Artifist - interaktiivinen tarinankerrontamoottori tietokonepeleihin

Minna Ruuska

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos/tko
Pro gradu -tutkielma
Ohjaaja: Erkki Mäkinen
28.11.2006

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos/tko
Minna Ruuska: Artifist - interaktiivinen tarinankerrontamoottori tietokonepeleihin
Pro gradu -tutkielma, 52 sivua
Marraskuu 2006

Tiivistelmä

Järjestelmiä interaktiiviseen tarinankerrontaan on tehty useissa yliopistoissa, mutta kaupallisissa tietokonepeleissä niitä ei vielä ole käytetty. Viime vuosina kiinnostus tekoälyä kohtaan on kuitenkin lisääntynyt peliteollisuudessakin. Tietokoneen ohjaamista älykkäistä ja luontevasti kommunikoivista pelihahmoista on tullut myyntivaltti. Tällaisten hahmojen tekeminen ilman apuvälineitä on kuitenkin hyvin työlästä, ja toisaalta monet tehtävään liittyvistä ongelmista ovat yhteisiä useille eri pelisovelluksille. Tietokonepelien tekoälyohjelmointia helpottaville työkaluille olisi siis käyttöä.

Tässä työssä käsitellään erilaisia tekoälytekniikoita, joita interaktiiviseen tarinankerrontaan voidaan hyödyntää, sekä esitellään joitakin olemassa olevia tarinankerrontajärjestelmiä. Lisäksi työssä esitellään Tampereen teknillisen yliopiston Ohjelmistotekniikan laitoksella suunniteltu ja toteutettu tarinankerrontajärjestelmä Artifist (Artificial Intelligence Framework for Interactive Storytelling).

Artifist on moniagenttijärjestelmä, jossa tarina muodostuu yksittäisten agenttien suorittamista valinnoista. Artifist käsittää kaksi sisäkkäistä päättelyjärjestelmää. Primäärinen päättelyjärjestelmä, joka käyttää probabilistisia päätöspuita ja graafeja, on tarkoitettu agenttien tavoista ja mieliteoista lähtevien valintojen mallintamiseen. Sekundäärisen päättelyjärjestelmän avulla käsitellään agenttien välistä tiedonvälitystä, juonielementtejä ja toimintosarjoja.

Artifist on suunnattu erityisesti tietokonepelien tekoälyn tekemiseen, ja sen suunnittelussa on kiinnitetty erityistä huomiota intuitiivisuuteen ja läpinäkyvyyteen. Tarkoituksena on, että myös pelisuunnittelijat, joilla ei välttämättä ole ohjelmointikokemusta, voisivat käyttää Artifistia. Kokeiltaessa järjestelmää on havaittu, että puiden ja graafien piirtäminen helpottaa sovellusten suunnittelua ja virheiden paikallistaminen on melko helppoa. Probabilistisen päättelymallin avulla hahmoista saadaan uskottavampia ja vähemmän ennalta arvattavia kuin täysin deterministisen mallin avulla.

Sisällys

1 Johdanto

Viime vuosina entistä parempi tekoäly ja uskottavammat tietokoneen ohjaamat hahmot ovat nousseet merkittäviksi myyntivalteiksi tietokonepeleissä. Grafiikan alalla ollaan päädytty tilanteeseen, jossa sen parantaminen ei enää merkittävästi paranna pelikokemusta. Tekoälyn puolella sen sijaan on runsaasti kehittämisen varaa. Se on vuosia jäänyt varjoon, kun lähes koko koneen laskentakapasiteetti on haluttu valjastaa grafiikan käyttöön. Nykypäivän tietokonepeleissä näyttävä grafiikka on arkipäivää, ja uusien pelien täytyy erottua muilla tavoin. Yhä useammin tämä tapa on parempi tekoäly, joka antaa tietokoneen ohjaamille hahmoille mahdollisuudet samoihin toimintoihin kuin pelaajillekin.

On selvää, että tällaisen tekoälyn ohjelmoiminen on kaukana triviaalista - ainakin, jos pelissä on tarkoitus harjoittaa hienovaraisempaakin vuorovaikutusta kuin pelkkää ammuskelua ja tappelua. Kun otetaan vielä huomioon, että monet aihepiiriin liittyvät ongelmat ovat yhteisiä suurelle joukolle pelejä, herää väistämättä ajatus tekoälymoottorista, jota voidaan käyttää pelitekoälyn luomiseen samaan tapaan kuin grafiikkamoottoria pelien grafiikan tuottamiseen.

Tässä työssä käsitellään Tampereen teknillisellä yliopistolla toteutettua tekoälymoottoria Artifistiä *(Artificial Intelligence Framework for Interactive Storytelling)* [Ruuska & Virtanen, 2006] keskittyen erityisesti sen tarinankerrontaalgoritmiin. Artifist on moniagenttijärjestelmä, ja sen tuottama tarina muodostuu yksittäisten agenttien tekemistä toiminnoista. Agenttien primäärinen päättelyjärjestelmä perustuu probabilistisiin päätöspuihin ja graafeihin. Lisäksi Artifistiin on lisätty sekundäärinen päättelyjärjestelmä, jonka avulla voidaan mallintaa suunnitelmallisempaa toimintaa ja agenttien välistä tiedonvälitystä.

Työn toisessa luvussa käsitellään erilaisia interaktiivisen tarinankerronnan lähtökohtia ja menetelmiä sekä esitellään joitakin olemassa olevia tarinankerrontajärjestelmiä. Kolmannessa luvussa esitellään Artifist-järjestelmä ja sen keskeiset osat. Neljännessä luvussa kuvataan agenttien primääristä päättelyjärjestelmää ja viidennessä luvussa sekundääristä päättelyjärjestelmää. Kuudennessa luvussa käydään läpi joitakin esimerkkejä siitä, mihin Artifistia voidaan käyttää. Viimeisessä luvussa arvioidaan Artifistin vahvuuksia ja heikkouksia, pohditaan sen jatkokehitysmahdollisuuksia sekä kootaan yhteen työn tulokset.

2 Interaktiivinen tarinoiden generointi

Interaktiivisen tarinankerronnan lähtökohdat ja tavoitteet ovat moninaiset. Osa tutkimuksesta on teknisesti orientoitunutta, osalla painopiste liittyy sisällöntuotantoon tai draaman ominaisuuksiin. Suurin osa alan tutkimuksesta on toistaiseksi ollut puhtaasti akateemista, eikä sen sovelluksia ole kaupallisissa peleissä vielä nähty. Koska Artifistin tavoitteet ovat sikäli käytännönläheiset, että siitä on pyritty tekemään myös todellisiin tietokonepeleihin soveltuva, tässä luvussa käsitellään paitsi interaktiivisen tarinankerronnan tutkimusta myös tietokonepelien tekoälyn nykytilannetta.

2.1 Käsitteitä

Tarinankerronnan tulee tuottaa mielekkäitä tarinoita. Mielekkäässä tarinassa toisaalta juonielementit liittyvät toisiinsa järkevällä tavalla, toisaalta henkilöt ovat uskottavia. Tähän tavoitteeseen voidaan pyrkiä eri lähtökohdista käsin. Tarinat voivat olla juoneltaan lineaarisia *(linear narrative)* tai haarautuvia *(branching narrative)*. Tietokonepeleistä puhuttaessa ollaan tietenkin kiinnostuneita lähinnä haarautuvajuonisten tarinoiden generoinnista, mutta muun viihteen tarpeisiin lineaaristenkin juonien automaattinen tuottaminen voi olla kiinnostavaa.

Tarinan generointimenetelmät jaotellaan usein henkilölähtöiseen *(character based)* ja juonilähtöiseen *(plot based)* kerrontatapaan. Henkilölähtöisessä kerronnassa juoni muodostuu autonomisten hahmojen virtuaalimaailmassa suorittamista toiminnoista. Juonilähtöisissä järjestelmissä tarina koostuu yhteen linkitetyistä korkean tason juonielementeistä. Toisinaan puhutaan myös suunnitelmalähtöisestä *(plan based)* kerronnasta, joka on melko lähellä juonilähtöistä kerrontatapaa.

Toisenlainen jaottelu sisältää kolme mallia: henkilökeskeisen *(character-centric)*, tarinakeskeisen *(story-centric)* ja kertojakeskeisen *(author-centric)*. Henkilökeskeinen malli vastaa hyvin henkilölähtöistä kerrontatapaa. Kertojakeskeisessä mallissa tarinan generointi perustuu siihen, miten ihmiskertoja tarinaa tuottaisi, kun taas tarinakeskeinen kerronta lähtee koostamaan tarinaa korkean tason juonielementeistä alaspäin aina lopulliseen ilmaisuun asti.

Toisinaan käytetään myös termejä vahva autonomia *(strong autonomy)* ja vahva tarina *(strong story)*. Vahva autonomia viittaa järjestelmään, jossa henkilöt tai agentit voivat tehdä valinnat täysin vapaasti virtuaalimaailman määrittelemissä puitteissa. Vahvan tarinan tuottaminen taas edellyttää jonkinlaista keskitettyä tarinankertojaa, joka valitsee kullekin henkilölle ne toiminnot, jotka kulloinkin

tarinan kaaren kannalta ovat edullisimmat. On helppo kuvitella, että vahva autonomia tuottaa helpommin tarinoita, joissa henkilöhahmot uskottavalla tavalla pyrkivät tavoitteisiinsa, kun taas vahvaan tarinaan pohjautuva järjestelmä tuottaa mielekkäämpiä ja kiinnostavampia tarinoita. Myös järjestelmät, jotka sisältävät elementtejä eri malleista, ovat mahdollisia. [Riedl, 2004]

2.2 Tavoitteet ja nykytilanne tietokonepeleissä

Toistaiseksi suurimmassa osassa tietokonepelejä on käytetty lineaarisia juonirakenteita. Peli muodostuu tasoista, joilla sijaitsevat ongelmat pelaajan täytyy selvittää. Tasot on linkitetty toisiinsa lineaarisilla välianimaatioilla. Joissakin peleissä juoneen on lisätty jonkin verran haarautumista siten, että etukäteen käsikirjoitettuun tarinaan on muodostettu vaihtoehtoisia reittejä kohti lopputilannetta, tai peliin on tehty vaihtoehtoisia loppuja. Koska tietokonepelien juonet ovat lähes poikkeuksetta etukäteen käsikirjoitettuja, ne ovat selvästi juoni- tai kertojakeskeisiä. Vain joitakin simulaatiopelejä, kuten Sims tai Creatures, voi pitää henkilökeskeisinä. Niissä ei kuitenkaan ole mainittavammin juonta, jos hahmon elinkaarta ei pidetä sellaisena.

Tekoäly on muutoinkin kuin tarinan generoinnin osalta jäänyt jo pitkään kaupallisissa peleissä näyttävän grafiikan varjoon. Kaikki liikenevä prosessoriaika on haluttu käyttää visuaalisen ilmeen parantamiseen. Tilanne on kuitenkin hiljalleen muuttumassa ja kiinnostus tekoälyä kohtaan on kasvussa. Tällä hetkellä tietokonepelien käyttämät tekoälymenetelmät rajoittuvat kahteen vaihtoehtoon: äärellisiin tilakoneisiin ja skripteihin. Tilakoneita käsitellään tarkemmin kohdassa 2.3.

Skriptien hyödyt liittyvät lähinnä ohjelmiston kehitysprosessin vauhdittamiseen, eikä niinkään pelien tekoälyn laadun parantamiseen. Skriptien kirjoittaminen on melko yksinkertaista, joten niitä käyttävät peliohjelmoijien lisäksi myös pelisuunnittelijat. Skriptit vaihtelevat lineaarisista liikesarjoista kokonaisiin kohtauksiin, jotka puolestaan on saatettu rakentaa esimerkiksi tilakoneina. [Cass, 2002]

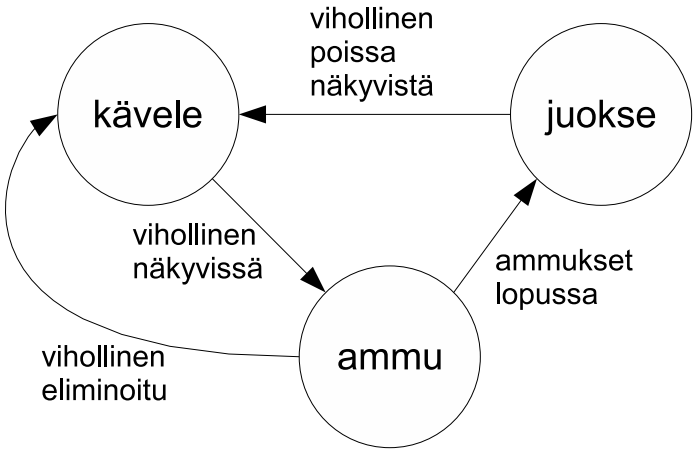
Tila-automaattien ja skriptien tuottamiseen on olemassa joitakin kaupallisia apuvälineitä *(middleware)*, kuten AI.implant [Dybsand, 2003a], DirectIA [Dybsand, 2003b], Renderware AI, [Dybsand, 2003c], SimBionic [Dybsand, 2003d] ja Spark! [Spark!, 2006]. Näiden käyttö ei kuitenkaan ole vielä kovin yleistä. Syyt ovat osittain psykologisia, osittain todellisia. Pelätään, että kolmannen osapuolen toimittaman ohjelmakoodin käyttö estää riittävän nopeusoptimoinnin. Toisaalta

huonossa tapauksessa välineen käytön opetteluun saattaa kulua enemmän aikaa kuin sen käytön avulla säästetään.

2.3 Tilakoneet

Tilakoneet koostuvat äärellisestä määrästä tiloja, jotka peliohjelmoija määrittelee etukäteen. Tilojen välillä on tilasiirtymiä. Tyypillisesti tilakoneiden avulla mallinnetaan yksittäisten tietokoneen ohjaamien hahmojen toimintoja, joten ne soveltuvat lähinnä henkilölähtöiseen tarinangenerointiin. Tilat määrittelevät agentin tavoitteet, ja agentin havainnot tuottavat tilasiirtymät. Tulosteet, eli agentin suorittamat toiminnot, voidaan liittää joko tilasiirtymiin (Mealyn tilakoneet) tai tiloihin (Mooren tilakoneet). Tilasiirtymiin sidotut toiminnot riippuvat sekä tilasta että syötteestä, jolloin saavutetaan monimutkaisempi toiminta, mutta samalla tilakoneen suunnittelu vaikeutuu. Mooren tilakoneiden suunnittelu on yksinkertaisempaa, mutta myös niiden toiminta on ennalta arvattavampaa. Kuvan 2.1 esimerkkitilakone on Mooren tilakone. Toisaalta myös hypertekstitarinaa voisi ajatella tilakoneena, jossa kuhunkin tilaan liittyy pätkä tekstiä tai välianimaatio ja jossa käyttäjän tekemät valinnat määrittelevät tilasiirtymät.

Tilakoneiden luonnollinen toteutustapa on graafi eli verkko, jossa solmut ovat tiloja ja kaaret tilasiirtymiä. Kaaret kannattaa yleensä esittää mieluummin kytkentälistojen kuin kytkentämatriisien avulla, koska tilakoneita kuvaavat graafit ovat harvoin kovin tiheitä. Tilasiirtymät toteutetaan yleensä funktioina.



Kuva 2.1: Esimerkki tilakoneesta

Vaikka tilakoneet teoreettisena mallina ovatkin paljon tutkittu kokonaisuus, niiden käyttö tietokonepeleissä ei ole täysin ongelmatonta. Kun syötteiden tilakoneiden teoriassa oletetaan tulevan jonossa, tietokonepelissä näin ei välttämättä ole. Suoritettavat toiminnot voivat riippua radikaalisti siitä, missä järjestyksessä havainnot käsitellään. Jos havainnot a ja b tulevat samanaikaisesti ja havainto a aiheuttaa siirtymisen tilaan C, jossa havainnolla b ei ole vaikutusta, jää havainto b kokonaan käsittelemättä. Tämä on selvästikin ongelma, mikäli havainto b on luonteeltaan sellainen, että se tuskin jäisi keneltäkään ihmiseltä huomaamatta. Yleinen ratkaisu ongelmaan on se, että agenttien viestinvälitysjärjestelmä toteutetaan tavallisen jonon sijasta prioriteettijonon avulla. Tällöin merkityksellisten havaintojen ei koskaan pitäisi jäädä käsittelemättä.

Toinen ongelma on se, että toimintoja suoritettaessa agentin muistin tila saattaa muuttua, ja tämä saattaa laukaista uuden tilasiirtymän ennen kuin edellinen on käsitelty loppuun. Seurauksena agentti voi juuttua kahden tilasiirtymän väliin pitkäksikin ajaksi. Tällaiset virhetoiminnot saadaan vältettyä kun tilasiirtymistä tehdään atomisia.

Pelkät tilakoneet ovat ilmaisuvoimaltaan melko vaatimattomia, eivätkä ne sovellu kovin monimutkaisiin sovelluksiin. Tämän vuoksi niihin on tehty monenlaisia laajennoksia. Peleissä hyvin yleisesti käytetty tapa kasvattaa tilakoneiden ilmaisuvoimaa on lisätä niihin muistia, jolloin tilasiirtymä saattaa riippua normaalin tilan ja syötteen lisäksi muistissa olevien muuttujien arvoista. Tällöin toiminnot voivat myös muuttaa muuttujien arvoja.

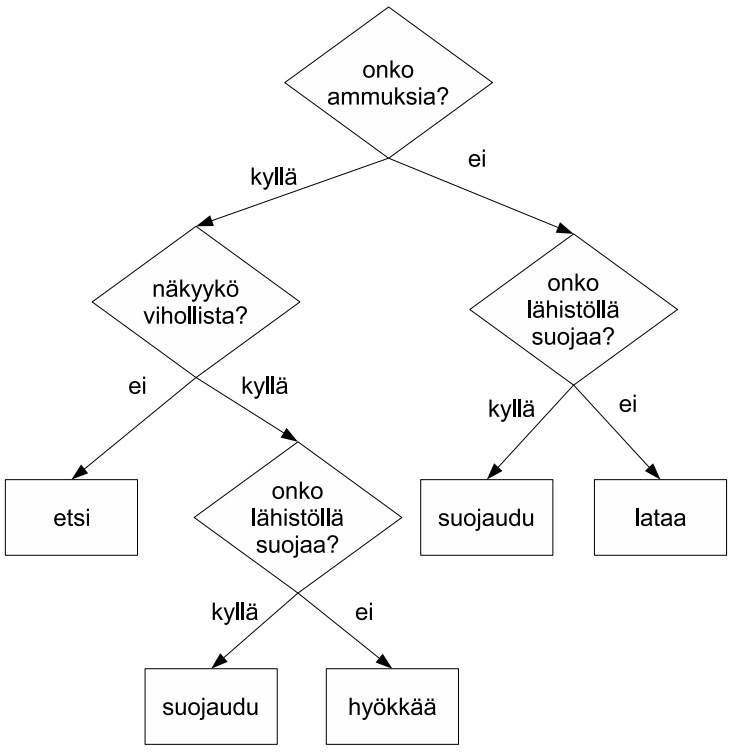
Tilakoneet voivat olla myös epädeterministisiä, jolloin tilakoneessa voi olla niin sanottuja epsilonsiirtymiä, jotka tapahtuvat ilman syötettä, sekä useita vaihtoehtoisia tilasiirtymiä samalla syötteellä. Tällöin täytyy olla jokin menetelmä tilasiirtymän valintaan. Tyypillisin tapa on valita tilasiirtymä probabilistisesti niin sanotun probabilistisen tilakoneen avulla. Siinä kuhunkin tilasiirtymään liitetään todennäköisyys, joka riippuu syötteestä ja mahdollisesti aikaisemmin tehdyistä tilasiirtymistä. Kussakin tilassa todennäköisyyksien summan tulee olla 1. Yleisimmin peleissä käytetyissä tilakoneissa tilasiirtymät riippuvat vain syötteestä ja tilasta, josta siirtymää ollaan tekemässä, eli niissä noudatetaan ns. ensimmäisen asteen Markovin ketjua.

Toinen vaihtoehto liiallisen säännönmukaisuuden välttämiseen on sumean logiikan käyttäminen tilasiirtymäehdoissa. Toisaalta, jos tilasiirtymäehdot ovat kovin monimutkaisia, ne voidaan määritellä päätöspuiden avulla. Päätöspuita käsitellään tarkemmin kohdassa 2.4. Tilakoneet voivat olla myös hierarkkisia, jolloin tila voikin olla kokonainen tilakone. Hierarkkisten tilakoneiden avulla pystytään

hallitsemaan huomattavasti suurempia kokonaisuuksia kuin tavallisten tilakoneiden avulla. Toinen tapa suurten kokonaisuuksien käsittelyyn ovat rinnakkaiset tilakoneet, joissa agenttien päättely eri asioihin liittyen on jaettu erillisiksi tilakoneiksi, joita suoritetaan samanaikaisesti. [Millington, 2006] [Dalmau, 2005]

2.4 Päätöspuut

Päätöspuut ovat puurakenteita, joiden sisäsolmut ovat ehtoja ja lehtisolmut ovat toimintoja. Ehtosolmujen evaluointi tuottaa tiedon siitä, mihin puun haaraan päättely etenee. Päättely aloitetaan aina puun juuresta ja se loppuu lehtisolmuun. Lehteen johtava polku määrittelee sen, millaisessa tilanteessa lehden toiminto suoritetaan. Kuvassa 2.2 on esimerkki binäärisestä päätöspuusta. Menetelmä soveltuu tietokonepeleihin varsin hyvin toisaalta intuitiivisuutensa, toisaalta tehokkuutensa vuoksi. Mikäli ehtojen evaluoiminen on vakioaikaista ja puu olisi binäärinen ja tasapainotettu, olisi yhden toiminnon valinnan suoritusaika $\Theta(log\ n)$.



Kuva 2.2: Esimerkki päätöspuusta

Samoin kuin tilakoneet myös päätöspuut voivat olla hierarkkisia, jolloin lehtisolmu onkin päätöspuu, tai epädeterministisiä. Niiden epädeterminismi voidaan toteuttaa joko probabilistisen valinnan tai sumean logiikan avulla. Näin voidaan vähentää ennalta arvattavuutta pelihahmojen käytöksessä. Päätöspuiden avulla on myös mahdollista toteuttaa oppivia agentteja. Tällöin päätöspuita ei rakenneta etukäteen, vaan niitä muokataan pelin kuluessa. [Millington, 2006]

2.5 STRIPS-tyyppinen suunnittelu

Akateemisissa piireissä ehkä suosituin tekninen ratkaisu tarinankerrontajärjestelmien suunnitteluun on ollut STRIPS-tyyppinen suunnittelu. Suunnittelualgoritmeja käytetään sekä henkilölähtöisessä että tarinalähtöisessä kerronnassa. Henkilölähtöisessä kerronnassa useat agentit suorittavat rinnakkain omia suunnittelualgoritmejaan. Tarinalähtöisessä kerronnassa suunnittelu suoritetaan keskitetysti.

STRIPS-tyyppisessä suunnittelussa maailma esitetään joukkona loogisia lauseita. Maailman tilan lisäksi järjestelmässä määritellään joukko tavoitteita *(goal)*, maailman tilaa kuvaavia loogisia lauseita, jotka järjestelmä pyrkii saattamaan voimaan. Maailman tilaa muutetaan suorittamalla toimintoja *(action)*. Toiminnot muodostuvat esiehdoista *(precondition)* ja vaikutuksista *(effect)*. Toiminto voidaan suorittaa vain, jos esiehdot ovat voimassa. Sekä esiehdot että vaikutukset esitetään loogisten lauseiden konjunktiona. Puhtaassa STRIPS-kielessä maailman tila, tavoitteet ja esiehdot esitetään positiivisten propositiologiikan literaalien konjunktioina. Maailman tilasta puuttuvat literaalit oletetaan epätosiksi. Vaikutuksissa sallitaan myös negatiiviset literaalit. Positiivisten literaalien kuvaamat propositiot merkitään tosiksi ja negatiivisten kuvaamat epätosiksi. Luettavuuden parantamiseksi joissain suunnittelujärjestelmissä vaikutukset on jaettu kahdeksi listaksi, lisäyslistaksi *(add list)* ja poistolistaksi *(delete list)*.

Taulukko 2.1 esittää pienen renkaanvaihtoa käsittelevän suunnitteluongelman STRIPS-kielen laajennoksella, joka sallii negatiiviset literaalit myös esiehdoissa. Alkutilanteessa voimassa on *Sisällä(vararengas,auto)* ja *Kiinni(tyhjä rengas)*. Tavoitteena on *Kiinni(vararengas)*. Tavoitteeseen voidaan päästä kahden eri suunnitelman avulla: voidaan ottaa vararengas esille, irrottaa tyhjä rengas ja kiinnittää vararengas, tai vaihtoehtoisesti kaksi ensimmäistä toimenpidettä voidaan suorittaa päinvastaisessa järjestyksessä.

Kun maailma, toiminnot ja tavoitteet on määritelty, suunnittelutehtäväksi jää etsiä reitti alkutilasta sellaiseen tilaan, jossa tavoitteet ovat voimassa. Tämä voidaan tehdä tila-avaruushaulla alkutilasta kohti tavoitetilaa. Jokaisessa tilassa

Toiminto	Esiehto	Vaikutukset
Poista(vararengas,auto)	Sisällä(vararengas,auto)	¬ Sisällä(vararengas,auto) ∧ Maassa(vararengas)
Irrota(tyhjä rengas)	Kiinni(tyhjä rengas)	¬ Kiinni(tyhjä rengas) ∧ Maassa(tyhjä rengas)
Kiinnitä(vararengas)	Maassa(vararengas) ∧ ¬Kiinni(tyhjä rengas)	¬ Maassa(vararengas) ∧ Kiinni(vararengas)

Taulukko 2.1: Esimerkki yksinkertaisesta suunnitteluongelmasta

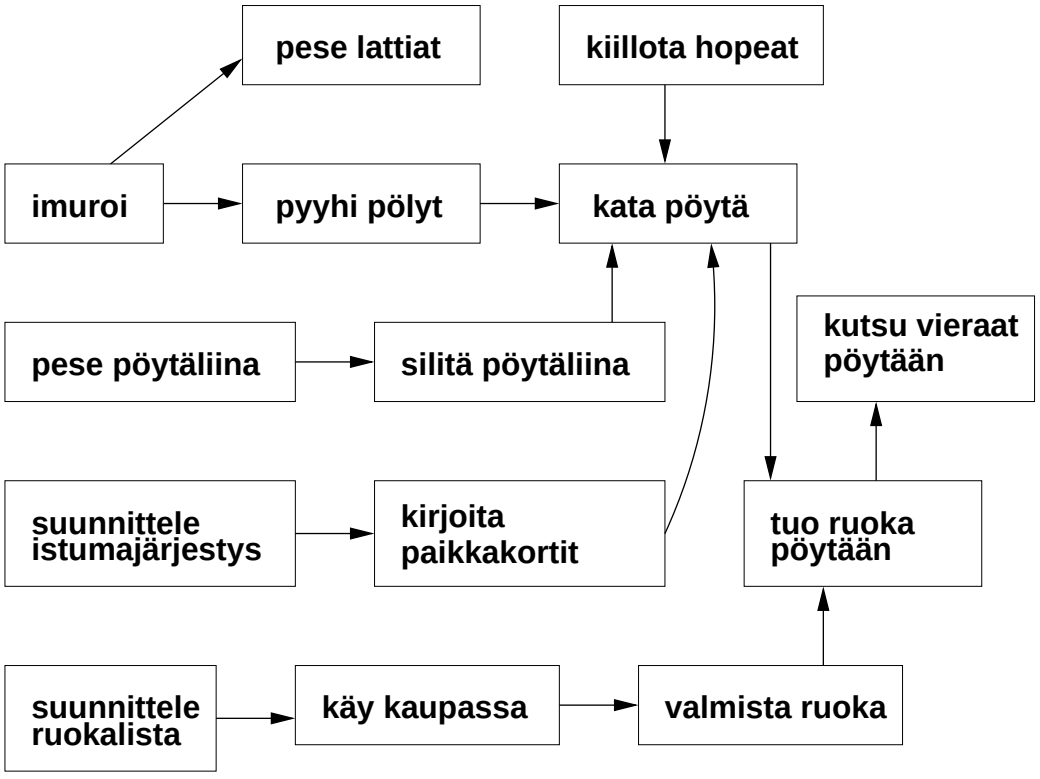
on mahdollista suorittaa ne toiminnot, joiden esiehdot kyseinen tila tyydyttää. Kun toiminnon vaikutukset sovelletaan tilaan, saadaan yksiselitteinen seuraajatila. Tällä tavoin käydään tila-avaruutta läpi jollakin graafihakualgoritmilla, kunnes kohdataan tavoitetila. On selvää, että tällainen haku ilman erittäin hyvää heuristiikkafunktiota on tehoton. Mikäli mahdollisia toimintoja kussakin tilassa on paljon, kuten tilanne useimmissa sovelluksissa on, hakupuun haarautumisaste on suuri. Ilman heuristiikkafunktiota hakupuun haaroja ei voida karsia menettämättä hakualgoritmin optimaalisuutta ja algoritmin suoritusaika nousee eksponentiaaliseksi. Tällainen algoritmi ei luonnollisestikaan kelpaa mihinkään suurempaan sovellukseen.

Jonkin verran tehokkaampi ratkaisu on käyttää taaksepäin etenevää hakua *(backward state-space search)*, joka aloittaa etsinnän tavoitetilasta ja etenee kohti alkutilaa. Menettelytapaan liittyy kuitenkin joitakin käytännön ongelmia: Tavoitetilojen joukko voi olla melko suuri, koska se on määritelty joukkona rajoitteita eikä eksplisiittisenä listana. Lisäksi tilasiirtymien määrittäminen seuraajatilasta edeltäjätilaan ei ole itsestään selvää. Puhtaan STRIPS-järjestelmän kohdalla tämä on kuitenkin tehtävissä varsin yksinkertaisesti. Taaksepäin etenevän haun pääasiallisena etuna on se, että on tarpeen tarkastella vain tehtävän kannalta olennaisia toimintoja. Olennaiset toiminnot ovat sellaisia, jotka saattavat jonkin tavoitekonjunktion literaaleista arvoon tosi. Kussakin tilassa olennaisten toimintojen joukko on yleensä huomattavasti pienempi kuin kaikkien mahdollisten toimintojen joukko. [Russel & Norvig, 2003]

2.6 Osittaisjärjestyssuunnittelu

Edellä esitetyt algoritmit tuottavat täysin järjestetyn joukon toimintoja, jotka suorittamalla järjestelmä siirtyy tavoitetilaan. Tällainen suunnitelma on kuitenkin varsin joustamaton. Osittaisjärjestyssuunnittelun *(partial-order planning)* avulla saadaan suunnitelma, joka on muodoltaan verkko eikä lineaarinen lista. Vain välttämättä tietyssä järjestyksessä suoritettavien toimintojen keskinäinen järjestys on määritelty. Kuvassa 2.3 on esimerkki osittaisjärjestyksessä olevasta suunnitelmasta. Yhdestä osittaisjärjestyksessä olevasta suunnitelmasta voidaan tuottaa useita täydessä järjestyksessä olevia suunnitelmia. Tämän ansiosta pelihahmot voivat valita vapaammin kussakin tilanteessa luontevimmalta vaikuttavan toiminnon, ja mikä tärkeintä, pelaajan tai muiden agenttien tekemät valinnat eivät invalidoi osittaisjärjestyssuunnitelmaa yhtä helposti kuin jäykän lineaarista suunnitelmaa.

Osittaisjärjestyksessä olevat suunnitelmat muodostuvat neljästä osasta: Toiminnot ovat osajoukko suunnitteluongelman toiminnoista, lisättynä toiminnoilla



Kuva 2.3: Esimerkki osittaisjärjestyksessä olevasta suunnitelmasta

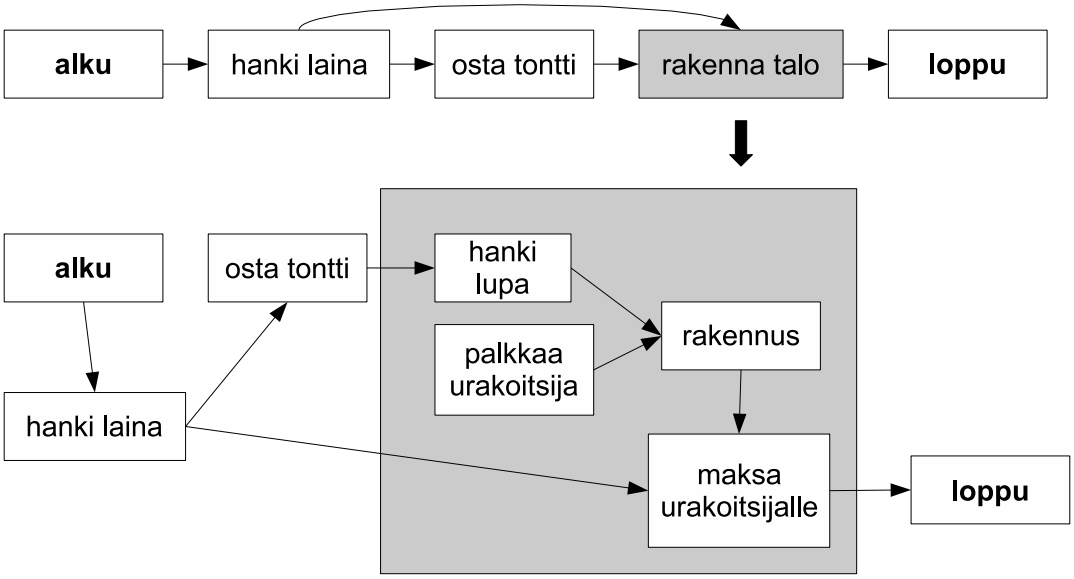
alku ja *loppu*. *Alulla* ei ole esiehtoja, ja sen vaikutuksina on suunnitteluongelman alkutila. *Lopulla* ei ole vaikutuksia, ja sen esiehtona on suunnitteluongelman tavoitejoukko. Järjestysrajoitteet *(ordering constraint)* ovat muotoa A ennen B:tä, mikä tarkoittaa, että A on suoritettava ennen B:tä, muttei välttämättä välittömästi ennen B:tä. Luonnollisestikaan järjestysrajoitteista ei saa muodostua silmukkaa. Kausaaliset linkit *(causal link)* tarkentavat toimintojen välistä yhteyttä määrittelemällä ominaisuuksia, joiden täytyy pysyä voimassa toimintojen välisen ajan. Mikäli toimintojen A ja B välillä on kausaalinen linkki, joka saattaa ominaisuuden p voimaan, ei niiden välillä voida suorittaa mitään toimintoa, joka muuttaa ominaisuuden p epätodeksi. Lisäksi suunnittelun aikana algoritmi ylläpitää joukkoa avoimia esiehtoja *(open preconditions)*, jonka koon se pyrkii pienentämään nollaan aiheuttamatta samalla ristiriitoja.

Osittaisjärjestyssuunnittelualgoritmi aloittaa toimintansa tilanteesta, jossa suunnitelma on tyhjä, eli siihen kuuluvat vain *alku* ja *loppu*, ja avoimien esiehtojen joukko koostuu *lopun* esiehdoista. Jokaisella askeleella avoimista esiehdoista valitaan jokin esiehto p, ja poistetaan se lisäämällä suunnitelmaan jokin toiminto, joka tyydyttää esiehdon. Uuden toiminnon ja p:tä edellyttäneen toiminnon välille muodostetaan kausaalinen linkki, ja mahdolliset ristiriidat poistetaan järjestysrajoitteiden avulla. Jos osittaisjärjestyssuunnitelma on konsistentti, eli siinä ei ole ristiriitoja eikä silmukoita, ja avoimien esiehtojen joukko on tyhjä, suunnitelma on ratkaisu alkuperäiseen ongelmaan. [Russel & Norvig, 2003]

2.7 HTN-suunnittelu

Osittaisjärjestyssuunnittelua voidaan tehostaa muuttamalla se hierarkkiseksi hierarkkisten tehtäväverkkojen eli HTN-suunnittelun *(Hierarchical Task Network Planning)* avulla. Hierarkkiset rakenteet ovat vahva menetelmä monimutkaisuuden hallintaan. HTN-suunnittelussa toiminnot jaetaan eri abstraktiotasoille siten, että ylemmän tason toiminnot määritellään alemman tason toiminnoista koostuvien osittaisjärjestyssuunnitelmien avulla. Tämä tehostaa suunnittelualgoritmin toimintaa, koska se rajoittaa kullakin tasolla valittavina olevien toimintojen määrää ja siten vähentää hakupuun haarautumista. Kuvan 2.4 esimerkki käsittelee talon rakentamisprosessia. Kuvan yläreunassa on esitetty korkean tason suunnitelma, ja kuvan alareunassa toiminto *rakenna talo* on hajotettu pienempiin osiin.

Käytännössä HTN-suunnittelu on osoittautunut tehokkaaksi pitkälti sen vuoksi, että se pystyy hyödyntämään hierarkkisen verkon rakenteeseen upotettua informaatiota. Tämä informaatio löytyy etukäteen rakennetuista suunnitel-



Kuva 2.4: Esimerkki HTN-suunnitelmasta

makirjastoista, jotka määrittelevät, miten korkeamman tason toiminnot rakentuvat alemman tason toiminnoista. Korkean tason toiminnon esiehdot ovat sen hajotelmien ulkoisten esiehtojen leikkaus ja vastaavasti sen vaikutukset ovat sen hajotelmien ulkoisten vaikutuksien leikkaus. Alemman tason toiminnoilla voi lisäksi olla sisäisiä esiehtoja ja vaikutuksia. Tehokkuuden kannalta on olennaista, että nämä eivät näy ylemmälle tasolle. Hyvin järjestettyjen suunnitelmakirjastojen laatiminen onkin vaativa tehtävä. Tietokonepelien kohdalla tämä ei kuitenkaan ole erityisen ongelmallista, koska mahdolliset toimintosekvenssit on joka tapauksessa mietittävä etukäteen melko tarkasti. [Russel & Norvig, 2003]

2.8 BDI-malli

Suunnittelu on luonteva tapa tuottaa koherentteja tarinoita, mutta tavoitteen toinen osa, henkilöiden uskottavuus, ei välttämättä synny pelkästä loogisesta suunnitelmasta. Eräs tapa parantaa henkilöiden uskottavuutta loogiseen päättelyyn perustuvissa henkilölähtöisissä kerrontajärjestelmissä on BDI-malli *(belief-desire-intention model)*. Siinä agenteille on määritelty uskomuksia, toiveita ja aikomuksia. Uskomukset ilmaisevat agentin näkemyksen maailman tilasta. Toiveet määrittelevät agentin tavoitteet ja aikomukset ovat suunnitelmia, joihin agentti on sitoutunut. Uskomusten ja toiveiden muuttuminen voi aiheuttaa muutoksia ai-

komuksiin. Järjestelmän vahvuus on siinä, että sen avulla agentit voivat mallintaa myös muiden agenttien toimintaa ja mielentiloja. Tämä mahdollistaa toisten agenttien toimien ennakoimisen ja omien toimintojen valinnan sen perusteella, mitkä niiden vaikutukset ovat toisten agenttien näkemyksiin ja valintoihin. BDI-malli kuvaa myös luontevalla tavalla sitä, että kaikki henkilöt eivät välttämättä näe maailmaa samalla tavalla. Osa uskomuksista voi olla vääriä. Jokaisella agentilla on ikään kuin oma hieman toisista poikkeava kopionsa pelimaailmasta. [Wooldridge, 2000]

2.9 Sääntöpohjainen päättely

Edelliset suunnitteluun perustuvat menetelmät tarinankerrontaan pohjaavat vahvasti rationaalisen älykkyyden käsitteeseen. Lähtökohdaksi voidaan kuitenkin ottaa myös behavioristinen käsitys älykkyydestä, siis järjestelmä on älykäs, jos se toimii kuten ihminen. Tätä ajattelutapaa edustaa esimerkiksi sääntöpohjainen päättely *(rule-based reasoning)*. Järjestelmälle annetaan joukko erilaisia "peukalosääntöjä", joiden perusteella se valitsee toimintansa. Menetelmä soveltuu erityisen hyvin yksinkertaisten reaktiivisten agenttien toteuttamiseen, mutta sitä on käytetty monimutkaisemmissakin tarinankerrontajärjestelmissä.

Yksinkertaisimmillaan sääntöpohjainen järjestelmä voi olla joukko *if-then*-lauseita. Kukin *if*-osa kuvaa yhden ehdon ja sitä vastaava *then*-osa toiminnon, joka ehdon täyttyessä suoritetaan. Tällaisia järjestelmiä peliteollisuudessa on käytetty runsaasti, joskaan niitä ei läheskään aina ole kutsuttu sääntöpohjaisiksi. Sääntöpohjaiset järjestelmät voivat kuitenkin olla huomattavasti monimutkaisempiakin. Erityisesti asiantuntijajärjestelmissä käytetään hyvinkin monimutkaisia kaupallisia sääntöjärjestelmämoottoreita käyttäviä sovelluksia.

Sääntöjärjestelmä koostuu työmuistista, jossa maailma on kuvattu joukkona assertioita. Sääntöjen ehto-osuudet tarkastelevat näitä assertioita. Toimintaosuudet voivat lisätä sääntökantaan uusia sääntöjä. Tällöin puhutaan varsinaisesta päättelyjärjestelmästä. On mahdollista, että voimassaolevat assertiot tyydyttävät useamman kuin yhden ehdoista. Päättelyjärjestelmissä tällöin tyypillisesti suoritetaan kaikkien tyydytettyjen ehtojen toimintaosat. Päättelytapaa kutsutaan eteenpäin ketjuttamiseksi *(forward chaining)*. Myös taaksepäin ketjuttaminen *(backward chaining)* on mahdollista. Siinä pyritään osoittamaan jo muodostetut hypoteesit oikeiksi etenemällä sääntöjä takaperin *then*-osista *if*-osiin.

Jos sääntöjen *then*-osat koostuvat pelkistä toiminnoista, eivätkä muokkaa sääntökantaa, järjestelmää kutsutaan reaktiojärjestelmäksi. Reaktiojärjestelmis-

sä kaikkien assertioiden tyydyttämien ehtojen toimintaosuuksia ei yleensä haluta suorittaa, vaan niistä täytyy valita vain yksi. Mahdollisia valintamenetelmiä on useita. Yksinkertaisinta on asettaa säännöt prioriteettijärjestykseen siten, että kun jonkin säännön ehto-osa havaitaan tyydytetyksi, voidaan sen toimintaosa suorittaa loppua sääntöjoukosta tarkastelematta. Ohjelmointikielten *if-then*-lauseilla toteutetut järjestelmät käyttävät tätä menetelmää. Muita vaihtoehtoja ovat esimerkiksi tarkimman säännön valitseminen tai viimeksi käytetyn säännön valitseminen. Taulukossa 2.2 esitetty reaktiojärjestelmä käyttää prioriteettijärjestystä. [Winston, 1993]

2.10 Tapauspohjainen päättely

Samankaltaisuuden tunnistaminen ja olemassa olevien tapausten käyttäminen esimerkkeinä ovat olennainen osa inhimillistä älykkyyttä. Tapauspohjainen päättely *(case based reasoning)* perustuu näihin ominaisuuksiin. Siinä järjestelmän muistissa on joukko tapauksia, joista päättelyalgoritmi pyrkii löytämään mahdollisimman hyvin vallitsevaa tilannetta vastaavan. Tarvittaessa valittua tapausta muokataan siten, että se on sovellettavissa käsillä olevaan ongelmaan. Tapaukseen liittyvät toiminnot suoritetaan ja yleensä muunneltu tapaus sekä tieto ratkaisun onnistuneisuudesta tallennetaan tietokantaan. Tapauspohjaisen päättelyn avulla voidaan siis tuottaa oppivia järjestelmiä.

Tapaustietokannan säilyttämiseen voidaan käyttää erityyppisiä tietorakenteita. Yksinkertaisimmillaan tapaukset voivat olla pelkkiä monikkoja, joissa osaa argumenteista käytetään tapausten samankaltaisuuden selvittämiseen, kun taas osa argumenteista kuvaa ratkaisuun kuuluvia askeleita. Myös monimutkaisemmat

Ehto-osa	**Toimintaosa**
Näkyvissä(vihollinen)	Pakene
Nälkä AND Hallussa(ruoka)	Syö(ruoka), not(Hallussa(ruoka))
Nälkä AND Näkyvissä(ruoka)	MeneJaPoimi(ruoka), Hallussa(ruoka)
Nälkä	Etsi(ruoka)
Oletus	Nuku

Taulukko 2.2: Esimerkki yksinkertaista pelihahmoa mallintavasta reaktiojärjestelmästä

esitystavat kuten todistuspuut ovat mahdollisia. Melko tavallinen esitystapa on joukko tilanne-toiminto -pareista muodostuvia sääntöjä.

Menetelmän suurin haaste on tapausten samankaltaisuuden määrittäminen. Tapauksien kun ei ole tarkoituskaan kattaa kaikkia mahdollisia tilanteita, jotka pelimaailmassa tulevat eteen, joten pelkkä samuuden määrittäminen ei riitä. Tässä keskeiseen rooliin nousee tarkasteltavien ominaisuuksien priorisointi: järjestelmän täytyy tarkastella kussakin tilanteessa luokittelun kannalta olennaisia ominaisuuksia. Tätä harvoin pystytään tekemään automaattisesti, vaan apuna joudutaan käyttämään asiantuntijaa. Tietokonepelien tapauksessa asiantuntija on tietenkin pelin suunnittelija, joka kuvaa, millaisessa tilanteessa mikäkin toiminto voidaan suorittaa. Toisaalta yksinkertaisemmissa peleissä tapauksia voidaan etsiä esimerkiksi ihmispelaajan suorittamista mallipeleistä. Taulukon 2.3 esimerkkitapauskanta voisi olla tällainen. Pelin kuluessa tapauskanta täydentyy sitä mukaa, kun hahmo onnistuu löytämään toimiviksi osoittautuvia toimintamalleja esimerkkitapausten ulkopuolisiin tilanteisiin.

Tilanne	**Toiminta**
Oma ase(pistooli), Luoteja(5) Oma ase(nuija) Vihollinen(isompi) Vihollisen ase(nuija)	Ammu
Oma ase(nuija) Vihollinen(isompi) Vihollisen ase(pistooli)	Pakene
Oma ase(nuija) Vihollinen(pienempi) Vihollisen ase(nuija)	Hyökkää nuijalla
Oma ase(pistooli), Luoteja(1) Vihollinen(isompi) Vihollisen ase(nuija)	Pakene
Oma ase(ei asetta) Vihollinen(pienempi) Vihollisen ase(nuija)	Pakene

Taulukko 2.3: Esimerkki tietokonepelin tapauskannasta

Tapauspohjainen menetelmä eroaa sääntöpohjaisesta menetelmästä tietokonepelien tapauksessa lähinnä siten, että kun sääntöpohjaisessa reaktiojärjestelmässä on haasteellista päättää, mikä useista tilanteeseen soveltuvista säännöistä valitaan, on tapauspohjaisessa järjestelmässä keskeisenä ongelmana valita se sääntö, joka on lähinnä käsillä olevaa tilannetta. Yleisesti ollaan sitä mieltä, että tapaukset kannattaa järjestää tavoitteiden mukaan. Muitakin heuristiikkoja tosin on. Voidaan esimerkiksi tarkastella ensin tärkeimpinä pidettyjä ominaisuuksia, useimmin käytettyjä tapauksia tai viimeksi käytettyjä tapauksia. Voidaan myös keskittyä etsimään mahdollisimman tarkasti etsityn kaltaisia tapauksia tai tapauksia, joiden soveltaminen vallitsevaan tilanteeseen on mahdollisimman helppoa. [Luger, 2005]

2.11 Olemassaolevia järjestelmiä

Muutaman viime vuosikymmenen aikana maailmalla on toteutettu useita eri tekoälymenetelmiin pohjautuvia tarinankerrontajärjestelmiä. Tässä aliluvussa esitellään muutamia tunnetuimpia.

2.11.1 Façade

Ehkä tämän hetken vakuuttavin esitys interaktiivisen tarinankerronnan saralla on Michael Mateaksen ja Andrew Sternin interaktiivinen draama Façade. Siinä käyttäjän avatar menee kylään ystäväpariskuntansa, Gracen ja Tripin, luokse. Käyttäjä pystyy kommunikoimaan heidän kanssaan toisaalta vapaan tekstinsyötön avulla, toisaalta liikkumalla kolmiulotteisessa virtuaalimaailmassa, siirtelemällä esineitä ja suorittamalla eleitä, kuten suutelemalla jompaa kumpaa tietokoneen ohjaamista hahmoista.

Façade-arkkitehtuuri koostuu useista osista: Henkilöhahmot on kirjoitettu reaktiivisella suunnittelukielellä ABL:llä. Draaman ohjausmoduuli jaksottaa dramaattiset sykäykset *(beat)*. Eteenpäin ketjuttava sääntöjärjestelmä tulkitsee luonnollista kieltä ja käyttäjän suorittamia eleitä. Animaatiomoottori suorittaa käyttäjälle näkyvät ja kuuluvat elementit, renderöinnin, dialogin, musiikin ja äänitehosteet.

Façade-draaman peruselementit ovat ABL:llä kirjoitettuja dialogitoimintoja, *(joint dialog behavior)*. Ne koostuvat 1-5 lausahduksesta, jotka muodostavat yhtenäisen katkelman dialogia, joka animoituna kestää tyypillisesti muutamia sekunteja. ABL:ssä toiminnot esitetään yhdestä tai useammasta matalan tason toiminnosta muodostuvina tavoitteina. Aktiivinen tavoite valitsee suoritukseen jonkin

toiminnoistaan, jotka muodostuvat joukosta peräkkäin tai rinnakkain suoritettavia askelia. Jos kaikki toiminnot saadaan suoritettua, tavoite onnistuu, eikä ole enää aktiivinen. Jos toiminto epäonnistuu, tavoite pyrkii suorittamaan korvaavan toiminnon. Jos korvaavaa toimintoa ei löydy, tavoite epäonnistuu. ABL pitää kirjaa aktiivisista tavoitteista, toiminnoista ja osatavoitteista aktiivisten toimintojen puun *(active behavior tree)* avulla.

Draaman kaarta ohjataan pääasiassa dramaattisten sykäysten avulla. Yksittäinen sykäys koostuu 10-100 dialogitoiminnosta, jotka käsittelevät samaa juonellista tavoitetta. Tällainen voisi olla esimerkiksi Gracen pakkomielteinen sisustaminen tai tärkeän salaisuuden paljastuminen. Tyypillinen dramaattinen sykäys muodostuu uuden aiheen aloittavasta dialogitoiminnosta, useista runkodialogeista, jotka esittelevät aiheen sekä toiminnosta, jossa Grace ja Trip odottavat aiheeseen liittyvää kommenttia käyttäjältä. Sykäyksen lopettaa oletussiirtymä siinä tapauksessa, että käyttäjä ei sitä ennen aloita uutta aihetta. Jos käyttäjä ei suorita minkäänlaisia toimintoja sykäyksen aikana, se esitetään ennalta suunnitellussa muodossa alusta loppuun.

Dramaattisiin sykäyksiin liittyy varsinaisen dialogitoimintosarjan lisäksi dialogitoimintoja, joiden tarkoituksena on toimia vastauksina käyttäjän antamiin palautteisiin *(beat mix-in)*. Ne muokkaavat käynnissä olevaa sykäystä siten, että käyttäjän kommentti sopii siihen luontevasti. Ne voivat lisätä toimintosarjaan uusia dialogin pätkiä, poistaa vanhoja ja muuttaa olemassa olevien pätkien järjestystä. Lisäksi yleisiä järjestelmän ymmärtämiä aiheita varten on olemassa globaaleja käyttäjän vuorovaikutuksen käsittelijöitä *(global mix-in)*, jotka huolehtivat uuden puheenaiheen esille ottamisesta. Sykäyksen loppuessa tai käyttäjän keskeyttäessä sykäyksen draaman ohjausmoduuli herää ja valitsee suoritukseen uuden sykäyksen maailman tilan ja käyttäjän antamaan palautteen perusteella. Jos keskeytetty sykäys ei ole edennyt käsikirjoittajan määrittelemään huippukohtaan, se pyritään ottamaan suoritukseen myöhemmin. [Mateas & Stern, 2002] [Mateas & Stern, 2005]

2.11.2 Cavazzan, Charlesin ja Meadin järjestelmä

Cavazza, Charles ja Mead ovat rakentaneet HTN-suunnitteluun perustuvan interaktiivisen tarinankerrontajärjestelmän. Interaktiivisuus on toteutettu antamalla käyttäjälle mahdollisuus puuttua tarinan kulkuun huutamalla hahmoille ohjeita ja siirtelemällä 3D-maailmassa olevia esineitä. Yksittäisen hahmon ohjaaminen sen sijaan ei ole mahdollista, joten useimpiin tietokonepeleihin järjestelmä

ei sovellu. Esimerkkisovelluksena ryhmä käyttää suositusta TV:n sitcom-sarjasta poimittua sosiaalista tilannetta, jossa yksi päähenkilöistä on rakastunut toiseen hahmoon ja yrittää saada tämän rakastumaan itseensä.

Tarinankerronta on henkilölähtöistä, jotta tapahtuva käyttäjän vuorovaikutus olisi mahdollista missä tahansa vaiheessa. Juonen eheyttä on edistetty kuvaamalla hahmojen käyttäytymistä roolien ja tavoitteiden avulla. Mahdolliset toiminnot on kuvattu etukäteen rakennetun hierarkkisen tehtäväverkon avulla. Tehtäväverkosta hahmot valitsevat tehtäviä, jotka vastaavat heidän roolinsa tavoitteita. Ylimmän tason tehtävät vastaavat hahmojen pääasiallisia tavoitteita. Esimerkiksi tarinan päähenkilö Ross pyrkii tavoitteeseensa valitsemalla ylimmän tason toiminnon “Rachelin viietteleminen”. Hajotettaessa edellinen toiminto jakautuu useisiin toimintoihin kuten “hanki tietoa Rachelista”, “pyri Rachelin ystäväksi”, “anna lahja” ja “pyydä Rachelia treffeille”. Nämä toiminnot puolestaan jakautuvat useiksi alemman tason tehtäviksi, kunnes hajottaminen on edennyt kaikkein matalimmalle tasolle, jolla toiminnolle löytyy siihen liittyvän animaation suorittava skripti.

Ohjelman käyttämät hierarkkiset tehtäväverkot ovat muodoltaan AND/OR graafeja. Yksittäinen tehtäväsolmu voi siis olla hajotettavissa joko sarjaksi alitehtäviä, jotka kaikki on suoritettava, tai joukoksi alitehtäviä, joista yksi on suoritettava. Tehtäväsarjat ovat täysin järjestettyjä. AND/OR graafit muutetaan AO*-algoritmin avulla suoritettaviksi ratkaisugraafeiksi. AO*-algoritmi käy tehtäväverkkoa läpi syvyys ensin -järjestyksessä vasemmalta oikealle, ja suorittaa kaikki tielleen osuvat primitiiviset toiminnot, joille on määritelty konkreettinen toteutus. Mikäli toiminnon suorittaminen epäonnistuu, algoritmi peruuttaa.

Ei-primitiiviset tehtävät algoritmi hajottaa osatehtäviksi. Hahmojen mieliala ja muut heuristiset arvot vaikuttavat siihen, mitkä rinnakkaisista osatehtävistä valitaan suoritettaviksi. Virtuaalimaailman tapahtumat voivat vaikuttaa näihin heuristisiin arvoihin ja siten muuttaa hahmojen tulevia valintoja. Järjestelmässä käytetään AO*-algoritmin reaaliaikaista muunnosta, joka suorittaa suunnittelua ainoastaan seuraavaan suoritettavissa olevaan tehtävään asti. [Cavazza & al., 2002a] [Cavazza & al., 2002b]

2.11.3 Mimesis

Interaktiiviseen tarinankerrontaan tarkoitettu Mimesis-arkkitehtuuri tukee suunnitelmalähtöistä kerrontaa, vaikka se onkin toteutettu moniagenttijärjestelmänä. Suunnitelmat muodostetaan kausaalisia linkkejä käyttävän hierarkkisen osittais-

järjestyssuunnittelualgoritmin avulla ennen interaktiivisen käytön alkua. Suunnitelma esittää kaikki toiminnot, jotka virtuaalimaailman agenttien on tarinan kuluessa tarkoitus suorittaa, mukaan lukien pelaajan ohjaaman agentin toiminnot. Tarina on siis valmis jo silloin, kun kerronta alkaa, mutta käyttäjän suorittaessa odottamattomia toimintoja sitä täytyy muokata. Käyttäjän vuorovaikutuksen hallinta onkin Mimesis-arkkitehtuurin keskeisin osa-alue.

Käyttäjän tekemät valinnat tarkastetaan tarinan rakenteen esittämää suunnitelmaa vasten. Jos käyttäjän valitsema toiminto on konsistentti suunnitelman kanssa, jatkotoimia ei tarvita. Jos käyttäjä valitsee toiminnon, jonka vaikutukset ovat ristiriidassa tulevien suunnitelmaan kuuluvien toimintojen esiehtojen kanssa, syntyy poikkeus. Mimesis-järjestelmään sisältyy kaksi eri tapaa käsitellä poikkeuksia. Poikkeuksen aiheuttaman toiminnon voidaan antaa toteutua, jolloin suunnitelmaa täytyy korjata siten, että se on jälleen konsistentti. Joissakin tapauksissa toiminnon toteutuminen muodossa, jossa se uhkaa suunnitelman eheyttä, voidaan estää. Tätä kutsutaan väliintuloksi *(intervention)*. Jos juonen kannalta on esimerkiksi välttämätöntä, että käyttäjän ohjaaman hahmon hallussa on tarinan myöhemmässä vaiheessa kolikko, voidaan käyttäjää estää tuhlaamasta kolikkoa juoma-automaattiin määräämällä, että juoma-automaatti on epäkunnossa ja palauttaa välittömästi saamansa kolikot. [Riedl & al., 2003]

2.11.4 DEFACTO

Nikitas Sgourosin DEFACTO-tarinankerrontajärjestelmä on henkilölähtöinen. Keskeisimpänä tarinaa eteenpäin vievänä elementtinä toimivat hahmojen motivaatiot. Käyttäjä ohjaa yhtä tarinan hahmoista ja muut hahmot ovat tietokoneen ohjauksessa. Juonta tuotetaan dynaamisesti tarinan edetessä ja käyttäjän tehdessä omia valintojaan. Juoni muodostuu vaiheittain. Ensin sääntöpohjaisen järjestelmän avulla etsitään kullekin hahmolle vallitsevassa tilanteessa mahdolliset toiminnot. Pelaajan ohjaama hahmo saa valita näistä haluamansa valikosta. Tietokoneen ohjaamien hahmojen toiminnot valitaan dramaattista merkityksellisyyttä kuvaavien sääntöjen avulla. Valitut toiminnot annetaan juonen ratkaisualgoritmille *(plot resolution algorithm)*, joka päättää, mitkä toiminnoista onnistuvat ja mitkä epäonnistuvat. Algoritmi olettaa, että kunkin hahmon toiminnot ovat seurausta hahmojen motiiveista eli henkilökohtaisista tavoitteista *(personal goals)* sekä ryhmien asenteita ja uskomuksia määrittelevistä normeista *(norms)*. Motiivit ajavat hahmoja sekaantumaan toisten hahmojen toimiin joko auttamalla tai estämällä heitä.

Ratkaisualgoritmi koostuu neljästä vaiheesta. Ensin algoritmi selvittää käyttäjälle mahdollisten toimintojen perusteella käyttäjän motiivit. Motiivit muodostavat kaksi joukkoa: joukon S+, eli niiden motiivien joukon, jotka onnistuvat, jos kaikki käyttäjätoiminnot onnistuvat, sekä joukon S- eli niiden motiivien joukon, jotka epäonnistuvat, jos käyttäjätoiminnot onnistuvat. Osa motiiveista voi kuulua käyttäjähahmolle, mutta osa voi kuulua myös muille hahmoille. Jos käyttäjän on esimerkiksi tarkoitus ryöstää jalokivi hahmolta X, seuraa ryöstötoiminnon onnistumisesta käyttäjän tavoitteen *hanki(käyttäjä, jalokivi)* onnistuminen ja *suojele(X, jalokivi)* epäonnistuminen.

Toisessa vaiheessa algoritmi päättelee, kumpi edellisessä vaiheessa määritellyistä motiivijoukoista on tärkeämpi. Yleensä normeja pidetään tärkeämpinä kuin henkilökohtaisia tavoitteita. Tasatilanteessa valitaan käyttäjän toimien seurauksena onnistuvien motiivien joukko tärkeämmäksi. Jos S+ vallitsee, kaikki käyttäjätoiminnot onnistuvat ja niitä estävät toiminnot epäonnistuvat, jos S- vallitsee, kaikki käyttäjätoiminnot epäonnistuvat ja niitä estävät toiminnot onnistuvat. Jos osa käyttäjän tai muiden hahmojen motiiveista on ristiriitaisia, niiden ajamat toiminnot epäonnistuvat automaattisesti.

Kolmannessa vaiheessa kaikki toistaiseksi käsittelemättömät toiminnot määritellään joko onnistumaan tai epäonnistumaan. Jos S+ vallitsee, kaikki S-:n motiivien ajamat toiminnot onnistuvat ja S+:n motiivien ajamat toiminnot epäonnistuvat. S-:n vallitessa tilanne on päinvastainen. Valinnan tarkoituksena on luoda juoneen yllätyksellisyyttä ja tasapainoa. Loppujen toimintojen tulos päätetään satunnaisesti.

Vaiheessa 4 päätetään toimintojen esitysjärjestys. Tähän käytetään kahta eri menetelmää. Hypertekstitarinoiden tapauksessa toiminnot esitetään täydessä järjestyksessä ja 3D-maailmassa tapahtuvassa tarinassa toiminnot tapahtuvat osittaisjärjestyksessä, jotta niiden kausaaliset suhteet säilyvät oikeina. [Sgouros, 2000]

2.11.5 OPIATE

Chris Fairclough'n ja Pádraig Cunninghamin OPIATE-järjestelmä perustuu Vladimir Proppin analyysiin tarinan rakenteesta. Järjestelmä on tarkoitettu interaktiivisten 3D-seikkailupelien tekemiseen. Pelissä voi olla yksi tai useampia pelaajan ohjaamia hahmoja. Tarina muodostuu itsenäisten pelihahmoagenttien ja tarinan ohjaaja-agentin yhteistyön tuloksena. Lähestymistapa on siis yhdistelmä juonilähtöistä ja henkilölähtöistä kerrontaa. Pelihahmot käyttävät sosiaalista simulaatiojärjestelmää, joka säätelee niiden käsitystä muista pelihahmoista. Käsityksiin

vaikuttavat paitsi hahmon omat kokemukset toisista hahmoista myös sen juorun välitysjärjestelmän *(gossiping system)* avulla saama tieto.

Tarinan ohjaaja-agentti tekee suunnitteluratkaisuja perustuen pelimaailmasta haettuihin tietoihin hahmojen asenteista ja sijainneista sekä pelaajan antamasta palautteesta. Suunnitelmat koostuvat sarjoista toimintoja, jotka mikä tahansa kriteerit täyttävistä pelihahmoista voi suorittaa. Suunnitelmien laatimiseen käytetään tapausperustaista suunnittelualgoritmia. Tapaustietokanta koostuu Proppin analyysien tuloksena syntyneistä kansantarujen perusrakenteista. Tapausten samankaltaisuus vallitsevan tilanteen kanssa lasketaan k lähintä naapuria -algoritmin *(k-nearest neighbour algorithm)* avulla. Tilanteeseen soveltuvat tapaukset järjestetään ja niistä valitaan joko parhaiten soveltuva tai niistä muodostetaan yhdistämällä kokonaan uusi tapaus. Tapausten yhdistäminen tehdään vaihtamalla sopivimpaan tapaukseen kuuluvia funktioita paremmin tilanteeseen sopiviin.

Kun sopiva tapaus on valittu, se muutetaan sarjaksi pelimaailman tapahtumia *(event)*. Tätä varten on valittava agentit, jotka suorittavat tapahtumiin liittyvät funktiot, sekä muutettava abstraktit funktiot pelimaailman toiminnoiksi. Agenttien valinta suoritetaan kiinnitysjärjestelmän *(casting system)* avulla dynaamisesti. Kullekin agentille valitaan yksi Proppin määrittelemistä rooleista, sankari, roisto, välittäjä, lahjoittaja, avustaja, väärä sankari, prinsessa, perhe ja kuningas. Sankari on aina pelaaja riippumatta suoritetuista toiminnoista. Muiden roolien kiinnitys riippuu hahmojen toimintahistoriasta ja asenteista. Toimintojen suorittajat valitaan roolien mukaan. Kun funktiolle on valittu suorittava agentti, käydään läpi agentille mahdolliset toiminnot ja valitaan niistä parhaiten haluttua funktiota vastaava. Toiminnot käsittävät animaation sekä yksinkertaista dialogia.

Pelissä voi olla samaan aikaan useita aktiivisia tapauksia. Tällöin pelaaja voi valita niiden joukosta kullakin hetkellä suoritettavan tapauksen. Jos pelaaja etenee uudelle alueelle, suorituksessa olevat tapaukset voivat jäävät taustalle odottamaan tilannetta, johon ne jälleen sopivat. Pelaaja voi lisäksi vaikuttaa juoneen suorittamalla toimintoja, jotka muuttavat muiden hahmojen asenteita häntä kohtaan. Asennemuutokset voivat vaikuttaa suoraan uusien tapausten valintaan. [Fairclough & Cunningham, 2004]

2.11.6 Erasmatron / Storytron

Chris Crawfordin suunnittelema Erasmatron-tarinankerrontajärjestelmä [Crawford, 1999] on lähtökohdiltaan lähempänä Artifistia kuin muut tässä luvussa esi-

tetyt järjestelmät. Erasmatron-järjestelmän pohjalta on toteutettu myöhemmin Storytron-järjestelmä [Storytron, 2006], jota tässä lähinnä esitellään. Se ei käytä suoraan mitään tässä luvussa käsiteltyjä menetelmiä, vaan sen toiminta perustuu useiden näyttelijöiksi *(actor)* kutsuttujen agenttien suorittamiin verbeihin *(verb)*. Tarina muodostuu näyttelijöiden suorittamista verbeistä.

Lause *(sentence)* on yksittäinen juonta eteenpäin vievä askel, jonka joku henkilöhahmoista, draaman näyttelijöistä, suorittaa. Se voi olla yksittäinen lausahdus tai kokonainen puhe, ja siihen voi liittyä suorittavan näyttelijän lisäksi useita muitakin hahmoja. Lause koostuu sanoista, joista tärkein on verbi. Verbi määrittää toimintoon liittyvät roolit, jotka näyttelijät omaksuvat, sekä toiminnon ehdot ja seuraukset. Ehdot ja seuraukset on esitetty skriptien avulla. Suorittamattomia lauseita kutsutaan suunnitelmiksi *(plan)*. Tarinankerronta muodostuu vuorotellen suoritettavista reaktiokierroksista *(reaction cycle)* ja toimintakierroksista *(action cycle)*. Kierrosten aikana näyttelijät saavat kukin vuoronsa prioriteettijärjestyksessä.

Reaktiokierroksen aikana näyttelijät muodostavat uusia suunnitelmia edellisen kierroksen tapahtumien *(event)* perusteella. Vuorollaan kukin näyttelijä tutkii edellisen tapahtuman rooleja ja pyrkii omaksumaan jonkin niistä. Jos yhtään sopivaa roolia ei löydy, näyttelijä siirtää vuoron seuraavalle. Näyttelijä suorittaa rooliin liittyvän reaktioskriptin, joka muuttaa hänen tunnetilaansa. Tämän jälkeen hän siirtyy muodostamaan reaktioon liittyviä suunnitelmia. Jos näyttelijä on tietokoneen ohjaama, hän valitsee suunnitelmista parhaalta vaikuttavan houkuttelevuusskriptin *(desirability script)* avulla. Pelaajan ohjaaman näyttelijän valinnan tekee pelaaja.

Toimintakierroksen aikana näyttelijät pyrkivät suorittamaan suunnittelemansa lauseet ja muuttamaan ne tapahtumiksi. Ensin näyttelijä tarkastaa, onko hän vapaa suorittamaan toiminnon. Näyttelijä, joka on parhaillaan suorittamassa toista toimintoa, ei ole vapaa. Tämän jälkeen näyttelijä tarkastaa, mitkä hänen suunnitelmistaan ovat toteutettavissa vallitsevassa tilanteessa. Jos tällaisia suunnitelmia ei ole, näyttelijä luopuu vuorostaan. Muussa tapauksessa hän suorittaa yhden suunnitelmista. Suunnitelma muuttuu tapahtumaksi, josta tiedotetaan kaikille tapahtuman yleisönä toimineille agenteille. Näyttelijä suorittaa kaikki tilanteessa suoritettavissa olevat suunnitelmat ja siirtää suoritusvuoron seuraavalle agentille.

Storytronissa on useita samankaltaisia ominaisuuksia kuin Artifistilla. Erityisesti sen verbeistä ja reaktioista muodostuva verkko muistuttaa läheisesti alakohdassa 4.2.1 esitettävää toimintograafia. Storytron on kuitenkin suunnattu olen-

naisesti erilaiseen tarkoitukseen, nimittäin pelkkien interaktiivisten tarinoiden tai näytelmien toteuttamiseen, eikä varsinaisiin tietokonepeleihin. Mahdollisuudet hahmojen ohjaamiseen ovat sen vuoksi rajallisemmat.

3 Artifist-järjestelmän yleiskuvaus

Artifist on tarinankerrontaan tarkoitettu moniagenttijärjestelmä, jossa tietokonepelin hahmot tekevät autonomisesti toimintoja. Tarina muodostuu agenttien toiminnoista. Artifistin lähestymistapa on siis vahtasti henkilölähtöinen.

3.1 Toimintafilosofia

Monet olemassaolevista tarinankerronta-algoritmeista käyttävät primäärisenä päättelyjärjestelmänään jonkinlaista suunnittelualgoritmia. Niissä päätökset tehdään luomalla mahdollisista toiminnoista ketjuja ja valitsemalla näistä ketjuista se, joka vaikuttaa parhaalta tai joka johtaa jonkin tavoitteen toteutumiseen. Menetelmä on tieteelliseltä pohjaltaan vankka ja mallintaa hyvin ihmistä rationaalisena olentona. Siihen liittyy kuitenkin joitakin ongelmia pelikäytössä.

Ensimmäinen ongelma on ilmeinen - ajan kulutus. Tietokonepelien tekoälyn on oltava reaaliaikaista. Silminhavaittava hitaus ei-pelaaja-hahmon toiminnassa rikkoo immersion. Jos mahdollisia toimintoja on paljon, optimaalisen suunnitelman laatiminen on hidasta. Hyvät likimääräisratkaisutkin ovat yleensä melko hitaita.

Toisaalta tietokonepeleissä yleensä pyritään mallintamaan ei-pelaaja-hahmot mahdollisimman uskottavina ja inhimillisen oloisina mieluummin kuin täysin rationaalisina toimijoina. Kuinka usein me ihmiset loppujen lopuksi teemme päätöksiä pohdittuamme erilaisia vaihtoehtoja? Toki jotkin päätökset edellyttävät suunnittelemista, mutta useimmiten me loppujen lopuksi teemme sitä, mitä meitä huvittaa tehdä, tai sitä, mitä olemme tottuneet tekemään: syömme kun on nälkä tai kun on ruoka-aika. Tällaisten rutiininomaisten toimintojen mallintamiseen suunnittelu tuntuu hiukan liian järeältä työkalulta.

Artifistin suunnittelufilosofia lähtee ajatuksesta, jonka mukaan tietokonepeleihin riittävät agentit, jotka ikään kuin esittävät haluttuja rooleja sen sijaan että eläisivät niitä. Ajatellaanpa vaikkapa keskiaikajuhlia, joille kiinnitetään ryhmä näyttelijöitä esittämään improvisoitua näytelmää. Näyttelijöille kerrotaan ensin yhteisesti joukko käyttäytymissääntöjä ja ajan henkeen sopivia puheenparsia. Tämän jälkeen kullekin näyttelijälle annetaan rooli ja siihen liittyvät luonteenpiirteet ja ominaisuudet. Henkilöhahmojen väliset suhteet ja alkutilanne sovitaan, ja näytelmä voi alkaa.

Vastaavasti tietokonepelin suunnittelija määrittelee joukon sääntöjä ja toimintatapoja, joita agentit noudattavat. Kullekin agentille määritellään henki-

lökohtaiset ominaisuudet, tarvittavat tiedot pelimaailmasta ja suhteet muihin agentteihin. Tämän jälkeen agentit alkavat suorittaa tarjolla olevia toimintoja omien ominaisuuksiensa ja tietämyksensä mukaisella tavalla. Tarina muodostuu agenttien valitsemista toiminnoista, aivan kuin improvisoitu näytelmä muodostuu näyttelijöiden valinnoista.

Toinen lähtökohta järjestelmän suunnittelussa on ollut käytön helppous. Tarinan generointi liittyy olennaisesti pelin konseptisuunnitteluun, ja näin ollen myös pelisuunnittelijan pitäisi pystyä osallistumaan siihen. Tästä syystä järjestelmän perusrakenteiksi valittiin päätöspuut ja graafit, jotka ovat ymmärrettäviä ja intuitiivisia rakenteita myös ohjelmointiin perehtymättömälle henkilölle. Ajatuksena on, että peli voidaan käsikirjoittaa piirtämällä puita ja graafeja, ja kirjoittamalla niiden solmuihin luonnollisella kielellä kuvaus solmun sisältämästä toiminnosta ja siitä, millä ehdolla toiminto suoritetaan. Toteutusvaiheessa ohjelmoijan vastuulle jää ainoastaan sanallisia kuvauksia vastaavien toteutusten kirjoittaminen tarvittaville funktioille.

Graafien ja päätöspuiden etuna on myös niiden läpinäkyvyys. Virheiden paikantaminen on yleensä melko nopeaa ja virhetoimintojen korjaaminen sitä kautta helppoa. Päätöspuut ja graafit määrittelevät tarkasti, millaiset toiminnot ovat missäkin tilanteessa mahdollisia ja mitkä eivät. Mahdottoman toiminnon ei periaatteessa missään tilanteessa pitäisi päätyä suoritukseen. Tietokonepeleissä tämä on erityisen tärkeä ominaisuus: vaikka monissa tekoälysovelluksissa satunnaiset karkeat virheet ovat hyväksyttäviä, mikäli ratkaisut ovat keskimäärin hyviä, ei sama päde tietokonepeleihin. Niissä tavoitteena on ennen kaikkea uskottavuus, ja hahmo, joka suorittaa silloin tällöin silminhavaiten järjettömiä toimintoja, ei luonnollisestikaan ole uskottava.

Toisaalta täysin rationaalisesti ja deterministisestikään toimivat hahmot eivät välttämättä ole uskottavia eivätkä ainakaan kiinnostavia. Täysin deterministinen toiminta luo myös mahdollisuuden dominoivien strategioiden syntyyn: kun pelaaja huomaa hahmon toimivan aina tietyllä tavalla tietyssä tilanteessa, hän voi käyttää tuota tietoa hyväkseen tavalla, joka selvästi heikentää pelin haasteellisuutta. Liian determinismin välttämiseksi agenttien suorittamat valinnat ovat Artifistissa aina oletusarvoisesti probabilistisia. Tosin joissakin tilanteissa mielekkäitä vaihtoehtoja voi olla vain yksi.

3.2 Toteutusnäkökohtia

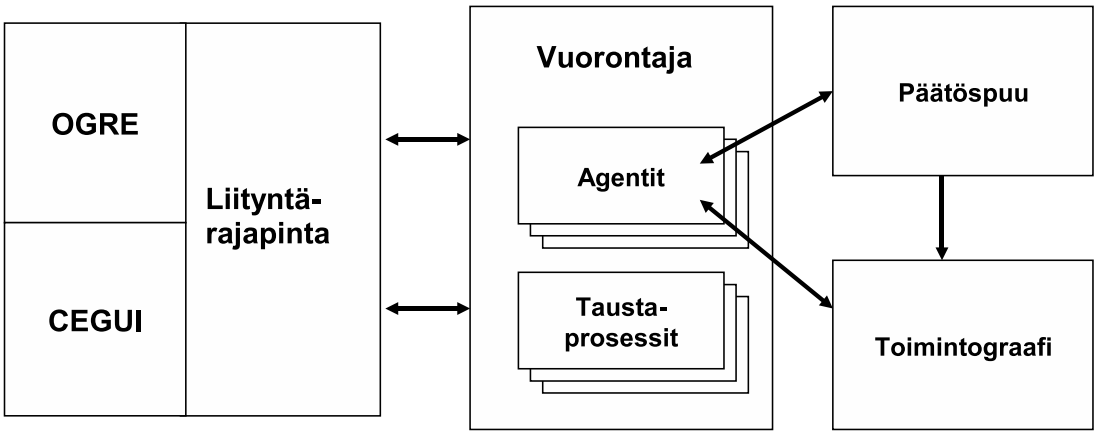
Yhtenä Artifistin suunnittelun perustavoitteena on ollut käyttökelpoisuus myös toteutettaessa kaupallisia PC-koneille ja pelikonsoleille tarkoitettuja pelejä. Tämä

tavoite on sanellut tiettyjä toteutusratkaisuja. Käytetty kieli on C++ ja kehitysympäristö Windows-käyttöjärjestelmässä toimiva Visual Studio. Toteutuksesta on kuitenkin pyritty tekemään mahdollisimman ympäristöriippumaton siten, että sen siirtäminen esimerkiksi Unix-ympäristöön ei olisi kovin työlästä. Lisäksi toteutus on kautta linjan oliokeskeinen ja helposti laajennettavissa.

Toteutuksessa on käytetty joitakin kolmannen osapuolen ohjelmistokomponentteja: 3D-grafiikan toteutukseen on käytetty Ogre3D-kirjastoa [OGRE, 2006] ja graafiset käyttöliittymäkomponentit ovat CEGUI-kirjastosta [CEGUI, 2006]. Lisäksi Boostin [Boost, 2006] kirjastoja on käytetty useisiin tehtäviin.

3.3 Järjestelmän rakenne

Kuva 3.1 esittää Artifistin perusrakennetta. Sen ydin koostuu kolmesta pääkomponentista: päätöspuista ja toimintograafista sekä agentti- ja taustaprosesseista, joita suoritetaan Artifistin vuorontajassa. Agentit käyttävät päätöspuita ja toimintograafia valitessaan suoritettavia toimintoja. Liityntä grafiikkamoottoriin ja muihin ulkopuolisiin komponentteihin tapahtuu geneerisen liityntärajapinnan kautta.



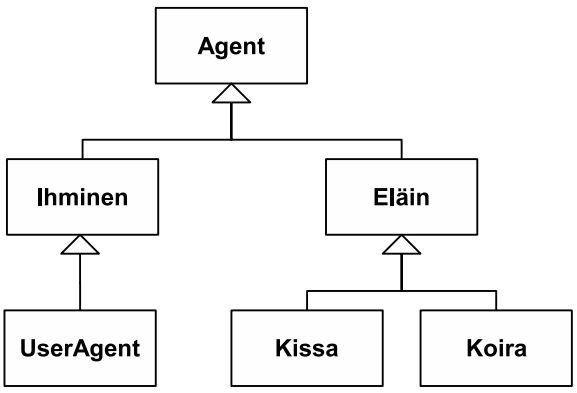
Kuva 3.1: Artifistin rakenne

3.4 Agentit

Pelin suunnittelija voi määritellä useita eri agenttityyppejä, jotka periytyvät joko suoraan tai välillisesti Artifistin sisäisestä *Agent*-luokasta. Pelaajan ohjaama agentti voi olla mitä tahansa pelin suunnittelijan määrittelemää tyyppiä, mutta tämän lisäksi se periytyy Artifistin sisäisestä *UserAgent*-luokasta kuvan 3.2 esittämällä tavalla.

Kaikki tietokoneen ohjaamat agentit käyttävät samaa perusalgoritmia valintojen tekemiseen. Niiden tekemät valinnat voivat kuitenkin poiketa toisistaan huomattavasti, koska eri tyyppisillä agenteilla voi olla erilaiset päätöspuut. Esimerkiksi ihmiselle ja koiralle tarjolla olevat toiminnot luultavasti poikkeavat huomattavasti toisistaan. On myös mahdollista antaa saman tyyppisille agenteille toisistaan poikkeavia päätöspuita, jos niiden roolit ovat kovin erilaiset. Lisäksi agenttityyppien tietämyskannat voivat olla rakenteeltaan erilaisia ja joillakin agenttityypeillä voi olla ominaisuuksia tai omaisuutta, jotka muilta puuttuvat. Saman tyyppiset agentit poikkeavat toisistaan ainoastaan tietämyskantojensa sisällön ja attribuuttiensa arvojen perusteella.

Algoritmia suoritetaan askel kerrallaan aina, kun agentti saa suoritusaikaa vuorontajalta. Eri agenteilla ja taustaprosesseilla voi olla erilainen prioriteetti, jonka mukaan suoritusaikaa annetaan. Jokaisella askeleella agentti tarkastaa ensin, onko se saanut viestejä toisilta agenteilta tai järjestelmältä. Jos viestejä on tullut, agentti käsittelee niistä prioriteetiltaan korkeimman. Mikäli käsiteltävä

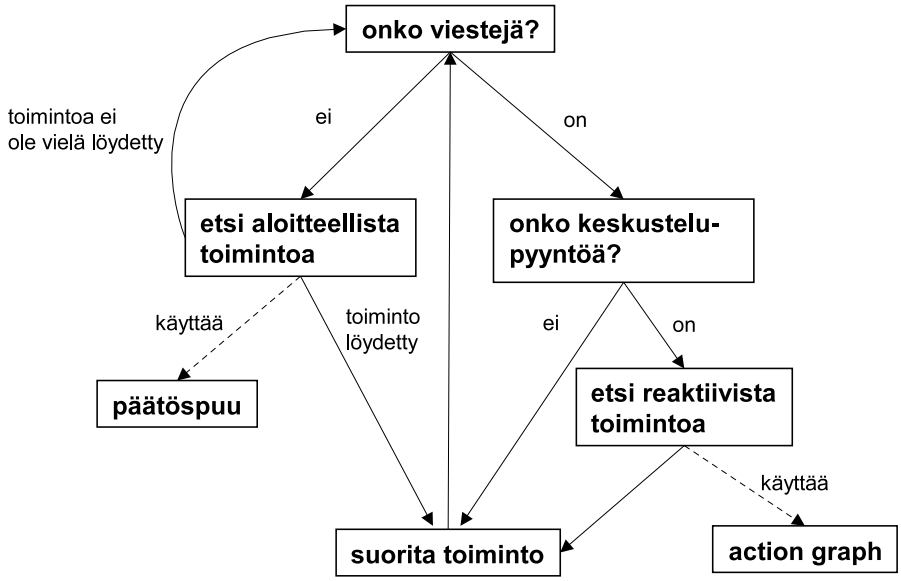


Kuva 3.2: Agenttihierarkia

viesti on keskustelupyyntö, agentti etsii toimintograafista seuraavan toiminnon ja suorittaa sen. Muussa tapauksessa viesti sisältää informaatiota ympäristöstä. Tällaiseen informaatioon liittyy reaktiivinen toiminto, jonka agentti suorittaa käsitellessään informaation. Jos viestejä ei ole, agentti pyrkii etsimään tilanteeseen sopivan aloitteellisen toiminnon käyttäen apunaan päätöspuuta.

3.5 Taustaprosessit

Taustaprosessit ovat algoritmeja, joita suoritetaan samaan tapaan askel kerrallaan kuin agenttien algoritmia. Askellusalgoritmin toteutus on täysin pelinohjelmoijan päätettävissä. Se voi muuttaa sekä globaaleja että agenttien paikallisia tietoja. Tällä on pyritty varmistamaan, että Artifist-järjestelmällä pystytään mallintamaan kaikkia haluttuja ilmiöitä. Taustaprosessit on tarkoitettu lähinnä sellaisten pelin kannalta olennaisien ilmiöiden mallintamiseen, jotka eivät riipu yksittäisten agenttien toimista. Esimerkkejä tällaisista ilmiöistä ovat sää, vuodenajat ja vuorokauden ajat. Myös muut luonnolliset ajan mukana tapahtuvat muutokset, kuten hahmojen vanheneminen, haavojen paraneminen, nälän lisääntyminen ja tunteiden haalistuminen, voidaan toteuttaa taustaprosessien avulla.



Kuva 3.3: Agenttien käyttämä algoritmi

3.6 Toiminnot

Toiminnot on mallinnettu *ActionNode*-olioina. Kullakin toiminnolla on varsinainen toimintaosa, joka sisältää toiminnon vaikutukset, siis siihen liittyvät fyysiset tapahtumat ja muutokset agenttien tietämykseen maailmasta. Lisäksi toimintoon liittyy kuvaus, joka toiminnosta annetaan käyttäjälle valintatilanteessa sekä esiehto-osa. Esiehto kertoo, mikä toiminnon utiliteetti on vallitsevassa tilanteessa. Sekä toiminta- että esiehto-osien toteuttaminen kuuluu pelin ohjelmoijan vastuulle. Ne liitetään olioihin funktio-osoittimien avulla. Toiminnot voivat liittyä keskusteluun tai olla irrallisia. Erilaisiin tarkoituksiin käytetään erityyppisiä toimintoja, jotka on esitelty taulukossa 3.1. Toiminnot muodostavat kommunikaatioketjuja, jotka päättyvät *EndAction*- tai *EndInteraction*-solmuihin. Yhden interaktion aikana voidaan suorittaa useita kommunikaatioketjuja. Tyypillisesti kommunikaatioketjut ovat yhtä aihetta koskevia keskustelunpätkiä, ja uuden kommunikaatioketjun aloittaminen vastaa keskustelunaiheen vaihtamista.

StartInteraction	aloittaa interaktion tai liittää uuden jäsenen olemassa olevaan interaktioon, jatkaa kommunikaatioketjua kommunikaatiopyynnöllä
EndInteraction	lopettaa interaktion tai poistaa jäsenen interaktiosta ja lopettaa kommunikaatioketjun
LeaveInteraction	irrottaa jäsenen interaktiosta, mutta jatkaa ketjua kommunikaatiopyynnöllä
BasicAction	jatkaa kommunikaatioketjua kommunikaatiopyynnöllä
EndAction	lopettaa kommunikaatioketjun
SimpleAction	suorittaa vain toiminnon
ChoiceAction	voi olla mikä tahansa edellisistä, edustaa kokonaista joukkoa toimintoja (esim. soita X:lle, Y:lle, Z:lle...)

Taulukko 3.1: Toimintotyypit

4 Primäärinen päättelyjärjestelmä

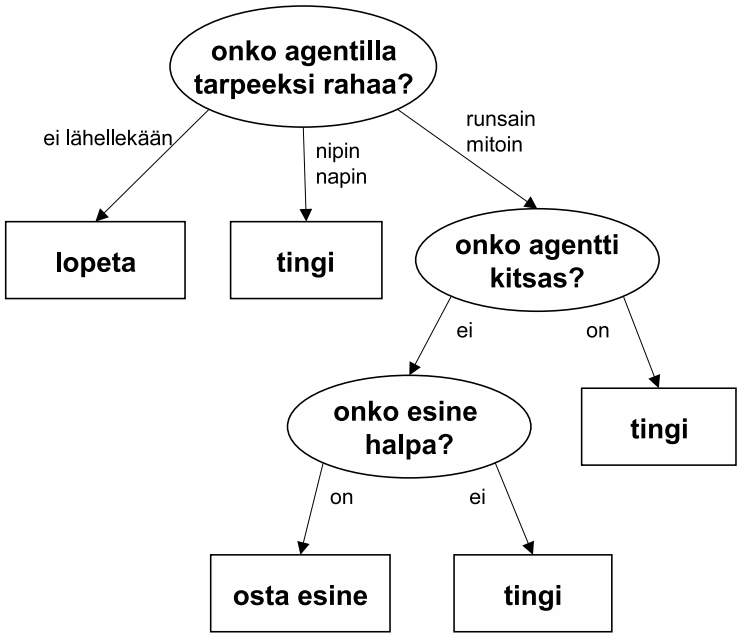
Olennaisin osa Artifist-järjestelmää on algoritmi, joka valitsee ja suorittaa toiminnot, joista tarina muodostuu. Primäärisesti tähän käytetään algoritmia, joka hyödyntää probabilistisia päätöspuita, toimintagraafeja ja agenttien viestinvälitystä. Samaa algoritmia suoritetaan vuorotellen kullekin agentille ja taustaprosessille askel kerrallaan. Algoritmi ei sisällä minkäänlaista suunnittelua, agentit vain toimivat niin kuin ne valitsevassa tilanteessa yleensäkin tekevät. Toiminnot on jaettu kahteen ryhmään: aktiivisiin ja reaktiivisiin toimintoihin.

4.1 Aloitteelliset toiminnot

Aloitteellisten toimintojen valitsemiseen käytetään probabilistisia päätöspuita. Niiden sisäsolmut ovat ehtosolmuja, jotka sisältävät tiedon lapsisolmuista, sekä evaluaatiofunktion, joka antaa kunkin lapsisolmun utiliteetin vallitsevassa tilanteessa. Evaluaatiofunktion kirjoittaminen on pelin ohjelmoijan vastuulla. Lapsisolmujen määrälle ei ole ylärajaa, mutta niitä on oltava vähintään kaksi, jotta valinta voidaan mielekkäästi suorittaa. Lehtisolmut ovat joko toimintoja tai alipuita. Kuvassa 4.1 on pieni esimerkki päätöspuusta. Artifistin päätöspuut eivät ole matemaattisessa mielessä puita vaan suunnattuja silmukattomia graafeja, sillä sama alipuu voi esiintyä niissä useampaan kertaan.

Aloitteellinen toiminto voidaan valita, kun agentti tai agentin kanssa samassa interaktiossa olevat agentit eivät ole suorittamassa kommunikaatioketjua, eikä agentilla ole viestejä odottamassa. Tällöin agentti ryhtyy etsimään sopivaa aloitteellista toimintoa päätöspuusta. Tällainen toiminto voisi olla esimerkiksi uuden interaktion aloittaminen, uuden etenemiskohteen valinta tai kysymyksen esittäminen jollekulle. Kaikilla agenteilla ei tosin tarvitse olla lainkaan päätöspuuta, jolloin ne eivät myöskään kykene tekemään minkäänlaisia aloitteellisia toimintoja. Statistin rooleissa puhtaasti reaktiiviset agentit voivat olla tarpeellisia. Jos päätöspuuta ei ole, askellusalgoritmi vain tarkastaa agentin viestijonon ja palaa.

Aloitteellisen toiminnon etsintäalgoritmi etenee jokaisella askeleella yhden solmun alaspäin päätöspuussa, kunnes se saavuttaa sellaisen lehtisolmun, joka on toiminto. Näin varmistetaan, että syvänkään päätöspuun käsitteleminen ei aiheuta tilannetta, jossa ympäristöstä tulevia viestejä tai keskustelupyyntöjä ei käsitellä silminhavaittavan pitkään aikaan. Sopivan haaran valinnassa käytetään tyypillisesti probabilistista menetelmää, mutta myös deterministinen valinta on mahdollinen. Tämä on tarpeen silloin, kun jotkut toiminnot ovat vallitsevassa tilantees-



Kuva 4.1: Osa agentin päätöspuuta

sa mahdottomia. Tällöin johonkin haaraan siirtymisen todennäköisyys voi olla nolla. Esimerkiksi keskustelussa jotkut keskustelunaiheet saattavat olla tarjolla agentille vain siinä tapauksessa, että puhekumppani on naispuolinen, tai ostoksen voi suorittaa vain, jos siihen on riittävästi rahaa. Kun toiminto on löydetty päätöspuusta, se suoritetaan. Mikäli toiminto on yksittäinen *SimpleAction* tai *EndInteraction*, agentin algoritmi jatkaa seuraavalla suoritusvuorolla evaluointia päätöspuun juuresta. Muussa tapauksessa agentti ja muut mahdollisesti interaktiossa olevat agentit siirtyvät suorittamaan kommunikaatioketjua, joka muodostuu reaktiivisista toiminnoista.

Aiemmin todettiin, että suunnitteluun perustuvien algoritmien ongelmana on usein liian suuri ajankulutus. Päätöspuun kohdalla tämän ei pitäisi muodostua ongelmaksi, jos sovellukset tehdään järkevästi, eli puusta pyritään rakentamaan kohtuullisen tasapainoinen eikä evaluaatiofunktioista tehdä kohtuuttoman raskaita. Jos oletetaan, että puu on tasapainoinen eli korkeudeltaan logaritminen solmujen määrään nähden ja evaluaatiofunktiot ovat vakioaikaisia, kuluu aloitteellisen toiminnon valintaan korkeintaan logaritminen määrä askelia. Pahimmillaan, puun ollessa mahdollisimman epätasapainoinen, sen korkeus on lineaarinen kaikkien aloittellisten toimintojen suhteen. Tällöin aloitteellisen toiminnon valinta on pahimmassa tapauksessa lineaarinen. Solmujen lapsiluvun kasvattaminen nostaa logaritmin kantalukua ja siten yleensä tehostaa algoritmia.

4.2 Reaktiiviset toiminnot

Reaktiivisia toimintoja on kahta perustyyppiä: vastauksia kommunikaatiopyyntöihin ja reagointia ympäristön ärsykkeisiin.

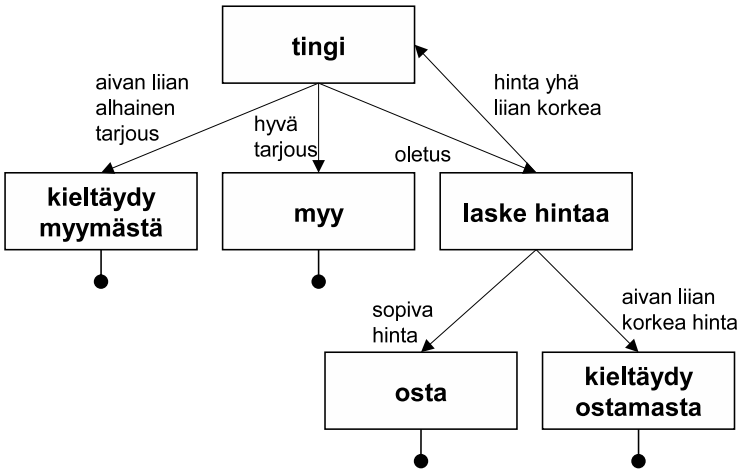
4.2.1 Reagointi kommunikaatiopyyntöihin

Agentin kommunikaatio toisten agenttien kanssa interaktiotilanteessa tapahtuu suorittamalla kommunikaatioketjuja, jotka muodostuvat kuljettaessa toimintagraafia läpi. Toimintagraafi on suunnattu graafi, joka koostuu eri tyyppisistä toiminnoista siten, että kommunikaatioketjuun kuulumattomien ja sen lopettavien solmujen (*SimpleAction*, *EndAction* ja *EndInteraction*) lähtöaste on nolla. Muiden toimintojen lähtöasteen tulee olla nollaa suurempi, jotta kommunikaatioketjua on mahdollista jatkaa. Lisäksi on tärkeää huomata, että minkään solmun naapurisolmujen utiliteetit eivät missään tilanteessa saa olla kaikki nollia. Mikäli tästä ei jonkin solmun kohdalla voida varmistua, tulee solmulle määritellä ylimääräiseksi naapuriksi oletustoiminto, jonka utiliteetti on aina nollasta poikkeava.

Toimintagraafi voi olla ja yleensä onkin epäyhtenäinen. Tyypillisesti kommunikaatioketjun aloittaviin solmuihin päästään vain päätöspuun kautta. Lisäksi graafissa voi olla osia, joihin päästään käsiksi ainoastaan agentin omistamien informaatioalkioiden sisältämien kahvojen kautta, jolloin ne ovat saavutettavia vain niille agenteille, joilla on hallussaan kyseinen informaatio. Informaatioalkioita käsitellään tarkemmin luvussa 5.

Kuvan 4.2 esimerkissä solmut osta, myy, kieltäydy ostamasta ja kieltäydy myymästä ovat *EndAction*-solmuja. Esimerkistä nähdään myös, että toimintagraafissa voi olla silmukoita. Tällöin on huolehdittava, että silmukan suoritus päättyy lopulta, kuten esimerkkitapauksessa, jossa hinta laskee kunnes myyjä tai ostaja kyllästyy tai kauppa syntyy. Vuoron vaihto määritellään toimintosolmun *TargetTypen* avulla. Se voi olla *SELF*, jolloin suoritus siirtyy agentille itselleen, *SINGLE*, jolloin kommunikaatiopyyntö lähetetään vain kohdeagentille, tai *MULTIPLE*, jolloin viesti lähetetään kaikille interaktion osallistujille lähettävää agenttia lukuunottamatta.

Kommunikaatiopyyntöön liitetään osoitin juuri suoritettuun toimintoon. Kun vastaanottava agentti huomaa kommunikaatiopyynnön viestijonossaan, se tutkii edellisen toiminnon naapuritoimintojen utiliteetit vallitsevassa tilanteessa ja tekee probabilistisen valinnan niiden joukosta. Agentti suorittaa valitun toiminnon ja, mikäli toiminto ei ole kommunikaatioketjun lopettava, jatkaa ketjua lähettämällä kommunikaatiopyynnön seuraavalle vastaanottajalle. Kommunikaatioket-



Kuva 4.2: Osa toimintagraafia

jun lopettavassa toiminnossa lähetetään kaikille interaktion jäsenille viesti, jossa kerrotaan kommunikaatioketjun päättyneen. Tällöin agentit voivat ryhtyä etsimään tilanteeseen sopivaan aloitteellista toimintoa.

Tyypillisesti kommunikaatioketjuja on järkevää suorittaa vain interaktiossa, vaikka toimintojen kohteeksi onkin mahdollista määritellä agentti itse: toimintosarja, jossa agentti suorittaa peräkkäin operaatioita, kannattaa yleensä mallintaa yksittäisenä toimintona, koska tuollaisessa tilanteessa viestinvälitys aiheuttaa turhaa kuormaa järjestelmälle. Poikkeuksena ovat toiminnot, jotka kuluttavat niin paljon aikaa, että peli näyttäisi jäävän jumiin jos ne suoritettaisiin atomisina, sekä moniportaisien käyttäjävalintojen tekeminen, josta puhutaan enemmän kohdassa 4.3.

Kommunikaatiopyyntöihin reagointi on selvästi riittävän nopeata, mikäli utiliteettifunktiot on toteutettu kohtuullisen tehokkaasta. Sopivan toiminnon valinta on lineaarista edellisen toiminnon naapureiden määrään nähden. Koska toimintagraafi on pelin suunnittelijan manuaalisesti kokoama, on kohtuullista olettaa, että naapureiden määrä on aina jokin pienehkö vakio. Ainoastaan *ChoiceAction* -solmujen käyttö voi kasvattaa vaihtoehdon joukon vakiokoon ulkopuolelle, koska silloin yhdestä solmusta voi generoitua periaatteessa rajaton määrä vaihtoehtoja. Tällaiset tilanteet ovat kuitenkin pelin pelattavuuden kannalta niin ongelmallisia, että niitä tuskin käytännössä juurikaan esiintyy.

4.2.2 Reagointi ympäristön ärsykkeisiin

Kommunikaatiopyyntöjen lisäksi agentti voi saada myös muun tyyppisiä viestejä, ärsykkeitä ympäristöstä. Tuollainen ärsyke voisi olla vaikkapa pommin räjähtäminen kuuloetäisyydellä tai havainto siitä, että Matti ja Maija suutelevat. Viesti sisältää tarvittavan informaation, ja tiedon siitä, miten tapahtumaan pitäisi reagoida. Osaan tapahtumista agentti reagoi vain päivittämällä tietämystään maailmasta, osasta seuraa näkyvää toimintaa. Pommin räjähtäminen luultavasti tuottaisi näkyvän reaktion, jossa agentti juoksee päinvastaiseen suuntaan pommista, kun taas suudelman seurauksena agentti luultavasti vain päättelisi, että Matti ja Maija pitävät toisistaan, ja tekisi asiasta merkinnän tietämyskantaansa. Kaikista tapahtumista agentti ei välttämättä ole kiinnostunut lainkaan. Kahden tuntemattoman agentin suudelma tuskin tuottaisi minkäänlaista reaktiota.

Agenttien viestijonot ovat prioriteettijonoja, joista korkean prioriteetin viestit käsitellään ennen vähemmän tärkeitä viestejä. Jos viestijono pääsee täyttymään, unohtaa agentti vähiten merkitykselliset viestit. Tämä ehkäisee sellaisen epäuskottavan tilanteen syntymistä, jossa agentit vain kaikessa rauhassa juttelevat keskenään, vaikka vierelle putoaisi pommi. Toisaalta runsaskaan informaatiotarjonta ei saa agenttia jättämään merkittäviä asioita huomiotta. Käytännössä viestit välitetään käyttämällä Artifistin *perseption*-moduulia, jonka avulla tapahtuma voi laukaista viestin lähetyksen kaikille halutulla etäisyydellä oleville agenteille. Viestin lähetyksen yhteydessä sille määritellään sopiva prioriteetti.

4.3 Pelaajan vuorovaikutus

Jotta voidaan puhua pelistä, täytyy pelaajalla luonnollisestikin olla mahdollisuus vaikuttaa ainakin yhden agentin toimintaan. Tämä on käytännössä hyvin haastava osa interaktiivista tarinankerrontaa. Puhutun kielen ymmärtäminen ei ole sillä tasolla, että puhekäyttöliittymää, joka ehkä olisi ihmiselle luonnollisin, voisi yleisessä tapauksessa käyttää. Vapaan tekstinsyötön kanssa on saman tyyppisiä ongelmia: tekstin tulkinta kuluttaa runsaasti aikaa, eikä tekstistä siitä huolimatta välttämättä löydetä juuri oikeaa merkitystä. Tästä syystä Artifist käyttää pelaajavuorovaikutukseen monivalintoja. Vapaan tekstinsyötön lisääminen järjestelmään olisi kuitenkin mahdollista suorittaa kohtuullisella vaivalla jokseenkin samaan tapaan kuin Façadessa [Mateas & Stern, 2004], mikäli riittävän hyvä tekstin tulkintamoduuli olisi käytettävissä.

Reaktiivisten toimintojen kohdalla pelaajalle tarjottavien vaihtoehtojen etsiminen on yksinkertaista. Riittää kun pelaajalle tarjotaan kaikkia sellaisia edellisen toiminnon naapureita, jotka vallitsevassa tilanteessa ovat mahdollisia. Aloitteellisten toimintovaihtoehtojen keräämiseen Artifistissa on tällä hetkellä toteutettuna kaksi vaihtoehtoista menetelmää.

Ensimmäinen vaihtoehto on kaikkien mahdollisten toimintojen kerääminen. Siinä missä tavallinen agentin askellusalgoritmi valitsee probabilistisesti yhden haaran päätöspuussa, kaikki vaihtoehdot keräävä algoritmi laajentaa etsintää kaikkiin sellaisiin haaroihin, joiden todennäköisyys on nollasta poikkeava. Jokainen näin löydetty toiminto lisätään pelaajalle tarjottavien toimintojen joukkoon. On selvää, että tällainen keräysoperaatio on huomattavasti hitaampi kuin agentin etsintäalgoritmi. Jos päätöspuu olisi täysin tasapainoinen, ei-pelaaja-agentin toiminnon etsimiseen kuluisi logaritminen määrä askelia koko puun solmujen määrään nähden. Pelaaja-agentin kaikkien vaihtoehtojen keräämiseen kuluisi taas pahimmillaan lineaarinen määrä askelia, mikäli kaikki vaihtoehdot olisivat mahdollisia. Tästä syystä pelaaja-agentille on annettava huomattavasti korkeampi prioriteetti kuin muille agenteille.

Selvästikään edellä kuvattu algoritmi ei ole käyttökelpoinen kaikissa tapauksissa. Jos mahdollisia vaihtoehtoja on hyvin paljon, algoritmi on hidas ja, mikä pahinta, vaihtoehtojen esittäminen käyttäjälle järkevällä tavalla ei ole mahdollista. Tilanteisiin, joissa päätöspuut ovat hyvin suuria, tarvitaan siis toisenlainen algoritmi. Siinä pelaaja-agentti suorittaa k kertaa ei-pelaaja-agentin etsintäalgoritmin, kerää siten saadut vaihtoehdot ja tarjoaa niitä pelaajalle poistettuaan ensin mahdolliset duplikaatit. Suurin osa jäljelle jääneistä vaihtoehdoista vaikuttaa todennäköisesti hyviltä ja ilmeisiltä, mutta joukkoon saattaa silloin tällöin eksyä myös epätodennäköisiä vaihtoehtoja. Askelia algoritmiin kuluu enää vakio k kertaa enemmän kuin ei-pelaaja-agentin algoritmiin. Kaikissa toimintojen etsintätapauksissa on mahdollista, että löydetään vain yksi vaihtoehto. Tällöin ainoa mahdollinen vaihtoehto suoritetaan käyttäjältä mitään kyselemättä.

Toisinaan vaihtoehtojen joukko pelisovelluksessa on niin suuri, että sen esittäminen käyttäjälle on mahdotonta. Tällöin vaihtoehdot voidaan esittää useampiportaisina. Ensimmäisellä tasolla tarjotaan vaihtoehtoja kuten “Pidätkö sinä X:stä?” tai “Pitääkö X Y:stä”. Jos pelaaja valitsee jommankumman vaihtoehdoista, tarjotaan seuraavalla tasolla samaa kysymystä siten, että X on sidottu kaikkiin mahdollisiin arvoihinsa. Jos valittiin jälkimmäinen kysymys, tarjotaan kolmannella tasolla kaikkia mahdollisia Y:n arvoja valittaviksi. Käyttöliittymältään sofistikoituneemmassa sovelluksessa moniportainen valinta voidaan esittää moniportaisena valikkorakenteena.

5 Sekundäärinen päättelyjärjestelmä

Artifist-järjestelmän sekundäärinen päättelyjärjestelmä muodostuu agenttien omistamista informaatioalkioista, joiden avulla voidaan määritellä agenttikohtaista tietoa ja toimintoja. Informaatioalkiot sisältävät tietoa, tavan tiedon esittämiseen ja funktion, jolla voidaan analysoida tiedon ilmaisemisen utiliteetti vallitsevassa tilanteessa. Informaatioalkiot voivat olla myös kahvoja toimintoihin. Näin voidaan määritellä agenttikohtaisia toimintoja, jotka vain osa agenteista voi suorittaa, ja mikä tärkeintä, agenteille voidaan määritellä suoritettavia tehtäväsarjoja, joita ei kuitenkaan ole pakko suorittaa peräkkäin, vaan ainoastaan keskenään tietyssä järjestyksessä. Artifistin sekundäärinen päättelyjärjestelmä siis mahdollistaa suunnitelmien laatimisen ja suorittamisen.

Sekundäärinen päättelyjärjestelmä on upotettu primäärisen päättelyjärjestelmän sisään siten, että päätöspuissa ja toimintagraafissa olevat toiminnot voivat sisältää informaatioalkioiden käsittelyä: agentti voi *ChoiceAction*-toiminnossa, käydä läpi tietyntyyppisiä informaatioalkioita ja suorittaa niistä jonkun. Valinta tehdään samaan tapaan kuin toimintojenkin kohdalla. Valitsevassa tilanteessa mahdollisten informaatioalkioiden joukosta valitaan utiliteettien perusteella probabilistisesti yksi. Jos kysymyksessä on pelaajan ohjaama agentti, valinnan mahdollisten informaatioalkioiden välillä tekee pelaaja.

5.1 Informaatioalkiot

Informaatioalkiot voidaan jakaa kategorioihin kahden eri ominaisuuden - rakenteen ja semantiikan - mukaan taulukossa 5.1 esitetyllä tavalla. Informaatioalkiot voivat olla huomioita, kysymyksiä, vastauksia tai kahvoja toimintoihin. Artifistin valmiit informaatiorakenteet ovat *StringInfo*, *FunctionInfo* ja *HandleInfo*. Ne eroavat toisistaan esitystavan ja analysointimenetelmän suhteen. *StringInfo* sisältää merkkijonon, joka toimii informaation esitystapana sekä kerrottaessa informaatiota että tarjottaessa sitä pelaajalle valintatilanteessa. Merkkijonon luominen on pelin ohjelmoijan vastuulla. *FunctionInfossa* sekä kuvaus- että toimintaosa annetaan funktio-osoittimina, jotka voivat osoittaa samaan tai eri funktioihin. Funktioiden kirjoittaminen on pelin ohjelmoijan vastuulla. *HandleInfot* ovat nimensä mukaisesti kahvoja toimintoihin, ja niiden toimintaosana on toiminnon toimintaosa ja kuvauksena toiminnon kuvaus. *HandleInfoissa* myös analyysifunktio haetaan toiminnosta, jollei sitä ole erikseen informaatioalkiolle määritelty. Muiden informaatioalkioiden kohdalla palautetaan tässä tapauksessa oletusutiliteetti

0,5. Pelin ohjelmoijan on mahdollista luoda myös omia informaatiorakenteita ja -tyyppejä periyttämällä Artifistin *Info*-luokasta.

Agentti tallentaa informaatioalkiot assosiatiiviseen säiliöön, jonka käytännön toteutus on C++:n Standard Template Libraryn multimap-tietorakenne. Kullekin informaatioalkiolle on määritelty kokonaisluku, joka toimii hakuavaimena. Hakuavaimet eivät ole uniikkeja vaan samaan asiaan liittyvillä informaatioalkioilla on sama avain. Näin kysymykset ja vaihtoehtoiset vastaukset voidaan linkittää toisiinsa, ja esimerkiksi informaatioiden utiliteettiosassa voidaan yksinkertaisesti tutkia, onko jokin tietyllä hakuavaimella varustetuista informaatioista lausuttu tai suoritettu.

Informaatioalkioilla voi olla yksi tai useampi parametri. Parametrit ovat käyttökelpoisia kun halutaan tehdä useita informaatioalkioita, jotka käsittelevät samaa asiaa, mutta esimerkiksi eri agentteihin liittyen. Tällainen asia voisi olla vaikkapa agentin vasenkätisyys: Pelissä voi olla parametrisoitu kysymys “Onko X vasen- vai oikeakätinen?”. Ennen kuin kysymys esitetään, parametri X sidotaan johonkin sen laillisista arvoista. Vastaukseksi kysymykseen kelpaa vain oikeantyyppinen vastaus, jonka parametri on sama kuin kysymyksessä.

5.2 Kysymysten esittäminen

Artifistista löytyy kysymysten esittämiseen geneerinen toiminto, jonka käyttämä algoritmi on esitetty kuvassa 5.1. Toiminto on tyyppiä *BasicAction*, ja sitä seuraa aina geneerinen vastausfunktio. Kysymysinformaatio voi olla myös tyyppiä *HandleInfo*, mutta tuolloin täytyy muistaa, että sen osoittamaa kommunikaatioketjua ei saa lopettaa *EndAction*-solmuun. Kommunikaatioketjun tulee loppua vasta vastaukseen.

	StringInfo	*FunctionInfo*	*HandleInfo*
NOTION	STRING_NOTION	FUNCTION_NOTION	HANDLE_NOTION
QUESTION	STRING_QUESTION	FUNCTION_QUESTION	HANDLE_QUESTION
ANSWER	STRING_ANSWER	FUNCTION_ANSWER	HANDLE_ANSWER
ACTION	-	-	HANDLE_ACTION

Taulukko 5.1: Informaatiotyypit

Jos oletetaan, että analyysifunktion suorittaminen on vakioaikainen operaatio ja kysymykset eivät ole parametrisoituja, on kysymyksien esittäminen ajankäytöltään lineaarista agentin informaatiosäiliön kokoon nähden. Mikäli kaikki kysymykset ovat parametrisoituja, on suoritusaika pahimmillaan $\Theta(nm)$, jossa n on agentin informaatiosäiliön koko ja m on kaikkien valittavissa olevien kysymysten määrä.

5.3 Kysymyksiin vastaaminen

Kysymyksiin vastaamiseen on olemassa geneerinen toiminto, joka käyttää kuvassa 5.2 esitettyä algoritmia. Vastaustoiminto on tyypiltään *SimpleAction*, eli se ei jatka kommunikaatioketjua, muttei myöskään automaattisesti lopeta sitä. Tämä mahdollistaa sen, että vastaus voi olla kahva kokonaiseen kommunikaatioketjuun, joka aikanaan päättyy johonkin ketjun lopetussolmuun. Jos vastaus ei ole kahva, vastaustoiminto lopettaa kommunikaatioketjun.

1. Hae kaikki QUESTION-tyyppiset informaatioalkiot agentin informaatiosäiliöstä.
2. Laajenna informaatioalkiot, joilla on sitomattomia parametreja sitomalla parametrit mahdollisiin arvoihinsa.
3. Karsi vallitsevassa tilanteessa mahdottomat ja kohteen jo tuntemat informaatioalkiot.
4. Tee probabilistinen valinta jäljelle jääneiden alkioiden kesken.
5. Jos valittu parametrisoitu informaatio on vasta laajennettu, tallenna laajennettu versio agentin informaatiosäiliöön.
6. Merkitse informaatio kuulijoiden tuntemaksi.
7. Lausu tai suorita informaatio.

Kuva 5.1: Kysymyksen esittäminen

Jos oletetaan, että analyysifunktion suorittaminen on vakioaikaista, mahdollisten vastausvaihtoehtojen hakemisen ja läpikäymisen aikavaatimus on $\Theta(\log n + m)$, jossa n on agentin informaatiosäiliön koko ja m on kaikkien valittavissa olevien vastausvaihtoehtojen määrä. Vastaaminen on siis selvästi kysymistä nopeampi toimenpide, mikä on hyvä, sillä hiljaiset hetket keskustelussa ovat vähemmän häiritseviä kommunikaatioketjujen välillä kuin niiden keskellä. Aitoonkin keskusteluun liittyy silloin tällöin selviä taukoja, mutta harvoin kysymys-vastaus -parien väliin.

5.4 Huomioiden esittäminen

Huomioiden tai käyttäjän määrittelemää tyyppiä olevien informaatioiden esittäminen tapahtuu vastaavalla tavalla kuin kysymysten, kuitenkin sillä erolla, että kommunikaatioketjun lopettaminen tapahtuu vastaavalla tavalla kuin vastausinformaatioiden kohdalla. Myös suoritusaika on sama kuin kysymysten esittämisessä.

1. Hae agentin informaatiosäiliöstä kaikki ANSWER-tyyppiset informaatioalkiot, joiden avain on sama kuin edellisellä kysymyksellä.
2. Karsi parametreiltaan väärät ja kohteen jo tuntemat informaatioalkiot.
3. Tee probabistinen valinta jäljelle jääneiden alkioiden kesken.
4. Merkitse informaatio kuulijoiden tuntemaksi.
5. Lausu tai suorita informaatio, tai jos vastausta ei löytynyt, suorita oletustoiminto.
6. Jos valittu toiminto ei ole *HandleInfo* tai toimintoa ei löytynyt, lopeta kommunikaatioketju.

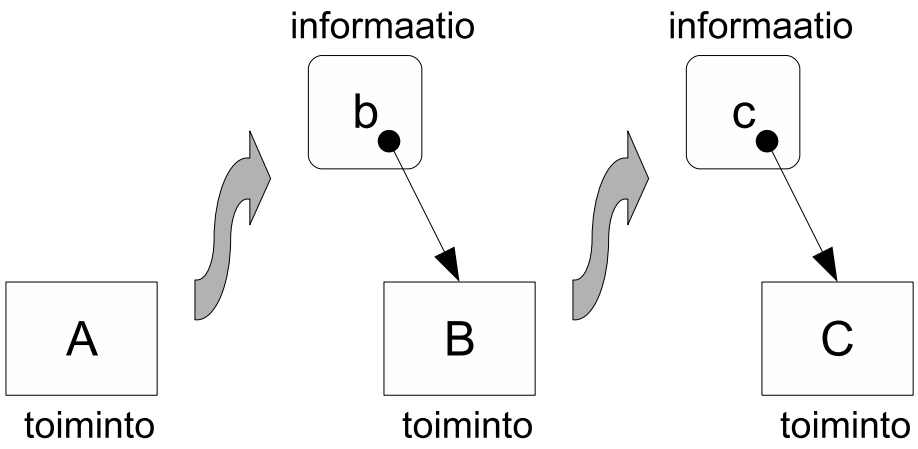
Kuva 5.2: Kysymykseen vastaaminen

5.5 Toimintosarjojen suorittaminen

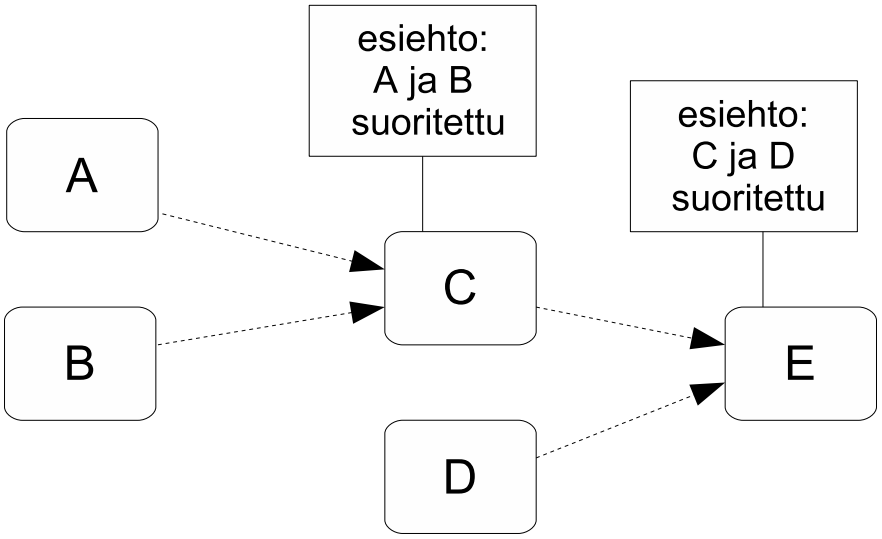
Koska informaatio voi toimia kahvana toimintoon, ja toiminto voi tuottaa uusia informaatioita, voidaan informaatioiden ja toimintojen avulla rakentaa toimintosarjoja, jotka täytyy suorittaa tietyssä järjestyksessä. Kuvassa 5.3 on esimerkki eräästä toimintosarjasta $A \rightarrow B \rightarrow C$. Siinä toiminnon A suorittaminen aiheuttaa informaation b lisäämisen agentin informaatiosäiliöön. Informaatio b on kahva toimintoon B, joka suoritetaan agentille sopivalla ajanhetkellä. Sopiva ajanhetki voidaan kuvailla toiminnon B utiliteettifunktiossa. Vastaavasti B:n suorittaminen johtaa c:n lisäämiseen agentin informaatioiden joukkoon ja lopulta myös toiminnon C suorittamiseen. Toimintojen A, B ja C välissä voidaan suorittaa mielivaltainen määrä muita tilanteeseen sopivia toimintoja. Haluttaessa käytetyt informaatiot voidaan poistaa agentin informaatiosäiliöstä algoritmin tehostamiseksi.

Toisaalta informaatioiden avulla voidaan tehdä osittaisjärjestyksessä olevia suunnitelmia, joita agentti soveltaa tilanteen mukaan. Kuvassa 5.4 on esitetty viisi informaatiokahvaa, joiden välillä vallitsee osittaisjärjestys. Informaation C utiliteetti on määritelty nollaksi, mikäli sekä informaatioalkion A että informaatioalkion B osoittamia toimintoja ei ole suoritettu. Samoin informaatioalkion E osoittama toiminto voidaan suorittaa vain, jos B ja C on suoritettu.

Informaatiokahvojen valinta on ajankäytöltään vastaavaa kuin kysymysten valinta, joskin parametrisoidut informaatiokahvat lienevät melko harvinaisia. Läheskään kaikkia pelissä esiintyviä toimintoja ei ole tarkoitus esittää informaatio-



Kuva 5.3: Esimerkki toimintosarjasta $A \rightarrow B \rightarrow C$



Kuva 5.4: Esimerkki osittaisjärjestyksessä suoritettavista toiminnoista

kahvojen avulla, vaan toistuvat, kaikille agenteille avoimet toiminnot on tarkoitus aina mallintaa primäärisen päättelyjärjestelmän avulla. Ainoastaan vahvasti agenttikohtaisia ja suunnitelmallisuutta vaativia toimintoja varten tarvitaan informaatioalkioita. Näin toimittaessa agenttien informaatiosäiliöiden kokojen pitäisi pysyä kohtuullisen pienenä, eikä informaatioiden valintaan kuluvan ajan pitäisi kasvaa niin suureksi, että se aiheuttaisi näkyviä viiveitä suoritusaikaan.

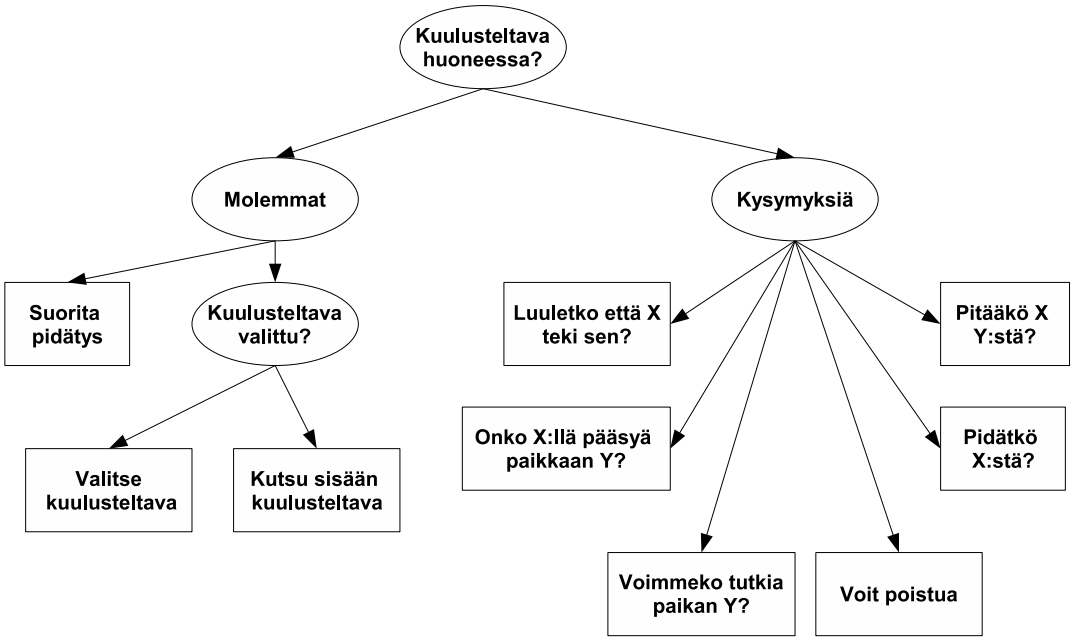
6 Käyttöesimerkkejä

Artifistin avulla on toteutettu muutamia esimerkkisovelluksia. Osa niistä on yksinkertaisia simulaatioita, joissa agentit liikkuvat ympäriinsä ja käyvät keskenään lyhyitä keskusteluja. Kaksi sovelluksista on kuitenkin hieman pelillisempiä ja sikäli mielenkiintoisempia. Toinen niistä on Cluedo-tyyppinen peli, Murhapeli, ja toinen on ohjelma, joka generoi interaktiivisesti Arthur Conan Doylen novelliin perustuvaa näytelmäkäsikirjoitusta.

6.1 Murhapeli

Murhapelissä pelaaja ohjaa poliisia, jonka tehtävänä on ratkaista, kuka epäillyistä on murhaaja. Koko peli tapahtuu kuulusteluhuoneessa, johon pelaaja voi vuorotellen kutsua epäiltyjä. Epäillyt näytetään pelaajalle valokuvina. Kullakin epäillyllä on muutamia eri ilmeitä, jotka heijastelevat heidän mielentilaansa. Kuulustelun aikana pelaaja voi esittää epäillylle kysymyksiä kuten “Luuletko, että X teki sen?”, “Pidätkö sinä X:stä?” tai “Pitääkö X Y:stä?”. Epäilty vastaa kysymyksiin joko rehellisesti tai epärehellisesti riippuen siitä, miten edulliselta toden puhuminen hänen kannaltaan vaikuttaa, ja toisaalta siitä, kuinka rehellinen hän ylipäänsä on. Tavallisten kysymysten lisäksi pelaaja voi pyytää epäillyltä lupaa tutkia tämän asuintilat. Mikäli epäilty myöntyy, etsintä suoritetaan ja sen tuloksesta ilmoitetaan pelaajalle. Lopuksi pelaaja voi pidättää yhden epäillyistä. Jos pidätetty tosiaan on murhaaja, pelaaja on voittanut pelin. Muussa tapauksessa todellinen murhaaja paljastetaan, ja pelaajalle ilmoitetaan, että hän hävisi pelin.

Yksinkertaisuuden vuoksi pelin tietokoneen ohjaamat agentit ovat kaikki puhtaasti reaktiivisia, eikä niillä ole päätöspuita. Pelimaailmassa siis tapahtuu jotakin ainoastaan silloin, kun pelaaja kuulustelee epäiltyjä. Tällöinkään epäillyt eivät esitä huomioita oma-aloitteisesti. Pelaajan ohjaamalla poliisiagentilla sen sijaan on kuvassa 6.1 esitetty päätöspuu, josta pelaajalle tarjottavat vaihtoehdot etsitään. Vaihtoehtojen etsintään käytetään algoritmia, joka hakee kaikki vallitsevassa tilanteessa mahdolliset toiminnot. Toisia agentteja koskevat kysymykset on toteutettu moniportaisina kohdassa 4.3 esitetyllä tavalla.



Kuva 6.1: Pelaaja-agentin päätöspuu Murhapelissä

6.2 Sherlock Holmes -esimerkki

Sherlock Holmes -esimerkkiohjelma generoi interaktiivisesti näytelmäkäsikirjoitusta, joka perustuu Arthur Conan Doylen novelliin The Boscombe Valley Mystery [Doyle, 1859-1930]. Pelaaja ohjaa luonnollisestikin etsivä Sherlock Holmesin valintoja. Muut agentit, avustavat hahmot tohtori Watson ja tarkastaja Lestrade, sekä kahdeksan epäiltyä, ovat tietokoneen ohjauksessa. Sovellukseen ei liity lainkaan grafiikkaa, vaan se on puhtaasti tekstipohjainen. Vaihtoehdot esitetään listana, josta käyttäjä valitsee haluamansa näppäilemällä vaihtoehdon järjestysnumeron. Itsestäänselvät toiminnot suoritetaan automaattisesti myös Holmesin kohdalla. Ohjelman suorituksen aikana käyttäjä voi haluamassaan järjestyksessä käydä rikospaikalla, haastatella epäiltyjä ja esittää epäilyksensä heille. Kun Holmes osoittaa syylliselle tietävänsä, mitä tapahtui, syyllinen tunnustaa ja tarina loppuu. Kuvassa 6.2 on esitetty katkelma ohjelman generoimaa dialogia.

Esimerkin keskeisenä tavoitteena on testata informaatioalkioiden käyttöä ja osoittaa, että Artifistilla voi tehdä myös juonellisia, melko vahvasti käsikirjoitettuja mutta epälineaarisia tarinoita. Tarinan epälineaarisuuden voi todeta kuvasta 6.3, jossa on esitetty neljän esimerkkiajon kulku. Rasteilla merkityssä ajossa tapahtumat on suoritettu novellin mukaisessa järjestyksessä, muissa tapauksis-

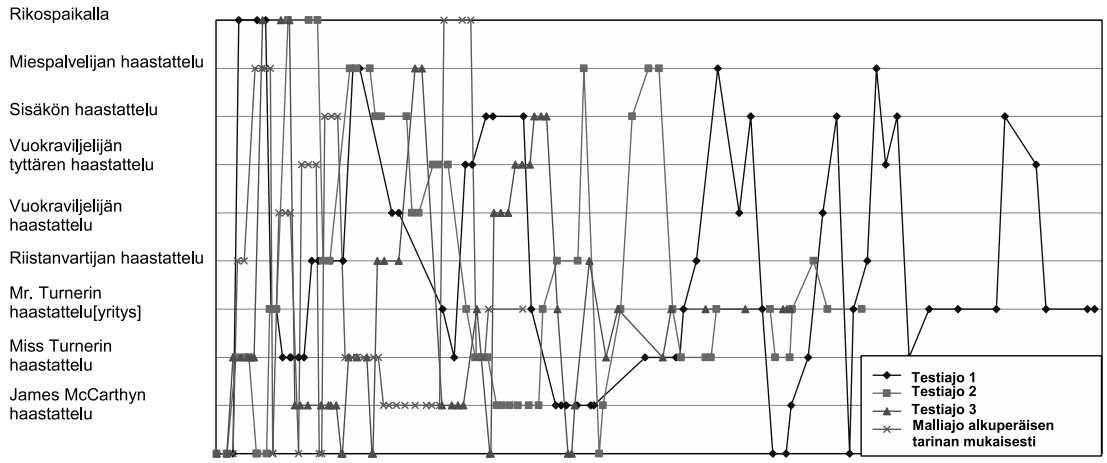
Holmes: Did your father say anything, before he passed away?
James McCarthy: I heard him mumble a few words, but I could only catch some allusion to a rat.
Holmes: A rat! That is peculiar. What do you understand about it?
James McCarthy: It conveyed no meaning to me. I thought that he was delirious.
Dr. Watson: It could not be delirium. A man dying from a sudden blow does not commonly become delirious.
Holmes: Do you know anyone who limps his foot?
James McCarthy: Yes, Mr. Turner, our neighbour, limps his foot.
Holmes: What kind of tobacco does Mr. Turner smoke?
James McCarthy: He smokes Indian cigars.
Holmes: What kind of tobacco do you smoke?
James McCarthy: I smoke Caribbean cigars.

Kuva 6.2: Katkelma Sherlock Holmes -esimerkkiohjelman generoimaa dialogia

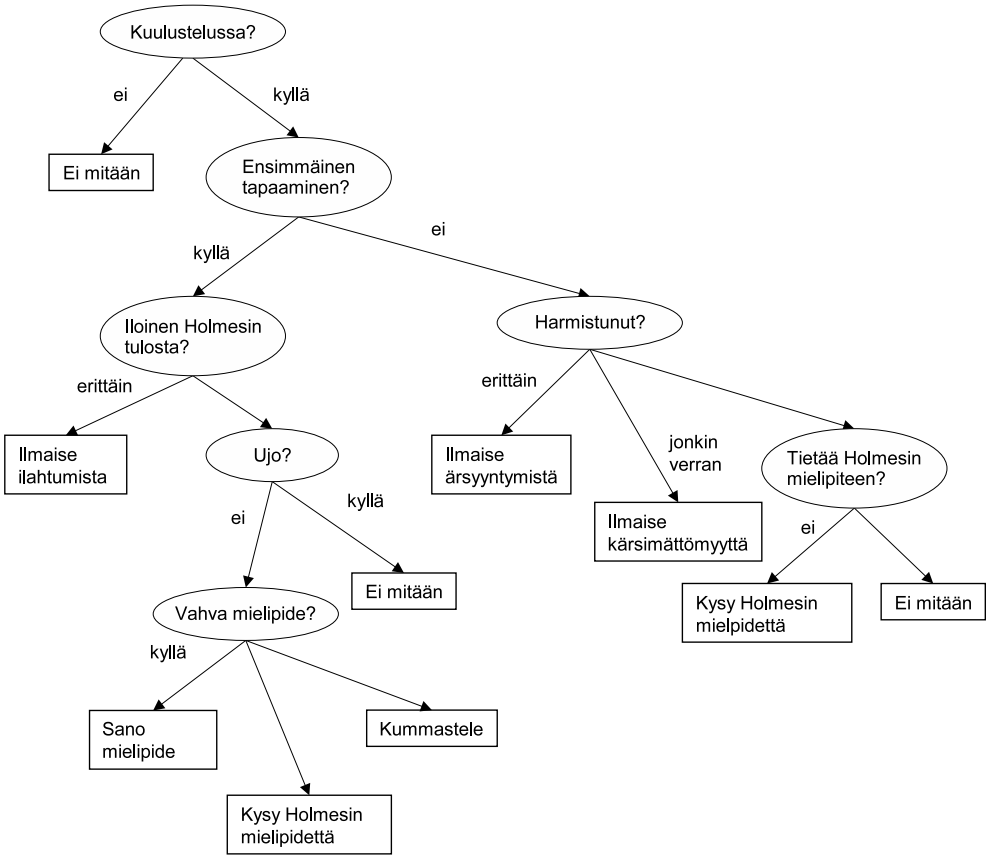
sa käyttäjä on ollut vapaaehtoinen testihenkilö, joka ei ole etukäteen tuntenut novellia. Esimerkissä on semantiikaltaan kolmen tyyppisiä informaatioita: huomioita, kysymyksiä ja vastauksia. Jotkut niistä ovat vain yksittäisiä lausuttavia tietoja, kun taas osa on kahvoja pidempiin toimintosarjoihin. Osa informaatioalkioista on agenteilla jo simulaation alkaessa. Osa on valittavissa välittömästi, kun taas osa edellyttää jonkin toisen informaation kertomista ensin. Lestrade tervehtii Holmesia ja Watsonia, ennen kuin käy varsinaiseen asiaan.

Toisaalta osa informaatioista annetaan agenteille vasta tarinan edetessä. Kun esimerkiksi käy ilmi, että murhaaja on vasenkätinen, Holmesille tallennetaan tieto siitä, sekä kysymys “Onko X vasenkätinen?”. X:n paikalle voidaan sijoittaa mikä tahansa tunnetuista epäillyistä. Luonnollisestikaan ei ole mielekästä kysyä, onko neiti Turner vasenkätinen, ennen kuin on kuultu, että on olemassa neiti Turner, jolla on jotakin tekemistä murhan uhrin kanssa.

Tässäkään esimerkissä tietokoneen ohjaamat hahmot eivät ole kovin itsenäisiä. Ne toimivat vain ollessaan interaktiossa Holmesin kanssa, mutta eivät kuitenkaan ole täysin reaktiivisia, vaan voivat esittää erilaisia huomioita ja lausahduksia myös oma-aloitteisesti. Tämä edellyttää sitä, että niillä on päätöspuut. Sherlock Holmes -esimerkissä on kolme erilaista päätöspuuta, Holmesin päätöspuu, tukihahmojen yhteinen päätöspuu ja epäiltyjen yhteinen päätöspuu, joka on esitetty kuvassa 6.4.



Kuva 6.3: Neljän esimerkkitarinan kulku Sherlock Holmes -esimerkissä



Kuva 6.4: Epäillyn päätöspuu Sherlock Holmes -esimerkissä

7 Arviointi ja yhteenveto

Tässä luvussa arvioidaan Artifist-järjestelmän ominaisuuksia ja sen soveltuvuutta tietokonepelien tekoälyohjelmointiin. Lisäksi käsitellään mahdollisia jatkokehitys-suunnitelmia sekä esitetään lyhyt yhteenveto.

7.1 Arviointi

Vaikkeivät edellisessä luvussa esitetyt esimerkkisovellukset ole peleinä kovin värikkäitä, on helppo kuvitella, että niiden ominaisuuksia yhdistelemällä voitaisiin tehdä monimutkaisempia ja kiinnostavampia sovelluksia. Poimimalla testisimulaatioista agenttien itsenäisen tavan liikkua, Murhapelistä agenttien muuttuvat mielentilat ja Sherlock Holmes -esimerkistä juonelliset elementit ja lisäämällä näihin vielä nykyvaatimusten mukaisen 3D-grafiikan, voisi toteuttaa varsin mielenkiintoisen salapoliisipelin. Epäillyt voisivat vaihtaa tietoa keskenään, osallistua keskusteluihin ja kuunnella niitä, ehkäpä jopa suorittaa uusia rikoksia salatakseen tai kostaakseen ensimmäisen. Toisaalta Murhapelin kaltainen kuulustelukohtaus saattaisi maustaa monia perinteisiä juoneltaan lineaarisia pelejä. Artifistia suunniteltaessa on muutoinkin pidetty mielessä soveltuvuus moniin suosittuihin pelityyppeihin, kuten rooli- ja seikkailupeleihin. Perinteisten pelityyppien lisäksi tavoitteena on kokeilla Artifistia uudentyyppisten, sosiaaliseen interaktioon ja tavoitteisiin perustuvien pelien toteuttamiseen.

Artifist on osoittautunut toteutettaessa esimerkkisovelluksia melko helppokäyttöiseksi, joskin sen käyttömukavuus paranisi huomattavasti, jos puiden ja graafien rakentamiseen olisi olemassa graafinen käyttöliittymä. Toistaiseksi puut ja graafit on piirretty paperille ja sitten toteutettu käsin ohjelmoimalla osoittimien avulla, mikä on hieman työlästä ja valitettavan virhealtista. Kuvien käyttö on kuitenkin helpottanut keskustelua sovelluksista huomattavasti, eikä väärinkäsityksiä niiden toiminnan suhteen ole juurikaan syntynyt. Luultavasti jopa henkilö, jolla ei ole lainkaan ohjelmointiosaamista voisi osallistua Artifistin avulla toteutettavan pelitekoälyn suunnitteluun piirtämällä ja arvioimalla päätöspuita ja graafeja.

Tämä on tärkeä tavoite, koska uskoakseni ainoa tapa saada aikaan laajemmassa mittakaavassa mielenkiintoisia tietokonepelejä, joissa juoni on epälineaarinen ja pelaajan toimista riippuva, on tuoda pelisuunnittelijat mukaan myös tekoälyn kehitykseen. Pelisuunnittelijoiden ohjelmointiosaaminen ei tyypillisesti ole kovin

vahva, joten käytettävän menetelmän on oltava mahdollisimman intuitiivinen ja läpinäkyvä.

Läpinäkyvyydestä on myös muuta hyötyä. Koska tietokonepelihahmojen keskeisenä vaatimuksena on uskottavuus, ei satunnainen järjetön käyttäytyminen ole hyväksyttävää. Läpinäkyvä järjestelmä on helpompi määritellä siten, että järjetöntä käyttäytymistä ei synny. Toisaalta jos testausvaiheessa havaitaan virheellisiä toimintoja, niihin on helpompaa puuttua, koska ongelmakohtaa käsittelevä ohjelmakoodi on helppo paikallistaa.

Moniagenttijärjestelmä soveltuu monenlaisiin peleihin varsin hyvin, koska se mallintaa intuitiivisesti maailmaa, jossa hahmot toimivat itsenäisesti tai yhteistyössä keskenään. Kullakin agentilla on oma näkemyksensä maailmasta, ja tätä näkemystä päivitetään agentin havaintojen perusteella. Agentti valitsee toimintonsa perustuen näkemykseensä maailmasta. Kaikki tämä on varsin hyvin linjassa sen kanssa, mitä todellinen maailma on ja mitä useimmissa pelimaailmoissa halutaan. Valitettavasti järjestelmiin, joissa useat toisistaan riippumattomat agentit suorittavat toimintoja, liittyy myös ikävä ongelma, nimittäin arvaamattomuus. Kun tarina muodostuu useiden agenttien toimista, siihen syntyy helposti odottamattomia kummallisuuksia. Tämä oli ilmeistä jo tässä työssä esitellyissä pienehköissä esimerkkiohjelmissa. Saman havainnon ovat tehneet muutkin tahot. Mm. suositun Oblivion-pelin [Oblivion, 2006] tekijät ovat raportoineet samansuuntaisista ongelmista.

Artifistin päättelyjärjestelmän punainen lanka on probabilistisuus. Se tuo mukanaan hieman samantapaisia arvaamattomuuteen liittyviä ongelmia kuin autonomiset agentit. Sen ongelmat ovat kuitenkin helpommin hallittavia ja yleensä ratkaistavissa satunnaiseen valintaan liittyvien vakioiden säätämisellä. Satunnaisuuden edut ovat huomattavasti haittoja suuremmat. Hahmoista voidaan tehdä uskottavampia, kun ne eivät käyttäydy täysin deterministisesti, vaan valitsevat useimmiten loogisimman toimintatavan, mutta voivat silloin tällöin tehdä yllättäviäkin valintoja - aivan kuten oikeat ihmiset. Pelin kulku muodostuu vaihtelevammaksi ja yllätyksellisemmäksi ja sitä kautta kiinnostavammaksi. Satunnaisuus luo mahdollisuuksia uudelleen pelattavien pelien tekemiselle. Jos pelin alkutilanne on mahdollista generoida jollakin tapaa satunnaisesti ja tarina muodostuu alkutilanteen ja satunnaisten valintojen pohjalta, on perusteltua uskoa, että peli ei menetä kiinnostavuuttaan ensimmäisen pelikerran jälkeen, kuten lineaaristen, valmiiksi käsikirjoitettujen pelien kohdalla usein tapahtuu. Toisaalta satunnaisuus tuo peliin lisää haastetta, kun varmoja dominoivia strategioita ei pääse syntymään.

Varsinaisia tehokkuusmittauksia Artifistista ei vielä ole tehty, koska esimerkkisovellukset eivät ole olleet riittävän suuria sellaisia varten. Voidaan kuitenkin todeta, että edes Sherlock Holmes -esimerkissä, joka on esimerkeistä monimutkaisin, ei synny silminhavaittavia viiveitä. Käytännössä reaktiivisten toimintojen kohdalla sellaisia ei koskaan pitäisikään syntyä, ellei mikään *ChoiceAction*-solmu generoi kohtuuttoman suurta määrää vaihtoehtoja. Ympäristön ärsykkeisiin reagoiminen on aina vakioaikaista, mikäli suoritettava toiminto itsessään on vakioaikainen. Pisimmät viiveet syntynevät aloitteellisessa toiminnossa valittaessa informaatioalkiota. Tällöin joudutaan käymään läpi koko agentin informaatiosäiliö sen lisäksi, että kuljetaan päätöspuu juuresta lehtisolmuun. Aloitteellisia toimintoja edeltävä viive ei kuitenkaan ole yhtä häiritsevä kuin reagointia edeltävä viive.

7.2 Jatkokehitys

Seuraavaksi Artifistiin on tarkoitus toteuttaa graafinen käyttöliittymä ainakin päätöspuiden ja graafien laatimista varten. Se on luonteva askel toisaalta siksi, että yhtenä tutkimuksen painopisteistä on tekoälyohjelmoinnin helpottaminen, johon käytettävyys liittyy olennaisesti. Toisaalta se on tarpeellinen myös muun jatkokehityksen kannalta, koska se mahdollistaa monipuolisemmin kaikkia Artifistin ominaisuuksia testaavien pelien toteuttamisen.

Toinen ilmeinen jatkokehitysalue on Artifistin muokkaaminen monen pelaajan peleille sopivaksi. Puhuttaessa vain muutamista pelaajista tehtävän pitäisi olla melko yksinkertainen, lähinnä pelaajan vuorovaikutusmenetelmiin liittyvä, mutta kasvatettaessa pelaajien määrää sellaiseksi, että Artifistia voitaisiin käyttää esimerkiksi internetin suurissa monen pelaajan peleissä *(Massive Multiplayer Online Game)*, kohdataan uusia haasteita. Nykyisin käytössä oleva vuorontaja on hyvin yksinkertainen, ja kuorman kasvaessa sitä täytyy optimoida. Vuorontaja suorittaa agenttien algoritmeja rinnakkain askel kerrallaan. Valinta on tehty pitkälti tehokkuussyistä, jotta algoritmien välillä ei tarvitsisi suorittaa poissulkemista kovin laajalti. On kuitenkin selvää, että jos pelaajia on satoja tai tuhansia, kaikkien agenttien algoritmeja ei mitenkään voida suorittaa vuorotellen samalla prosessorilla, vaan kuorma täytyy pystyä jakamaan useille eri prosessoreille. Tämän tekeminen tehokkaasti ja joustavasti on algoritmisesti haastavaa.

Myös Artifistin sovittaminen muissa kuin PC- tai konsoliympäristössä pelattaviin peleihin on ollut esillä. Lähinnä kiinnostavia ympäristöjä olisivat WWW-selain ja erilaiset mobiililaitteet. Artifist-agenttien testaaminen joissakin internetin virtuaalimaailmoista olisi myös kiinnostava hanke. Siinä kaikkia muita verkossa liikkuvia hahmoja mallinnettaisiin pelaajan ohjaamina hahmoina. Tätä varten Artifistiin täytyisi lisätä luonnollista kieltä tulkitseva moduuli.

7.3 Yhteenveto

Tässä työssä tutustuttiin useisiin olemassa oleviin interaktiivisiin tarinankerrontajärjestelmiin sekä erilaisiin lähestymistapoihin, joita automaattiseen tarinankerrontaan on. Lisäksi esiteltiin Tampereen teknillisen yliopiston Ohjelmistotekniikan laitoksella kehitetty Artifist-tarinankerrontajärjestelmä.

Toimivan ja monipuolisen interaktion luominen tietokonepeleihin on vaativaa ja aikaa vievää työtä ilman apuvälineitä. Useimmat nykyisistä tietokonepeleistä tarjoavatkin hyvin rajalliset mahdollisuudet kommunikaatioon pelaajan ja tietokoneen ohjaamien hahmojen kesken, monissa peleissä ei ole minkäänlaista verbaalista interaktiota. Juonellisemmissakin peleissä tarina on usein varsin lineaarinen ja sopeutuu huonosti käyttäjän tekemiin yllättäviin valintoihin. Tällaiset pelit yleensä menettävät kiinnostavuutensa viimeistään yhden pelikerran jälkeen.

Artifist on erityisesti suunniteltu näiden ongelmien helpottamiseen. Sen avulla verbaalista kommunikaatiota sisältävän juoneltaan epälineaarisen pelin tekeminen on selvästi helpompaa kuin ilman apuvälineitä. Graafisen käyttöliittymän valmistuminen tulee kasvattamaan tätä eroa. Päättelyjärjestelmän visuaalisuus ja intuitiivisuus mahdollistavat myös ohjelmointitaidottomien henkilöiden osallistumisen tekoälysuunnitteluun, mikä helpottaa tekoälyn integroimista kiinteämmin pelin juoneen.

Koska Artifistin tuottama tarina muodostuu itsenäisten agenttien tekemistä valinnoista ja agenttien valinnat ovat probabilistisia, on tarina erilainen joka pelikerralla. Tämä mahdollistaa uudelleen pelattavien pelien kirjoittamisen ja toisaalta vahvastikin käsikirjoitetussa pelissä pelaaja voi vapaasti valita, missä järjestyksessä suorittaa tehtäviä.

Artifist on moniagenttijärjestelmä, joka sisältää kaksi sisäkkäistä päättelyjärjestelmää. Primäärinen päättelyjärjestelmä perustuu probabilistisiin päätöspuihin ja graafeihin, joista agentit etsivät suoritettavia toimintoja. Sekundäärinen päättelyjärjestelmä perustuu informaatioalkioihin, jotka mallintavat agentin tietoja ja niiden esitystapaa. Yksittäisessä primäärisen päättelyjärjestelmän solmus-

sa agentti voi käydä läpi informaatiosäiliötään ja valikoida suoritettavan toiminnon sitä kautta. Primäärinen päättelyjärjestelmä on tarkoitettu agenttien tapojen, tarpeiden ja mielitekojen mallintamiseen, kun taas sekundäärisellä päättelyjärjestelmällä voidaan luoda suunnitelmallista käyttäytymistä ja epälineaarisia juonielementtejä.

Viiteluettelo

[Boost, 2006] Boost (2006). *Boost.* Accessed: 13.10.2006. `http://www.boost.org/`.

[Cass, 2002] Stephen Cass. Mind Games. *IEEE Spectrum December 2002.* 40-44.

[Cavazza & al., 2002a] Marc Cavazza & Fred Charles & Steven J. Mead. 2002. Interacting with virtual characters in interactive storytelling. In *Proceedings of Autonomous Agents and Multi Agent Systems.* Bologna, Italy. 318 - 325.

[Cavazza & al., 2002b] Marc Cavazza & Fred Charles & Steven J. Mead. 2002. Emergent situations in interactive storytelling. In *Proceedings of ACM Symposium on Applied Computing.* Madrid, Spain. 1080 - 1085.

[CEGUI, 2006] CEGUI (2006). *CEGUI.* Accessed: 13.10.2006. `http://www.cegui.org.uk/`.

[Crawford, 1999] Chris Crawford 1999. Assumptions underlying the Erasmatron interactive storytelling engine. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence.* AAAI Press, 1999.

[Dalmau, 2005] Daniel Sánches-Crespo Dalmau. *Core Techniques and Algorithms in Game Programming.* New Riders, Indianapolis, Indiana, 2003.

[Doyle, 1859-1930] Arthur Conan Doyle. *The Original Illustrated Sherlock Holmes.* Castle Books, New Jersey, 1859-1930.

[Dybsand, 2003a] Eric Dybsand. *AI Middleware: Getting into Character Part 1 - Biographic Technologies' AI.implant.* Gamasutra 18.7.2003.

[Dybsand, 2003b] Eric Dybsand. *AI Middleware: Getting into Character Part 2 - MASA's DirectIA.* Gamasutra 22.7.2003.

[Dybsand, 2003c] Eric Dybsand. *AI Middleware: Getting into Character Part 3 - Criterion's Renderware AI 1.0.* Gamasutra 23.7.2003.

[Dybsand, 2003d] Eric Dybsand. *AI Middleware: Getting into Character Part 4 - Stottler Henke's SimBionic.* Gamasutra 24.7.2003.

[Fairclough & Cunningham, 2004] Chris R. Fairclough & Pádraig Cunningham. 2004. AI structuralist storytelling in computer games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*. Reading, UK.

[Luger, 2005] George F. Luger. *Artificial Intelligence Structures and Strategies for Complex Problem Solving. Fifth Edition*. Pearson Education, Harlow, England, 2005.

[Mateas & Stern, 2002] M. Mateas & A. Stern. 2002. A behavior language for story-based believable agents. In *Proceedings of Artificial Intelligence and Interactive Entertainment, AAAI symposium*, 2002. 39-47.

[Mateas & Stern, 2004] M. Mateas & A. Stern. 2004. Natural Language Understanding in Façade: Surface Text Processing. In *Proceedings of Technologies for Interactive Digital Storytelling and Entertainment*. Darmstadt, Germany.

[Mateas & Stern, 2005] M. Mateas & A. Stern. 2005. Structuring Content in the Façade Interactive Drama Architecture. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment*. Los Angeles, USA.

[Millington, 2006] Ian Millington. *Artificial Intelligence for Games*. Morgan Kaufmann Publishers, San Francisco, California, USA, 2006.

[Oblivion, 2006] Oblivion (2006). *Oblivion, Wikipedia*. Accessed: 13.10.2006. `http://en.wikipedia.org/wiki/The_Elder_Scrolls_IV:_Oblivion`.

[OGRE, 2006] OGRE (2006). *OGRE*. Accessed: 13.10.2006. `http://www.ogre3d.org/`.

[Riedl, 2004] Mark Owen Riedl. *Narrative Generation: Balancing Plot and Character.* Ph. D. dissertation, North Carolina State University Raleigh, USA. 2004.

[Riedl & al., 2003] Mark Riedl & C. J. Saretto & R. Michael Young. 2003. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of Autonomous Agents and Multi Agent Systems*. Melbourne, Australia. 741-748.

[Russel & Norvig, 2003] Stuart Russell & Peter Norvig. *Artificial Intelligence, A Modern Approach.* Pearson Education, New Jersey, USA, 2003.

[Ruuska & Virtanen, 2006] Minna Ruuska & Antti Virtanen. 2006. ARTIFIST - Artificial Intelligence Framework for Interactive Storytelling In *8th International Conference on Computer Games: AI and Mobile Systems.* Louisville, Kentacky. 29-33.

[Sgouros, 2000] Nikitas M. Sgouros. 2000. Using Character Motives to Drive Plot Resolution in Interactive Stories. In *Applied Intelligence 12, 2000.* 239-249.

[Spark!, 2006] Spark! (2006). *Spark! Features.* Accessed: 4.10.2006. `http://www.louderthanabomb.com/spark_features.htm`.

[Storytron, 2006] Storytron (2006). *Storytron.* Accessed: 7.11.2006. `http://www.storytron.com/`.

[Winston, 1993] Patrick Henry Winston. *Artificial Intelligence.* Addison-Wesley, 1993.

[Wooldridge, 2000] Michael Wooldridge. *Reasoning about Rational Agents.* The MIT Press, Cambridge, Massachusetts, 2000.