi** (A34)
public static int[][] liikutaHahmoa(int[][] hahmo, char liike, char[][] kentta) {

(74) // **Metodi, joka muuttaa hahmon koordinaatteja halutun käskyn mukaisesti** (A31)
public static int[][] hahmonLiikutus(char[][] merkit, int[][] paikat, char kasky){

Esimerkkien 73 ja 74 tapauksissa suhtautumista ilmaistaan tehdään käyttäen pelin ja koodin todellisuuksien välistä eroavaisuutta, jota käsittelen tarkemmin ideationaalisen metafunktion kannalta luvussa 5.2.1. Molemmat esimerkit ovat metodinesittelykommentteja, joissa kerrotaan, mitä metodi saa aikaan, ja molempien esimerkkien metodit aiheuttavat suurpiirteittäin saman asian. Esimerkissä 74 hahmoon suhtaudutaan elollisena olentona, sillä metodi on *hahmoa liikuttava*. Esimerkissä 73 puolestaan kuvataan metodin toiminta teknisten toteutustapojen kautta, sillä siinä *muutetaan hahmon koordinaatteja*, jossa *koordinaattien muuttamisella* viitataan samaan tapahtumaan kuin esimerkin 74 *liikkumisella*. Nämä suhtautumistavat paitsi rakentavat peli- ja koodimaailmoja, osoittavat esimerkkien 71 ja 72 tavoin vakavuusasteen, jolla koodiin suhtaudutaan. Koodi voidaan näyttää teknisenä ja ammattimaisena kokonaisuutena sen avulla, että käytetään tarkkoja ja sävyttömiä sanavalintoja, jotka liittyvät ohjelmointiin, kuten esimerkissä 73. Toisaalta suhtautuminen koodiin voi lähteä sen luoman ohjelman kautta, millä osoitetaan suhtautumista koodiin toissijaisena ja peliin ensisijaisena. Esimerkissä 74 suhtautuminen on kevyempää ja vähemmän ammattimaista kuin esimerkissä 73, mutta se ei kuitenkaan aiheuta esimerkin 71 tavoin leikittelevää ja hulluttelevaa vaikutelmaa.

Esimerkin 75 kommentissa käytetään sanoja *hahmon pää* ja *niska*, joten tilanne on näiltä osin samankaltainen pelihahmon ensisijaisuudessa kuin esimerkissä 74. Sanavalinnat eivät ole esimerkiksi *ensimmäisen solun koordinaatti* ja *toisen solun koordinaatti*, vaan sanat viittaavat hahmoon.

(75) // **Tarkistus jos hahmon pää yrittää mennä "niskan" paikalle.** (A25)
if(!(kehys[riviPaa][sarakePaa] == NISKA)){

Esimerkissä 75 huomion ansaitsevat ennen kaikkea *niska*-sanan kehystämät lainausmerkit. Lainausmerkit sanan ympärillä voivat osoittaa kirjoittajan ironista tai kriittistä asennetta (Kielikello 2/2006: Lainausmerkit ”). Tämän kommentin yhteydessä on selvää, että lainausmerkit ilmaisevat kirjoittajan asennetta, sillä ne ovat vain yhden sanan ympärillä. Sanana *niska* on tässä kommentissa kaikkein tarkin pelihahmon fysiikan määrittelijä, sillä kommentissa myös käytetty sana *pää* voisi viitata ohjelmoinnissa esimerkiksi jonon ensimmäiseen merkkiin, kun taas *niska* tarkoittaa ainoastaan elollisen olennon kaulan takaosaa. Lainausmerkkejä käyttämällä kirjoittaja ehkä haluaa osoittaa, ettei ota pelin hahmoa tosissaan niin, että käyttäisi sen ruumiinosien nimiä vakavissaan koodinsa kommenteissa. Koodi ja kommenttien välittämä ammattimainen vaikutelma ovat niin tärkeitä, että heittomerkkien ympäröimäksi asettamalla osoitetaan, että puheenaiheena on nyt vakavasti otettava koodi, johon on vain helppo viitata käyttämällä hahmon fysiikkaan liittyviä sanoja.

Kommenttien sanavalinnoilla voidaan ilmaista suhtautumista ammattimaisuuteen ja siihen, miten vakavasti koodi otetaan. Ammattimaisuuden ilmaisun lisäksi suhtautuvia sanavalintoja ei aineistossa esiinny, sillä koodin kantaaottamaton kuvaaminen on kommenttien ensisijainen tehtävä, jota suurin osa aineiston kommenttien kirjoittajista noudattaa kuuliaisesti.

5.2 Ideationaalinen metafunktio

Ideationaalisen metafunktion avulla puhuja hahmottaa ympäröivää todellisuutta ja rakentaa siitä oman tulkintansa, eli ideationaaliset merkitykset ovat niitä, joilla konstruoidaan ympäröivää maailmaa (Halliday & Hasan 1985: 45, Luukka 2002: 192). Ideationaalinen metafunktio jaetaan vielä kahtia eksperientaaliseen ja loogiseen, joista ensimmäinen tarkoittaa sisältöä ilmentävää merkitystä ja toinen asioiden välisiä suhteita rakentavaa merkitystä (Halliday & Hasan 1985: 44–45, Luukka 2002: 102). Ideationaalinen metafunktio heijastaa tekstin alaa, eli sitä, mitä kielenkäyttötilanteessa tapahtuu: millainen sosiaalinen tilanne on kyseessä ja mitä aihetta käsitellään (Halliday & Hasan 1985: 25, Halliday 1985: 33).

En tee tässä työssä varsinaista eroa eksperientaalisen ja loogisen metafunktion välillä, vaan tämä luku käsittelee yleisellä tasolla ohjelmointikoodin kommenttien ideationaalisia merkityksiä kantavia piirteitä. Pyrin osoittamaan, millä tavalla kommentit osallistuvat ohjelmointikoodin merkitysten rakentamiseen, minkä teen kuvailemalla erilaisia maailmoja, joita kommentit luovat. Osoitan myös lyhyesti, miten erityissanasto merkitsee kommenttien taustalle rakentuvan oman diskurssiyhteisönsä.

5.2.1 Kommentit luomassa koodimaailmaa

Ohjelmointikoodia kirjoitettaessa koodi ohjaa sitä, miten ja mitä kommentoidaan (ks. luku 4.1). Ohjelmointikooditiedosto on kuitenkin tietynlainen tekstikokonaisuus, jossa mitkään osat eivät ole tyystin irrallisia toisistaan, joten on mahdollista huomata, että myös kommentit ovat omalta osaltaan rakentamassa koodikokonaisuuden luomaa todellisuutta. Ne eivät vaikuta konkreettiseen koodiin, sillä koodin käytänteet ovat vakiintuneita, tarkkoja ja riippumattomia luonnollisesta kielestä, mutta koodikokonaisuutta lukevalle ihmiselle ne voivat rakentaa käsityksiä ja mielikuvia koodista. Määritteleehän Kielitoimiston sanakirja kommentin selitykseksi (Kielitoimiston sanakirja: kommentti), joten voi olettaa, että ohjelmointikoodissa kommentti avaa koodin tarkoitusta tavalla tai toisella.

Hallidayn ideationaalinen metafunktio tarkoittaa sitä, että kielellä pyritään kuvaamaan kokemus- ja mielikuvitusmaailmaamme (Halliday & Hasan 1985: 44–45, ks. myös Shore 2012b: 161). Luonnollisella kielellä kirjoitetun kommentin yksi pääasiallisista tarkoituksista onkin juuri se, että kuvaa koodin luomasta maailmasta rakennetaan todelliseksi kielen avulla. Ajatus mielikuvitusmaailmasta toimii hyvin tässä tapauksessa, sillä maailma, jota kielellä luodaan, ei ole käytännön tasolla todellinen. Se on olemassa vain tekstitiedostossa sijaitsevina algoritmeina, jotka synnyttävät pieneksi hetkeksi pelin ja siihen liittyvän maailman. Kommenttien kieli rakentaa tätä maailmaa silloinkin, kun lukija ei ymmärrä kaikkia algoritmeja tai näe varsinaista peliä.

Kommenttien luomia suhtautumistapoja koodiin voi karkeasti sanoa olevan kaksi. Joko kommentit suhtautuvat koodiin sen rakentaman kuvitteellisen pelimaailman kautta tai ne suhtautuvat koodiin teknisenä koodina. Useasti nämä kaksi sekoittuvat kommenteissa, jolloin kommentit ohjaavat näkemään tietyt asiat koodin näkökulmasta ja tietyt asiat pelimaailman näkökulmasta. Kommentit voivat myös ottaa jonkin tapahtuman kohdalla huomioon molemmat näkökulmat. Seuraavissa luvuissa esittelen nämä kolme erilaista tapaa rakentaa maailmoja: aloitan tapauksista, joissa pelimaailma on ensisijainen, minkä jälkeen siirryn käsittelemään koodimaailma edellä esiintyviä kommentteja ja tämän jälkeen esittelen omassa luvussaan vielä molempia maailmoja yhdistäviä kommentteja.

5.2.1.1 Pelimaailma ensisijaisena

Esimerkkien 76–78 kommentit esittävät koodin pelimaailman näkökulmasta. Niissä ei selitetä koodirakenteen toimintaa tai käytetä teknisiä ohjelmointiin liittyviä ilmauksia, vaan koodia kommentoidaan aivan kuin sen luoma pelimaailma olisi todellinen ja ensisijainen. Kommentit synnyttävät mielikuvan pelistä, joka on koodin visuaalinen ja konkreettinen lopputulos ja jonka hahmojen toimintaa on mahdollista seurata.

(76) // **Tarjoillaan ruoka.** (A42)
```
Automaatti.tarjoile(merkkitaulukko);
```

Esimerkin 76 kommentissa *tarjoillaan ruoka*, eikä kommenttia voisi mistään sisällöllisestä piirteestä yhdistää Java-koodiin ilman kontekstiaan. Kommentissa luodaan kuvaa pelimaailmasta, jossa ruoka ei vain ilmesty kentälle, vaan se on jonkun tarjoilema – kommentti on metafora, jossa kommenttipisteen tulostamista pelikentälle verrataan ravintolassa tapahtuvaan tarjoilutilanteeseen. Kommentissa ei rakenneta kuvaa ainoastaan koodin visuaalisesta todentumasta eli pelistä, joka tulostuu näytölle, kun käyttäjä pelaa, vaan siinä lisätään mielikuvia vielä konkreettisen pelitodellisuuden päälle. *Ruoan tarjoileminen* johtaa ajatukseen siitä, että tilanteessa on läsnä tarjoilija. Näin tapahtuu siitä huolimatta, että passiivimuotoinen predikaattiverbi jättää tekijän näennäisesti puuttumaan, sillä passiivilauseessa subjekti jää taka-alalle (VISK § 1313). Passiivimuoto ei ole ohjelmointikoodin kommentissa tunnusmerkillinen piirre eikä myöskään tarkoita, että tekijättömyyttä pyrittäisiin korostamaan (ks. luku 5.1.1). Implisiittiseksi tekijäksi tässä kommentissa voi nähdä ruokaa tarjoilevan tarjoilijan, sillä lauseen tosiasiallinen tekijä voi olla kontekstin perusteella passiivilauseessakin yksiselitteinen (VISK § 1325). Kommentin kielellä pelitodellisuuden päälle rakennetaan vielä yksi todellisuus, jossa tarjoilija tuo hahmolle ruoan pelikentälle.

Esimerkin 77 kommentissa metafora ei ole yhtä selkeä, mutta kommentti ei anna teknistä kielellistä kuvausta ainoastaan sitä seuraavalle for-silmukalle ja sijoitusoperaatioille, vaan luo näistä tapahtumista pelimaailman kannalta oleellisen mielikuvan.

(77) // **liikutetaan hahmon häntäosaa niskaan asti** (A34)
```
for (int rivi = hahmo.length - 1; rivi > 0; rivi--) {
    hahmo[rivi][0] = hahmo[rivi - 1][0];
    hahmo[rivi][1] = hahmo[rivi - 1][1];
}
```

Esimerkin 77 tapauksessa puhutaan koordinaattien ja muuttujannimien sijasta *hahmon häntäosasta* ja *niskasta*, jotka aihepiiriltään koskettavat hahmojen biologiaa enemmän kuin ohjelmoinnin tekniikkaa. Pelimaailman kommentointi koodin sijaan voi joissakin tapauksissa helpottaa ja yksinkertaistaa kommentoijan tehtävää, sillä esimerkin 77 sisältö ilman metaforaa voisi olla esimerkiksi seuraavanlainen: ”*// Vaihdetaan hahmo-taulukon paikkoja niin kauan, että kaikki paikat ovat vaihtuneet edelliseen.*”

Esimerkissä 77 koodissa ohjatut tapahtumat sijoitetaan siihen lopputulokseen, jonka koodi saa aikaan, jolloin koodin ulkopuolisen lukijan käsitys siitä, mitä pelissä ja koodissa tapahtuu, parantuu. Tällaisen kommentin avulla koodin ulkopuolisen lukijan ei tarvitse lukea koko koodia ja rakentaa

vasta sen perusteella kuvaa tapahtumasta, joka pelissä koodin myötä tapahtuu. Paitsi nopeuttaa ymmärtämistä, pelimaailmaan totena viittaaminen auttaa myös välttämään virheet, joita koodin lukija mahdollisesti tekisi ilman kommenttia.

Myös esimerkki 78 kommentoi koodia pelimaailman näkökulmasta, sillä siinä puhutaan *hahmon siirtymisestä taaksepäin* kuin kyseessä olisi elollinen olento.

(78) // **Jos hahmo koittaa siirtyä taaksepäin** (A45)
```
else if(hahmo[1][0] == hahmoy && hahmo[1][1] == hahmox){
```

Syynä esimerkin 78 kommentin nostamiselle esimerkiksi on etenkin se, miten hahmo on kommenttilauseen aktiivinen subjekti: *hahmo koittaa siirtyä*. Suurimmassa osassa aineiston kommentteja predikaattiverbi on passiivimuotoinen (ks. luku 5.1.1), jolloin sen implisiittinen tekijä on ainoastaan lukijan pääteltävissä, joten aktiivimuotoinen predikaatti subjekteineen on poikkeuksellinen rakenne. Syy tällaisen rakenteen käytölle voi löytyä subjektista, jonka aktiivisuutta kommentin kirjoittaja haluaa korostaa. Kun *hahmo koittaa siirtyä*, ei tietokoneella, koodaajalla tai pelin pelaajalla ole lauseen rakenteessa osaa hahmon siirtymiseen, eikä heidän voi ajatella edes implisiittisesti liittyvän lauseeseen, vaikka käytännössähän hahmon ensisijaisen siirtymisen hoitavat koodaaja ja tietokone ja toissijaisen siirtymisen pelin pelaaja. Näin kommentoimalla pelimaailmasta tehdään todentuntuista ja sen merkitystä koodiin verrattuna korostetaan.

5.2.1.2 Koodimaailma ensisijaisena

Kommentit voivat myös jättää pelimaailman liki kokonaan huomiotta, jolloin kommentoinnin kohteena on ensisijaisesti vain koodi ja koodissa tapahtuvat muutokset. Esimerkkien 79 ja 80 kommenttien koodikeskeisyys tulee esiin etenkin verrattaessa aiempiin esimerkkeihin 76–78, joissa keskiössä on pelimaailman todellisuus. Lopullisessa pelissä koodin ansiosta tapahtuvia asioita ei esimerkeissä 79 ja 80 juuri selvennetä.

(79) //**Luodaan aputaulu päivittämistä varten** (A10)
```
int[][] aputaulu = new int[RIVILKM][SARAKELKM];
```

Esimerkissä 79 on tyypillinen tapaus kommentista, jossa ainoa kommentoinnin kohde on koodimaailma ja pelin maailma jätetään tyystin huomiotta. Kommentin keskeinen termi *aputaulu* on ainoastaan koodiin kytköksissä, sillä *taulu*-sana on lyhenne ohjelmoinnin rakenteesta nimeltä *taulukko*, eikä koodin rakentamassa pelissä esiinny *aputaulu*-nimistä osaa. *Päivittäminenkin* liittyy vain koodin osuuteen pelissä, sillä peli päivittyy jatkuvasti jokaisen siirron jälkeen, mutta sitä ei kutsuta pelimaailmassa päivittämiseksi, joka rikkoisi illuusion sujuvasta ja ohjailtavasta pelihahmosta. Kom-

mentti ei sanavalintojensa tai merkitystensä osalta ota mitenkään kantaa siihen, miten peli liittyy siihen, että *luodaan aputaulu päivittämistä varten*.

Tyypillisen koodimaailman ensisijaisena esittävän kommentin esimerkin 79 kommentista tekee se, ettei sen kommentoima koodi liity pelin maailmaan mitenkään. Jotta pelissä saadaan rakennettua koodilla tiettyjä toimintoja, on koodilla pakko tehdä sellaista pohjatyötä, joka ei suoraan ole kytköksissä peliin ja sen maailman tapahtumiin. Tässä tapauksessa kommentin kommentoima koodi on viemässä koodia siihen suuntaan, jota pelimaailmassa toivotun tapahtuman toteutuminen vaatii, eikä siihen täten voisi edes kytkeä pelimaailman näkökulmaa. Jos niin tehtäisiin, pitäisi kommentin kattaa taulukon luonnin lisäksi huomattavasti enemmän koodia, sillä pelin kannalta oleellinen osa seuraa koodissa vasta myöhemmin. Tällaisessa tilanteessa se, että lukijalle rakennetaan kuvaa yksinomaan koodimaailmasta, on luonnollista. Koodin näkökulma on ensisijainen ja myös ainoa tapauksessa, jossa peliä ei voi suoraan kytkeä koodiin mitenkään. Kommentin kirjoittaja voi käyttää tällaista kielellistä keinoa osoittamaan lukijalle, että seuraava koodi on oleellinen vain ja ainoastaan koko koodin toimimisen kannalta, eikä lukijankaan tarvitse rakentaa koodia lukiessaan mielikuvaa pelin maailmasta.

Esimerkin 80 kommentin sanasto on ohjelmointipainotteista: käsitteet *komento, sijoittaa, arvo* ja *muuttuja* liittyvät Java-ohjelmoinnin tekniikkaan, ja kyseiset sanavalinnat voisivat olla mistä tahansa koodista. Juuri tämän pelin tarpeisiin on osoitettu kommentin sanastosta *komento* ja *liikkumismuuttuja*, mutta peliä tarkemmin tuntemattomalle ne eivät anna juurikaan lisätietoa ohjelman toiminnasta, sillä samanlaisia sanavalintoja voisi olla todella monenlaisissa muissakin ohjelmissa.

```
(80) // Riippuen komennosta sijoitetaan tietty arvo liikkumismuuttujille. (A25)
     if(komento == VASEN)
         sarakeMuutos = -1;
     …
```

Esimerkin 80 kommentin loppuosa *sijoitetaan tietty arvo liikkumismuuttujille* koskettaa ainoastaan sitä seuraavassa koodissa tapahtuvia asioita, ei sitä, mitä koodissa tapahtuvat asiat tarkoittavat. Koodia tulkitsemalla kommentin loppuosa tarkoittaa pelimaailman kannalta sitä, että hahmoa ruvetaan siirtämään haluttuun suuntaan, mistä kommentissa ei kielellisesti anneta mitään vihjettä. *Riippuen komennosta* voi sitoa kommenttia löyhästi pelitodellisuuteen, sillä pelissä pelaaja antaa hahmon liikkumissuunnan määräävän komennon, mutta on paljon mahdollista, että sillä viitataan kommenttia seuraavan ehtorakenteen ehdossa esiintyvään *komento*-muuttujaan. Näin ollen tismalleen koodin rakentaman pelin todellisuuteen kommentissa ei viittaa suoraan ja tarkkaan mikään, vaan ensisijaista on kommentin jälkeen seuraava Java-koodi.

Oletusarvo tällaisessa kommentissa on, että lukija ymmärtää pelin ja koodin yhteyden toisiinsa, jolloin hänelle riittää tieto siitä, mitä pelkässä koodissa juuri tässä yhteydessä tapahtuu. Lukijaksi oletetaan siis toinen diskurssiyhteisön jäsen (ks. Swales 1990: 24–25). Lukijan kannalta tällainen ratkaisu ei auta ymmärtämään koodia yhtä hyvin, kuin pelimaailman toimintojen osoittaminen auttaisi, sillä työtä koodia tulkiten pitää tehdä kommentista huolimatta. Kommentti ei esitä koodin yksityiskohtia niin tyhjentävästi, ettei itse koodia tarvitsisi lukea. Toisaalta kommenttien tarkoitus ei olekaan korvata koodia, vaan helpottaa koodin ymmärtämistä ja nopeuttaa sen lukemista. Tässä kommentissa sen kirjoittaja on pitänyt lukijalle hyödyllisempänä tietoa siitä, mitä koodissa konkreettisesti tapahtuu, kuin siitä, mitä se pelin kannalta tarkoittaa.

5.2.1.3 Kahden maailman yhdistelmä

Useissa kommenteissa ei ole tehty valintaa koodimaailman ja pelimaailman välillä, vaan niissä esitellään asiat molemmista näkökulmista. Tästä ovat esimerkit 81–83, joissa rakennetaan kielellisillä valinnoilla sekä koodi- että pelimaailmaa.

(81) **/* Lisätään hahmomerkit kenttätaulukkoon */** (A27)
addSnakeToStage(snake, stage);

Esimerkissä 81 kohdistetaan kommentti koodiin käyttäen määrittelyssä apuna peliin liittyviä käsitteitä. Termit *hahmomerkki* ja *kenttätaulukko* osoittavat hahmon ja kentän tyypit koodin kannalta, sillä *merkki* ja *taulukko* ovat kiinteästi ainoastaan koodiin liittyviä sanoja, eikä niitä pelkän pelin näkökulmasta kommentoitaessa tarvittaisi lainkaan. Pelimaailman kannalta taas riittäisivät ainoastaan yhdyssanojen määriteosat *hahmo* ja *kenttä*, jotta kommentin merkitys pysyisi jotakuinkin samana (*/* Lisätään hahmo taulukkoon */*), sillä pelin kannalta oleellista olisi ainoastaan pelissä liikkuva *hahmo* ja *kenttä*, jolle se sijoittuu. Kommentissa yhdyssanojen määriteosat siis viittaavat käsitteisiin, joita ei minkä tahansa Java-ohjelman kommentoinnissa käytetä, mistä syystä esimerkin 81 kommentin liittyminen juuri kyseessä olevaan peliin on selkeämpää kuin tapauksessa, jossa huomioidaan ainoastaan koodimaailman näkökulma.

Kommentin tekninen, koodia korostava sisältö olisi ollut ymmärrettävä myös ilman yhdyssanojen määriteosia (*/* Lisätään merkit taulukkoon */*), mutta kommentin kirjoittaja on nostanut molemmat näkökulmat esiin yhdistämällä peli- ja kooditodellisuuden yhdyssanojen avulla. Vaikutelma, joka näistä yhdyssanoista syntyy, on kuitenkin tekninen. Pelimaailma jää hieman sivummalle koodimaailmaan nähden, kun peliin liittyvät käsitteet *hahmo* ja *kenttä* ovat vain osana määrittelemässä koodin kannalta oleellisia termejä. Kommentoija on valinnut kommenttiin lähtökohdan, joka esittää ohjelmointikoodin asteen verran tärkeämpänä kuin sen pelin, jonka se luo. Kommentissa

maailmojen epäsymmetrisestä huomioinnista huolimatta otetaan sekä pelin että koodin näkökulma huomioon.

Esimerkin 82 kommentin ensimmäisessä lauseessa luodaan mielikuva tapahtumasta, joka sijoittuu pelimaailmaan, ja toisessa lauseessa kuvataan, mitä se tarkoittaa koodimaailman kannalta.

(82) **//Pää liikkuu vasemmalle eli vähennetään sarakkeesta 1.** (A22)
int sarakeliikkuu = paikkaXsarakkeella - 1;

Esimerkin 82 ensimmäisen lauseen subjektina toimii *pää*, joka jo itsessään sitoo osan kiinteästi peliin ja kuvitteellisen hahmon liikkumiseen pelikentällä, sillä *päällä* viitataan pelihahmon ruumiinosaan. Vasemmalle liikkuminen on sekin mahdollista vain pelin maailmassa, ei tekstimuotoisessa koodin maailmassa, jossa asiat luetaan karkeasti ajateltuna vasemmalta oikealle ja ylhäältä alas. Kommentin alussa rakennetaan lukijalle kuva siitä, että seuraavalla rivillä seuraava koodi aiheuttaa pelin näkökulmasta hahmon pään liikkumisen vasemmalle.

Kommentin toisessa osassa kuvataan luonnollisella kielellä tarkalleen sama asia, joka tehdään seuraavan rivin koodissa, sillä *vähennetään sarakkeesta 1* on suora kielellinen vastine seuraavan rivin koodille *paikkaXsarakkeella - 1*. Kommentin ja koodin välisen semanttisen vastaavuuden lisäksi sen tunnistaa koodimaailmaa kuvaavaksi passiivimuotoisesta teknisestä predikaattiverbistä *vähentää*, joka sitoo sen koodissa tapahtuvaan vähennysoperaatioon. Sana *sarake* esiintyy sekä kommentissa että koodin muuttujannimissä (*sarakeliikkuu* ja *paikkaXsarakkeella*) ja se on osa Javan taulukkorakenteen termistöä. Ohjelmointikoodin kommenteille tyypillinen predikaatin passiivimuotoisuus (ks. luku 5.1.1) esiintyy vain koodimaailmaa rakentavassa osassa, mikä osaltaan osoittaa, että tämä osa kytkeytyy juuri koodimaailmaan.

Oleellista esimerkissä 82 on se, miten sulavasti yhteen kommenttiin on yhdistetty molempien maailmojen näkökulma. Keino, jolla tämä saadaan aikaan on kahden päälauseen yhdistäminen toisiinsa *eli*-rinnastuskonjunktiolla. Tällaisessa rinnastuksessa konjunktio luo kytkettyjen elementtien välille semanttisen tasapainon (VISK § 1080), ja sehän kommentin kahden osan rinnastuksen tarkoituksena lieneekin, siis esittää sama asia kahdesta eri näkökulmasta siten, että sisältö viittaa molempien näkökulmien kannalta samaan tapahtumaan. Kommentin lauseet eivät ole rakenteeltaan samanlaiset, sillä ensimmäinen on muotoa subjekti + predikaatti + adverbiaali, ja toinen koostuu passiivimuotoisesta predikaatista, adverbiaalista ja objektista. Tämä ei kuitenkaan estä niiden semanttista vastaavuutta, sillä maailmat, joita ne kuvaavat, ovat niin erilaiset, ettei niiden kuvaaminen tismalleen samanlaisilla lauserakenteilla olisi välttämättä edes järkevää.

Kommentoijan ei ole tarvinnut tehdä valintaa koodimaailman ja pelimaailman kuvaamisen välillä, vaan on valinnut käytettäväkseen rakenteen, jossa molempien maailmojen valaiseminen

lukijalle onnistuu. Kuten aiemmista esimerkeistä voi havaita, olisi kommentissa voinut valottaa maailmoista vain toista, mutta tässä molemmat on otettu tarkoituksella huomioon. Syynä sille, ettei kommentissa rakenneta ainoastaan pelimaailmaa, jolloin kommentti olisi vain *//Pää liikkuu vasemmalle*, voi olla kommentoijan halu sitoa kommentti vahvasti seuraavaan koodiriviin. Näin tekemällä ei jää epäselväksi, minkä verran koodia kommentti kattaa, kun sen esittämä tapahtuma toteutuu heti yhden koodirivin aikana. Koodin ja kommentin loppuosan yhteys on ilmiselvä Java-kieltä ymmärtävälle, sillä koodirivi on todella yksinkertainen. Kommentissa molemmat maailmat yhdistämällä lukijalle selvitetään, mitä pään liikkuminen vasemmalle koodissa tarkoittaa.

Esimerkissä 83 on esimerkin 82 tavoin kaksi osaa, mutta osat ovat eri järjestyksessä. Kommentissa huomioidaan ensin koodimaailma ja vasta tämän jälkeen pelimaailma.

(83) // **Muuten lippu vaihtuu ja hahmo kuolee.** (A46)
```
else {
    return true;
}
```

Esimerkin 83 kommentin *muuten*-alkuisuus johtuu siitä, että kommentti jatkaa sisällöllisesti edellisen kommentin ajatusta, mutta se ei ole maailmojen rakentumisen kannalta oleellista, joten jätän *muuten*-alun tässä yhteydessä ilman sen suurempaa huomiota. Kommentin ensimmäinen lause rakentaa selkeästi ainoastaan koodimaailmaa, sillä sen sisältö viittaa kaikin puolin tiettyyn Javan koodirakenteeseen. Ensimmäisen lauseen subjekti on *lippu*, jolla viitataan Java-ohjelmoinnin lippumuuttujaan. Lippumuuttuja voi olla joko tosi tai epätosi, jolloin *vaihtua*-predikaatti ei tarvitse määritettä siitä, mihin lippu vaihtuu, sillä vaihtoehtoja on aina vain yksi. Lopullisen varmuuden sille, että kyseessä on ainoastaan lippumuuttujarakenne, antaa se, ettei koodin toteuttamaan pelimaailmaan liity minkäänlaista lippua.

Kommentin toinen lause muuttaa näkökulman pelimaailmaan, sillä lauseen subjektina on pelin *hahmo*. Tämä subjekti yhdessä predikaattiverbin *kuolla* kanssa luo kommentin lukijalle voimakkaan mielikuvan pelimaailman tapahtumasta, jossa hahmo menettää henkensä. *Kuolla*-verbillä tarkoitetaan elämästä lakkaamista, jolloin kuoleminen voi tapahtua ainoastaan elolliselle olennolle tai asialle, eli esimerkiksi *lippumuuttujan kuolemasta* ei olisi luontevaa kommentoida. Tämän takia verbi liittää lukijan ajatukset pelimaailmaan, jossa mahdollisia ovat sellaiset asiat, joita teknisessä koodissa ei voisi tapahtua. Jos kommentin lukijalla on tieto siitä, miten peliä pelataan, hänen ei tarvitse edes tehdä päättelyä *kuolla*-verbin mahdottomuudesta viitata koodiin, sillä hahmon kuoleminen on pelin kannalta keskeistä. Sen lisäksi, että kommentin loppuosan kielen avulla rakennetaan sen lukijalle vahva kuva juuri pelimaailmasta, annetaan vaikutelma hahmosta elollisena olentona.

Olisi lähestulkoon vastaava asia sanoa *peli päättyy* kuin *hahmo kuolee*, sillä hahmon kuoleman aiheuttama lopputulos pelin kannalta on pelin päättyminen. Kommenttiin on kuitenkin valittu pelimaailman sisältä hahmon näkökulma, jolloin *kuolla*-verbillä osoitetaan tapahtuman traagisuus kuolevan hahmon näkökulmasta.

Esimerkissä 83 on yhdistetty kommentin yhteen virkkeeseen selkeä koodimaailman ja selkeä pelimaailman näkökulma, kuten edellisessäkin esimerkissä. Esimerkissä 82 tämä yhdistäminen tehdään kuvaamalla yksi tapahtuma kahden maailman näkökulmasta *eli*-konjunktion erottamana, mutta esimerkin 83 tapaus ei ole yhtä suoraviivainen. Esimerkin osat eivät viittaa samaan tapahtumaan kahdesta näkökulmasta, vaan ne viittaavat kahteen eri aikaan sijoittuvaan tapahtumaan, joista toinen kuvataan koodin ja toinen pelin näkökulmasta. Tässäkin esimerkissä eri maailmoja rakentavat päälauseet erotetaan toisistaan rinnastuskonjunktiolla. *ja*-konjunktion avulla esitetään lauseiden välinen aikasuhde: ensin *lippu vaihtuu* ja sitten *hahmo kuolee*, mistä johtuu järjestys, jossa siirrytään koodimaailmasta pelimaailmaan. Tapahtumajärjestys ohjelmoitaessa on se, että jokin tapahtuma täytyy ensin kirjoittaa koodiin, jotta se voi tapahtua itse ohjelmassa, joka tässä tapauksessa siis on peli. Kommentilla paitsi rakennetaan näitä molempia maailmoja, myös asetetaan ne suhteeseen toistensa kanssa. Kommentin lauseet ovat syntaktisilta ominaisuuksiltaan samanarvoiset, sillä molemmat muodostuvat yksinkertaisesti samalla tavalla subjektista ja predikaatista. Tästä huolimatta kommentilla osoitetaan, että pelimaailma on ainakin tässä tapauksessa alisteinen koodimaailmalle.

Eroja siinä, millaista tekstitodellisuutta kommenteilla rakennetaan, voi pyrkiä selittämään koodilähtöisesti. Kohdissa, joissa kommentoidaan pelimaailman näkökulmasta, tapahtuu jotain pelin kannalta oleellista ja kohdissa, joissa kommentoidaan ainoastaan koodia, ei välttämättä ole suoraan osoitettavaa vaikutusta itse peliin. Monessa tapauksessa molemmat maailmat otetaan kommentissa huomioon, jolloin tapahtuma voi olla merkittävä sekä koodin että pelin näkökulmasta, tai maailmojen keskinäistä suhdetta halutaan korostaa.

5.2.2 Erityissanasto diskurssiyhteisön merkkinä

Shore (2012: 163) tarkastelee tekstin ideationaalisen metafunktion yhtenä osana tekstin leksikaalisia valintoja, joten otan kommenttien sanastolliset ominaisuudet yhdeksi osaksi ideationaalista metafunktiota. En käsittele tässä luvussa kaikenlaisia kommenttien maailmaa rakentavia sanavalintoja, vaan keskityn kommenteissa esiin nousevaan erityissanaston käyttöön, sillä erityissanasto muodostaa selkeimmän ja yhtenäisimmän leksikaalisten valintojen joukon kommenteissa.

Suuri syy erityissanaston nostamiselle omaan lukuunsa on se, että Swales nimeää erityissanaston olemassaolon yhdeksi diskurssiyhteisön kriteeriksi, mikä tarkoittaa, että ryhmää ei voida kutsua

diskurssiyhteisöksi, mikäli sillä ei ole omaa sanastoaan käytettävissään. Erityissanastoa eli tarkkoja ja teknisiä termejä käytetään, jotta kommunikointi alan asiantuntijoiden välillä olisi mahdollisimman tehokasta, ja erityissanastoksi lasketaan yleensä sellainen sanasto, jota ulkopuoliset, diskurssiyhteisöön kuulumattomat eivät ymmärrä. (Swales 1990: 24–25.) Tässä luvussa osoitan esimerkkien avulla, millaista kommenttien erityissanasto on, ja miten se perustelee kommentoijien kuulumista omaan diskurssiyhteisöönsä.

Aivan jokaisessa aineiston kommentissa ei välttämättä käytetä erityissanastoa, mutta sitä esiintyy tasaisesti läpi koko aineiston sellaisissa paikoissa, joissa sen käyttö helpottaa koodiin viittaamista. Se ei myöskään ole vain joidenkin kommentoijien käyttämä keino, vaan erityissanasto on osana jokaisen metodin kommentistoa, eli jokainen aineiston ohjelmointikoodin kirjoittaja on osannut ottaa erityissanaston osaksi kommenttejaan. Olen alleviivannut esimerkeistä 84 ja 85 erityissanaston piiriin lukeutuvat sanavalinnat.

(84) // **For-silmukka syötetaulukon häntää edeltävien koordinaattien korvaamiseen.** (A39)
```
for (int i = pituus - 1; i > 0; i--) {
    paikat[i][0] = paikat[i-1][0];
    paikat[i][1] = paikat[i-1][1];
}
```

(85) //**Lippumuuttuja jonka ollessa true, hahmoa voidaan siirtää.** (A22)
```
boolean valmisliikkumaan = false;
```

Esimerkissä 84 erityissanastoa edustavat sanat *for-silmukka* ja *syötetaulukko*, ja esimerkissä 85 sanat *lippumuuttuja* ja *true*. Tällaisten sanavalintojen avulla kommentteja kytketään täsmällisesti ohjelmoinnin ja koodin alaan kiinni ja osoitetaan, että kommentti liittyy juuri siihen tilanteeseen, jossa kommentoidaan. Näiden sanojen käyttö ohjelmointikoodista irrallisessa kontekstissa – esimerkiksi sanomalehden artikkelissa – olisi harvassa tapauksessa perusteltua, sillä niiden merkitys valjennee ainoastaan ohjelmoinnin tekniikkaan perehtyneelle. Erityissanaston avulla kuva ohjelmointimaailmasta rakentuu tehostetusti, sillä ilman sen käyttöä koodin kommentointi olisi monimutkaisempaa ja tehottomampaa.

Tällaisten sanavalintojen avulla syntyy yhteneväisyys kaikkien aineiston kommenttien välille, sillä erityissanastoa esiintyy kaikkien koodaajien kommenteissa ainakin jonkin verran. Kommentit käsittelevät samaa aihetta eli Java-koodia, ja ainoa aiheen osoittava tekijä ei ole niiden sijainti koodin seassa, vaan myös kyseisen alan erityissanasto kytkee kommentteja toisiinsa. Niissä on saman alan sanastoa, jota ei voisi käyttää missä tahansa tekstissä. Koodin kommentit eivät ole ainoa paikka, jossa alan erityissanastoa käytetään, sillä esimerkiksi ohjelmoinnin perusteoksissa ja lausekieli-

sen ohjelmoinnin peruskurssin luentokalvoilla käytetään samoja teknisiä sanoja (ks. esim. Wikla 2003), mutta erityissanasto rajaa kommentteja pienemmälle ja tarkemmalle alalle.

Alaan liittyvien sanojen runsaahkolla käytöllä osoitetaan, että oletusarvoisena lukijana on joku, joka ymmärtää ohjelmoinnista niin paljon, etteivät tekniset sanavalinnat tuota ymmärtämisvaikeuksia. Tämä on yksi keino rakentaa ja ylläpitää diskurssiyhteisöä, sillä erityissanastoa käyttämällä ulkopuolelle jätetään aina sellaiset, jotka eivät yhteisöön kuulu (Swales 1990: 25). Esimerkiksi esimerkki 85 aukeaa vain lukijalle, joka tietää, mikä yhteys on termillä *lippumuuttuja* ja seuraavan rivin *boolean*-alkuisella muuttujalla. Lukijan pitää myös kyetä yhdistämään kommentissa esiintyvä sana *true* boolean-muuttujaan. Sanaston avulla siis paitsi luodaan erityisesti ohjelmointiin liittyvää ilmapiiriä, myös rajataan lukijakuntaa, jolle kommenttien kieli on osoitettu.

Tämä lukijakunnan rajaus on pääteltävissä jo niiden tarkoituksesta ohjelmointikoodin selittäjinä, mutta ideationaalisen metafunktion kannalta on tärkeää huomata, että rajausta luodaan kielellistenkin seikkojen avulla. Vaikka suurin osa aineiston koodien kirjoittajista on vasta ohjelmointiuransa alkutaipaleella, osoittavat he erityissanastoa käyttämällä, että ovat jo sitoutuneet osaksi ohjelmoinnin akateemista yhteisöä[8]. He kykenevät käyttämään ohjelmoinnin erityissanastoa tilanteissa, joissa se kytkee kommentit osaksi koodia, kuten esimerkeissä 84 ja 85.

Ohjelmoinnin erityissanaston avulla sidotaan kommentit erottamattomaksi osaksi ohjelmointikoodia sekä kaikkea siihen liittyvää kirjoitusta, ja sen yksi keskeinen tehtävä on osoittaa tarkalleen, mihin kommentit liittyvät. Sanasto toimii osana rakentamassa akateemisen yhteisön yhteistä maailmaa, jonka täydellinen ymmärtäminen on mahdollista ainoastaan yhteisön jäsenille. Erityissanaston käyttäminen on keino osoittaa, että sen käyttäjä on osana diskurssiyhteisöä, ja se myös rajaa lukijaksi vain diskurssiyhteisön toisen jäsenen, joka on yhtä kykeneväinen ymmärtämään erityissanaston merkityksiä (Swales 1990: 25).

5.3 Tekstuaalinen metafunktio

Tekstuaalisen metafunktion näkökulmasta tarkastellaan yleensä erilaisia koheesiokeinoja, jotka voidaan jakaa leksikaalisisiin ja kieliopillisiin, sekä teemankulkua (Shore 2012b: 180–181). Teemankululla tarkoitetaan sitä, miten tekstiin tuodaan uusia tarkoitteita ja miten niihin viitataan (Shore 2012b: 180). Teemat sidotaan aiemmin sanottuun ja reemoissa kehitellään tekstin varsinaista aihetta eteenpäin, jolloin reemoista voi syntyä uusia teemoja (Shore 2008: 40). Teemankulun tutkiminen on ohjelmointikoodin kommenttien osalta melko sisällötöntä, sillä kommentit eivät luo varsinaista

[8] Akateemiseen yhteisöön ajatellaan yleensä kuuluvaksi yliopistojen piirissä toimivat henkilöt opetushenkilöstöstä opiskelijoihin (Luukka 1995: 74).

tekstikokonaisuutta keskenään (ks. luku 4.3). Kommenttien järjestys on tyystin riippuvainen koodin järjestyksestä, eli yhteistä kommenttien tekstilajin informaatiorakennetta on mahdotonta pyrkiäkään etsimään, sillä koodilla voi saada aikaan lähes mitä tahansa. Jos jonkinlainen rakenne löytyisikin, olisi kyseessä ohjelmointikoodin informaatiorakenne, ei kommenttien informaatiorakenne. Kommenteista kuitenkin löytyy erilaisia koheesiokeinoja sekä pienimuotoista jatkuvuutta, joten koheesiokeinojen huomioiminen on kommenttienkin osalta järkevää.

Kiinnitän huomioni siihen, miten leksikaalisia ja kieliopillisia sidoksia ilmenee kommenttien rajojen yli, eli miten koodikokonaisuuksissa viitataan kommentista toiseen. Tällaisen rajauksen tekeminen on mielekästä kommenttien tekstilajin hahmottamisen kannalta, sillä monessa kohdassa nousee esiin kysymys siitä, miten kommentit muodostavat tekstejä – siis onko tutkittava teksti yksittäinen kommentti vaiko kommenttien muodostama kokonaisuus. Kohesiiviset sidokset kommenttirajojen yli osoittavat, että kommentit ovat kytköksissä toisiinsa myös kommenttirajojen yli.

Kommenttien väliset kohesiiviset keinot ovat melko vähäisiä. Esittelen yhden metodin kaikki kommentit esimerkissä 86 havainnollistamaan sitä, kuinka toisistaan irrallisia kommentit yleensä ovat. Tämän yhden metodin perusteella ei tietenkään voi tehdä yleistystä koko aineistosta, mutta esimerkki on hyvin tyypillinen kokooma kommenteista ja niiden välisistä ja sisäisistä koheesiokeinoista. Kommentit ovat esimerkissä järjestyksessä ilman niiden välissä sijaitsevia koodirivejä, jotta niiden lukeminen kuin yhtenä tekstinä olisi mahdollista. Ensimmäiset kaksi riviä sisältävät metodinesittelykommentin, ja muuten kukin rivi pitää sisällään oman kommenttinsa.

(86)
```
// Hahmo liikkuu. Alkiot yksi kerrallaan siirretään uusille paikoille. (A9)
// Palauttaa uuden hahmon.
  // Muistin tarkistus.
    // Hahmon pää.
    // Miinustetaan pituutta, koska "unohdetaan" hahmon viimeinen alkio.
  // Palautetaan uusi hahmo.
```

Esimerkin 86 kommentit toimivat kukin itsenäisinä teksteinään, ja niiden välinen kohesiivisuus on vähäistä. Esittelen seuraavissa luvuissa, millaisia kieliopillisia ja leksikaalisia sidoksia kommenttien kokonaisuuksista on mahdollista löytää, sillä aivan sidostamattomia kommentit eivät toisiinsa nähden ole.

5.3.1 Kieliopilliset sidokset

Kohesiivinen kieliopillinen sidos voi toteutua kieliopillisessa sanassa tai morfeemissa, joita ovat esimerkiksi anaforinen pronomini eli sellainen, joka saa tulkintansa aiemmin sanotusta (VISK § 714), anaforinen nolla, kytkentäilmaus eli esimerkiksi liitepartikkeli *-kin* tai konnektiivi *lisäksi,* ja

vertailuilmaus eli esimerkiksi *erilainen* (Shore 2012b: 181). Kieliopillisia sidoksia ei esiinny lähellekään jokaisen metodin kommenteissa, kuten jo esimerkin 86 kommenttikokonaisuudesta voi havaita, mutta niitä kuitenkin löytyy.

Anaforisia pronomineja tai nollia aineistossa ei ole lainkaan viittaamassa kommentista toiseen. Myöskään vertailuilmauksia ei sellaisenaan löydy. Toisistaan irrallisista kommenteista kieliopilliseksi sidosteisuudeksi voi kuitenkin mahdollisesti tulkita löyhemmin perustein muitakin kohesiivisia keinoja kuin yllä esittelemäni. Aloitan kieliopillisten sidosten esittelyn aineistossa näistä Shoren (2012b: 181) nimeämistä keinoista, ja luvun loppupuolella otan käsittelyyn aineistossa tyypilliseksi nousseet muut kieliopilliset sidoskeinot.

Aineistossa kommentteja yhdistetään toisiinsa konnektiivien avulla. Konnektiiveilla osoitetaan lauseiden tai muiden yksiköiden välisiä semanttisia suhteita, ja niiden avulla voidaan kytkeä toisiinsa myös lausetta laajempia tekstikokonaisuuksia (VISK § 820). Tämän takia konnektiivit ovat erinomaisia yhdistämään toisiinsa kommentteja, jotka kyllä usein ovat vain lauseen tai virkkeen mittaisia, mutta joiden kuitenkin voi ajatella olevan hieman lausetta laajempia kokonaisuuksia. Kommentit sijaitsevat eristettyinä edellisistä ja seuraavista kommenteista, ja ne ovat usein järkeviä ilman toisten kommenttien selvennysapua. Esimerkit 87 ja 88 ovat malleja konnektiivien käytöstä koodin kommenteissa.

(87)
```
// Jos hahmo jahtaa omaa häntäänsä, eli pää on liikkumassa viimeiseen hännän paikkaan,
// niin silloin hahmo ei kuole. (A46)
if (hahmo[hahmonPituus - 1][0] == arvo1 && hahmo[hahmonPituus - 1][1] == arvo2) {
}
// Muuten lippu vaihtuu ja hahmo kuolee.
else {
    return true;
}
```

Esimerkin 87 konnektiivi on adverbi *muuten* (VISK § 820). Ensimmäisen kommentin lauseiden välillä on konditionaalinen suhde, jossa ehdon (*Jos hahmo jahtaa omaa häntäänsä*) täytyttyä *hahmo ei kuole*. Toisessa kommentissa kerrotaan *muuten*-konnektiivin avulla, mitä tapahtuu, jos ehto ei täyty, eli toisessa kommentissa täydennetään tietämys ehdon toteutumattomuuden aiheuttamista toimista. Ehdon toteutumattomuus ei välttämättä aiheuta koodissa mitään, mutta tässä tapauksessa koodissa ehtolausetta seuraa sitä täydentävä *else*-ehtolause, joka toteutuu ensimmäisen ehtolauseen toteutumatta jäätyä. Konnektiivin avulla osoitetaan se, että toinen kommentti täydentää ja jatkaa ensimmäisen kommentin sisältöä, ja yhtä lailla myös se, että toinen ehtorakenne on jatkoa ensimmäisen ehtolauseen koodille.

Esimerkissä 88 esiintyvää adverbia *jälleen* ei sellaisenaan luokitella konnektiiviksi, mutta esimerkiksi sana *taas* on konnektiivi, jonka merkityssisältö on lähellä sanan *jälleen* merkitystä, joten tässä tilanteessa myös *jälleen* pystyy nähdäkseni toimimaan konnektiivin tavoin (VISK § 820). Kun käytetään adverbia *jälleen* kuvaamaan jonkin tilanteen toistuvuutta, on edellytyksenä käytölle vähintään yksi aiempi vastaava tilanne (VISK § 651). *Jälleen*-adverbin ollessa kolmannen kommentin ensimmäinen sana (*Jälleen jos ollaan reunalla*) on aiemman esiintymän löydyttävä jostakin varhaisemmasta kommentista. Tämä varhaisempi kommentti löytyy esimerkin ensimmäiseltä riviltä (*Jos ollaan reunalla*).

(88)
```
// Jos ollaan reunalla, siirrytään kentän vastakkaiselle laidalle (A38)
    if (hahmonPaa[1] == sarakkeet) {
        hahmonPaa[1] = 0;
    }
    // Jos ollaan siirtymässä hahmon kaulan päälle, peruutetaan siirto
    if (compareArrays(hahmonPaa, hahmo[1])) {
        hahmonPaa[1] --;
        // Jälleen jos ollaan reunalla, siirrytään takaisin kentän vastakkaiselle laidalle
        if (hahmonPaa[1] < 0) {
            hahmonPaa[1] = sarakkeet - 1;
        }
```

Sen, mihin kommenttiin *jälleen* viittaa esimerkissä 88, tunnistaa kommenttien tismalleen samasta sisällöstä. Kolmannessa kommentissa on ensimmäiseen verrattuna kaksi ylimääräistä sanaa, jotka tekevät kommenteista ulkoisesti toisistaan hieman poikkeavia, mutta sisältö on yhtä kaikki sama (*[Jälleen] jos ollaan reunalla, siirrytään [takaisin] kentän vastakkaiselle laidalle*). Adverbin avulla sidotaan kolmas kommentti ensimmäisen merkityssisältöön ja luodaan jatkuvuutta kommenttien välille. Kommentit voivat muodostaa sellaisen kokonaisuuden, jossa kommentit kommunikoivat keskenään ja vaikuttavat toistensa merkityssisältöihin. Vaikka jatkuvuutta ei osoitettaisikaan konnektiivien tai minkään muun kieliopillisen keinon avulla, voi sellaista vallita ainakin samaan kokonaisuuteen liittyvien kommenttien välillä.

Aineistosta löytyi ainoastaan yksi edustus *-kin*-liitepartikkelista liittämässä kahta kommenttia toisiinsa, ja tämä edustus löytyy esimerkistä 89. Olen jättänyt esimerkin välistä kieliopillisten sidosten kannalta turhat rivit pois ja alleviivannut keskeisen kielenaineksen.

(89)
```
//Jos paikassa mihin hahmo on menossa ei ole ruokamerkkia.   (A37)
    if (kentta[paanRivi][paanSarake] != '+') {

    [14 riviä koodia, 9 riviä kommentteja]

    //Jos paikalla onkin ruokamerkki.
```

```
else {
```

Esimerkissä 89 kohesiivisen kieliopillisen sidoksen kahden eri kohdassa sijaitsevan kommentin välille luo liitepartikkeli *-kin*. Liitepartikkeli *-kin* esiintyy myöntölauseissa, kun taas liitepartikkeli *-kAAn* esiintyy kieltolauseissa, eli voi sanoa näiden elementtien ilmentävän polaarista vaihtelua positiivisen ja negatiivisen välillä (VISK § 1635). Ensimmäisessä kommentissa on kielteinen olla-verbi yhdistettynä sen *ruokamerkki*-predikatiiviin, ja toisessa kommentissa on myönteinen olla-verbi, johon on liittynyt myönteisyyttä ilmaiseva *-kin*-partikkeli, sekä *ruokamerkki*-predikatiivi. Sen, että nämä kaksi kommenttia liittyvät toisiinsa, voi osoittaa kahdella perusteella: myöntö- ja kieltolauseen vastakkainasettelulla, joka on kytköksissä *-kin*-partikkelin ominaisuuksiin, sekä lauseita yhdistävällä *ruokamerkki*-predikatiivilla ja *paikka*-adverbiaalilla (*paikassa* ja *paikalla*). Kommentit ovat kaukana toisistaan, mutta silti niiden välillä selkeä kieliopillinen kytkös. Kommenttien välissä on yhteensä 23 riviä muuta asiaa, mikä osoittaa, että kommentit voivat liittyä toisiinsa yli kommenttirajojen huolimatta välissä olevista sidokseen liittymättömistä kommenteista. Vaikka aineistossa on ainoastaan tämä yksi *-kin*-partikkelilla rakennettu kommenttirajat ylittävä sidos, on se tärkeä osoitus siitä, että kommentit voivat liittyä toisiinsa ja ne muodostavat jonkinlaisen tekstikokonaisuuden myös muiden kuin välittömässä läheisyydessään sijaitsevien kommenttien kanssa.

Tekstin kieliopillinen sidosteisuus on helposti rajattavissa edellä kuvaamani kaltaisiin elementteihin tekstimäisessä tekstilajissa, kuten Shoren artikkelin esimerkkinä toimivassa oppikirjatekstissä (Shore 2012b: 162). Ohjelmointikoodin kommentit ovat hyvin erilaisia, sillä niiden tekstimäisyys ja eri kommenttien välinen jatkuvuus on haparoivaa mutta kuitenkin olemassa niin, ettei sitä voi jättää huomiotta. Ei voi sanoa, että kommentit olisivat jokainen yksittäisiä tekstejään, mutta niiden muodostamissa kokonaisuuksissa sidosteisuus on usein vähäistä. Tästä syystä kommenttien välisessä sidosteisuudessa otan huomioon tavan yhdistää kaksi erillistä kommenttia yhdeksi kokonaisuudeksi rinnastuskonjunktion avulla, vaikkeivät rinnastuskonjunktiot monessa tekstissä olisivatkaan merkittäviä sidosteisuuden kannalta. Esimerkki 90 havainnollistaa rinnastuskonjunktiolla yhdistämistä aineistossa.

```
(90) // Käydään alkiot läpi... (A42)
     if (merkkitaulukko != null) {
         for (int rivi = 0; rivi < rivlkm; rivi++) {
             for (int sarake = 0; sarake < sarlkm; sarake++) {
                 // ...ja kopioidaan, kun löydetään merkki.
                 if (merkkitaulukko[rivi][sarake] == RUOKA) {
                     ruuanRiv = rivi;
                     ruuanSar = sarake;
```

Esimerkissä 90 on kaksi toisistaan erillistä kommenttia, jotka liittyvät kieliopillisesti ja sisällöllisesti yhteen. Kieliopillinen kytkentä tehdään toisen kommentin alussa sijaitsevan *ja*-rinnastuskonjunktion avulla. Rinnastuskonjunktiota käytetään kytkemään toisiinsa samanfunktioisia ja syntaktisesti toisistaan riippumattomia elementtejä, jolloin sen yhdistämät lauseet ovat muodollisesti ja semanttisesti itsenäisiä (VISK § 817). Kun rinnastuskonjunktio *ja* aloittaa virkkeen, on sen tehtävänä sitoa asiaa edellä olevaan laajempaan kokonaisuuteen (Eronen 1999), joten konjunktion voi ainakin tämän aineiston yhteydessä nimetä kieliopilliseksi koheesiota rakentavaksi keinoksi. Molemmat yksivirkkeiset kommentit pitävät sisällään itsenäisen passiivimuotoisen päälauseen, ja toinen kommentti alkaa pienellä alkukirjaimella kirjoitetulla *ja*-konjunktiolla. Toinen kommentti jatkaa sisältöä siitä, mihin ensimmäinen jäi, ja kommentit ikään kuin muodostavat yhdessä yhden kokonaisen virkkeen, joka on jakautunut kahteen kommenttiin. Kommenttien välissä on kolme riviä koodia, joten ei voi epäillä, että kyseessä olisi kahdelle riville jakautunut kommentti, vaan ne ovat omilla riveillään ja omilla paikoillaan, joten ne todella ovat omia kommenttejaan.

Rinnastuskonjunktiolla rinnastetaan kaksi kommenttia kuulumaan yhteen, mikä on tehty huolimatta siitä, että sisältö tulisi molemmista kommenteista selville ilman ilmikirjoitettua kytköstä: ”// *Käydään alkiot läpi*” ja // ”*Kopioidaan, kun löydetään merkki*” välittäisivät koodin ja kommenttien lukijalle saman tietosisällön koodin aiheuttamista toimista kuin aineistossa esiintyvä rinnastuskonjunktiollinen versio. Rinnastusta käyttämällä kommenttien kirjoittaja on halunnut alleviivata näiden kahden kommentin ja niiden myötä kommentteihin liittyvien koodirivien yhteenkuuluvuutta, vaikka koodirivien yhteenkuuluvuus ilmenee jo rakenteen sisennysten havainnollistamasta hierarkiasta. Jatkamalla virkettä kuin kommentit olisivat yhtä osoitetaan myöhemmästä koodista rakenteen kannalta merkityksellinen osa, sillä muuten koko virke olisi voinut olla ensimmäisessä kommentissa ja toista kommenttia ei tarvitsisi olla lainkaan.

Valitsin juuri tämän esimerkin 90 kommenttien välisestä tyypillisestä rinnastuskonjunktiolla tehtävästä sidosteisuudesta, sillä rinnastuskonjunktion lisäksi siinä esiintyvät aineistosta muutamasta muustakin kohdasta löytyvät kolme pistettä. Kolme pistettä on välimerkki, jonka avulla osoitetaan lauseen jäävän kesken, ja sen jälkeen seuraava pieni alkukirjain tarkoittaa, että seuraava jakso kuuluu samaan virkkeeseen (Kielikello 2/2006: Kolme pistettä …).

Esimerkissä kolmea pistettä käytetään ensimmäisen kommentin lopussa ja toisen kommentin alussa osoittamaan, että nämä kaksi lausetta kuuluvat yhteen, ja että toinen kommentti täydentää ensimmäisen kommentin sisältöä. Myös kolme pistettä on siis osallisena sidostamassa kahta kommenttia toisiinsa, sillä ensimmäisen kommentin päättyessä kolmeen pisteeseen lukija voi päätellä, että kommentti jatkuu vielä jollakin tavalla. Seuraavan kommentin alussa sijaitsee sama välimerkki, jonka avulla osoitetaan, että kokonaisuuden lukemista kuuluu jatkaa tästä kohdasta. Lauseenalkuiset

kolme pistettä eivät ole kielenhuollollisesti pätevä keino yhdistää tekstejä toisiinsa, sillä Kielikellon (2/2006: Kolme pistettä …) artikkelissakaan lauseenalkuiseen välimerkkiin ei oteta kantaa. Ohjelmointikoodin kommenttien epämuodollisuuteen ja tarkan ohjeistuksen puutteeseen kuitenkin sopii se, että jatkuvuutta merkataan kahdessa paikassa sijaitsevan kolmen pisteen avulla. Sitä voi ajatella melko teknisenä ratkaisuna, sillä lukijalle ei haluta jättää epäselväksi sitä, että lause jatkuu, vaikka jatkumisen voisi päätellä kommenttien merkityssisällön sekä rinnastuskonjunktion avulla. Kolme pistettä on silta, joka kulkee koodin päältä yhdistäen kaksi kommenttia toisiinsa.

Yhteistä niille kommenteille, joissa käytetään jotakin kieliopillista keinoa sidostamaan kommentteja toisiinsa, on niiden kommentoima koodi. Kaikki kommentit esimerkeissä 87–90, joissa kommentit liittyvät toisiinsa erilaisten sidoskeinojen avulla, liittyvät koodissa yhteen kiinteään kokonaisuuteen. Esimerkkien 87 ja 89 sidostetut kommentit ovat osa moniosaista valintarakennetta, jossa siis valintarakenteen järjestyksessä seuraava osa suoritetaan ainoastaan, jos edellistä osaa ei voitu suorittaa. Tällaisissa kohdissa sidosteisuuden osoittaminen kielellisin keinoin on luontevaa, sillä kyseessä olevassa koodissa on joko tai -tilanne, jonka voi odottaa näkyvän myös kommenteissa, vaikka suurimmassa osassa moniosaisia valintarakenteita sidosteisuutta ei kommentteihin olekaan liitetty. Esimerkeissä 88 ja 90 sidosteisuus liittyy hierarkisiin rakenteisiin, minkä voi huomata koodin ja kommenttien sisennyksistä. Kieliopillinen sidosteisuus siis esiintyy usein yhden metodia pienemmän kokonaisuuden sisällä, jossa myös koodirivit ovat jollakin tavalla kytköksissä toisiinsa.

5.3.2 Leksikaaliset sidokset

Leksikaalisilla sidoksilla tarkoitetaan sisältösanojen välisiä suhteita tekstin etenemisen kannalta (Shore 2012b: 181). Leksikaalisia sidoksia kommenttien välillä on hieman enemmän kuin kieliopillisia sidoksia, sillä saman koodin sisäsäissä kommenteissa samat sanat voivat toistua paljonkin, mikä voi olla osasyy sille, että kieliopillisten sidosten määrä jää niin vähäiseksi. Esimerkiksi anaforisia pronomineja ei tarvita, jos viittauksen kohde toistetaan uudestaan sen jokaisessa käyttöyhteydessä. Aiemmin esitellyssä lyhyessä esimerkissä 86 leksikaalista sidosteisuutta ei ilmene muuten kuin *uusi hahmo* -lausekkeen käytöllä objektina kahdessa eri kohdassa. Esimerkissä 91 on hieman pidemmän ja tiheämpään kommentoidun metodin kaikki kommentit ilman niiden välissä sijaitsevaa koodia, ja siinä leksikaalista sidoteisuutta on enemmän. Kommentit on numeroitu esimerkissä juoksevasti niin, että huomaa, mitkä peräkkäiset rivit muodostavat yhden kommentin.

(91)
```
1 /* Metodi hahmon liikuttamiseen. (A4)
   */
2 // Tarkistetaan, että parametreissa on järkeä.
3    // Selvitetään hahmon pituus ja kentän mitat.
4          // Lasketaan päälle uusi paikka.
```

```
5          // Jos kuljetaan oikealle.
6             // Pyörähdetään oikelta vasemmalle.
7          // Jos kuljetaan vasemmalle.
8             // Pyörähdetään vasemmalta oikealle.
9          // Jos kuljetaan alas.
10            // Pyörähdetään alhaalta ylös.
11         // Jos kuljetaan ylös.
12            // Pyörähdetään ylhäältä alas
13      // Päätellään ollaanko peruuttamassa. Peruuttaessa pään ja päätä
        // seuraavan merkin paikat ovat samat.
14         // Siirretään hahmoa, jos ei olla peruuttamassa.
15            // Siirretään hahmon merkkejä päätä lukuunottamatta:
16            // Päivitetään paikkatietotaulu.
17               // Hahmon uuden sijainnin rivi- ja sarakeindeksit päivittyvät aina paikkatietotaulun
                    seuraavan rivin tiedoilla.
18            // Päivitetään hahmon pään paikka.
```

Esimerkissä 91 leksikaalista sidosteisuutta rakennetaan ennen kaikkea toistolla, sillä samat verbit ja substantiivit toistuvat useaan kertaan kaikissa metodin kommenteissa. Toisto ei kosketa ainoastaan peräkkäisiä kommentteja, vaan samoja sanoja käytetään kommenttirajojen yli. Esimerkissä pelin ohjailtava olento eli *hahmo* esiintyy metodin kahdeksassatoista kommentissa kuuteen kertaan. Esiintymistä neljä sijaitsee peräkkäisissä kommenteissa (numerot 14 ja 15 sekä 17 ja 18), mikä tarkoittaa, että näissä viittaamisen olisi voinut tehdä myös ilman saman sanan toistoa: kommentti 15 olisi voinut anaforisia pronomineja käyttäen olla esimerkiksi: ”*Siirretään sen merkkejä päätä lukuunottamatta:*” etenkin, kun kommentit 14 ja 15 kuuluvat samaan koodikokonaisuuteen, jossa ne ovat alisteisia kommentin 13 kommentoimalle ohjausrakenteelle. Tällaista valintaa korvata saman sanan toisto jollakin anaforisella pronominilla ei ole tehty aineistossa kertaakaan (ks. luku 5.3.1), joten sen voi ajatella olevan kommentoijien tietoinen valinta.

Verbien (kuten *päivittää*, *pyörähtää* ja *siirtää*) toisto esimerkissä ei olisi korvattavissa millään kieliopillisella viittauskeinolla, eikä niiden muutenkaan voi sanoa vahvasti rakentavan leksikaalista sidosteisuutta. Verbien toisto luo kommenteille kuitenkin koheesion tuntua siten, että pelin ja koodin aihepiiri ja kommentoitavat asiat liittyvät toisiinsa, mutta toistettavat verbit eivät tarvitse toisiaan tullakseen ymmärretyiksi. Näin höllin keinoin sidostetussa tekstissä saman verbin useaan kertaan esiintyminen tekee kommenteista juuri yhden metodin ja yhden ohjelman kommentteja, sillä ne käsittelevät samaa asiaa, joten vaikkei sen voi sanoa olevan varsinainen leksikaalinen sidoskeino, rakentaa se yhteneväisyyttä koko metodin sisälle.

Toiston lisäksi esimerkistä 91 löytyy leksikaalinen sidoskeino, joka on hyvin yleinen aineiston metodien kommenteissa. Esimerkin kommenttien 5, 7, 9 ja 11 lauseet ovat tismalleen samanmuotoiset: *//Jos kuljetaan oikealle/vasemmalle/alas/ylös*. Suunta, johon kuljetaan, on jokaisessa kommentissa eri. Neljässä kommentissa käydään pelikentän kaikki neljä mahdollista liikkumissuuntaa läpi, mikä luo kommenttien välille koheesiota suuntiin liittyvillä leksikaalisilla valinnoilla. Yk-

sikään suunta ei jää mainitsematta, jolloin koko suuntavaihtoehtosanojen kirjo toteutuu kommenteissa. Samanlaista suuntasanoin sidostamisen keinoa käytetään myös esimerkissä 92, jossa kommentit eivät ole lausemuotoisia vaan sisältävät ainoastaan kulloinkin kyseessä olevan suuntasanan.

(92) **//vasen** (A17)
```
if (komento == 'l') {
    ...
}
//oikea
if (komento == 'r') {
    ...
}
//ylös
if (komento == 'u') {
    ...
}
//alas
if (komento == 'd') {
    ...
}
```

Suuntasanojen järjestys ei ole esimerkeissä 91 ja 92 sama, mutta niiden esiintymisjärjestyksessä on molemmissa esimerkeissä logiikka: vastakohtaiset *oikea* ja *vasen* ovat peräkkäin ja vastakohtaiset *ylös* ja *alas* ovat peräkkäin. Sen lisäksi, että sanat ovat yhdestä selkeästä aihepiiristä, ne myös järjestyksellään auttavat tekstin – niin kommenttien kokonaisuuden kuin koodirivienkin – koheesiota ja helppolukuisuutta. Esimerkin 92 leksikaalinen sidosteisuus ylittää koodin ja kommenttien rajan. Esimerkin kommenteissa käytetään suuntasanoja, ja suuntasanoihin viitataan myös itse koodissa, sillä *komento*-muuttujan pitää kyseisen opiskelijan koodissa olla joko *l*, *r*, *u* tai *d*, joista *l* (englannin *left* ’vasen’) sijaitsee *//vasen*-kommentin jälkeen, *r* (englannin *right* ’oikea’) *//oikea*-kommentin jälkeen ja niin edelleen. Kommenteilla vahvistetaan jako, joka jo koodissa on tehty.

Leksikaalisilla sidoksilla voidaan yhdistää myös kommenttia ja koodia toisiinsa, ja keskeinen keino tällaisten sidosten tekemisessä on koodin muuttuja- ja metodinimien käyttäminen sekä koodissa että kommenteissa. Koodissa sen kirjoittaja nimeää metodit ja muuttujat haluamallaan tavalla, joten suomeksi kommentoidussa aineistossa muuttujien ja metodien nimet ovat lähes kaikissa koodeissa suomenkielisiä. Muuttujien nimiä käytetään useimmissa tapauksissa juuri ennen koodiriviä sijaitsevassa kommentissa, minkä avulla osoitetaan, että kommentti ja koodi liittyvät toisiinsa, kuten esimerkeissä 93 ja 94. Esimerkeistä on alleviivattu ne sanavalinnat eli muuttujannimet, joita käytetään sekä koodissa että kommenteissa.

(93) **//Luodaan aputaulu päivittämistä varten** (A10)

```
int[][] aputaulu = new int[RIVILKM][SARAKELKM];


(94) // Palautetaan uusi hahmo (A45)
     return uusihahmo;
```

Esimerkin 93 kommentissa kerrotaan, että luodaan *aputaulu*, ja seuraavalla rivillä luodaan samanniminen taulukko. Samaten esimerkissä 94 kerrotaan palautettavan *uusi hahmo*, jonka jälkeen koodissa palautetaan *uusihahmo*-niminen muuttuja. Ensimmäisessä esimerkissä sanavalinta on suoraan muuttujan nimi, ja toisessa lauseke *uusi hahmo* viittaa muuttujaan, jossa adjektiivi ja substantiivi on ohjelmointiteknisistä syistä kirjoitettu yhteen. Tekemällä kommenteissa ja koodissa samoja sanavalintoja osoitetaan, kuinka kiinteä yhteys kommenttien ja ohjelmointikoodin maailmoilla on toisiinsa.

Ohjelmointikoodi ja luonnollisella kielellä ihmisen luettaviksi tarkoitetut kommentit ovat käytännön tasolla hyvin erilaiset siltä kannalta, mitä varten ne on kirjoitettu eli mitä niillä pyritään saamaan aikaan, kuten myös siltä kannalta, millaiseen muotoon ja kenen luettaviksi ne on tarkoitettu. Kommentti on ihmistä varten ja koodi tietokonetta varten, mutta käyttämällä tismalleen samoja nimiä viittaamaan samaan tarkoitteeseen eli koodissa sijaitsevaan muuttujaan asetetaan nämä kaksi hyvin erilaista tekstimaailmaa hetken ajaksi päällekkäin. Niillä on yhteistä omaisuutta, jonka avulla paitsi käytännössä viitataan kommentista koodiin, myös osoitetaan, että ne ovat yksi tekstimaailma. Ei ole olemassa koodin kommenttia ilman itse koodia, ja siksi kommenttien tekstitodellisuuden hahmottelun kannalta on tärkeää ottaa huomioon sen yhteys koodin tekstitodellisuuteen.

Kommentit eivät ole sellaisia, että kaikki kommentit pitäisi lukea järjestyksessä ensimmäisestä viimeiseen, jotta voisi ymmärtää yhden kommentin merkityksen. Yksi niiden keskeinen käyttökontekstin määräämä ominaispiirre on se, että niiden täytyy lähes aina olla ymmärrettäviä myös yksinään, sillä koodista voi esimerkiksi korjata tai tarkistaa ainoastaan yhden pienen palasen vilkaisemattakaan sen ympärillä sijaitsevaa koodia. Tästä johtuu, että niin leksikaaliset kuin kieliopillisetkin sidokset liittyvät usein koodikokonaisuuksiin, jotka pitää todennäköisesti joka tapauksessa lukea kokonaan, vaikka korjaisi ainoastaan yhden rivin kokonaisuuden sisältä. Sidosteisuutta ei ole sellaisissa paikoissa, joissa voi olettaa, että lukija ei ole lukenut aiempaa kommenttia. Vaikka kommentit sisältävät sidosteita välillään, ovat ne monella tavalla erittäin itsenäisiä paloja osana suurempaa tekstikokonaisuutta.

6 Päätäntö

Tutkimuksen päämääränä on ollut selvittää, millainen on ohjelmointikoodin kommenttien tekstilaji. Ohjelmointikoodin kommentin tekstilajin piirteiden lisäksi tavoitteenani on ollut tutkia genreä yli perinteisten kielitieteellisten ja kirjallisuustieteellisten rajojen ja yhdistää se tietojenkäsittelytieteiden piiriin kuuluvaan ohjelmointiin. Luonnollisen kielen ja ohjelmointikoodin yhdistelmää ei ole suomen kielen osalta juurikaan tutkittu, vaikka se on kiinnostavaa nykyisen tietoyhteiskunnan näkökulmasta. Syynä tieteenalarajojen ylittämiselle on paitsi genretutkimuksen laajentaminen myös sen osoittaminen, että suomen kieli on elinvoimainen tietojenkäsittelyn piirissä.

Ohjelmointikoodin kommentit muodostavat melko yhtenäisen tekstilajin. Niillä on paljon yhdistäviä pirteitä, ja niistä on mahdollista osoittaa erilaisia tyypillisiä muotoja. Tyypillisiä ja ohjelmointikoodin kommentit muista kommenteista erottavia piirteitä ovat esimerkiksi passiivimuotoisen predikaatin yleisyys sekä kommentin alku //- tai /*-merkillä, jonka avulla se on selkeästi erotettu muusta tekstistä eli koodista. Kommenttien tekstilajille on ominaista myös se, että ne ovat hyvin paljon riippuvaisia sen kommentoimasta koodista.

Kommenttien riippuvaisuus kommentoimastaan koodista on määrittävä tekijä kommenttien ja koodin suhteessa. Koodi määrää kommenttien järjestyksen suhteessa toisiinsa, sillä kommentti esiintyy aina ennen sitä koodia, johon se liittyy. Kommentit myös mukailevat koodin sisäistä hierarkiarakennetta, ja koodin hierarkia määrää sen, millaisen määrän koodia kommentti pystyy kattamaan. Koodin sisältö vaikuttaa siihen, millainen kommentti sitä edeltää, sillä erilaisia koodin rakennetyyppejä kommentoidaan eri tavoin. Kommentit ovat ikään kuin koodin luonnolliskielinen jatke, ja ne ovat täysin riippuvaisia kommentoimastaan koodista.

Kommentteihin liittyviä eri tasojen piirteitä on mahdollista hahmottaa Hallidayn metafunktioiden avulla. Kommenteissa keskeiseksi piirteeksi nousee passiivin käyttö predikaatin yleisimpänä persoonamuotona. Yksi syy passiivin käytölle on tavoite häivyttää tekijä taka-alalle, mutta kommenttien implisiittinen tekijä on mahdollista tunnistaa. Yleisin implisiittinen tekijä on tietokone, mutta myös koodin kirjoittaja, pelihahmo tai pelin pelaaja voivat olla passiivimuotoisen lauseen implisiittisiä subjekteja. Kommentin implisiittisen tekijän tunnistamisessa keskeisessä asemassa on se, mitä kommentoitavassa koodissa tapahtuu, mutta myös predikaattiverbin semantiikka voi auttaa tunnistamisessa. Kommenteissa modaalisuutta ja suhtautumista ilmentäviä sanavalintoja on vain vähän, ja etenkin modaalisuus liittyy ainoastaan sellaisiin kommentteihin, joissa kuvaillaan pelin pelaajan tekemiä valintoja. Syynä tälle voidaan pitää kommenttien tarkoitusta kuvailla koodia ja siinä tehtyjä valintoja objektiivisesti. Edellä luettelemieni piirteiden voidaan luokitella kuuluvan interpersoonaisen metafunktion piiriin.

Ohjelmointikoodin kommenteissa rakennetaan kahdenlaisia maailmoja, minkä voi nähdä osana kommenttien ideationaalisia merkityksiä. Kommenteissa voidaan korostaa pelimaailman ensisijaisuutta, jolloin pelistä pyritään kommenttien avulla tekemään todentuntuista ja pelissä tapahtuvat asiat kuvaillaan kuin ne tapahtuisivat oikeasti. Tällä tavalla kommentoimalla autetaan lukijaa hahmottamaan koodin rakentamaa lopputulosta. Toisaalta kommenteissa voidaan nostaa koodimaailma ensisijaiseksi, jolloin kirjoittaja olettaa lukijan ymmärtävän pelin ja koodin yhteyden toisiinsa ilman, että koodin rakentamaan peliin otetaan luonnollisella kielellä kantaa. Mahdollinen on myös kommentti, jossa molempien maailmojen näkökulmat otetaan huomioon eikä valintaa tehdä. Kommenteista nousee myös esiin kommenttien diskurssiyhteisö, jota rakennetaan esimerkiksi runsaahkon erityissanaston käytön avulla. Erityissanastoa käyttämällä osoitetaan, että kommentin kirjoittaja on osa kommenttien diskurssiyhteisöä, ja rajataan lukijoiksi vain toiset diskurssiyhteisön jäsenet.

Kommenteista löytyy kommenttirajat ylittävää sidosteisuutta, jonka voi jakaa kieliopilliseen ja leksikaaliseen. Kieliopillista sidosteisuutta kommentteihin rakennetaan lähinnä konnektiivien ja rinnastuskonjunktioiden avulla, ja leksikaalista sidosteisuutta toiston ja samaan aihepiiriin liittyvien sanavalintojen avulla. Tällaista sidosteisuutta kommentista toiseen esiintyy vain vähän, mutta kuitenkin juuri sen verran, että saman koodin kommenttien voi sanoa muodostavan löyhän tekstikokonaisuuden keskenään. Ohjelmointikoodin kommentit eivät siis ole tekstejä vain omina yksittäisinä kommentteinaan, vaan niissä käytetään keinoja, joilla kommenteista luodaan suurempia tekstikokonaisuuksia. Nämä koheesiokeinot voidaan luokitella tekstuaalisen metafunktion piiriin kuuluviksi piirteiksi.

Koska kaikenlaisten kommenttien yhtenäistä tekstilajia ei varsinaisesti ole, on ohjelmointikoodin kommentit sijoitettava jollakin tavalla aiemmin tutkittujen kommenttien kentälle. Rakenteen tasolla ohjelmointikoodin kommenteilla ei ole juuri mitään yhteistä esimerkiksi uutiskommenttien kanssa. Yksittäisen kommentin rakenne riippuu pitkälti siitä, millaista koodia se kommentoi, ja kokonaisen metodin kommenttien muodostama kokonaisrakenne taas riippuu yksinomaan siitä, missä järjestyksessä koodi metodissa esiintyy. Ohjelmointikoodin kommentti ei kuitenkaan ole tyystin irrallinen muista kommenteista. Kielitoimiston sanakirjan määritelmän mukaan kommentti on selitys tai reunahuomautus, mitä myös ohjelmointikoodin kommentti on, ja monissa lähteissä kommentin keskeiseksi piirteeksi nimetään sen tehtävä tuoda lisäarvoa jollekin toiselle tekstille, minkä kommentti myös tekee toimiessaan luonnolliskielisenä selityksenä ohjelmointikoodille. Ohjelmointikoodin kommentti on siis perustellusti nimeltään kommentti, vaikka sillä onkin monia ainoastaan sen omaan tekstilajiin kuuluvia piirteitä.

Tutkimukseni on onnistunut luomaan yleiskuvan ohjelmointikoodin kommentin tekstilajista, sillä kommenteista löytyy tarpeeksi yhteisiä piirteitä, jotka ovat omanlaisiaan verrattuna muihin

kommenteiksi kutsuttuihin tekstilajeihin. Kommenttien analyysin jaottelu metafunktioiden avulla on myös osoittautunut onnistuneeksi ratkaisuksi, sillä metafunktioita käyttämällä kaikki tekstin ulottuvuudet pääsevät varmasti osallisiksi tekstilajia tutkittaessa. Jos yhden metafunktion jättäisi pois, ei kuvaus tekstilajista olisi enää yhtä kattava.

Tutkimuksen laajuus on myös ollut sen yksi selkeä rajoite. Kun tavoitteena on ollut luoda yleinen määritelmä aineiston kommenttien tekstilajista, on joitakin kiinnostavia yksityiskohtia ollut pakko jättää vähemmälle huomiolle ja keskittyä selkeisiin linjoihin, joita kommenteista päällimmäisenä löytyy. Esimerkiksi kuhunkin metafunktioon olisi ollut mahdollista keskittyä kokonaisen tutkimuksen verran. Toinen tutkimuksen yleistettävyyttä rajoittava tekijä on sen aineiston yksipuolisuus. Kun aineisto koostuu ainoastaan opiskelijoiden kurssia varten tekemistä harjoitustöistä, ovat koodit ja kommentit ulkoista tavoitetta varten kirjoitettuja. Toisaalta opiskelija-aineiston käyttö jätti tutkimuksessa tilaa juuri kommentteihin keskittymiselle, kun aineisto on yhteneväistä ja aihepiirit ovat samoja.

Tämä tutkimus on ollut vasta pintaraapaisu siihen kaikkeen, mitä ohjelmointikoodin kommenteista olisi mahdollista tutkia. Koko tekstilajiin voisi perehtyä vielä huomattavasti tarkemmin esimerkiksi jonkun tietyn metafunktion näkökulmasta. Interpersoonainen metafunktio tarjoaisi monia mahdollisuuksia perehtyä syvemmin koodin tekijyyteen sekä kaikkiin niihin rooleihin, joita kommenteissa on läsnä. Tekstuaalisen metafunktion osalta kommenttien tekstimäisyyteen olisi mahdollista paneutua paljon syvemmin ottamalla huomioon kokonaisten ja suurten koodien koheesiokeinot sekä yksittäisten kommenttien tekstimäisyys.

Ohjelmointikoodin kommentteja ja niiden tekstilajia käsittelevän tutkimuksen voisi tehdä laajemmalla ja monipuolisemmalla aineistolla. Aineisto, joka koostuisi ohjelmoinnin ammattilaisten todellista käyttöä varten kirjoittamista koodeista, tarjoaisi varmasti autenttisemman näkökulman kommentteihin. Myös muilla kuin Java-kielellä tehtyjen koodien yhteyteen kirjoitettujen kommenttien piirteitä voisi tutkia ja pohtia, onko erikielisiin koodeihin liittyvillä kommenteilla eroja toisiinsa nähden. Tällaisen tutkimuksen avulla voitaisiin osoittaa, miten paljon koodin sisäiset rakenteelliset seikat vaikuttavat kommentointiin.

Aivan oman tutkimuksensa saisikin koodin ja kommenttien suhteesta, jonka syvemmässä kuvauksessa voisi esimerkiksi nimetä kaikki mahdolliset rakennetyyppeihin liittyvät kommentit. Myös koodin- ja kommenttienkirjoitusprosessia seuraamalla saisi kiinnostavaa tietoa siitä, millaisessa järjestyksessä koodin ja kommenttien suhde alusta asti rakentuu. Luonnollisen kielen ja ohjelmointikielen yhdistelmää tutkimalla olisi mahdollista hahmottaa, miten luonnollinen kieli on vaikuttanut ohjelmointikieleen sen syntyessä, ja myös, miten luonnollinen kieli muokkautuu vastaamaan koodikielen tarpeita. Tiedon lisääntyessä koodin ja kielen suhteesta lisääntyvät myös mahdollisuudet luo-

da sovelluksia, joissa kieltä käytetään tietokonelähtöisesti. Etenkin suomen kielelle tällaisen tutkimuksen lisääntyminen olisi hyväksi, sillä englanti hallitsee niin monella saralla tietokoneisiin ja internetiin liittyvissä sovelluksissa ja työkaluissa.

Toisesta näkökulmasta tutkimukseni osoittaa sen, että erilaisten kommenttien tekstilajien tutkiminen olisi aiheellista. Kommentin määritelmä on monessa yhteydessä ympäripyöreä, eikä konkreettiseksi esimerkiksi tekstilajin edustajasta usein nimetä muuta kuin uutiskommentti. Eri kommenttien tekstilajeilla on yhteisiä piirteitä, mutta piirteitä ei ole missään tutkimuksessa tarkkaan määritelty ja nimetty. Monet kommentit, kuten verkkouutisiin tai blogikirjoituksiin liittyvät, ovat verrattain uusia tekstilajeja, ja niihin perehtymällä kokonaiskuva kommenteista lisääntyisi. Useaan kommentin tekstilajiin perustuvan kokonaiskuvan avulla olisi mahdollista pohtia, onko kommentti yläkäsite erilaisille, eri paikoissa esiintyville kommenteille.

Ohjelmointikoodin kommentin tekstilajin määritelmä paitsi lisää tietoa kyseisestä tekstilajista, myös avaa ovia uusille näkökulmille genrentutkimukseen, ohjelmoinnin ja kielitieteen yhdistämiseen sekä kommenttien tekstilajin pohdintaan.

Lähteet

Kirjallisuuslähteet

ALHO, IRJA – KAUPPINEN, ANNELI 2009: *Käyttökielioppi*. Suomalaisen Kirjallisuuden Seuran Toimituksia 1154, Tiede. Helsinki: Suomalaisen Kirjallisuuden seura.

ERONEN, RIITTA 1999: Monenlaisia rinnastuksia: eli, tai, mutta. - *Kielikello* 3/1999. Viitattu 17.10.2015. Saatavissa: http://www.kielikello.fi/index.php?mid=2&pid=11&aid=532.

GRÜNN, KARL - GRÜNTHAL, SATU - UUSI-HALLILA, TUULA 2004: *Kivijalka. Lukion äidinkielen ja kirjallisuuden oppikirja*. Jyväskylä: Kustannusosakeyhtiö Tammi.

HALLIDAY, M. A. K. 1978: *Language as social semiotic. The social interpretation of language and meaning.* London: Arnold.

HALLIDAY, M.A.K. 1985. *An introduction to functional grammar*. London: Edward Arnold, 1. painos, (neljäs muutettu painos 2014).

HALLIDAY, M.A.K. & R. HASAN 1985. *Language, context and text. Aspects of language in a social-sentiotic perspective.* Geelong: Deaking University Press.

HAKULINEN, AULI - KALLIOKOSKI, JYRKI - KANKAANPÄÄ, SALLI - KANNER, ANTTI - KOSKENNIEMI, KIMMO - LAITINEN, LEA - MAAMIES, SARI - NUOLIJÄRVI, PIRKKO 2009: *Suomen kielen tulevaisuus. Kielipoliittinen toimintaohjelma*. Kotimaisten kielten tutkimuskeskuksen verkkojulkaisuja 7. Helsinki: Kotimaisten kielten tutkimuskeskus. Saatavissa: http://scripta.kotus.fi/www/verkkojulkaisut/julk7/suomen_kielen_tulevaisuus_kotus_verkkojulkaisuja_7.pdf

HEIKKINEN, VESA – VOUTILAINEN, EERO 2012: Kieli. Teoksessa Vesa Heikkinen, Eero Voutilainen, Petri Lauerma, Ulla Tiililä & Mikko Lounela (toim.): *Genreanalyysi – tekstilajitutkimuksen käsikirja*. Helsinki: Gaudeamus.

HEIKKINEN, VESA 2012: Teksti. Teoksessa Vesa Heikkinen, Eero Voutilainen, Petri Lauerma, Ulla Tiililä & Mikko Lounela (toim.): *Genreanalyysi – tekstilajitutkimuksen käsikirja*. Helsinki: Gaudeamus.

KALLIOKOSKI, JYRKI 1996: Johdanto. Teoksessa Jyrki Kalliokoski (toim.): *Teksti ja ideologia. Kieli ja valta julkisessa kielenkäytössä.* Helsingin yliopiston suomen kielen laitos: Helsinki.

KALLIOKOSKI, JYRKI 2002: Tekstilajin taju. Teoksessa Ilona Herlin, Jyrki Kalliokoski, Lari Kotilainen ja Tiina Onikki-Rantajääskö (toim.): *Äidinkielen merkitykset*. Helsinki: Suomalaisen Kirjallisuuden Seura. 147–159.

KEINÄNEN, MINNA – PAALANEN, PIILA – TIAINEN, MARJA-LEENA 2007: *Aktiivi 7. Äidinkieli ja kirjallisuus.* Jyväskylä: Kustannusosakeyhtiö Tammi.

KIIVERI, KAISA 2006: Matkalla lukutaitoon. Kaksi kuvausta lukutaidon oppimisesta koulussa. Lapin yliopistokustannus: Rovaniemi.

Kielikello 2/2006: *Kolme pistettä ....* Viitattu 17.10.2015. Saatavissa: http://www.kielikello.fi/index.php?mid=2&pid=11&aid=1678.

Kielikello 2/2006: *Lainausmerkit "*. Viitattu 23.10.2015. Saatavissa: http://www.kielikello.fi/index.php?mid=2&pid=11&aid=1684. (

KNUTH, DONALD E. 1984: Literate Programming – *The Computer Journal* 27 (2) s. 97–111. Saatavissa: http://comjnl.oxfordjournals.org/content/27/2/97.full.pdf+html.

KUIRI, KAIJA 2000: Kielellistä passiivisuutta: miksi me mennään. - *Kielikello* 3/2000. Viitattu 29.10.2015. Saatavissa: http://www.kielikello.fi/index.php?mid=2&pid=11&aid=1195

LAURIKKALA, JORMA 2014a: *Pseudokoodi.* Luento Tampereen yliopiston kurssilla lausekielinen ohjelmointi. Tampere.

LAURIKKALA, JORMA 2014b: *Algoritmi.* Luento Tampereen yliopiston kurssilla lausekielinen ohjelmointi. Tampere.

LAURIKKALA, JORMA 2014c: *Hyvä ohjelmointitapa (osa 1).* Luento Tampereen yliopiston kurssilla lausekielinen ohjelmointi. Tampere.

LAURIKKALA, JORMA 2014d: *Muuttujat ja Java.* Luento Tampereen yliopiston kurssilla lausekielinen ohjelmointi. Tampere.

LAURIKKALA, JORMA 2014e: *Olio-ohjelmointia Javalla.* Luento Tampereen yliopiston kurssilla lausekielinen ohjelmointi. Tampere.

LINCOLN, K. NICHOLAS - VERES, SANDOR M. 2013: Natural Language Programming of Complex Robotic BDI Agents. *Journal of Intelligent & Robotic Systems* s. 211–230. Saatavissa: http://link.springer.com/content/pdf/10.1007%2Fs10846-012-9779-1.pdf.

LUUKKA, MINNA-RIITTA 1995: *Puhuttua ja kirjoitettua tiedettä. Funktionaalinen ja yhteisöllinen näkökulma tieteen kielen interpersonaalisiin piirteisiin.* Jyväskylä: Jyväskylän yliopisto.

LUUKKA, MINNA-RIITTA 2002: M.A.K Halliday ja systeemis-funktionaalinen kielitiede. Teoksessa Hannele Dufva & Mika Lähteenmäki (toim.), *Kielentutkimuksen klassikoita* s. 89–123. Soveltavan kielentutkimuksen teoriaa ja käytäntöä 4. Jyväskylä: Soveltavan kielentutkimuksen keskus.

MARTIN, ROBERT C. 2009: *Clean Code. A handbook of agile software craftsmanship.* Pearson Educations.

MÄNTYNEN, ANNE 2006: Genre ja intertekstuaalisuus. Teoksessa Mäntynen, Anne – Shore, Susanna – Solin, Anna (toim.): *Genre – tekstilaji.* Tietolipas 213. Helsinki: Suomalaisen Kirjallisuuden Seura.

MÄNTYNEN, ANNE – SHORE, SUSANNA 2008: Tekstilajien lumo. TeoksessaTiina Onikki-Rantajääskö & Mari Siiroinen (toim.): *Kieltä kohti.* Helsinki: Otava.

NEVALAINEN, HEIDI 2013: *"JÄI JOTAIN HAMPAANKOLOON". Sanomalehtien kommenttijuttujen tekstilajisuus*. Pro gradu -tutkielma. Jyväskylän yliopiston kielten laitos. Saatavissa: http://urn.fi/URN:NBN:fi:jyu-201312232867.

NIEMINEN, TOMMI 2010: *Lajien synty. Tekstilaji kielitieteen semioottisessa metateoriassa.* Jyväskylä Studies in Humanities 136. Jyväskylä: Jyväskylän yliopisto. Saatavissa: http://urn.fi/URN:ISBN:978-951-39-3871-0.

PIETIKÄINEN, SARI – ANNE MÄNTYNEN 2009: *Kurssi kohti diskurssia*, Tampere: Vastapaino

SAUKKONEN, PAULI 2001: *Maailman hahmottaminen teksteinä. Tekstirakenteen ja tekstilajien teoriaa ja analyysia.* Helsinki: Yliopistopaino

SHORE, SUSANNA – MÄNTYNEN, ANNE 2006: Johdanto. Teoksessa Mäntynen, Anne – Shore, Susanna – Solin, Anna (toim.): *Genre – tekstilaji.* Tietolipas 213. Helsinki: Suomalaisen Kirjallisuuden Seura.

SHORE, SUSANNA 2008: Lauseiden tekstuaalisesta jäsennyksestä. – Virittäjä 112 s. 24–65.

SHORE, SUSANNA 2012A: Kieli, kielenkäyttö ja kielenkäytön lajit systeemis-funktionaalisessa teoriassa. Teoksessa Vesa Heikkinen, Eero Voutilainen, Petri Lauerma, Ulla Tiililä & Mikko Lounela (toim.): *Genreanalyysi – tekstilajitutkimuksen käsikirja*. Helsinki: Gaudeamus.

SHORE, SUSANNA 2012B: Systeemis-funktionaalinen teoria tekstien tutkimisessa. Teoksessa Vesa Heikkinen, Eero Voutilainen, Petri Lauerma, Ulla Tiililä & Mikko Lounela (toim.): *Genreanalyysi – tekstilajitutkimuksen käsikirja*. Helsinki: Gaudeamus.

SWALES, JOHN M. 1990: *Genre Analysis. English in Academic and Research Settings*. Cambridge: Cambridge University Press.

VALTONEN, PÄIVI 2012: *Abiturientti uutistoimittajana. Tekstilajin taju ja uutisen tuottaminen äidinkielen tekstitaidon kokeessa.* Turun yliopiston julkaisuj. Turun yliopisto: Turku.

VENTOLA, EIJA 2006: Genre systeemis-funktionaalisessa kielitieteessä. Esimerkkinä asiointitilanteet. Teoksessa Mäntynen, Anne – Shore, Susanna – Solin, Anna (toim.): *Genre – tekstilaji.* Tietolipas 213. Helsinki: Suomalaisen Kirjallisuuden Seura.

VISK = AULI HAKULINEN, MARIA VILKUNA, RIITTA KORHONEN, VESA KOIVISTO, TARJA RIITTA HEINONEN JA IRJA ALHO 2004: *Iso suomen kielioppi*. Helsinki: Suomalaisen Kirjallisuuden Seura. Verkkoversio, viitattu 22.6.2015. Saatavissa: http://scripta.kotus.fi/visk.

VUORIJÄRVI, AINO 2013: *Tekstilaji ja yhteisö. Ammattikorkeakoulun opinnäytetyön diskussio tekstinä.* Helsinki: Unigradia Oy.

WIKLA, ARTO 2003: *Ohjelmoinnin perusteet Java-kielellä.* 4. täydennetty painos. Espoo: OtaDATA ry.

Verkkolähteet

Kielitoimiston sanakirjan verkkoversio. Kotimaisten kielten keskuksen verkkojulkaisuja 35. URN:NBN:fi:kotus-201433, ISSN 2323-3370. Julkaistu verkossa 11.11.2014. Päivitettävä julkaisu. [Viitattu 26.5.2015.]

Tampereen yliopisto, Monikielisen viestinnän ja käännöstieteen koulutusohjelma: *Käännöskommentit*. Päivitetty 30.11.2010. Saatavissa: http://www.uta.fi/ltl/trans/tyokielet/venaja/kaannoskommentti.html [Viittauspäivä 26.5.2015.]

Tampereen yliopisto: *Tietojenkäsittelytieteiden opinto-opas* 2012–2015. Saatavissa: https://www10.uta.fi/opas/opintojakso.htm?rid=6896&idx=0&uiLang=fi&lang=fi&lvv=2014 [Viittauspäivä 9.6.2015.]