Ristinolla

Tietorakenteiden harjoitustyö

Samuli Kaipiainen / 2004 syksy

15.10.2004
Helsingin yliopisto
Tietojenkäsittelytieteen laitos
Ohjaaja Janne Rinta-Mänty

Johdanto

Ristinolla on ulkoasultaan yksinkertainen mutta miellyttävä, tietokonepelaajalla varustettu peli, jossa on tarkoituksena saada isolla ruudukolla 5 omaa merkkiä (risti tai nolla) peräkkäin vaakaan, pystyyn tai vinottain.

Tietokonepelaaja on toteutettu minmax-algoritmilla, jota on tehostettu (nopeutettu) alpha-beta -karsinnalla ja turhien siirtojen karsinnalla; alpha-beta -karsinta ei vaikuta tietokoneen tekemiin siirtoihin, turhien (reuna)siirtojen karsinta saattaa joskus estää tietokonetta siirtämästä jotakin hyvää siirtoa.

Minmax-algoritmi valitsee vuorotellen tietokoneen siirtovaihtoehdoista (max) parhaan ja vastustajan vaihtoehdoista (min) tietokoneelle huonoimman (vastustajalle parhaan); se olettaa kummankin pelaajan pelaavan parhaalla mahdollisella tavalla.

Minmax laskee siirtoja annetun hakusyvyyden verran eteenpäin, pisteyttää lopputilanteet ja valitsee parhaaseen lopputulokseen johtavan siirron.

Pisteytys tapahtuu peliruudukon 1-ulotteinen suora (*slice*) kerrallaan; näin joka siirrolla tarvitsee päivittää vain vakioinen määrä suoria, ja koko pistelasku on lineaarinen peliruudukon leveyden suhteen; muuten se olisi lineaarinen ruudukon pinta-alan suhteen.

Itse minmax-algoritmi on eksponentiaalinen, ja määrää koko haun ajankäytön vallitsevasti.

Tietokone on varsin hyvä pelaaja erityisesti hakusyvyydestä 3 eteenpäin, ja sopivan kokoisella (ei liian isolla) ruudukolla 5-hakusyvyys on vielä hyvin pelattava. Jos voitat koneen turhan helposti, anna sen aloittaa ja itke.

Sisältö

Johdanto **2**
Sisältö **3**
Käyttöohje **4**
Yleiskuvaus käyttäjälle 4
Ajo- ja asennusohje 4
Tarvittavat tiedostot 4
Käännösohjeet 4
Ajo-ohjeet 4
Ohjelman käyttäminen 5
Peli-valikko 5
Asetukset-valikko 6
Help-valikko 6
Ohjelman toiminta ja rakenne **6**
Järjestelmän yleiskuvaus ja ratkaisutapa 6
Infrastruktuurin rakennus 6
Pelin hallinnointi 7
Tietokonepelaaja 7
Tärkeimmät tietorakenteet ja algoritmit 7
Minmax-algoritmi alpha-beta -karsinnalla 7
Slice-aputaulukot 8
Pisteiden laskeminen 9
Turhien reunasiirtojen karsinta 11
Tietokonepelaajan kutsurakenne 12
Puutteet ja parannusehdotukset 13
Aikavaativuuspuutteet/ehdotukset 13
Pelaamisen laadun puutteet/ehdotukset 14
Testauksen kuvaus **16**
Rakennustestaus 16
Rasitustestaus 16
Käyttäjätestaus/betatestaus 16
Mahdolliset synkronointiongelmat 16
Testauksen puuttumisen piinaava tuskallisuus 16
Liitteet **17**
Tietokonepelaajan lähdekoodi (Tietokone.java)
Työtuntilista

Käyttöohje

Yleiskuvaus käyttäjälle

Ristinolla-pelissä on tarkoitus saada 5 omaa merkkiä (rasti tai pallo) peräkkäin vaakaan, pystyyn tai vinottain. Voittaja on se, joka tuon tavoitteen ensimmäiseksi saavuttaa. Merkkejä laitetaan vuorotellen, pelin aloittaa rasti (pelaaja 1). Pelissä on mukana tietokonevastustaja, joka tarjoaa varsin pätevän vastuksen vähän osaavammallekin pelaajalle.

Ajo- ja asennusohje

Ohjeissa oletetaan, että osaat käyttää ympäristösi komentotulkkia/konsolia.

Ohjelma toimii Java-ympäristössä, joko omana sovelluksenaan tai applettina.

Tarvittavat tiedostot

Seuraavat tarvitaan kaikissa tapauksissa:

Ristinolla.java (pääohjelma)
Ruudukko.java (peliruudukko ja pelin hallinnointi)
Menu.java (valikko)
Pelaaja.java (pelaaja-rajapinta)

Loput ovat pelaajia, joista tarvitaan vain ne kenellä haluaa pelata:

Ihminen.java (ihmispelaaja)
Tietokone.java (tietokonepelaaja)
Satu.java (satunnaispelaaja)
Satuhali.java (lähelle pelaava satunnaispelaaja)

Lisäksi applet-versiota varten tarvitaan:

Ristinollaa.java (pääohjelman applet-versio)
ristinolla.html (appletin käynnistävä html-tiedosto)

Ristinolla.javan tulee olla ristinolla.html:ään nähden "classes" -hakemistossa, jotta html löytää sen.

Käännösohjeet

Sikäli kun olet hakemistossa, jossa kaikki (yllä luetellut) java-tiedostot ovat, voit kääntää ne class-tiedostoiksi komennolla

"javac *.java".

Jos haluat class-tiedostot eri hakemistoon, tapahtuu se esim.

"javac -d classes *.java",

tai jos java-tiedostot ovat src-hakemistossa ja olet itse sen yläpuolella,

"javac -d classes src/*.java" (windowsissa vaihda / -> \).

Ajo-ohjeet

Suoritettuasi jonkin version äskeisistä käännösohjeista, olet saanut ison kasan class-tiedostoja, joista haluat suorittaa Ristinolla.class -tiedoston. Tämä tapahtuu menemällä hakemistoon, jossa kaikki class-tiedostot ovat, ja komentamalla siellä

"java Ristinolla".

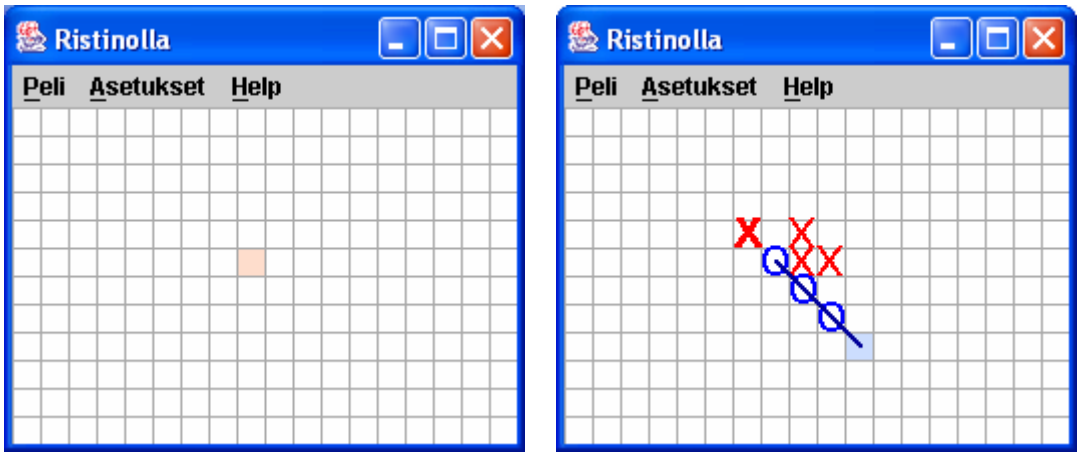
Jos haluat muuttaa ruudukon kokoa, saat siihen lisäohjeita komennolla

"java Ristinolla x" (tai laittamalla "help" tilalle mitä tahansa, mikä ei ole numero).

Ohjelman käyttäminen

Kuvat Windows XP -ympäristöstä.

Ristinolla juuri käynnistyttyään (seuraavana vuorossa ykköspelaaja rastit eli ihminen), sekä muutaman siirron jälkeen (rasti siirsi viimeksi, seuraavana pallon vuoro):



Peliä voi pelata joko hiirellä tai näppäimistöllä. Hiirellä rastien/pallojen plottaaminen tapahtuu vasemmalla hiiren napilla, näppäimistöllä haetaan ruutu nuolinäppäimillä ja plotataan enterillä tai välilyönnillä.

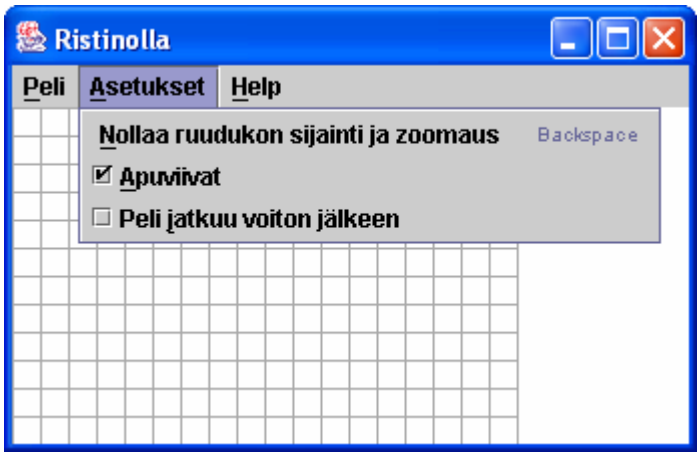
Hiirellä voi lisäksi oikealla napilla vierittää pelikenttää ja rullalla zoomata sitä; kummastakaan ei tosin ole mitään järkevää hyötyä.

Oletuksena pelin aloittaessa on ykköspelaaja (rastit) ihminen, ja kakkospelaaja (pallot) tietokone; nämä voi muuttaa Peli-valikosta milloin vain.

Peli-valikko

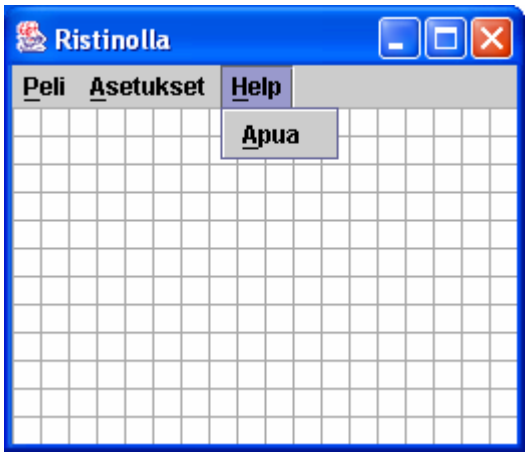
Uusi peli (F2)	Aloittaa uuden pelin
Pelaaja 1 / Rastit	Alivalikko, josta voi valita 1-pelaajan
Pelaaja 2 / Pallot	Alivalikko, josta voi valita 2-pelaajan
x Pelaaja / asetukset	Pelaajanvaihtovalikon alla oleva pelaajan oma asetusvalikko
Pois	Poistuu ristinolla-pelistä (paitsi applet-versiosta)

Asetukset-valikko



Nollaa ruudukon sijainti ja zoomaus (Backspace)	Palauttaa ruudukon alkutilaansa mahdollisten hiiren oikean napin ja rullan pyörittelyn jälkeen
Apuviivat	Pelaamista auttavat tai haittaavat apuviivat päälle/pois
Peli jatkuu voiton jälkeen	Voiko peliä jatkaa voitonkin jälkeen päälle/pois

Help-valikko



Apua	Näyttää pikaisen käyttöohjeen; pitkälti samaa asiaa kuin mitä nyt käydään läpi.

Ohjelman toiminta ja rakenne

Järjestelmän yleiskuvaus ja ratkaisutapa

Infrastruktuuri- ja hallinnointi-osiot vain nopeasti kuvattu; paljon tarkemmat selvitykset löytyvät ohjelman kommenteista tai api-dokumentista.

Infrastruktuurin rakennus

Ristinolla luo uuden Ruudukon ja Menun, sekä lisää ne omaan ikkunaansa; kun kaikki on rakennettu valmiiksi, käynnistää Ruudukon.

Ruudukko luo oletuspelaajat ja aloittaa uuden pelin saatuaan käynnistä-komennon.

Menu kutsuu Ruudukon metodeita hoitaakseen pyydetyt toimenpiteet; menua rakennettaessa pyytää Ruudukolta listan Pelaajista, jotka ovat pelissä mukana.

Pelin hallinnointi

Ruudukko aloittaa uuden pelin ja pyörittää peliä hallinnointithreadissa:

- ilmoittaa kummallekin pelaajalle uudesta pelistä
- pyytää vuorotellen kummaltakin pelaajalta uuden siirron (ja kertoo samalla vastustajan äskeisen siirron)
- lopettaa pelin, jos edellisellä siirrolla syntyi voittotilanne (eikä pelin jatkaminen voiton jälkeen -asetus ole päällä); ilmoittaa voittamisesta iloisella tekstillä
- ulostaa System.outiin tarpeelliseksi katsomiaan ilmoituksia, kuten pelin päättyminen

Tietokonepelaaja

Työn keskeisin osa, kuvataan tarkemmin "tärkeimmät tietorakenteet ja algoritmit" -osassa.

Tietokone tallentaa pelitilannetta jatkuvasti omaan 2-ulotteiseen ruudukko-taulukkoon, sekä lisäksi 1-ulotteisiin slice-aputaulukoihin, joiden avulla lasketaan pelitilanteiden pisteet.

Siirtonsa kone selvittää rekursiivisella minmax-algoritmilla, jota on tehostettu eli nopeutettu alpha-beta -karsinnalla; lisäksi kone karsii turhat reunasiirrot pois, yrittäen siirtää vain pelatun pelialueen viereen. Minmax laskee siirtoja eteenpäin asetettuun hakusyvyyteen asti, ja valitsee parhaimpaan pelitilanteeseen johtavan siirron.

Alpha-beta -karsinnan ja turhien siirtojen karsinnan saa tietokoneen asetusvalikosta päälle/pois, josta voi vaihtaa myös minmax-algoritmin hakusyvyyden.

Tietokone tulostaa System.outiin jokaisen siirtonsa jälkeen nykyisen pelitilanteen hyvyyden (pisteytyksen), lasketun hakusyvyyden päässä odottavan pelitilanteen hyvyyden sekä siirron laskemiseen kuluneen ajan.

Tärkeimmät tietorakenteet ja algoritmit

Minmax-algoritmi alpha-beta -karsinnalla

max- ja min-metodit; ilman alpha-beta -karsintaa yhdistetty minmax-metodi

Minmax-algoritmi käy läpi pelipuuta annettuun hakusyvyyteen asti. Ensimmäiseksi on tietokoneen oma vuoro, joka on algoritmin max-vuoro, vastustajan vuorot ovat min-vuoroja. Algoritmi kutsuu siis syvemmälle mentäessä vuorotellen max- ja min-metodeita. Viimeisellä tasolla tai voittotilanteessa palautetaan suoraan kyseisen pelitilanteen hyvyysarvo (pisteytys).

Maxissa valitaan mahdollisista siirtovaihtoehdoista paras, minissä taas huonoin; min-vastustaja pyrkii siis valitsemaan itselleen parhaan siirron, joka on meille tietokoneelle huonoin mahdollinen siirto. Algoritmi siis olettaa kummankin pelaajan pelaavan parhaalla mahdollisella tavalla (tuon parhaan tavan taas päättää pisteytysfunktio).

Alpha-beta -karsinta katkoo pelipuusta pois kokonaisia turhia haaroja, joista ei parasta/huonointa siirtoa voi löytyä; jos ollaan esim. etsimässä parasta siirtoa (max-vuoro), niin lapsien arvojen parantuessa ylemmän tason minin siihen asti löytämän huonoimman arvon yli, tiedetään minin valitsevan varmasti jokin huonompi siirto kuin nykyinen solmu, jolloin solmun lopuilla lapsilla ei ole merkitystä eikä niitä tarvitse käydä läpi. Alpha-beta -karsinta ei siis vaikuta tietokoneen valitsemiin siirtoihin; se vain nopeuttaa niiden löytämistä.

Vastaavasti min-osassa; lasten arvojen huonontuessa alle ylemmän tason maxin siihen asti löytämän parhaan siirron, ei max valitse tätä siirtoa vaan jonkin (tähän mennessä tai myöhemmin löytyvän) paremman siirron, eikä tämän solmun lapsia tarvitse enää tutkia.

Lisäksi minmax-algoritmissa mukana samanarvoisten siirtojen arpominen, jolloin tietokone pelaa joka kerta hieman erilailla. Alpha-beta -karsinnan toteutuksessa tämä aiheutti sen, että min-osassa karsintaan johtavassa vertailussa "if (min < alpha)", vastaava kuin maxissa "if (max >= beta)", ei yhtäsuuruus saa olla mukana, koska muuten max voisi arpoa jonkun oikeasti huonomman siirron luullen sitä samanarvoiseksi. En tosin ole aivan varma, milloin tarkalleen yhtäsuuruus saa olla mukana ja milloin ei; tämä ratkaisu vaikuttaa toimivan samoin kuin alpha-betaton minmax.

Voidaan todistaa, että parhaassa tapauksessa, jos max-osassa käydään siirtoja läpi paremmuusjärjestyksessä ja min-osassa huonommuusjärjestyksessä, alpha-beta -karsinnalla tehostettu minmax-haku käy yhtä monta siirtoa läpi kuin vastaava minmax ilman alpha-betaa pelipuussa, jonka syvyys on puolet alpha-beta-version puusta. Eli parhaan tapauksen alpha-beta -karsinnalla voidaan käydä siirtoja läpi kaksi kertaa syvemmälle kuin ilman alpha-beta -karsintaa.

Minmax-algoritmi on aikavaativuudeltaan O(n^d), missä n on siirtovaihtoehtojen määrä ja d hakusyvyys; algoritmi on siis eksponentiaalinen ja määrääkin hallitsevasti tietokonepelaajan nopeuden. Alpha-beta -karsinta nopeuttaa hakua huomattavasti, muttei poista eksponentiaalisuutta.

Algoritmin max-osa pseudo-java-koodina (lyhennettynä; mukana ei ole parhaan siirron talteen laittamista eikä samanarvoisten siirtojen arvontaa). Min-osa vastaavasti, mutta ikään kuin negatiivisena.

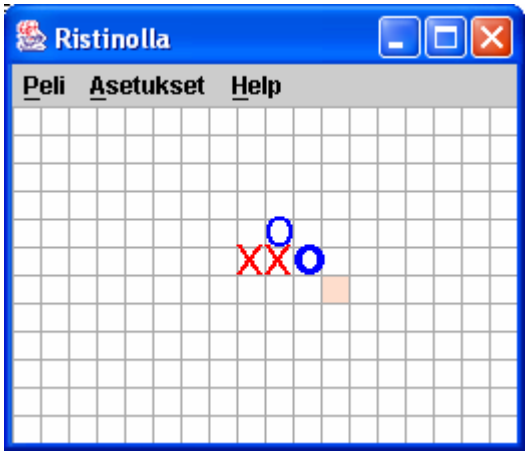
```
int max(int syvyys, int alpha, int beta)
{
     if (syvyys == maxsyvyys || voittotilanne) return hyvyysarvo();

     int max = Integer.MIN_VALUE;

     for (siirto <- jokainen testattava siirto) {

          teeSiirto(siirto);

          int h = min(syvyys + 1, alpha, beta);

          peruSiirto(siirto);

          if (h > max) {
               max = h;
               if (max > alpha) alpha = max;
               if (max >= beta) break for;
          }
     }

     return max;
}
```

Slice-aputaulukot

int[][][] slice

Sen lisäksi, että tietokone tallettaa pelitilannetta jatkuvasti omaan 2-ulotteiseen ruudukkoonsa, luodaan myös ruudukon jokaisesta 1-ulotteisesta pätkästä (vaakaan, pystyyn, vinottain x2) oma suora, väsyneen koodaustunteen mukaan nimetty *slice*, jota käytetään apuna pisteiden laskussa. Niiden ansiosta tarvitsee joka siirrolla laskea uudestaan vain 4 slicen (jotka osuvat siirron kohdalle) pisteet, ja pisteidenlasku nopeutuu huomattavasti.

Jos koko ruudukon pisteet laskettaisiin uudestaan, olisi pisteiden lasku slicejen suhteen O(n), missä n on slicejen lukumäärä, eli pisteiden lasku olisi lineaarinen slicejen määrän suhteen, kun se nyt on O(1) eli vakioinen. (Yhden slicen pisteiden lasku on vielä erikseen lineaarinen, tämä käsitellään pisteidenlasku-algoritmin yhteydessä.)



Yllä olevasta pelitilanteesta saadaan seuraavat tyhjästä eroavat slicet (myös slicejen alusta ja lopusta on tässä esimerkissä karsittu tyhjät pätkät):

_ tyhjä
x rasti, ihmispelaajan
o pallo, tietokoneen merkki

```
länsi-itä          _o_
                   _xxo_

pohjoinen-etelä    _x_
                   _ox_
                   _o_

luode-kaakko       _x_
                   _x_
                   _oo_

lounas-koillinen   _xo_
                   _x_
                   _o_
```

Pisteiden lasku -osassa katsotaan, mitä pisteitä kukin slice saa; paljastettakoon tässä sen verran, että palloilla on enemmän vapautta rakentaa suoraansa (molemmilla puolilla tyhjää tilaa), ja pallo saa paremmat pisteet.

Pisteiden laskeminen

score_slice -metodi

Pisteiden laskeminen suoritetaan jokaisen testattavan siirron jälkeen, erikseen jokaiselle siirron päällä kulkevalle slicelle (4 sliceä). Jos pisteet laskettaisiin aina koko ruudukosta ilman slice-aputaulukoita, voisi pistelaskun tehdä vain pelipuun lehtisolmuissa, mutta koska slicet päivitetään joka siirrolla on myös niiden pisteet laskettava, jotta pistetilanne pysyy hanskassa.

Pistelaskun sai myös kätevästi yhdeksi simppeliksi funktioksi, jota kutsutaan erikseen joka slicelle, jotka ovat 1-ulotteisia int-taulukoita. Ja näin ollen yhden slicen pisteiden laskun aikavaativuus on O(n), missä n on slicen pituus, eli pistelasku on slicejen pituuden suhteen lineaarinen. Parannusehdotuksia -osassa pohditaan, miten tämän saisi vakioiseksi.

Pisteiden laskeminen on yleisellä tasolla eräänlainen hahmontunnistusongelma; slicestä etsitään erilaisia suoria ja suoran alkuja. Tässä työssä pistelasku on kuitenkin yksinkertaistettu, eikä se etsi aivan kaikkia suoran alkuja, vaan vain yhtenäiset suorat; se siis näkee suoran alut, joissa on rako

välissä, vain kahtena erillisenä suorana. Tämä sen takia, jotta pisteiden laskusta saisi puristettua mahdollisimman yksinkertaisen ja tehokkaan.

Se ei siis huomaa esim. _xx_xx_ suoran alkua, koska siinä on keskellä rako. Vaikka tuosta suorasta saisi yhdellä siirrolla viiden (voitto)suoran, pistelasku näkee sen kahtena suhteellisen arvottomana erillisenä suorana. Minmaxin rekursio kuitenkin pitää huolen siitä, että tämäkin suoran alku tulee huomatuksi, kun tarkastellaan tilannetta 1-2 siirtoa eteenpäin.

Pisteet lasketaan sekä omille että vastustajan suorille; omien suorien pisteet lisätään kokonaispisteisiin, vastustajan pisteet vähennetään.

Kissaa hännästä kiinni; suorat saavat seuraavanlaiset pisteet:

x laskettavan suoran merkki (voi olla rasti tai pallo)
o eri merkki kuin laskettavassa suorassa tai ruudukon seinä
_ tyhjä, eli kuinka paljon suoran vieressä on vähintään tyhjää

```
o_x__o
oxxxxo
tms.      0

ox____    1
_x___     2
oxx___    2
_xx__     8
oxxx__    8

_xxx_     32 jos oma suora
          256 jos vastustajan suora

oxxxx_    40 oma suora
          1024 vastustajan suora

_xxxx_    1024 oma suora
          1536 vastustajan suora

oxxxxx    4096

_xxxxx_
oxxxxxx
...       >4096; tarkemmin p << 9, missä p on (x-)merkkien lukumäärä suorassa
          kertaa 2, +1 jos suoran molemmilla puolilla on tyhjää tilaa; << on
          bittisiirto vasemmalle (p:n bittejä siirretään 9 pykälää vasemmalle)
```

Pointteja pisteytyksestä:

- jos ei ole tilaa (5 merkin) suoralle, ei suorasta saa yhtään pistettä
- 1 ja 2 merkin pituiset suorat, joiden molemmilla puolilla on tyhjää tilaa, saavat saman verran pisteitä kuin 1 merkkiä pidempi suora, jonka toisella puolella ei ole tilaa rakentaa; myös muut suorat saavat enemmän pisteitä, jos niiden molemmilla puolella on tilaa. Tämä kertoo pelin luoteesta; suora täytyy estää molemmilta puolilta ja jo kolmen suora, jonka molemmilla puolilla on tilaa, on estettävä heti, tai sen muuttumista viiden suoraksi ei enää voi estää
- vastustajan suora, johon on vastattava heti tai vastustaja voittaa, saa enemmän pisteitä kuin samanlainen oma suora; tämä sen takia, jottei tietokone lähde kilpasille oman suoransa kanssa, vaikka vastustaja ehtii saada omansa ensin valmiiksi. Tästä seuraa myös se, että parillisilla hakusyvyyksillä (vastustajan siirto viimeisenä) tietokone on ylipuolustava; tarkemmin parannusehdotuksia -osassa

- voittotilanteesta saa reilusti enemmän pisteitä, jottei tietokone yritä jotain hämärää vaikka voisi tehdä voittosiirron (tosin suuremmilla hakusyvyyksillä näin käy kuitenkin välillä; tästäkin lisää parannusehdotuksia -osassa)

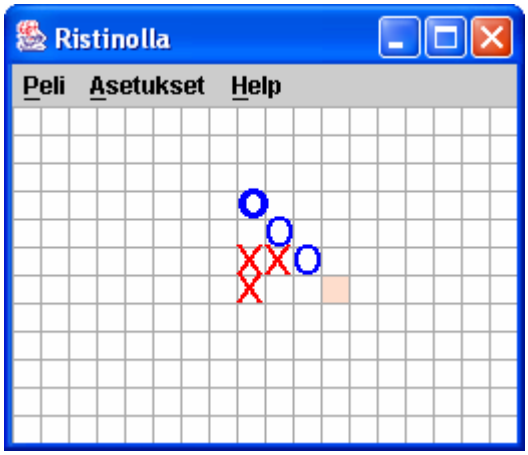
Nyt voimme katsoa, paljonko slice-aputaulukot -osassa esitetyn esimerkkipelitilanteen slicet saavat pisteitä; x:t olivat ihmispelaajan merkkejä ja o:t tietokoneen. Sisäisesti tietokone tallettaa omat merkkinsä rasteina ja vastustajan palloina, mutta sillä ei ole tässä merkitystä, unohda koko asia!

suunta	**slice**	**pallot – rastit = pisteet**
länsi-itä	_o_	2 – 0 = **2**
	xxo	1 – 2 = **–1**
pohjoinen-etelä	_x_	0 – 2 = **–2**
	_ox_	1 – 1 = **0**
	_o_	2 – 0 = **2**
luode-kaakko	_x_	0 – 2 = **–2**
	_x_	0 – 2 = **–2**
	_oo_	8 – 0 = **8**
lounas-koillinen	_xo_	1 – 1 = **0**
	_x_	0 – 2 = **–2**
	_o_	2 – 0 = **2**
yhteensä		17 – 12 = **5**

Näin, eli pelitilanteen pistetilanne ja hyvyysarvo tietokoneen hyväksi 5. Hyvä alku.

Otetaan rakentavan iloisuuden vuoksi vielä 2 siirtoa eteenpäin, hieman monimutkaisempi tilanne:

suunta	**slice**	**m**	**pallot – rastit = pisteet**
länsi-itä	_o_	2	2 – 0 = **2**
	_o_		2 – 0 = **2**
	xxo		1 – 2 = **–1**
	_x_	1	0 – 2 = **–2**
pohjoinen-etelä	_o_xx_	12	2 – 8 = **–6**
	_ox_		1 – 1 = **0**
	_o_		2 – 0 = **2**
luode-kaakko	_x_	1	0 – 2 = **–2**
	_x_		0 – 2 = **–2**
	_x_		0 – 2 = **–2**
	ooo	2	32 – 0 = **32**
lounas-koillinen	_o_	2	2 – 0 = **2**
	_xo_		1 – 1 = **0**
	_xx_	1	0 – 8 = **–8**
	_o_		2 – 0 = **2**
yhteensä			47 – 28 = **19**



Jesjes, eli tietokone johtaa jo 19 pisteellä. Kannattaa blokata tuo koneen kolmen suora seuraavaksi.

M-sarake kertoo, muuttuiko slice seuraavalla (1) rastin siirrolla vai sitä seuraavalla (2) pallon siirrolla; huomaa että sekä ykkösiä että kakkosia on 4, sillä joka siirrolla tasan 4 sliceä muuttuu.

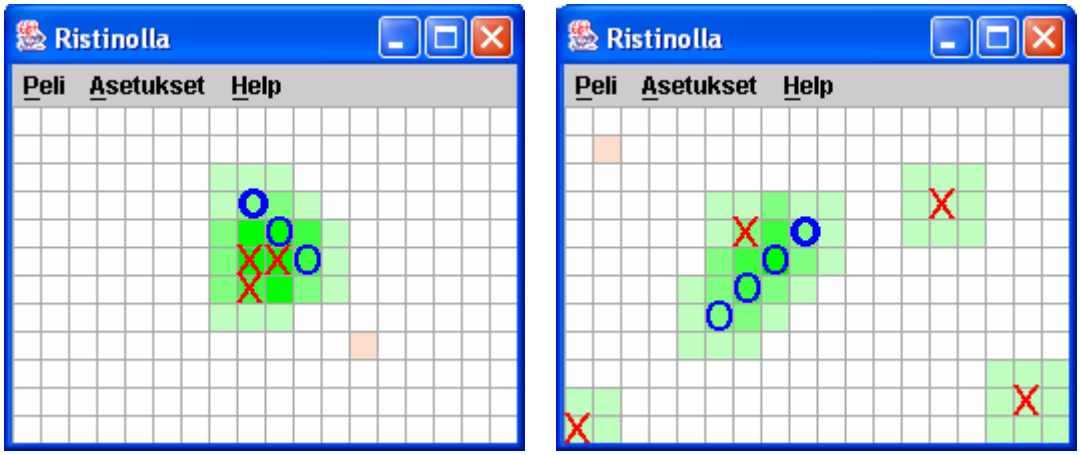
Turhien reunasiirtojen karsinta

alphaSiirto -metodi, int[][] ralpha

Turhien siirtojen karsinta, hienommalta nimeltään *ruudukon alpha-kanava -karsinta*, karsii ruudukon reunasiirrot, ja testaa siirtoja vain pelatun pelialueen viereen. Joka siirrolla merkataan erilliseen alpha-kanava -ruudukkoon kyseisen siirtoruudun ympäröivät ruudut, ja myös itse siirtoruutu, eli yhteensä 9 ruutua (paitsi jos siirtoruutu on ruudukon reunassa).

Alpha-kanavan merkkaaminen tapahtuu kasvattamalla kyseisen ruudun arvoa yhdellä (aluksi kaikki ruudut ovat 0, paitsi ruudukon keskeltä arvottu mahdollinen aloitusruutu). Siirtoa peruttaessa

vastaavasti vähennetään arvoa yhdellä, jolloin rekursio toimii oikein ja minmaxin tehtyä laskunsa palautuu alpha-kanava ennalleen, päivittyen kuitenkin jokaisen oikean tehdyn siirron jälkeen.



Kaksi pelitilannetta; alpha-kanava vihreällä, tummempaa on kasvatettu enemmän.

Tietokone valitsee oman siirtonsa alpha-kanavan vapaista ruuduista. Huomaa kuitenkin, että tämä koskee minmax-haussa vain ensimmäistä hakusyvyyttä; jokainen siellä testattu siirto päivittää alpha-kanavaa, jolloin seuraavalla syvyystasolla voi siirtää myös edellisen tason siirron viereisiin ruutuihin.

Minmax löytää siis uusia siirtoja joka syvyystasolla enintään yhden ruudun päähän edellisestä tasosta, jolloin esim. hakusyvyydellä 5 minmax käy siirtoja läpi enintään 5 ruudun päähän nykyisen pelitilanteen pelatuista ruuduista.

Alpha-kanava -karsinnan voisi toteuttaa myös niin, että alpha-kanavaa ei päivitettäisi minmaxin sisällä, vaan jokaisen oikean siirron jälkeen merkattaisiin hakusyvyyden verran siirron viereisiä ruutuja (hakusyvyydellä 5 merkattaisiin 11x11 neliö, jossa tehty siirto keskipisteenä). Tämä poistaisi alpha-kanavan hallinnointiin kuluvan ajan minmax-algoritmista, mutta toisaalta siirtoja pitäisi käydä läpi paljon enemmän (koska jo ensimmäisellä hakusyvyydellä pitäisi testata siirtoja esim. 5 ruudun päähän), joten oletan valitun tavan olevan tehokkaampi.

Karsinta voi myös teoriassa estää tietokonetta näkemästä jotain hyvää siirtoa, joka olisi kahden ruudun päässä pelatuista ruuduista. Käytännössä tällaista tilannetta ei juuri tule, koska usein lähettyvillä on aina joitain muita, oleelliseen pelitilanteeseen liittymättömiä, pelattuja ruutuja, jolloin tuo pirullisen ovela 2 ruudun päähän -siirto onkin mahdollinen. Lisäksi melkein kaikki parhaat siirrot ovat suoraan pelattujen ruutujen viereen tulevia siirtoja.

Alpha-kanava -karsinnasta tulee pientä yleisrasitetta minmax-algoritmiin, mutta kyseessä on jokaiselle testattavalle siirrolle vakioinen operaatio, jolla on järkyttävän suuri iloinen vaikutus nopeuteen; käytännössä karsinta tekee minmax-algoritmista mahdollisten siirtovaihtoehtojen suhteen eksponentiaalisen, mikä nopeuttaa erityisesti alkupeliä dramaattisesti.

Tietokonepelaajan kutsurakenne

Pikainen kaavio tietokonepelaajan java-sisäisestä metoditoiminnasta siirron selvittämisen aikana.

Homma alkaa, kun Ruudukko pyytää Tietokoneelta uutta siirtoa siirto -metodilla:

```
siirto()
     etsiSiirto() {
          teeSiirto()
               ... (käsitellään maxin sisällä)
          max()
               for (siirrot) {
```

```
                    teeSiirto() {
                         alphaSiirto()
                         update_slices()
                              update_slice()x4
                                   score_slice()
                    } // teeSiirto()
                    min()
                         ... (sama kuin maxissa, paitsi kutsuu max())
                    peruSiirto()
               } // for(siirrot)
     } // etsiSiirto()
<- (takaisin Ruudukkoon)
```

Puutteet ja parannusehdotukset

Tietokone on nykyisessä tilassaan erinomaisen hyvä pelaamaan, sekä teknisesti varsin edistynyt. Se pelaa jo 1-hakusyvyydellä hyvin ja johdonmukaisesti; suuremmilla hakusyvyyksillä sen peliin tulee mukaan kokemuksen tuomaa varmuutta ja taitoa (sekä muutamia sivuoireita). Minmax-algoritmissa on alpha-beta -karsinta, turhien siirtojen karsinta sekä samanarvoisten siirtojen arvonta, mitkä yhdessä lineaarisen ja nopean pistelaskujärjestelmän kanssa tekevät tietokoneesta teknisesti ja psykologisesti erittäin kehittyneen.

Tämä markkinointipätkä sulavana johdantona pieniin puutteisiin, joita tietokonepelaajasta löytyy, sekä näiden tai muiden parannusehdotuksiin. Puutteet voidaan jakaa lähinnä aikavaativuuksiin ja pelaamisen laatuun liittyviin ongelmiin; niistä seuraavassa.

Aikavaativuuspuutteet/ehdotukset

Koska koko pelihaku-ongelma on yleensäkin eksponentiaalinen, ei aikavaativuuksiltaan pikkuruisten algoritmien tehostaminen juurikaan auta suurempien hakusyvyyksien nopeuttamiseen. Tietokone on sopivan pienellä ruudukolla hyvin pelattava vielä 5-hakusyvyydellä, mutta alkaa siitä eteenpäin (syvemmälle tai huomattavan suurilla ruudukoilla) hidastua rankasti.

Lineaarinen score_slice

Olisi mukava saada vakioiseksi. Tällöin ruudukon koko ei vaikuttaisi tietokoneen nopeuteen käytännössä ollenkaan; nyt tietokone hidastuu pikkuhiljaa ruudukon kasvaessa.

Koska joka siirrolla sliceen päivitetään vain yksi merkki, tuntuu turhalta käydä koko slice läpi pisteiden laskussa. Vakioiseksi homman saisi jakamalla slicet vakiopituisiin osiin, jolloin tarvitsisi päivittää vain kyseisen osan pisteet. Jos jokin suora jatkuisi seuraavaan osaan, pitäisi nämä osat yhdistää, homma säilyisi silti käytännössä vakioisena.

Helpommin toteutettava vaihtoehto olisi tehdä score_slicestä lineaarinen suunnilleen tehtyjen siirtojen suhteen; tämä onnistuisi laittamalla sliceihin talteen merkkien alku- ja loppukohdat, jotka päivitettäisiin aina sliceen kohdistuvan siirron yhteydessä. Nyt pisteiden laskussa voisi aloittaa ensimmäisen merkin vasemmalta puolelta ja lopettaa viimeisen oikealle puolelle, koska ylimääräisillä tyhjillä ei ole merkitystä, kunhan vain suoralle on tilaa, mikä pitäisi tarkistaa.

Vakioisen score_slicen voisi tehdä kerralla kunnolla tallentamalla slicet 16-bittisiin short-taulukoihin niin, että 1 merkki (tyhjä, x tai o) veisi tilaa 2 bittiä. Nyt yhden 16-bittisen shortin kertoman 8 merkin pätkän pisteet voisi lukea suoraan 2^16 (16 megatavua) kokoisesta lookup-taulukosta, mikä olisi *nopeaa*. Tässä olisi vain sama ongelma kuin muutenkin slicejä pätkiessä, eli jos suora jatkuu pätkän ulkopuolelle, joutuu pätkät yhdistämään ja laskemaan pisteet yhdistetystä pätkästä, eikä tähän voisi enää välttämättä käyttää lookup-taulukkoa.

Siirtojen järjestäminen alpha-beta -karsintaa varten

Kuten minmax-algoritmia käsitellessä todettiin, alpha-beta -karsinta on sitä tehokkaampi, mitä suotuisammassa järjestyksessä siirrot käydään läpi. Siirrot voisi järjestää vaikkapa niistä suoraan seuraavan pelitilanteen hyvyysarvon mukaan (joka lasketaan muutenkin slice-päivitysten takia).

Eli minmaxissa, kun käydään läpi solmun kaikki mahdolliset siirrot, kerättäisiin ensin kaikkien siirtojen hyvyysarvot järjestykseen taulukkoon, ja alettaisiin käydä siirtoja oikeassa järjestyksessä läpi.

Suoraan yhdestä siirrosta seuraava pelitilanne ei tietenkään kerro, kuinka hyvä tilanne siitä on kehittymässä, mutta antaisi hyvän lähtökohdan alpha-beta -karsinnalle.

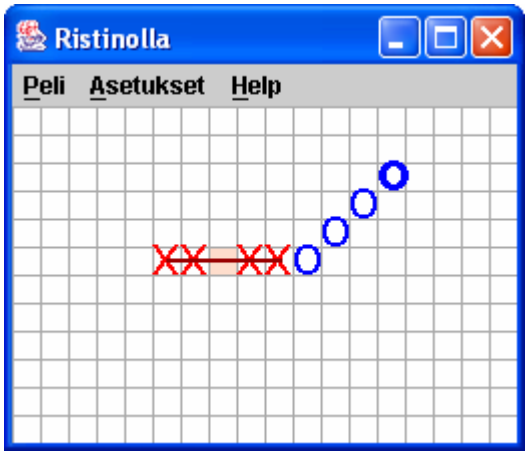
Pelaamisen laadun puutteet/ehdotukset

Tietokone pelaa kyllä erittäin hyvin, mutta harrastaa joitain mahdollisesti ei-toivottavia tapoja.

Ainiin, jos tuntuu että tietokoneen voittaa liian helposti, niin annapa sen aloittaa ja itke!

1-hakusyvyys ei huomaa suoria joissa on rako

(Tämä ei ole ei-toivottava tapa vaan harkittu puute.)



Hakusyvyydellä 1 tietokone ei huomaa suoria, joiden keskellä on rako, jolloin sen voittaa helposti hyödyntämällä tätä puutetta. Esim. tekemällä 2 kahden suoraa ruudun päähän toisistaan, "xx_xx", ei tietokone arvaa että suorasta on tulossa voittosuora, eikä yritä estää sitä.

Tämä puute johtuu pistelaskun tekemisestä yksinkertaiseksi ja tehokkaaksi, ja on näin ollen harkitusti mukana. Ongelma korjaantuu jo 2-hakusyvyydellä, jolloin tietokonetta ei enää voita äsken mainitulla menetelmällä; 3-hakusyvyydellä tietokone saattaa tarkoituksella itsekin hyökätä kyseisellä pirullisen ovelalla viritelmällä.

Ongelmaan läheisesti liittyen, 1-syvyydellä tietokone myös luulee "o_xxx_o" suoraa yhtä hyväksi, kuin jos sen ympärillä olisi enemmän tilaa (esim. "o_xxx__"). Tämäkin korjaantuu myöhemmillä hakusyvyyksillä, jolloin huomataan suorasta tulevan suljettu.

Puutteet saisi toki korjattua rakentamalla pistelaskuun lisää älyä, mutta ei maksane vaivaa. Lookup-taulukoilla toteutettuun pistelaskuun ominaisuuden voisi tosin saada "ilmaiseksi" mukaan.

Tietokone luovuttaa huomatessaan hävinneensä

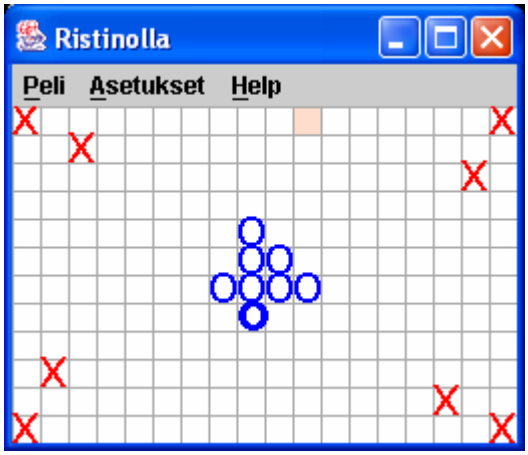
Jos tietokone toteaa varman voiton vastustajalle, ei se enää yritä estää vastustajan suoria, vaan yrittää omiaan. Tämä on ehkä hieman ylpeää toimintaa, muttei välttämättä huono juttu: jos vastustaja ei huomaa voittomahdollisuuttaan, ei tietokonekaan sitä puolustamalla kerro.

Toisaalta, vastustaja saattaa huomata vain osan (esim. toisen muodostuneen "_xxx_" -suoran) voittomahdollisuudestaan, jolloin tietokone voisi vielä saada sen estettyä; pitäisi vain arvata, mitä vastustaja tietää ja mitä ei :)

Ongelma johtuu siitä, että tietokone saa vastustajan voitosta hieman paremmat pisteet, jos koneella on parempia omia suoria voiton tullessa. Tämän voisi estää priorisoimalla varma voitto -suoria enemmän pistelaskussa, mutta siitä saattaa seurata uusia ongelmia; pistelasku on varovaista tiedettä, pienikin muutos voi rikkoa jonkin toimintakuvion.

Tietokone ei aina halua voittaa heti, vaan nöyryyttää vastustajaa ensin

Niin no, ei ehkä reilua, mutta oma vikansa jos ei anna koneelle hyvää vastusta ;) Eli, vaikka tietokone voisi voittaa seuraavalla siirrolla, saattaa se siirtää jotain muuta, jos vain senkin siirron kautta seuraa varma voitto. Parhaiten ongelman huomaa siirtämällä itse satunnaisiin paikkoihin; ainakin 5-hakusyvyydellä kone ei useinkaan tee heti suoraa vaan rakentaa oman tuomiopäivän järjestelmänsä ensin:



Johtuu taas siitä, että nöyryytyksen takaa löytyy parempipisteisiä pelitilanteita, joissa tietokoneella on enemmän uhkaavia suoria. Voisi ratkaista laittamalla minmaxissa muistiin löytyneiden voittotilanteiden syvyydet ja valitsemalla niistä aina matalimmalla olevan, mutta käytännön pelissä nöyryytystilannetta pääsee harvoin tulemaan (jos ihminen pelaa kunnolla).

Parilliset hakusyvyydet ovat ylipuolustavia

Parillisilla hakusyvyyksillä, joissa siis viimeisenä testataan vastustajan, ei tietokoneen, siirto, tietokone on turhan ylipuolustava. Ei välttämättä paljoa huonompi pelaaja, mutta erityylinen. Kone kyllä voittaa, jos siihen tarjoutuu tilaisuus, muttei hyökkää yhtä helposti.

Tämä johtuu siitä, että pisteidenlaskuun on hard-koodattu paremmat pisteet vastustajan varma voitto -suorille; parillisella hakusyvyydellä vastustajan suorat saavat liian hyvät pisteet, koska seuraavana onkin tietokoneen, eikä vastustajan vuoro. Ongelma oli tarkoitus korjatakin, mutta se osoittautui hankalaksi, koska oikeat tehdyt siirrot ja minmaxissa testattavat siirrot pitäisi pisteyttää erilailla, eikä se toiminut mitenkään päin.

Joten, kun eivät parilliset hakusyvyydet niin musertavasti ole huonompia (vaikkakin aloittavan 1-syvyyden ja 2-syvyyden voitot menevät suunnilleen tasan; tosin pelin luonteeseen kuuluu, että aloittaja voittaa), sai niistä kätevästi erityylisiä pelaajia ja tilanne on tämä.

Testauksen kuvaus

Näin äkkiä voisi tietysti sanoa, ettei tätä ole testattu… Mutta onhan toki, vaikkei sitä olekaan dokumentoitu. Joten kerron tässä nopeasti joitain testausperiaatteita ja muisteloita. Keksin erityyppisille testauksille puoliksi omat nimet, niin vaikuttaa siltä että tiedän asiasta enemmän kuin kukaan muu.

Tässä kerrottujen yleisten testausten lisäksi pisteiden lasku -osassa olevat pisteytysesimerkit on dokumentoidusti testattu; tietokone pisteytti ne oikein.

Rakennustestaus

Ristinollaa on jatkuvasti kehitettäessä testattu, kaikki uudet ominaisuudet on testattu ääritapauksissa ja mahdolliseen vanhaan versioon verraten, jotta toimii samalla tavalla.

Itse pelaamista on testattu *paljon*, kaikki ajateltavissa olevat siirtokombinaatiot ja pelikuviot on käyty läpi ja katsottu, että tietokone pisteyttää ne oikein ja valitsee järkevän siirron. Ruudukon reunaruutujen toimivuus pelissä on varmistettu.

Rasitustestaus

Eritasoisia tietokoneita on laitettu vastakkain monen monta kertaa, ja kaikki on toiminut hyvin, myös "peli jatkuu voiton jälkeen" -asetuksen ollessa pois päältä: ruudukko tulee täyteen ja vuorossa oleva tietokone luovuttaa todeten siirron mahdottomuuden.

Käyttäjätestaus/betatestaus

Ristinollaa on testattu ainakin kahdella eri käyttäjällä (minun lisäksi), joiden kommentit on otettu huomioon (kumpikaan käyttäjä ei tosin löytänyt mitään bugeja tai tietokoneen pelaamiseen liittyviä puutteita).

Mahdolliset synkronointiongelmat

Testaukseen liittyen; Ruudukossa pyörii oma pelinhallintathreadi, mistä aiheutuu mahdollisia synkronointiongelmia, jotka on kyllä paikallistettu mutta niiden epätodennäköisyyden takia asialle ei ole tehty mitään. Ongelmat saattavat ilmestyä menun toimintoja käytettäessä. Zap.

Testauksen puuttumisen piinaava tuskallisuus

Niin, kun nyt oikeasti ristinollaa ei ole juurikaan testattu, vaan koodia on lisätty ja sen jälkeen toivottu että toimii. Ja kun näyttää toimivan, niin hyvä. Paitsi kaivertava pelon tunne siitä, että jotain pahaa tulee tapahtumaan.

Harmi vain, en alusta alkaenkaan jonkin pienen idean kuolemisen johdosta laittanut koodiin edes helppoja asserteja, jotka olisivat säästäneet monilta ongelmilta ja painajaisilta. Joten, kaikki kyllä tuntuu toimivan, mutten voi siitä täysin varma olla, enkä pääse nukkumaan.

Liitteet

Tietokonepelaajan lähdekoodi (Tietokone.java)

…

Työtuntilista

…

käyttöliittymä	33 tuntia (7.9. – 16.9.)
tietokonepelaaja	42 tuntia (17.9. – 13.10.)
dokumentti	10 tuntia (14.10. – 15.10.)
YHTEENSÄ	**85 tuntia** + ehkä n. 20 %, koska ajat on aina arvioitu hieman alaspäin